



HP Unified Functional Testing

Software Version: 12.52

Windows ® operating systems

User Guide

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

© Copyright 1992 - 2016 Hewlett-Packard Development Company, L.P.

Trademark Notices

Adobe® and Acrobat® are trademarks of Adobe Systems Incorporated.

Google™ and Google Maps™ are trademarks of Google Inc

Intel® and Pentium® are trademarks of Intel Corporation in the U.S. and other countries.

Microsoft®, Windows®, Windows® XP, and Windows Vista ® are U.S. registered trademarks of Microsoft Corporation.

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, go to:

<https://softwaresupport.hp.com>.

This site requires that you register for an HP Passport and sign in. To register for an HP Passport ID, go to <https://softwaresupport.hp.com> and click **Register**.

Support

Visit the HP Software Support Online web site at: <https://softwaresupport.hp.com>

This web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. To register for an HP Passport ID, go to: <https://softwaresupport.hp.com> and click **Register**.

To find more information about access levels, go to: <https://softwaresupport.hp.com/web/softwaresupport/access-levels>.

HP Software Solutions & Integrations and Best Practices

Visit **HP Software Solutions Now** at <https://h20230.www2.hp.com/sc/solutions/index.jsp> to explore how the products in the HP Software catalog work together, exchange information, and solve business needs.

Visit the **Cross Portfolio Best Practices Library** at <https://hpln.hp.com/group/best-practices-hpsw> to access a wide variety of best practice documents and materials.

Contents

Part 1: UFT Introduction	25
Chapter 1: What is UFT?	26
UFT Overview	27
UFT Program Use	29
Licensing	29
Required access permissions	29
Demo applications	31
Accessibility	31
Unicode Compliancy	31
Chapter 2: UFT Licenses	32
UFT License Types	33
Install UFT Licenses with the License Wizard	34
Install UFT Licenses from the Command Line	39
UFT Licensing FAQs	41
UFT Licenses - Troubleshooting and Limitations	44
Chapter 3: UFT Document Management	45
UFT Test Documents	46
UFT and Version Control Systems	47
Relative Paths for Test Resources	49
Portable Copies of Tests	50
Opening Tests with Locked Resources	51
Upgrading Documents From Previous Versions	51
Upgrading QuickTest tests or components	51
For Service Test tests or components	52
Known Issues - Opening Documents	56
Part 2: UFT Panes	58
Chapter 4: Active Screen Pane	59
Saving Active Screen content	59
The Active Screen and Insight	59
The Active Screen and Web-based applications	60
Stop saving Active Screen information	60
Update a single Active Screen capture	60
Chapter 5: Bookmarks Pane	61
Chapter 6: The Canvas	62
Chapter 7: Data Pane	63
Data table content	63

Data table values	63
Changing column headers	63
Importing Excel files to the Data Pane	63
Data Pane Specifications	64
Chapter 8: Debug Panes	65
Chapter 9: Document Pane	66
Chapter 10: Errors Pane	67
Code syntax errors	67
Missing resources	67
Missing references	68
Missing property values	68
Chapter 11: Output Pane	69
Chapter 12: Properties Pane	71
Chapter 13: Run Step Results Pane	72
Chapter 14: Search Results Pane	73
Chapter 15: Solution Explorer Pane	74
Chapter 16: Tasks Pane	75
Manage TODO comments	75
Chapter 17: Toolbox Pane	77
The Toolbox pane and GUI testing	77
The Toolbox pane and API testing	77
See also:	78
The Toolbox pane and Business Process Testing	78
Part 3: UFT Configuration	79
Chapter 18: Global Options	80
Chapter 19: Document Settings	81
Chapter 20: Set Options Programmatically	83
Part 4: GUI Testing Design with the UFT IDE	85
Chapter 21: GUI Test Creation Overview	86
Methodologies for Creating Tests	87
Keyword-driven methodology	87
Recording	87
Importing tests from Sprinter	88
Enhancing Your Tests	88
Checkpoints	89
Parameterization	89
Output Values	89
Programming Statements	89
Active Screen Updates	90

Editor and Keyword View - A Comparison	90
Sample Test	91
Chapter 22: Create a Keyword-Driven GUI Test	93
Analyze your application	93
Prepare the testing infrastructure	94
Add steps to the actions in your test action repository	95
Enhance your test	95
Chapter 23: Record GUI Tests and Components	96
Recording Overview	97
Normal Recording	97
Analog Recording	97
How to Record a GUI Test or Component	99
Known Issues When Recording	102
Chapter 24: Actions in GUI Testing	103
Actions in GUI Testing	104
Structure Your Test with Actions	105
Display / Modify Action Data	106
Create an action template	106
Action and action call properties	107
Exit an action using programming statements	107
Known Issues with Actions	108
Copies of tests	108
Chapter 25: Keyword View	109
Keyword View User Interface	110
Item column	110
Operation column	111
Value column	111
Assignment column (actions only)	111
Comment column (actions only)	112
Output column (components only)	112
Shortcut keys in the Keyword View	112
Standard Steps in the Keyword View	113
Define or modify an item value	113
Parameterize an Item value for an argument	114
Encode a password	114
Add a standard step after a conditional or loop block	115
Comments in the Keyword View	115
Comments in actions	115
Comments in components	116
Conditional and Loop Statements	116
Known Issues - Keyword View	119
Chapter 26: Integration with API Tests	120

Integrating API Testing in GUI Tests	121
Licensing for calling API tests and actions	121
Insert or modify a call to an API test	122
Use API test parameters in a GUI test	123
Using API Tests in a GUI Test - Use-Case Scenario	123
Calling API Tests with Parameters - Use-case Scenario	125
Chapter 27: Maintain GUI Tests or Components	133
Why Tests or Components Fail	134
Application errors	134
Application changes	134
Missing objects	134
Maintenance Run Mode	135
Maintenance Run Mode prerequisites	136
Determine UFT wait time	136
Run the test or component in Maintenance Run Mode	136
Merge changes to your shared object repository	136
Maintenance Run Wizard Workflow	137
Update Run Mode	138
Smart Identification	138
How to Update Test Object Descriptions, Checkpoints, or Output Values, or Active Screen Captures	140
Known Issues in Maintenance and Update Run Modes	142
Known issues in Update Run Mode	142
Chapter 28: Recovery Scenarios for GUI Testing	144
Recovery Scenarios Overview	145
When to Use Recovery Scenarios	146
Programmatically Controlling the Recovery Mechanism	147
How to Create and Manage Recovery Scenarios	147
How to Manage Recovery Scenario Associations	148
Troubleshooting and Limitations - Recovery Scenarios	151
Chapter 29: Using Performance Testing and Business Service Management Products with UFT GUI Tests	152
HP Performance Testing and Business Service Management Products Overview	153
Designing Tests for HP Performance Testing Products	155
Designing GUI Tests for HP Business Process Monitor	155
Running GUI Tests from HP Performance Testing Products	156
How to Insert and Run GUI Tests in Performance Center and LoadRunner	157
Running GUI Tests from HP Business Process Monitor	157
Measuring Transactions	158
Silent Test Runner	160

Part 5: GUI Testing: Test Objects, Checkpoints and Output Values	162
Chapter 30: The Test Object Model	163
Test Object Model - Overview	164
How UFT Learns Objects	164
How UFT Identifies Objects During a Run Session	165
How UFT Applies the Test Object Model Concept	166
Test Object Descriptions	168
UFT Test Object Hierarchy	169
Properties and Operations for Test Objects and Run-Time Objects	170
Object Identification Process Workflow	172
How to Use the Object Spy to View Object Properties and Operations or Add an Object to a Repository	173
How to Use the Remote Object Spy to View or Add Objects Running on a Remote Computer	174
Troubleshooting and Limitations - Object Spy	177
Chapter 31: Test Objects in Object Repositories	178
Object Repository Types - Overview	179
Deciding Whether to Use Local or Shared Object Repositories	180
Local Object Repository - Overview	181
Shared Object Repositories Overview	182
Exporting Local Objects to a Shared Object Repository	183
Local Copies of Objects from Shared Object Repositories	183
Considerations for Working with Shared Object Repositories	184
Object Repository Window - Overview	186
Comparison of Object Repository Window and Object Repository Manager	187
Adding and Deleting Test Objects in a Local or Shared Object Repository	188
Guidelines for Copying, Pasting, and Moving Objects	190
Locating Objects	191
Maintaining Identification Properties	192
Working with Repository Parameters	195
Repository Parameter Value Mappings	196
Working with Test Objects During a Run Session	197
Managing Shared Object Repositories Using Automation	198
How to Add a Test Object to an Object Repository	199
How to Maintain Test Objects in Object Repositories	202
How to Create and Manage Shared Object Repositories	206
How to Locate an Object in an Object Repository	209
How to Define a Visual Relation Identifier for a Specific Test Object - Use-Case Scenario	210
Object Repository Window	215
Object Repository Manager Main Window	222
Troubleshooting and Limitations - Object Repositories	226
Chapter 32: Configuring Object Identification	227

Object Identification Configuration - Overview	228
Mandatory and Assistive Properties	229
Ordinal Identifiers	230
The Index Ordinal Identifier	231
The Location Ordinal Identifier	232
The CreationTime Ordinal Identifier	233
Visual Relation Identifiers	234
How UFT Interprets Horizontal and Vertical Visual Relations	235
Considerations for Working with Visual Relation Identifiers	236
Smart Identification	237
When to Use Smart Identification	238
The Smart Identification Process	238
Smart Identification Information in the Run Results	239
How UFT Uses Smart Identification - Use-Case Scenario	239
Test Object Mapping for Unidentified or Custom Classes	241
How to Configure Object Identification for a Test Object Class	242
How to Manage Identification Properties of a Test Object Class	243
How to Map an Unidentified or Custom Class to a Standard Windows Class	243
Chapter 33: Comparing and Merging Object Repositories	245
Object Repository Comparison Tool Overview	246
Object Repository Merge Tool Overview	247
Object Conflicts	248
How to Compare Two Object Repositories	250
How to Merge Two Shared Object Repositories	252
How to Update a Shared Object Repository From a Local Object Repository	254
Object Repository Comparison Tool Main Window	256
Object Repository Merge Tool Main Window	261
Chapter 34: Extending UFT Object Identification	265
Object Identification Challenges in UFT	266
Identifying Objects Using Insight	266
Creating Insight test objects	267
Creating steps with Insight Test Objects	267
Running tests with Insight	268
Learn more!	268
How to Work with Insight Test Objects	268
Add an Insight object	269
Modify an Insight test object's image	270
Retrieve text from an Insight Object	270
Update Insight test object details	270
UI Automation in UFT	271
Enable UI Automation support	272
How Does UFT Use the UI Automation Framework?	272

Control Types and UFT Test Objects	272
Supported Patterns and Test Object Methods	273
When Should You Use UFT UI Automation Support?	274
Native UI Automation Methods	279
How to Use UFT UI Automation Support	281
Learn objects in UI Automation mode and add them to the object repository	281
Record test/component steps in UI Automation mode	282
Identifying Unsupported Objects in Test Runtime	282
UI Automation Support Limitations	285
Chapter 35: Virtual Objects	286
Virtual Objects - Overview	287
How Virtual Objects are Defined and Recognized	287
How to Define Virtual Objects for Unsupported Objects in Your Test or Scripted Component	288
Chapter 36: Checkpoints in GUI Testing	289
Checkpoints Overview	290
Adding Existing Checkpoints to a Test	290
Simple and Advanced Mode	291
Checkpoint Types	291
Standard Checkpoints Overview	293
Accessibility Checkpoints Overview	293
Bitmap Checkpoints Overview	294
Fine-Tuning the Bitmap Comparison	296
Database Checkpoints Overview	298
File Content Checkpoints Overview	299
Table Checkpoints Overview	300
Page Checkpoints Overview	300
Checking Text Overview	301
Text Recognition Overview	301
Guidelines for Text Recognition	302
Text Recognition and Development Environments	303
Checking Text in an Image - Use-Case Scenario	305
XML Checkpoints Overview	307
XML Checkpoint Types	309
Using XML Objects and Methods to Enhance Your Test or Scripted Component	309
How to Insert a Checkpoint in a GUI Test or Component	309
How to Include and Ignore Areas When Comparing a Bitmap - Use-Case Scenario	315
How to Configure Text Recognition Settings	318
Checkpoint Properties Dialog Box	320
Add Existing Checkpoint Dialog Box	321
Troubleshooting and Limitations - Using Checkpoints	322
Chapter 37: Output Values in GUI Testing	324

Output Values Overview	325
Output Value Categories	325
Output Types and Settings	327
Viewing and Editing Output Values	328
Storing Output Values	328
How to Create or Modify an Output Value Step	330
Output Values for in the Keyword View	333
Output Value Properties Dialog Box	333
Add Existing Output Value Dialog Box	334
Advanced File Content Output Value Properties Dialog Box	335
Chapter 38: Parameterizing Object Values	336
Parameterizing Values Overview	338
Default Parameter Values	339
How to Parameterize Values for Operations or Local Objects	340
How to Parameterize a Checkpoint Property Value	341
Data Tables and Sheets in GUI Tests and Components	342
Using Different Data Tables	343
Data Table Objects, Methods, and Properties	344
Formulas in Data Tables	344
How to Define and Manage Data Tables in a GUI Test	346
How to Insert Formulas into Data Tables for Use in Checkpoints in a GUI Test	348
How to Import Data Into a GUI Test Using Microsoft Query	349
Data Table Parameters	349
When to Choose Global or Action Data Table Parameters	352
Environment Variable Parameters	353
Built-in Environment Variables	356
How to Use User-Defined External Environment Variables	357
How to Create an External Environment Variables File	358
Automatically Parameterizing Steps	360
Guidelines and Considerations for Automatically Parameterizing Steps	360
Data Driver	364
Test and Action Parameters	364
Passing Test and Action Parameter Values	367
Sharing Action Information	368
Using Action Parameters in Steps in the Editor	370
How to Use Action Parameters	371
Regular Expressions Overview	372
Regular Expressions for GUI Test and Component Object Property Values	374
Regular Expressions in GUI Test and Component Checkpoints	374
Regular Expression Characters and Usage Options	375
Regular Expressions in the Find and Replace Dialog Boxes	380
Chapter 39: Value Configuration and Parameterization	382

Value Configuration Overview	383
Working with Local and Component Parameters	383
How to Configure Constant and Parameter Values	384
How to Parameterize Input Values (Keyword Components)	385
Part 6: API Testing Design with the UFT IDE	388
Chapter 40: API Test Creation Overview	389
UFT API Testing Overview	390
Automated Testing Tool Integration	391
Automatically Generating API Tests - Overview	392
How to Create an API Test	393
How to Create an API Test - Use-Case Scenario	397
How to Automatically Generate API Tests	403
Chapter 41: Standard Activities	406
Activity Overview	408
Checkpoint Validation for Test Steps	408
XPath Checkpoints	409
How to Set Array Checkpoints for Test Steps	410
How to Set XPath Checkpoints for Test Steps	412
How to Use Flow Control Activities	413
How to Execute Database Commands or Retrieve Data	416
How to Send a Multipart HTTP or REST Service Request	418
How to Create a Call to a Java Class	420
How to Use JMS Activities	422
How to Create an SAP API Test Step	425
How to Call External Tests or Actions	426
How to Prepare and Run a Load Test	429
How to Test Web Sockets Communication	431
Standard Activity Types	434
Flow Control Activities	437
Condition Activity	437
Loop Activity	438
Wait Activity	439
Miscellaneous Activities	439
Run Program Activity	439
End Program Activity	440
Report Message Activity	441
String Manipulation Activities	442
Replace String Activity	442
Substring Activity	443
Trim String Activity	443

System Activities	444
Math Activities	444
Date and Time Activities	444
Increment Date Activity	445
Date/Time Difference Activity	445
Format Date/Time Activity	446
File System Activities	447
File Copy Activity	447
File Move Activity	448
Create File Activity	449
File Rename Activity	449
Write to File Activity	450
Read From File Activity	451
Get Folder Contents Activity	451
Database Activities	452
Open Connection/Close Connection Activity	453
Select Data Activity	454
Execute Command Activity	455
Begin Transaction Activity	455
Commit Transaction/Rollback Transaction Activity	456
FTP Activities	457
FTP Download Activity	457
FTP Upload Activity	459
FTP Rename Activity	460
FTP File Delete Activity	461
FTP File Get Size Activity	462
FTP Directory Delete Activity	462
FTP Directory Create Activity	463
FTP Directory Get Content Activity	464
Network Activities	465
HTTP Request Activity	466
HTTP Receiver Activity	468
SOAP Request Activity	469
JSON Activities	470
String to JSON Activity	471
JSON to String Activity	471
Java Activities	471
JMS Activities	472
Publish Message to JMS Topic Activity	472
Receive Message from JMS Queue Activity	473
Receive Message from JMS Topic Activity	474
Send Message to JMS Queue Activity	475

Send/Receive Messages from JMS Queue Activity	475
Subscribe to JMS Topic Activity	477
Browse JMS Queue Activity	477
Load Testing Activities	478
IBM WebSphere MQ Activities	479
Connect to MQ Queue Manager Activity	480
Disconnect from MQ Queue Manager Activity	481
Commit/Backout Pending MQ Messages Activity	481
Browse Messages in MQ Queue Activity	482
Put Message to MQ Queue Activity	484
Get Message from MQ Queue Activity	485
Put and Get Message from MQ Queue	486
Subscribe to MQ Topic Activity	488
Unsubscribe from MQ Topic Activity	489
Publish Message to MQ Topic Activity	489
Receive Message from MQ Topic Activity	490
HP Automated Testing Tools Activities	492
SAP Activities	492
XML Activities	494
Transform XML Activity	495
Compare XML Activity	495
XML to String Activity	496
String to XML Activity	496
Validate XML Activity	497
Web Sockets Activities	497
Open Socket Activity	497
Close Socket Activity	498
Send Message Activity	498
Receive Message Activity	499
Troubleshooting and Limitations - Standard Activities	500
Chapter 42: Custom Activities	502
Custom Activity Overview	503
Activity Sharing	506
How to Perform Activity Sharing	506
Negative Testing of Web Services	507
Passing REST Service Properties	508
Exposing REST Service Properties	511
How to Import a WSDL-Based Web Service	512
How to Create a REST Service	515
How to Send and Receive a JSON Request for a REST Service	521
How to Import a Web Application Service	522
How to Import a Network Capture File	525

How to Import and Create a .NET Assembly API Test Step	526
Troubleshooting and Limitations - Custom Activities	529
Chapter 43: Actions for API Tests	531
Actions in API Testing - Overview	532
How to Use Actions in an API Test	533
Actions Using the Data Table	534
Chapter 44: Data Usage in API Tests	536
Using Data in Your API Test or Component Steps	537
Populating Property Values Manually	537
Linking Property Values to a Data Source	539
Linking Property Values to Other Test Steps	541
Outgoing Links	543
Linking Property Values to Multiple Sources of Data	544
Data Assignment in Arrays	545
How to Add Data Sources to an API Test	548
How to Assign Data to API Test/Component Steps	551
How to Assign Data to API Test Steps - Tutorial	555
How to Define API Test Properties or User/System Variables	558
How to Parameterize XML Data	560
How to Data Drive Array Checkpoints	562
Navigating Within a Data Source	564
Parent/Child Data Source Relations	565
How to Set the Data Source Navigation Properties	566
Data Keywords	567
Troubleshooting and Limitations - Using Data in API Tests and Components	569
Chapter 45: Updating Services and Assemblies	570
Updating Services	571
How to Update a Web Service	572
How to Update a .NET Assembly	574
How to Resolve Conflicts in a REST Service Test Step	574
How to Update an SAP RFC or IDoc	575
Update Port Security Wizard	577
Update Step/Activity Wizard	577
Resolve REST Method Steps Wizard	578
Troubleshooting and Limitations - Updating	580
Chapter 46: Web Service Security	581
Setting Security Overview	582
Security Scenarios Overview	582
Web Service Scenario Overview	583
WCF Service Scenarios Overview	584
How to Set Security for a Standard Web Service	586
How to Customize Security for WCF Type Web Services	590

How to Set up Advanced Standards Testing	594
Troubleshooting and Limitations - Web Service Security	596
Chapter 47: Asynchronous Service Calls	597
Asynchronous Services	598
How to Create an API Test for an Asynchronous Web Service	599
Troubleshooting and Limitations - Asynchronous Testing	602
Chapter 48: API Testing Extensibility	603
Creating New Activities - Overview	604
Custom Activity Files	604
Runtime Files	605
Signature Files	605
Property Definitions	608
Addin Files	610
Resource Files	612
How to Use the Wizard to Create a Custom Activity - C#	612
How to Use the Wizard to Create an Activity - Java	614
How to Manually Create a Custom Activity in C#	616
How to Create a Runtime File	619
Troubleshooting and Limitations - Extensibility (API Testing)	624
Part 7: Creating and Enhancing UFT Tests with Code	625
Chapter 49: Using the UFT Editor	626
Editing Text and Code Documents	627
Statement Completion in the Editor	627
Learn more!	628
Statement Completion Options	628
Statement Completion Considerations	631
Automatic Code Completion for GUI Testing	632
Searching and Replacing in the Editor	633
File and Item Types Included in String Searches	635
How to Use Code Snippets and Templates	635
How to Search for References or Classes in Documents in the Editor (API Testing Only)	638
Editor User Interface	640
Class and Function Browser Examples	643
Editor Icons	644
Troubleshooting and Limitations - Statement Completion	646
Chapter 50: Programming in GUI Testing Documents in the Editor	647
Programming in GUI Testing Documents in the Editor - Overview	649
Programmatic Descriptions	649
Static Programmatic Descriptions	651
Guidelines for Using Static Programmatic Descriptions	652

Dynamic Programmatic Descriptions	655
Retrieving Child Objects	657
Programmatic Description Checks	658
Opening and Closing Applications Programmatically	660
Comments, Control-Flow, and Other VBScript Statements	661
Retrieving and Setting Identification Property Values	661
Checkpoint and Output Statements	662
Native Properties and Operations	663
Running DOS Commands	664
Filtering the GUI Steps Reported in the Run Session	664
Using the Windows API in GUI Testing	665
How to Enhance Your Actions, Scripted Components, and Function Libraries Using the Windows API	665
Basic VBScript Syntax	667
General VBScript Syntax Rules and Guidelines	668
Handling VBScript Syntax Errors	669
Formatting VBScript Text	670
Comments in VBScript	671
Parameter Indications in VBScript	672
Parentheses in VBScript	673
Calculations in VBScript	674
Variables in VBScript	675
Do...Loop Statement	676
For...Each Statement	677
For...Next Statement	678
If...Then...Else Statement	678
While...Wend Statement	680
With Statement	680
Report Modes	681
Chapter 51: User-Defined Functions and Function Libraries	683
Function Library Overview	684
Associated Function Libraries	684
Working with Associated Function Libraries in ALM	685
User-Defined Functions	685
User-Defined Function Storage and Access	687
Registered User-Defined Functions	688
Preparing the User-Defined Function for Registration	689
Registering User-Defined Functions as Test Object Methods	689
Unregistering User-Defined Test Object Methods	690
Running an Overriding User-Defined Test Object Method	691
Loading Function Libraries During a Run Session	693
How to Create and Manage Function Libraries	693

How to Edit a Function Library	696
How to Manage Function Library Associations	697
How to Create and Work with a User-Defined Function	700
How to Create and Register a User-Defined Function Using the Function Definition Generator	703
Function Definition Generator Dialog Box	708
Troubleshooting and Limitations - Function Libraries	715
Chapter 52: Generated Programming Operations	717
Programming Statements Overview	718
Conditional Statements	718
With Statements	720
Message Statements	722
Run session messages in the HTML report	722
Run session messages in the Run Results Viewer	722
Step messages in the Run Results Viewer	722
Display messages during the run session	723
Test Synchronization	723
Synchronization Points	724
Exist and Wait Statements	725
Timeout Values Modification	725
Step Generator	726
How to Insert Steps Using the Step Generator	726
How to Generate With Statements for Your Test	727
To generate With statements for existing actions:	728
Add Synchronization Point Dialog Box	730
Step Generator Dialog Box	732
Storage Location Options Dialog Box	738
Chapter 53: UFT Automation Scripts	740
UFT Automation Object Model Overview	741
When to Use UFT Automation Scripts	742
Application Object	743
UFT Automation Object Model Reference	744
Generated Automation Scripts	744
How to Create a UFT Automation Script	745
How to Run Automation Scripts on a Remote Computer	748
Troubleshooting and Limitations - Automation Scripts	750
Chapter 54: Writing Event Handlers for API Test Steps	751
Event Based Coding - Overview	753
Writing Code for API Test Events - Overview	755
Writing Events for API Tests - Use-case Scenarios	756
Custom Code Steps for API Tests	764
How to Open a Window for Writing Custom Code	765

How to Manipulate Web Service Call/HTTP Request/SOAP Request Step Input/Output Properties	767
How to Stop an API Test Run from an Event	778
How to Manipulate Data Programmatically	779
How to Access and Set the Value of Step Input, Output, or Checkpoint Properties	784
How to Report Test Run-Time Information	789
How to Retrieve and Set Test or User Variables	792
How to Encrypt and Decrypt Passwords	794
API Event Coding - Event Structure Overview	795
API Event Coding - Standard Event Structure	795
API Event Coding - Web Service Event Structure	799
API Test Event Coding Common Objects	802
Activity Object	803
Assert Object	804
Checkpoint Object	804
CurrentIterationNumber Object	805
EncryptionMngr Object	806
EnvironmentProfile Object	807
InputAttachment Object	807
InputEnvelope Object	809
OutputAttachment Object	810
OutputEnvelope Object	812
Parent Object	813
TestProfile Object	814
UserLogger Object	814
API Test Event Coding Common Methods	815
Export Method	816
ExportToExcelFile Method	817
GetDataSource Method	818
GetValue Method	819
GetVariableNames Method	820
GetVariableValue Method	821
Import Method	822
ImportFromExcelFile Method	823
Info Method	824
Report Method	825
SelectSingleNode Method	825
SetValue Method	826
SetVariableValue Method	827
Troubleshooting and Limitations - Using Data in API Tests and Components	828

Part 8: UFT Running and Debugging Operations	830
Chapter 55: Running Tests and Components	831
Running Tests and Components - Overview	832
Running GUI Tests with a Disconnected Remote Desktop Connection	833
Command Line Syntax for Running API Tests	834
How to Run a Test or Component	835
How to Run a GUI Test with a Disconnected Remote Desktop Connection	838
UFT Runtime Engine - Overview	840
Test Batch Runner Overview	842
How to Create and Run a Test Batch	842
How to Run a Test Batch Using the Windows Command Line	844
Using UFT for Continuous Integration	845
Optional Steps	846
Default Optional Steps	846
Log Tracking	847
Manually configure log tracking settings	847
How to Set Optional Steps	849
Troubleshooting and Limitations - Run Sessions	849
Chapter 56: Using Run Results	853
UFT Run Results - Overview	854
Checkpoint and Output Value Results	861
How to Interpret Run Results	871
Chapter 57: Running Tests with Virtualized Services and Networks	876
Running Tests with Virtualized Services - Overview	877
Assigning Data and Performance Models to a Virtualized Service	877
Running a Test with a Network Emulation - Overview	878
How to Use a Virtualized Service for a UFT Test	879
How to Run a Test Using an Emulated Network	883
Chapter 58: Debugging Tests and Components	887
Debugging Overview	888
Considerations for Debugging Tests and Components	889
Running to a Step and Debugging from a Step	890
Modifying and Watching the Values of Variables and Properties of Objects During a Run Session	892
How to Debug Your Test, Component, Function Library, or User Code File	893
How to Debug a GUI Action or a Function - Exercise	897
How to Step Into, Out of, or Over a Specific Step in a GUI Test - Exercise	900
How to Debug an API User Code File - Exercise	903
Breakpoints	906
Enabling and Disabling Breakpoints	907
How to Use Breakpoints	908

Run Errors	910
Troubleshooting and Limitations - Debugging	911
Chapter 59: Running Tests with the Runtime Engine	913
Part 9: UFT Integration With HP ALM	914
Chapter 60: ALM Integration	915
ALM Integration Overview	916
How to Work with Tests and Components in ALM	917
Running UFT Tests from ALM	920
Running Tests in Server-Side Execution	920
AUT Environment Parameters	921
How to Run a Test Using Server-Side Execution	922
ALM Template Tests	925
How to Create a Template GUI Test	926
How to Create a GUI Test in ALM Using a Template Test	926
Data Awareness in ALM	928
How Does Data Awareness Work?	929
Advantages of Data Awareness	930
Considerations for Data Awareness	930
How to Data Drive a Test Using Data Stored in ALM	931
How to Enable the BPT Remote Agent	935
How to View or Modify Remote Agent Settings	936
How to Install a Certificate for ALM Servers in a Environment Using External Authentication	937
Troubleshooting and Limitations - General ALM Integration	937
Troubleshooting and Limitations - ALM Integration with GUI Testing	940
Troubleshooting and Limitations - ALM Integration with API Testing	941
Chapter 61: Resources and Dependencies Model	942
Resources and Dependencies Model Overview	943
Resources and Dependencies Model Terminology	943
Asset Dependencies - Advantages	945
Relative Paths for Tests and Resources Saved in ALM	947
Updating Relative Paths for Tests and Resources Saved in ALM	948
Troubleshooting and Limitations - Resources and Dependencies	950
Chapter 62: Version Control in ALM	951
Managing Versions of Assets in ALM	952
How ALM Manages Assets	953
Version History Versus Baseline History	955
Asset Comparison Tool and Asset Viewer	955
How to Use ALM Version Control	958
How to Work with the Asset Comparison Tool and Asset Viewer	960

Troubleshooting and Limitations - ALM Version Control	964
Troubleshooting and Limitations - Asset Comparison Tool	965
Chapter 63: HP Sprinter	967
Part 10: Business Process Testing in UFT	969
Chapter 64: Business Process Testing in UFT	970
Business Process Testing Overview	971
Business Process Test and Flow Overview	971
Business Process Testing in UFT - Overview	972
Comparing BPT in UFT to BPT in ALM	973
Business Process Testing Methodologies	974
How to Set Up UFT for Business Process Testing	978
How to Create, Maintain, and Run Business Process Testing Tests and Flows in UFT	981
Business Process Testing in UFT - End-to-End Scenario	984
Business Process Testing in UFT User Interface	993
Business Process Testing in UFT - BPT View	996
Troubleshooting and Limitations - Business Process Testing in UFT	1001
Chapter 65: Business Components	1002
Business Components - Overview	1003
Business Component Categories	1004
Keyword GUI Components Overview	1005
Converting Manual Components	1006
Tips and Guidelines for Selecting Application Areas for Your Component	1007
Scripted GUI Components Overview	1007
How to Create and Manage GUI Business Components	1008
How to Convert a Keyword GUI Component to a Scripted GUI Component	1011
GUI Component Statuses	1012
Troubleshooting and Limitations - Business Components	1013
Chapter 66: Application Areas	1014
Application Areas - Overview	1015
How to Create and Manage Application Areas	1016
Application Area User Interface	1018
Troubleshooting and Limitations - Application Areas	1020
Chapter 67: Creating Business Process Test Steps	1021
Creating Steps in a Business Process Test - Overview	1022
How to Create Test Steps in a Business Process Test	1023
How to Record a Business Process Test	1025
How to Add Test Objects to a Component with Capture	1027
Chapter 68: Using Data in Business Process Testing	1031
Using Data in Business Process Tests - Overview	1032
Business Process Testing Parameter Categories	1033

Linking Parameters in Business Process Tests	1034
Linking Parameters When Running Iterations in a Business Process Test	1035
Linking Parameters in a Business Process Test - Examples	1036
Promoting Parameters in a Business Process Test	1038
How to Use Data in a Business Process Test	1039
Troubleshooting and Limitations - Using Data with Business Process Testing in UFT	1044
Chapter 69: Running Business Process Tests	1045
Running Iterations in a Business Process Test	1046
Running Iterations of Component Groups in a Business Process Test	1047
Considerations for BPT Test Iterations and Parameters	1050
Run Conditions Overview	1051
Running Business Process Tests with Test Configurations	1052
Data For Test Configurations	1053
Test Cases Generator Overview	1054
Use-Case Scenario: Test Cases Generator	1055
Learn more!	1059
How to Set Run Conditions for a Business Process Test	1059
How to Set Up and Run Test Configurations	1061
How to Use the Test Cases Generator to Create Test Configurations	1064
Chapter 70: Business Process Testing with the BPT Packaged Apps Kit	1067
BPT Packaged Apps Kit - Overview	1068
Learning Tests and Flows	1069
How UFT Reuses Learned Components	1071
Detecting and Resolving Changes	1072
How to Learn Business Process Tests and Flows	1073
How to Detect and Resolve Changes Using Change Detection Mode	1079
Business Process Testing with the BPT Packaged Apps Kit - Troubleshooting and Limitations	1083
 Part 11: Appendix	1084
Appendix A: UFT Terminology Quick Reference	1085
Appendix B: Developing Custom Comparers for Bitmap Checkpoints (GUI Testing)	1092
About Developing Custom Comparers for Bitmap Checkpoints	1093
Custom Comparator for Images Whose Location Changes in the Application - Use-Case Scenario	1093
Custom Bitmap Comparator Development	1094
How to Develop a Custom Comparator	1095
How to Implement the Bitmap Comparator Interfaces	1097
How to Install Your Custom Comparator and Register it to UFT	1100
How to Use the Bitmap Checkpoint Custom Comparator Samples	1102
How to Develop a Custom Comparator - Tutorial	1104

The Bitmap Checkpoint Comparer Interfaces	1113
The IVerifyBitmap Interface	1113
The IBitmapCompareConfiguration Interface	1114
Appendix C: GUI Checkpoints and Output Values Per Add-in	1116
Supported Checkpoints	1117
Supported Output Values	1120
Appendix D: Frequently Asked Questions for GUI Testing	1122
Creating Tests or Components	1123
Programming in the Editor and Working with Function Libraries	1123
Working with Dynamic Content	1125
Advanced Web Issues	1127
Standard Windows Environment	1130
Test and Component Maintenance	1131
Testing Localized Applications	1132
Improving GUI Testing Performance	1133
Appendix E: UFT Troubleshooting and Limitations	1137
Troubleshooting and Limitations - UFT Program Management	1138
Windows Notification Area Messages	1140
Troubleshooting and Limitations - Multilingual Applications in GUI Testing	1141
Troubleshooting and Limitations - Operating Systems in GUI Testing	1143
Troubleshooting - Naming Conventions	1144
Send Us Feedback	1147

Part 1: UFT Introduction

Chapter 1: What is UFT?

Relevant for: GUI testing and API testing

This chapter includes:

• UFT Overview	27
• UFT Program Use	29
• Licensing	29
• Required access permissions	29
• Demo applications	31
• Accessibility	31
• Unicode Compliancy	31

UFT Overview

Relevant for: GUI testing and API testing

Welcome to HP Unified Functional Testing (UFT), HP's solution for functional test and regression test automation, combined with functional testing for headless systems.

GUI testing

Use UFT's keyword-driven testing method to create GUI test steps early and maintain them with only minor updates. Add-ins enable you to test objects (controls) created in commonly used environments.

API testing

UFT's API (service) testing solution provides tools for the construction and execution of functional tests for headless (GUI-less) systems. For example, you can use UFT to test standard Web Services, non-SOAP Web Services, such as REST, and so on.

Create an API test by dragging and dropping activities from the UFT Toolbox pane into the test, displayed in the canvas. The toolbox provides a collection of activities for functional testing in areas such as REST, Web Services, JMS, and HTTP. You can add more activities to the toolbox by importing WSDLs or providing other contract definitions.

Integrated testing

UFT is able to provide cross-capability and cross product integration.

Integration between GUI and API Testing

Integrate your GUI and service testing processes in a single test by including calls from your GUI test to API tests, or from your API tests to GUI tests. When you insert a call to another test, the call is displayed as nested under the relevant action in the canvas.

For details about calling API tests from GUI tests, see "[Integration with API Tests](#)" on page 120.

For details about calling GUI tests from API tests, see "[How to Call External Tests or Actions](#)" on page 426.

Integration with ALM

Use UFT together with ALM to manage the entire testing process. For example, use ALM to:

- Create a project (central repository) of manual and automated tests
- Report and track defects

Tests and components created in UFT can be saved directly to your ALM project. Run UFT tests and review and manage the results in ALM.

For details see "[ALM Integration](#)" on page 915, and the *HP Application Lifecycle Management User Guide*.

Note: Unless otherwise specified, references to **Application Lifecycle Management** or **ALM** apply to all currently supported versions of ALM and Quality Center. Note that some features and options may not be supported in the specific edition of ALM or Quality Center that you are using.

For details, see the *HP Unified Functional Testing Product Availability Matrix* and the *HP Application Lifecycle Management User Guide* or the *HP Quality Center User Guide*.

Business Process Testing in UFT

Business Process Testing (BPT) works within UFT or ALM as a component-based testing framework. Working with a testing framework provides many advantages to enterprises, including streamlining the creation and maintenance of both manual and automated tests, and maximizing efficiency for testing complete business processes.

For more details, see "[Business Process Testing in UFT](#)" on page 969.

UFT Program Use

To check for software updates, patches, or service packs for UFT, visit the [HP Software Support Site](#). The **HP Update** program does not provide updates for UFT.

Licensing

Working with UFT requires a license. When you install UFT, you select one of the following license types:

- A permanent **seat** license that is specific to the computer on which it is installed (includes a 30-day demo license)
- A network-based **concurrent** license that can be used by multiple UFT users

You can change your license type at any time (as long as you are logged in with administrator permissions on your computer). For example, if you are currently working with a seat license, you can choose to connect to a concurrent license server, if one is available on your network.

For information on modifying your license information, see the *HP Unified Functional Testing Installation Guide*.

Note: You can also open UFT using a legacy license, although the functionality will be limited to the service that you are licensed to use. For example, you can open UFT using a legacy QuickTest Professional or Service Test license and access GUI testing or API testing functionality.

Required access permissions

Make sure the following access permissions are set to run UFT or to work with ALM.

Permissions Required When Working with UFT

You must have the following file system permissions:

- Full read and write permissions to the Temp folder
- Read permissions to the Windows folder and to the System folder
- Full read and write permissions to the folder on which you are saving solutions, tests, or run results
- Full read and write permissions to the <Program Files>\Common Files\Mercury Interactive folder
- If you are working on Windows 7 or Windows Server 2008 operating systems: Full read and write permissions to the <Program Data>\HP folder
- Full read and write permissions to the User Profile folders
- Full read and write permissions to the <Windows>\mercury.ini file

- Full read and write permissions to the following AppData folders:
 - %userprofile%\AppData\Local\HP
 - %appdata%\Hewlett-Packard\UFT
 - %appdata%\HP\API Testing

Note: Read/write permissions to these folders should also enable permission to any subfolders contained in the folders listed above. If not, the system administrator must grant administrative permissions to the subfolders contained in these folders.

You must have the following registry key permissions:

- Full read and write permissions to the keys under HKEY_CURRENT_USER\Software\Mercury Interactive or [HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Hewlett-Packard]
- Full read and write permissions to all the keys under HKEY_CURRENT_USER\SOFTWARE\Hewlett Packard
- Read and Query Value permissions to all the HKEY_LOCAL_MACHINE and HKEY_CLASSES_ROOT keys

Permissions Required When Working with ALM

You must have the following permissions to use UFT with ALM:

- Full read and write permissions to the ALM cache folder
- Full read and write permissions to the <Program Data>\HP folder
- Full read and write permissions to the UFT Add-in for ALM installation folder
- Administrative permissions for the first connection to ALM

Permissions Required When Working with Business Process Testing

Make sure you have the required ALM permissions before working with business components and application areas.

- To work with component steps in ALM, you must have the appropriate **Add Step**, **Modify Step**, or **Delete Step** permissions set. You do not need **Modify Component** permission to work with component steps. The **Modify Component** permission enables you to work with component properties (the fields in the component Details tab).
- To work with parameters in ALM or in a testing tool, you must have all the parameter task permissions set in ALM.
- To modify application areas, you must have the separate permissions for resources required for modifying components, and adding, modifying, and deleting steps. All four permissions are required. If one of these permissions is not assigned, you can open application areas only in read-only format.

For more information on setting user group permissions in the Business Components module, see the *HP Business Process Testing User Guide*.

Demo applications

Many examples in this guide use the Mercury Tours sample Web site. The URL for this Web site is:
<http://newtours.demoaut.com>.

Note that you must register a user name and password to use this site.

A sample Flight Windows-based application is also provided with the UFT installation. You can access it from:

- **Start > Programs > HP Software > HP Unified Functional Testing > Sample Applications > Flight API or Flight GUI**
- <UFT installation folder>/samples/Flights Application/FlightsGUI.exe (for the Flight GUI application)
- <UFT installation folder>/samples/Flights Application/FlightsAPI.exe (for the Flight API application)
- **Windows 8 and higher:** C:\Program Files (x86)\HP\Unified Functional Testing

Accessibility

Many operations are performed using the mouse.

In accordance with Section 508 of the W3C accessibility standards, UFT also recognizes operations performed using the **MouseKeys** option in the Windows Accessibility Options utility.

Additionally, you can perform many operations using shortcut keys.

Unicode Compliancy



Unified Functional Testing is Unicode compliant according to the requirements of the [Unicode standard](#), enabling you to test applications in many international languages.

Test non-English language applications as long as the relevant Windows language support is installed on the UFT computer.

Names and paths of tests and resources (for example, function libraries, object repositories, and recovery scenarios) are not Unicode compliant and therefore should be specified in English or in the language of the operating system.

Chapter 2: UFT Licenses

Relevant for: GUI testing and API testing

When using UFT, you must install a valid license on your computer. This license can be one of the following types:

- Seat (formerly known as a local or standalone license)
- Concurrent (also known as a floating license)
- Commuter

The license enables you to access specific UFT features, depending on the license used.

This chapter includes:

• UFT License Types	33
• Install UFT Licenses with the License Wizard	34
• Install UFT Licenses from the Command Line	39
• UFT Licensing FAQs	41
• UFT Licenses - Troubleshooting and Limitations	44

UFT License Types

When using UFT, you can choose between different types of licenses:

- **Seat**
- **Concurrent**
- **Commuter**

The table below summarizes the differences between the license types:

Topic	Seat Licenses	Concurrent Licenses	Commuter Licenses
General description	A machine-specific license linked to your computer.	A license taken from a license server on a per-session basis.	A license checked out for a period of time to use when you are not connected to the license server. You can check out a commuter license directly from a license server or have another user remotely check out a license for you.
Number of installations per license key	One	Unlimited The license server regulates the number of licenses in use at any given time. Note: You can install a special tool to track license usage (both for UFT and other products) across your network. This tool is available here: https://hpln.hp.com//contentoffering/usage-tracking	One for a limited period of time.
Other issues	The license key is based on a specific locking code per computer. Note: A computer with multiple bootable partitions may generate a different locking code for each partition. When obtaining a seat license key, you need to use the	You must have an active network connection to install and check out commuter licenses.	The license key is based on the machine identification. The license is specific for the computer making the request. You or another

	<p>locking code for the partition on which you want to use UFT.</p> <p>If you modify the MAC address or host name of the computer after installing a seat license, you must regenerate and install your seat licenses again.</p>		<p>user must have an active network connection to install and check out commuter licenses.</p>
Entering the license key in UFT	<p>A seat license key needs to be entered only once.</p>	<p>Each time UFT starts, UFT tries to connect to the License Server for an available license.</p>	<p>A commuter license key needs to be entered only once.</p> <p>After the commuter license expires, UFT automatically returns to the previously used license type.</p>

Install UFT Licenses with the License Wizard

Note: Before upgrading to UFT 12.50 from a previous version of QuickTest, Service Test, or UFT, you must upgrade your license. Customers with a valid Entitlement Order Number or Numbers can obtain the new license key(s) in the HP Licensing portal, found at

<https://h30580.www3.hp.com/poeticWeb/portalintegration/hppWelcome.htm?lang=en&cc=us&hp>.

Detailed instructions on how to use the Licensing Portal are provided at the top of the License Portal window.

Note that your previous UFT license keys will not work with UFT version 12.50.

The Functional Testing License Wizard enables you to install or check out a seat, concurrent, or commuter license for UFT. You can also use the wizard to switch from one license type to another.

When you first install UFT, it uses the demo license. To continue after the demo license period, you must install a valid license. To install a license, use the Functional Testing License Wizard.

You can reach the License Wizard in any of the following ways:

- **Start > All Programs > HP Software > Unified Functional Testing > Tools > Functional Testing License Wizard**
- From the warning message displayed when starting UFT
- By selecting **Help > License Wizard**.
- **Windows 8 only: C:\Program Files (x86)\HP\Unified Functional Testing**

When you start the License Wizard, it displays the currently installed license. You can also view license information by selecting **Help > About Unified Functional Testing** and clicking the **License** button.

UFT also displays a warning message in the status bar at the bottom of the UFT IDE if your license is going to expire. If you have multiple installed licenses, UFT displays the status of the license closest to the expiration date.

Note: You can also install UFT licenses via command line. For details, see "[Install UFT Licenses from the Command Line](#)" on page 39.

You can do the following using the license wizard:

- "[Install a Seat license](#)" below
- "[Install a Concurrent license](#)" below
- "[Check out and install a Commuter license](#)" on the next page
- "[Return a Commuter license](#)" on page 37
- "[Check out and install a Remote Commuter license](#)" on page 38
- "[Return a Remote Commuter license](#)" on page 39

[Install a Seat license](#)

1. In the License Wizard start screen, select **Seat license**.
2. In the Seat License installation screen, do one of the following:
 - Click **Load License Key File** and select your license key file. The seat license key file has a .dat extension.
 - Paste the license key in the edit field and click **Verify**.
- If you don't yet have a license key, expand the **How can I get a license key file** section for instructions.
3. After verifying that the license key is valid, click **Install**.
4. Click **Exit Wizard** to exit the wizard. To apply the new license, you must restart UFT.

Note:

- If you install a time-limited seat license, do not modify the date on your computer. Doing so will block your active seat license and prevent future UFT seat license installations on that computer.
For questions about this issue, contact your HP license supplier.
- If you modify the MAC address or host name of the computer after installing a seat license, you must regenerate and install your seat licenses again.

[Install a Concurrent license](#)

1. **Prerequisite:** Make sure you are connected to the network and can access the License Server.
2. In the License Wizard start screen, select **Concurrent license**.

3. In the Concurrent License Installation screen, enter the License Server address in the format <license server address>:<port>. If you do not enter a port number, the default port, **5814** is automatically inserted after the server address.

Note: The format of the address must be the same as the format used in the **Main** tab of the Configuration pane in the License Server. For details on setting up the address of your license server, see the *Autopass License Server User Guide*, included with the License Server installation in the UFT installation.

4. Click **Connect** to connect UFT to the License Server.

5. (Optional) Expand the **Add Redundant Server** link.

Enter the address for the redundant License Server. If your primary License Server is unavailable, UFT will connect to the redundant License Server to obtain a license.

Note:

You do not need an additional license for the redundant license server. You install the same licenses on the primary server and the primary and redundant servers sync the licenses.

The primary and redundant license servers are automatically synchronized as part of their setup and configuration. For details on setup and configuration of a redundant license server, see the *Autopass License Server User Guide*.

6. From the product license drop-down list, select the appropriate license and click **Install**.

7. Click **Exit Wizard** to exit the wizard. To apply the license, you must restart UFT.

Check out and install a Commuter license

If you usually use a concurrent license, but cannot connect to a License Server (for example, during a business trip), you can install a commuter license for those periods when you cannot access the License Server. Once you install the commuter license, you can use UFT without an active network connection.

You must have access to a License Server to check out a commuter license for yourself. If you do not have access to the License Server, see the section on [installing remote commuter licenses](#) below.

Note: Commuter licenses can be checked out only if your License Server has available concurrent licenses.

To install a commuter license:

1. **Prerequisite:** Make sure you are connected to the network and can access the License Server.
2. In the License Wizard start screen, click the **Additional Options** drop-down link.
3. Select **Commuter License**.
4. In the Commuter License Installation screen, enter the License Server address with the format <license server address>:<port>. By default, port number 5814 is used.

Note: The format of the address must be the same as the format used in the **Main** tab of the Configuration pane in the License Server. For details on setting up the address of your license server, see the *Autopass License Server User Guide*, included with the License Server installation in the UFT installation.

5. Click **Connect** to connect to the License Server.
6. After the list of available licenses is displayed, ensure that **Available** is selected below the License Server address field.
7. From the list of available licenses, select the licenses you need.
8. In the **Check out licenses for (days)** field, enter the number of days for which you need the commuter license.

Note: You can check out a commuter license for a maximum of 180 days.

9. Click **Check Out**. The license is checked out and is immediately displayed in the **Checked Out** section.
10. Click **Next** to install the license.
11. Click **Exit Wizard** to exit the wizard. To apply the concurrent license, you must restart UFT.

Note: The commuter license check out time always ends at 23:59 of the expiration day. Thus, if you check out a license for X days and start using it immediately in UFT, the Add-in Manager will display X days + Y hours (where Y is the number of hours until midnight).

Return a Commuter license

After you are finished using a commuter license, you should return the license to the License Server to make it available for other users.

You must have access to a License Server to return the commuter license. If you do not have access to the License Server, see the section on [installing remote commuter licenses](#) below.

To return a commuter license to the License Server:

1. **Prerequisite:** Make sure you are connected to the network and can access the License Server.
2. Select **Commuter License**.
3. In the Commuter License Installation screen, the License Server address should already be displayed and connected.

If needed, enter the License Server address with the format <license server address>:<port>. By default, port number 5814 is used and click **Connect** to connect to the License Server.

Note: The format of the address must be the same as the format used in the **Main** tab of the Configuration pane in the License Server. For details on setting up the address of your license

server, see the *Autopass License Server User Guide*, included with the License Server installation in the UFT installation.

4. After the list of available licenses is displayed, ensure that **Checked Out** is selected below the License Server address field.
5. Click **Check In All Licenses**. The list of checked out licenses is cleared.

Note: If you do not want to return all the licenses that you checked out, you must still return all your checked out commuter licenses and then re-check out the licenses you need.
6. Click **Next**. The license wizard reports that the license type was switched back to the previous license type (either seat or concurrent). The next time you open the license wizard, it displays the relevant type as the active license.
7. Click **Exit Wizard** to exit the wizard. To apply the return of the commuter license and revert to your previous license, you must restart UFT.

Note: If you do not check in your commuter license before the expiration date, UFT automatically reverts to the previous license state when the commuter license expires.

Check out and install a Remote Commuter license

If you need a commuter license when you are not able to connect with your License Server to check one out, you can use a **Remote Commuter** license. In this case, you generate a request, and then another user who has access to the License Server checks out the license and sends you the required key.

Note: Remote commuter licenses can be checked out only if your License Server has available concurrent licenses.

To install a remote commuter license when you do not have access to the License Server:

1. In the License Wizard start screen, expand the **Additional Options** drop-down link.
2. Select **Remote Commuter license**.
3. In the Remote Commuter License Installation screen, ensure that **Generate Request File** is selected.
4. From the list of available licenses, select the license you need.

Note: You can check out multiple types of licenses.

5. In the **Check out licenses for (days)** field, enter the number of days for which you need the commuter license.

Note: You can check out a remote commuter license for a maximum of 180 days.

6. Click **Generate Request File**. In the save dialog, the location for the request file (with a .lcor file

extension) is displayed.

7. Send the request file to a License Server administrator or to a user with access permissions to the License Server. The other user checks out a license key file for you using the generated request file. For details on checking out commuter licenses from the license server, see the *Autopass License Server User Guide*.
8. When you receive the license key file from the other user, return to the Remote Commuter License Installation screen. Ensure that **Install License** is selected.
9. Click **Choose File**. In the Open dialog, navigate to the location where you stored the license key file.
10. Click **Install**.
11. Click **Exit Wizard** to exit the wizard. To apply the commuter license, you must restart UFT..

Return a Remote Commuter license

1. In the License Wizard start screen, expand the **Additional Options** drop-down link.
2. Select **Remote Commuter license**.
3. In the Remote Commuter License Installation screen, ensure that **Generate Request File** is selected.
4. In the Remote Check In Request Generation screen, the list of currently checked out commuter licenses is displayed. In the Generation screen, click **Generate and Save Check In Request**. In the Save dialog, the location for the check in request file (with a .lcir file extension) is displayed.
5. Click **Next**. The license wizard reports that the remote commuter license is uninstalled and UFT reverts to the previous license type as the active license.
6. Click **Exit Wizard** to exit the wizard. To apply the return of the commuter license and revert to your previous license, you must restart UFT.

Note: If you do not check in your commuter license before the expiration date, UFT automatically reverts to the previous license state when the commuter license expires.

Install UFT Licenses from the Command Line

You can install a seat or concurrent license directly from the command line without opening the License Wizard. You can also use the command line to check the status of licenses in the License Server.

To install a license from the command line:

In a command window, enter the following command, followed by the relevant parameters as described below:

```
"<UFT installation directory>\bin\HP.UFT.LicenseInstall.exe"
```

Seat license	• seat "<license key string>"
--------------	-------------------------------

	<p>Note: If the license key contains a quotation mark character ("") in the license key string, make sure to add a backslash character (\) before the quotes.</p> <ul style="list-style-type: none"> seat "<path to the license key file>"
Concurrent license	<pre>concurrent <license ID> <license version> <server name/address> [<redundant server name/address>] [/force]</pre> <p>Note:</p> <ul style="list-style-type: none"> The <server name/address> or <redundant server name/address> uses the format server name/address:port. The port number is optional. Default = Port 5814. The <server name/address> or <redundant server name/address> must be in the same format as the server name or address listed in the Main tab of the Configuration pane of the License Server. For more details on setting up the License Server address, see the <i>Autopass License Server User Guide</i>. Both the [<redundant server name/address>] and /force parameters are optional. The /force parameter saves the license installation information even if the current installation fails. In subsequent UFT sessions, UFT will check the listed license server for the listed license.
Change server connection information	<p>Change the primary License Server address: config protocol.primary <http/https></p> <p>Change the secondary License Server address: config protocol.second <http/https></p>
To check the available licenses	<pre>licenses <server name/address> [<redundant server name address>]</pre>

Examples

Install a seat license key from a file saved locally:

```
"C:\Program Files (x86)\HP\Unified Functional
Testing\bin\HP.UFT.LicenseInstall.exe" seat "Downloads\HP UFT-licfile.dat"
```

Install a seat license key from a license key string:

```
"C:\Program Files (x86)\HP\Unified Functional
Testing\bin\HP.UFT.LicenseInstall.exe" seat "9CDG C9MA H9P9 8HW3 UXB5 HMWF Y9JL
KMPL B89H MZVU 6R4Q LHWE JHRP 3FQ3 CMRG HPMR MFVU A5K9 MWECC EKW9 HKDU LWNP SRL7
QPJQ YMM5 YQVW NV6G AG2A QZWD HY9B N4ZF BGWB B8GX 7YRF T8XT W7VB QW54 G83H 2TRY
KBTD EQUZ M8LB DZU7 WE6H 4NMU BG55 4XKB 27LX ATQB UKF8 3F9N JQY5 \" HP Unified
Functional Testing
```

Install a concurrent license:

```
"C:\Program Files (x86)\HP\Unified Functional  
Testing\bin\HP.UFT.LicenseInstall.exe" concurrent 11.11.111.111:5814 /force
```

Check the available licenses on a License Server:

```
"C:\Program Files (x86)\HP\Unified Functional  
Testing\bin\HP.UFT.LicenseInstall.exe" licenses 11.11.111.111:5814
```

UFT Licensing FAQs

This topic answers a number of frequently asked questions about using and installing UFT licenses:

- "[Can I use my old license \(from versions of UFT prior to 12.50\) with the new License Server?](#)" below
- "[Since I need a new license, how to do I get this new license?](#)" below
- "[If I have any trouble when regenerating licenses on the HP License Portal, what can I do?](#)" on the next page
- "[When I open the License Wizard, how do I know which type of license to select?](#)" on the next page
- "[How do I install the Autopass License Server?](#)" on the next page
- "[Can I install the License Server with silent installation?](#)" on page 43
- "[If I am using concurrent licenses, how do I get UFT to work with the License Server?](#)" on page 43
- "[How do I install licenses if I am deploying UFT across an enterprise network?](#)" on page 43
- "[How do I manage the concurrent licenses on the License Server?](#)" on page 43
- "[Can I set up my License Server to work with a redundant \(backup\) License Server?](#)" on page 43
- "[What is a cleanup license?](#)" on page 43

Can I use my old license (from versions of UFT prior to 12.50) with the new License Server?

No. UFT 12.50 has changed the license mechanism and the concurrent license server to the Autopass License Server. Prior versions of UFT used the Sentinel Concurrent License Server.

Note: The Autopass License Server and accompanying documentation is provided with the UFT Setup program.

In order to use your licenses with versions of UFT 12.50 and later, or to install them on the Autopass License Server, you need to upgrade your licenses. For details, see the topic on upgrading licenses in the *HP Unified Functional Testing Installation Guide*.

Since I need a new license, how to do I get this new license?

In order to use UFT 12.50 and higher, you have to upgrade your license. This enables you to convert your

old license to a license compatible with UFT 12.50 and higher and the new Autopass License Server.

You upgrade your license through the HP Licensing portal. For details, see the topic on upgrading licenses in the *HP Unified Functional Testing Installation Guide*.

If I have any trouble when regenerating licenses on the HP License Portal, what can I do?

Contact your regional support for HP Licensing. You can find the appropriate person for yourself here: <https://h30580.www3.hp.com/poeticWeb/portalintegration/hppWelcome.htm?stepaction=contactLicenseSupport>

When I open the License Wizard, how do I know which type of license to select?

In UFT, you can install a number of different license types:

- **Seat:** A machine-specific license that is used only by the computer on which the license is installed
- **Concurrent:** A multi-user license that is taken from a central License Server and returned after the user's session is complete
- **Commuter:** A machine-specific license that is checked out from a central License Server for a defined period of time. This license is either checked in to the License Server or expires.
- **Remote Commuter:** A machine-specific license that is checked out for one user by another user who has an active connection to the License Server.

In order to know which type of license you need, answer the following questions:

Scenario	License Type to Install
Are you assigned a specific license (with its own unique license key)?	Seat
Are you part of a group that uses licenses on an as-needed basis?	Concurrent. Note: You will need the IP address of your License Server where the licenses are installed.
Are you assigned the IP address from which to check out a license?	Concurrent
Are you travelling and will not have access to a license server?	Commuter
Are you already travelling and cannot access the License Server to get a license?	Remote Commuter

How do I install the Autopass License Server?

In the UFT Setup, there is a link to the **License Server** setup. If you click the link, a second window opens with links to install the License Server and view the *Autopass License Server User Guide*. The User Guide contains full instructions for setup and installation.

Can I install the License Server with silent installation?

Yes. The UFT installation uses the LICSVR command (as in previous versions of UFT).

For details on silent installation, see the silent installation section of the *HP Unified Functional Testing Installation Guide*.

If I am using concurrent licenses, how do I get UFT to work with the License Server?

In the UFT License Wizard, if you select **Concurrent license**, you need to enter the License Server IP address. This checks the connection between UFT and the License Server, and also gives you a list of possible licenses to install.

After you initially install the license, UFT checks the specified License Server address each time UFT starts and takes the requested license.

How do I install licenses if I am deploying UFT across an enterprise network?

UFT provides a command-line tool that enables you to install UFT licenses without using the License Wizard interface. For details on the commands to install these licenses, see .

The command line license installation is supported for seat and concurrent licenses.

How do I manage the concurrent licenses on the License Server?

The Autopass License Server has a full Web-based interface that enables you to install, manage, and administer all your licenses (both concurrent and commuter). You can see full details on how to use and manage this License Server in the *Autopass License Server User Guide*, provided with the UFT Setup program (in the **License Server** link).

You can also install a special tool to track license usage (both for UFT and other products) across your network. This tool is available here: <https://hpln.hp.com//contentoffering/usage-tracking>.

Can I set up my License Server to work with a redundant (backup) License Server?

Yes. You need to install the License Server on two separate servers, and then set one server to be the primary and the other to be the redundant server. This configuration is done in the Autopass License Server Web UI.\

You also can supply this information to UFT in the License Wizard, which enables UFT to take a concurrent license from the redundant License Server in the event that the primary License Server is not available.

For details on the redundant License Server setup, see the *Autopass License Server User Guide*.

What is a cleanup license?

If your computer is clock-tampered after installing the License Server, both the License Server and UFT's connection to the License Server do not work. In this case, you must get a cleanup license for your License Server. This enables you to reset all license capabilities.

For details on cleanup licenses, contact your HP license supplier.

UFT Licenses - Troubleshooting and Limitations

Relevant for: GUI testing and API testing

- If you install a time-limited seat license, do not modify the date on your computer. Doing so will block your active seat license and prevent future UFT seat license installations on that computer.
For questions about this issue, contact your HP license supplier.
- The License Server does not support the use of Network Address Translation (NAT).
- The concurrent license does not include a demo license and does not work without an active connection to a License Server and an installed license key.
- You must have administrator permissions to change the license type from seat to concurrent or vice versa.
- If you modify the MAC address or host name of the computer after installing a seat license, you must regenerate and install your seat licenses again.

Chapter 3: UFT Document Management

Relevant for: GUI tests and components and API testing

This chapter includes:

• UFT Test Documents	46
• UFT and Version Control Systems	47
• Relative Paths for Test Resources	49
• Portable Copies of Tests	50
• Opening Tests with Locked Resources	51
• Upgrading Documents From Previous Versions	51
• Upgrading QuickTest tests or components	51
• For Service Test tests or components	52
• Known Issues - Opening Documents	56

UFT Test Documents

Relevant for: GUI tests and components and API testing

A testing **document** is any file that enables you to test your application. You create, maintain, and edit steps, parameters, functions, and other details in the document to enable the testing of your application.

Testing documents can include:

GUI testing documents	<ul style="list-style-type: none">• GUI tests• GUI actions• Function libraries
API testing documents	<ul style="list-style-type: none">• API tests• API actions• C# files containing event handler code (TestUserCode.cs files)• C# files containing user-generated code
BPT documents	<ul style="list-style-type: none">• Business process tests• Business process flows• Business components, including API, keyword GUI, and scripted GUI components• Application areas

Work with testing documents in the "["Document Pane"](#)", which supports the following types of views and functionality:

Canvas	Graphically displays and enables you to edit the flow of your test, action, or component.
Keyword View	Enables you to create and view the steps of your action or component in a keyword-driven, modular, table format.
Editor	Provides text and code editing features that facilitate the design of your text, script, and code documents.
Application Area	Displays and enables you to edit the application area settings and resource associations.
BPT in UFT	Displays and enables you to create and edit business process tests and flows that are stored in ALM.

UFT and Version Control Systems

Relevant for: GUI tests, API tests, and function libraries

Work with version control systems, such as SVN or GIT, directly from UFT.

Note:

If you are using SVN, you must use SVN version 1.8.x.

If your documents and resources are saved in an ALM project or a version-controlled ALM project, see "[Version Control in ALM](#)" on page 951

Set up user credentials

Before you can begin working with UFT and your version control software, you must enter your user credentials.

SVN	You are prompted for your user credentials the first time you add, update, or commit to an SVN repository from UFT.
GIT	You are prompted for your user credentials the first time you push to the remote repository.

Update changes for a document

Update changes directly from the Solution Explorer.

Only documents that stand alone - those not dependent on a parent document, like an action to a test - can be updated directly from UFT.

For example, if you want to update changes from one action in a test or a local object repository in a test, you must commit the test.

To update a document, do the following:

SVN	Right-click the document name (or the parent document name), and select Update .
GIT	Make sure that the folder containing the document is synced with the Git repository. Right-click the document name and select Git Pull .

The first time that you update or commit to an SVN repository in UFT, you will be prompted for your user credentials. You can save these credentials to avoid entering them each time you update or commit.

Note: If you need to clear your user credentials, select **Tools > Clear All Credentials**. Your user credentials for the SVN or GIT repository are removed and you will be prompted to enter them on the next update or commit.

Commit changes for a document

Commit changes directly from the Solution Explorer.

Only documents that stand alone - those not dependent on a parent document, like an action to a test - can be committed directly from UFT. For example, if you want to commit changes from one action in a test or a local object repository in a test, you must commit the test.

To commit changes, do the following after creating the test or document in UFT:

SVN	In the Solution Explorer, right-click the document (or parent document) name and select Commit .
GIT	<ol style="list-style-type: none">1. Make sure that the folder containing the document is synced with the Git repository.2. In the Solution Explorer, right-click the document and select Git Commit. This adds the test/document to the local repository (if necessary) and commits the changes. You can commit only tests and external resource files (like a function library, object repository, or recovery scenario) to a repository.3. (Optional, when committing to the remote repository) In the Solution Explorer, right-click the document and select Git Push.

Note: If you have external documents associated with a test (such as an external action or a function library), the external documents are not saved and committed when you commit the test. You must save the external document separately.

Compare a document with the repository version

Use the default diff functionality, or specify another diff tool, including the UFT Asset Comparison Tool.

These tools are specified in the **Version Control Systems** pane of the Options dialog box (**Tools > Options > General tab > Version Control Systems** node).

Do the following to perform a diff comparison of your documents:

1. In the **Version Control System** pane of the Options dialog box (**Tools > Options > General tab > Version Control Systems** node), specify a diff tool for each type of document.
2. In the Solution Explorer, right-click the document name, and select **Diff with Previous Version**.
The selected diff tool opens, enabling you to perform your diff comparison.

Revert a document

Right-click the document name in the Solution Explorer, and select **Revert**. UFT reloads the previous version as saved in the local copy of the repository and your changes are removed.

Resolve conflicts between document versions

When there are conflicts between document versions, UFT displays a dialog listing the conflicts, and enables you to choose which version of the document to save. You can also merge two versions of the document if necessary.

Do the following to resolve the conflicts:

1. In the conflict list dialog, select a document with repository conflicts.
2. At the bottom of the dialog, choose one of the following options:

Use My Changes	This option saves the changes you made to the document and overwrites the version saved in the repository.
Use Their Changes	This option takes the version in the repository and overwrites your changes.
Merge	<p>This option merges both versions together.</p> <p>Note: You must define a specific tool for merging in the Version System pane of the Options dialog box (Tools > Options > General tab > Version System pane).</p>
View in Explorer	Enables you to open the folder containing the document, where you can manually modify the conflicts.

3. Commit the updated version.

Relative Paths for Test Resources

Relevant for: GUI tests and components

Relative paths are useful if your test's resources are stored in the file system and you want other users or HP products to be able to run this test on other computers.

During a run session, UFT searches for the resource file in the folder for the current test or component, and then in the folders listed in the **Folders** pane of the Options dialog box (**Tools > Options > GUI Testing tab > Folders** node).

Note: If you are working with the Resources and Dependencies model with ALM, specify an absolute ALM path. For details, see "[Updating Relative Paths for Tests and Resources Saved in ALM](#)" on page 948.

UFT enables you to specify whether to convert the path to a relative path or store an absolute path. One of the following message boxes opens, depending on whether the path you specified, or a part of the path, exists in the **Folders** pane:

Path Exists in the Folders Pane

If the resource path you specify matches an existing search path in the Folders pane, you are prompted whether to define the path using only the relative part of the path.

In cases where a part of the path you enter matches more than one path in the Folders pane, the closest match is applied. For example, if both C:\Current_Version and C:\Current_Version\Libraries are defined in the search path list, the latter is applied.

Path Does Not Exist in the Folders Pane

If the resource path you specify does not match an existing search path in the Folders pane, you are prompted whether to add the resource's location path to the Folders pane and define the path relatively.

Portable Copies of Tests

Relevant for: GUI tests and API tests

You may need to create a portable copy of a test for use when you do not have access to a test's resources.

Portable GUI tests	<p>File > Save (Other) > Save with Resources</p> <p>Save a standalone copy of your GUI test and its resource files to a local drive or to another storage device. UFT creates a copy of the following and saves the files in the location you specify:</p> <ul style="list-style-type: none">• Source test.• Resource files, such as function libraries and shared object repositories.• Called actions, including actions stored in other tests. <p>UFT does not save the resource files associated with other tests. If you call actions from other tests, ensure that the relevant resources are associated with your test.</p> <ul style="list-style-type: none">• Called API tests.
Portable API tests	<p>Save a standalone copy of your API test and its resources to a local drive or to another storage device by saving the activity as a local activity.</p> <p>To save a copy of an API test with its resources, do the following:</p> <ol style="list-style-type: none">1. In the Toolbox pane, right-click on the service name and select Move to > File System Activities. The service moves to the File System Activities section of the Toolbox pane.2. Select File > Save (Other) > Save with Resources.

Tip: If you use UFT with a concurrent license but do not have access to the concurrent license server (for example, during a business trip), install a commuter license.

For more details, see the *HP Unified Functional Testing Installation Guide*.

Opening Tests with Locked Resources

Relevant for: GUI tests only and API tests

The **Locked Resources** message box opens when you try to open a document that is associated with a locked resource file, such as a locked data table or object repository file.

Open the document as one of the following:

Read-Only (recommended)	Opens the document in read-only mode, enabling you to run it but not make any changes to it. You can: <ul style="list-style-type: none">Save the document under a new name, and then edit the document.Run the document, and choose to save the results file.
Read-Write	Opens the document in read-write mode, enabling you to run and edit it, and save any changes you make to it. However, any changes to referenced and locked UFT files cannot be saved. Locked resource files are opened in read-only mode even if the referencing document is opened in read-write mode.

Upgrading Documents From Previous Versions

If you have documents created in QuickTest, Service Test, or previous versions of UFT, you can upgrade them for UFT.

Upgrading QuickTest tests or components

• **Files last saved in QuickTest 9.5**

UFT will open the asset in read-only mode.

• **Files last saved in QuickTest earlier than 9.5**

An error message is displayed in the UFT Error Pane.

- If the asset is saved in the file system, you must first open it in QuickTest 10.00 or 11.00 to upgrade it.
- If the asset is saved in Quality Center or ALM, you must use the QuickTest Asset Upgrade Tool for HP ALM (available with the QuickTest Professional 10.00 or 11.00 DVD).

• **Files saved in QuickTest 10.00 or later**

Assets last saved in QuickTest 10.00 or later can be opened in UFT.

• **QuickTest components**

Sharing values between components in a BPT test cannot be done using user-defined environment variables.

Instead, use user-defined run-time settings that you create using the **Setting.Add** method.

If your QuickTest components used any user-defined environment variables, you must change them to use run-time settings when you upgrade to UFT.

- **QuickTest 11.00 and Chrome**

If you previously had QuickTest 11.00 installed on your computer and you installed one of the patches or hotfixes that added support for working with the Google Chrome browser (**QPTWEB00088** or another Chrome-related patch or hotfix), you must delete your user profile in the Chrome browser before you can use UFT to test applications in Chrome.

To do this, open the Chrome Settings window in your Chrome browser. In the Users section, click the **Delete this user** button.

- **RunScript methods**

If you previously used the **Frame.RunScript**, **Frame.RunScriptFromFile**, **Page.RunScript**, or **Page.RunScriptFromFile** methods in your tests of a Web site or Web application, you should update the RunScript argument to use either an eval function or an anonymous function.

For example, if you used this syntax previously:

```
Browser("MySearchEngine").Page("MySearchEngine").Frame("Web
Search").RunScript "var remove = document.getElementById('logo');
remove.parentNode.removeChild(remove)"
```

you should update this to:

```
Browser("MySearchEngine").Page("MySearchEngine").Frame("Web
Search").RunScript "eval(var remove = document.getElementById('logo'));
remove.parentNode.removeChild(remove);;"
```

OR

```
Browser("MySearchEngine").Page("MySearchEngine").Frame("Web
Search").RunScript "(function(){var remove = document.getElementById('logo');
remove.parentNode.removeChild(remove;});})();"
```

For Service Test tests or components

UFT12.52 provides a Batch Upgrader command line tool, **STBatchUpgrader.exe**, located in the **<UFT installation>/bin** folder.

This tool lets you run a batch file to upgrade tests last saved in Service Test, version 11.10 or 11.20, making them compatible for UFT12.52.

- If you do not upgrade your tests with the Batch Upgrader tool, when you open a test created in version 11.10 or 11.20, it prompts you to upgrade the test.
- For tests and components created in Service Test 11.00, you must first open and save them in Service Test 11.10, before you can upgrade them to UFT12.52.

Upgrade a test last saved in Service Test version 11.10 or 11.20

1. Make sure that UFT is not running.

If you ran the upgrader tool once while UFT was running, the logs may become corrupted. Backup and delete all of the existing logs in the <Unified Functional Testing installation>\bin\logs folder before proceeding.

If desired, create a backup copy of the older tests.

2. In the command line, enter the location of the STBatchUpgrader.exe file in the Unified Functional Testing/bin sub-folder.
3. Add the relevant command line options as described in "[API Test Batch Upgrader Command Line Options](#)" below.

Use the following syntax:

```
STBatchUpgrader.exe source [destination] [/ALM url domain project] [/login  
username password] [/log logfile] [/report reportfile]
```

For example, the following string runs the upgrade on all tests in the ST_11_1 folder on the ALM server, pumpkin, for the TEST1 project in the AUTOMATION domain. It places the report in c:\logs\MyLogFile.log.

```
STBatchUpgrader.exe Subject\ST11_1 /ALM http://pumpkin:8080/qcbin AUTOMATION  
TEST1 /login user password /log c:\logs\MyLogFile.log.
```

4. Run the command. Verify the validity of the tests in the destination folder.

API Test Batch Upgrader Command Line Options

The following table describes the command line options:

UI Elements	Description
/ALM	Indicates that ALM connection information will follow.
/log	Indicates that the log will be written to an alternate file. By default, log files are stored in the <UFT installation>\bin\logs folder.
/login	Indicates that login information will follow.
/report	Instructs the upgrader tool to generate a summary report.
destination	<ul style="list-style-type: none">• For tests on the File system: The full UNC path of the folder in which to store the tests after the upgrade.• For tests in ALM: a target folder path under the Test Plan module, in which to store the upgraded tests.

UI Elements	Description
	<p>Note:</p> <ul style="list-style-type: none"> The destination value must be a local or remote folder with the same write access permissions as the source path. The destination folder should be an empty folder that does not include any tests. If you do not specify a destination folder, the tests will be upgraded in the source folder, overwriting the originals.
domain	The name of an ALM domain in which the source tests are stored.
logfile	The full path of the file to which the log will be written.
project	The name of an ALM project in which the source tests are stored.
reportfile	The full path of an existing folder, with the file name including an extension, to where the summary report should be written. For example, c:\logs\MySummary.txt.
source	<ul style="list-style-type: none"> For tests on the File system: The full UNC path of the folder containing the tests to be upgraded. For tests in ALM: a source folder path under the Test Plan module).
url	The URL of an ALM instance to which to connect in the following form: http://{instance_domain}:8080/qcbin
username, password	For ALM mode, the username and password with which to log in to the ALM project.

If you were unable to upgrade

If the UFT API test upgrade was unable to upgrade the test, you may need to modify the event handler code to make it compatible with the current version:

- Change the user code file to **TestUserCode.cs**.
- Change the namespace at the beginning of the file to **Script**.
- Change the class definition to **public class TestUserCode : TestEntities**

For example:

```
namespace Script
{
    using System;
    using System.Xml;
    using System.Xml.Schema;
    using HP.ST.Ext.BasicActivities;
    using HP.ST.Fwk.RunTimeFWK;
    using HP.ST.Fwk.RunTimeFWK.ActivityFWK;
    using HP.ST.Fwk.RunTimeFWK.Utilities;
    using HP.ST.Fwk.RunTimeFWK.CompositeActivities;
```

```
using System.Windows.Forms;
using HP.ST.Ext.FTPActivities;

[Serializable()]
public class TestUserCode : TestEntities
...
```

Known issues with Service Test tests

Security	When upgrading a Service Test test from Service Test version 11.00 (via 11.10), UFT does not retain the Security settings. To upgrade the security settings as well, save the Security Scenario to an .stss file in Service Test 11.00. Then import this file for your service when you upgrade the test.
Call QuickTest Professional Test steps	Tests last modified in Service Test 11.20 that contain a Call QuickTest Professional Test step, cannot be opened in UFT.

Known Issues - Opening Documents

Relevant for: GUI tests and components and API testing

This section describes general troubleshooting and limitations for creating, opening and saving testing documents.

Hidden files

UFT does not support hidden files. If you mark a UFT test, or other UFT file as hidden (by selecting the **Hidden** attribute), UFT may behave unexpectedly.

Documents last saved in earlier versions

Last saved in QuickTest 9.5	Document opens in read-only mode. Open the document in QuickTest 10.00 or 11.00 to upgrade it.
Last saved in a QuickTest version earlier than 9.5	An error is displayed in the Error pane, and the document is not opened. Open the document in QuickTest 10.00 or 11.00 to upgrade it.
Last saved in QuickTest 10.00 or later	Can be opened and modified directly in UFT.

When upgrading a document:

- Relevant add-ins must be installed.
- Tests saved in the upgraded format can no longer be used with previous versions of QuickTest.

Tip: If you have many tests to upgrade, create a script that iterates through all your tests to open and save each one in the new format. For more details, see "["UFT Automation Object Model Overview" on page 741](#).

Opening and saving tests with resources

When you save a test together with its resources, UFT also saves any tests containing external actions.

However, these called tests contain only the specific actions called from your test, and not all the actions in the called test. These called tests cannot be opened from the local copy.

If you need full access to all called tests, as well as your main test, save all tests locally, and then manually edit all references to the relevant actions from the main test.

You cannot save a test with resources when the test calls an external action which, in turn, calls another external action.

Libraries in Windows 7 and higher

The Windows Libraries feature of is not supported. If you attempt to open a test from a library location or save a test to a library location, UFT may behave unexpectedly.

Browse to your saved tests using standard file paths, even if they are included in a library.

User Account Control (UAC)

If you open a test from a protected location (such as Program Files) and UAC is enabled, the test is opened in read-only mode.

To modify the test, copy it to another location and open the files from there.

Part 2: UFT Panes

Chapter 4: Active Screen Pane

Relevant for: GUI tests and scripted GUI components

The Active Screen pane enables you to view snapshots of your application as it appeared during a step in a recording session.

Use the Active Screen when you want to view the application state during a particular step in the test, or if you need to add steps, checkpoints, or output values after editing and running the test.

The Active Screen enables you to add these additional steps without the need to have the application open.

View objects, or insert steps, output values, and checkpoints, by right-clicking the object in the Active Screen and selecting the relevant option. To insert steps, select **Step Generator**.

To access	<ol style="list-style-type: none">Do one of the following:<ul style="list-style-type: none">Ensure that a GUI test, action, or component is in focus in the document pane.In the solution explorer, select a GUI test or component node, or one of its child nodes.Select View > Active Screen.
Important information	When you are recording on an MDI (Multiple Document Interface) application, the Active Screen does not capture information for non-active child frames. If you are testing an application using a UFT add-in, see the <i>HP Unified Functional Testing Add-ins Guide</i> to determine whether special Active Screen screen capture options exist for that environment.

Saving Active Screen content

Save the content of the Active Screen with your test if you want to edit the saved test directly from the Active Screen.

If you need to conserve disk space after you finish editing the test, and you plan to use the test only for test runs you can save the test again without the content of the Active Screen.

The Active Screen and Insight

- For steps that contain Insight test objects, the Active Screen provides only a visual reference to the application and the test object's context. The Active Screen displays the Insight test object highlighted within a screen capture of its parent object.

You cannot use the Active Screen of an Insight step to view object properties, to insert checkpoints or output values, or to add objects to the object repository.

- The OCR text recognition mechanism is not supported for objects in the Active Screen.

- When creating objects, checkpoints, or output values from the Active Screen for objects in a modal dialog box within a Web browser, they are created under the Browser object and not under the Window object.

Workaround: Add the object directly from the application, for example by using the **Add Objects to Local** button in the Object Repository window.

The Active Screen and Web-based applications

The Active Screen displays the captured HTML content using an Internet Explorer browser control, even if you ran your steps on another browser.

Additionally, depending on your settings, the Active Screen may capture your HTML page before or after various scripts ran on the page.

Some pages may look slightly different in the Active Screen than they appeared during your record or update run session, and some property values may be different or may not be available.

UFT stores the URL path to images and other resources on the page, rather than downloading and storing the images with your test. Therefore, you may need to provide login information to view password-protected resources in the Active Screen.

Stop saving Active Screen information

When you stop saving Active Screen information, Active Screen files are no longer saved, and you can no longer edit your test from the Active Screen.

- Open the relevant test.
- Select **File > Save As** and clear the **Save Active Screen files** check box.
- Click **Save** to apply your changes.

Update a single Active Screen capture

- Make sure that your application is displaying the window or page that you want to use to replace what is currently displayed in the Active Screen pane.
- In the Keyword View, click a step that you want to change. The window or page is displayed in the Active Screen pane.
- Select **Tools > Change Active Screen**. The UFT window is hidden and the mouse pointer becomes a pointing hand.
- Click the window or page displayed in your application.
- When a message prompts you to change your current Active Screen display, click **Yes**.

Chapter 5: Bookmarks Pane

Relevant for: GUI actions, scripted GUI components, function libraries, and user code files.

The Bookmarks pane displays the location of bookmarks in your action, scripted component, function library, or user code files, and enables you to navigate to these bookmarks.

Use bookmarks when editing files in the Editor. When you assign a bookmark, a bookmark  icon is added to the left of the selected line of your document.

To access	Select View > Bookmarks .
Important information	Bookmarks are saved, per solution, on your file system on a per-user basis. When working with documents saved in ALM, bookmarks are maintained until you close UFT.

User elements and bookmark pane functions are described below:

UI Element	Description
<bookmarks list>	Lists all bookmarks connected with the open solution. Double-click the bookmark line to navigate directly to the relevant line in your document.
	Insert/Remove Bookmark. Inserts or removes a bookmark in the current document in the line where the cursor is currently located.
 	Previous/Next Bookmark. Navigates to previous or next bookmark in the pane.
	Delete. Deletes the currently selected bookmark from the relevant document and from the list displayed in the pane.
	Clear All Bookmarks. Deletes all bookmarks you have added from all documents and from the list displayed in the pane.

Chapter 6: The Canvas

Relevant for: GUI tests and API testing

The canvas provides a visual representation of the GUI or API test flow. It opens as a tab in the document pane.

GUI tests in the canvas

Working with the canvas closed can improve UFT's performance when creating or opening a GUI test.

Drag actions to reorder them.

Right-click an element in the canvas to access settings, or to run or debug a test from a specific action.

API tests in the canvas

Manipulate steps by dragging them to reorder, copying and pasting, or deleting.

Add special steps, including:

Loop steps	Define loop step details in the Properties pane.
Two-branched conditional steps.	Add activities as part of the conditional branches.
Break steps	Move the test run to the step after a loop.
Continue steps	Begin the next iteration.

Test individual steps using the **Run Step** command.

Chapter 7: Data Pane

Relevant for: GUI testing, API testing, and business process testing

UFT enables you to insert and run steps that are driven by data, including mathematical formulas, in the Data pane.

Data table content

Enter data top to bottom and left to right, leaving no gaps of entire rows or columns.

Very large numbers in the data table may cause unexpected behavior.

UFT does not support colors, formatting, and special cell formats.

Data table values

Values returned from the data table are always converted to strings. Use VBScript conversion functions to convert values to other types.

For example:

```
Window("Flight Reservation").WinComboBox("Fly From:").Select CInt(DataTable  
("ItemNumber", dtGlobalSheet))
```

Changing column headers

If you change a data parameter (column header), also update the relevant name of the parameter wherever else it is used, such as checkpoint or output values, or any ALM parameter mappings.

Importing Excel files to the Data Pane

Excel files may be incompatible with UFT if they contain features or functionality unavailable for GUI tests. Remove the incompatible functionality and try again.

Special formatting in Excel files are not imported, and data is displayed in the Data pane with fixed values.

UFT does not support cross references to other Excel sheets.

For details on supported versions of Microsoft Excel, see the *HP Unified Functional Testing Product Availability Matrix*.

Data Pane Specifications

Item	Details
Maximum worksheet size	65,536 rows by 256 columns
Maximum number of worksheets	256 (255 sheets in addition to the Global data table).
Column width	0 to 255 characters
Text length	16,383 characters
Formula length	1024 characters
Number precision	15 digits
Largest positive number	9.9999999999999E307
Smallest positive number	1E-307
Largest negative number	-1E-307
Smallest negative number	-9.9999999999999E307
Maximum number of names per workbook	Limited by available memory
Maximum length of name	255
Maximum length of format string	255
Maximum number of tables (workbooks)	Limited by system resources (windows and memory)

See also:

- ["Data Awareness in ALM" on page 928](#)
- ["How to Data Drive a Test Using Data Stored in ALM" on page 931](#)

Chapter 8: Debug Panes

Relevant for: GUI actions, scripted GUI components, function libraries and user code files

After creating a test, component, function library, or user code file, you can check to see if they run properly. Using the debug panes, you can then debug your tests, components, function libraries, or user code files using UFT's debugging capabilities.

The debug panes are the primary interface for viewing information about your application during run sessions, as well as addressing any errors in the run session.

The debug panes include:

- Breakpoints Pane
- Call Stack Pane
- Loaded Modules Pane
- Threads Pane (Testing)
- Local Variables Pane
- Console Pane
- Watch Pane

Note:

For GUI testing: (Relevant only if Microsoft PDM 9.x or later is installed on your computer.) If you add Action automation objects to the Watch pane, and then close and re-open your test without closing and re-opening UFT, these actions may not load successfully in the re-opened test.

If this occurs, restart UFT and open your test.

See also:

- "[Debugging Overview](#)" on page 888
- "[How to Debug Your Test, Component, Function Library, or User Code File](#)" on page 893
- "[How to Debug a GUI Action or a Function - Exercise](#)" on page 897
- "[How to Debug an API User Code File - Exercise](#)" on page 903
- "[How to Step Into, Out of, or Over a Specific Step in a GUI Test - Exercise](#)" on page 900

Chapter 9: Document Pane

Relevant for: GUI tests and components and API testing

The document pane is the main design area in UFT and displays all open documents as separate tabs.

You use this pane to view and edit UFT documents.

To access	Create or open a document.
-----------	----------------------------

Documents open in the following views:

Canvas	Graphically displays and enables you to edit the flow of your test, action, or component. For details, see: " The Canvas " on page 62
Keyword View	Enables you to create and view the steps of your action or component in a keyword-driven, modular, table format. For details: " Keyword View User Interface " on page 110
Editor	Provides text and code editing features that facilitate the design of your text, script, and code documents. For details: <ul style="list-style-type: none">• "Editing Text and Code Documents" on page 627• "Programming in GUI Testing Documents in the Editor - Overview" on page 649• "How to Open a Window for Writing Custom Code" on page 765
BPT in UFT	Displays and enables you to edit business process tests and flows, including grouping components and flows, and modifying run order. For details: " Business Process Testing in UFT - Overview " on page 972
Application area	Displays and enables you to edit the application area settings and resource associations. For details: " Application Area User Interface " on page 1018

Chapter 10: Errors Pane

Relevant for: GUI tests and components, function libraries and API testing

The Errors pane lists syntax errors found in your testing documents. Double-click an error to locate its source.

To access	Select View > Errors .
-----------	----------------------------------

Errors may have the following severity levels: **Message**, **Warning**, or **Error**, and include the following types:

Code syntax errors

GUI testing	UFT checks for syntax errors when switching from the Editor to Keyword view and vice versa. Check manually by selecting Design > Check Syntax . View a description of each of the VBScript errors in the VBScript Reference.
API testing	See the Microsoft C# Reference for help resolving code errors.

Missing resources

Missing resources may cause a run to fail, and include:

- Missing GUI actions.
- Missing function libraries.
- Missing object repositories.
- Missing recovery scenarios
- Missing environment variable files
- Unmapped Shared Object Repository Parameter Values

Double-click an error description to resolve the error. Navigate to the missing resource and associate it with your test.

Note:

Missing actions are not displayed if a test is open in read-only format.

Resources located in password-protected areas are always listed as missing if they are opened after opening the test or application area.

Missing references

Reference files located in password-protected areas are always listed as missing if they are opened after opening the test.

To locate a missing reference file:

1. Locate the source of the missing reference files.
The relevant test code file opens and the cursor flashes at the place in which the missing reference file is called during a test run displaying the reference file's name.
2. In the Solution Explorer, right-click the **References** node located under an API test and select **Add Reference**.
3. Navigate to the missing reference file and associate it with your test.

Missing property values

For details on input properties for API test steps, see "[Standard Activities](#)" on page 406.

To locate a missing test step property value:

1. Locate the source of the missing reference files.
The field requiring the missing property value is highlighted in the Properties pane.
2. Enter the required information for the property value.

Chapter 11: Output Pane

Relevant for: GUI tests and components and API testing

The Output pane displays information set to the output in a test or component step. This includes:

- Details about assets that cannot be located or loaded during a run session.
- Information sent using the Print Utility statement during a GUI testing run session.
- Run-time logs of an API testing run session.

To access	Select View > Output . During a paused run session, click the Output Pane toolbar button  .
Important information	The Output tab may be unable to display the run results for very large tests, exceeding more than a thousand steps.

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
<Show output from>	The type of output to display: <ul style="list-style-type: none">• Build Displays all API test build information. Click a row to navigate to the relevant code.• Debug: Includes debug information, such as all Print command (print log) outputs and details about API tests called from GUI tests.
	Clear All Lines. Clears all of the messages from the message list.
	Toggle Word Wrap. Wraps the text of each message onto the next line.
Locate	Jumps to the location in the API testing source document relevant to the selected output message.
	<Find box> . The text string you want to find. Click arrows to jump to the previous or next instance. Refine your search using the following options: <ul style="list-style-type: none">• Match Case.• Match Whole Word.• Use Regular Expression. Note: Extended regular expressions and multi-line searches are not supported.
	Save Output to a Text File. Enables you to save the contents of the message list as a text file.

See also:

- ["Message Statements" on page 722](#)
- ["How to Run a Test or Component" on page 835](#)

Chapter 12: Properties Pane

Relevant for: GUI tests and components, API testing, and business process tests and flows

The properties pane is used for viewing and editing global and specific properties of your tests and components.

The view and information displayed in the Properties pane is different depending on what you are testing:

GUI testing	Properties of your test, component, and associated add-ins. Related parameters Tests saved in ALM that call a selected action.
API testing	The Properties pane is the primary place for providing the values your steps you use to run the test. The Properties pane displays tabs that enable you customize the step properties or parameters, including: <ul style="list-style-type: none">• General properties of the step• Input or checkpoint properties• Data source properties• Event handlers to use with your test step• HTTP and security information for your HTTP, SOAP, or Web Service call steps
Business Process Testing	The information displayed is dependent upon the type of business process document selected: <ul style="list-style-type: none">• When a business process test or flow is selected, the Properties pane displays basic information about the test or flow and enables you to manage the test or flow parameters.• When a component is selected in the document pane, the Properties pane enables you set On Failure conditions for the component as well as parameters for the individual component. You can also view the general properties for the component.

Chapter 13: Run Step Results Pane

Relevant for: API testing only

The Run Step Results pane displays the results of a single step performed using the **Run Step** command:

- For a simple built-in activity, it shows the status of the run.
- For SOAP messages, it shows the Input and Output envelopes.

To access	In the canvas, right-click a step and select Run Step . The Run Results Pane automatically opens displaying the run step result.
Important information	<ul style="list-style-type: none">• When working in non-English operating systems, certain entries in the Run Step Results pane are hard-coded in English and not translated.• The Run Step command is not supported for Java or JMS activities.

Chapter 14: Search Results Pane

Relevant for: GUI tests and components and API testing

The Search Results pane displays all occurrences of the search criteria you defined.

Note: Searches are not supported in the Keyword View or in the canvas.

To access	Do one of the following: <ul style="list-style-type: none">• Select View > Search Results to view the results of your last search.• In the Find dialog box, define the search criteria and click Find All.• Perform a search for API testing references using the Search menu options.
------------------	--

See also:

["Searching and Replacing in the Editor " on page 633](#)

Chapter 15: Solution Explorer Pane

Relevant for: GUI tests and components and API testing

The Solution Explorer displays the currently open solution, with its associated documents and their associated files.

A **solution** is a collection of testing documents and other resources, similar to a binder or notebook. Use solutions to organize your testing documents to help you perform comprehensive application testing.

For example, suppose you want to test a Web application for a flight booking application. You can create a solution containing several tests or components that verify various aspects of your application, such as logging in, booking a reservation, verifying the connection between your application and database, and verifying the transfer of booking information from your application to an airline server.

To access	Click the Solution Explorer toolbar button  .
Important information	<ul style="list-style-type: none">New documents are automatically assigned to a solution.To open a document that is part of the current solution without closing any other documents you currently have open, always open the document from the Solution Explorer.When using UFT on Windows Server 2012, some items are not displayed correctly in the Solution Explorer. Restart the computer to display the items correctly.Solutions stored on a network location are not locked when opened by UFT.

Chapter 16: Tasks Pane

Relevant for: GUI tests and components and user code files

The Tasks pane enables you to view and access TODO comments in GUI actions, GUI components, function libraries, or user code files.

TODO comments are reminders that are inserted as comments adjacent to the relevant steps in your testing document. For example, provide instructions to someone else during a handover, or you can remind yourself to do something.

To access	Select View > Tasks
Important information	Text display is limited to 260 characters.

Manage TODO comments

Add comments	In the Editor, insert comments adjacent to the relevant step in your document, beginning with any of the following: <ul style="list-style-type: none">• To Do• todo• to-do• TODO When you create event handlers in an API test, the editor automatically inserts TODO text, indicating where you need to enter your code.
Delete comments	Delete the source line in the document.
Sort comments	Click a column header.
Rearrange columns	Drag column headers.
Filter comments	Click Show Tasks From , and select a source document.
	<Show / Hide comments> . Toggle to show or hide comments stored in external actions or function libraries.
	Export comments. Click Export Task List

See also:

- "[Comments in the Keyword View](#)" on page 115
- "[Comments in VBScript](#)" on page 671

Chapter 17: Toolbox Pane

Relevant for: tests, components, actions, function libraries, and flows

The **Toolbox** pane contains items that you can use to create steps in your testing document.

The Toolbox pane and GUI testing

The Toolbox pane displays the test objects and functions available for the current action, component, or function library.

Add objects and functions to your document by dragging and dropping or copying and pasting.

Examples

- If you drag and drop a button object into an action or component, a step is added using the button with a **Click** operation (the default operation for a button object).
- If you drag and drop the function `count_items (obj, num)` into an action, component, or function library, a comment and a call to that function are added:

```
' count_items (obj, num)  
count_items
```

The Toolbox pane and API testing

The Toolbox pane provides a collection of service activities for functional testing. To add these activities to a test, drag and drop them in the test canvas.

Standard Activities	Standard API application activities, such as File System activities, Date/Time Activities, Network communication activities, and the like.
Local Activities	These are custom activities imported into the test by importing a WSDL/WADL file or creating the REST Service prototype.
File System Activities	These are custom activities that have been moved to the file system. By default, local activities are saved only with the test, but can be moved to a common location on the file system. Once they are moved to the file system, they are displayed as File System Activities.

Add additional activities to the Toolbox pane by importing WSDL and WADL files, or creating REST Service methods. Create new custom activities using the Activity Wizard tool.

See also:

- ["Standard Activity Types" on page 434](#)
- ["API Testing Extensibility" on page 603](#)

The Toolbox pane and Business Process Testing

The Toolbox pane displays the components and flows that are available for you to add the current business process test flow. Double click or drag and drop a component or flow to add it to the BPT test or flow that is open in the document pane.

Components and flows are organized according to their ALM path, and you can use the search bar to find a specific component or flow by name.

Part 3: UFT Configuration

Chapter 18: Global Options

Relevant for: GUI testing and API testing

The Options dialog box enables you to modify the general appearance and behavior of UFT. The values you set remain in effect for all documents and for subsequent testing sessions.

For example, you can define the user interface language, set startup options, or modify the font and colors of code elements in the Editor.

To access	Tools > Options
Important information	The Restore Factory Defaults button resets all Options dialog box options to their defaults.

In the Options dialog, set your options in the following panes:

Tab	Available Panes
General	<ul style="list-style-type: none">• Startup Options• Run Sessions Pane• Output Pane• Network Virtualization Pane• Version Control Systems Pane
GUI Testing tab	<ul style="list-style-type: none">• General Pane• Text Recognition Pane• Test Runs Pane• Folders Pane• Active Screen Pane• Screen Capture Pane• Insight Pane• Remote Connection Pane
API Testing tab	<ul style="list-style-type: none">• General Pane• Auto-Values Pane• SAP Connections Pane
BPT Testing tab	<ul style="list-style-type: none">• General Pane• Recording Settings pane• BPT Packaged Apps Kit Pane
Coding tab	<ul style="list-style-type: none">• Code Templates Pane
Text Editor tab	<ul style="list-style-type: none">• General Pane• Fonts and Colors Pane

Chapter 19: Document Settings

Relevant for: GUI tests and components

Use the Test or Business Component Settings dialog box or the Additional Settings pane in an application area to set testing options that affect how UFT works with a specific test or component.

Different panes and settings are available, depending on the document in focus (test, component, or application area), as well as the Add-ins loaded.

To access	<ul style="list-style-type: none">Have a test open in the document pane, or selected in the solution explorer.Select File > Settings.Create or open an application area.
Important information	The Restore Factory Defaults button resets all Options dialog box options to their defaults. Most business components inherit settings from the component's application area and are displayed as read-only in the Business Component Settings dialog box.

Note: You can also set testing options that affect all tests and components. For details, see "[Global Options](#)" on page 80.

The table below lists the panes contained in the Settings dialog box.

Node	Options
Properties	The properties of the test or component, for example, its description and associated add-ins. You can also set the status of a component.
Snapshot (components only)	Options for capturing or loading a snapshot image to be saved with the component for display in ALM.
Run (tests only)	The run session preferences for your test.
Applications (components only)	The Windows-based applications on which the component can record and run.
Resources	The resources associated with your test or component, such as function libraries, shared object repositories, and data tables.
Environment (tests/scripted components only)	Options for viewing existing built-in and user-defined environment variables, adding, modifying and saving user-defined environment variables, and selecting the active external environment variables file.
Recovery	Options for setting how UFT recovers from unexpected events and errors that occur in your testing environment during a run session.
Log Tracking	Options for activating and setting run-time preferences for tracking log messages generated by the log framework that monitors events occurring in your application.
Local System	Options for activating and setting preferences for tracking system counters during a run session.

Node	Options
Monitor	

Chapter 20: Set Options Programmatically

Relevant for: GUI tests and scripted GUI components

Use the **Setting** object in the Editor to control how UFT runs tests by setting and retrieving testing options during a run session.

Set an option using the following syntax:

```
Setting (testing_option) = new_value
```

To change an option, insert the **Setting** object statement at a relevant point in the action, such as after a specific page opens. Then, insert another statement with the **Setting** object to reset the changed setting before the next part of your test.

The defined setting remains in effect until it is changed again, either by another **Setting** statement or by modifying an option in the UI, or until the end of your current UFT session.

Note: If you make and save other changes in a related **Options** or **Settings** dialog box pane, the settings defined by the Setting statement are saved for your next session.

For detailed information on all the available methods and properties for the **Setting** object, see the **Utility Objects** section of the *HP UFT Object Model Reference for GUI Testing*.

Example: Setting options for an entire test

If you run the following statement with the Web Add-in loaded:

```
Setting("AutomaticLinkRun")=1
```

UFT disables automatically created checkpoints in the test.

This is the same as selecting the **Ignore automatic checkpoints while running tests or components** option in the **Web Advanced** pane of the Options dialog box.

If you run the following statement:

```
Setting("WebTimeOut")=50000
```

UFT automatically changes the amount of time it waits for a Web page to load before running a test step to 50000 milliseconds.

This is the same as setting the **Browser Navigation Timeout** option in the **Web** pane of the Test Settings dialog box.

Example: Setting options for a specific section of a test

If you want to change the **DefaultTimeOut** testing option to 5 seconds for objects on one Web page only, insert the following statement after the Web page opens in your test script:

```
'Keep the original value of the DefaultTimeOut testing option  
old_delay = Setting ("DefaultTimeOut")  
'Set temporary value for the DefaultTimeOut testing option  
Setting("DefaultTimeOut")= 5000
```

To return the **DefaultTimeOut** testing option to its original value at the end of the Web page, insert the following statement immediately before linking to the next page in the script:

```
'Change the DefaultTimeOut testing option back to its original value.  
Setting("DefaultTimeOut")=old_delay
```

See also:

["Programming in GUI Testing Documents in the Editor" on page 647](#)

Part 4: GUI Testing Design with the UFT IDE

Chapter 21: GUI Test Creation Overview

Relevant for: GUI tests and components

This chapter includes:

• Methodologies for Creating Tests	87
• Keyword-driven methodology	87
• Recording	87
• Importing tests from Sprinter	88
• Enhancing Your Tests	88
• Checkpoints	89
• Parameterization	89
• Output Values	89
• Programming Statements	89
• Active Screen Updates	90
• Editor and Keyword View - A Comparison	90
• Sample Test	91

Methodologies for Creating Tests

Relevant for: GUI tests only

You can create tests using the keyword-driven methodology, step recording, importing steps from HP Sprinter, or a combination of all of these methods.

Keyword-driven methodology

This methodology requires an infrastructure for all of the required resources, including shared object repositories, function libraries, and recovery scenarios.

Setting up the infrastructure requires in-depth knowledge of your application and a high level of UFT expertise.

Although setting up the infrastructure may initially require a longer time-investment in comparison to recording tests, using the keyword-driven methodology enables you to create tests that are more application-specific and have a more structured design.

This enables you to maintain your tests more efficiently and provides you with more flexibility than a recorded test.

Advantages of the keyword-driven methodology:

Designed at the business level	Keyword-driven testing enables you to design tests at a business level rather than at the object level. For example, UFT may recognize a single option selection in your application as several steps: a click on a button object, a mouse operation on a list object, and then a keyboard operation on a list sub-item. You can create an appropriately-named function to represent all of these lower-level operations in a single, business-level keyword.
Easier to read	Technical operations, such as a synchronization statement that waits for client-server communications to finish, are incorporated into higher level keywords. This makes tests easier to read and easier for less technical application testers to maintain when the application changes.
Separated resources and tests	Keyword-driven testing naturally leads to a more efficient separation between resource maintenance and test maintenance. This enables the automation experts to focus on maintaining objects and functions while application testers focus on maintaining the test structure and design.
Earlier start	Automation experts can add objects and functions based on detailed product specifications even before a feature has been added to a product. Using keyword-driven testing, you can begin to develop tests for a new product or feature earlier in the development cycle.

Recording

Let UFT generate test steps by recording the typical processes that you perform on your application.

As you navigate through your application, each step you perform is graphically displayed as a row in the Keyword View.

A step is anything a user does that changes the content of a page or object in your application, such as clicking a link or typing data into an edit box.

Recording may be easier for new UFT users or when beginning to design tests for a new application or a new feature.

Advantages of recording

Better for new users	Recording helps novice UFT users learn how UFT interprets the operations you perform on your application, and how it converts them to UFT objects and built-in operations.
Better for new applications or features	Recording can be useful for more advanced UFT users when working with a new application or major new features of an existing application. Recording is also helpful while developing functions that incorporate built-in UFT keywords.
Quick test creation	Recording can be useful when you need to quickly create a test that tests the basic functionality of an application or feature, but does not require long-term maintenance.

Importing tests from Sprinter

Sprinter, HP's manual testing solution, enables the manual tester to perform operations (user actions) on an application while Sprinter captures and saves information about each user action in the background. This process is similar to recording steps on your application in UFT.

After the Sprinter run session ends, the manual tester can export the captured user actions, test objects, and comments, to an automated test data file in XML format.

Import this file to UFT to convert it to a UFT GUI test with a local object repository.

This method helps to increase testing coverage for the application, as it creates a more seamless workflow between manual testers and automation experts that are testing the same application.

Enhancing Your Tests

Relevant for: GUI tests only

After creating an initial test, you can further enhance it by adding and modifying steps in the Keyword View or the Editor.

Checkpoints

Add checkpoints to your test to compare specified items during a run session with the values stored for the same items within the test.

Checkpoints enable you to identify whether or not your application is functioning correctly, and have several types. For details, see "[Checkpoints Overview](#)" on page 290

Tip: You can also use the CheckProperty method, which enables you to verify the property value of an object without using the checkpoint interface.

Parameterization

Parameterizing your test is when you replace fixed values with values from an external source during your run session.

Do this to test the same operations with different data.

You can supply data from a data table, environment variables you define, or values that UFT generates during the run session.

For more details, see "[Parameterizing Object Values](#)" on page 336.

Output Values

Retrieve values from your test and store them in the data table as output values to subsequently use these values as an input parameter in your test.

For more details, see "[Output Values Overview](#)" on page 325.

Programming Statements

Use special UFT options to enhance your test with programming statements.

- The Step Generator guides you step-by-step through the process of adding recordable and non-recordable operations (methods and properties) to your test.
- Synchronize your test to ensure that your application is ready for UFT to perform the next step in your test.
- Measure the amount of time it takes for your application to perform steps in a test by defining and measuring transactions.

For more details, see "[Generated Programming Operations](#)" on page 717.

You can also manually enter standard VBScript statements, as well as statements using UFT test objects and operations. For details, see "[Programming in GUI Testing Documents in the Editor](#)" on page 647.

Active Screen Updates

As the content of your application changes, update the selected Active Screen display.

Then, use the Active Screen to add new steps to your test instead of re-recording steps on new or modified objects.

For details, see "[How to Update Test Object Descriptions, Checkpoints, or Output Values, or Active Screen Captures](#)" on page 140.

Editor and Keyword View - A Comparison

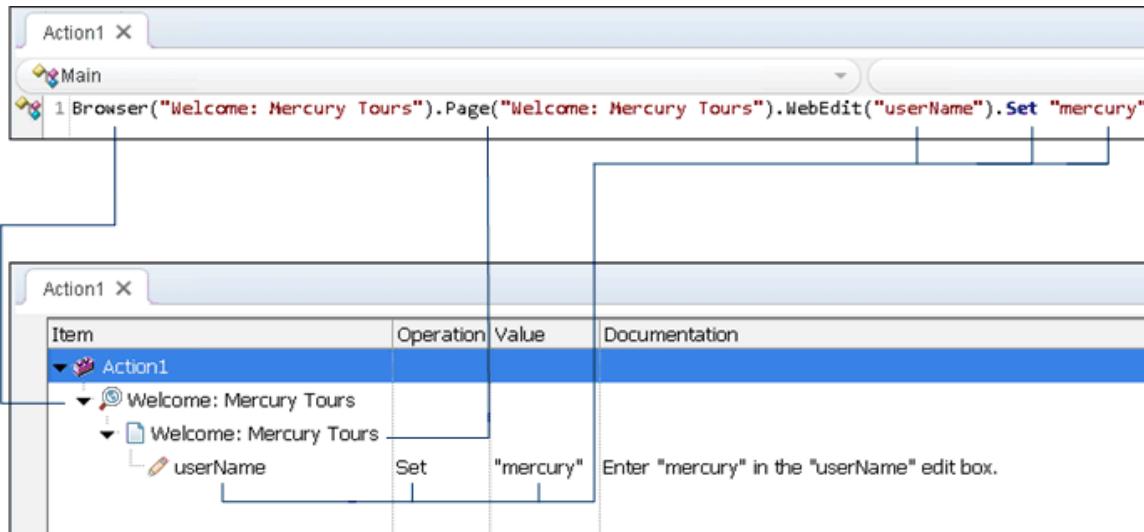
Relevant for: GUI actions and scripted GUI components

If you prefer to work with VBScript statements when editing actions or scripted components, you can choose to work in the Editor, as an alternative to using the Keyword View. Move between the two views as you wish, by selecting the Editor / Keyword View toggle button.

The Editor displays the same steps and objects as the Keyword View, but in a different format:

- In the Keyword View, UFT displays information about each step and shows the object hierarchy in an icon-based table. For details, see "[Keyword View](#)" on page 109.
- In the Editor, UFT displays each step as a VBScript line or statement. In object-based steps, the VBScript statement defines the object hierarchy.

The following diagram shows how the same object hierarchy is displayed in the Editor and in the Keyword View:



Each line of VBScript in the Editor represents a step. The example above represents a step in an action in which the user inserts the name mercury into an edit box. The hierarchy of the step enables you to

see the name of the site, the name of the page, the type and name of the object in the page, and the name of the operation performed on the object.

The table below explains how the different parts of the same step are represented in the Keyword View and the Editor:

Keyword View	Editor	Explanation
Login Welcome: Mercury Tours	Browser ("Welcome: Mercury Tours")	The name of the browser test object is Welcome: Mercury Tours.
Welcome: Mercury Tours	Page ("Welcome: Mercury Tours")	The name of the current page is Welcome: Mercury Tours.
Username	WebEdit ("userName")	The object type is WebEdit; the name of the edit box on which the operation is performed is userName.
	Set	The method performed on the edit box is Set.
"mercury"	"mercury"	The value inserted into the username edit box is mercury.

For more details, see:

- ["Checkpoint and Output Statements" on page 662](#)
- ["Parameter Indications in VBScript" on page 672](#)
- ["Basic VBScript Syntax" on page 667](#)
- ["Automatic Code Completion for GUI Testing" on page 632](#)

Sample Test

Relevant for: GUI tests only

The following is a sample action of a login procedure to the Mercury Tours site, the sample Web site. When you create tests, UFT creates both a graphical representation and script of the steps you perform on your application.

The graphical representation of these steps is displayed in the Keyword View.

Item	Operation	Value	Documentation
Welcome: Mercury Tours - Welcome: Mercury Tours			
userName	Set	"mercury"	Enter "mercury" in the "userName" edit box.
password	SetSecure	"49005a449159085e22d51484d..."	Enter the encrypted password in the "password" edit box.
Sign-In	Click		Click the "Sign-In" image.
+ NEW STEP			

The table below provides an explanation of each step in the Keyword View.

Step	Description
Welcome: Mercury Tours	The browser invokes the Welcome: Mercury Tours Web site.

Step	Description
	Welcome: Mercury Tours is the name of the Web page test object.
	userName is the name of the edit box test object. Set is the method performed on the edit box. tutorial is the value property of this edit box.
	password is the name of the edit box test object. SetSecure is an encryption method performed on the edit box. 4ff405198a68b24867227da98b21e547cf0c2f47d31 is the encrypted value of the password.
	Sign-In is the name of the image link test object. Click is the method performed on the image. 26, 4 are the x- and y-coordinates where the image was clicked.

The Editor displays these same steps using a VBScript program based on the UFT object model.

```
Browser("Welcome: Mercury Tours").Page("Welcome: Mercury Tours").WebEdit  
("userName").Set "tutorial"  
Browser("Welcome: Mercury Tours").Page("Welcome: Mercury Tours").WebEdit  
("password").SetSecure "4ff405198a68b24867227da98b21e547cf0c2f47d31"  
Browser("Welcome: Mercury Tours").Page("Welcome: Mercury Tours").Image("Sign-  
In").Click 26,4
```

Chapter 22: Create a Keyword-Driven GUI Test

Relevant for: GUI tests only

This task describes how to create a test using the keyword-driven methodology, and includes the following steps:

1. ["Analyze your application" below](#)
2. ["Prepare the testing infrastructure " on the next page](#)
3. ["Add steps to the actions in your test action repository" on page 95](#)
4. ["Enhance your test " on page 95](#)

After creating your test, run it to test your application and analyze the results, debug the test, or run in Maintenance Mode or Update Run Mode to maintain changes.

For details, see ["UFT Running and Debugging Operations" on page 830](#)["UFT Running and Debugging Operations" on page 830](#) and ["Maintain GUI Tests or Components " on page 133](#).

Analyze your application

Before you begin creating a test, analyze your application and determine your testing needs. You need to determine:

The development environments	Determine the development environments in which your application controls were developed, such as Web, Java, or .NET, so that you can load the required UFT add-ins.
The functionality to test	Determine the functionality that you want to test. To do this, consider the various activities that customers perform in your application to accomplish specific tasks. <ul style="list-style-type: none">• Which objects and operations are relevant for the set of business processes that need to be tested?• Which operations require customized keywords to provide additional functionality?
How to organize your test	Decide how to divide these processes into smaller units that will be represented by your test's actions. Each action should emulate an activity that a customer might perform when using your application. As you plan, try to keep the amount of steps you plan to include in each action to a minimum. Creating small, modular actions helps make your tests easier to read, follow, and maintain.

Prepare the testing infrastructure

Prepare the infrastructure needed for your test.

Build the set of resources for your test

This includes:

Shared object repositories	Contain test objects, which are representations of the objects in your application. For details, see " Test Objects in Object Repositories " on page 178.
Function libraries	Contain functions that enhance UFT functionality. For details, see " User-Defined Functions and Function Libraries " on page 683
Recovery scenarios	Instruct UFT to recover from unexpected events and errors that occur in your testing environment during a run session. For details, see " Recovery Scenarios for GUI Testing " on page 144.
Additional optional files	Includes data table files and environment variable files. For details, see " Parameterizing Object Values " on page 336.

Configure UFT according to your testing needs

This can include setting up global testing and test-specific preferences, as well as run session preferences.

Associate any recovery scenarios with your test, and create automation scripts to set required configurations at the start of a run session.

For details, see:

- "[Global Options](#)" on page 80
- "[Document Settings](#)" on page 81
- "[Recovery Scenarios for GUI Testing](#)" on page 144
- "[UFT Automation Scripts](#)" on page 740

Create one or more tests that serve as action repositories

This enables you to store the actions to be used in your test and maintain your actions in one central location.

For details, see "[Actions in GUI Testing](#)" on page 103.

Associate your resources with your test and the relevant actions

Associate your function libraries and recovery scenarios with the relevant tests so that you can insert steps using keywords.

Also, associate object repositories with relevant actions so you can insert steps using the stored test objects.

For details, see:

- ["Test Objects in Object Repositories" on page 178](#)
- ["User-Defined Functions and Function Libraries" on page 683](#)
- ["Recovery Scenarios for GUI Testing" on page 144](#)

Add steps to the actions in your test action repository

Create steps using keyword-driven functionality. You can use the table-like, graphical Keyword View—or you can use the Editor if you prefer to program steps directly in VBScript.

You can add steps to your test in one or both of the following ways:

Drag objects from your object repository or from the Toolbox pane	This adds keyword-driven steps in the Keyword View or the Editor. The object repository and Toolbox pane contain all of the objects that you want to test in your application.
Record on your application	As you navigate through your application during a recording session, each step you perform is graphically displayed as a row in the Keyword View. For details, see "Keyword View" on page 109.

Enhance your test

Enhance the testing process by modifying your test with special testing options and/or with programming statements.

For details, see:

- ["Checkpoints Overview" on page 290](#)
- ["Output Values Overview" on page 325](#)
- ["Parameterizing Object Values" on page 336](#)
- ["User-Defined Functions and Function Libraries" on page 683](#)
- ["Generated Programming Operations" on page 717](#)

Chapter 23: Record GUI Tests and Components

Relevant for: GUI tests and components

This chapter includes:

• Recording Overview	97
• Normal Recording	97
• Analog Recording	97
• How to Record a GUI Test or Component	99
• Known Issues When Recording	102

Recording Overview

Relevant for: GUI tests and scripted GUI components and business process tests

You can create the main body of a test or component by recording the typical processes that users perform on your application.

UFT records the operations you perform, adding them as steps to the selected test action or component.

Then, during a run session, UFT uses the recorded steps to replicate the operations you performed while recording.

While you record the steps, UFT creates test objects representing the objects in your application on which you perform operations, and stores them in the test or component's local object repository. This enables UFT to identify the objects in your application both while creating the test or component and during a run session.

UFT also enters the correct methods, and argument values for the objects in your application. During a recording session, you can also add checkpoint and output value steps, to check or retrieve values from your application.

UFT provides the following recording modes:

- "[Normal Recording](#)" below
- "[Analog Recording](#)" below
- "[Insight recording](#)" on page 99
- "[Low-level recording](#)" on the next page

Note: UFT supports an additional recording mode, Standard Windows recording, which is relevant when recording tests or components on SAP GUI for Windows applications.

For details, see the SAP GUI for Windows section of the *HP Unified Functional Testing Add-ins Guide*.

Normal Recording

Records the objects in your application and the operations performed on them.

This mode is the default and takes full advantage of the UFT test object model, recognizing the objects in your application regardless of their location on the screen.

Analog Recording

Records the exact mouse and keyboard operations that you perform, in relation to either the screen or the application window.

This mode is useful for recording operations that cannot be recorded at the level of an object, such as a digital signature produced by dragging the mouse.

The steps recorded are saved in a separate data file stored with the action.

A single **RunAnalog** statement is added to your action or component, which calls the recorded analog file.

Note:

- You cannot edit analog recording steps from within UFT.
- Analog recording requires more disk space than normal recording mode.

Low-level recording

Records on any object in your application, whether or not UFT recognizes the specific object or the specific operation.

Use low-level recording:

- For recording on environments or objects not supported by UFT, if the *appearance* of the objects might change, but their *location* will not. If the object's appearance will not change, you can use [Insight recording](#) for unsupported environments or objects.
- If the location of the object is important to your test or scripted component. This way, the step will pass only if the object is in the correct position.

This mode records all parent level objects as **Window** test objects and all other objects as **WinObject** test objects. They are displayed in the Active Screen as standard Windows objects.

The following methods are supported:

Window test objects WinObject test objects

- | | |
|--|---|
| <ul style="list-style-type: none">• Activate• Click• DblClick• Drag• Drop• Maximize• Minimize• Restore• Type | <ul style="list-style-type: none">• Click• DblClick• Drag• Drop• Type |
|--|---|

Note:

- Low-level recording mode is not fully supported for multibyte character input.
- Steps recorded using low-level recording mode may not run correctly on all objects.

- Low-level recording requires more disk space than normal recording mode.

Insight recording

Records on any object displayed on your screen, whether or not UFT recognizes the object's technology and is able to retrieve its properties or activate its methods.

UFT recognizes objects based on their appearance, and not their native properties. This can be useful to test controls from an environment that UFT does not support or even from a remote computer running a non-Windows operating system.

For more details, see "[Identifying Objects Using Insight](#)" on page 266.

Note: Insight recording requires more disk space than normal recording mode.

To control the amount of space used, adjust the number of snapshots saved and their size in the **Insight** pane of the Options Dialog Box.

See also:

- "[How to Record a GUI Test or Component](#)" below
- "[Known Issues When Recording](#)" on page 102

How to Record a GUI Test or Component

Relevant for: GUI tests only

This task describes how to create a test in UFT by recording the steps that you perform on your application.

If you are testing mobile applications, see the [Mobile Center Help](#).

This task includes the following steps:

- "[Start a recording session](#)" on the next page
- "[Record steps into the test](#)" on the next page
- "[Use the Record toolbar to manage your recording session](#)" on the next page
- "[Capture objects to an object repository](#)" on the next page
- "[Switch to other recording modes](#)" on page 101
- "[How to Record a GUI Test or Component](#)" above
- "[How to Record a GUI Test or Component](#)" above
- "[How to Record a GUI Test or Component](#)" above

1. Prerequisites

- Close all unnecessary applications to avoid recording unnecessary user actions.
- In the Record and Run Settings Dialog Box, decide how you want to open the application when you record and run your test.

For Web applications:

- If you have **Record and run tests on any open browser** selected in the Record and Run Settings dialog box, ensure that the browser window was opened after you opened UFT.
- Determine the web site's security zone to help manage security alert dialog boxes in the browser window.
- Select a predefined configuration level in the Web Event Recording Configuration dialog box (**Record > Web Event Recording Configuration**).

2. Start a recording session

In the toolbar, click the **Record** button  to start recording. In the BPT View, click the **Record a New Business Component** button.

UFT is minimized, and a standalone Record Toolbar is displayed.

3. Record steps into the test

Perform user actions in your application.

UFT records each step you perform and adds it to your test.

In addition, in the local object repository, UFT adds a test object for each object on which you performed a step.

Note: If you are recording on a Web object, you must perform an action with the object in order for UFT to record the step.

For example, if you want to select an item in a list that is already selected, you must first select another item, and then go back to select the original item.

4. Use the Record toolbar to manage your recording session

<Action Name> drop down list	Select an action to include the recorded steps.
	Insert Call to New Action Select the type of new action you want to call.
	Insert Checkpoint and Output Value Select the type of checkpoint or output value to insert.

5. Capture objects to an object repository

While recording, you can learn objects without having to perform actions on them.

In the Record Toolbar, click the **Capture** button , and select the area for which you want to learn objects.

Highlight the area in your application that you want to learn.

UFT adds the test objects to the local repository.

6. Switch to other recording modes

In the record toolbar, select a mode from the Recording Modes dropdown.

-  **Analog recording**
-  **Low-level recording**
-  **Insight recording**
- **Standard Windows recording** (relevant when recording on SAP GUI for Windows applications)

For details about each mode, see ["Recording Overview" on page 97](#).

When you want to return to normal recording mode, select the **Default**  recording mode.

Tip: Recording in Insight mode may be slower than in other modes. Follow the recording progress by checking the number of recorded steps in the Record toolbar's title bar.

After recording in Insight mode

- Delete extra Insight snapshots from the Object Repository (**Tools > Delete Insight Snapshots**).
- Delete any unnecessary steps or make other adjustments. For example:

Recording Type steps	UFT records the Type method on a Standard Windows test object, and not on the Insight test object. After recording, you can delete this step, and replace it with a Type step performed on the relevant Insight test object.
Clicking before typing	If you click or press TAB to focus on a control before typing, UFT records a step for the click or TAB press. However, by default, the InsightObject 's Type method clicks in the control before typing, and the preceding step is redundant. After recording, delete the redundant Click or Type step.

Known Issues When Recording

Relevant for: GUI tests and components

This section describes troubleshooting and limitations for recording tests and components.

Identification properties

UFT does not record the **visual relation** identifier property.

This property can be added only manually from the Object Properties dialog box or the Object Repository Manager or window.

Start menu / Quick Launch panel

- **Windows 7 / Windows Server 2008 R2:** You must restart your computer after installing in order to record on the Start menu or Quick Launch panel.
- UFT does not record launching Windows Help from the **Start** menu.
- To record **Start** menu items customized as menus, customize them as links instead, or record their activation in some other way.

Dragging the Record toolbar

By default, dragging the Record toolbar to the top of the screen in Windows 7 or Windows 8.x or higher is not allowed.

To allow this, in the Windows Control Panel, select **Ease of Access > Ease of Access Center Make the mouse easier to use**, and disable the **Prevent windows from being automatically arranged when moved to the edge of the screen** option.

Chapter 24: Actions in GUI Testing

Relevant for: GUI tests only

This chapter includes:

• Actions in GUI Testing	104
• Structure Your Test with Actions	105
• Display / Modify Action Data	106
• Create an action template	106
• Action and action call properties	107
• Exit an action using programming statements	107
• Known Issues with Actions	108
• Copies of tests	108

Actions in GUI Testing

Relevant for: GUI tests only

In a GUI test, each test is comprised of calls to **actions**. An action is a separate modular test script, including all of the steps in that action, and any objects in its local object repository and any associated shared object repositories.

Each new test contains a call to a single action. Add calls to other new or existing actions, or copies of actions, both in the same or a different test. Ideally, each action should contain no more than a few dozen test steps.

The actions used in the test, and the order in which they are run, are displayed in the canvas.

Tip: Create tests that call multiple actions to make your test more logical, modular, and efficient. For example, separate your actions by the main sections of a Web site, or specific activities that you perform in your application.

UFT provides the following types of actions:

Action type	Description
Reusable actions	<ul style="list-style-type: none">• Default type.• Can be called multiple times by the local test and other tests.• Must be updated from the original test.• Can be marked as non-reusable to change its type.
Non-reusable actions	<ul style="list-style-type: none">• Can be called only once, and in the local test.• Can be copied.• Can be marked as reusable to change its type.
External	<ul style="list-style-type: none">• Stored with another test.• Read-only in the calling test. <p>You can choose to use a local, editable copy of the Data pane information.</p>
Nested	<ul style="list-style-type: none">• Called from another action

See also:

["Troubleshooting - Naming Conventions" on page 1144](#)

Structure Your Test with Actions

Relevant for: GUI tests only

- **Insert a call** to a new action, an existing action, or a copy of an action from the **Design** menu or the Record toolbar, or by right-clicking in the Solution Explorer or the canvas.
- **Nest an action** within an existing action from the Keyword View. Highlight the step after which you want to insert the call, and add the call as you would any other action.
- **Change the run order of actions** from the canvas by right-clicking, dragging, or using the arrow keys. Dragging is supported for top-level actions only.
- **Remove calls to actions** from the following locations:

Solution Explorer	Remove all calls to a specific action. If the action is local, the action is also deleted.
Canvas / Keyword View	Remove specific calls to an action. If the action is local, removing all calls to the action also deletes the action.

Caution: Be careful when deleting a local reusable action. If the action is called by other tests, deleting the action may cause the other tests to fail.

You can also call actions dynamically during a run session using the LoadAndRunAction statement.

For details, see the **Utility Objects** section of the *HP UFT Object Model Reference for GUI Testing*.

Organizing Actions in Your Test

Iterative actions

If your action runs more than one iteration, the action must end at the same point in your application as it started, so that it can run another iteration without interruption.

For example, suppose you are testing a sample flight reservation site. If the action starts with a blank flight reservation form, it should conclude with a blank flight reservation form.

Changing actions

If you expect certain elements of your application to change regularly, it is a good idea to divide the steps related to changeable elements into a separate action so that it will be easy to change the required steps, if necessary, after the application is modified.

Associated Object Repositories

Right-click an action to open the associated object repository.

Multiple associated object repositories

You can associate as many object repositories as needed with an action, and the same object repository can be associated with different actions as needed. You can also set the default object repositories to be associated with all new actions in a test.

Order of object repositories

The order of the object repositories in the list determines the order in which UFT searches for a test object description.

If there are test objects in different object repositories with the same name, object class, and parent hierarchy, UFT uses the first one it finds based on the priority order defined in the Associated Repositories Tab of the Action Properties Dialog Box.

The local object repository is always listed first and cannot be moved down the priority list or deleted.

Relative paths to object repositories

You can enter an associated object repository as a relative path. During the run session, UFT searches for the file in the folders listed in the Folders pane of the Options dialog box (**Tools > Options > GUI Testing** tab > **Folders** node), in the order in which the folders are listed.

Associate an object repository dynamically

You can associate an object repository dynamically during a run session using the `RepositoriesCollection` statement. For details, see the **Utility Objects** section of the *HP UFT Object Model Reference for GUI Testing*.

See also:

["How to Debug Your Test, Component, Function Library, or User Code File" on page 893](#)

["Considerations for Using Action Parameters" on page 365](#)

Display / Modify Action Data

Relevant for: GUI tests only

Double-click an action to show only that action in the Keyword View or Editor.

Create an action template

1. Create a single text file containing the comments, function calls, and other statements that you want to include in all new actions.
The text file must be in the structure and format used in the Editor.
2. Save the text file in your UFT installation folder under **\dat\ActionTemplate.mst**.

All new actions you create contain the script lines from the action template.

Action and action call properties

Action and action call properties are displayed in the Properties pane and the Action Properties dialog box.

The following tabs are included:

Action Properties dialog box	<ul style="list-style-type: none">• General Tab (Action Properties Dialog Box)• Parameters Tab (Action Properties Dialog Box)• Associated Repositories Tab (Action Properties Dialog Box)• Used By Tab (Action Properties Dialog Box)• External Action Tab (Action Properties Dialog Box)
Action Properties pane	<ul style="list-style-type: none">• General Properties Tab (Properties Pane - Testing)• Parameters Tab (Properties Pane - Testing)• Used By Tab (Properties Pane - Testing)

Exit an action using programming statements

Use one of the following exit action statements:

- **ExitAction.** Exits the current action, regardless of its iteration attributes.
- **ExitActionIteration.** Exits the current iteration of the action.

For more details, see the *HP UFT Object Model Reference for GUI Testing*.

See also:

["Keyword View" on page 109](#)

["UFT Document Management" on page 45](#)

["Programming in GUI Testing Documents in the Editor" on page 647](#)

Known Issues with Actions

Relevant for: GUI tests only

Test names in actions

If a test contains a call to an action stored in another test, and that other test was renamed in ALM, the original test name still appears (in square brackets) in the canvas.

The obsolete name in the canvas does not affect UFT's ability to locate and run the action.

If it is important to display the correct test name, delete the action call from the test and reinsert it.

Nested actions

You cannot add a new action as a nested action to an external action.

Instead, open the external action and add the call to the nested action directly.

Copies of tests

If you make a copy of an existing test, you cannot insert call to parallel actions in both tests from the same test.

Instead of copying the test, use **Save As** to create a duplicate of the test.

Chapter 25: Keyword View

Relevant for: GUI tests and components

This chapter includes:

• Keyword View User Interface	110
• Item column	110
• Operation column	111
• Value column	111
• Assignment column (actions only)	111
• Comment column (actions only)	112
• Output column (components only)	112
• Shortcut keys in the Keyword View	112
• Standard Steps in the Keyword View	113
• Define or modify an item value	113
• Parameterize an Item value for an argument	114
• Encode a password	114
• Add a standard step after a conditional or loop block	115
• Comments in the Keyword View	115
• Comments in actions	115
• Comments in components	116
• Conditional and Loop Statements	116
• Known Issues - Keyword View	119

Keyword View User Interface

Relevant for: GUI actions and components

The Keyword View enables you to create and view the steps of your actions or components in a modular, table-like format.

Each step has a separate row in the table, and each column represents the different parts of your step.

Note: When working with a business component, the Keyword View that you see in UFT is the same as the **Automation** tab in ALM.

To access	<ol style="list-style-type: none">1. Make sure that an action tab or component tab is the active document.2. If the Editor is displayed, click the Editor / Keyword View toggle button.
Important Information	<ul style="list-style-type: none">• Right-click the table header and select the columns to view in the Keyword View Options dialog box.• Drag columns and steps to reorder them.• Right-click a step to view its properties, modify your test or component such as inserting a new step, or to run the test from the selected step.

Add the following types of steps, comments, and programming:

Standard steps	A test step, with an object in your application with a specific operation performed on the object. For details, see " Standard Steps in the Keyword View " on page 113.
Checkpoint steps	A step to test the state of an object in your application at a specific point in the application/test flow. For details, see " How to Insert a Checkpoint in a GUI Test or Component " on page 309.
Output value steps	A step to produce a value from an object which can be used later in the test. For details, see " Output Values Overview " on page 325 (when working with tests) and " Output Values for in the Keyword View " on page 333.
Comments	Lines in the script designed to add descriptive details about the test/component or the specific step. For details, see " Comments in the Keyword View " on page 115.
Steps using programming logic	You can use programming logic to perform a number of tasks, such as sending information to the run results or synchronizing your test with your application. For details, see " Generated Programming Operations " on page 717, " Conditional and Loop Statements " on page 116.

The Keyword View has the following columns:

Item column

The item on which you want to perform the step. In a component, you must select an option from the **Item** column before you can add additional content to a step.

An item can be any of the following:

Actions	<ul style="list-style-type: none"> • a test object from the object repository • a utility object • a function call • a VBScript statement, for example, a Dim statement. VBScript statements should be viewed and modified in the Editor. • a step generated by the Step Generator. For details, see "How to Insert Steps Using the Step Generator" on page 726.
Components	<ul style="list-style-type: none"> • a test object from the object repository • Comment. Free text cell that spans the entire row. • a user-defined function (Operation). For example, a function that opens an application at the start of a business component or checks the value of a specific property. Available only if user-defined functions were added to a function library that is associated with the component's application area. For details, see "Associated Function Libraries" on page 684 and "User-Defined Functions and Function Libraries" on page 683.

Operation column

The operation to be performed on the item. This column contains a list of all available operations (methods, functions, or properties (or sub-procedures for components)) that can be performed on the item selected in the **Item** column, for example, **Click** and **Select**.

Value column

The argument values for the selected operation, partitioned according to the number of arguments of the selected option in the **Operation** column.

In **components**, an argument value can be a constant or a parameter:

Local parameters	Specific to the business component and can only be accessed by that component. It is intended for use in a single step or between component steps, for example, as an output parameter for one step and an input parameter for a later step.
Component parameters	Can be accessed by any component in your ALM project.

Assignment column (actions only)

The assignment of a value to or from a variable.

Double-click the left part of the cell to select either **Store in** or **Get from**. Then click in the right part of the cell to specify the name of the variable.

Store in X	Value is equivalent to an X = <step> line in the Editor.
Get From X	Value is equivalent to a <step> = X line in the Editor.

For example, **Store in cCols** would store the return value of the current step in a variable called `cCols`, which you could then use later in the test.

Comment column (actions only)

A free text edit box for any information you want to add regarding the step. These are also displayed as inline comments in the Editor.

For details, see "[Comments in the Keyword View](#)" on page 115.

Output column (components only)

The parameter in which output values for the step are stored.

Local parameters	Specific to the business component and can only be accessed by that component. It is intended for use in a single step or between component steps, for example, as an output parameter for one step and an input parameter for a later step.
Component parameters	Can be accessed by any component in your ALM project.

For details, see "[Output Values for in the Keyword View](#)" on page 333.

Example

If you select an output parameter named `cCols`, the output value of the current step would be stored in the `cCols` parameter.

You can then use the value stored in the output parameter later in the component as an input parameter.

Shortcut keys in the Keyword View

SHIFT+INSERT	Add new step after a conditional or loop block. Actions and scripted components only.
INSERT	Add new step below selected step.
F7	Add new step below the selected step using the Step Generator. Actions and scripted components only.
Arrow keys	Change the selected item in a cell list.

TAB, SHIFT+TAB	Move focus within a single row
CTRL+F11	Open the Checkpoint Properties dialog box when relevant.
SHIFT+F4	Open a list for a cell, when relevant.
Type a letter or sequence of letters	Move to a value in a cell list that starts with the entered string

Standard Steps in the Keyword View

Relevant for: GUI actions and components

Add a new step using the **NEW STEP** button, right-clicking, or dragging and dropping from the Toolbox pane.

You can also click in the **Item** cell, and select an object for your step. To specify a function instead of an object, select **Operation** from the Item list.

Define or modify an item value

In the Item column, click in the **Value** cell to activate it, and enter a value for each argument.

- You can enter **constant** or **parameterized** values.
- Separate multiple argument values with commas.

In tests, after first defining a regular statement (such as **x=10**), it can be edited only in the Editor.

Parameterize an Item value for an argument

Click the  button in the required **Value** cell. The parameter list opens, with individual tabs for each type of parameter.

Double-click the parameter you want to use.

To add a new parameter, click **Add New Parameter** at the bottom of the parameter list. Then, define the type, name, and optionally the location in a data table for the new parameter.



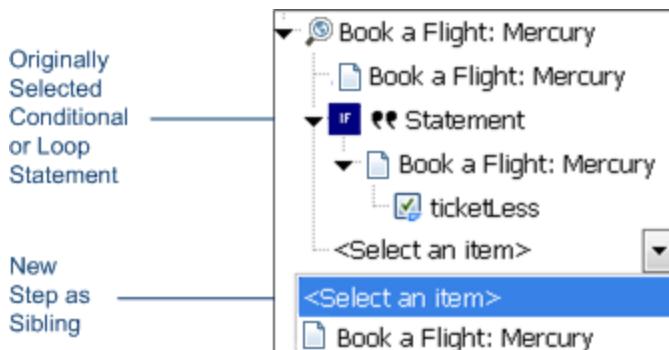
Encode a password

You can encode passwords for use in method arguments and data table cells. For details on how to encode passwords, see:

Actions	Use the Password Encoder tool (Start > All Programs > HP Software > HP Unified Functional Testing > Tools > Password Encoder).
Components stored in ALM	<p>Do not encode passwords in UFT.</p> <p>When business component tests run from ALM, all values are treated as strings.</p> <p>If you encode a password in UFT using the Password Encoder Tool, the resulting string, for example, 4b8a4a999d0d0e2c9b6cce8bb8e3, will be used instead of the password it represents.</p>
Components stored in UFT	<p>Use the Password Encoder tool (Start > All Programs > HP Software > HP Unified Functional Testing > Tools > Password Encoder).</p> <p>When your components are later upgraded to ALM, the upgrade process automatically converts existing encoded strings to a format recognized by ALM.</p>

Add a standard step after a conditional or loop block

1. Select the conditional or loop statement step after and outside of which you want to add the new step.
2. Right-click the conditional or loop statement and select **Insert New Step After Block**, or press SHIFT+INSERT. A new step is added to the Keyword View at the end of the conditional or loop block, outside of the conditional or loop statement (as a sibling).



3. Specify the content of the step by modifying it, as described in "[Standard Steps in the Keyword View](#)" on page 113.

Comments in the Keyword View

Relevant for: GUI actions and components

A **Comment** is a free text entry added to improve readability and make an action or component easier to update.

For example, add a comment to describe what is being tested, or to plan your steps before the application is ready to be tested.

Comments are indicated in the Keyword View by a icon.

You can also insert a comment step.

Comments in actions

In actions, comments are displayed in the **Comment** column for the relevant step. The **Comment** column is hidden by default.

To add or modify a comment, select the step, and enter your comment in the Comment column.

You can also add a comment as a separate step to your action.

Comments in components

In components, comments are displayed as a separate step, and are always displayed in the Keyword View.

To add a comment step, do one of the following:

- Select **Edit > Format > Comment**.
- Click in the **Item** cell and select **Comment** from the displayed list.
- Right-click a component step and select **Insert Comment**. A comment row is added below the selected step.

Note:

- UFT does not process comments during a run session.
- After you insert a comment, you cannot change it to a step.

Conditional and Loop Statements

Relevant for: GUI actions and scripted GUI components

Use conditional statements and loop statements to add decision making and iterations to your tests. Add them to your tests and components in the Keyword View as you would other steps.

Conditional statements

Conditional statements perform a step or a series of steps based on specific conditions. If a condition is not fulfilled, the next **Elseif** condition or **Else** statement is examined.

To add a conditional statement in the Keyword View:

1. Select the step before which you want to add the conditional statement.
2. Select **Edit > Code Snippet**, and select the statement you want to add.

The following conditional statements are available from the Keyword View:

- **IF** **If...Then**
- **ELSE IF** **Elseif...Then**
- **ELSE** **Else**

Each part of the conditional statement is added as a separate step.

For example, select **If** to add an **If** statement. After defining the details of the **If** statement, add a **Then** statement as a separate step.

3. In the row for the new conditional statement:

- Click in the **Item** cell, and select the object on which you want to perform the conditional statement.
 - Click in the **Operation** cell, and select the operation you want to perform.
 - If needed, click in the **Value** cell and enter the required condition.
4. Add the second part of your conditional statement, for example a **Then** statement.
 Right-click the new conditional statement and select **Insert New Step After Block**.
 Set the values for the new step in the **Operation** and **Value** columns.
 You can also record steps. After adding a conditional statement, all recorded steps are automatically inserted within the conditional statement block.
5. If the conditional statement replaces the statement just before it, delete the row immediately above the new statement.
6. Complete a statement with an **Else** statement, or by nesting an additional level in your statement.
 Select the new statement, and then select **Edit > Code Snippet**, and select the new statement you want to add.

Example

The statements below check that the User Name edit box exists in the Mercury Tours site.

If the edit box exists, **Then** a user name is entered; **Else** a message is sent to the Run Results.

<input checked="" type="checkbox"/> HP MyFlight Sample Application			
<input checked="" type="checkbox"/> agentName	Exist		Check whether the "agentName" edit box exists. If so: Enter john in the "agentName" edit box.
<input checked="" type="checkbox"/> agentName	Set	john	Otherwise:
<input checked="" type="checkbox"/> Statement			Report "The User Name field does not exist." to the run results and
<input checked="" type="checkbox"/> Reporter	ReportEvent	micFail,"UserName Check","Th...	

The same example is displayed in the Editor as follows:

```
If Browser("Welcome: Mercury").Page("Welcome: Mercury").WebEdit
("userName").Exist Then
  Browser("Welcome: Mercury").Page("Welcome: Mercury").WebEdit("userName").Set
  DataTable ("p_UserName", dtGlobalSheet)
Else
  Reporter.ReportEvent micFail, "UserName Check", "The User Name field does
  not exist."
End If
```

Loop statements

Use loop statements to run a group of steps repeatedly, while or until a condition is true, or a specific number of times without any conditions.

1. Select the step before which you want to add the loop statement.
2. Select **Edit > Code Snippet**, and select the statement you want to add.

The following loop statements are available from the Keyword View:

- ⌚ **While...Wend.** Performs a series of statements as long as a specified condition is True.
- ⌚ **For...Next.** Uses a counter to perform a group of statements a specified number of times.
- ⌚ **Do...While.** Performs a series of statements indefinitely, as long as a specified condition is True.
- ⌚ **Do...Until.** Performs a series of statements indefinitely, until a specified condition becomes True.

3. In the **Value** column, enter a required condition.
4. To complete the loop statement:
 - Select the loop statement step and record a new step to add it to your loop statement.
 - Select the loop statement step and right click and then select **Insert New Step**.

Example

The following example counts the number of items in a list and then selects them one by one.

After each of the items has been selected, the test continues.



The same example is displayed in the Editor as follows:

```
itemsCount = Browser("Welcome: Mercury").Page("Find a Flight:").
    WebList("toDay").GetROProperty ("items count")
For i = 1 To ItemsCount-1
    ItemName = Browser("Welcome: Mercury").Page("Find a Flight:").
        WebList("toDay").GetItem (i)
    Browser("Welcome: Mercury").Page("Find a Flight:").WebList("toDay").
        Select ItemName
```

Next

See also

["Add a standard step after a conditional or loop block" on page 115](#)

[VBScript documentation \(select Help > HP Unified Functional Testing Help > VBScript Reference\)](#)

Known Issues - Keyword View

Relevant for: GUI actions and components

Object properties	If you use the Object property in a step in the Keyword View, it may take a long time for UFT to retrieve the object information from the application. This may affect UFT's response time when you open and select from the various drop-down lists in the step. If this occurs, use the Editor when working with the Object property.
Calls to other actions	If you insert a call to another action, you cannot expand the action node to view the steps in the called action. Instead, double-click the called action node to open the action steps in another tab.

Chapter 26: Integration with API Tests

Relevant for: GUI tests only

This chapter includes:

- [Integrating API Testing in GUI Tests](#) 121
- [Licensing for calling API tests and actions](#) 121
- [Insert or modify a call to an API test](#) 122
- [Use API test parameters in a GUI test](#) 123
- [Using API Tests in a GUI Test - Use-Case Scenario](#) 123
- [Calling API Tests with Parameters - Use-case Scenario](#) 125

Integrating API Testing in GUI Tests

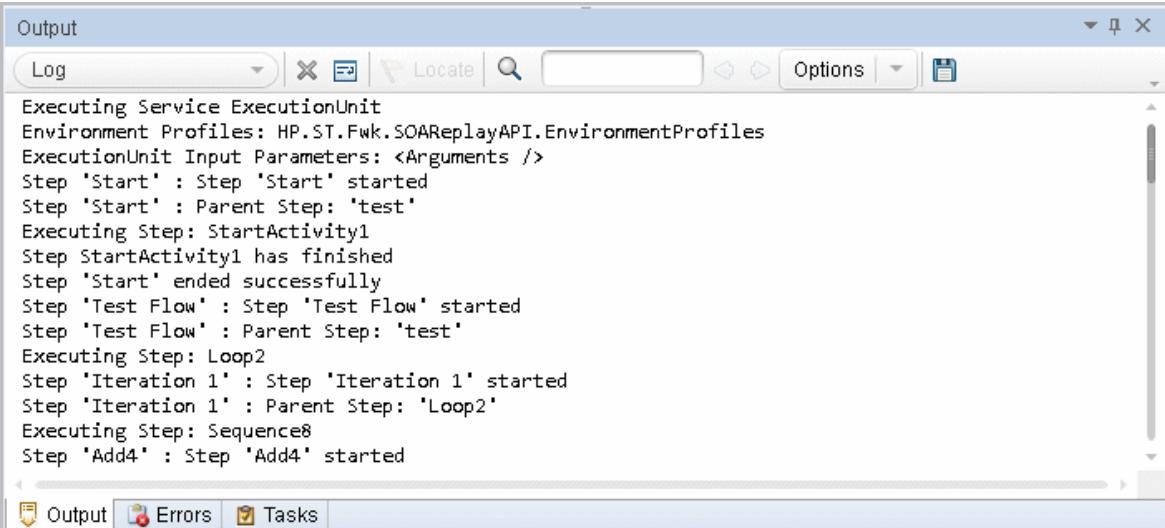
Relevant for: GUI tests only

If you have both GUI and API tests for your applications, you can run both tests together in a single unified test run.

Insert a call to an API test or action in any GUI test action. During the test run, UFT pauses the steps of the GUI test, runs the API test or action in its entirety, and then continues the GUI test run.

During the run session, the Output pane displays a real-time log of the steps that are being performed in the API test.

Example



```
Output
Log | X | Locate | Options | 
Executing Service ExecutionUnit
Environment Profiles: HP.ST.Fwk.SOAReplayAPI.EnvironmentProfiles
ExecutionUnit Input Parameters: <Arguments />
Step 'Start' : Step 'Start' started
Step 'Start' : Parent Step: 'test'
Executing Step: StartActivity1
Step StartActivity1 has finished
Step 'Start' ended successfully
Step 'Test Flow' : Step 'Test Flow' started
Step 'Test Flow' : Parent Step: 'test'
Executing Step: Loop2
Step 'Iteration 1' : Step 'Iteration 1' started
Step 'Iteration 1' : Parent Step: 'Loop2'
Executing Step: Sequence8
Step 'Add4' : Step 'Add4' started
Output Errors Tasks
```

Licensing for calling API tests and actions

To call API tests and actions, you must be using a Unified Functional Testing license.

If a Unified Functional Testing license type is not available, the run session behavior differs, depending on how you are running the test:

When running from UFT	The GUI test runs until the step calling the API test is reached, and then fails.
When running from ALM	If the GUI test contains a call to an API test, UFT does not open and the test does not run.

For a use case scenario describing this process, see "[Using API Tests in a GUI Test - Use-Case Scenario](#)" on page 123.

Insert or modify a call to an API test

Insert a new call	1. Click the Insert Call to New Action button. 2. Select Call to Existing API Test/Action .
Modify an existing call	Right-click the step select Edit Call to API Test/Action .

Note: Do not insert a call to an API test or action that contains a call to a GUI test, as this can cause unexpected behavior.

UFT inserts a step that calls the API test or action.

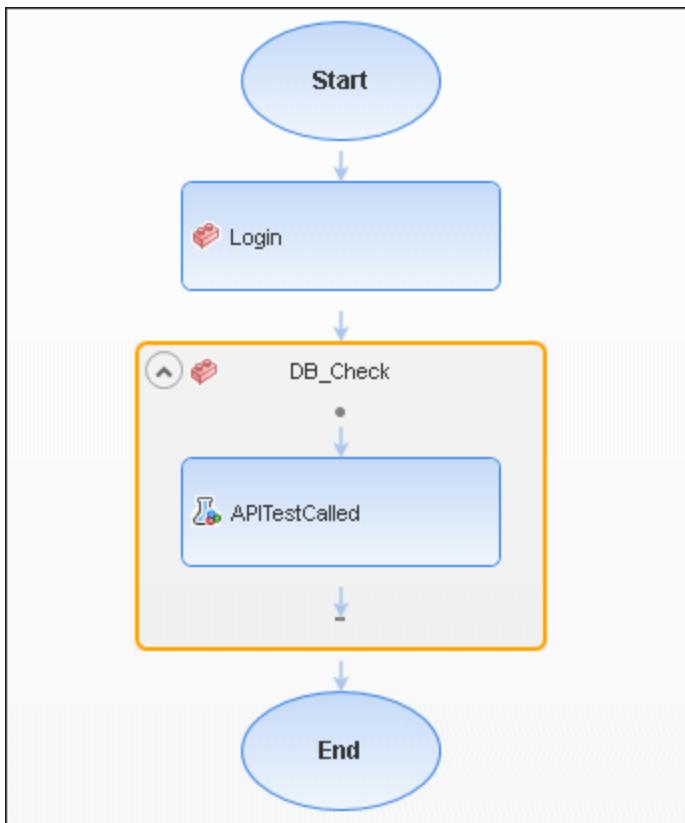
The call to the API test is displayed in the canvas as a call inside the relevant action of the GUI test.

Example

In the Editor:

```
RunAPITest "<API test name>"
```

In the canvas:



Use API test parameters in a GUI test

After you add a call to an API test or action, the input and output parameters are available for use in your GUI test.

Input parameters	If you want to use values for the API test input parameters during a GUI test run, specify the parameter value in the Call to Test/Action Dialog Box. When the API test is run during the GUI test, UFT uses the values you specified for the appropriate parameters in the API test.
Output parameters	If you want to use values for the API test output parameters, you must assign a variable name to the output parameter in the Call to Test/Action Dialog Box, or assign the API test output value to a variable or a data table parameter in the GUI test. This variable or data table parameter is then available to use in other steps of the GUI test.

Using API Tests in a GUI Test - Use-Case Scenario

Relevant for: GUI tests only

This use-case scenario describes an example of how to incorporate tests for the API (service) layer of your application into a GUI test.

For the purposes of this scenario, you will be using a flight booking application similar to the Mercury Tours Flight GUI and Flight API applications provided with the UFT installation and used in the GUI tutorial for Web applications.

Tip: For additional information and user interface details, see "[Integrating API Testing in GUI Tests](#)" on page 121

In your application, you have four different pages, which correspond to the different tasks involved in booking a flight:

Logging in to the booking site	Login page
Finding flight options based on customer selections	Flight Finder page
Selecting a flight from the list of flight options	Select Flight page
Booking and confirming a customer's flight selection	Book Flight page

In addition, your application has a number of API processes to help the application process flight booking requests:

Finding user login credentials in the database	Login operation
Searching for a list of all available flights and displaying the flight list	FindFlights operation
Creating a flight order	CreateFlight operation

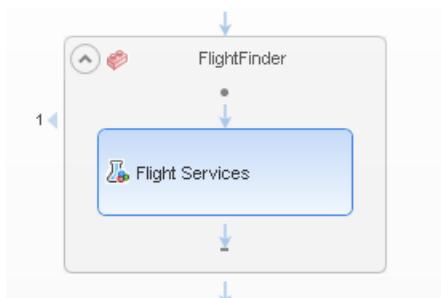
Confirming a flight booking	ConfirmBookFlight operation
Updating a flight order	UpdateFlightOrder operation
Deleting a flight order	DeleteFlightOrder operation
Deleting all flight orders	DeleteAllFlightOrders operation

You do the following:

1. Create a separate GUI action for each application page, giving it the same name as the page name.
2. Create a separate test for each API process, naming each test with the process name.
3. In order to fully test your application, you decide to place an API test after each GUI test. The API test checks to see whether the API processes run by that specific application's page (**Login**, **Flight Finder**, **Select Flight**, or **Book Flight**) are working correctly.
4. In your GUI actions, you insert a call to the corresponding API test.

The API test appears is displayed as nested inside the GUI action.

For example:



After you insert all the calls to the corresponding API tests, you have a call to an API test inside each of your GUI test actions as listed in the table below:

GUI Test Action Name	Calls API test
Login Page	Login
Flight Finder Page	FindFlights
Select Flight Page	CreateFlight
Book Flight Page	ConfirmBookFlight

Note: If you want to pass data from a API test to use in an GUI test, you must create a test output parameter in your API test.

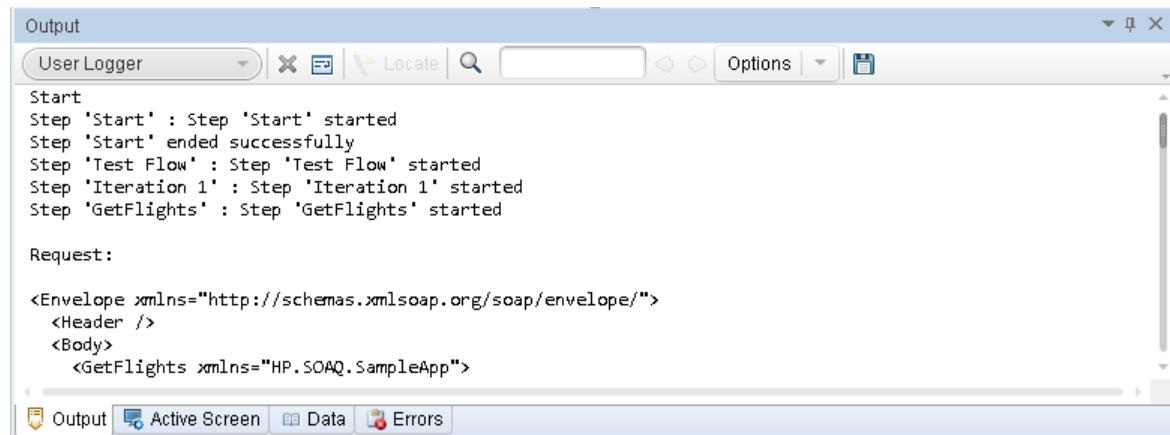
Running the test

After adding the necessary calls to your API tests, you can run the test.

The GUI test executes each step the user interface in the flight booking application user interface.

For each API test call in the GUI test, UFT compiles the API test and runs it. UFT displays the API test steps running in the Output pane.

For example:



The screenshot shows the UFT Output pane with the following content:

```
Output
User Logger | X | Locate | Options | 
Start
Step 'Start' : Step 'Start' started
Step 'Start' ended successfully
Step 'Test Flow' : Step 'Test Flow' started
Step 'Iteration 1' : Step 'Iteration 1' started
Step 'GetFlights' : Step 'GetFlights' started

Request:
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Header />
  <Body>
    <GetFlights xmlns="HP.SOAQ.SampleApp">
```

The Output pane has tabs at the bottom: Output (selected), Active Screen, Data, and Errors.

Run results

After the test run is complete, you can check the test results for the GUI test, including calls to each of the API tests. The run results show the completion and pass/fail status for each step.

Calling API Tests with Parameters - Use-case Scenario

Relevant for: GUI tests only

When you are testing your application, you may want to create both GUI and API tests for your application to ensure that the user interface and non-GUI (service) layers perform correctly.

In UFT, run the tests together in a single unified test run by calling the API test from a GUI test (or vice versa).

Sometimes, the GUI layer of your application requires data from the API layer to use in performing user tasks. When creating your test, if you insert a call to an API test from a GUI test, UFT enables you to pass the parameter from the API and then helps you select the right place to store this value until the GUI uses the parameter's value.

This use-case scenario demonstrates how a GUI test can call an API test and pass a parameter value from the API test back to the calling GUI test. For the purposes of this scenario, you will be using the flight booking application included with the UFT installation.

Tip: For additional information and user interface details, see "[Integrating API Testing in GUI Tests](#)" on page 121.

In the flight reservation application, you have five main user areas:

- A **login** area
- A **flight finder** window, where users enter their flight preferences
- A **flight selection** window, where users select the best flight for their flight preferences
- A **flight booking** window
- A **flight order** search

The application has an API that retrieves a flight order, including details about the airline, departure and arrival cities for the flight, departure and arrival times for the flight, flight number, and the price of the flight. You use this information in the user interface of the application to retrieve flight orders saved in the database.

You want to test multiple things:

- The API retrieves the flight information
- The GUI performs searches in the flight database correctly
- The GUI can take a value retrieved by the API and use it to search the flight database

To perform the tests of the application, you must run both the API and GUI tests, but also call the API test from the GUI test.

1. Since the flight number data (retrieved from the API) is contained in the response of the **Get** step, the GUI test has no ability to access individual step outputs.

To retrieve this value, you must create a test output parameter in the API test that enables you to pass the API test step output to the GUI test.

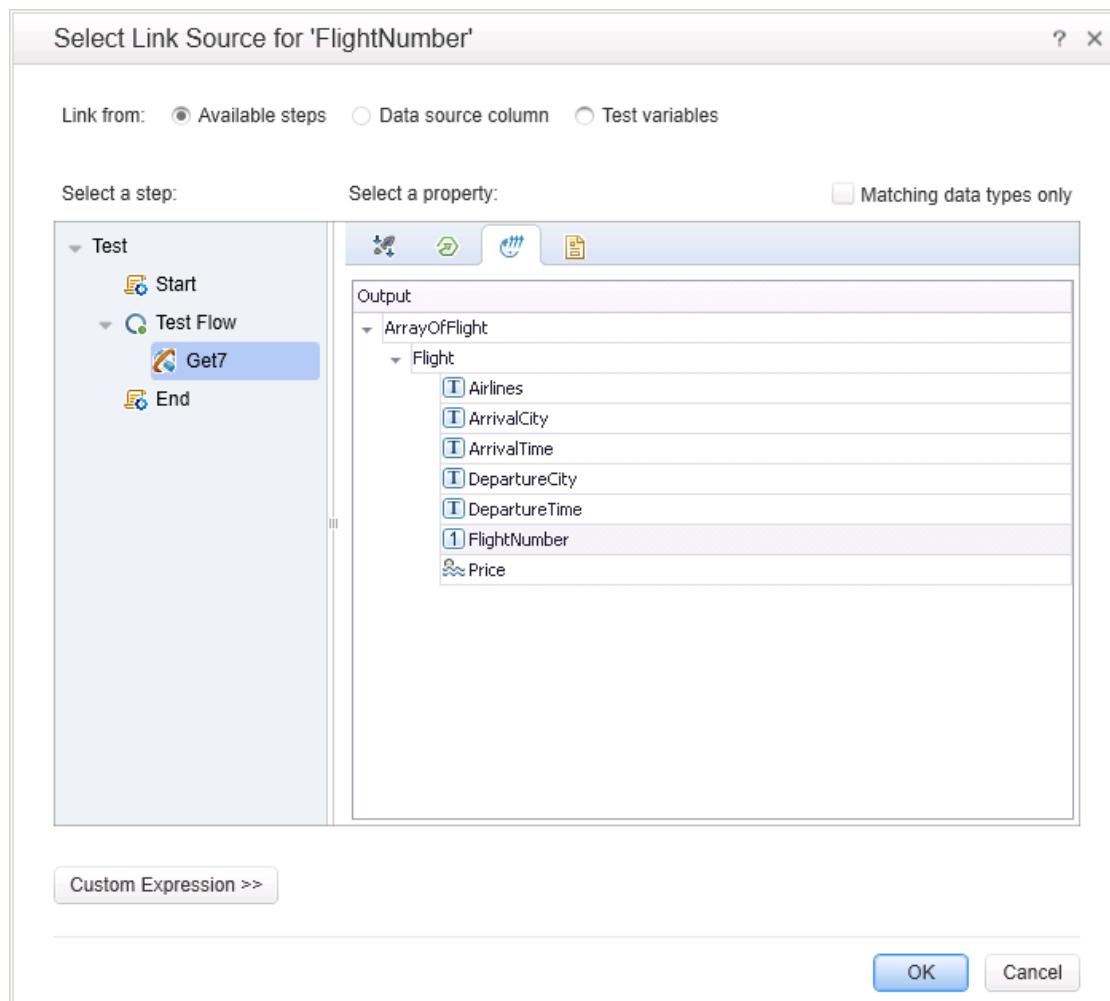
In the API test, click on the **End** step.

In the **Test Input/Output Parameters** tab of the Properties pane, create a new output parameter, called **FlightNumber**.

The test output parameter type must be **Float**.

2. After you create the parameter, you must link the test output parameter to the **Get** step output.

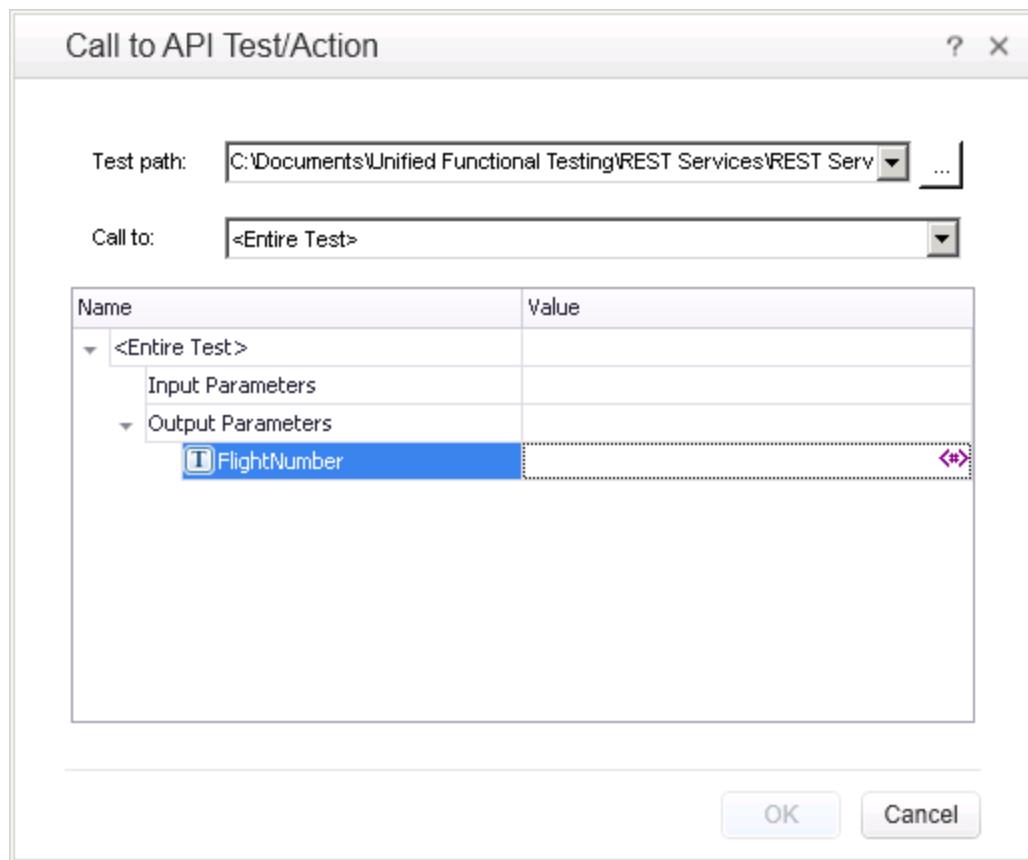
Click the **Link to data source** button  in the Test Input/Output Parameters tab, and then select the **Get** steps from the **Available steps** option.



3. Now that you have created an output parameter for the API test and linked it to the test step's output, you can call the API test from the GUI test.

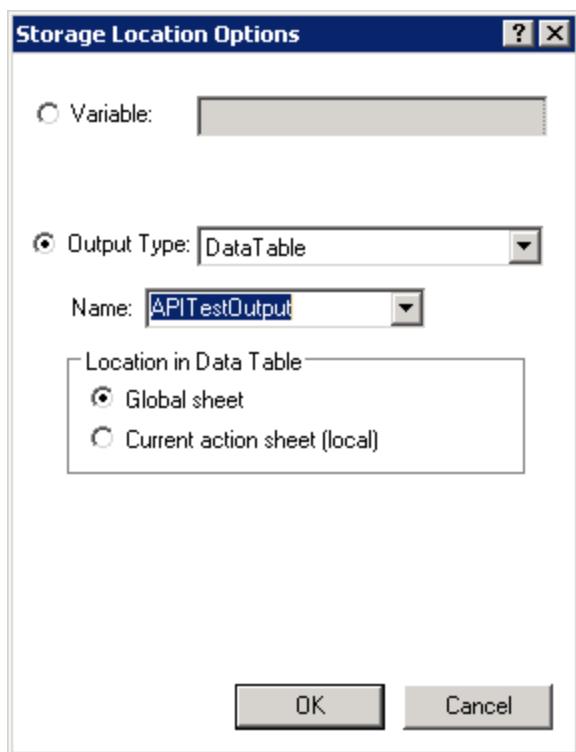
In the GUI test action, insert a call to an API test using the **Design > Call to Existing API Test/Action** command.

Then you can see the output parameter in the **Call to API Test/Action** dialog box after you navigate to the API test.



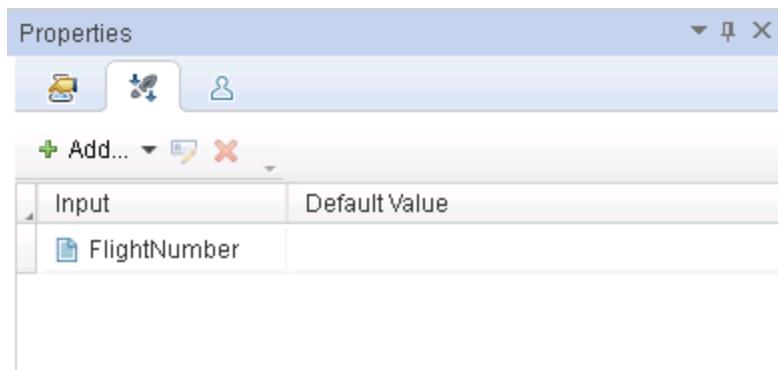
- From this dialog box, select where in the GUI test to store the output parameter data.
In the **Value** column for the **FlightNumber** parameter, click the **Configure** icon and open the Storage Location Options dialog box.

For this use-case scenario, you will save the output parameter as a GUI test data table parameter in the **Global** data table:



5. In order for the steps in the action to access this value, however, you must create an action parameter in the GUI test and link this action parameter to the data table.

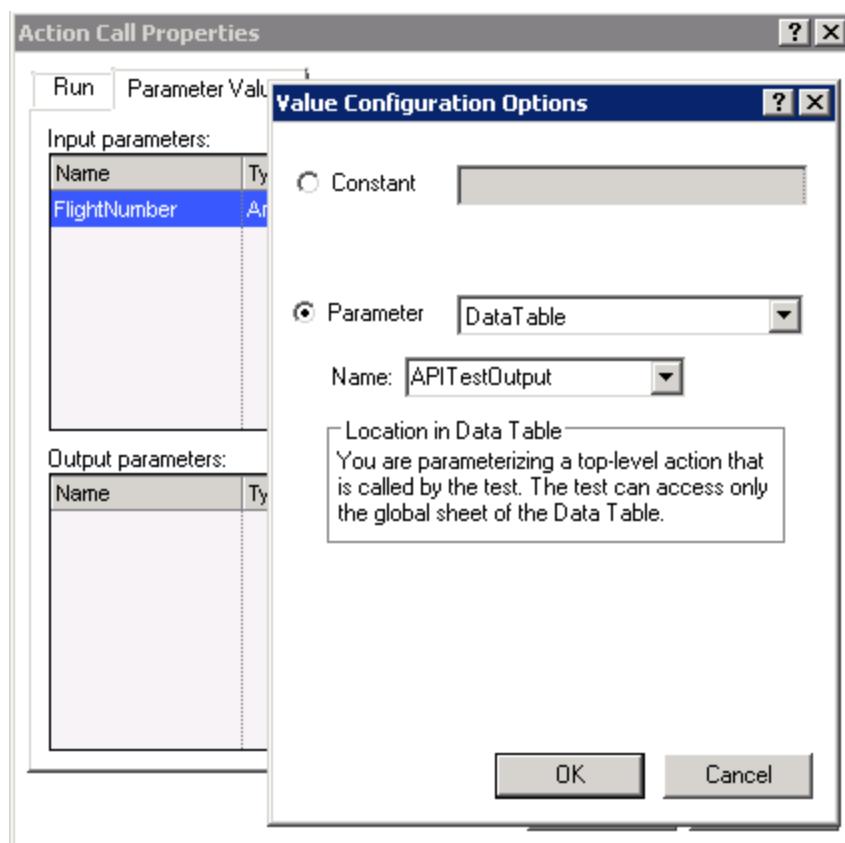
In the **Parameters** tab of the Properties pane, add an input parameter for the action called **FlightNumber** of type **Number**.



6. Link the value of the action parameter to the Data Table parameter.

In the canvas, right-click the action name and select **Action Call Properties**.

In the Action Call Properties dialog box, in the **Parameter Values** tab, click the **Configure** icon again and link to the Data table parameter:



7. Once the action parameter is linked to a GUI Data table parameter, you can parameterize the GUI test steps.

In the test steps, parameterize the **byNumberWatermark.Set** step with this data table parameter:

Item	Operation	Value	Documentation
▼ Search			
Function Call	RunAPITest	"REST Services",DataTable("APITestOutput", dtGl...	Run the "REST Services" API test/action u
▼ HP MyFlight Sample Application			
WpfTabStrip	Select	"SEARCH ORDER"	Select the "SEARCH ORDER" tab in the "W
byNumberRadio	Set		Select the "byNumberRadio" radio button.
byNumberWatermark	Set	"John Smith"	Enter "John Smith" in the "byNumberWat
SEARCH	Click		Click the "SEARCH" button.
ordersDataGrid	SelectCell		Select the cell in row "1", column "1" in th
SELECT ORDER	Click		Click the "SELECT ORDER" button.

Value Configuration Options

Constant: John Smith

Parameter: Test/action parameter

Parent action parameters

Parameter: FlightNumber

Output from previous action call(s)

Action:

Parameter:

OK Cancel

Your test steps now look like this:

Item	Operation	Value	Documentation
▼ Search			
Function Call	RunAPITest	"REST Services",DataTable("APITestOutput", dtGl...	Run the "REST Services" API test/action u
▼ HP MyFlight Sample Application			
WpfTabStrip	Select	"SEARCH ORDER"	Select the "SEARCH ORDER" tab in the "W
byNumberRadio	Set		Select the "byNumberRadio" radio button.
byNumberWatermark	Set	Parameter("FlightNumber")	Enter <the value of the 'FlightNumber' act
SEARCH	Click		Click the "SEARCH" button.
ordersDataGrid	SelectCell	"1","1"	Select the cell in row "1", column "1" in th
SELECT ORDER	Click		Click the "SELECT ORDER" button.

In the Editor, the steps are displayed like this:

```

RunAPITest "REST Services" ,DataTable("APITestOutput", dtGlobalSheet)
WpfWindow("HP MyFlight Sample Application").WpfTabStrip
("WpfTabStrip").Select "SEARCH ORDER"
WpfWindow("HP MyFlight Sample Application").WpfRadioButton
("byNumberRadio").Set
WpfWindow("HP MyFlight Sample Application").WpfEdit("byNumberWatermark").Set
Parameter("FlightNumber")
WpfWindow("HP MyFlight Sample Application").WpfButton("SEARCH").Click
WpfWindow("HP MyFlight Sample Application").WpfTable
("ordersDataGrid").SelectCell "1", "1"
WpfWindow("HP MyFlight Sample Application").WpfButton("SELECT ORDER").Click0

```

When the test runs, UFT stores the output parameter value from the API test in the Data table, and this parameter is then used in the GUI test step:

A1	1023	B
1	APITestOutput	B
2		
3		
4		
5		
6		
7		
8		
9		

Chapter 27: Maintain GUI Tests or Components

Relevant for: GUI tests and components

This chapter includes:

• Why Tests or Components Fail	134
• Application errors	134
• Application changes	134
• Missing objects	134
• Maintenance Run Mode	135
• Maintenance Run Mode prerequisites	136
• Determine UFT wait time	136
• Run the test or component in Maintenance Run Mode	136
• Merge changes to your shared object repository	136
• Maintenance Run Wizard Workflow	137
• Update Run Mode	138
• Smart Identification	138
• How to Update Test Object Descriptions, Checkpoints, or Output Values, or Active Screen Captures	140
• Known Issues in Maintenance and Update Run Modes	142
• Known issues in Update Run Mode	142

Why Tests or Components Fail

Relevant for: GUI tests and components

Tests or components fail when UFT encounters a step it cannot perform or the results of a step indicate failure.

Application errors

In many cases this is due to the application being tested not functioning properly, such as when checkpoints encounter conditions in the application being tested that are unexpected.

UFT then provides you with run results that assist you in understanding how to fix your application.

Application changes

Sometimes a test or component fails because the application being tested has changed and the test or component needs to be updated to reflect those changes.

For checkpoints, use Update Run Mode to update the checkpoints in your test or component to reflect changes in the application.

For example:

- Suppose your application has an edit box whose default value used to be <Enter value>.
- You have a checkpoint that checks this value before a new value is entered in the edit box.
- If the default value in the application changes to be <Enter name> then your checkpoint will fail.

Update Run Mode enables you to update the expected values of your checkpoint to reflect the change in the application.

For details, see "[Update Run Mode](#) " on page 138.

Missing objects

Your object repository may also be missing some of the objects it needs to run the test.

UFT provides tools that help identify and resolve some of these issues.

The object does not exist in the application	UFT cannot find an object in the application that matches the description of the object in the object repository. The Maintenance Run Wizard enables you to identify the object that you want your test or component to use.
The parent object	UFT cannot find an object in the application that matches and has the same hierarchy as the object in the object repository.

changed	The Maintenance Run Wizard enables you to identify the object that you want your test or component to use.
The object description property values changed	UFT cannot find an object in the application that is similar to, and has the same description property values as the object in the object repository. The Maintenance Run Wizard enables you to identify the object that you want your test or component to use.
The object does not exist in the object repository	UFT looks for the object to which the test or component refers, in the associated object repositories before attempting to identify that object in the application. If the object in your test or component cannot be found in any associated object repository, the Maintenance Run Wizard enables you to identify the object in your application that you want to add to your repository and use in your test or component.

For details, see "[Maintenance Run Mode](#)" below.

Maintenance Run Mode

Relevant for: GUI tests and components

Use Maintenance Run Mode to update the test objects in the object repositories associated with your test or component when UFT cannot locate one or more objects in your application during a run session.

When you run a test or component in Maintenance Run Mode, the Maintenance Run Wizard opens each time it encounters any of following problems and provides the described solutions:

Problem	Solution
Object cannot be identified in application	If you point to an object in the application being tested, the Maintenance Run Wizard compares that object to the objects in the associated object repositories. Depending on how the property values of the object to which you point compare to the property values of the objects in the associated repositories, the Maintenance Run Wizard suggests one of several options for updating your test or component to reflect the changes in the application. You can also choose to add a comment to your test or component before the failed step.
Object is missing from the object repository	The Maintenance Run Wizard helps you add the missing object to the repository. You can also choose to add a comment to your test or component before the failed step.
Object exists but can only be identified through Smart Identification	Identifying objects using Smart Identification may cause tests or components to run slower. The Maintenance Run Wizard helps you modify the description of the object, so that Smart Identification is not needed. For more details, see " Smart Identification " on page 237.

Note: Maintenance Run Mode does not support complex checkpoint or output value types such as File and XML checkpoints and output values.

During the Maintenance Run, these checkpoints and output values run as they would in a regular run session and will fail if there are differences between expected and actual values.

Tip: Alternately, update individual test object descriptions from the object in your application using the **Update from Application** option in the Object Repository window or Object Repository Manager.

For details, see "[Maintaining Identification Properties](#)" on page 192.

Maintenance Run Mode prerequisites

Install Microsoft Script Debugger	If it is not installed, you can use the UFT Additional Installation Requirements Utility to install it. Access the Additional Installation Requirements Utility from the Start menu or the <UFT installation folder>\bin\UFTInstallReqs.exe.
UFT set to Normal test run mode	Maintenance Run Mode can be run only when UFT is set to use the Normal run mode.
User interface	Maintenance Run Mode can only be run on applications that have a user interface

Determine UFT wait time

Determine how long UFT waits for an object to be displayed before determining that it cannot be found. The default setting is 20 seconds.

Change the object synchronization timeout in the Run pane of the Test Settings dialog box.

Tip: After Maintenance Run Mode finishes you may want to return this setting to its previous value for regular test runs.

Run the test or component in Maintenance Run Mode

1. Click the down arrow next to the **Run** button in the toolbar and select **Maintenance Run Mode**.
2. Specify the results location and the input parameter values (if applicable) for the Maintenance Run Mode session.
3. Follow the steps in the Maintenance Run Wizard .

For details, see "[Maintenance Run Wizard Workflow](#)" on the next page.

The run results open by default when the run session ends.

Merge changes to your shared object repository

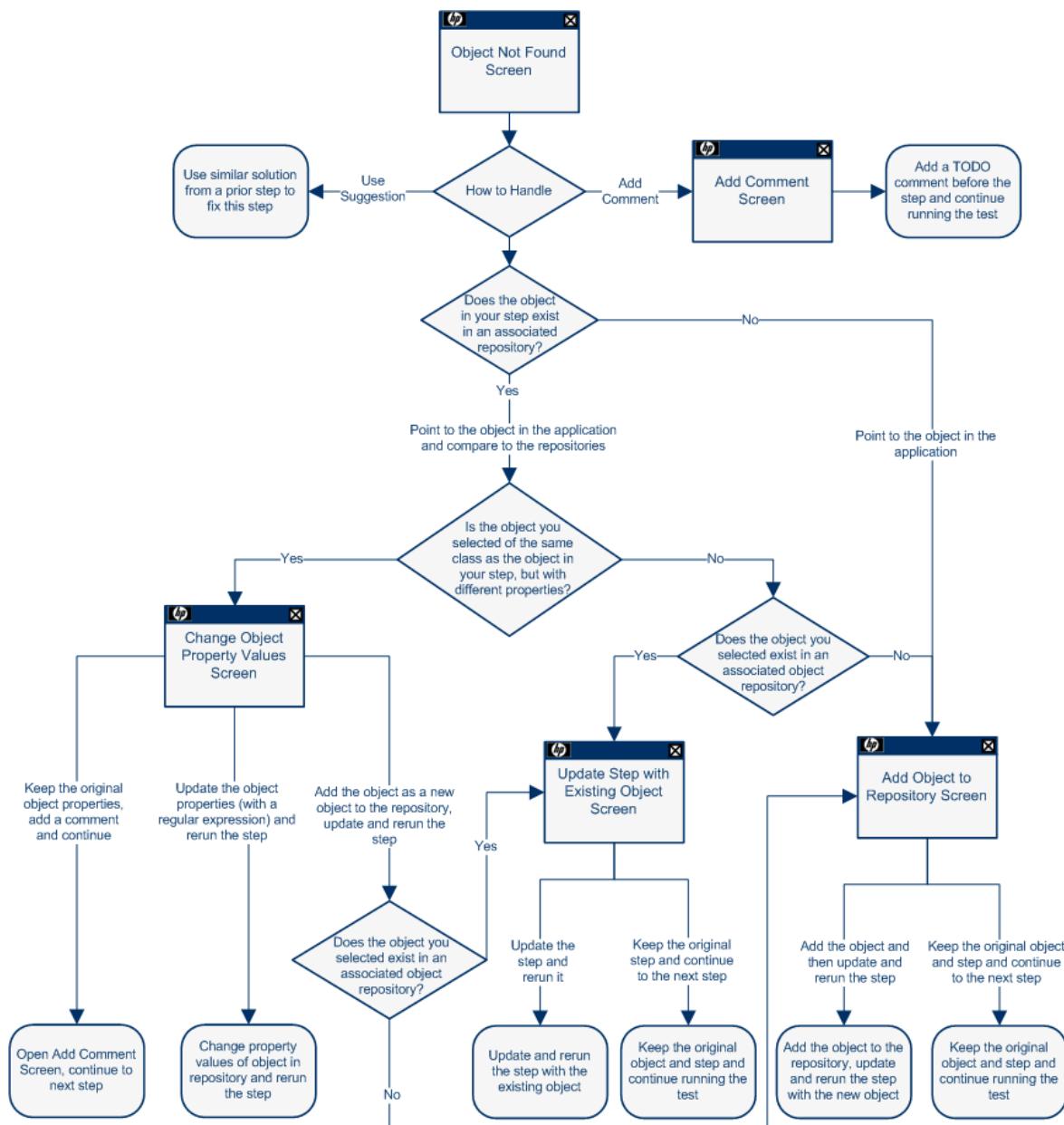
After using the Maintenance Run Wizard, you may want to merge the objects from the local repository back to a shared object repository.

Use the **Update from Local Repository** option in the Object Repository Manager.

For details, see "[How to Update a Shared Object Repository From a Local Object Repository](#)" on page [254](#).

Maintenance Run Wizard Workflow

Relevant for: GUI tests and components



Note: The Object Not Found Page does not open when UFT uses Smart Identification to identify an object in your test.

In that case, the Maintenance Run Wizard suggests updating the object properties according to the properties currently defined in the Object Identification Dialog Box.

Update Run Mode

Relevant for: GUI tests and components

Update Run Mode runs the test or component to update the following:

- The set of identification properties used for test object descriptions
- The Active Screen images and values
- The expected checkpoint values

UFT updates the set of identification properties for each object class in your associated object repositories according to the properties currently defined in the Object Identification Dialog Box.

Example

Suppose you design a test or component for the English version of part of your application.

You now want to use the same test or component for the French version of your application.

To do this:

1. Define a property that is not language-dependent, such as **target**, so that UFT can use these properties instead of text-based properties for object identification.
2. Perform an update run on the English version of this part of your application using these new properties.
3. Run the test or component on the French version of your application.

Smart Identification

If your objects are identified using Smart Identification, change the set of properties

Smart Identification

If you have a test or component that runs successfully, but in which certain objects are identified using Smart Identification, you can change the set of properties used for object identification.

Then, use the **Update test object descriptions** option to update the test object description to use the set of properties that Smart Identification used to identify the object.

When you run the test or component with **Update test object descriptions** selected, UFT finds the test object specified in each step based on its current test object description.

If UFT cannot find the test object based on its description, it uses the Smart Identification properties to identify the test object (if Smart Identification is enabled).

After UFT finds the test object, it then updates its description based on the mandatory and assistive properties that you define in the Object Identification Dialog Box.

Parameters or Regular Expressions

Any properties that were used in the previous test object description and are no longer part of the description for that test object class, as defined in the Object Identification Dialog Box, are removed from the new description.

This occurs even if the values were parameterized or defined as regular expressions.

If the same property appears both in the test object's new and previous descriptions, and the property value in the previous description was parameterized or specified as a regular expression, one of the following occurs:

If the previous value and the current value match ...	<p>... UFT keeps the property's previous parameterized or regular expression value.</p> <p>For example, if the previous property value was defined as the regular expression button, and the new value is button1, the property value remains button.</p>
If the previous value and the current value do not match ...	<p>... but the object is found using Smart Identification, UFT updates the property value to the new, constant property value.</p> <p>For example, if the previous property value was button, and the new value is My button, if Smart Identification definition enabled UFT to find the object, My button becomes the new property value.</p> <p>In this case, any parameterization or use of regular expressions is removed from the test object description.</p>

Property Case-Sensitivity

In some cases, the case-sensitivity of some identification properties may change from one version of UFT to the next, or as a result of a patch or hotfix installation.

In those cases, when you use Update Run to update test object descriptions, UFT also updates any case-sensitivity settings that may have changed.

See also:

["How to Update Test Object Descriptions, Checkpoints, or Output Values, or Active Screen Captures" on the next page](#)

How to Update Test Object Descriptions, Checkpoints, or Output Values, or Active Screen Captures

Relevant for: GUI tests and components

This task describes how to update your test or component data so that it is accurate for subsequent runs.

This task includes the following steps:

- ["Determine which data types you want to update" below](#)
- ["Run in Update Run Mode" below](#)
- ["Export and merge changes to your shared object repository \(Optional\)" below](#)
- ["Analyze the results of the Update Run Mode in the run results" on the next page](#)

1. **Determine which data types you want to update**

For details, see ["Update Run Mode" on page 138](#).

2. **Run in Update Run Mode**

- Specify the settings for the update run process.
- Enter any required input parameter values in the Input Parameters tab.

When UFT updates tests, it runs through only one iteration of the test and one iteration of each action in the test, according to the run option selected. For details on actions, see ["Actions in GUI Testing" on page 103](#).

When a test runs in Update Run Mode, it does not update parameterized values, such as Data pane data and environment variables. For details on parameterized values and environment variables, see ["Parameterizing Object Values" on page 336](#). Update Run Mode does not modify the property values of existing object descriptions in the object repository. To fix the object property values to match your application, use **Maintenance Run Mode**. For more details, see ["Maintenance Run Mode" on page 135](#).

3. **Export and merge changes to your shared object repository (Optional)**

When UFT updates tests or components, it always saves the updated objects in the local object repository, even if the objects being updated were originally from a shared object repository. The next time you run the tests or components, UFT uses the objects from the local object repository, as the local object repository has a higher priority than any shared object repositories.

After using Update Run Mode to update the test or component, you may want to use the Update from Local Repository option in the Object Repository Manager to merge the objects from the local repository back to a shared object repository. For details, see ["How to Update a Shared Object Repository From a Local Object Repository" on page 254](#).

4. **Analyze the results of the Update Run Mode in the run results**

The run results for an update run session are always saved in a temporary location.

Test objects that cannot be identified during the update process are not updated. As in any run session, if an object cannot be found during the update run, the run session fails, and information on the failure is included in the Run Results. In these situations, you may want to use **Maintenance Run Mode** to resolve these problems.

Because Update Run does not support updating complex checkpoint types such as File checkpoints and XML checkpoints, then these checkpoints run as they would in a regular run session and will fail if there are differences between expected and actual values.

By default, when the run session ends, the run results opens. If you cleared the **View results when run session ends** check box in the **Run Sessions** pane of the Options dialog box (**Tools > Options > General tab > Run Sessions** node), the run results does not open at the end of the run session.

When the update run ends, the run results can show:

- Updated values for checkpoints.
- Updated test object descriptions.

Known Issues in Maintenance and Update Run Modes

Relevant for: GUI tests and components

This section describes troubleshooting and limitations for Maintenance Mode.

- When running in Maintenance Mode, if a step contains a programmatic description for an object that is not found in an application, it may take a while for Maintenance Mode to indicate there is a problem. If you use the option to point to the object, it may take a while for Maintenance Mode to reopen afterward.
- When the Maintenance Run Wizard cannot find an object, and the test object description for that object does not have any mandatory or assistive properties (it is identified only by its ordinal identifier, such as a Browser test object), then when you point to the object, the wizard is unable to fix the problem and displays a message that the object you pointed to has a test object description that is similar to the object that UFT could not identify.

Workaround: Use the **Update from Application** option in the Object Repository window (for objects in the local respiratory) or in the Object Repository Manager (for objects in a shared object repository) to fix the test object description.

- When your Windows Display settings are set to use large fonts, the text in the Maintenance Run Wizard screens may be truncated.
- When the Maintenance Run Wizard cannot find an object in the application, and you point to a different object class to replace it, the Maintenance Run Wizard offers to add a step with that object and the object's default method. However, the wizard does not insert any method arguments for the step. If the step method has required arguments and you accept the step that the Maintenance Run Wizard proposes without modifying it, the step fails when you run it.

Workaround: Enter valid method arguments for the step.

- When running in Maintenance Mode, UFT may replace test objects with XPath or CSS identifier property values with new objects from your application.

Workaround: Use the **Update from Application** option in the Object Repository Manager to update specific test objects with XPath or CSS identifier property values.

- Maintenance Mode cannot fix problems in an object's properties, if the object is used in function library that is called by a step. If Maintenance Mode detects a problem in an object in a function library, a message is displayed indicating that the test is read-only and that Maintenance Mode is disabled.

Known issues in Update Run Mode

Insight	The Update Run mode does not update Insight test objects. To update an Insight test object, open the object in the object repository, and click the Change Test Object Image  button in the Test object image area.
----------------	---

Checkpoints and output values	Update Run does not support updating complex checkpoint and output value types such as File and XML. During the Update Run, these checkpoints and output values run as they would in a regular run session and will fail if there are differences between expected and actual values.
--------------------------------------	---

Chapter 28: Recovery Scenarios for GUI Testing

Relevant for: GUI tests and components

This chapter includes:

• Recovery Scenarios Overview	145
• When to Use Recovery Scenarios	146
• Programmatically Controlling the Recovery Mechanism	147
• How to Create and Manage Recovery Scenarios	147
• How to Manage Recovery Scenario Associations	148
• Troubleshooting and Limitations - Recovery Scenarios	151

Recovery Scenarios Overview

Relevant for: GUI tests and components

Unexpected events, errors, and application crashes during a run session can disrupt your run session and distort results. This is a problem particularly when tests or components run unattended—the run pauses until you perform the operation needed to recover. To handle situations such as these, UFT enables you to create recovery scenarios and associate them with specific tests or application areas. Recovery scenarios activate specific recovery operations when trigger events occur.

The Recovery Scenario Manager provides a wizard that guides you through the process of defining a recovery scenario, which includes a definition of an unexpected event and the operations necessary to recover the run session. For example, you can instruct UFT to detect a **Printer out of paper** message and recover the run session by clicking the **OK** button to close the message and continue running.

A recovery scenario consists of the following:

- **Trigger Event.** The event that interrupts your run session. For example, a window that pops up on the screen, or a run error.
- **Recovery Operations.** The operations to perform to enable the run session to continue after the trigger event interrupts the session. For example, clicking an **OK** button in a pop-up window, or restarting Microsoft Windows.
- **Post-Recovery Test Run Option.** The instructions on how UFT should proceed after the recovery operations are performed, and from which step to continue, if at all. You may want to restart the run from the beginning, or skip a step entirely and continue with the next step.

After you create recovery scenarios, you associate them with selected tests or components (via the application area) so that the appropriate scenarios can run if a trigger event occurs. You can prioritize the scenarios and set the order in which to apply the scenarios during the run session. You can also choose to disable specific scenarios, or all scenarios, that are associated with a test or application area.

You can also define which recovery scenarios will be used as the default scenario for all new tests.

For tests: You can associate, remove, enable, disable, prioritize, and view the properties of the recovery scenarios associated with a GUI test in the Solution Explorer.

For components: You define recovery scenarios for components in the application area. For details, see ["Application Areas" on page 1014](#).

To learn more, see:

- [When to Use Recovery Scenarios](#) 146
- [Programmatically Controlling the Recovery Mechanism](#) 147

When to Use Recovery Scenarios

Relevant for: GUI tests and components

Recovery scenarios are intended for use **only** with events that you cannot predict in advance, or for events that you cannot otherwise synchronize with a specific step in your test or component.

By default, recovery scenario operations are activated only after a step returns an error. This can potentially occur several steps after the step that originally caused the error. The alternative, checking for trigger events after every step, may slow performance. For this reason, it is best to handle predictable errors directly in your test or component.

If you can predict that a certain event may happen at a specific point in your test or component, it is highly recommended to handle that event directly within your test or component, rather than depending on a recovery scenario. To do this in a test, add steps such as If statements or optional steps. To do this in a component, use a user-defined function with conditional steps.

Handling an event directly within your test or component enables you to handle errors more specifically than recovery scenarios, which by nature are designed to handle a more generic set of unpredictable events. It also enables you to control the timing of the corrective operation with minimal resource usage and maximum performance.

Examples

- If you know that an Overwrite File message box may open when a **Save** button is clicked during a run session:

You can handle this event with an If statement that clicks **OK** if the message box opens or by adding an optional step in the test to click **OK** in the message box. (For keyword components, you define this If statement in a user-defined function and make it available via the associated application area.)

- You can define a recovery scenario to handle printer errors. Then if a printer error occurs during a run session, the recovery scenario could instruct UFT to click the default button in the Printer Error message box.

You would use a recovery scenario in this example because you cannot handle this type of error directly in your test or component. This is because you cannot know at what point the network will return the printer error. Even if you try to handle this event by adding an If statement in a user-defined function or in your test immediately after a step that sends a file to the printer, your test or component may progress several steps before the network returns the actual printer error.

Programmatically Controlling the Recovery Mechanism

Relevant for: GUI tests and components

You can use the Recovery object to control the recovery mechanism programmatically during the run session. For example, you can enable or disable the entire recovery mechanism or specific recovery scenarios for certain parts of a run session, retrieve status information about specific recovery scenarios, and explicitly activate the recovery mechanism at a certain point in the run session.

By default, UFT checks for recovery triggers when an error is returned during the run session. You can use the Recovery object's **Activate** method to force UFT to check for triggers after a specific step in the run session. For example, suppose you know that an object property checkpoint will fail if certain processes are open when the checkpoint is performed. You want to be sure that the pass or fail of the checkpoint is not affected by these open processes, which may indicate a different problem with your application.

However, a failed checkpoint does not result in a run error. So by default, the recovery mechanism would not be activated by the object state. You can define a recovery scenario that looks for and closes specified open processes when an object's properties have a certain state. This state shows the object's property values as they would be if the problematic processes were open. You can instruct UFT to activate the recovery mechanism if the checkpoint fails so that UFT can check for and close any problematic open processes and then perform the checkpoint again. This ensures that when the checkpoint is performed the second time it is not affected by the open processes.

For details on the Recovery object and its methods, see the **Recovery** object topic in the **Utility Objects** section of the *HP UFT Object Model Reference for GUI Testing*.

How to Create and Manage Recovery Scenarios

Relevant for: GUI tests and components

This task describes how to perform different recovery scenario creation and management operations in the Recovery Scenario Manager Dialog Box.

This task includes the following:

- "[Define a recovery scenario file in which to store the recovery scenarios](#)" below
- "[Create a new recovery scenario operation using the Recovery Scenario Wizard](#)" on the next page
- "[Manage existing recovery scenarios](#)" on the next page

[Define a recovery scenario file in which to store the recovery scenarios](#)

By default, the Recovery Scenario Manager dialog box opens with a new recovery file. You can use this new file, or you can open an existing recovery file, in one of the following ways:

- Click the arrow next to the **Open** button to select a recently-used recovery file from the list.
- Click the **Open** button and select an existing recovery file using the Open dialog box.

Create a new recovery scenario operation using the Recovery Scenario Wizard

1. In the Recovery Scenario Manager Dialog Box, click the **New Scenario** button  . The Recovery Scenario Wizard opens.
2. Follow the on-screen instructions.

Manage existing recovery scenarios

- View properties for your recovery scenarios in the Recovery Scenario Properties Dialog Box.
- Edit, save, and remove existing recovery scenarios in the Recovery Scenario Manager by using the toolbar buttons.

How to Manage Recovery Scenario Associations

Relevant for: GUI tests and components

This task describes how to perform different recovery scenario management and association operations using the Recovery pane in the Test Settings dialog box or an application area's Additional Settings pane.

This task includes the following:

- "Associate a recovery scenario with your test in the Test Settings dialog box" below
- "Associate a recovery scenario with your test in the Solution Explorer" on the next page
- "Associate a recovery scenario with your component or application area" on the next page
- "View read-only recovery scenario properties" on the next page
- "Prioritize recovery scenarios" on page 150
- "Remove recovery scenarios from your test or application area" on page 150
- "Enable/disable specific recovery scenarios" on page 150
- "Set default recovery scenario settings for all new tests" on page 150

Associate a recovery scenario with your test in the Test Settings dialog box

1. In the Recovery Pane of the Settings dialog box, click the **Add** button .
2. In the Add Recovery Scenario dialog box, click the **Browse** button and navigate to the recovery scenario file. A list of recovery scenario operations contained in this file opens.
3. In the list of recovery scenarios, select a recovery scenario and click **Add Scenario**.

The recovery scenario is displayed in the Recovery pane of the Settings dialog box and is added under the Recovery Scenarios node found under your GUI test in the Solution Explorer.

Associate a recovery scenario with your test in the Solution Explorer

1. In the Solution Explorer, do one of the following:
 - a. Right-click a GUI test node and select **Add > Associate Recovery Scenario**.
 - b. Right-click the Recovery Scenarios Node and select **Associate Recovery Scenario**.
2. In the Add Recovery Scenario dialog box, click the **Browse** button and navigate to the recovery scenario file. A list of recovery scenario operations contained in this file opens in the Add Recovery Scenario dialog box.
3. In the list of recovery scenarios, select a recovery scenario and click **Add Scenario**.
The recovery scenario is added under the Recovery Scenarios node, found under your GUI test in the Solution Explorer.

Associate a recovery scenario with your component or application area

1. In Recovery pane of the Additional Settings pane of your application area, click the **Add** button 
2. In the Add Recovery Scenario dialog box, click the **Browse** button and navigate to the recovery scenario file. A list of all recovery scenario operations contained in this file opens in the Add Recovery Scenario dialog box.
3. In the list of recovery scenarios, select a recovery scenario and click **Add Scenario**.
The recovery scenario is displayed in the Recovery pane in the Additional Settings pane of your application area and is associated with the component through its application area or is associated with the application area.

View read-only recovery scenario properties

In the Recovery Pane (Test/Component Settings / Application Area Additional Settings)

1. In the **Scenarios** box, select the recovery scenario whose properties you want to view.
2. Do one of the following:

- Click the **Properties** button .

- Double-click a scenario in the **Scenarios** box.

The Recovery Scenario Properties dialog box opens, displaying read-only information regarding the settings for the selected scenario.

In the Solution Explorer:

Right-click the name of the recovery scenario whose properties you want to view and select **Recovery Scenario Properties**.

The Recovery Scenario Properties dialog box opens, displaying read-only information regarding the settings for the selected scenario.

Prioritize recovery scenarios

In the Recovery Pane (Test Settings / Application Area Additional Settings)

1. In the **Scenarios** box, select the scenario whose priority you want to change.
2. Click the **Up** or **Down** button . The selected scenario's priority changes according to your selection.

In the Solution Explorer:

Right-click the scenario whose priority you want to change and select **Move Up** or **Move Down**. The selected scenario's priority changes according to your selection.

Remove recovery scenarios from your test or application area

In the Recovery Pane (Test Settings / Application Area Additional Settings)

1. In the **Scenarios** box, select the scenario you want to remove.
2. Click the **Remove** button . The selected scenario is no longer associated with the test or application area.

In the Solution Explorer:

Right-click the scenario you want to remove and select **Remove Recovery Scenario from List**. The selected scenario is no longer associated with your test or application area and no longer appears under the test or application area node in the Solution Explorer.

Enable/disable specific recovery scenarios

Do one of the following:

- In the **Scenarios** box of the Recovery Pane of the Settings dialog box, perform one of the following:
 - Select the check box to the left of one or more individual scenarios to enable them.
 - Clear the check box to the left of one or more individual scenarios to disable them.
- In the Solution Explorer, right-click the scenario you want to disable and select **Disable Recovery Scenario**.

Set default recovery scenario settings for all new tests

Click the **Set as Default** button in the Recovery pane of the Test Settings dialog box to set the current list of recovery scenarios to be the default scenarios for all new tests. Any future changes you make to the current recovery scenario list only affect the current test, and do not change the default list that you defined.

Troubleshooting and Limitations - Recovery Scenarios

Relevant for: GUI tests and components

This section describes troubleshooting and limitations for recovery scenarios.

- If you specify multiple function libraries from different locations with the same name in the same recovery scenario, only the first function library is used.

Workaround: Rename the function libraries so that each function library has a unique name.

Chapter 29: Using Performance Testing and Business Service Management Products with UFT GUI Tests

Relevant for: GUI tests only

This chapter includes:

- [HP Performance Testing and Business Service Management Products Overview](#) 153
- [Designing Tests for HP Performance Testing Products](#) 155
- [Designing GUI Tests for HP Business Process Monitor](#) 155
- [Running GUI Tests from HP Performance Testing Products](#) 156
- [How to Insert and Run GUI Tests in Performance Center and LoadRunner](#) 157
- [Running GUI Tests from HP Business Process Monitor](#) 157
- [Measuring Transactions](#) 158
- [Silent Test Runner](#) 160

HP Performance Testing and Business Service Management Products Overview

Relevant for: GUI tests only

UFT enables you to create complex tests that examine the full spectrum of your application's functionality to confirm that every element of your application works as expected in all situations.

After you use UFT to create and run a suite of tests that test the functional capabilities of your application, you may want to test how much load your application can handle or to monitor your application as it runs.

- **HP performance testing products (LoadRunner and Performance Center)** test the performance and reliability of an entire system under controlled and peak load conditions. To generate load, these performance testing products run hundreds or thousands of virtual users. These virtual users provide consistent, repeatable, and measurable load to exercise your application just as real users would.
- **HP Business Service Management (formerly HP Business Availability Center)** enables real-time monitoring of the end user experience. Business Process Monitor runs synthetic users to perform typical activities on the monitored application.

If you have already created and perfected a test in UFT that is a good representation of your user's actions, you may be able to use your test as the basis for performance testing and application management activities.

You can use **Silent Test Runner** to check in advance that a test will run correctly from LoadRunner, Performance Center, and Business Process Monitor.

Features for Use with Performance Testing and Business Service Management

UFT offers several features that are designed specifically for integration with LoadRunner, Performance Center, and Business Process Monitor.

Note: These products are designed to run tests using virtual or synthetic users representing many users simultaneously performing standard user operations. Some features may not be available when integrating these products with UFT.

You can use the **Services** object and its associated methods to insert statements that are specifically relevant to Performance Testing and Business Service Management. These include:

AddWastedTime	EndTransaction	SetTransaction
EndDistributedTransaction	LogMessage	SetTransactionStatus
GetEnvironmentAttribute	Rendezvous	ThinkTime
StartDistributedTransaction	StartTransaction	UserDataTableUserDataTable

For details on these methods, see the **Services** object of the *HP UFT Object Model Reference for GUI Testing* and your HP performance testing or Business Service Management documentation.

For details on transactions, see "[Measuring Transactions](#)" on page 158.

Advantages of Running Tests in LoadRunner

The main advantages of running tests in LoadRunner are:

- To check how your application's functionality is affected by heavy load
- To measure the response time that a typical user experiences on the client side while your application is under load (end-to-end response time)

For example, you can add tests to specific points in a LoadRunner scenario to confirm that the application's functionality is not affected by the extra load at those sensitive points.

Another advantage of using a virtual GUI user (GUI Vuser) script as part of your LoadRunner scenario is that the GUI Vuser script runs on your screen during the scenario, enabling you to watch the actual steps executed by the Vuser in real time.

Designing Tests for Use HP Performance Testing and HP Business Service Management Products

If you plan to use the same test in both UFT and LoadRunner, Performance Center, and/or Business Process Monitor, you should take into account the different options supported in each product as you design your test. For details, see:

- "[Designing Tests for HP Performance Testing Products](#)" on the next page
- "[Designing GUI Tests for HP Business Process Monitor](#)" on the next page

Running Tests from HP Performance Testing and HP Business Service Management Products

The run mechanisms used in all HP Performance Testing and HP Business Service Management products are the same. This means that you can create tests that are compatible with LoadRunner, Performance Center, and Business Process Monitor, enabling you to take advantage of tests or test segments that have already been designed and debugged in UFT.

For example, you can add tests to specific points in a performance test to confirm that the application's functionality is not affected by the extra load at those sensitive points. You can also run tests on Business Process Monitor to simulate end user experience and ensure that your application is running correctly and in a timely manner.

If you plan to use the same test in both UFT and LoadRunner, Performance Center, and/or Business Process Monitor, you should take into account the different options supported in each product before you run your test. For details, see:

- "[Running GUI Tests from HP Performance Testing Products](#)" on page 156
- "[Running GUI Tests from HP Business Process Monitor](#)" on page 157

Designing Tests for HP Performance Testing Products

Relevant for: GUI tests only

Consider the following guidelines when designing tests for use with performance testing products:

- The tests you use with LoadRunner and Performance Center should be simple, designed to pinpoint specific operations, and should avoid using external actions and references to other external files (including resources stored in ALM). Also, when working with action iterations, corresponding **StartTransaction** and **EndTransaction** statements must be contained within the same action.
- Every test must contain at least one transaction to provide useful information in the performance test. LoadRunner and Performance Center use only the data that is included within a transaction, and ignore any data in a test outside of a transaction.
- Do not include references to external actions or other external resources (including resources stored in ALM), such as an external data table file, environment variable file, shared object repositories, function libraries, and so forth. This is because LoadRunner or Performance Center may not have access to the external action or resource.

(However, if the resource can be found on the network, UFT will use it. For example, you can try defining external resources via an absolute path, or by adding them as supplementary files and transferring them to Load Generator in the GUI test folder.)

- Make sure that the last step(s) in the test closes the application being tested, as well as any child processes that are running. This enables the next iteration of the test to open the application again.

For details on working with LoadRunner or Performance Center, see your HP performance testing documentation.

Designing GUI Tests for HP Business Process Monitor

Relevant for: GUI tests only

Note: As of Business Service Management 9.0, profiles are no longer used. Instead, Business Process Monitor uses Business Transaction Flows, which are included in the parent application's run unit and can be run as part of that unit or independently, as needed.

Consider the following guidelines when designing tests for use with Business Process Monitor:

- The tests you use with Business Process Monitor should be simple, designed to pinpoint specific operations, and should avoid using external actions and references to other external files (including resources stored in ALM). Also, when working with action iterations, corresponding **StartTransaction** and **EndTransaction** statements must be contained within the same action.
- Every test must contain at least one transaction to provide useful information in Business Process Monitor. Business Process Monitor uses only the data that is included within a transaction, and ignores any data in a test outside of a transaction.

- Business Process Monitor does not support running tests that require access to external resources, including resources stored in ALM (such as a shared object repository, function library, external data table, external actions, and so forth). Tests that require external resources may fail to run on Business Process Monitor. (However, if the resource can be found on the network, UFT will use it.)
- Make sure that the last steps in the test close the application being tested, as well as any child processes that are running. This cleanup step enables the next test run to open the application again.
- When measuring a distributed transaction over two different Business Process Monitor profiles or Business Transaction Flows (depending on the version), the profile with the **StartDistributedTransaction** statement must be run before the profile with the associated **EndDistributedTransaction**.
- When measuring distributed transactions, make sure that you relate the tests to a single Business Process Monitor instance. Business Process Monitor searches for the end transaction name in all instances, and may close the wrong distributed transaction if it is included in more than one instance.
- When measuring a distributed transaction over two Business Process Monitor profiles, make sure that the timeout value you specify is large enough so that the profile or Business Transaction Flow (depending on the version) that contains the **StartDistributedTransaction** step and all the profiles that run before the profile that contains the **EndDistributedTransaction** step, will finish running in a time that is less than the value of the specified timeout.

Running GUI Tests from HP Performance Testing Products

Relevant for: GUI tests only

Consider the following guidelines when running tests from HP Performance Testing Products:

- You can run only one GUI Vuser concurrently per computer. (A GUI Vuser is a Vuser that runs a GUI test.)
- Ensure that UFT is closed on the UFT computer before running a test in Performance Center or LoadRunner.
- The settings in the LoadRunner or Performance Center Run-time Settings dialog box are not relevant for tests.
- You cannot use the **ResultDir** environment variable when running a performance test.
- Transaction breakdown is not supported for tests (scripts) created with UFT.
- UFT cannot run on a computer that is:
 - Logged off or locked. In these cases, consider running UFT on a terminal server.
 - Already running a test. Make sure that the test is finished before starting to run another test.

Tip: You can simulate how the test will run from a performance testing product by using Silent Test Runner. For details, see "[Silent Test Runner](#)" on page 160.

How to Insert and Run GUI Tests in Performance Center and LoadRunner

Relevant for: GUI tests only

The following are various tasks that you can perform when working with HP Performance Center and LoadRunner.

To insert a test in a LoadRunner scenario

In the Controller Open Test dialog box, browse to the test folder and select **QuickTest Tests or GUI Scripts** (for tests created in the SAP environment) in the **Files of type** box. This enables you to view the tests in the folder.

To use a UFT test in Performance Center

Create a zipped version of the test, and upload it to the Performance Center User Site Vuser Scripts Page.

To run multiple GUI Vusers on the same application

Open a terminal server session for each GUI Vuser. For details, refer to the HP performance testing documentation.

For details on working with LoadRunner or Performance Center, see your HP performance testing documentation.

Running GUI Tests from HP Business Process Monitor

Relevant for: GUI tests only

Consider the following guidelines when running tests from HP Business Process Monitor:

- Before you try to run a test in Business Process Monitor, check which versions of UFT are supported by your version of Business Process Monitor. For details, see the Business Process Monitor documentation.
- To run a test in Business Process Monitor, UFT must be installed and closed on the Business Process Monitor computer
- Business Process Monitor can run only one test at a time. Make sure that the previous UFT run session is finished before starting to run another test.
- Transaction breakdown is not supported for tests created with UFT.

- Tests must be zipped before uploading them to Business Service Management Admin. If you make changes to your local copy of a test after uploading it to Business Service Management, upload the zipped test again to enable Business Process Monitor to run the test with your changes.
- UFT cannot run tests on a computer that is logged off, locked, or running UFT as a non-interactive service.
- You should start Business Process Monitor by running the `magentproc.exe` program when running tests from Business Process Monitor.
- You cannot use the **ResultDir** environment variable when running a test in Business Process Monitor.

For details on working with Business Service Management, see the relevant documentation—specifically the sections describing UFT scripts in the Business Process Monitor Administration guide and the End User Management guide.

Tip: You can simulate how the test will run from Business Process Monitor by using Silent Test Runner. For details, see "[Silent Test Runner](#)" on page 160.

Measuring Transactions

Relevant for: GUI tests only

You can measure how long it takes to run a section of your test by defining **transactions**. A transaction represents the process in your application that you are interested in measuring. Your test must include transactions to be used by LoadRunner, Performance Center, or the Business Process Monitor. These products use only the data that is included within a transaction, and ignore any data in a test outside of a transaction.

You define transactions within your test by enclosing the appropriate sections of the test with **start** and **end** transaction statements. For example, you can define a transaction that measures how long it takes to reserve a seat on a flight and for the confirmation to be displayed on the client's terminal.

During the run session, the **StartTransaction** step signals the beginning of the time measurement. The time measurement continues until the **EndTransaction** step is reached. The test results for the **EndTransaction** step include the transaction's name, end status, total duration, and wasted time.

During a run session, UFT runs background processes that add to the time it takes to run a test. Wasted time is the time within the total duration that was added as a result of UFT running the transaction. If the application ran the transaction without UFT, the total duration would equal the total duration minus the wasted time.

Note: If you start a transaction while there is already open transaction with the same name, the previous transaction is ended with **Fail** status and then the new transaction is started.

There is no limit to the number of transactions that can be added to a test.

Tip:

You can:

- Insert a transaction within a transaction.
- Insert a variety of transaction-related statements using the Step Generator or Editor. For details, see the **Services** object topic in the *HP UFT Object Model Reference for GUI Testing*.
- Enter Start Transaction and End Transaction steps using UFT.

For details on the statements you can use in transactions, see the *HP UFT Object Model Reference for GUI Testing*.

Example of a test with a transaction

Part of a sample test with a transaction is shown below, as it is displayed in the Keyword View:

Services	StartTransaction	"ReserveSeat"	Start the "ReserveSeat" transaction.
Find a Flight: Mercury	Select	"London"	Select the "London" item in the "fromPort" list.
fromPort	Select	"Frankfurt"	Select the "Frankfurt" item in the "toPort" list.
toPort	Select	"12"	Select the "12" item in the "toDay" list.
toDay	Select	"Business"	Select radio button "Business" in the "servClass" radio button group.
servClass	Select	"Blue Skies Airlines"	Select the "Blue Skies Airlines" item in the "airline" list.
airline	Click	65,12	Click the "findFlights" image.
findFlights	Select	"Blue Skies Airlines"	Select radio button "Blue Skies Airlines" in the "outFlight" radio button group.
Select a Flight: Mercury...	Select	"Blue Skies Airlines"	Select radio button "Blue Skies Airlines" in the "inFlight" radio button group.
outFlight	Select	"Blue Skies Airlines"	Select radio button "Blue Skies Airlines" in the "inFlight" radio button group.
inFlight	Click	46,8	Click the "reserveFlights" image.
reserveFlights	EndTransaction	"ReserveSeat"	End the "ReserveSeat" transaction.
Services			
Book a Flight: Mercury_2			

 ``` End transaction ``` |

Services	StartTransaction	"ReserveSeat"	Start the "ReserveSeat" transaction.
Find a Flight: Mercury	Select	"London"	Select the "London" item in the "fromPort" list.
fromPort	Select	"Frankfurt"	Select the "Frankfurt" item in the "toPort" list.
toPort	Select	"12"	Select the "12" item in the "toDay" list.
toDay	Select	"Business"	Select radio button "Business" in the "servClass" radio button group.
servClass	Select	"Blue Skies Airlines"	Select the "Blue Skies Airlines" item in the "airline" list.
airline	Click	65,12	Click the "findFlights" image.
findFlights	Select	"Blue Skies Airlines"	Select radio button "Blue Skies Airlines" in the "outFlight" radio button group.
Select a Flight: Mercury...	Select	"Blue Skies Airlines"	Select radio button "Blue Skies Airlines" in the "inFlight" radio button group.
outFlight	Select	"Blue Skies Airlines"	Select radio button "Blue Skies Airlines" in the "inFlight" radio button group.
inFlight	Click	46,8	Click the "reserveFlights" image.
reserveFlights	EndTransaction	"ReserveSeat"	End the "ReserveSeat" transaction.
Services			
Book a Flight: Mercury_2			

The same part of the test is displayed in the Editor as follows:

```
Services.StartTransaction "ReserveSeat"
Browser("Welcome: Mercury Tours").Page("Find a Flight: Mercury"). WebList
("fromPort").Select "London"
Browser("Welcome: Mercury Tours").Page("Find a Flight: Mercury"). WebList
("toPort").Select "Frankfurt"
Browser("Welcome: Mercury Tours").Page("Find a Flight: Mercury"). WebList
("toDay").Select "12"
Browser("Welcome: Mercury Tours").Page("Find a Flight: Mercury"). WebRadioGroup("servClass").Select "Business"
Browser("Welcome: Mercury Tours").Page("Find a Flight: Mercury"). WebList
("airline").Select "Blue Skies Airlines"
Browser("Welcome: Mercury Tours").Page("Find a Flight: Mercury"). Image
("findFlights").Click 65,12
Browser("Welcome: Mercury Tours").Page("Select a Flight: Mercury"). WebRadioGroup("outFlight").Select "Blue Skies Airlines"
Browser("Welcome: Mercury Tours").Page("Select a Flight: Mercury"). WebRadioGroup("inFlight").Select "Blue Skies Airlines"
Browser("Welcome: Mercury Tours").Page("Select a Flight: Mercury"). Image
```

```
("reserveFlights").Click 46,8
Services.EndTransaction "ReserveSeat"
```

## Silent Test Runner

### **Relevant for: GUI tests only**

Silent Test Runner enables you to simulate the way a test runs from LoadRunner, Performance Center, and Business Service Management. When you run a test using Silent Test Runner, it runs without opening the UFT user interface, and the test runs at the same speed as when it is run from LoadRunner, Performance Center, or Business Service Management. At the end of the test run, you can view information about the test run and transaction times. For details, see "[Test Run Information for Silent Runs](#)" below.

You can also use Silent Test Runner to verify that your test is compatible with LoadRunner, Performance Center, and Business Service Management. A test will fail when run using Silent Test Runner if it uses a feature that is not supported by these products. For details on features that are not supported, see "[Designing Tests for HP Performance Testing Products](#)" on page 155, "[Running GUI Tests from HP Performance Testing Products](#)" on page 156, "[Designing GUI Tests for HP Business Process Monitor](#)" on page 155, and "[Running GUI Tests from HP Business Process Monitor](#)" on page 157.

## Test Run Information for Silent Runs

Silent Test Runner provides test run information in log files. Each test generates a test run log, and any test with transactions generates an additional transaction summary.

### **[Viewing the Test Run Log](#)**

The test run log is saved as `output.txt` in the `<Unified Functional Testing>\Tests\<test name>` folder. A log file is saved for each test run with Silent Test Runner and is overwritten when you rerun the test. To open the log file, click **Test Run Log**.

The log file displays information about the test run. For example, information is shown about each iteration, action call, step transaction, failed step, and so forth. Each line displays a message or error ID. For details on message and error codes in the log file, see your Performance Center or Business Service Management documentation.

## Viewing the Transaction Summary

The transaction summary is saved as `transactions.txt` in the `<Unified Functional Testing>\Tests\<test name>` folder. A transaction summary is saved for each test that includes transactions and is overwritten when you rerun the test. To open the log file, click **Transaction Summary**. The transaction summary displays a line for each transaction in the test. For each transaction, the status is displayed together with the total duration time and any wasted time (in seconds). The transaction measurements in Silent Test Runner are exactly the same as if the test was run from LoadRunner, Performance Center, or Business Service Management.

**Note:**

- A transaction summary is available only for a test that contains transactions ending with an **EndTransaction** statement. If a transaction started but did not end because of test failure, it is not included in the transaction summary.
- Distributed transactions (transactions that start in one test and end in another) are not reported in the transaction summary but are included in the test run log.
- Any transaction information included in the transaction summary is also included in the test run log.

# Part 5: GUI Testing: Test Objects, Checkpoints and Output Values

# Chapter 30: The Test Object Model

**Relevant for: GUI tests and components**

This chapter includes:

• <a href="#">Test Object Model - Overview</a> .....	164
• <a href="#">How UFT Learns Objects</a> .....	164
• <a href="#">How UFT Identifies Objects During a Run Session</a> .....	165
• <a href="#">How UFT Applies the Test Object Model Concept</a> .....	166
• <a href="#">Test Object Descriptions</a> .....	168
• <a href="#">UFT Test Object Hierarchy</a> .....	169
• <a href="#">Properties and Operations for Test Objects and Run-Time Objects</a> .....	170
• <a href="#">Object Identification Process Workflow</a> .....	172
• <a href="#">How to Use the Object Spy to View Object Properties and Operations or Add an Object to a Repository</a> .....	173
• <a href="#">How to Use the Remote Object Spy to View or Add Objects Running on a Remote Computer</a> .....	174
• <a href="#">Troubleshooting and Limitations - Object Spy</a> .....	177

# Test Object Model - Overview

## **Relevant for: GUI tests and components**

UFT tests your dynamically changing application by learning and identifying test objects and their expected properties and values. To do this, UFT analyzes each object in your application in much the same way that a person would look at a photograph and remember its details.

The following sections introduce the concepts related to the test object model and describe how UFT uses the information it gathers to test your application.

To learn more, see:

- [How UFT Learns Objects](#) ..... 164
- [How UFT Identifies Objects During a Run Session](#) ..... 165

## How UFT Learns Objects

### **Relevant for: GUI tests and components**

UFT learns objects just as you would. For example, suppose as part of an experiment, Alex is told that he will be shown a photograph of a picnic scene for a few seconds during which someone will point out one item in the picture. Alex is told that he will be expected to identify that item again in identical or similar pictures one week from today.

Before he is shown the photograph, Alex begins preparing himself for the test by thinking about which characteristics he wants to learn about the item that the tester indicates. Obviously, he will automatically note whether it is a person, inanimate object, animal, or plant. Then, if it is a person, he will try to commit to memory the gender, skin color, and age. If it is an animal, he will try to remember the type of animal, its color, and so forth.

The tester shows the scene to Alex and points out one of three children sitting on a picnic blanket. Alex notes that it is a Caucasian girl about 8 years old. In looking at the rest of the picture, however, he realizes that one of the other children in the picture could also fit that description. In addition to learning his planned list of characteristics, he also notes that the girl he is supposed to identify has long, brown hair.

Now that only one person in the picture fits the characteristics he learned, he is fairly sure that he will be able to identify the girl again, even if the scene the tester shows him next week is slightly different.

Since he still has a few moments left to look at the picture, he attempts to notice other, more subtle differences between the child he is supposed to remember and the others in the picture—just in case.

If the two similar children in the picture appeared to be identical twins, Alex might also take note of some less permanent feature of the child, such as the child's position on the picnic blanket. That would enable him to identify the child if he were shown another picture in which the children were sitting on the blanket in the same order.

UFT uses a very similar method when it learns objects.

First, it "looks" at the object being learned and stores it as a **test object**, determining in which test object class it fits. In the same way, Alex immediately checked whether the item was a person, animal, plant, or inanimate object. UFT might classify the test object as a standard Windows dialog box (Dialog), a Web button (WebButton), or a Visual Basic scroll bar object (VbScrollBar), for example.

Then, UFT "considers" the **identification properties** for the test object. For each test object class, UFT has a list of **mandatory** properties that it always learns; similar to the list of characteristics that Alex planned to learn before seeing the picture. When UFT learns an object, it always learns these default property values, and then "looks" at the rest of the objects on the page, dialog box, or other parent object to check whether this **description** is enough to uniquely identify the object. If not, UFT adds **assistive** properties, one by one, to the description, until it has compiled a unique description; similar to when Alex added the hair length and color characteristics to his list. If no assistive properties are available, or if those available are not sufficient to create a unique description, UFT adds a special **ordinal identifier**, such as the object's location on the page or in the source code, to create a unique description, just as Alex would have remembered the child's position on the picnic blanket if two of the children in the picture had been identical twins.

## How UFT Identifies Objects During a Run Session

### Relevant for: GUI tests and components

UFT uses a human-like technique for identifying objects during the run session.

Suppose as a continuation to the experiment (described in "[How UFT Applies the Test Object Model Concept](#)" on the next page), Alex is now asked to identify the same "item" he initially identified but in a new, yet similar environment.

The first photograph he is shown is the original photograph. He searches for the same Caucasian girl, about eight years old, with long, brown hair that he was asked to remember and immediately picks her out. In the second photograph, the children are playing on the playground equipment, but Alex is still able to easily identify the girl using the same criteria.

Similarly, during a run session, UFT searches for a **run-time object** that exactly matches the description of the test object it learned previously. It expects to find a perfect match for both the mandatory and any assistive properties it used to create a unique description while learning the object. As long as the object in the application does not change significantly, the description learned is almost always sufficient for UFT to uniquely identify the object. This is true for most objects, but your application could include objects that are more difficult to identify during subsequent run sessions.

For example, Alex is told he will have to recognize a particular tree out of multiple trees in the photo, and he knows he is going to have to be able to identify it again in a photo taken from a different angle. If there isn't enough clearly identifying information about the tree itself, then he might take note of where the tree is located relative to some other permanent item, such as a nearby lamp post or picnic table. Then he will be able to identify the tree again, even if the next picture he sees is from a different angle (as long as all the required items are still visible in the picture).

This is similar to the **visual relation identifier** property, which enables UFT to identify test objects according to their neighboring objects in the application. You use this property to link less stable test objects to more unique test objects, and as long as those objects in the application maintain their relative location to your object, UFT should still be able to identify the test object even after predictable user interface changes in the application.

Consider the final phase of Alex's experiment. In this phase, the tester shows Alex another photograph of the same family at the same location, but the children are older and there are also more children playing on the playground. Alex first searches for a girl with the same characteristics he used to identify the girl in the other pictures (the test object), but none of the Caucasian girls in the picture have long, brown hair. Luckily, Alex was smart enough to remember some additional information about the girl's appearance when he first saw the picture the previous week. He is able to pick her out (the run-time object), even though her hair is now short and dyed blond.

How is he able to do this? First, he considers which features he knows he must find. Alex knows that he is still looking for a Caucasian female, and if he were not able to find anyone that matched this description, he would assume she is not in the photograph.

After he has limited the possibilities to the four Caucasian females in this new photograph, he thinks about the other characteristics he has been using to identify the girl—her age, hair color, and hair length. He knows that some time has passed and some of the other characteristics he remembers may have changed, even though she is still the same person.

Thus, since none of the Caucasian girls have long, dark hair, he ignores these characteristics and searches for someone with the eyes and nose he remembers. He finds two girls with similar eyes, but only one of these has the petite nose he remembers from the original picture. Even though these are less prominent features, he is able to use them to identify the girl.

UFT uses a very similar process of elimination with its **Smart Identification** mechanism to identify an object, even when the learned description is no longer accurate. Even if the values of your identification properties change, UFT maintains the reusability of your test or component by identifying the object using Smart Identification. For details on Smart Identification, see "[Configuring Object Identification](#)" on page 227.

The remainder of this guide assumes familiarity with the concepts presented here, including test objects, run-time objects, identification properties (including mandatory and assistive properties), visual relation identifiers, and Smart Identification. An understanding of these concepts will enable you to create well-designed, functional tests and components for your application.

For a flowchart illustrating the general object identification process, see "[Object Identification Process Workflow](#)" on page 172.

## How UFT Applies the Test Object Model Concept

### Relevant for: GUI tests and components

The test object model is a large set of object types or classes that UFT uses to represent the objects in your application. Each test object class has a list of identification properties that UFT can learn about

the object, a sub-set of these properties that can uniquely identify objects of that class, and a set of relevant operations that UFT can perform on the object.

A **test object** is an object that UFT creates in a test or component to represent the actual object in your application. UFT stores information on the object that will help it identify and check the object during the run session.

A **run-time object** is the actual object in your application on which methods are performed during the run session.

When UFT learns an object in your application, it adds the corresponding test object to an **object repository**, which is a storehouse for objects. You can add test objects to an object repository in several ways. For example, you can use the Navigate and Learn option, add test objects manually, or perform an operation on your application while recording. For details, see "[Test Objects in Object Repositories](#)" on [page 178](#).

When you add an object to an object repository, UFT:

- Identifies the UFT test object class that represents the learned object and creates the appropriate test object.
- Reads the current value of the object's properties in your application and stores the list of **identification properties** and values with the test object.
- Chooses a unique name for the test object, generally using the value of one of its prominent properties.

## Example

Suppose you add a **Search** button with the following HTML source code:

```
<INPUT TYPE="submit" NAME="Search" VALUE="Search">
```

UFT identifies the object as a **WebButton** test object. In the object repository, UFT creates a WebButton object with the name Search, learns a set of identification properties for the object, and decides to use the following properties and values to uniquely identify the **Search** WebButton:

Name	Value
- Description properties	
type	submit
name	Search
html tag	INPUT

If you add an object to an object repository by recording on your application, UFT records the operation that you performed on the object using the appropriate UFT test object method. For example, UFT records that you performed a **Click** method on the WebButton.

**In an action:** UFT displays your step in the Editor like this:

```
Browser("Search Results: Search").Page("Search Results: Search").WebButton("Search").Click
```

and in the Keyword View like this:

▼  Search Results: Search	Sync	Wait for the browser to complete the current navigation.
▼  Search Results: Search	Sync	Wait for the Web page to synchronize before continuing the run.
 Search	Click	Click the "Search" button.

**In a component:** The hierarchy is not displayed, and the step is displayed like this:

Item	Operation	Documentation
 Search	Click	Click the "Search" button.

When you run a test or component, UFT identifies each object in your application by its test object class and its **description** (the set of identification properties and values used to uniquely identify the object). The list of test objects and their properties and values are stored in the object repository. In the above example, UFT would search in the object repository during the run session for the WebButton object with the name Search to look up its description. Based on the description it finds, UFT would then look for a WebButton object in the application with the HTML tag INPUT, of type submit, with the value Search. When it finds the object, it performs the **Click** method on it.

To learn more, see:

- [Test Object Descriptions](#) ..... 168
- [UFT Test Object Hierarchy](#) ..... 169
- [Properties and Operations for Test Objects and Run-Time Objects](#) ..... 170

## Test Object Descriptions

### Relevant for: GUI tests and components

For each test object class, UFT learns a set of identification properties when it learns an object, and selects a sub-set of these properties to serve as a unique object description. UFT then uses this description to identify the object when it runs a test or component.

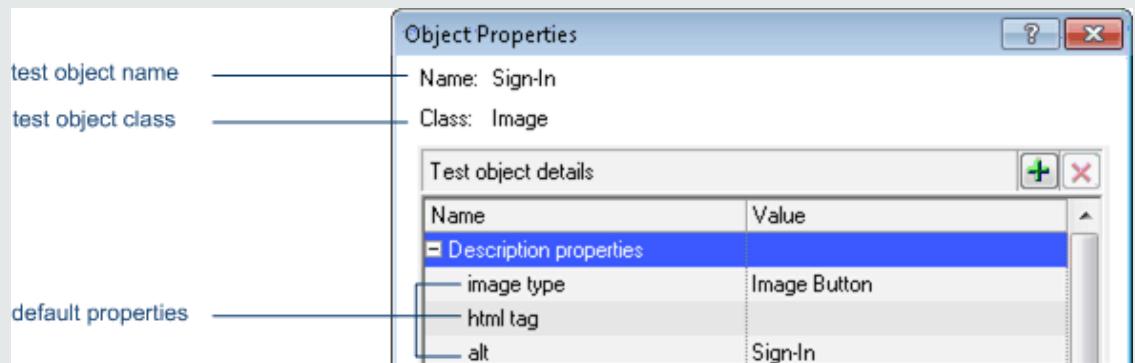
When the test or component runs, UFT searches for the object that matches the description it learned. If it cannot find any object that matches the description, or if it finds more than one object that matches, UFT may use the Smart Identification mechanism to identify the object.

You can configure the mandatory, assistive, and ordinal identifier properties that UFT uses to learn the descriptions of the objects in your application, and you can enable and configure the Smart Identification mechanism. For details, see "[Configuring Object Identification](#)" on page 227.

### Example

By default, UFT learns the image type (such as plain image or image button), the **html tag**, and

the **Alt** text of each Web image it learns.



If these three mandatory property values are not sufficient to uniquely identify the object within its parent object, UFT adds some assistive properties and/or an ordinal identifier to create a unique description.

When using Insight, UFT learns objects based on their appearance instead of retrieving properties from the objects. For the description property, UFT stores an image of the object, which can be used later to identify the object. If parts of the object do not always look the same, you can instruct UFT to ignore those areas when it uses the image to identify the object.

If necessary, UFT can also use an ordinal identifier to create a unique description for the object. Other aspects of object configuration, such as mandatory and assistive properties, and smart identification, are not relevant for Insight test objects.

After UFT creates a description for an Insight test object, you can add visual relation identifiers to improve identification of the object. If necessary, you can also add the **similarity** identification property to the test object description. This property is a percentage that specifies how similar a control in the application has to be to the test object image for it to be considered a match.

## UFT Test Object Hierarchy

### **Relevant for:** GUI tests and components

The UFT test object hierarchy comprises one or more levels of test objects. The top level object may represent a window, dialog box, or browser type object, depending on the environment. The actual object on which you perform an operation may be learned as a top level object, a second level object, for example, Window.WinToolbar, or a third level object, for example, Browser.Page.WebButton.

In some cases, even though the object in your application may be embedded in several levels of objects, the hierarchy does not include these objects. For example, if a WebButton object in your application is actually contained in several nested WebTable objects, which are all contained within a Browser and Page, the learned object hierarchy is only Browser . Page . WebButton.

An object that can potentially contain a lower-level object is called a container object. All top-level objects in the object hierarchy are container objects. If a second-level object contains third-level objects

according to the UFT object hierarchy, then that object is also considered a container object. For example, in the step `Browser.Page.Edit.Set "David"`, Browser and Page are both container objects.

For details on the UFT object hierarchy for specific environments, see the relevant section in the *HP Unified Functional Testing Add-ins Guide*.

## Properties and Operations for Test Objects and Run-Time Objects

### Relevant for: GUI tests and components

UFT uses unique terms to differentiate between the properties and operations for test objects and run-time objects. The table below introduces some of these terms.

UFT Test Objects	Run-time Objects From Your Application
<p><b>Identification properties</b> are UFT-specific properties that UFT uses to identify objects in applications, to retrieve and store information about those objects, or to compare stored values with the current values of an object in an application.</p> <p>The identification properties available for a test object are determined by its test object class (and not by the actual properties available for the object in the application).</p>	<p><b>Native properties</b> are the properties created by the object creator for each run-time object. (Examples of object creators include Microsoft for Microsoft Internet Explorer objects and the product developer for ActiveX objects.)</p>
<p>A <b>test object operation</b> is a method or property that UFT can perform on an object from a particular test object class.</p> <p><b>Example:</b> UFT can perform the <b>Click</b> method on a WebButton test object.</p>	<p><b>Native operations</b> are the methods of the object in your application as defined by the object creator.</p>

As you add steps to your test or component, you specify which operation to perform on each test object. If you record steps, UFT records the relevant operation as it is performed on an object. During a run session, UFT performs the specified test object operation on the run-time object.

The sections below describe some of the ways in which you can view and modify properties and operations for test objects and run-time objects.

### For test objects

- You can retrieve or modify identification property values manually while designing your test or component, or you can use **SetTOProperty** statements during a run session (via an operation defined in a function library).

For details, see "[Test Objects in Object Repositories](#)" on page 178, "[Retrieving and Setting Identification Property Values](#)" on page 661, and the **SetTOProperty** method in the **Common Methods and Properties** section of the *HP UFT Object Model Reference for GUI Testing*.

- You can use regular expressions in function libraries to identify property values based on conditions or patterns you define.

For details, see ["Regular Expressions Overview" on page 372](#).

- You can parameterize identification property values with data table parameters so that a different value is used during each iteration of the test.

For details, see ["Parameterizing Object Values" on page 336](#).

- You can view or modify the identification property values that are stored with your test or component in the Object Properties or Object Repository window.

For details, see ["Maintaining Identification Properties" on page 192](#).

- You can view the current identification property values of any visible object using the Properties tab of the Object Spy.

For details, see ["How to Use the Object Spy to View Object Properties and Operations or Add an Object to a Repository" on page 173](#).

- You can view the syntax of the test object operations of any visible object using the Operations tab of the Object Spy. For details, see ["How to Use the Object Spy to View Object Properties and Operations or Add an Object to a Repository" on page 173](#).

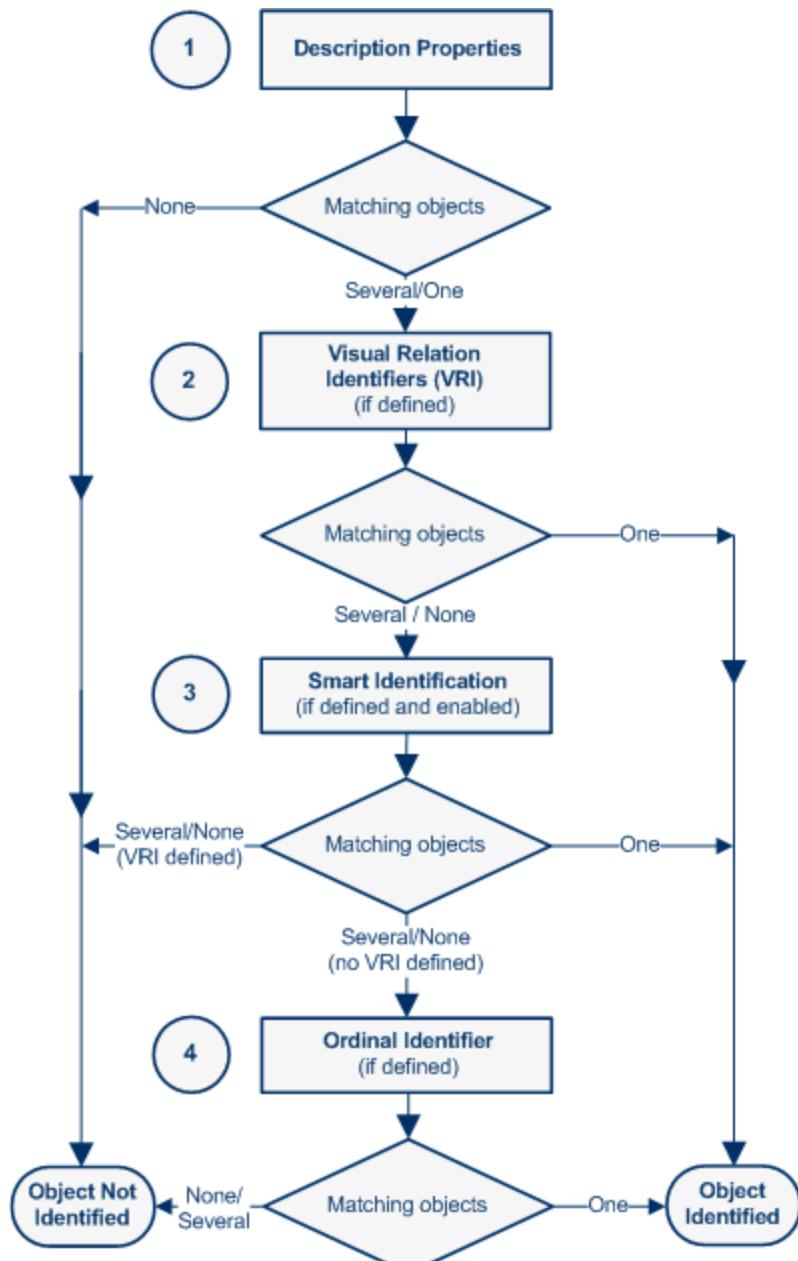
## For run-time objects

- You can view the syntax of the native operations of any visible object using the Operations tab of the Object Spy. For details, see ["How to Use the Object Spy to View Object Properties and Operations or Add an Object to a Repository" on page 173](#).
- You can retrieve native property values from the run-time object during the run session by adding **GetRProperty** statements. For details, see ["Retrieving and Setting Identification Property Values" on page 661](#).
- If the available test object operations and identification properties do not provide the functionality you need, you can access the internal operations and native properties of the run-time object using the **Object** property. You can also use the **attribute** object property to identify Web objects in your application according to user-defined properties. For details, see ["Native Properties and Operations" on page 663](#) and the **Object** property in the **Common Methods and Properties** section of the *HP UFT Object Model Reference for GUI Testing*.
- When using Insight test objects, which UFT recognizes in the application based on the object's appearance, UFT does not use the object's programming interface and therefore native operations and properties are not relevant.

# Object Identification Process Workflow

## Relevant for: GUI tests and components

The following flowchart provides a general overview of the main process of UFT object identification.



**Note for Web-based objects:**

- If you defined Web object identifiers (such as XPath/CSS properties) for these test objects, they are used before the description properties. If one or more objects are found, UFT continues to identify the object using the description properties. For details, see the section on Web Object Identifiers (described in the *HP Unified Functional Testing Add-ins Guide*).
- Additional UFT-generated properties, such as **source index** or **automatic XPath**, may also affect the object identification process. You enable these properties in the Advanced Web Options tab of the Options dialog box (described in the *HP Unified Functional Testing Add-ins Guide*) (**Tools > Options > GUI Testing tab > Web > Advanced** node).

## How to Use the Object Spy to View Object Properties and Operations or Add an Object to a Repository

### Relevant for: GUI tests and components

This task describes how to use the Object Spy to view identification properties, native properties, test object operations, or native operations, as well as add an object to an object repository.

If you are working with Web applications running in Safari on a remote Mac computer, see "[How to Use the Remote Object Spy to View or Add Objects Running on a Remote Computer](#)" on the next page.

This task includes the following steps:

- "Prerequisites" below
- "Open the Object Spy dialog box" below
- "Select the details you want to view for the object" below
- "Resize the Object Spy dialog box - Optional" on the next page
- "Use the pointing hand to select the application object on which you want to Spy" on the next page
- "Add selected objects to the object repository - Optional" on the next page

### 1. Prerequisites

Open your application to the page containing the object on which you want to spy.

### 2. Open the Object Spy dialog box

Select Tools > Object Spy.

### 3. Select the details you want to view for the object

In the Object Spy dialog box, select the **Properties** or **Operations** tab, and then select the radio button to view the **Native** or **Identification** properties. For details about using the run-time object's operations or retrieving the values of its properties, see "[Retrieving and Setting Identification Property Values](#)" on page 661 and "[Native Properties and Operations](#)" on page 663.

#### 4. Resize the Object Spy dialog box - Optional

This is useful if the objects on which you want to spy have a deep hierarchy or long property names and values, as it enables you to view all of the information without scrolling.

#### 5. Use the pointing hand to select the application object on which you want to Spy

In the Object Spy, click the pointing hand button and then mouse over or click an object in your application. In most environments, as you mouse over objects in your application, the Object Spy highlights the object it identifies and displays the relevant information in the Object Spy, according to the options you selected in the Object Spy dialog box.

**Note:** If UFT does not recognize your objects in the correct location, check to see that you are viewing the page at 100%, and are not zooming in or out of the page.

For example, if you view the page at 90% or 120%, you may be required to click or select an area to the left or the right of the actual object in order to recognize it.

When you click an object in your application, the Object Spy captures its information, and you can:

- Change the selected radio button or tab in the Object Spy to view additional details.
- View properties, values, or operations of other test objects currently displayed in the **Object hierarchy** tree, by selecting that test object in the tree.
- Highlight the object in the application, by clicking the **Highlight in Application** button .

#### 6. Add selected objects to the object repository - Optional

After clicking an object you can add it (or any other object in the **Object hierarchy** tree) to the object repository using the **Add to Repository** button .

If an object in the **Object hierarchy** tree already exists in a repository associated with the active action or component, a repository icon  is displayed in the lower-right corner of the object's icon.

## How to Use the Remote Object Spy to View or Add Objects Running on a Remote Computer

### Relevant for: GUI tests and components

Use the Remote Object Spy when working with Web applications running in Safari on a Remote Mac computer. The object information and abilities provided by the Remote Object Spy are similar to those provided by the Object Spy.

This task describes the steps required to enable the Remote Object Spy to capture the object information.

**Note:** You perform some of the steps in UFT and some on the Mac computer. You can perform the Mac steps directly on the Mac computer or using a remote access program, such as Virtual Network Computing (VNC).

This task includes the following steps:

- "Prerequisites" below
- "In UFT: Open the Remote Object Spy dialog box and click the pointing hand" below
- "On the Mac computer: Use the mouse to select the application object on which you want to spy" below
- "How to Use the Remote Object Spy to View or Add Objects Running on a Remote Computer" on the previous page

### 1. Prerequisites

- a. **In UFT:** Make sure that UFT is connected to a Remote Mac computer, using the Remote Connection button  in UFT's toolbar. For details, see the topic on How to Connect to a Remote Mac Computer in the *HP Unified Functional Testing Add-ins Guide*.
- b. **On the Mac computer:** Open Safari to the page containing the object on which you want to spy, and make sure that the relevant object is visible.

### 2. In UFT: Open the Remote Object Spy dialog box and click the pointing hand

- a. Make sure that a GUI test or action is in focus in the document pane or selected in the Solution Explorer.
- b. Use one of the following:
  - Click the down arrow near the **Object Spy** toolbar button , and select the **Remote Object Spy**  button.
  - Select **Tools > Remote Object Spy**.

### 3. On the Mac computer: Use the mouse to select the application object on which you want to spy

- When you click the pointing hand in UFT, the Unified Functional Testing Agent Extension icon  in the Safari toolbar on the Mac changes to a UFT Spy button , indicating that the Spy mode is active.

You may want to suspend the Spy mode and use your mouse as usual on the Mac, to load Web pages, move applications, or perform any other steps necessary to display the object on which you want to spy.

- Click the **Pause/Resume UFT Spy** toggle button  in the Safari toolbar to pause and resume the Spy session. This affects all open Safari browsers simultaneously.

- Hold the Mac's Command key  to momentarily suspend the Spy mode.

**Tip:** The Command key may be mapped to your Windows **start** key, or to the **ALT** key, depending on how you connect to your Mac computer.
  - While the Spy mode is active, as you mouse over Web objects in Safari, the Object Spy highlights the object and displays the relevant Web element's class and html tag properties. Use this information to select the object on which you want to Spy.
  - When you click a Web object in Safari, the Remote Object Spy captures its properties and hierarchy, and displays the information in UFT.
4. **In UFT: In the Remote Object Spy dialog box, view object properties, operations and hierarchy, highlight the object in the application, or add it to your object repository**
- You can do any of the following:
- Select the various radio buttons and tabs to view the details of the object's test object properties and operations and its native properties and operations.
  - Select other test objects currently displayed in the **Object hierarchy** tree, and view their properties, values, or operations.
  - Click **Highlight in Application**  to highlight the object in Safari on the Mac.

UFT highlights only objects that are currently visible on the Mac computer.
  - Click **Add to Repository**  to add the object currently selected in the **Object hierarchy** tree to the object repository.
  - Click **Copy Identification Properties to Clipboard**  to copy all of the properties and values for the object currently selected in the **Object hierarchy** tree. You can paste the copied data from the Clipboard into any document.

# Troubleshooting and Limitations - Object Spy

## **Relevant for: GUI tests and components**

The Object Spy does not support Insight test objects.

# Chapter 31: Test Objects in Object Repositories

**Relevant for: GUI tests and components**

This chapter includes:

• <a href="#">Object Repository Types - Overview</a> .....	179
• <a href="#">Deciding Whether to Use Local or Shared Object Repositories</a> .....	180
• <a href="#">Local Object Repository - Overview</a> .....	181
• <a href="#">Shared Object Repositories Overview</a> .....	182
• <a href="#">Exporting Local Objects to a Shared Object Repository</a> .....	183
• <a href="#">Local Copies of Objects from Shared Object Repositories</a> .....	183
• <a href="#">Considerations for Working with Shared Object Repositories</a> .....	184
• <a href="#">Object Repository Window - Overview</a> .....	186
• <a href="#">Comparison of Object Repository Window and Object Repository Manager</a> .....	187
• <a href="#">Adding and Deleting Test Objects in a Local or Shared Object Repository</a> .....	188
• <a href="#">Guidelines for Copying, Pasting, and Moving Objects</a> .....	190
• <a href="#">Locating Objects</a> .....	191
• <a href="#">Maintaining Identification Properties</a> .....	192
• <a href="#">Working with Repository Parameters</a> .....	195
• <a href="#">Repository Parameter Value Mappings</a> .....	196
• <a href="#">Working with Test Objects During a Run Session</a> .....	197
• <a href="#">Managing Shared Object Repositories Using Automation</a> .....	198
• <a href="#">How to Add a Test Object to an Object Repository</a> .....	199
• <a href="#">How to Maintain Test Objects in Object Repositories</a> .....	202
• <a href="#">How to Create and Manage Shared Object Repositories</a> .....	206
• <a href="#">How to Locate an Object in an Object Repository</a> .....	209
• <a href="#">How to Define a Visual Relation Identifier for a Specific Test Object - Use-Case Scenario</a> .....	210
• <a href="#">Object Repository Window</a> .....	215
• <a href="#">Object Repository Manager Main Window</a> .....	222
• <a href="#">Troubleshooting and Limitations - Object Repositories</a> .....	226

# Object Repository Types - Overview

## Relevant for: GUI actions and components

Objects can be stored in two types of object repositories—a shared object repository and a local object repository.

A **shared object repository** stores objects in a file that can be accessed by multiple tests or components (via their application areas) (in read-only mode). You can use the same shared object repository for multiple actions or components. You can also use multiple object repositories for each action or component.

A **local object repository** stores objects in a file that is associated with one specific action or component, so that only that action or component can access the stored objects.

## Planning where to store objects

When you plan and create tests or components, you must consider how you want to store their test objects. You can:

- Store the objects for each action or component in its corresponding local object repository.
- Store the objects in one or more shared object repositories. By storing objects in shared object repositories and associating these repositories with your actions or component's application areas, you enable multiple actions and components to use the objects. Use a combination of objects from your local and shared object repositories, according to your needs.
- Transfer local objects to a shared object repository, if required. This reduces maintenance and enhances the reusability of your tests and components because it enables you to maintain the objects in a single, shared location instead of multiple locations. For details, see "[Deciding Whether to Use Local or Shared Object Repositories](#)" on the next page.

**Note:** If you want to use a shared object repository from ALM, you must save the shared object repository in the Test Resources module in your ALM project before you associate the object repository using the Associated Repositories tab of the Action Properties dialog box or the Associate Repositories dialog box.

You can save the shared object repository to your ALM project using the Object Repository Manager (as long as the Object Repository Manager is connected to your ALM project).

Tests or components always use the object repositories that are specified in the Associated Repositories tab of the Action Properties dialog box or in the Associate Repositories dialog box of the application area with which the component is associated. Shared object repositories are read-only when accessed from tests or components; you edit them using the Object Repository Manager.

**Note:** Note the following, however, if you have an object with the same name in multiple associated repositories:

- If an object with the same name is located in both the local object repository and in a shared object repository associated with the same action or component, the local object definition is used.
- If more than one shared object repository is associated with the same action or component, the object definition is used from the first occurrence of the object, according to the order in which the shared object repositories are associated with the action or component.

You perform many object repository-related tasks either in the Object Repository window or in the Object Repository Manager. To view a list of the tasks that you can perform in each, see "[Comparison of Object Repository Window and Object Repository Manager](#)" on page 187.

### Note for users of previous QuickTest versions

If you open a test stored in the file system that was created using QuickTest 9.0 or earlier, the object repository associations are changed as follows:

- If the test previously used per-action repositories, the objects in each **per-action repository** are transferred to the **local object repository** of each action in the test.
- If the whole test previously used one shared object repository, the same shared object repository is associated with each of the actions in the test, and the action's local object repositories are empty.

If the test is opened in read-only mode, these changes are not saved.

## Deciding Whether to Use Local or Shared Object Repositories

### Relevant for: GUI actions and components

To choose where to save objects, you need to understand the differences between local and shared object repositories.

The following table describes when it is preferable to use each type of object repository:

Use this object repository type...	In these situations...
<b>local object repository</b>	<ul style="list-style-type: none"><li>• You are creating single-action tests.</li><li>• You are creating simple tests or components, especially under the following conditions:<ul style="list-style-type: none"><li>• You have only one, or very few, tests or components that correspond to a given application, interface, or set of objects.</li><li>• You do not expect to frequently modify object properties.</li><li>• You are new to using UFT. You can record and run tests or components without creating, choosing, or</li></ul></li></ul>

Use this object repository type...	In these situations...
	<p>modifying shared object repositories because all objects are automatically saved in a local object repository that can be accessed by its corresponding action or component.</p> <p><b>See also:</b> "<a href="#">Local Object Repository - Overview</a>" below</p>
<b>shared object repository</b>	<ul style="list-style-type: none"> <li>• You are creating tests or components using keyword-driven methodologies (not by recording).</li> <li>• You have several tests or components that test elements of the same application, interface, or set of objects.</li> <li>• You often work with multi-action tests and regularly use the <b>Insert Copy of Action</b> and <b>Insert Call to Action</b> options.</li> <li>• You expect the object properties in your application to change from time to time and/or you regularly need to update or modify object properties.</li> <li>• If you are familiar with testing, it is probably most efficient to save objects in a shared object repository. In this way, you can use the same shared object repository for multiple actions or components—if they use the same objects.</li> </ul> <p>Object information that applies to many actions or components is kept in one central location. When the objects in your application change, you can update them in one location for all the actions and components that use this shared object repository.</p>

## Local Object Repository - Overview

### Relevant for: GUI actions and components

When you use a local object repository, UFT uses a separate object repository for each action or component. (You can also use one or more shared object repositories if needed. For details, see "[Shared Object Repositories Overview](#)" on the next page.) The local object repository is fully editable from within its action or component.

When working with a local object repository:

- UFT creates a new (empty) object repository for each action or component.
- When UFT learns new objects (either because you add them to the local object repository, or you record operations on objects in your application), it automatically stores the information about those objects in the corresponding local object repository (if the test objects do not already exist in an associated shared object repository).

UFT adds all new objects to the local object repository even if one or more shared object repositories are already associated with the action or component. (This assumes that a test object with the same description does not already exist in one of the associated shared object repositories).

- If a child object is added to a local object repository, and its parents are in a shared object repository, its parents are automatically added to the local object repository.
- Every time you create a new action or component, UFT creates a new, corresponding local object

repository and adds test objects to the repository as it learns them.

- If UFT learns the same object in your application in two different actions or components, the test object is stored as a separate test object in each of the local object repositories.
- When you save your test or component, its local object repositories (or repository, respectively) are automatically saved with the test or component. The local object repository is not accessible as a separate file (unlike the shared object repository).

## Shared Object Repositories Overview

### Relevant for: GUI tests and components

A shared object repository contains information that enables UFT to identify the objects in your application. UFT enables you to maintain the reusability of your tests or components by storing all the information regarding your test objects in shared object repositories.

You use the Object Repository Manager to create and maintain shared object repositories. You can work with shared object repositories saved both in the file system and in an ALM project.

When objects in your application change, the Object Repository Manager provides a single, central location in which you can update test object information for multiple tests or components.

### Advantages of Shared Object Repositories

- You can use the same shared object repository with multiple actions or components, instead of saving objects directly with an action or a component in a local object repository. This enables them to be accessed from multiple actions or components.
- You can use multiple shared object repositories with each action or component.
- If your shared object repositories are stored in ALM, you can apply version control to them. For details, see "[Version Control in ALM](#)" on page 951.

### Importing and Exporting Shared Object Repositories Using XML

You can import and export shared object repositories from and to XML files. XML provides a structured, accessible format that enables you to make changes to shared object repositories using the XML editor of your choice and then import them back into UFT.

You can view the required format for the shared object repository in the *HP Unified Functional Testing Object Repository Schema Help* ([Help > HP UFT GUI Testing Automation and Schema References > Object Repository Schema Reference](#)), or by exporting a saved shared object repository.

You can import and export files from and to the file system or an ALM project (if UFT is connected to ALM).

For details, see:

- ["Import a shared object repository from XML" on page 208](#)
- ["Export a shared object repository to XML" on page 208](#)

## Exporting Local Objects to a Shared Object Repository

### **Relevant for: GUI tests and components**

You can export all of the test objects, checkpoint objects, and output value objects contained in an action's or component's local object repository to a shared object repository in the file system, or to an ALM project (if UFT is connected to ALM). This enables you to make the local objects accessible to other actions or components.

When you export local objects to a shared object repository, the parameters of any parameterized objects are converted to repository parameters, using the same name as the source parameter. The default (mapped) value of each repository parameter is the corresponding source parameter. You can modify the mapping used within your action or component using the Map Repository Parameters Dialog Box. For details on repository parameters, see ["Working with Repository Parameters" on page 195](#).

**Tip:** After you export the local objects, you can use the Object Repository Merge Tool to merge the test objects from the shared object repository containing the exported objects with another shared object repository.

**For actions:** You can choose to export only the local objects to a shared object repository, or to export and replace the local objects. The **Export and Replace Local Objects** option exports the local objects to a shared object repository, associates the new shared object repository with your action, and then deletes the objects in the local object repository.

**For components:** The **Export and Replace Local Objects** option (**File > Export and Replace Local Objects**) is not available.

## Local Copies of Objects from Shared Object Repositories

**Note:** Available from the Object Repository window only.

### **Relevant for: GUI tests and components**

You can create a local copy of any object stored in a shared object repository that is associated with the action or component currently displayed in the in the object repository tree.

Copying an object to the local repository is useful, for example, if you want to modify an object in the current action or component, without affecting other actions or components that use the shared object repository.

When you create a local copy of an object and modify it in the Object Repository window, the changes you make affect only the action or component in which you make the change. Conversely, if you modify

the object in the shared object repository using the Object Repository Manager, the changes you make are reflected in all actions or components that use the shared object repository. However, if you modify an object in a shared object repository, and a copy of the object (with the same name) exists in the local repository, your changes do not affect the local copy of the object in your action or component.

During a run session, UFT uses the test object in the local object repository to identify the object in your application. This is because the local object repository has higher priority than any shared object repository associated with the action or component.

### Considerations for Copying an Object to the Local Object Repository

- When you copy an object to the local object repository, its parent objects are also copied to the local object repository.
- You cannot copy an object to the local object repository if an object or its parent objects use unmapped repository parameters. You must make sure that all repository parameters are mapped before copying an object to the local object repository.
- If an object or its parent objects are parameterized using one or more repository parameters, the repository parameter values are converted when you copy the object to the local object repository. If the value is a constant value, the property receives the same constant value.
- If you are copying multiple objects to the local object repository, during the copy process you can choose to skip a specific object if it has unmapped repository parameters, or if it has mapped repository parameters whose values you do not want to convert. You can then continue copying the next object from your original selection.

## Considerations for Working with Shared Object Repositories

### Relevant for: GUI tests and components

Consider the following when working with shared object repositories:

### Opening, Modifying, and Saving Shared Object Repositories

- **Enabling editing.** If you opened the shared object repository in read-only mode, you must enable editing for the shared object repository before you can modify it. This locks the shared object repository and prevents it from being modified simultaneously by multiple users. For details on enabling editing, see ["How to Create and Manage Shared Object Repositories" on page 206](#).
- **Unlocking.** When you enable editing for a shared object repository, the shared object repository is locked so that it cannot be modified by other users. To enable other users to modify the shared object repository, you must first unlock it (by disabling edit mode, or by closing it). If a shared object repository is already locked by another user, if it is saved in read-only format, or if you do not have the permissions required to open it, you cannot enable editing for it.

- **Applying changes.** All changes you make to a shared object repository are automatically updated in all tests or components open on the same computer that use the shared object repository as soon as you make the change—even if you have not yet saved the shared object repository with your changes.  
If you close the shared object repository without saving your changes, the changes are rolled back in any open tests or components that were open at the time.
- **Updating changes.** When you open a test or component on the same computer on which you modified the shared object repository, the test or component is automatically updated with all saved changes made in the associated shared object repository. To see saved changes in a test or component or repository open on a different computer, you must open the test or component or shared object repository file or lock it for editing on your computer to load the changes.
- **Merging.** You can modify a shared object repository by merging it with another shared object repository. When you merge two shared object repositories, a new shared object repository is created, containing the content of both shared object repositories. If you merge a shared object repository with a local object repository, the shared object repository is updated with the content of the local object repository.
- **Closing a test/solution with an associated repository.** When you close a solution or test that contains an associated open shared object repository, the shared object repository is automatically closed, even if the repository is currently open.

## Managing Objects in Shared Object Repositories

- If one or more of the property values of an object in your application differ from the property values UFT uses to identify the object, your test or component may fail. Therefore, when the property values of objects in your application change, you should modify the corresponding identification property values in the corresponding object repository so that you can continue to use your existing tests or components.
- The following table describes UFT behavior in cases of duplicate objects in shared object repositories:

If...	UFT Uses...
An object with the same name and description is located in both the local object repository and in a shared object repository that is associated with the same action or component.	The local object definition.
An object with the same name and description is located in more than one shared object repository, and these shared object repositories are all associated with the same action or component.	The object definition from the first occurrence of the object, according to the order in which the shared object repositories are associated with the action or component.

## General Tips and Guidelines

- If your test or component contains test objects from environments that are not installed/loaded with UFT, the test objects are displayed with a question mark in the shared object repository.
- When you modify a shared object repository, an asterisk (\*) is displayed in the title bar until the

object repository is saved.

- You can use the **Edit > Undo** and **Edit > Redo** options to cancel or repeat your changes as necessary. The **Undo** and **Redo** options are related to the active document. When you save a shared object repository, you cannot undo and redo operations that were performed on that file before the save operation.
- You perform search operations in the shared object repository the same way you perform them in the local object repository. For details, see "[Locating Objects](#)" on page 191.

## Object Repository Window – Overview

### **Relevant for: GUI tests and components**

The Object Repository window displays a tree of all test objects, and all checkpoint and output objects in the current action or component, including all local objects and all objects in associated shared object repositories.

For each object you select in the tree, the Object Repository window displays information on the object, including its type, the repository in which it is stored, and its object details. Local objects are editable (black); shared objects are read-only (gray).

You can continue using UFT while the Object Repository window is open, including modifying objects and object repositories. The Object Repository window reflects changes you make to it in real time. For example, if you add objects to the local object repository, or if you associate an additional object repository with the current action or component, the Object Repository window immediately displays the updated content.

**Note:** You modify objects in a shared object repository using the Object Repository Manager.

You can also modify an object in a shared object repository by copying it to the local object repository and then modifying the local copy. For details, see "[Local Copies of Objects from Shared Object Repositories](#)" on page 183.

## Comparison of Object Repository Window and Object Repository Manager

### **Relevant for: GUI tests and components**

You perform many object repository-related tasks either in the Object Repository window or in the Object Repository Manager. Some object repository-related tasks can also be performed in both. The following table lists features and functionality, indicating if they are available in the Object Repository window or the Object Repository Manager:

Functionality	Object Repository window	Object Repository Manager
"Adding and Deleting Test Objects in a Local or Shared Object Repository" on the next page	✓	✓
"Highlighting an Object in Your Application" on page 191	✓	✓
"Locating a Test Object in the Object Repository" on page 192	✓	✓
"Maintaining Identification Properties" on page 192	✓	✓
"Maintaining Identification Properties" on page 192	✓	✓
"Maintaining Identification Properties" on page 192	✓	✓
"Maintaining Identification Properties" on page 192	✓	✓
"Maintaining Identification Properties" on page 192	✓	✓
"Removing Properties from a Test Object Description" on page 204	✓	✓
"Exporting Local Objects to a Shared Object Repository" on page 183	✓	✗
"Managing Shared Object Repositories" on page 207 (includes creating, opening, saving, closing, and enabling editing functionality)	✗	✓
"Adding a Test Object to an Object Repository" on page 199	✓	✓
"Adding Test Objects Using the Navigate and Learn Option" on page 208	✗	✓
"Working with Repository Parameters" on page 195	✗	✓
"Merging Two Shared Object Repositories" on page 252	✗	✓
"To export to XML:" on page 208	✗	✓

# Adding and Deleting Test Objects in a Local or Shared Object Repository

## Available from:

- Object Repository window for the local object repository
- Object Repository Manager for shared object repositories

## Relevant for: GUI tests and components

When you create an object repository for your keyword-driven testing infrastructure, you can add test objects to it in various ways, for example:

- You can use the **Navigate and Learn** option to add objects to a shared object repository according to your defined filter.
- You can record an action or component to add each object on which you perform an operation to the local object repository. (This occurs for objects that do not already exist in an associated shared object repository.)
- You can add test objects to the local object repository while editing your test or component. For example, for tests and scripted components, you can add tests objects from the Active Screen.

When you add test objects to an object repository, you can choose to add only a selected test object, to add all test objects of a certain type (such as all button objects), or to add all test objects of a specific class (such as all **WebButton** objects).

For example, you may find that users need to perform a step on an object that is not in the object repository. You may also find that an additional object was added to the application you are testing after you built the object repository. You can add the object directly to a shared object repository using the Object Repository Manager, so that it is available in all actions and components that use this shared object repository. Alternatively, you can add it to the local object repository of the action or component.

You can add test objects to a local or shared object repository directly from your application. You can choose to add a specific test object either with or without its descendants. You can also control which descendants to add, according to their object and class types, based on selections that you define in the object filter.

If needed, you can merge test objects from the local object repository into a shared object repository.

You can also add test objects to a shared object repository while navigating through your application.

This section also includes:

- "[Adding Test Objects to the Local Object Repository from the Active Screen \(GUI actions and scripted components only\)](#)" on the next page
- "[Defining New Test Objects](#)" on the next page
- "[Deleting Objects in the Object Repository](#)" on the next page

## Adding Test Objects to the Local Object Repository from the Active Screen (GUI actions and scripted components only)

You can add test objects to the local object repository of the current action by selecting the required object in the Active Screen.

To add test objects to the object repository using the Active Screen, the Active Screen must contain information for the object you want to add. You can control how much information is captured in the Active Screen in the Active Screen node of the Options dialog box (**Tools > Options > GUI Testing tab > Active Screen node**).

When you add a test object to the object repository in one of the ways described in this section, the test object is added to the local object repository and can be used only by the current action or scripted component. If you want to add the test object to the shared object repository so that it can be used in multiple actions or scripted components, add it using the Object Repository Manager (not from the Active Screen).

## Defining New Test Objects

You can define test objects in your object repository that do not yet exist in your application. This enables you to prepare an object repository and create actions or components for your application before the application is ready for testing.

For example, you may already know the names, types, and descriptive properties of some of the objects in your application, and know only the types of other objects in your application. Before your application is ready, you can create WebEdit objects for UserName and Password fields in your Login page (plus the relevant parent Page and Browser objects). If you know the property values for these objects, you can also add them. If not, you can add them when your application is ready for testing.

When you manually define a new object in the object repository, the object is added to the local object repository and can be used only by the current action or component. If you want to add the object to the shared object repository so that it can be used in multiple actions or components, you must add it using the Object Repository Manager.

After you define a new test object, you can modify the test object description at any time. For example, you may need to update the test object description if the properties of the object in your application do not match the test object description that you defined, or if an object was updated in your application. For details, see "[Maintaining Identification Properties](#)" on page 192.

## Deleting Objects in the Object Repository

When you remove a step from your action or component, its corresponding test object remains in the object repository. You can delete the test object if it is not in use elsewhere.

For example, if you are working with a local object repository, make sure that the test object is not used in any other steps within that action or component. If you are working with a shared object repository, confirm that the test object is not used in any other action or component using that shared object

repository. Also, confirm that the test object you are deleting is not used in a visual relation identifier for another test object.

You delete objects in the local object repository using the Object Repository window, and objects in the shared object repository using the Object Repository Manager.

**Note:** If your action or component contains references to an object that you deleted from the object repository, your run session will fail.

## Guidelines for Copying, Pasting, and Moving Objects

### Relevant for: GUI tests and components

When copying, pasting, or moving objects, consider the following guidelines:

- You cannot modify the root node of an object repository.
- If you change the object hierarchy, ensure that the new hierarchy is valid.
- If you paste or move an object to a different hierarchical level, you can choose whether to copy all objects up to the shared parent object (in the message displayed when you perform such an operation).
- In the Object Repository window, when you copy, paste, and move objects from a shared object repository associated with an action or a component, the objects are copied, pasted, or moved to the local object repository of the action or component.
- If you move an object to its immediate parent, UFT creates a copy of the object (renamed with an incremental suffix) and pastes it as a sibling of the original object.
- If you cut or copy an object, and then paste it on its parent object, UFT creates copy of the object (renamed with an incremental suffix) and inserts it at the same level as the original object.
- You cannot move an object to any of its descendants.
- You cannot copy or move an object to be a child of a bottom-level object (an object that cannot contain a child object) in the object hierarchy.
- You cannot copy, paste, or move objects that have unmapped repository parameters from a shared object repository to the local object repository.
- If you copy, paste, or move an object from a shared object repository to the local object repository, and the object or one of its parent objects are parameterized using repository parameters, the repository parameter values are converted when you copy, paste, or move the object. If the value is a constant value, the property receives the same constant value.
- If you delete a test object that are used in a visual relation identifier for another test object, object identification for that test object will fail.

# Locating Objects

**Available from:**

- Object Repository window for the local object repository
- Object Repository Manager for shared object repositories

**Relevant for: GUI tests and components**

You can search for a specific object in your object repository in several ways. You can search for an object according to its type. For example, you can search for a specific edit box, or you can point to an object in your application to automatically highlight that same object in your repository. You can select an object in your object repository and highlight it in your application to check which object it is. For local objects (and shared objects in an editable shared object repository when using the Object Repository Manager), you can also replace specific property values with other property values. For example, you can replace the property value `userName` with `user name`.

For task details, see "[How to Locate an Object in an Object Repository](#)" on page 209.

**Note:** When locating test objects, UFT also takes into account the visual relation identifier property, if defined for that test object. For details, see "[Visual Relation Identifiers](#)" on page 234.

This section also includes:

- "[Finding Objects in an Object Repository](#)" below
- "[Highlighting an Object in Your Application](#)" below
- "[Locating a Test Object in the Object Repository](#)" on the next page

## Finding Objects in an Object Repository

You can use the Find and Replace dialog box to find an object, property, or property value in an object repository. You can also find and replace specified property values.

You replace property values for objects in the local object repository using the Object Repository window. You replace property values for objects in shared object repositories using the Object Repository Manager.

## Highlighting an Object in Your Application

You can select a test object in your object repository and highlight it in the application you are testing. When you choose to highlight a test object, UFT indicates the selected object's location in your application by temporarily showing a frame around the object and causing it to flash briefly. The application must be open to the correct context so that the object is visible.

For example, to locate the User Name edit box in a Web page, you can open the relevant page in the Web browser and then select the **User Name** test object in the object repository. When you select the **Highlight in Application** option, the User Name edit box in your browser is framed in the Web page and flashes several times. Both the frame and the flashing behavior are temporary.

### Locating a Test Object in the Object Repository

You can select an object in the application you are testing and highlight the test object in the object repository.

For example, to locate a Find a Flight image in a Web page, you can select the **Locate in Repository** option, and then select the image in your Web page using the pointing hand mechanism. After you select the Find a Flight image object from the selection dialog box and click **OK**, the parent hierarchy in the object repository tree expands and the Find a Flight image test object is highlighted.

## Maintaining Identification Properties

### Available from:

- Object Repository window for the local object repository
- Object Repository Manager for shared object repositories

### Relevant for: GUI tests and components

As applications change, you may need to change the property values of the steps in your action or component. Suppose an object in your application is modified. If that object is part of your action or component, you should modify its values so that UFT can continue to identify it. For example, if a company Web site contains a **Contact Us** hypertext link, and the text string in this link is changed to **Contact My Company**, you need to update the object's details in the object repository so that UFT can continue to identify the link properly (assuming that the text property is included in the test object's description).

If you are using an Insight object and the text is included in the test object image, you might need to update the test object so that UFT can continue to identify it. You can modify the test object's image to include the updated text, or you can add a **similarity** identification property to the test object description, or lower the property's value, to enable UFT to match the test object with the object in the application despite the differences in the text.

You can modify identification properties in a number of ways. For an object stored in a local object repository, you can modify its properties directly from the Object Repository window. For an object stored in a shared object repository, you can either open it in the Object Repository Manager and modify its properties, or you can copy it to the local object repository and then modify its properties.

There are a number of things you can do to maintain your object's properties:

<b>Specify or</b>	You can specify or modify values for properties in the test object description. You can specify a value using a constant value (either a simple value or a constant value that includes regular expressions) or you can
-------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<b>modify property values</b>	<p>parameterize it. You can also change the set of properties used to identify that object.</p> <p>You can also automatically update the description of one or more test objects in your object repository based on the actual updated object properties in your application. For details, see "<a href="#">Maintaining Identification Properties</a>" on the previous page.</p> <p>You can also find and replace specific identification property values.</p> <p><b>Note:</b> In some cases, the Smart Identification mechanism may enable UFT to identify a test object, even when some of its property values change. However, if you know about property value changes for a specific test object, you should try to correct the test object definition so that UFT can identify the test object from its basic object description. For details on the Smart Identification mechanism, see "<a href="#">Configuring Object Identification</a>" on page 227.</p>
<b>Update identification properties from an object in Your application</b>	<p>You can update a test object in your object repository by selecting the corresponding object in your application and relearning its properties and property values from the application. When you update a test object description in this way, all currently defined properties and values are overwritten, including description properties and values, the ordinal identifier, and Smart Identification information. An Insight test object's image is not updated.</p> <p>The updated object description is based on the current definitions in the Object Identification dialog box. Only the object-specific comments, if any, are retained.</p> <p>This is useful if an object's properties have changed since you added it to the object repository, since UFT may not be able to recognize the object unless you update its description.</p> <p>You can also use this option to update an object that you defined (using the <b>Object &gt; Define New Test Object</b> option) before the application was completely developed, and as a result some of the identification properties and values are missing in the test object description, or are no longer sufficient to identify the object.</p> <p><b>Note:</b> If you just want to restore the original test object description property set, while retaining any property values you have modified, you can use the <b>Restore mandatory property set</b> option. For details, see "<a href="#">Maintaining Identification Properties</a>" on the previous page.</p>
<b>Restore default mandatory properties for a test object</b>	<p>You can restore the default properties for a selected test object. When you restore the default properties, it restores the mandatory property set defined for the selected object class, based on the settings that were set in the Object Identification dialog box at the time the object was learned. If you added or removed properties to or from the description, those changes are overwritten. However, if property values were defined or modified for any of the mandatory properties, those values are not modified when you choose this option. In addition, restoring the default mandatory property set does not change the values for the ordinal identifier or Smart Identification settings for the test object.</p> <p>You can use the <b>Restore mandatory property set</b> option to restore the object description property set to the mandatory properties that were defined for that class when your object was learned. If the mandatory properties in the Object Identification dialog box are currently different for this test object class than they were when your object was learned, and you want to use the new definition, you can use the <b>Update From Application</b> option, which relearns the object properties and values based on the current definitions in the Object Identifications dialog box. For more details, see "<a href="#">Maintaining Identification Properties</a>" on the previous page.</p>
<b>Rename test objects</b>	<p>When an object changes in your application, or if you are not satisfied with the current name of a test object for any reason, you can change the name that UFT assigns to the stored object. You can also provide test objects with meaningful names to assist users in identifying them when using them in steps.</p> <p>For example, suppose you have a graphics application in which all the tools in the toolbar are saved as</p>

	<p>WinObjects in the object repository with the names ToolChild1, ToolChild2, ToolChild3, and so forth. You may want to rename all the buttons to their actual labels to make them easier for you to identify in the test, for example, Color_Picker, Eraser, Airbrush, and so forth.</p> <p>Renaming a test object does not affect the way UFT recognizes the object in your application, as the test object name is not included in the test object description.</p> <p>If you are working with a shared object repository, your change applies to all occurrences of the test object in all actions and components that use this shared object repository.</p> <p>If you are working with a local object repository, your change applies to all occurrences of the test object in the selected action or component. If other actions or components also include operations on the local test object, you should modify the test object's name in each relevant action or component.</p> <p>When you modify the name of a test object in the local object repository, the name is automatically updated for all occurrences of the test object in the action or component. When you modify the name of a test object in a shared repository, the name is automatically updated in all tests and components open on the same computer that use the object repository as soon as you make the change, even if you have not yet saved the object repository with your changes. If you close the object repository without saving your changes, the changes are rolled back in any open tests or components that were open at the time. Changes that are saved are also automatically updated in any other tests and components that use the object repository, as soon as you open them. To load and view saved changes in an action, component, or object repository that is currently open on a different computer, you must open the object repository or lock it for editing on your computer.</p> <p><b>Tip:</b> If you do not want to automatically update test object names in the action or component for all occurrences of the test object, you can clear the <b>Automatically update test and component steps when you rename test objects</b> check box in the General pane of the Options dialog box (<b>Tools &gt; Options &gt; GUI Testing tab &gt; General node</b>). If you clear this option, you will need to manually change the test object names in all steps in which they are used, otherwise your run session will fail.</p> <p><b>Note:</b> If you rename test objects in a shared object repository and save the changes, when you open another test or component using the same shared object repository, that test or component updates the test object name in all of its relevant steps. This process may take a few moments. If you save the changes to the second test or component, the renamed steps are saved. However, if you close the second test or component without saving, then the next time you open the same test or component, it will again take a few moments to update the test object names in its steps.</p>
<b>Add properties to a test object description</b>	<p>You can add to the list of properties that UFT uses to identify an object. For each object class, UFT has a default property set that it uses for the object description for a particular test object. You can use the Add Properties dialog box to change the properties that are included in the test object description.</p> <p><b>Note:</b> You can also add any valid identification property to a test object description, even if it does not appear in the Add Properties dialog box.</p> <p>Adding to the list of properties is useful when you want to create and run tests or components on an object that changes dynamically. An object may change dynamically if it is frequently updated, or if its property values are set using dynamic content (for example, from a database).</p> <p>You can also change the properties that identify an object if you want to reference objects using properties that UFT did not learn automatically when it learned the object. For example, suppose you are testing a Web site that contains an archive of newsletters. The archive page includes a hypertext link to the current newsletter and additional hypertext links to all past newsletters.</p> <p>The text in the first hypertext link on the page changes as the current newsletter changes, but it always links to a page called <code>current.html</code>. Suppose you want to create a step in your test or component in which you</p>

	<p>always click the first hypertext link in your archive page. Since the news is always changing, the text in the hypertext link keeps changing. You need to modify how UFT identifies this hypertext link so that it can continue to find it.</p> <p>The default properties for a <b>Link</b> object (hypertext link) are <b>text</b> and <b>HTML tag</b>. The text property is the text inside the link. The HTML tag property is always <b>A</b>, which indicates a link.</p> <p>You can modify the default properties for a hypertext link for the learned object so that UFT can identify it by its destination page, rather than by the text in the link. You can use the <b>href</b> property to check the destination page instead of using the <b>text</b> property to check the link by the text in the link.</p> <p><b>Note:</b> You can also modify the set of properties that UFT learns when it learns objects from a particular object class using the Object Identification dialog box. Such a change generally affects only those objects that UFT learns after you make the change. For details, see "<a href="#">Configuring Object Identification on page 227</a>". You can also apply the changes you make in the Object Identification dialog box to the descriptions of all objects in an existing test or component using the <b>Update Run Mode</b> option.</p>
<b>Define new identification properties</b>	<p>You can add any valid identification property to a test object description, even if it does not appear in the Add Properties dialog box.</p> <p>For example, suppose you want UFT to use a specific property to identify your object, but that property is not listed in the Add Properties dialog box. You can open the Add Properties dialog box and add that property to the list.</p>
<b>Add ordinal identifiers</b>	<p>An ordinal identifier assigns a numerical value to a test object that indicates its order or location relative to other objects with an otherwise identical description (objects that have the same values for all properties). This ordered value provides a backup mechanism that enables UFT to create a unique description to recognize an object when the defined properties are not sufficient to do so. For details on ordinal identifiers, see "<a href="#">"Ordinal Identifiers" on page 230</a>".</p> <p><b>Note:</b> When visual relation identifiers are used, the <b>Ordinal identifier</b> option is disabled in the Object Repository Manager or window. For details on visual relation identifiers, see "<a href="#">"Visual Relation Identifiers" on page 234</a>".</p>

## Working with Repository Parameters

### Relevant for: GUI tests and components

Repository parameters enable you to specify that certain property values should be parameterized, but leave the actual parameterization to be defined in each test or component that is associated with the shared object repository that contains the parameterized identification property values.

Repository parameters are useful when you want to create and run tests and components on an object that changes dynamically. An object may change dynamically if it is frequently updated in the application, or if its property values are set using dynamic content, for example, from a database.

### Example

Suppose you have a button whose text property value changes in a localized application depending

on the language of the user interface. You can parameterize the **name** property value using a repository parameter, and then in each test or component that uses the shared object repository you can specify the location from which the property value should be taken. For example, in one test or component that uses this shared object repository you can specify that the property value comes from an environment variable or a component parameter, respectively. In another test or component it can come from the Data pane or a local parameter, respectively. In a third test or component you can specify it as a constant value.

You define all the repository parameters for a specific shared object repository using the [Manage Repository Parameters Dialog Box](#).

### Considerations for Working with Repository Parameters

- When you delete a repository parameter that is used in a test object definition, the identification property value remains mapped to the parameter, even though the parameter no longer exists. Therefore, before deleting a repository parameter, you should make sure that it is not used in any test object descriptions, otherwise tests or components that have steps using these test objects will fail when you run them.
- When you open a test or component that uses a shared object repository with a repository parameter that has no default value, an indication that there is a repository parameter that needs mapping is displayed in the Errors pane. You can then map the repository parameter as needed in the test or component. You can also map repository parameters that have default values, and change mappings for repository parameters that are already mapped.

## Repository Parameter Value Mappings

### Relevant for: GUI tests and components

You use **repository parameters** to specify that certain property values of an object in a shared object repository should be parameterized, but that the actual values should be defined in each action or component associated with the shared object repository. For details on repository parameters, see "[Working with Repository Parameters](#)" on the previous page.

Mapping a repository parameter to a value or to a parameter specifies the property values used to identify the test object during a run session. You can specify that the property value is taken from a constant value, or parameterize it using any of the following:

<b>Tests</b>	<ul style="list-style-type: none"><li>• Data table</li><li>• random number</li><li>• environment</li><li>• test or action parameter</li></ul>
<b>Components</b>	<ul style="list-style-type: none"><li>• local parameter</li><li>• component parameter</li></ul>

You can map each repository parameter as required in each test or component that has an associated object repository containing repository parameters. For example, you may want to retrieve the username object's text property value from an environment variable parameter for one test or component, and from a constant value, Data pane parameter, or local parameter for another.

Before you map repository parameters, if you have more than one repository parameter with the same name in different shared object repositories that are associated with the same action or component, the repository parameter from the shared object repository with the highest priority (as defined in the shared object repositories list) is used.

After you map repository parameters, UFT uses the mappings you defined. In addition, changing the priority or default values has no effect after the parameters are mapped.

When you open a test or component that uses an object repository with an object property value that is parameterized using a repository parameter with no default value, the Errors pane indicates that there is a repository parameter that needs mapping. You can then map the repository parameter as needed in the test or component. You can also map repository parameters that have default values, and change mappings for repository parameters that are already mapped.

If you do not map a repository parameter, the default value that was defined with the parameter, if any, is used during the run session. If the parameter is unmapped, meaning no default value was specified for it, the test or component run may fail if a test object cannot be identified because it has an unmapped parameter value.

## Working with Test Objects During a Run Session

### **Relevant for: GUI tests and components**

The first time UFT encounters an object during a run session, it creates a temporary version of the test object for that run session. UFT uses the object description to create this temporary version of the object. For the remainder of the run session, UFT refers to the temporary version of the test object rather than to the test object in the object repository. The Object Repository window is read-only during a run session.

## Creating Test Objects During a Run Session

You can use programmatic descriptions to create temporary versions of test objects that represent objects from your application. You can perform operations on those objects without referring to the object repository. For example, suppose an edit box was added to a form on your Web site. You can use a programmatic description to add a statement in a user-defined function or in the Editor (actions only) that enters a value in the new edit box. UFT could then identify the object even though the object was never added to the object repository. For details on programmatic descriptions, see ["Programmatic Descriptions" on page 649](#).

# Managing Shared Object Repositories Using Automation

## Relevant for: GUI tests and components

UFT provides an Object Repository automation object model that enables you to manage UFT shared object repositories and their contents from outside of UFT. The automation object model enables you to use a scripting tool to access UFT shared object repositories via automation.

Just as you use the UFT automation object model to automate your UFT operations, you can use the objects and methods of the Object Repository automation object model to write scripts that manage shared object repositories, instead of performing these operations manually using the Object Repository Manager. For example, you can add, remove, and rename test objects; import from and export to XML; retrieve and copy test objects; and so forth.

After you retrieve a test object, you can manipulate it using the methods and properties available for that test object class. For example, you can use the **GetTOPProperty** and **SetTOPProperty** methods to retrieve and modify its properties. For details on available test object methods and properties, see the **Common Methods and Properties** section in the *HP UFT Object Model Reference for GUI Testing*.

Automation programs are especially useful for performing the same tasks multiple times or on multiple shared object repositories. You can write your automation scripts in any language and development environment that supports automation. For example, you can use VBScript, JavaScript, Visual Basic, Visual C++, or Visual Studio .NET. For general information on controlling UFT using automation, see "[UFT Automation Scripts](#)" on page 740.

## Using the Unified Functional Testing Object Repository Automation Reference

The Unified Functional Testing Object Repository Automation Reference is a Help file that provides detailed descriptions, syntax information, and examples for the objects and methods in the UFT shared object repository automation object model.

The Help topic for each automation object includes a list and description of the methods associated with that object. Method Help topics include detailed description, syntax, return value type, and argument value information.

You can open the *Unified Functional Testing Object Repository Automation Reference* from the main **UFT Help** menu (**Help > HP UFT GUI Testing Automation and Schema References > Object Repository Automation Reference**).

**Note:** The syntax and examples in the Help file are written in VBScript-style. If you are writing your automation program in another language, the syntax for some methods may differ slightly from what you find in the corresponding Help topic. For details on syntax for the language you are using, see the documentation included with your development environment or to general documentation for the programming language.

# How to Add a Test Object to an Object Repository

## Relevant for: GUI tests and components

This task describes how to add test objects to local or shared object repositories. This functionality is available in the Object Repository window for the local object repository, and the Object Repository Manager for shared object repositories.

This section includes:

- ["Add test objects to the object repository using the Add Objects to Local or Add Objects option" below](#)
- ["Add an Insight test object to the object repository using the Add Insight Object or Add Insight Object to Local button" on the next page](#)
- ["Add a test object to the local object repository while adding a step to your test or component" on the next page](#)
- ["Define a new test object" on page 201](#)
- ["Add a test object to the object repository using the Object Spy dialog box" on page 201](#)
- ["Add a test object to the local object repository using the View/Add Object option from the Active Screen \(tests only\)" on page 201](#)
- ["Add a test object to the local object repository by inserting a step from the Active Screen \(tests only\) " on page 201](#)

### Note:

- You can add a test object to the local object repository only if that test object does not already exist in a shared object repository that is associated with the action or component. If a test object already exists in an associated shared object repository, you can add it to the local object repository using the **Copy to Local** option. For details, see ["Local Copies of Objects from Shared Object Repositories " on page 183](#).
- You cannot add **WinMenu** objects directly to an object repository using the **Add Objects to Local** button in the Object Repository window or the **Add Objects** button in the Object Repository Manager. If you want to add a **WinMenu** object to the object repository, you can use the **Add Objects** or **Add Objects to Local** button to add its parent object and then select to add the parent object together with its descendants, or you can record a step on a **WinMenu** object and then delete the recorded step.

## [Add test objects to the object repository using the Add Objects to Local or Add Objects option](#)

1. Perform one of the following:

- To add the test object to the *local* repository, making it available only in the current action or component:

In the Object Repository window, Select **Object > Add Objects to Local** or click the **Add Objects to Local** toolbar button .

- To add the test object to a *shared* repository, making it available for multiple actions or components:

In the Object Repository Manager, select **Object > Add Objects** or click the **Add Objects** toolbar button .

UFT and the Object Repository window or Object Repository Manager are hidden, and the pointer changes into a pointing hand. In some environments, as you move the pointing hand over your application, the test objects are highlighted.

2. Click the object you want to add to your object repository.
3. If the location you click is associated with more than one object, the Object Selection dialog box opens. Select the object you want to add to the repository and click **OK**.  
If the object you select in the Object Selection dialog box is a bottom-level object in the test object hierarchy, for example, a **WebButton** object, it is added directly to the object repository.  
If the object you select in the Object Selection dialog box is a parent (container) object, such as a browser or page in a Web environment, or a dialog box in a standard Windows application, the Define Object Filter dialog box opens. The Define Object Filter dialog box retains the settings that you defined in the previous add object session.  
The new object's parent objects are also added, if they do not already exist in the object repository. Local objects are shown in black in the object repository tree to indicate they are editable; shared objects are shown in gray and can be edited only in the Object Repository Manager.

### Add an Insight test object to the object repository using the Add Insight Object or Add Insight Object to Local button

For details, see "[Add an Insight object](#)" on page 269.

### Add a test object to the local object repository while adding a step to your test or component

You can add a test object to the local object repository by choosing it from your application when you add or edit a step in your test or component.

You do this in the Select Test Object Dialog Box, which you can open when selecting an item for a step in the Keyword View or from the Step Generator.

## Define a new test object

1. Select the object under which you want to define the new object, according to the correct object hierarchy.
2. Click the **Define New Test Object** button  or select **Object > Define New Test Object**. The Define New Test Object Dialog Box opens.

## Add a test object to the object repository using the Object Spy dialog box

1. Click the **Object Spy** button  from UFT or the Object Repository Manager.
2. Click the **Add Object** button  . Depending on from where you opened the Object Spy dialog box, the object is added to the local or shared object repository.

## Add a test object to the local object repository using the View/Add Object option from the Active Screen (tests only)

1. If the Active Screen is not displayed, select **View > Active Screen**.
2. Select a step in your test whose Active Screen contains the object that you want to add to the object repository.
3. In the Active Screen, right-click the object you want to add and select **View/Add Object**.
4. If the location you clicked is associated with more than one object, the Object Selection dialog box opens. Select the object you want to add to the object repository, and click **OK** to close the Object Selection dialog box.
5. The Object Properties dialog box opens and displays the default identification properties for the object.

## Add a test object to the local object repository by inserting a step from the Active Screen (tests only)

1. If the Active Screen is not displayed, select **View > Active Screen**.
2. Select a step in your test whose Active Screen contains the object for which you want to add a step.
3. In the Active Screen, right-click the object for which you want to add a step and select the type of step you want to insert (checkpoint, output value, Step Generator, and so forth).
4. If the location you clicked is associated with more than one object, the Object Selection dialog box opens. Select the object for which you want to add a step, and click **OK**.  
The appropriate dialog box opens, enabling you to configure your preferences for the step you want to insert.
5. Set your preferences and select whether to insert the step before or after the step currently selected in the Keyword View or in the Editor. Click **OK** to close the dialog box. A new step is inserted in your test, and the object is added to the local object repository for the current action (if

it was not yet included).

## How to Maintain Test Objects in Object Repositories

### Relevant for: GUI tests and components

The following steps describe different options for maintaining and modifying the test object details of objects in your repositories.

- "Specify an identification property value" below
- "How to Maintain Test Objects in Object Repositories" above
- "Update identification properties from an object in your application" on the next page
- "Restore the mandatory property set" on the next page
- "Rename test objects" on the next page
- "Add properties to a test object description" on page 204
- "Define a new identification property" on page 204
- "Remove properties from a test object description" on page 204
- "Specify an ordinal identifier" on page 204
- "Define related objects for a specific test object" on page 205
- "Export the objects from a local object repository" on page 205
- "Copy an object to the local object repository" on page 206
- "Modify identification properties during a run session" on page 206

### Specify an identification property value

1. In the Object Repository window or the Object Repository Manager, select the test object whose property value you want to specify.

#### Tip:

- For a test object in the local object repository, you can also right-click the step containing the test object and select **Object Properties**, and then make the following property value changes in the Object Properties dialog box.
- If you want to view all objects in the action or component, click the **View in Repository** button. The Object Repository window opens and displays all objects stored in the repository in a repository tree.
- You can also open the object repository for the selected action or component by choosing **Tools > Object Repository Window** or by clicking the **Object Repository** toolbar button



2. In the **Test object details** area, click in the value cell for the required property.
3. Specify the property value in one of the following ways:
  - If you want to specify a constant value, enter it in the value cell.
  - If you want to parameterize the value or specify a constant value using a regular expression, click the parameterization button in the value cell . If you specify a constant value using a regular expression, the  icon is displayed next to the value.  
If you specified a constant value, it is shown in the **Value** column of the **Test object details** area. If you parameterized the value, the parameter name is shown with one of the following icons in the **Value** column.

### Update identification properties from an object in your application

1. In the object repository tree, select the test object whose description you want to update. (You cannot use this process to update the test object image of an Insight test object.)
2. Select **Object > Update from Application** or click the **Update from Application** button . UFT is hidden, and the pointer changes into a pointing hand. In some environments, as you move the pointing hand over your application, the test objects are highlighted.
3. Find the object in your application whose properties you want to update in the object repository and click it. You must choose an object of the same object class as the test object you selected in the object repository tree.

The properties and property values for the selected object are updated in the object repository, according to the properties and values required to identify the object that were learned by UFT when you clicked the object in your application. Note that all properties and property values in the **Test object details** area are updated, together with the ordinal identifier and Smart Identification selections. Any object-specific comments that you may have entered are not removed.

### Restore the mandatory property set

1. In the object repository tree, select the test object whose description you want to restore.
2. In the **Test object details** area, click the **Restore mandatory property set** button .
3. Click **Yes** to confirm the operation. The test object's description properties are restored to the mandatory property set for the selected object class at the time that the object was learned.

### Rename test objects

1. In the object repository tree of the Object Repository window or Manager, select the test object that you want to rename.
2. In the **Name** box in the Object Properties pane, enter the new name for the test object. Then click anywhere else to remove the focus from the object. For a list of naming conventions, see the **Test object name** section in "Troubleshooting - Naming Conventions" on page 1144. Test object names are not case-sensitive.

## Add properties to a test object description

1. In the object repository tree of the Object Repository window or Manager, select the test object whose description you want to modify.
2. In the **Test object details** area, click the **Add description properties** button .
3. The Add Properties dialog box opens listing the properties that can be used to identify the object (properties that are not already part of the test object description).

**Tip:** For a test object in the local object repository, you can also select the required test object and select **Edit > Step Properties > Object Properties**, click the **Add description properties** button , and then perform the following steps in the Add Properties dialog box.

## Define a new identification property

1. In the object repository tree of the Object Repository window or Manager, select the test object for which you want to define a new property.
2. In the **Test object details** area, click the **Add description properties** button . The Add Properties dialog box opens.

**Tip:** For a test object in the local object repository, you can also select the required test object, right-click on the object and select **Object Properties**, click the **Add description properties** button , and then perform the following steps in the Add Properties dialog box.
3. Click the **Define new property** button . The New Property dialog box opens.

## Remove properties from a test object description

1. In the object repository tree of the Object Repository window or Manager, select the test object whose description you want to modify.
2. In the **Test object details** area, select one or more properties that you want to remove from the test object description.

**Tip:** For an object in the local object repository, you can also select the required test object, right-click and select **Object Properties**, and then perform the following steps in the Object Properties dialog box.
3. Click the **Remove selected description properties** button . The selected properties are removed from the test object description.

## Specify an ordinal identifier

1. In the object repository tree of the Object Repository window or Manager, select the test object whose ordinal identifier you want to specify.

2. In the **Test object details** area, click in the cell to the right of the **Type, Value** cell under the **Ordinal identifier** row.

**Tip:** For an object in the local object repository, you can also select the required test object, right-click and select **Object Properties**, click in the cell to the right of the **Type, Value** cell under the **Ordinal identifier** row, and then perform the following steps in the Object Properties dialog box.

3. Click the **Browse** button. The Ordinal Identifier dialog box opens.

### Define related objects for a specific test object

1. In the **Visual Relation Identifier Settings** row of the Object Repository window or Object Properties dialog box, click in the **Value** cell.
2. Click the **Browse** button in the cell. The Visual Relation Identifier dialog box opens.
3. Set the options for the visual relation identifier.

#### Results:

- The visual relation identifier is added to the selected test object, and the text in the **Value** cell indicates that a visual relation identifier is defined.
- Any related objects you specified are linked to the test object for which you are using a visual relation identifier. You cannot define visual relations for those objects.
- The **Ordinal identifier** property is disabled in the **Object Details** area of the local or shared object repository, and is not used during the object identification process. However, UFT still uses this property during the learn process, when comparing existing objects with the objects to be learned, and therefore the ordinal identifier value should not be manually changed or removed.

For considerations on working with visual relation identifiers, see "["Considerations for Working with Visual Relation Identifiers" on page 236](#).

For a use-case scenario related to this task, see "["How to Define a Visual Relation Identifier for a Specific Test Object - Use-Case Scenario" on page 210](#).

### Export the objects from a local object repository

In the local object repository window, select **File > Export Local Objects**, or, for actions only, **File > Export and Replace Local Objects**. The Save Shared Object Repository window opens.

If you chose **Export Local Objects**, the local objects are exported to the specified shared object repository (a file with a **.tsr** extension). Your test or component continues to use the objects in the local object repository, and the new shared object repository is not associated with your test.

You can now use the new shared object repository like any other shared object repository.

**For actions:** If you chose **Export and Replace Local Objects**, the new shared object repository (a file with a **.tsr** extension) is associated with your test, and the objects in the local object repository are deleted. The objects in the Object Repository window are read-only, as they are now in a shared object

repository. In the Object Properties section of the Object Repository window, the repository location indicates the path and filename of the new shared object repository instead of **Local**.

## Copy an object to the local object repository

This task describes how to copy an object from a shared object repository to the local object repository.

1. Open the test or component that contain the local object repository to which you want to copy the object.
2. Open the Object Repository window by selecting **Resources > Object Repository** or clicking the **Object Repository** button .
3. In the object repository tree of the Object Repository window, select the action or component associated with the shared object repository containing the object you want to copy.
4. Select the object that you want to copy to the local object repository. (Objects in a shared object repository are read-only.) You can select multiple objects as long as the selected objects have the same parent object.
5. Select **Object > Copy to Local** or right-click the objects and select **Copy to Local**. The objects (and parent objects, if any) are copied to the local object repository and are made editable.

## Modify identification properties during a run session

You can modify the properties of the temporary version of the object during the run session without affecting the permanent values in the object repository. To do this, add a **SetTOProperty** statement in a user-defined function, or in your action.

Use the following syntax for the **SetTOProperty** method:

```
Object(description).SetTOPropertyProperty, Value
```

For details, see the topic on the **SetTOProperty** method in the **Common Methods and Properties** section of the *HP UFT Object Model Reference for GUI Testing*.

# How to Create and Manage Shared Object Repositories

## Relevant for: GUI tests and components

This task describes the different operations you can perform to manage shared object repositories using the Object Repository Manager.

This task includes the following steps:

- "Prerequisites and considerations" on the next page
- "Create a new shared object repository" on the next page
- "Enable editing for a shared object repository" on the next page
- "Associate a shared object repository with actions or components" on the next page

- "Add test objects using the Navigate and Learn option" on the next page
- "Manage repository parameters" on the next page
- "Merge object repositories into shared ones" below
- "Import a shared object repository from XML" on the next page
- "Export a shared object repository to XML" on the next page

## Prerequisites and considerations

- If you want to edit a shared object repository stored in the file system, and the shared object repository was last modified using a version of QuickTest earlier than version 9.0, UFT must convert it to the current format before you can edit it. If you do not want to convert it, you can view it in read-only format. After the file is converted and saved, you cannot use it with earlier versions of QuickTest.
- If you are working with shared object repositories that are stored in an ALM project, you must connect to ALM either from UFT or from the Object Repository Manager by choosing **ALM > ALM Connection** or clicking the **ALM Connection** button .

## Create a new shared object repository

In the Object Repository Manager window, select File > New. A new, empty object repository opens for you to add objects.

## Enable editing for a shared object repository

Select **File > Enable Editing** or click the **Enable Editing** button . The shared object repository becomes editable.

For considerations on enabling editing, see "["Considerations for Working with Shared Object Repositories" on page 184](#).

## Associate a shared object repository with actions or components

You can associate the shared object repository with one or more actions or components (via the component's application area) from within UFT.

If you want to associate the shared object repository with an application area so that it can be accessed by components, you must save it to your ALM project.

## Merge object repositories into shared ones

You can associate the shared object repository with one or more actions or components (via the component's application area) from within UFT.

If you want to associate the shared object repository with an application area so that it can be accessed by components, you must save it to your ALM project.

## Add test objects using the Navigate and Learn option

1. Select **Object > Navigate and Learn** or press F6. The **Navigate and Learn** toolbar opens.

**Note:** You cannot learn Insight test objects using this option.
2. Click the parent object (for example, Browser, Dialog, Window) you want to add to the shared object repository to focus it. The **Learn** button in the toolbar is enabled.
3. Click the **Learn** button or focus the **Navigate and Learn** toolbar and press ENTER. A flashing highlight surrounds the focused window and the object and its descendants are added to the shared object repository according to the defined filter.
4. When you finish adding the required objects to the shared object repository, click the **Close** button in the **Navigate and Learn** toolbar or press Esc. The **Navigate and Learn** toolbar closes and the Object Repository Manager is redisplayed, showing the objects you just added to the shared object repository.

## Manage repository parameters

Use the Manage Repository Parameters Dialog Box.

## Import a shared object repository from XML

You can import an XML file (created using the required format) as a shared object repository. The XML file can either be a shared object repository that you exported to XML format using the Object Repository Manager, or an XML file created using a tool such as UFT Siebel Test Express or a custom built utility. You must adhere to the XML structure and format.

### To import from XML:

1. In the Object Repository Manager, select **File > Import from XML**. In the Open Dialog Box, navigate to the XML file to import..

The XML file is imported and a summary message box opens showing information regarding the number of test objects, checkpoint and output objects, parameters, and metadata that were successfully imported from the specified file.
2. Click **OK** to close the message box. The imported XML file is opened as a new shared object repository. You can now modify it as required and save it as a shared object repository.

## Export a shared object repository to XML

You can export the objects in a shared object repository to an XML file. This enables you to edit it using any XML editor, and also enables you to save it in an accessible, versatile format.

### To export to XML:

1. Make sure that the shared object repository whose objects you want to export is the active window.
2. Make sure that the shared object repository is saved.

3. In the Object Repository Manager, select **File > Export to XML**. In the Open Dialog Box, select a location and provide a name for the XML file.  
UFT exports the objects in the shared object repository to the specified XML file, and a summary message box opens showing information regarding the number of test objects, checkpoint and output objects, parameters, and metadata that were successfully exported to the specified file.
4. Click **OK** to close the message box. You can now open the XML file and view or modify it with any XML editor.

## How to Locate an Object in an Object Repository

### Relevant for: GUI tests and components

The following steps describe how to locate a specific object in your object repository.

- "[Find an Object in an Object Repository](#)" below
- "[Highlight an Object in Your Application](#)" below
- "[Locate an Object from Your Application in the Object Repository](#)" below

### [Find an Object in an Object Repository](#)

1. Make sure that the relevant object repository is open (in the Object Repository window or Object Repository Manager).
2. Click the **Find & Replace** button . The Find and Replace dialog box opens.

### [Highlight an Object in Your Application](#)

1. Make sure your application is open to the correct window or page.

**Tip:** If you want to highlight an Insight test object located on the desktop (and not in an application), make sure that UFT is not hiding part of the object.

2. Select the test object you want to highlight in your object repository.
3. Click the **Highlight in Application** button .

### [Locate an Object from Your Application in the Object Repository](#)

1. Make sure your application is open to the correct window or page.
  2. Click the **Locate in Repository** button .
- UFT is hidden, and the pointer changes into a pointing hand. In some environments, as you move the pointing hand over your application, the test objects are highlighted.
3. Use the pointing hand to click the required object in your application.

- If the location you clicked is associated with more than one object, the Object Selection dialog box opens. The selected object is highlighted in the object repository.

**Note:**

- If the relevant object repository is not open or the object cannot be found, the object is not highlighted.
- In the Object Repository Manager, if more than one shared object repository is open, and UFT cannot locate the selected object in the active object repository, you can choose whether to look for the object in all of the currently open object repositories.
- **Locate in Repository** is not supported for Insight test objects.

## How to Define a Visual Relation Identifier for a Specific Test Object - Use-Case Scenario

**Relevant for: GUI tests and components**

This scenario describes the process you would follow to define a visual relation identifier for a specific test object that would otherwise require the use of ordinal identifiers.

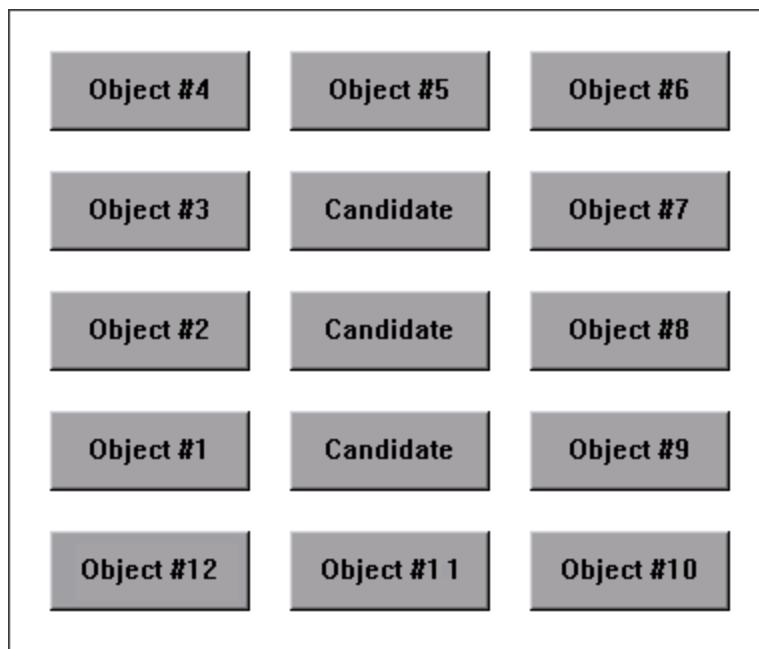
**Note:** For a task related to this scenario, see "[How to Maintain Test Objects in Object Repositories](#)" on page 202.

This scenario includes the following steps:

- "Background" below
- "Access the Visual Relation Identifier dialog box" on the next page
- "Highlight the objects in the application that match the test object's description" on the next page
- "Define the first related test object using horizontal visual relations" on page 212
- "Define the second related object using vertical visual relations" on page 213
- "Define the third related object using distance visual relations" on page 214
- "Results" on page 215

### 1. **Background**

- The application you are testing contains three identical instances of the Candidate object.



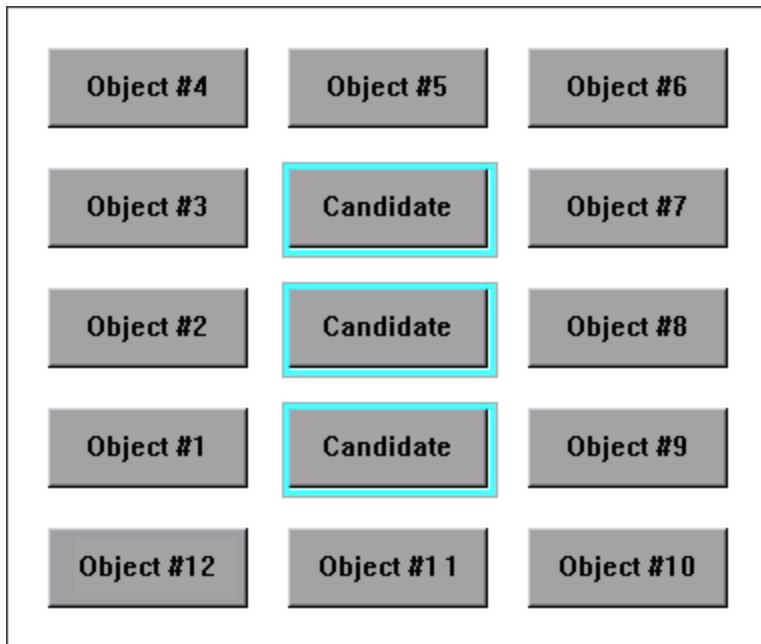
- When UFT learned the objects in the application, it assigned an ordinal identifier to each Candidate test object.
- For the purpose of this exercise, Object #1 and Object #9 make up an object pair, which is always to the left and right of the Candidate object to identify. You want to instruct UFT to identify the instance of the Candidate object that is located between the Object #1 and Object #9 object pair during every run session, even if the sorting order of the object pairs changes between run sessions.

## 2. Access the Visual Relation Identifier dialog box

- a. In UFT, open the relevant object repository and select the Candidate test object to identify.
- b. Verify that you have selected the correct test object by selecting **View > Highlight in Application**, and making sure that the correct object is highlighted in the application.
- c. Open the Visual Relation Identifier dialog box.

## 3. Highlight the objects in the application that match the test object's description

In the Visual Relation Identifier dialog box, click the **Preview** button. This instructs UFT to highlight all objects that match the test object description (ignoring the ordinal identifiers). The main UFT window is hidden, and each instance of the Candidate object in the application is highlighted, including the instance of the test object you want to identify.



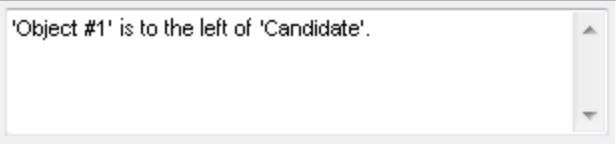
You then click the **Preview** button again to restore the UFT window.

#### 4. Define the first related test object using horizontal visual relations

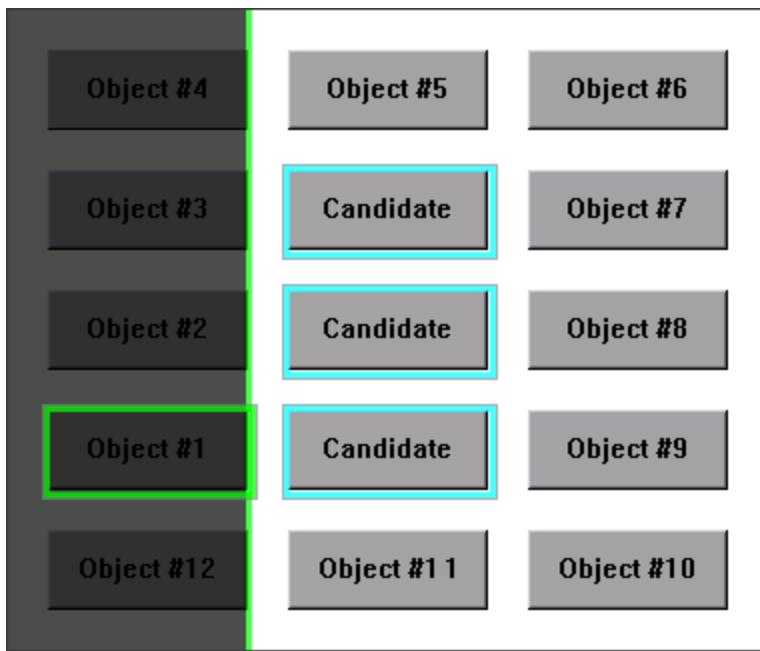
- In the **Related Objects** area, click the **Add** button. The Select Test Object dialog box opens, enabling you to either select a test object from the object repository, or add an object from the application.

For the purpose of this scenario, the first related test object is Object #1, which is located to the left of the Candidate object in the application.

- In the **Relation Details** area, select the first checkbox and then select **Left** from the drop-down list. The description area displays a summary of the visual relation identifier.



- c. To verify that the visual relation is defined correctly, click the **Preview** button again. The main UFT window is hidden, and the visual relation identifier displays the objects that match the test object's description, including the currently defined visual relation. It also highlights the selected related object, and a visual representation of the defined relation details. Since Object #1 is to the left of all three Candidate buttons, all three buttons are still highlighted when you use the **Preview** button.



## 5. Define the second related object using vertical visual relations

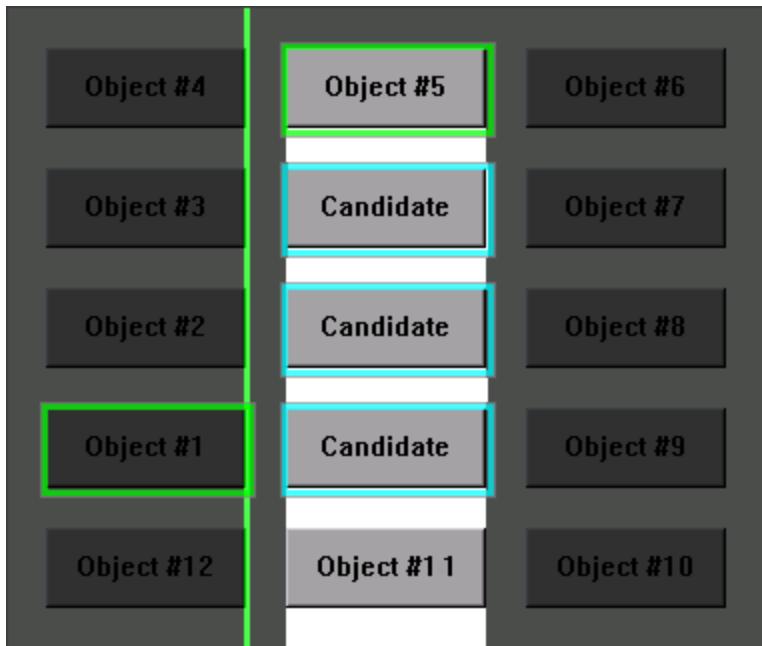
- a. In the **Related Objects** area, click the **Add** button. The Select Test Object dialog box opens, enabling you to select or add another object.

For the purpose of this scenario, the second related test object is Object #5, which is located above and vertically in line with the Candidate object.

- b. In the **Relation Details** area, select the second check box. From the drop-down list select **Above**, and then select the **In line (vertically)** checkbox. The description area displays a tooltip of all the selected visual relations.

'Object #1' is to the left of 'Candidate'.  
'Object #5' is above and vertically in line with 'Candidate'

- c. To verify that the visual relations are defined correctly, click the **Preview** button again. Since Object #5 is above all three Candidate objects, all three are still highlighted. That means you still need to select another related object to create a visual relation identifier that uniquely identifies your object.



## 6. Define the third related object using distance visual relations

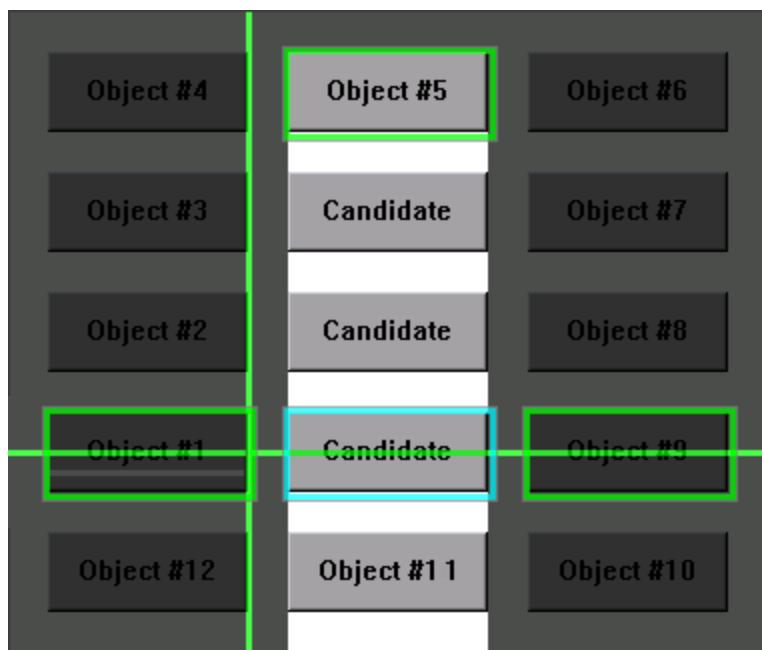
- a. In the **Related Objects** area, click the **Add** button. The Select Test Object dialog box opens, enabling you to select another test object.

For the purpose of this scenario, the third related test object is Object #9, which is the closest object to the right of the Candidate object.

- b. In the **Relation Details** area, select the third checkbox and then select **Closest on the Y-axis** from the drop-down list. The description area displays an updated summary of the visual relation identifier.

'Object #1' is to the left of 'Candidate'.  
'Object #5' is above and vertically in line with 'Candidate'  
'Object #9' is the closest object on the Y-axis to 'Candidate'.

- c. To verify that the visual relations are defined correctly, click the **Preview** button again. you can now see that this third related object enables UFT to uniquely identify the correct object.



## 7. Results

After you finish defining all of the necessary visual relations:

- The desired Candidate object is the only object in the application that is identified when you use **Preview**.
- UFT can now correctly identify the desired Candidate object during every run session, even if the user interface changes, as long as the Candidate object maintains its relative location to the three related objects you defined.
- The Ordinal Identifier property is disabled in the Object Repository Manager or window.

# Object Repository Window

## Relevant for: GUI tests and components

This window enables you to manage identification properties and object repository associations for your action or component.

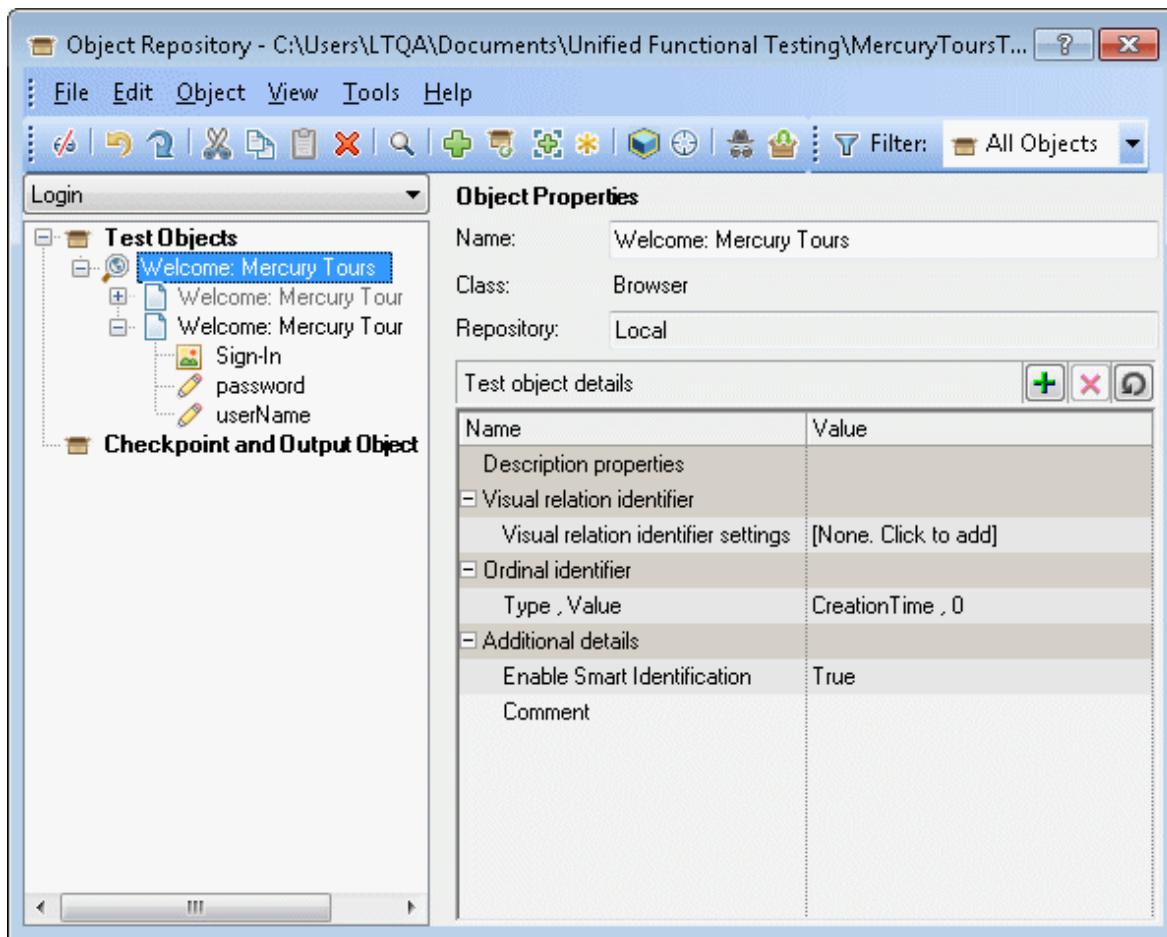
You can use the Object Repository window to:

- View the object description of any object in the repository (in local and shared object repositories).
- Modify local objects and their properties.
- Add test objects to your local object repository.

- Drag and drop test objects to your test or component. When you drag and drop a test object to your test or component, UFT inserts a step with the default operation for that test object in your test or component.

For example, if you drag and drop a button object to your test or component, a step is added using the button object, with a **Click** operation (the default operation for a button object).

The following image shows the Object Repository window displaying the local object repository of an action, with an Image test object selected. The options on this dialog box differ slightly when displaying a component's object repository, or when selecting other types of objects.



To access	<ol style="list-style-type: none"><li>Do one of the following:<ul style="list-style-type: none"><li>Ensure that a GUI test, action, or component is in focus in the document pane.</li><li>In the Solution Explorer, select a GUI test or component node or one of its child nodes.</li></ul></li><li>Do one of the following:<ul style="list-style-type: none"><li><b>UFT main window:</b> Click the <b>Object Repository</b> button , or choose <b>Resources &gt; Object Repository</b></li></ul></li></ol>
-----------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	<ul style="list-style-type: none"> <li><b>Solution Explorer pane:</b> Double-click an object repository, or right-click an object repository and choose <b>Open Repository</b></li> <li><b>GUI Test:</b> Right-click an action in the canvas and choose <b>Object Repository</b></li> <li><b>Toolbox pane:</b> Right-click a test object and choose <b>Open Resource</b></li> <li><b>Record toolbar:</b> Click the <b>Object Repository</b> button  during a recording session.</li> </ul>
<b>Important information</b>	<ul style="list-style-type: none"> <li>Local objects are editable (black). Objects from a shared object repository are read-only format (gray).</li> <li>You can modify checkpoint and output value details for objects saved in the local object repository.</li> <li>You cannot drag and drop checkpoint and output objects from the Object Repository window to your testing document.</li> <li>You can copy an object from a shared object repository to the local object repository, and then modify it.</li> <li>Test objects for environments that are not installed/loaded with UFT are displayed with a question mark icon.</li> <li>The Object Repository window is read-only during a record or run session.</li> </ul>
<b>Relevant tasks</b>	<a href="#">"How to Define a Visual Relation Identifier for a Specific Test Object - Use-Case Scenario" on page 210</a>
<b>See also</b>	<ul style="list-style-type: none"> <li>For an overview of this window, see <a href="#">"Object Repository Window - Overview" on page 186</a>.</li> <li>For details on <b>dragging and dropping test objects from other locations</b>, see <a href="#">"Toolbox Pane" on page 77</a>.</li> <li>For details on <b>modifying the properties of a test object during a run session</b>, see <a href="#">"Working with Test Objects During a Run Session" on page 197</a>.</li> <li>For details on <b>viewing and modifying object properties from other locations</b>, see <a href="#">"Maintaining Identification Properties" on page 192</a>.</li> </ul>

User interface elements are described in the following sections:

- ["Object Repository Window - Edit Toolbar" below](#)
- ["Object Repository Window - Filter Toolbar" on page 219](#)
- ["Object Repository Window Options" on page 219](#)
- ["Object Repository Window - Object Details Area" on page 220](#)
- ["Object Repository Window - Test Object Image Area" on page 221](#)

## Object Repository Window - Edit Toolbar

Button	Name	Description
	<b>Compact View</b>	<b>Compact View</b> mode displays only the object repository tree, while <b>Full View</b> mode displays the object repository tree together with the object details area.
	<b>Full View</b>	

<b>Button</b>	<b>Name</b>	<b>Description</b>
	<b>Undo</b>	All changes you make to a local object are automatically updated in all steps that use the local object as soon as you make the change. You can use the <b>Edit &gt; Undo</b> and <b>Edit &gt; Redo</b> menu options or <b>Undo</b> and <b>Redo</b> toolbar buttons to cancel or repeat your changes. After you save the current test or component, you cannot undo or redo operations that were performed before the save operation.
	<b>Redo</b>	
	<b>Cut</b>	Cuts the selected object from the object repository tree.
	<b>Paste</b>	Pastes the object in the clipboard into the object repository tree as a child of the object selected in the tree. Bottom level objects cannot contain children.
	<b>Copy</b>	Copies the selected object from the object repository tree into the clipboard.
	<b>Delete</b>	Deletes the selected object from the object repository tree.
	<b>Find &amp; Replace</b>	Finds and replaces an object in the object repository.
	<b>Add Objects to Local</b>	Enables you to add objects to the local object repository. For details, see " <a href="#">Adding and Deleting Test Objects in a Local or Shared Object Repository</a> " on page 188.
	<b>Update from Application</b>	Enables you to update the identification properties from an object in the application. For details, see " <a href="#">Maintaining Identification Properties</a> " on page 192.
	<b>Add Insight Object to Local</b>	Enables you to add an Insight object to the local object repository. For details, see " <a href="#">Adding and Deleting Test Objects in a Local or Shared Object Repository</a> " on page 188.
	<b>Define New Test Objects</b>	Enables you to define a new test object.
	<b>Highlight in Application</b>	Highlights the selected object in the object repository tree, in the application. For details, see " <a href="#">Highlighting an Object in Your Application</a> " on page 191.
	<b>Locate in Repository</b>	Enables you to select an object in the application you are testing and highlight the test object in the object repository. For details, see " <a href="#">Locating a Test Object in the Object Repository</a> " on page 192.
	<b>Object Spy</b>	Enables you to view the native properties and operations of any object in an open application, as well as the test object hierarchy, identification properties, and operations of the test object that uses to represent that object. You can also add an object to the local object repository and highlight an object in the application.
	<b>Associate Repositories (actions only)</b>	Enables you to manage the shared object repository associations of your action.

## Object Repository Window - Filter Toolbar

UI Element	Description
	<p>You can use the Filter toolbar to filter the objects shown in the Object Repository window.</p> <p>You can choose to show objects that meet one of the following criteria:</p> <ul style="list-style-type: none"> <li>• All objects in the current component or in the selected action, including all local objects and all objects in any shared object repositories associated with the selected action.</li> <li>• Only the local objects in the current component or in the selected action.</li> <li>• Only the objects in a specific shared object repository associated with the current action or component.</li> </ul> <p>To filter the Object Repository window:</p> <p>In the <b>Filter</b> toolbar list, select one of the following options:</p> <ul style="list-style-type: none"> <li>• <b>All Objects</b></li> <li>• <b>Local Objects</b></li> <li>• The name of a specific shared object repository associated with the current action or component</li> </ul> <p>The object repository tree is filtered to display only the objects from the location that you selected. The title bar of the Object Repository window indicates the current filter.</p>

## Object Repository Window Options

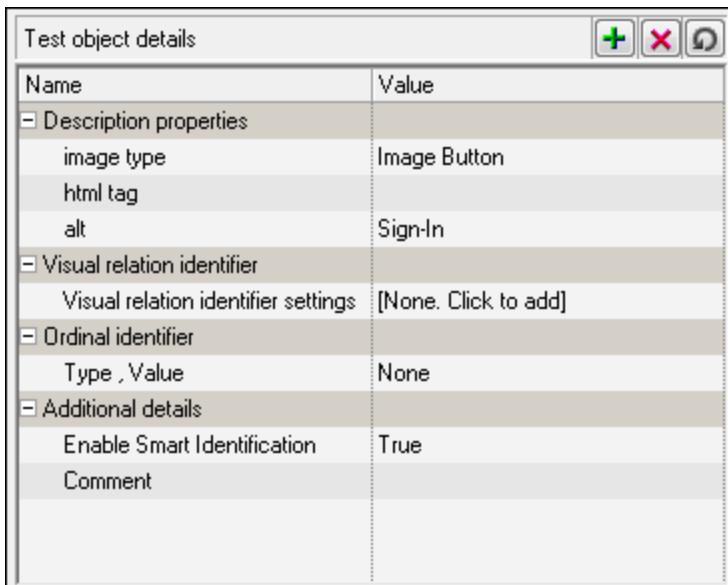
User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Elements	Description
<b>Action (for actions)</b>	Enables you to select the action whose objects you want to view.
<b>Business Component (for components)</b>	Indicates that the current testing document is a business component.
<b>&lt;Object repository tree&gt;</b>	<p>Displays all of the test objects, checkpoint objects, and output objects in the local and shared object repositories associated with in the current component or in the selected action.</p> <p>You can filter the objects shown in the object repository tree.</p> <p><b>Note:</b> If there are test objects in different associated object repositories with the same name, object class, and parent hierarchy, the object repository tree shows only the first one it finds based on the priority order defined.</p>
<b>Name</b>	The name that UFT assigns to the object. You can change the name of a object in the local object repository. For details, see " <a href="#">Maintaining Identification Properties</a> " on page 192.
<b>Class</b>	The class of the object.
<b>Repository</b>	The location (file name and path) of the object repository in which the object is located. If the object is located

UI Elements	Description
	in the local object repository, <b>Local</b> is displayed.
<b>&lt;Object details area&gt;</b>	Displays and enables you to modify one of the following: <ul style="list-style-type: none"> <li>The properties and property values used to identify a test object during a run session</li> <li>The properties of a checkpoint or output object.</li> </ul>
<b>Test object image area</b>	Displays the image used to identify the object and enables you to modify it.

## Object Repository Window - Object Details Area

The **Object details** area in the lower right side of the Object Repository window enables you to view and modify the properties and property values used to identify an object during a run session or the properties of a checkpoint or output object.



### For test objects:

UI Elements	Description
<b>Description properties</b>	The properties and property values used to identify the object during a run session.  <b>Tip:</b> <ul style="list-style-type: none"> <li>For details on specifying a property value as a constant or parameterizing a value, see "<a href="#">Maintaining Identification Properties</a>" on page 192.</li> </ul>
<b>Visual relation identifier</b>	A set of definitions that enable you to identify the object in the application according to its neighboring objects in the application. When this option is defined and enabled, the Ordinal identifier option is disabled.

UI Elements	Description
	<p><b>Note:</b> If one or more related objects cannot be found in the object repository, indicating text is displayed in the cell.</p>
<b>Ordinal identifier</b>	<p>A numerical value that indicates the object's order or location relative to other objects with an otherwise identical description (objects that have the same values for all properties).</p> <p><b>Note:</b> If a visual relation identifier is defined for a specific test object, this option is disabled. For details, see "Considerations for Working with Visual Relation Identifiers" on page 236.</p>
<b>Additional details</b>	<p>Contains the following options:</p> <ul style="list-style-type: none"> <li>• <b>Enable Smart Identification.</b> Enables you to select <b>True</b> or <b>False</b> to specify whether UFT should use Smart Identification to identify the test object during the run session if it is not able to identify the object using the test object description.</li> </ul> <p><b>Note:</b> This option is available only if Smart Identification properties are defined for the test object's class in the Object Identification Dialog Box. For details on Smart Identification, see "Smart Identification" on page 237.</p> <ul style="list-style-type: none"> <li>• <b>Comment.</b> Enables you to add textual information about the test object.</li> </ul>

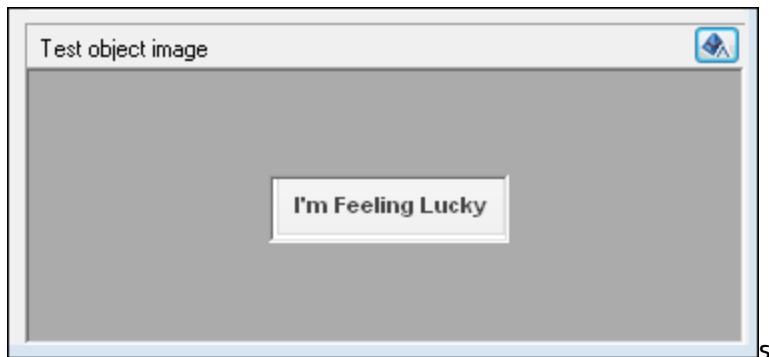
**For checkpoints:** The object details area contains the same information as the Checkpoint Properties dialog box.

**For output objects:** The object details area contains the same information as the Output Value Properties dialog box.

## Object Repository Window - Test Object Image Area

The **Test Object Image** area below the test object details area is available only for Insight test objects.

It displays the image that UFT uses to identify the object in the application. To modify the image, click the **Change Test Object Image** button  in the title bar of this area. The Change Test Object Image / Add Insight Test Object Dialog Box opens. In this dialog you can change the test object image and also the default location to click in the object when performing methods on the object.



# Object Repository Manager Main Window

## Relevant for: GUI tests and components

This window enables you to open multiple shared object repositories and modify them as needed.

The options available when specifying property values for objects in shared object repositories are different from those available when specifying properties for objects in local repositories.

<b>To access</b>	Do one of the following: <ul style="list-style-type: none"><li>• Select <b>Resources &gt; Object Repository Manager</b>.</li><li>• In the Solution Explorer, double-click a shared object repository.</li></ul>
<b>Important information</b>	See " <a href="#">Considerations for Working with the Object Repository Manager</a> " below
<b>See also</b>	<ul style="list-style-type: none"><li>• "<a href="#">Shared Object Repositories Overview</a>" on page 182</li><li>• "<a href="#">Considerations for Working with Shared Object Repositories</a>" on page 184</li></ul>

## Considerations for Working with the Object Repository Manager

- You can open as many shared object repositories as you want.
- Each shared object repository opens in a separate document window. You can then resize, maximize, or minimize the windows to arrange them as you require to copy, drag, and move objects between different shared object repositories, as well as perform operations on a single object repository.
- While the Object Repository Manager is open, you can continue working with other UFT windows.
- You cannot add checkpoint or output value objects to a shared object repository via the Object Repository Manager. They are added to the local object repository as the test object they are performed on. You can then upload them to the shared object repository, if needed. For details, see "[How to Update a Shared Object Repository From a Local Object Repository](#)" on page 254.
- When you choose a menu item or click a toolbar button in the Object Repository Manager, the operation you select is performed on the shared object repository whose window is currently active (in focus).
- If UFT is connected to an ALM project with version control enabled, you can view and manage versions of your shared object repositories, view comparisons of two shared object repository versions, and view baseline history. For details, see "[Version Control in ALM](#)" on page 951.
- Even when steps containing an object are deleted from your action or component, the objects remain in the shared object repository.
- When you close a solution or test that contains an associated open shared object repository, the shared object repository is automatically closed, even if the repository is currently open.

User interface elements are described in the sections below:

- "[Object Repository Document Window](#)" on the next page
- "[Object Details Area \(Test Objects\)](#)" on the next page

- "Test Object Image Area (Insight Test Objects)" below
- "Object Details Area (Checkpoint Objects)" on the next page
- "Object Details Area (Output Value Objects) " on the next page
- "Toolbar Buttons" on the next page

## Object Repository Document Window

Each open object repository contains the following user interface elements (unlabeled elements are shown in angle brackets):

UI Element	Description
<b>Title bar</b>	Displays the name and file path of the shared object repository currently active (in focus).
<b>Object Repository tree</b>	Located on the left side of the Object Repository window, and contains all objects in the shared object repository. Test objects of environments that are not installed with UFT are displayed with a question mark icon in the test object tree.
<b>Name</b>	Specifies the name that UFT assigns to the selected object. You can change the object name. For details, see " <a href="#">Maintaining Identification Properties</a> " on page 192.
<b>Class</b>	Specifies the class of the selected object.

## Object Details Area (Test Objects)

Enables you to view the properties and property values used to identify a test object during a run session.

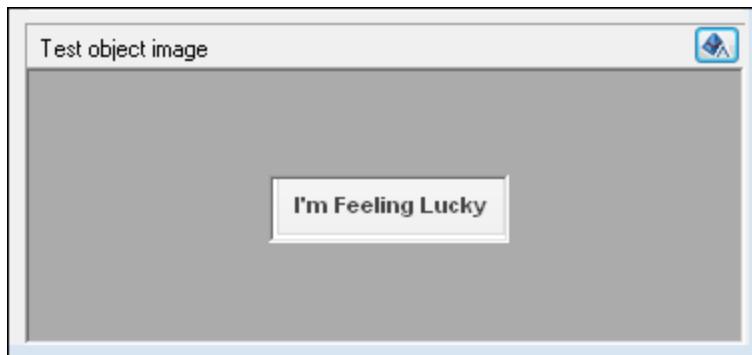
For details on key functionality of this area, see:

- "[Maintaining Identification Properties](#)" on page 192
- "[Maintaining Identification Properties](#)" on page 192
- "[How to Maintain Test Objects in Object Repositories](#)" on page 202
- "[Maintaining Identification Properties](#)" on page 192
- "[How to Define a Visual Relation Identifier for a Specific Test Object - Use-Case Scenario](#)" on page 210

## Test Object Image Area (Insight Test Objects)

The **Test Object Image** area below the test object details area is available only for Insight test objects.

It displays the image that UFT uses to identify the object in the application. To modify the image, click the **Change Test Object Image** button  in the title bar of this area. The Change Test Object Image dialog box opens. In this dialog you can change the test object image and also the default location to click in the object when performing methods on the object.



### Object Details Area (Checkpoint Objects)

Enables you to view the properties of a checkpoint object, the same way as you do in the relevant checkpoint properties dialog box.

### Object Details Area (Output Value Objects)

Enables you to view the properties of an output value object, the same way as you do in the relevant checkpoint properties dialog box.

### Toolbar Buttons

Button	Description
	<b>New.</b> Enables you to create a new shared object repository. For details, see " <a href="#">Create a new shared object repository</a> " on page 207.
	<b>Open.</b> Enables you to open a shared object repository from the file system or from ALM. For details, see " <a href="#">How to Create and Manage Shared Object Repositories</a> " on page 206.
	<b>Save.</b> Enables you to save the active shared object repository to the file system or to ALM. For details, see " <a href="#">How to Create and Manage Shared Object Repositories</a> " on page 206.
	<b>Enable Editing.</b> Enables you to edit the active shared object repository, by making the shared object repository editable. For details, see " <a href="#">Enable editing for a shared object repository</a> " on page 207.
	<b>Undo.</b> Enables you to undo the previous operation performed in the active shared object repository. You do this in the same way as in a local object repository.
	<b>Redo.</b> Enables you to redo the operation that was previously undone in the active shared object repository. You do this in the same way as in a local object repository.
	<b>Cut.</b> Enables you to cut the selected item or object in the active shared object repository. You do this in the same way as in a local object repository.

Button	Description
	<b>Copy.</b> Enables you to copy the selected item or object to the Clipboard in the active shared object repository. You do this in the same way as in a local object repository.
	<b>Paste.</b> Enables you to paste the data from the Clipboard to the active shared object repository. You do this in the same way as in a local object repository.
	<b>Delete.</b> Enables you to delete the selected item or object in the active shared object repository. You do this in the same way as in a local object repository.
	<b>Find and Replace.</b> Enables you to find an object, property, or property value in the active shared object repository. You can also find and replace specified property values. You do this in the same way as in a local object repository.
	<b>Add Objects.</b> Enables you to add objects to the active shared object repository. You do this in the same way as in a local object repository. For details, see " <a href="#">Adding and Deleting Test Objects in a Local or Shared Object Repository</a> " on page 188.
	<b>Update from Application.</b> Enables you to update identification properties in the active shared object repository according to the actual properties of the object in your application. You do this in the same way as in a local object repository. For details, see " <a href="#">Maintaining Identification Properties</a> " on page 192.
	<b>Add Insight Object.</b> Enables you to add an Insight object to the active shared object repository. You do this in the same way as in a local object repository. For details, see " <a href="#">Adding and Deleting Test Objects in a Local or Shared Object Repository</a> " on page 188.
	<b>Define new Test Object.</b> Enables you to define a test object that does not yet exist in your application and add it to the active shared object repository. You do this in the same way as in a local object repository.
	<b>Highlight in Application.</b> Enables you to select an object in the active shared object repository and highlight it in your application. You do this in the same way as in a local object repository. For details, see " <a href="#">How to Locate an Object in an Object Repository</a> " on page 209.
	<b>Locate in Repository.</b> Enables you to select an object in your application and highlight it in the active shared object repository. You do this in the same way as in a local object repository. For details, see " <a href="#">Locating a Test Object in the Object Repository</a> " on page 192.
	<b>ALM Connection.</b> Enables you to connect to ALM to work with object repository files stored in an ALM project. You can connect to ALM from the main UFT window or from the Object Repository Manager.
	<b>Object Spy.</b> Opens the Object Spy Dialog Box, enabling you to view the native properties and operations of any object in an open application, as well as the test object hierarchy, identification properties, and operations of the test object that UFT uses to represent that object. For details, see " <a href="#">How to Use the Object Spy to View Object Properties and Operations or Add an Object to a Repository</a> " on page 173.
	<b>Manage Repository Parameters.</b> Enables you to add, edit, and delete repository parameters in the active shared object repository.

# Troubleshooting and Limitations - Object Repositories

## **Relevant for: GUI tests and components**

This section describes troubleshooting and limitations for working with object repositories.

- If you modify the name of a test object in the Object Repository while your test or component script contains a syntax error, the new name is not updated correctly within your test or component steps.

**Workaround:** Clear the **Automatically update test and component steps when you rename test objects** check box (**Tools > Options > GUI Testing tab > General** node) and perform the renames in the steps manually (recommended) or solve the syntax error, and then close and reopen the document in UFT to display the renamed objects in your steps.

- **For actions:** If you use the Export and Replace Local Objects option for an object repository that contains action parameters, the created repository parameters are mapped to test parameters instead of action parameters.

**Workaround:** Manually adjust the mapping in the exported object repository.

# Chapter 32: Configuring Object Identification

**Relevant for: GUI tests and components**

This chapter includes:

• <a href="#">Object Identification Configuration - Overview</a> .....	228
• <a href="#">Mandatory and Assistive Properties</a> .....	229
• <a href="#">Ordinal Identifiers</a> .....	230
• <a href="#">Visual Relation Identifiers</a> .....	234
• <a href="#">Smart Identification</a> .....	237
• <a href="#">When to Use Smart Identification</a> .....	238
• <a href="#">The Smart Identification Process</a> .....	238
• <a href="#">Smart Identification Information in the Run Results</a> .....	239
• <a href="#">How UFT Uses Smart Identification - Use-Case Scenario</a> .....	239
• <a href="#">Test Object Mapping for Unidentified or Custom Classes</a> .....	241
• <a href="#">How to Configure Object Identification for a Test Object Class</a> .....	242
• <a href="#">How to Manage Identification Properties of a Test Object Class</a> .....	243
• <a href="#">How to Map an Unidentified or Custom Class to a Standard Windows Class</a> .....	243

# Object Identification Configuration - Overview

## Relevant for: GUI tests and components

When UFT learns an object, it learns a set of properties and values that uniquely describe the object within the object hierarchy. In most cases, this description is sufficient to enable UFT to identify the object during the run session.

If you find that the description UFT uses for a certain object class is not the most logical one for the objects in your application, or if you expect that the values of the properties in the object description may change frequently, you can configure the way that UFT learns and identifies objects. You can also map user-defined objects to standard test object classes and configure the way UFT learns objects from your user-defined object classes.

UFT has a predefined set of properties that it learns for each test object. If these mandatory property values are not sufficient to uniquely identify a learned object, UFT can add some assistive properties and/or an ordinal identifier to create a unique description.

**Mandatory properties** are properties that UFT always learns for a particular test object class.

**Assistive properties** are properties that UFT learns only if the mandatory properties that UFT learns for a particular object in your application are not sufficient to create a unique description. If several assistive properties are defined for an object class, then UFT learns one assistive property at a time, and stops as soon as it creates a unique description for the object. If UFT does learn assistive properties, those properties are added to the test object description.

### Note:

- If the combination of all defined mandatory and assistive properties is not sufficient to create a unique test object description, UFT also learns the value for the selected ordinal identifier. For details, see "[Ordinal Identifiers](#)" on page 230.
- If a specific test object relies mainly on ordinal identifiers, you can also define visual relation identifiers for that test object, to help improve identification reliability for that object. For details, see "[Visual Relation Identifiers](#)" on page 234.

When you run a test or component, UFT searches for the object that matches the description it learned (without the ordinal identifier). If it cannot find any object that matches the description, or if more than one object matches the description, UFT uses the **Smart Identification** mechanism (if enabled) to identify the object. In many cases, a Smart Identification definition can help UFT identify an object, if it is present, even when the learned description fails due to changes in one or more property values. The test object description is used together with the ordinal identifier only in cases where the Smart Identification mechanism does not succeed in narrowing down the object candidates to a single object.

You use the Object Identification Dialog Box to configure the mandatory, assistive, and ordinal identifier properties that UFT uses to learn descriptions of the objects in your application, and to enable and configure the Smart Identification mechanism.

The Object Identification dialog box also enables you to configure new user-defined classes and map them to an existing test object class so that UFT can recognize objects from your user-defined classes when you run your test or component.

To learn more, see:

- [Mandatory and Assistive Properties](#) ..... 229
- [Ordinal Identifiers](#) ..... 230
  - [The Index Ordinal Identifier](#) ..... 231
  - [The Location Ordinal Identifier](#) ..... 232
  - [The CreationTime Ordinal Identifier](#) ..... 233
- [Visual Relation Identifiers](#) ..... 234
  - [How UFT Interprets Horizontal and Vertical Visual Relations](#) ..... 235
  - [Considerations for Working with Visual Relation Identifiers](#) ..... 236

## Mandatory and Assistive Properties

### **Relevant for: GUI tests and components**

If you find that the description UFT uses for a certain object class is not the most logical one for the objects in your application, or if you expect that the values of the properties currently used in the object description may change, you can modify the mandatory and assistive properties that UFT learns when it learns an object of a given class.

During the run session, UFT looks for objects that match all properties in the test object description—it does not distinguish between properties that were learned as mandatory properties and those that were learned as assistive properties.

For example, the default mandatory properties for a Web Image object are the **alt**, **html tag**, and **image type** properties. There are no default assistive properties defined. Suppose your Web site contains several space holders for different collections of rotating advertisements. You want to create a test or component that clicks on the images in each one of these space holders.

However, since each advertisement image has a different **alt** value, one **alt** value would be added when you create the test or component, and most likely another **alt** value will be captured when you run the test or component, causing the run to fail. In this case, you could remove the **alt** property from the Web Image mandatory properties list. Instead, since each advertisement image displayed in a certain space holder in your site has the same value for the image **name** property, you could add the **name** property to the mandatory properties to enable UFT to uniquely identify the object.

Also, suppose that whenever a Web image is displayed more than once on a page (for example, a logo displayed on the top and bottom of a page), the Web designer adds a special **ID** property to the Image tag. The mandatory properties are sufficient to create a unique description for images that are displayed only once on the page, but you also want UFT to learn the **ID** property for images that are

displayed more than once on a page. To do this, you add the **ID** property as an assistive property, so that UFT learns the **ID** property only when it is necessary for creating a unique test object description.

## Ordinal Identifiers

### Relevant for: GUI tests and components

In addition to learning the mandatory and assistive properties specified in the Object Identification Dialog Box, UFT can also learn a backup ordinal identifier for each test object. The **ordinal identifier** assigns the object a numerical value that indicates its order relative to other objects with an otherwise identical description (objects that have the same values for all properties specified in the mandatory and assistive property lists). This ordered value enables UFT to create a unique description when the mandatory and assistive properties are not sufficient to do so.

The assigned ordinal property value is a relative value and is accurate only in relation to the other objects displayed when UFT learns an object. Therefore, changes in the layout or composition of your application page or screen can cause this value to change, even though the object itself has not changed in any way. For this reason, UFT learns a value for this backup ordinal identifier only when it cannot create a unique description using all available mandatory and assistive properties.

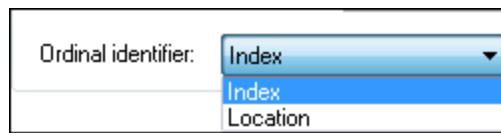
In addition, even if UFT learns an ordinal identifier, it will use the identifier during the run session only if:

- The learned description and the Smart Identification mechanism are not sufficient to identify the object in your application.
- A visual relation identifier is not defined for the test object. For details, see "["Visual Relation Identifiers" on page 234](#).

UFT can use the following types of ordinal identifiers to identify an object:

- **Index.** Indicates the order in which the object appears in the application code relative to other objects with an otherwise identical description. For details, see "["The Index Ordinal Identifier" on the next page](#).
- **Location.** Indicates the order in which the object appears within the parent window, frame, or dialog box relative to other objects with an otherwise identical description. For details, see "["The Location Ordinal Identifier" on page 232](#).
- **CreationTime.** (Browser object only.) Indicates the order in which the browser was opened relative to other open browsers with an otherwise identical description. For details, see "["The CreationTime Ordinal Identifier" on page 233](#).

By default, an ordinal identifier type exists for each test object class. To modify the default ordinal identifier, you can select the desired type from the **Ordinal identifier** box.



**Tip:** When learning an object, if UFT successfully creates a unique test object description using the

mandatory and assistive properties, it does not learn an ordinal identifier value. You can add an ordinal identifier to an object's identification properties at a later time using the **Ordinal Identifier Dialog Box**, available from the **Object Properties** or **Object Repository** window. For details, see "[Test Objects in Object Repositories](#)" on page 178.

To learn more about ordinal identifiers, see:

- "[The Index Ordinal Identifier](#)" below
- "[The Location Ordinal Identifier](#)" on the next page
- "[The CreationTime Ordinal Identifier](#)" on page 233

## The Index Ordinal Identifier

### **Relevant for: GUI tests and components**

While learning an object, UFT can assign a value to the test object's **Index** property to uniquely identify the object. The value is based on the order in which the object appears within the source code. The first occurrence is 0.

**Index** property values are object-specific. Therefore, if you use `Index:=3` to describe a **WebEdit** test object, UFT searches for the fourth **WebEdit** object in the page. However, if you use `Index:=3` to describe a **WebElement** object, UFT searches for the fourth **Web** object on the page—regardless of the type—because the **WebElement** object applies to all **Web** objects.

For example, suppose a page contains the following objects:

- An image with the name **Apple**
- An image with the name **UserName**
- A **WebEdit** object with the name **UserName**
- An image with the name **Password**
- A **WebEdit** object with the name **Password**

The following statement refers to the third item in the list, as this is the first **WebEdit** object on the page with the name **UserName**:

```
WebEdit("Name:=UserName", "Index:=0")
```

In contrast, the following statement refers to the second item in the list, as that is the first object of any type (**WebElement**) with the name **UserName**:

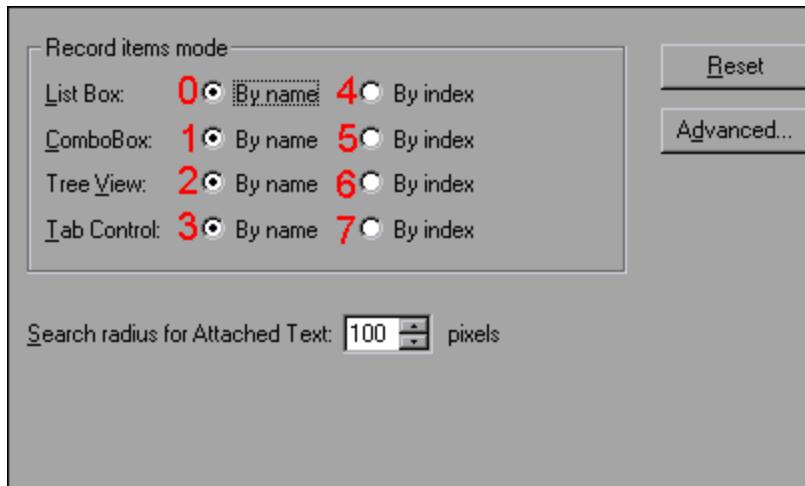
```
WebElement("Name:=UserName", "Index:=0")
```

## The Location Ordinal Identifier

### Relevant for: GUI tests and components

While learning an object, UFT can assign a value to the test object's **Location** property to uniquely identify the object. The value is based on the order in which the object appears within the window, frame, or dialog box, in relation to other objects with identical properties. The first occurrence of the object is 0. Values are assigned in columns from top to bottom, and left to right.

In the following example, the radio buttons in the dialog box are numbered according to their **Location** property:



**Location** property values are object-specific. Therefore, if you use `Location:=3` to describe a `WinButton` test object, UFT searches from top to bottom, and left to right for the fourth `WinButton` object in the page. However, if you use `Location:=3` to describe a `WinObject` object, UFT searches from top to bottom, and left to right for the fourth standard object on the page—regardless of the type—because the `WinObject` object applies to all standard objects.

For example, suppose a dialog box contains the following objects:

- A button object with the name OK
- A button object with the name Add/Remove
- A check box object with the name Add/Remove
- A button object with the name Help
- A check box object with the name Check spelling

The following statement refers to the third item in the list, as this is the first check box object on the page with the name Add/Remove:

```
WinCheckBox("Name:=Add/Remove", "Location:=0")
```

In contrast, the following statement, refers to the second item in the list, as that is the first object of any type (WinObject) with the name Add/Remove:

```
WinObject("Name:=Add/Remove", "Location:=0")
```

## The CreationTime Ordinal Identifier

### **Relevant for: GUI tests and components**

While learning a browser object, UFT assigns a value to the CreationTime identification property. This value indicates the order in which the browser was opened relative to other open browsers. The first browser that opens receives the value CreationTime = 0.

During the run session, if UFT is unable to identify a browser object based solely on its test object description, it examines the order in which the browsers were opened, and then uses the CreationTime property to identify the correct one.

For example, if UFT learns three browsers that are opened at 9:01 pm, 9:03 pm, and 9:05 pm, UFT assigns the CreationTime values, as follows: CreationTime = 0 to the 9:01 pm browser, CreationTime = 1 to the 9:03 pm browser, and CreationTime = 2 to the 9:05 pm browser.

At 10:30 pm, when you run a test or component with these browser objects, suppose the browsers are opened at 10:31 pm, 10:33 pm, and 10:34 pm. UFT identifies the browsers, as follows: the 10:31 pm browser is identified with the Browser test object with CreationTime = 0, 10:33 pm browser is identified with the test object with CreationTime = 1, 10:34 pm browser is identified with the test object with CreationTime = 2.

If there are several open browsers, the one with the lowest CreationTime is the first one that was opened and the one with the highest CreationTime is the last one that was opened. For example, if there are three or more browsers open, the one with CreationTime = 2 is the third browser that was opened. If seven browsers are opened during a recording session, the browser with CreationTime = 6 is the last browser opened.

If a step was created on a Browser object with a specific CreationTime value, but during a run session there is no open browser with that CreationTime value, the step will run on the browser that has the highest CreationTime value. For example, if a step was created on a Browser object with CreationTime = 6, but during the run session there are only two open browsers, with CreationTime = 0 and CreationTime = 1, then the step runs on the last browser opened, which in this example is the browser with CreationTime = 1.

**Note:** It is possible that at a particular time during a session, the available CreationTime values may not be sequential. For example, if you open six browsers during a record or run session, and then during that session, you close the second and fourth browsers (CreationTime values 1 and 3), then at the end of the session, the open browsers will be those with CreationTime values 0, 2, 4, and 5.

## Visual Relation Identifiers

### Relevant for: GUI tests and components

When testing applications with multiple identical objects, UFT assigns an ordinal identifier to each test object. This may lead to unreliable object identification. However, it may not (immediately) result in a failed step. For details, see ["Ordinal Identifiers" on page 230](#).

To improve object identification, you can create a **visual relation identifier**, which is a set of definitions that enable you to identify the object in the application according to the relative location of its neighboring objects. You can select neighboring objects that will maintain the same relative location to your object, even if the user interface design changes. This enables you to help UFT identify similar objects much as a human tester would, and helps create more stable object repositories that can withstand predictable changes to the application's user interface.

### How Visual Relation Identifiers Work

Suppose that someone shows you a photograph of identical twins sitting at different desks in a classroom and asks you to remember the differences between them so that you can identify each twin successfully when shown different photographs at a later time.

You are told that one differentiating characteristic is that one twin (twin A) always carries a blue school bag, and that the other twin (twin B) always carries a red school bag. You are then told that each twin has an assigned desk partner, which means that even if the twins sit at different desks in other photographs, they always sit next to their assigned partners.

One way to remember which twin is which is by identifying twin A in this manner: Always look for the blue school bag and the twin's desk partner to locate twin A.

UFT uses visual relation identifiers in a similar manner. It compares the relative locations of the test objects you defined in the visual relation identifier with the multiple identical objects.

### How UFT Uses Visual Relation Identifiers

- During a run session, UFT first attempts to identify the test object using the object's description properties. For details, see ["How UFT Identifies Objects During a Run Session" on page 165](#).
- If UFT finds one or more objects matching the test object's description, it attempts to identify the object using the visual relation identifier. For details, see ["How to Define a Visual Relation Identifier for a Specific Test Object - Use-Case Scenario" on page 210](#).
- After the visual relation identifier is applied, if no objects or more than one object is found, the visual relation identifier fails, and UFT continues to Smart Identification (when defined for that test object class). For details, see ["Smart Identification" on page 237](#).
- After Smart Identification is applied, if no objects or more than one object is found, no ordinal identifiers are used for that test object.

For general considerations on working with visual relation identifiers, see "[Considerations for Working with Visual Relation Identifiers](#)" on the next page.

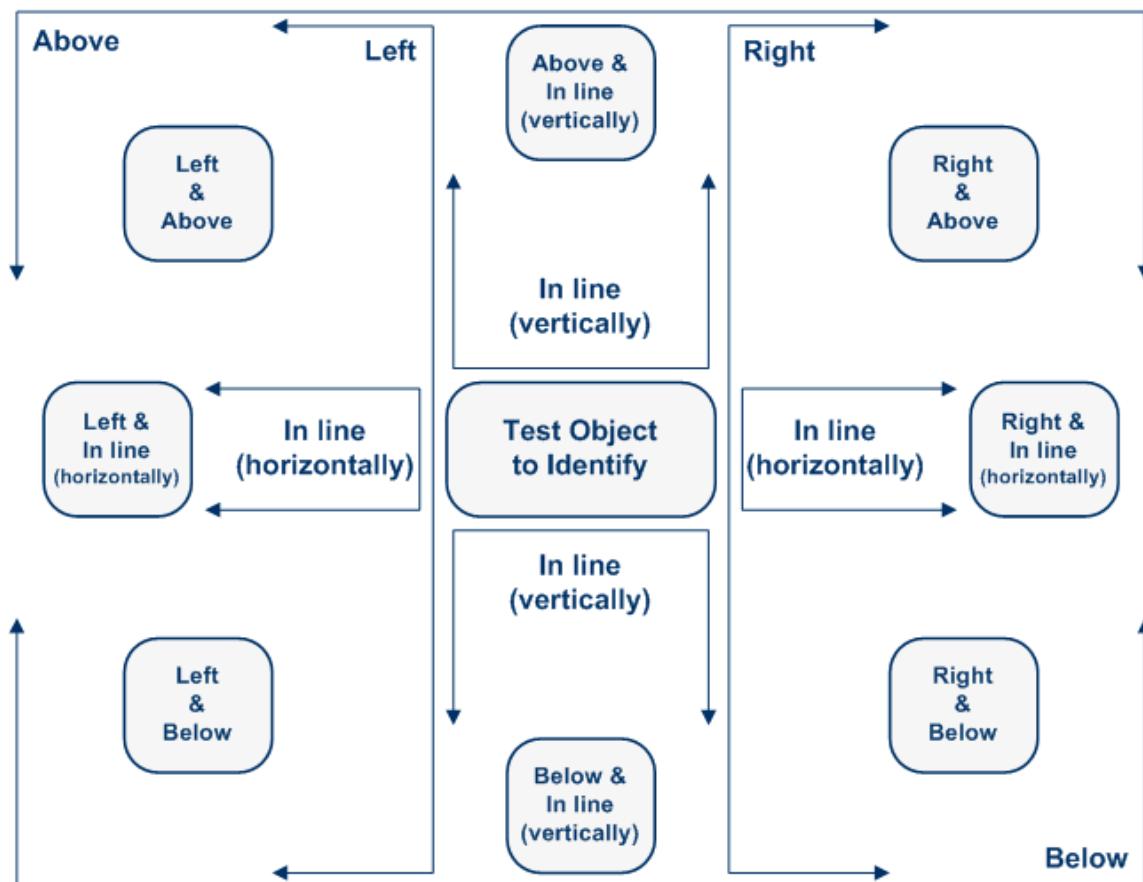
For a workflow of the general object identification process, see "[Object Identification Process Workflow](#)" on page 172.

You define visual relations in the Visual Relation Identifier Dialog Box, which is accessible from the local or shared object repository, or from the Object Properties Dialog Box.

## How UFT Interprets Horizontal and Vertical Visual Relations

### Relevant for: GUI tests and components

The following diagram illustrates the way UFT interprets horizontal and vertical visual relations. It also shows the boundaries that are used for determining **in line** related objects.

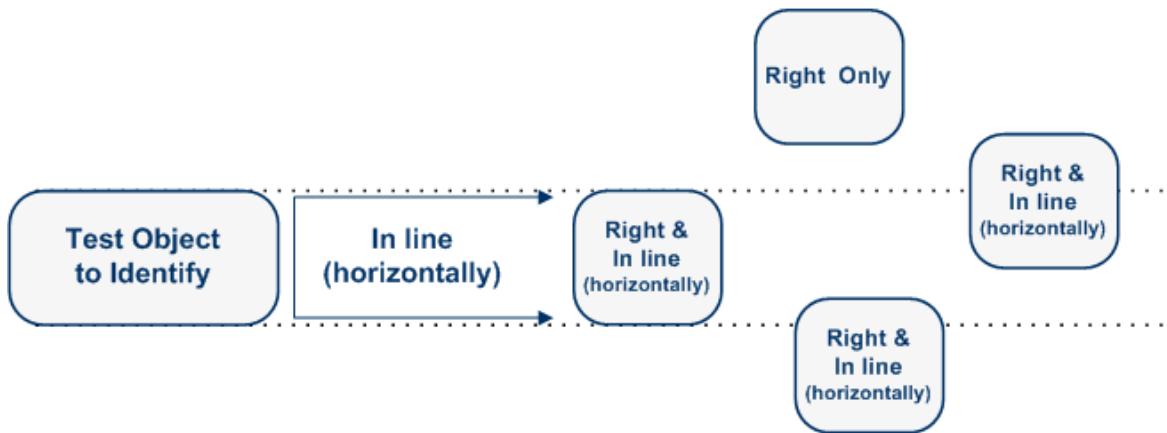


### How in-line related objects are identified

When you select a related object as a horizontal and/or vertical relation in the Visual Relation Identifier dialog box, you can also fine-tune that definition by indicating that it is in line with the test object to identify.

UFT identifies the related object as **in line** even if the area of the related object surface is only partially in line with the test object.

The following example illustrates how UFT identifies related objects that are in line with the test object to identify.



## Considerations for Working with Visual Relation Identifiers

### Relevant for: GUI tests and components

Consider the following when working with visual relation identifiers:

- If you define a visual relation identifier for a test object, then that test object's ordinal identifier (if it exists) is not used during the test object identification process (when running steps, highlighting objects in the application, etc.). To indicate this, the **Ordinal identifier** option for that test object is disabled in the Object Repository window.
- If you add a related object that was not previously in the object repository, then the test object for the related object is added to that object repository, even if you click **Cancel** in the Visual Relation Identifier dialog box.

**Workaround:** If you do not need that test object, manually delete it from the object repository.

- If you delete a test object (A) that is used in the visual relation identifier for another test object (B), you must make sure to remove the deleted test object A from the **Related Object** list for test object B.
- Visual relation identifiers are used only during a run session, or when identifying objects in the application (for example, from the Object Repository window or the Object Spy). Therefore, even if you define related objects for a specific test object, if UFT re-learns the object, it uses only the identification properties that are defined in the Object Identification Dialog Box for that test object class, as well as an ordinal identifier (if needed). This may cause UFT to learn the same object more than once.

**Example:** If you learned the Object1 test object with an ordinal identifier property value of 1, and then you manually remove the ordinal identifier or otherwise significantly modify the

description properties (because you are depending on the visual relation identifiers to fine tune the description), then the next time a learn session includes this object, UFT will not recognize **Object1** as the same object and will learn a new test object with the name **Object1\_1**.

- UFT uses visual relation identifiers only when one or more objects match the test object's description properties during the identification process. If no objects in the application match the test object's description properties, then the visual relation identifier you defined is ignored, and UFT continues to Smart Identification (if defined for that test object class).

For details on the complete flow that UFT uses to identify objects, see "[Object Identification Process Workflow](#)" on page 172.

- A test object cannot be used as a related object to itself.
- The following test objects cannot act as related objects for another test object's visual relation identifier:
  - A test object that has a visual relation identifier.
  - A child test object for one of its parent objects.
- You can retrieve or replace the visual relation identifier settings of a specific test object during a run session using the **VisualRelationsCollection** object. For details, see the **VisualRelationsCollection** object in the **Utility Objects** section of the *HP UFT Object Model Reference for GUI Testing*.

## Smart Identification

### Relevant for: GUI tests and components

When UFT uses the learned description to identify an object, it searches for an object that matches all of the property values in the description. In most cases, this description is the simplest way to identify the object, and, unless the main properties of the object change, this method will work.

If UFT is unable to find any object that matches the learned object description, or if it finds more than one object that fits the description, then UFT ignores the learned description, and uses the Smart Identification mechanism (if defined and enabled) to try to identify the object.

The Smart Identification Properties Dialog Box enables you to create and modify the Smart Identification definition that UFT uses for a selected test object class. Configuring Smart Identification properties enables you to help UFT identify objects in your application, even if some of the properties in the object's learned description have changed.

While the Smart Identification mechanism is more complex, it is more flexible. Therefore, if configured logically, a Smart Identification definition can probably help UFT identify an object, if it is present, even when the learned description fails.

The Smart Identification mechanism uses two types of properties:

- **Base Filter Properties.** The most fundamental properties of a particular test object class; those whose values cannot be changed without changing the essence of the original object. For example, if a Web link's tag was changed from <A> to any other value, you could no longer call it the same object.
- **Optional Filter Properties.** Other properties that can help identify objects of a particular class. These properties are unlikely to change on a regular basis, but can be ignored if they are no longer applicable.

Smart identification is not relevant for Insight test objects.

To learn more, see:

- [When to Use Smart Identification](#) ..... 238
- [The Smart Identification Process](#) ..... 238
- [Smart Identification Information in the Run Results](#) ..... 239
- [How UFT Uses Smart Identification - Use-Case Scenario](#) ..... 239

## When to Use Smart Identification

### **Relevant for: GUI tests and components**

You should enable the Smart Identification mechanism only for test object classes that have defined Smart Identification configuration. However, even if you define a Smart Identification configuration for a test object class, you may not always want to learn the Smart Identification property values. If you do not want to learn the Smart Identification properties, clear the **Enable Smart Identification** check box.

Even if you choose to learn Smart Identification properties for an object, you can disable use of the Smart Identification mechanism for a specific object in the Object Properties or Object Repository window. For tests, you can also disable use of the mechanism for an entire test in the Run pane of the Test Settings dialog box. For details, see "[Test Objects in Object Repositories](#)" on page 178.

However, if you do not learn Smart Identification properties, you cannot enable the Smart Identification mechanism for an object later.

## The Smart Identification Process

### **Relevant for: GUI tests and components**

If UFT activates the Smart Identification mechanism during a run session (because it was unable to identify an object based on its learned description), it follows the following process to identify the object:

1. UFT "forgets" the learned test object description and creates a new **object candidate** list containing the objects (within the object's parent object) that match all of the properties defined in the Base Filter Properties list.
2. UFT filters out any object in the object candidate list that does not match the first property listed

in the Optional Filter Properties list. The remaining objects become the new object candidate list.

3. UFT evaluates the new object candidate list:
  - If the new object candidate list still has more than one object, UFT uses the new (smaller) object candidate list to repeat the filter for the next optional filter property in the list.
  - If the new object candidate list is empty, UFT ignores this optional filter property, returns to the previous object candidate list, and repeats the filter for the next optional filter property in the list.
  - If the object candidate list contains exactly one object, then UFT concludes that it has identified the object and performs the statement containing the object.
4. UFT continues the filtering process described above until it either identifies one object, or runs out of optional filter properties to use.

If, after completing the Smart Identification elimination process, UFT still cannot identify the object, then UFT uses the learned description plus the ordinal identifier to identify the object.  
If the combined learned description and ordinal identifier are not sufficient to identify the object, then UFT pauses the run session and displays a Run Error message.

## Smart Identification Information in the Run Results

### **Relevant for: GUI tests and components**

If the learned description does not enable UFT to identify a specified object in a step, and a Smart Identification definition is defined (and enabled) for the object, then UFT tries to identify the object using the Smart Identification mechanism.

If UFT successfully uses Smart Identification to find an object after no object matches the learned description, the step is assigned a **Warning** status in the run results, and the result details for the step indicate that the Smart Identification mechanism was used.

If the Smart Identification mechanism cannot successfully identify the object, UFT uses the learned description plus the ordinal identifier to identify the object. If the object is still not identified, the test or component fails and a normal failed step is displayed in the results.

## How UFT Uses Smart Identification - Use-Case Scenario

### **Relevant for: GUI tests and components**

The following example walks you through the object identification process for an object:

Suppose you have the following statement in your test or component:

```
Browser("Mercury Tours").Page("Mercury Tours").Image("Login").Click 22,17
```

When you created your test or component, UFT learned the following object description for the Login image:

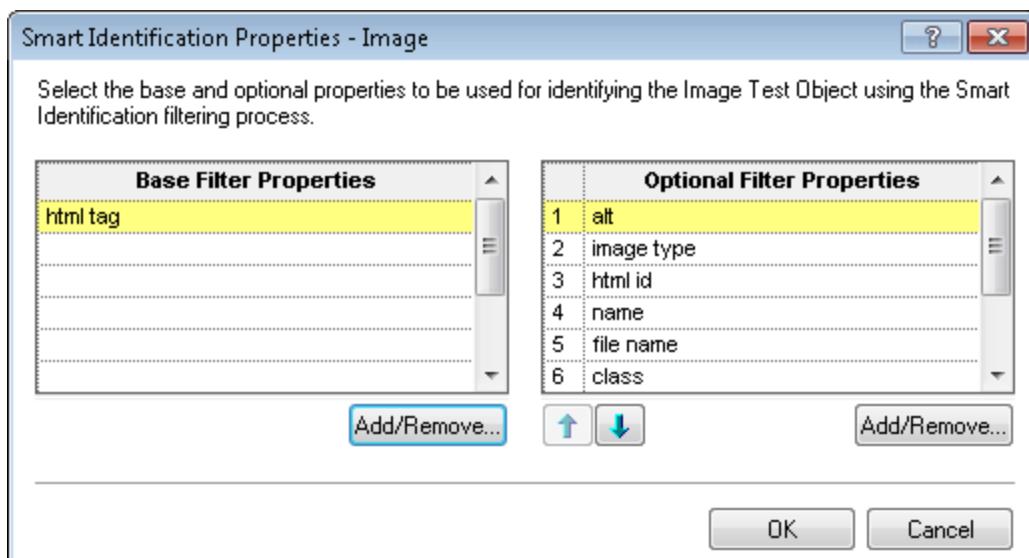
Name	Value
<b>Description properties</b>	
image type	Image Button
html tag	INPUT
alt	Sign-In

However, at some point after you created your test or component, a second login button (for logging into the VIP section of the Web site) was added to the page, so the Web designer changed the original Login button's **alt** tag to: basic login.

The default description for Web Image objects (**alt**, **html tag**, **image type**) works for most images in your site, but it no longer works for the Login image, because that image's **alt** property no longer matches the learned description. Therefore, when you run your test or component, UFT is unable to identify the Login button based on the learned description. However, UFT succeeds in identifying the Login button using its Smart Identification definition.

The following explanation describes the process that UFT uses to find the Login object using Smart Identification:

- According to the Smart Identification definition you have for Web image objects, UFT learned the values of the following properties when it learned the Login image:



The learned values are as follows:

#### Base Filter Properties:

Property	Value
html tag	INPUT

#### Optional Filter Properties:

Property	Value
<b>alt</b>	Login
<b>name</b>	login
<b>file name</b>	login.gif
<b>class</b>	<null>
<b>visible</b>	1

2. UFT begins the Smart Identification process by identifying the five objects on the Mercury Tours page that match the base filter properties definition (**html tag** = INPUT). UFT considers these to be the object candidates and begins checking the object candidates against the **Optional Filter Properties** list.
3. UFT checks the **alt** property of each of the object candidates, but none have the **alt** value: Login, so UFT ignores this property and moves on to the next one.
4. UFT checks the **name** property of each of the object candidates, and finds that two of the objects (both the basic and VIP Login buttons) have the name: login. UFT filters out the other three objects from the list, and these two login buttons become the new object candidates.
5. UFT checks the **file name** property of the two remaining object candidates. Only one of them has the file name login.gif, so UFT correctly concludes that it has found the Login button and clicks it.

## Test Object Mapping for Unidentified or Custom Classes

### Relevant for: GUI tests and components

The Object Mapping Dialog Box enables you to map an object of an unidentified or custom class to a standard Windows class. For example, if your application has a button that cannot be identified, this button is learned as a generic WinObject.

You can teach UFT to identify your object as if it belonged to a standard Windows button class. Then, when you click the button while recording, UFT records the operation in the same way as a click on a standard Windows button.

When you map an unidentified or custom object to a standard object, your object is added to the list of standard Windows test object classes as a user-defined test object class. You can configure the object identification settings for a user-defined test object class just as you would any other test object class.

You should map an object that cannot be identified only to a standard Windows class with comparable behavior. For example, do not map an object that behaves like a button to the Edit class.

# How to Configure Object Identification for a Test Object Class

## **Relevant for: GUI tests and components**

This task describes steps you can perform to configure the properties that UFT uses to learn and identify objects. Perform one or more of these steps to configure object identification for a specific test object class.

Repeat the task for all relevant test object classes.

This task includes the following steps:

- ["Prerequisites" below](#)
- ["Set the properties that UFT learns for identifying an object in this test object class" below](#)
- ["Set the properties that UFT uses for Smart Identification" below](#)

## **Prerequisites**

Determine what you would like to change about how UFT identifies objects of various test object classes within your application. For details, see ["Object Identification Configuration - Overview" on page 228](#).

## **Set the properties that UFT learns for identifying an object in this test object class**

In the Object Identification Dialog Box, set the properties that are learned for the test object description:

1. Select the environment.
2. Select the test object class.
3. Set mandatory and assistive properties.
4. Select an ordinal identifier.

## **Set the properties that UFT uses for Smart Identification**

1. In the Object Identification Dialog Box, select the **Enable Smart ID** check box. The **Configure** button is enabled.
2. Click the **Configure** button to open the Smart Identification Properties Dialog Box.
3. Set the base and optional properties.
4. Set the order in which the optional properties are used.
5. If you do not want UFT to learn the Smart Identification properties, clear the **Enable Smart Identification** check box. The configuration is saved, but not used.

For more details, see ["Smart Identification" on page 237](#).

# How to Manage Identification Properties of a Test Object Class

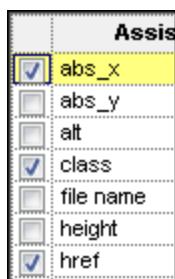
## Relevant for: GUI tests and components

The following task describes how to manage the identification properties of a specified test object class in the Add/Remove Properties Dialog Box and includes the following:

- "Add and remove properties to the object identification property lists" below
- "Add identification properties to the test object class" below

## Add and remove properties to the object identification property lists

To include a property in the **Mandatory**, **Assistive**, **Base Filter**, or **Optional Filter Properties** lists, open the Add/Remove Properties Dialog Box, select the check box next to the property name you want to add. To remove a property from the list, clear the corresponding check box.



## Add identification properties to the test object class

To add a new property to the test object class, open the Add/Remove Properties Dialog Box and click **New**. The **New Property** dialog box opens. Enter a valid property in the format `attribute/ <PropertyName>` (for Web objects) or `<PropertyName>` (for other environment objects) and click **OK**. The new property is added to the available properties list.

# How to Map an Unidentified or Custom Class to a Standard Windows Class

## Relevant for: GUI tests and components

This task describes how to map an unidentified or custom class to a standard Windows class. This instructs UFT to identify objects of the specified class in the same way as it identifies objects of the Windows class to which it is mapped.

**To map an unidentified or custom class to a standard Windows class:**

1. Open the Object Mapping Dialog Box by selecting **Standard Windows** in the **Environment** box in the Object Identification dialog box and then clicking the **User-Defined** button.
2. Click the pointing hand  and then click the object whose class you want to add as a user-defined test object class. The name of the user-defined object is displayed in the **Class name** box.
3. In the **Map to** box, select the standard object class to which you want to map your user-defined test object class and click **Add**. The class name and mapping is added to the object mapping list.
4. Click **OK**. The Object Mapping dialog box closes and your object is added to the list of standard Windows test object classes as a user-defined test object class. Note that your object has an icon with a red U in the lower-right corner, identifying it as a user-defined class.
5. Configure the object identification settings for your user defined test object class just as you would any other test object class. For details, see "[How to Configure Object Identification for a Test Object Class](#)" on page 242.

**Caution:** If you click the down arrow on the **Reset Test Object** button and select **Reset Environment**, when **Standard Windows** is selected in the **Environment** box, all of the user-defined test object classes are deleted.

# Chapter 33: Comparing and Merging Object Repositories

**Relevant for: GUI tests and components**

This chapter includes:

- [Object Repository Comparison Tool Overview](#) ..... 246
- [Object Repository Merge Tool Overview](#) ..... 247
- [Object Conflicts](#) ..... 248
- [How to Compare Two Object Repositories](#) ..... 250
- [How to Merge Two Shared Object Repositories](#) ..... 252
- [How to Update a Shared Object Repository From a Local Object Repository](#) ..... 254
- [Object Repository Comparison Tool Main Window](#) ..... 256
- [Object Repository Merge Tool Main Window](#) ..... 261

# Object Repository Comparison Tool Overview

## Relevant for: GUI tests and components

The Object Repository Comparison Tool enables you to compare two shared object repositories, and view the differences in their objects, such as different object names, different test object descriptions, and so on. The tool is accessible from the Object Repository Manager.

Differences between objects in the two object repository files are identified according to default rules. During the comparison process, the object repository files remain unchanged.

After the comparison process, the Comparison Tool provides a graphic presentation of the objects in the object repositories, which are shown as nodes in a hierarchy. Objects that have differences, as well as unique objects that are included in one object repository only, can be identified according to a color configuration that you can specify. Objects that are included in one object repository only are indicated in the other object repository by the text "Does not exist". You can also view the properties and values of each object that you select in either object repository.

You can use the information displayed by the Object Repository Comparison Tool when managing or merging object repositories.

This section also includes:

- ["When to Use the Object Repository Comparison Tool" below](#)
- ["Understanding the Repository Panes" below](#)

## When to Use the Object Repository Comparison Tool

In addition to this tool, you can perform merge and comparison operations using the Asset Comparison Tool and the Object Repository Merge Tool. Consider the following when selecting which tool to use:

- The Object Repository Comparison Tool is designed for comparing repositories that are different, but have a set of overlapping objects. This tool is useful if you want to decide whether to merge two repositories without performing the actual merge and addressing object conflicts in the Object Repository Merge Tool. For details, see ["How to Compare Two Object Repositories" on page 250](#).
- The Asset Comparison Tool can also compare two repositories, but it is designed for comparing different versions of the same repository to identify changes between versions.

## Understanding the Repository Panes

The object repository panes of the Object Repository Comparison Tool display the hierarchies of the objects, and their properties and values, in the object repository files that you are comparing. The file path is shown above each object hierarchy.

To make it easier to see the status of an object at a glance, the text and background of object names in the object repositories are displayed using different colors, according to the type of comparison found.

The Object Repository Comparison Tool enables you to navigate the two object repositories independently. You can also resize the various panes to display only some of the objects contained in the object repositories. When using large object repositories, this can result in the various panes displaying different areas of the object repository hierarchies, making it difficult to locate and track specific objects affected by the comparison process. You can synchronize the object repositories to display the same object in both views.

## Object Repository Merge Tool Overview

### **Relevant for: GUI tests and components**

UFT enables you to merge two shared object repositories into a single shared object repository using the Object Repository Merge Tool.

This tool also enables you to merge objects from the local object repository of one or more actions or components into a shared object repository. For example, if UFT learned objects locally in a specific action in your test or in a component, you may want to add the objects to the shared object repository, so that they are available to all actions (even in different tests) or components that use that object repository.

### **When to Merge Shared Object Repositories**

When you have multiple shared object repositories that contain test objects from the same area of your application, it may be useful to combine those test objects into a single object repository for easier maintenance. You could do this by manually moving or copying objects in the Object Repository Manager. However, if you have test objects in different object repositories that represent the same object in your application, and the descriptions for these objects in the different object repositories are not identical, it may be difficult to recognize and handle these conflicts.

The Object Repository Merge Tool helps you to solve the above problem by merging two selected object repositories for you, and providing options for addressing test objects with conflicting descriptions.

Using this tool, you merge two shared object repositories (called the **primary** object repository and the **secondary** object repository), into a new, third object repository, called the **target** object repository.

Objects in the primary and secondary object repositories are automatically compared, and then added to the target object repository according to configurable rules that define the defaults for how conflicts between objects are resolved.

After the merge process, the Object Repository Merge Tool provides a graphic presentation of the original objects in the primary and secondary object repositories, which remain unchanged, as well as the objects in the merged target object repository. Objects that had conflicts are highlighted. The conflict of each object that you select in the target object repository is described in detail. The Object Repository Merge Tool provides specific options that enable you to keep the default resolution for each conflict, or modify conflict resolutions individually, according to your requirements.

For more details, see ["How to Merge Two Shared Object Repositories" on page 252](#).

**Note:** If you want to compare two shared object repositories without merging, you can use the Object Repository Comparison Tool.

## When to Update a Shared Object Repository from Local Object Repositories

You can update a shared object repository by merging local object repositories associated with specific components (via their application area), or with specific actions from one or more tests, into the shared object repository. The objects that are merged from the local object repositories are then available to any components that use that shared object repository, or to any actions that use that shared object repository in any tests.

In the merge process, the objects in the local object repository for the selected actions or components are moved to the target shared object repository (and then removed from the local object repository). The action or component steps then use the objects from the updated shared object repository.

You can view or change how conflicting objects are dealt with during the update process in the Settings dialog box in the Object Repository Merge Tool.

If you choose to merge local object repositories from more than one action or component, UFT performs multiple merges, merging each action's or component's local object repository with the target object repository one at a time, for all the actions or components in the list. You can view and modify the results of each merge if necessary.

For more details, see ["How to Update a Shared Object Repository From a Local Object Repository" on page 254](#).

## Object Conflicts

### Relevant for: GUI tests and components

Merging two object repositories can result in conflicts arising from similarities between the objects they contain.

Conflicts between objects in the primary and secondary object repositories are resolved automatically by the Object Repository Merge Tool, according to the default resolution settings that you can configure before performing the merge.

Conflicts between checkpoint or output value objects with the same name but different content are always resolved by merging both objects into the new repository and renaming one of them.

The Object Repository Merge Tool also allows you to change the way the merge was performed for each individual object that causes a conflict.

## Example

An object in the primary object repository could have the same name as an object in the secondary object repository, but have a different description. You may have defined in the default settings that in this case, the object with the more generic object description, meaning the object with fewer properties, should be added to the target object repository. However, when you review the conflicts after the automatic merge, you could decide to handle a specific conflict differently, for example, by keeping both objects.

Changes that you make to the default conflict resolution can themselves affect the target object repository by causing new conflicts. In the above example, keeping both objects would cause a name conflict. Therefore, the target object repository is updated after each conflict resolution change and redisplayed.

The Object Repository Merge Tool identifies three possible conflict types:

### Different Objects with the Same Name Conflict

An object in the primary object repository and an object in the secondary object repository have the same name, but completely different content.

You can resolve this conflict type by:

- Keeping the object added from the primary object repository only.
- Keeping the object added from the secondary object repository only.
- Keeping the objects from both object repositories. In this case, the Object Repository Merge Tool automatically renames the object that is added from the secondary file by adding an incremental numeric suffix to the name, for example, Edit\_1.
- Ignoring the object from the local object repository and keeping the object from the shared object repository (when updating a shared object repository from a local object repository).

By default, the conflict resolution settings for conflicts of this type are configured so that the target object repository takes the object from both files. The object that is added from the secondary file is renamed by adding an incremental numeric suffix to the name, for example, Edit\_1.

**Note:** Test objects with different visual relation identifier definitions are treated as objects with different descriptions.

### Identical Description Different Name Conflict (Test Objects Only)

A test object in the primary object repository and a test object in the secondary object repository have different names, but the same description properties and values.

You can resolve this conflict type by:

- Taking the test object name from the object in the primary object repository.
- Taking the test object name from the object in the secondary object repository.
- Ignoring the test object from the local object repository and keeping the test object from the shared object repository (when updating a shared object repository from a local object repository).

By default, the conflict resolution settings for conflicts of this type are configured so that the target object repository takes the object name from the primary source file.

### **Similar Description Conflict (Test Objects Only)**

A test object in the primary object repository and a test object in the secondary object repository have the same name, and they have similar, but not identical, description properties and values. One of the test objects always has a subset of the properties set of the other test object. For example, a test object named Button in the secondary object repository has the same description properties and values as a test object named Button in the primary object repository, but also has additional properties and values.

You can resolve this conflict type by:

- Keeping the test object added from the primary object repository only.
- Keeping the test object added from the secondary object repository only.
- Keeping the test objects from both object repositories. In this case, the Object Repository Merge Tool automatically renames the test object that is added from the secondary file by adding an incremental numeric suffix to the name, for example, Button\_1.
- Ignoring the test object from the local object repository and keeping the test object from the shared object repository (when updating a shared object repository from a local object repository).

By default, the conflict resolution settings for conflicts of this type are configured so that the target object repository takes the test object that has fewer identifying properties than the test object with which it conflicts.

## How to Compare Two Object Repositories

### **Relevant for: GUI tests and components**

This task describes how to compare two object repositories according to predefined settings that define how comparisons between objects are identified.

This task includes the following steps:

- "[Prerequisites](#)" on the next page
- "[Select the Shared Object Repositories to compare](#)" on the next page
- "[Analyze the initial comparison results](#)" on the next page

- "Analyze the detailed comparison results" below
- "Utilize additional tools to help you perform the comparison - Optional" below

## 1. Prerequisites

- Determine the shared object repositories you want to compare. Generally, shared object repositories should contain objects from the same application, and may have differences in objects or test object descriptions because the repositories were created at different times or under different circumstances.
- Make sure that a GUI test or component is currently open.
- Make sure that the Object Repository Manager window is open. For details on working with the Object Repository Manager, see "[Object Repository Manager Main Window](#)" on page 222.
- Make sure the color settings are configured to match your needs.

## 2. Select the Shared Object Repositories to compare

- a. In the Object Repository Manager window, select **Tools > Object Repository Comparison Tool** to open the Object Repository Comparison Tool. The New Comparison Dialog Box opens.
- b. Specify the two object repository files you want to compare.

## 3. Analyze the initial comparison results

After the comparison is complete, you can view the results summary in the Comparison Statistics Dialog Box.

## 4. Analyze the detailed comparison results

Review and analyze the comparisons between the repositories in the Object Repository Comparison Tool Main Window.

## 5. Utilize additional tools to help you perform the comparison - Optional

- Synchronize the object repositories to display the same object in both views by clicking the **Synchronized Nodes** button.
- Filter the objects and show only the objects that you want to view using the Filter Dialog Box of the Object Repository Comparison Tool.
- Locate one or more objects in a selected object repository whose name contains a specified string using the Find Dialog Box of the Object Repository Comparison Tool.
- Adjust the colors of text and background of object names, and empty nodes representing objects that exist in the other object repository only, using the Color Settings Dialog Box of the Object Repository Comparison Tool.

# How to Merge Two Shared Object Repositories

## Relevant for: GUI tests and components

This task describes how to merge two shared object repositories according to predefined settings that define how conflicts between objects are resolved.

This task includes the following steps:

- "Prerequisites" below
- "Select the shared object repositories to merge" below
- "Analyze the initial merge results" below
- "Analyze the detailed merge results" below
- "Utilize additional tools to help you perform the comparison (Optional)" on the next page
- "Adjust object conflict resolutions" on the next page
- "Save the target object repository" on the next page
- "Associate the target object repository with actions or application areas" on the next page

### 1. Prerequisites

- Determine the shared object repositories you want to compare. You generally merge two shared object repositories that contain objects from the same application, and that may have differences in objects or test object descriptions because the repositories were created at different times or under different circumstances.
- Make sure that a GUI test or component is in focus.
- Make sure that the Object Repository Manager window is open. For details on working with the Object Repository Manager, see "[Object Repository Manager Main Window](#)" on page 222.
- Make sure the resolution and color settings are configured to match your needs.

### 2. Select the shared object repositories to merge

- a. In the Object Repository Manager window, select **Tools > Object Repository Merge Tool** to open the Object Repository Merge Tool. The New Merge Dialog Box opens.
- b. Specify the two object repository files you want to merge.

### 3. Analyze the initial merge results

After the merge is complete, you can view the results summary in the Merge Statistics Dialog Box of the Object Repository Merge Tool.

### 4. Analyze the detailed merge results

Review and analyze the merge between the repositories in the Object Repository Merge Tool Main Window.

## 5. Utilize additional tools to help you perform the comparison (Optional)

- Change the view presented by the Object Repository Merge Tool according to your working preferences, by dragging the edges of the panes to resize them, or selecting the appropriate option from the **View** menu.
- Filter the objects and show only the objects that you want to view by using the Filter Dialog Box of the Object Repository Merge Tool.
- Locate one or more objects in a selected object repository whose name contains a specified string using the Find Dialog Box of the Object Repository Merge Tool.

## 6. Adjust object conflict resolutions

If one or more of the merge resolutions does not match your needs, follow the steps below to adjust them:

- a. In the target object repository, select an object that had a conflict, as indicated by the icon to the left of the object name. The conflicting objects are highlighted in the source object repositories.

A description of the conflict and the resolution method used by the Object Repository Merge Tool is described in the Resolution Options pane. A radio button for each possible alternative resolution method is displayed. For details on each of the conflict types, see "[Object Conflicts](#)" on page 248.

- b. In the Resolution Options pane, select a radio button to choose an alternative resolution method. The target object repository is updated according to your selection and redisplayed.
- c. In the Resolution Options pane, click the **Previous Conflict** or **Next Conflict** buttons to jump directly to the next or previous conflict in the target object repository hierarchy.

## 7. Save the target object repository

When the object conflicts are resolved satisfactorily, save the new merged shared object repository. UFT saves the object repository with a **.tsr** extension in the specified location and displays the file name and path above the target object repository in the Object Repository Merge Tool window.

If you are connected to ALM, you can save your merged shared object repository in the Test Resources module of your project.

## 8. Associate the target object repository with actions or application areas

You can now associate the new merged object repository with actions or application areas, especially ones that were previously associated with the original object repositories, so that the objects in the object repository can be accessed by the tests or components.

# How to Update a Shared Object Repository From a Local Object Repository

## Relevant for: GUI tests and components

This task describes how to move the objects from local object repositories to a shared object repository.

This task includes the following steps:

- "Prerequisites" below
- "Select the shared object repository and the local repositories that you want to merge into it" below
- "Analyze the initial merge results" on the next page
- "Analyze the detailed merge results" on the next page
- "Utilize additional tools to help you perform the comparison (Optional)" on the next page
- "Adjust object conflict resolutions" on the next page
- "Save the target object repository" on the next page

### 1. Prerequisites

- Make sure that the shared object repository you want to update from the local object repositories is already associated with the repository actions or components.
- Make sure the tests or components containing the local object repositories are not part of an open solution.
- Make sure that a GUI test or component is in focus.
- Make sure that the Object Repository Manager window is open. For details on working with the Object Repository Manager, see "[Object Repository Manager Main Window](#)" on page 222.
- Make sure the resolution and color settings are configured to match your needs.

### 2. Select the shared object repository and the local repositories that you want to merge into it

- a. In the Object Repository Manager, open the shared object repository into which you want to merge the local repositories. If the object repository opened in read-only mode, select **File > Enable Editing**.
- b. Select **Tools > Update from Local Repository** to open the Update from Local Repository Dialog Box.
- c. In the Update from Local Repository Dialog Box, select the tests or components that contain the local object repositories you want to merge, and click **Update All**.

### 3. Analyze the initial merge results

View the initial merge results in the Merge Statistics Dialog Box of the Object Repository Merge Tool.

### 4. Analyze the detailed merge results

Review and analyze the detailed merge results in the Object Repository Merge Tool - Multiple Merge Window.

### 5. Utilize additional tools to help you perform the comparison (Optional)

- Change the view presented by the Object Repository Merge Tool according to your working preferences, by dragging the edges of the panes to resize them, or selecting the appropriate option from the **View** menu, as described in Object Repository Merge Tool - Multiple Merge Window.
- Filter the objects and show only the objects that you want to view by using the Filter Dialog Box of the Object Repository Merge Tool.
- Locate one or more objects in a selected object repository whose name contains a specified string using the Find Dialog Box of the Object Repository Merge Tool.

### 6. Adjust object conflict resolutions

If one or more of the merge resolutions does not match your needs, follow the steps below to adjust them:

- a. In the target object repository, select an object that had a conflict, as indicated by the icon to the left of the object name. The conflicting object is highlighted in the local object repository. A description of the conflict and the resolution method used by the Object Repository Merge Tool is described in the Resolution Options pane. A radio button for each possible alternative resolution method is displayed. For details on each of the conflict types, see "[Object Conflicts](#)" on page 248.
- b. In the Resolution Options pane, select a radio button to choose an alternative resolution method. The target object repository is updated according to your selection and redisplayed.
- c. In the Resolution Options pane, click the **Previous Conflict** or **Next Conflict** buttons to jump directly to the next or previous conflict in the target object repository hierarchy.

### 7. Save the target object repository

When the object conflicts are resolved satisfactorily, save the new merged shared object repository. UFT saves the object repository with a **.tsr** extension in the specified location and displays the file name and path above the target object repository in the Object Repository Merge Tool window.

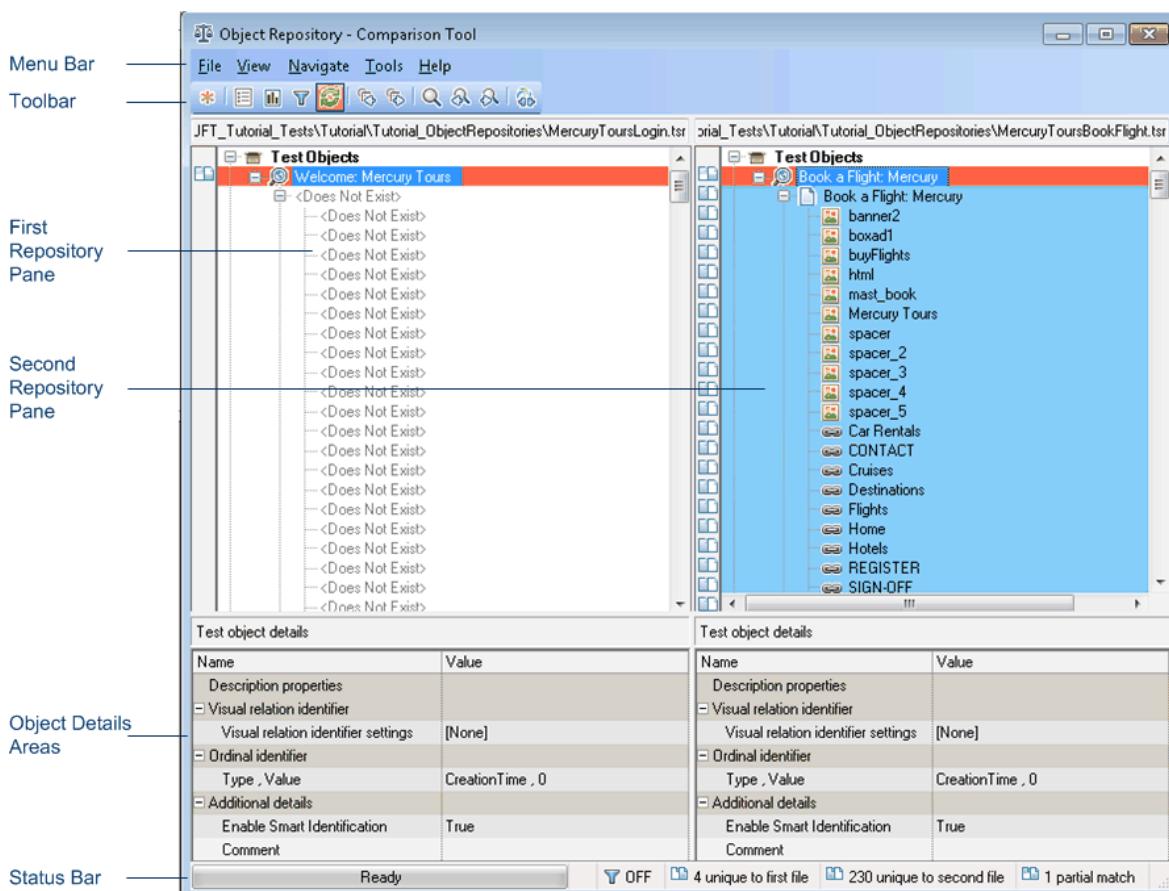
If you are connected to ALM, you can save your merged shared object repository in the Test Resources module of your project.

**Note:** The objects that are merged into the shared object repository are **removed** from the local object repositories. The steps in the actions or components then use the objects from the updated shared object repository.

## Object Repository Comparison Tool Main Window

### Relevant for: GUI tests and components

This window displays the two repositories selected for comparison and provides tools for analyzing the comparison.



To access	In the <b>Object Repository Manager</b> , select <b>Tools &gt; Object Repository Comparison Tool</b> .
Important information	<ul style="list-style-type: none"> <li>You cannot work with the Object Repository Manager or the Object Repository Merge Tool while the Object Repository Comparison Tool is open.</li> <li>The Object Repository Comparison Tool gives precedence to matching test object descriptions over the matching of test object names.</li> </ul>

**Relevant tasks**["How to Compare Two Object Repositories" on page 250](#)

The Object Repository Comparison Tool window contains the following key elements:

- "Menu Commands and Toolbar Buttons" below
- "Repository Panes" on the next page
- "Test Object Details Areas" on page 259
- "Status Bar" on page 259

## Menu Commands and Toolbar Buttons

### File Menu

Command	Shortcut Key	Function
 <b>New Comparison</b>	CTRL+N	Opens the New Comparison Dialog Box, enabling you to specify two object repositories on which to perform a new comparison operation.
 <b>ALM Connection</b>		Enables you to connect the Object Repository Comparison Tool to an ALM project.
<b>Exit</b>		Closes the Object Repository Comparison Tool window.

### View Menu

Command	Function
 <b>Statistics</b>	Opens the Comparison Statistics Dialog Box, which summarizes the comparison between the two repositories, including the number and type of any differences found.
<b>Collapse All</b>	Collapses the entire hierarchy in both comparison panes.  <b>Tip:</b> While <b>Synchronized Nodes</b> is on, double-clicking an expanded node collapses it in both panes simultaneously.
<b>Expand All</b>	Expands the entire hierarchy in both comparison panes.  <b>Tip:</b> Double-clicking a collapsed node expands it in both panes simultaneously.

### Navigate Menu

Command	Shortcut Key	Function
 <b>Next Difference</b>	F4	Finds the next difference between objects in the object repositories.
	SHIFT+F4	Finds the previous difference between objects in the object repositories.

Command	Shortcut Key	Function
<b>Previous Difference</b>		
 <b>Find</b>	CTRL+F	Opens the Find Dialog Box (Object Repository Comparison Tool).
 <b>Find Next</b>	F3	Finds the next object in the object repositories according to the search specifications in the Find dialog box.
 <b>Find Previous</b>	SHIFT+F3	Finds the previous object in the object repositories according to the search specifications in the Find dialog box.

### Tools Menu

Command	Function
 <b>Synchronized Nodes</b>	Enables you to navigate the two object repository panes simultaneously or independently of one another.
 <b>Filter</b>	Opens the Filter Dialog Box (Object Repository Comparison Tool), enabling you to specify the comparison types that you want to show.
 <b>Color Settings</b>	Opens the Color Settings Dialog Box (Object Repository Comparison Tool), enabling you to specify the text and background color of the object names and empty nodes displayed in the comparison panes.

### Help Menu

Command	Shortcut Key	Function
Object Repository Comparison Tool Help	F1	Opens the Object Repository Comparison Tool Help.

## Repository Panes

The repository panes display a hierarchical view of the objects in the object repositories being compared. The differences and similarities between objects in the two panes are indicated by colored text and background colors for each object.

Differences can also be identified by the icons displayed to the left of the objects in the object repository panes, as follows:

UI Elements	Description
	The number of objects that are unique to the first file.
	The number of objects that are unique to the second file.

UI Elements	Description
	The number of objects in the first and second file that are not identical, but partially match.

The object repository panes provide the following functionality:

- While in **Synchronized Nodes** mode, when you select an object in one object repository pane, the corresponding object in the other file hierarchy is located and highlighted. You can press the **CTRL** button when you select an object to highlight only the selected object without highlighting the corresponding object in the other file.
- When you select an object in an object repository pane, its properties and values are displayed in the respective **Test object details** area at the bottom of the pane.
- When you position your cursor over an icon to the left of an object in an object repository pane, the comparison details are displayed as a tooltip, for example, Partial match, or Unique to second file.
- You can expand or collapse the hierarchy of a parent node by double-clicking the node, or by clicking the expand (+) or collapse (-) symbol to the left of the node name. You can also expand or collapse the entire hierarchy in the object repository pane by choosing **Collapse All** or **Expand All** from the **View** menu.
- You can jump directly to the next or previous difference in the object repository hierarchy by choosing **Next Difference** or **Previous Difference** from the **Navigate** menu, by clicking the **Next Difference** or **Previous Difference** buttons in the toolbar, or by using keyboard shortcuts. For details about shortcuts, see "["Menu Commands and Toolbar Buttons" on page 257](#)".
- You can drag the edges of the panes to resize them in the Object Repository Comparison Tool window.

## Test Object Details Areas

Shows the properties and values of the object selected in an object repository pane. For details, see "["Understanding the Repository Panes" on page 246](#)".

## Status Bar

Shows the status of the comparison process and details of the comparisons found during the object repository comparison.

User interface elements of the status bar are described below (unlabeled elements are shown in angle brackets):

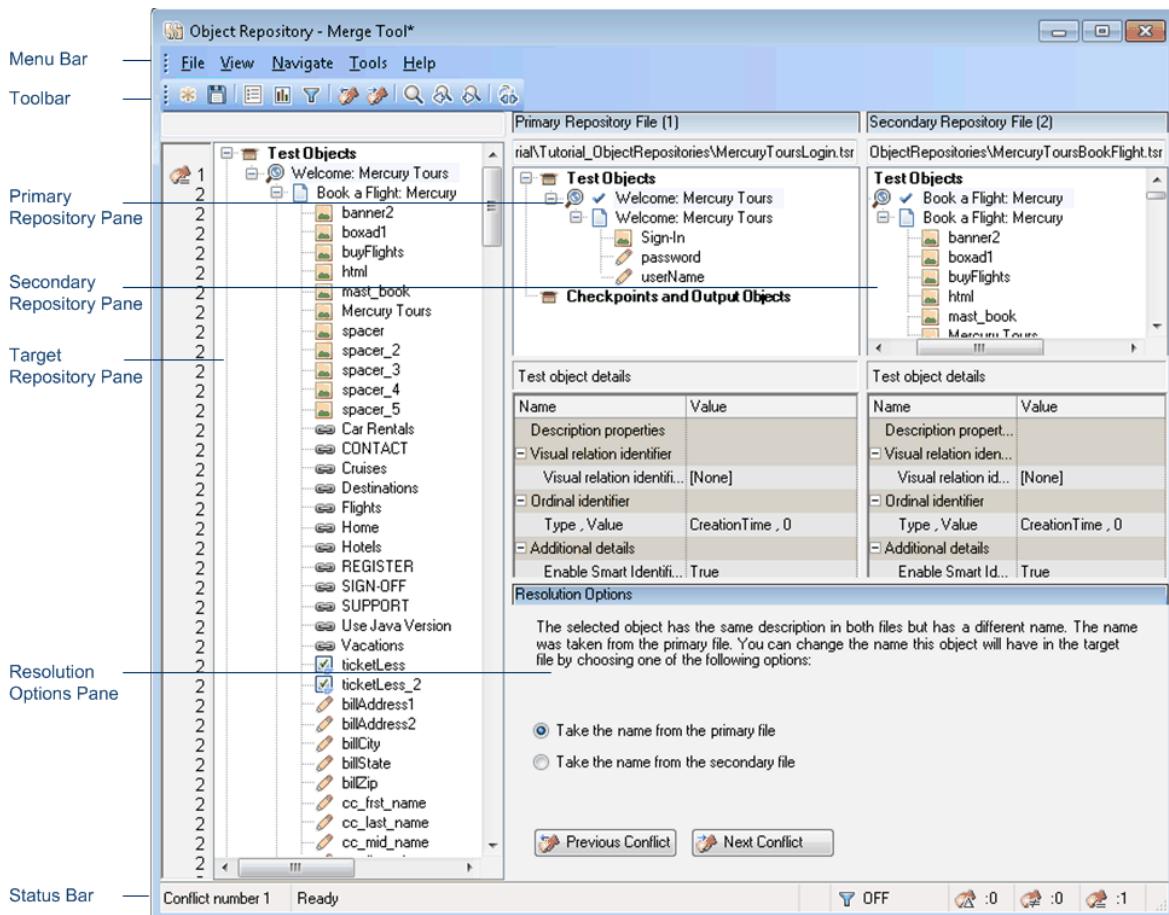
UI Elements	Description
<b>&lt;progress bar&gt;</b>	Displays the comparison process status on the left of the status bar. <b>Ready</b> is displayed when the process is complete.
	<b>ALM Connection.</b> Displayed only when UFT is connected to an ALM project.

UI Elements	Description
<filter>	<p>Opens the Filter Dialog Box (Object Repository Comparison Tool). The icon image indicates the filter status of the repository panes.</p> <p>Possible values:</p> <ul style="list-style-type: none"><li>•  OFF . Indicates that the object repositories are not filtered and all objects are shown.</li><li>•  ON . Indicates a filter is active and that some objects may have been filtered out of the display.</li></ul>
	The number of objects that are unique to the first file.
	The number of objects that are unique to the second file.
	The number of objects in the first and second file that are not identical, but partially match.

# Object Repository Merge Tool Main Window

## Relevant for: GUI tests and components

This window displays the two repositories selected for merging and the target repository containing the merged content. This window also provides tools for analyzing the merge and resolving conflicts.



To access	In the <b>Object Repository Manager</b> , select <b>Tools &gt; Object Repository Merge Tool</b> .
Important information	<ul style="list-style-type: none"> <li>You cannot work with the Object Repository Manager or the Object Repository Comparison Tool while the Object Repository Merge Tool is open.</li> <li>Test objects that do not have description properties, such as Page or Browser objects, are compared by name only. If the same object is contained in both the source object repositories but with different names, they will be merged into the target object repository as two separate objects.</li> </ul>
Relevant tasks	<a href="#">"How to Merge Two Shared Object Repositories" on page 252</a>

The Object Repository Merge Tool main window contains the following key elements:

- "Menu Bar and Toolbar" below
- "Target Repository Pane" below
- "Primary and Secondary Repository Panes" on the next page
- "Resolution Options Pane" on the next page
- "Status Bar" on page 264

## Menu Bar and Toolbar

Displays menus with Object Repository Merge Tool commands.

## Target Repository Pane

The target object repository pane displays a hierarchy of the objects, as well as their respective properties and values, that were merged from the primary and secondary object repositories. In the column to the left of the object hierarchy, the pane displays the source file of each object (1 is displayed for the primary file and 2 for the secondary file), and an icon representing the type of conflict, if any.

When you save the target object repository, the file path is displayed above the object hierarchy.

**Note:** To make it easier to see the status of an object at a glance, the text colors of the object names in the target object repository can be set according to their source and whether they caused a conflict.

Merging two object repositories can result in a target object repository containing a large number of objects. To make navigation and the location of specific objects easier in the target object repository pane, the Object Repository Merge Tool enables you to filter the objects in the pane and show only the objects that had conflicts that were resolved during the merge.

The target object repository pane provides the following functionality:

- When you select an object in the target object repository, the corresponding object in the primary and/or secondary source file hierarchy is located and indicated by a check mark.
- When you select an object in the target object repository, its properties and values are displayed in the **Object Properties - Target File** area at the bottom of the target object repository pane (**View > Target Repository Object Properties**).
- If the merge results in a conflict, an icon is displayed to the left of the conflicting object in the target object repository. You can see a tooltip description of the conflict type by positioning your pointer over the icon.
- When you right-click an object, a context-sensitive menu opens. You can expand an option or collapse the entire hierarchy in the target object repository, or, when applicable, you can change the conflict resolution method and result.
- You can expand or collapse the hierarchy of the node by double-clicking a node. You can also expand or collapse the entire hierarchy in the target object repository by choosing **Collapse All** or **Expand All** from the **View** menu.

- You can jump directly to the next or previous conflict in the target object repository hierarchy by choosing **Next Conflict** or **Previous Conflict** from the **Navigate** menu, or by clicking the **Next Conflict** or **Previous Conflict** buttons  in the toolbar or Resolution Options pane.
- You can locate one or more objects in the target object repository by using the "Find Dialog Box (Object Repository Merge Tool)".
- You can show or hide the target object repository object properties by choosing **View > Target Repository Object Properties**.

## Primary and Secondary Repository Panes

The primary and secondary object repository panes display the hierarchies of the objects, and their properties and values, in the original source object repositories that you chose to merge. The file path is shown above each object hierarchy.

The panes provide the following functionality:

- You can expand or collapse the hierarchy of a selected item by double-clicking the item.
- You can view the properties and values of an object in the **Test object details** area by selecting it in the relevant pane.
- You can show or hide the panes by selecting or clearing **Primary Repository** or **Secondary Repository** in the **View** menu.

## Resolution Options Pane

The Resolution Options pane provides information about any conflict encountered during the merge for the object selected in the target object repository. The pane also provides options that enable you to keep or change the conflict resolution method that was applied using the default resolution options.

The Resolution Options pane provides the following functionality:

- When you select a conflicting object in the target object repository, the pane displays a textual description of the conflict and the resolution method used by the Object Repository Merge Tool. A choice of alternative resolution methods is offered.
- You can select a radio button to choose an alternative resolution method for the conflict. Every time you make a change, the target object repository is automatically updated and is redisplayed.
- You can jump directly to the next or previous conflict in the target object repository hierarchy by clicking the **Previous Conflict** or **Next Conflict** buttons.
- For a local object repository merge, you can click the **Ignore Object** button to exclude a specific local object repository object from the merge process. The object remains in the local object repository when the merge is complete.
- You can show or hide the pane by selecting or clearing **Resolution Options** in the **View** menu.

## Status Bar

The status bar shows the following UI elements (unlabeled elements are shown in angle brackets):

UI Elements	Description
<b>Conflict number</b>	The conflict number (if any) of the object selected in the target object repository pane.
<b>&lt;progress bar&gt;</b>	Displayed during the merge process. <b>Ready</b> is displayed when the merge is complete.
	<b>Filter.</b> Displays the filter status of the target object repository pane. Possible states: <ul style="list-style-type: none"> <li>• <b>ON.</b> Indicated that a filter is currently in use.</li> <li>• <b>OFF.</b> indicates that the object repositories are not filtered and all objects are shown.</li> </ul> <p><b>Note:</b> Click the <b>Filter</b> icon to open the Filter Dialog Box (Object Repository Merge Tool).</p>
	<b>ALM Connection.</b> Displayed when UFT is connected to an ALM project.
	<b>Similar Description Conflict.</b> For details on object conflicts, see " <a href="#">Object Conflicts</a> " on page 248.
	<b>Same Name Different Description Conflict.</b> For details on object conflicts, see " <a href="#">Object Conflicts</a> " on page 248.
	<b>Same Description Different Name Conflict.</b> For details on object conflicts, see " <a href="#">Object Conflicts</a> " on page 248.

# Chapter 34: Extending UFT Object Identification

**Relevant for: GUI tests and components**

This chapter includes:

• Object Identification Challenges in UFT .....	266
• Identifying Objects Using Insight .....	266
• Creating Insight test objects .....	267
• Creating steps with Insight Test Objects .....	267
• Running tests with Insight .....	268
• Learn more! .....	268
• How to Work with Insight Test Objects .....	268
• Add an Insight object .....	269
• Modify an Insight test object's image .....	270
• Retrieve text from an Insight Object .....	270
• Update Insight test object details .....	270
• UI Automation in UFT .....	271
• Enable UI Automation support .....	272
• How Does UFT Use the UI Automation Framework? .....	272
• When Should You Use UFT UI Automation Support? .....	274
• Native UI Automation Methods .....	279
• How to Use UFT UI Automation Support .....	281
• Learn objects in UI Automation mode and add them to the object repository .....	281
• Record test/component steps in UI Automation mode .....	282
• Identifying Unsupported Objects in Test Runtime .....	282
• UI Automation Support Limitations .....	285

# Object Identification Challenges in UFT

UFT uses an object's identification properties to locate it in an application, and then maps these properties to a test object type.

UFT provides numerous types of test objects across many technologies. However, there are times when UFT cannot locate or learn an object in your application.

For example:

- UFT does not support the technology, or the technology version
- UFT cannot uniquely identify the object
- UFT identifies the object at too basic a level, such as identifying a table control as a general Object

In cases such as these, use one of the following methods to create test objects, and by extension meaningful functional tests:

- **Insight**, an image-based object recognition method. For details, see "[Identifying Objects Using Insight](#)" below.
- Support for the Microsoft **UI Automation** framework. For details, see "[UI Automation in UFT](#)" on page 271.

## Identifying Objects Using Insight

### **Relevant for: GUI tests and components**

Insight enables UFT to recognize objects in your application based on what they look like, instead of properties that are part of their design. This can be useful if you are working with an application whose technology is not supported by UFT, or with an application running on a remote computer.

With Insight, UFT stores an image of the object with the Insight test object (along with ordinal identifiers, if necessary), and uses this image as the main description property to identify the object in the application.

You can create **InsightObject** test objects during recording sessions, or when adding test objects to an object repository manually.

When adding objects to an object repository, you can add an object directly from the application, or even from a picture of the object displayed on your screen.

In the object repository, you can edit the object's properties, such as its name and image, and the default location to click in the control when performing test object methods. You can also define visual relation identifiers to improve UFT's ability to accurately identify the object.

**Note:** Insight is supported only on the primary monitor.

## Creating Insight test objects

UFT learns Insight objects using the following elements:

<b>Description property</b>	UFT stores an image of the object for the description property. This image is used later to identify the object.
<b>Ignore areas</b>	If parts of the object do not always look the same, you can instruct UFT to ignore those areas when it uses the image to identify the object.
<b>Object configuration</b>	UFT can use an ordinal identifier to create a unique description for the object. Other aspects of object configuration, such as mandatory and assistive properties, and smart identification, are not relevant for Insight test objects.
<b>Visual relation identifiers</b>	After UFT creates a description for an Insight test object, add visual relation identifiers to improve identification of the object.
<b>Similarity identification property</b>	Add the <b>similarity</b> identification property to the test object description. This property is a percentage that specifies how similar a control in the application has to be to the test object image for it to be considered a match.

The Insight object is always added to the object repository as a child of the test object that represents its containing application, such as a Window or Browser object. (The new object's parent object is also added if it does not already exist in the object repository.)

**Note:** Insight test objects require more disk space than other test objects, because of the test object images and the snapshots stored with the test objects.

To control the amount of space used, limit the size of the snapshot in the Insight Pane of the Options Dialog Box.

After you add all of the relevant test objects and finish modifying them to your satisfaction in the object repository, you can delete all of the snapshots to reduce the amount of disk space used.

## Creating steps with Insight Test Objects

Create steps using Insight test objects in much the same way as you would with other types of test objects.

In the Editor, the test object image is displayed in the step instead of the test object name. When you hold the cursor over the image, an enlarged view of the image is displayed. Double-click the image to open the object repository with the **InsightObject** selected.

```
Window("Calculator").InsightObject().Click
Window("Calculator").InsightObject().Click
Window("Calculator").InsightObject().Click
```

**Tip:** To show these images, select the relevant option in the Options dialog box (**Tools > Options > GUI Testing** tab > **Insight** pane).

For details on the methods and properties supported by Insight test objects, see the **Insight** section of the *HP UFT Object Model Reference for GUI Testing*.

## Running tests with Insight

During a run session, UFT searches for the Insight test object that matches the image stored with the test object.

UFT searches for the matching object within the Insight object's parent test object. You can help focus the search on a smaller area by creating smaller parent objects and building a hierarchy of Insight test objects in your object repository.

UFT's image matching algorithm allows for some variation, enabling UFT to recognize the object even if it changed slightly. However, the algorithm is not based on object properties, and therefore does not use the smart identification mechanism.

**Note:** Steps with Insight objects may take longer than usual to run, especially if there are many similar objects with the same parent.

## Learn more!

- "[How to Record a GUI Test or Component](#)" on page 99
- "[How to Work with Insight Test Objects](#)" below
- "[Test Object Descriptions](#)" on page 168

## How to Work with Insight Test Objects

### **Relevant for: GUI tests and components**

Add Insight objects to the object repository directly from the application, or even from a picture of the object displayed on your screen.

This task includes the following steps:

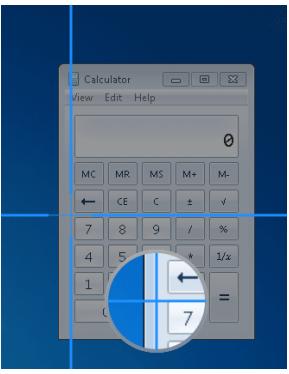
- "[Add an Insight object](#)" on the next page
- "[Modify an Insight test object's image](#)" on page 270
- "[Retrieve text from an Insight Object](#)" on page 270
- "[Update Insight test object details](#)" on page 270

## Add an Insight object

1. Click the **Add Insight Object to Local** toolbar button .
2. In the Select Learn Mode dialog box, select the mode you want for learning the Insight object, and then select the control.

**Note:** If you previously selected **Do not show me again** on the Select Learn Mode dialog box, the learning session automatically begins using the mode you used most recently.

To display the Select Learn Mode dialog box, enable this option in the Options dialog box (**Tools > Options > GUI Testing tab > Insight node**).

<b>Automatic</b>	<p>The pointer changes to a pointing hand. Click on the control in the application. UFT automatically detects the borders of the control, and takes a snapshot of it. This mode is the faster mode and should be satisfactory in most cases.</p>
<b>Manual</b>	<p>The pointer changes to a crosshair, with an adjacent circle displaying a magnified image of the area around the center of the crosshair.</p> <p>For example:</p>  <p>Take a snapshot of the control in the application, manually specifying the borders of the control. Use this mode in cases where the automatic mode does not correctly detect the borders of the control, such as when it selects an area of the application which is much larger than the control.</p> <p><b>Tip:</b> By holding the left <b>CTRL</b>, you can temporarily change the pointing hand or crosshair to a standard pointer. This enables you to change the window focus or perform operations in UFT or in your application.</p>

UFT takes a snapshot of the control, and the Add Insight Test Object dialog box opens.

3. In the Add Insight Test Object dialog box, you can:
  - **Adjust the borders** of the image saved with the test object in the object repository.
  - **Take a new snapshot** to replace the image entirely.
  - **Specify areas to exclude** from the test object image. UFT will ignore these areas when it

searches for the image on the screen to identify the object.

- **Modify the test object's ClickPoint.** This is the location to click in the control when running a test object method on it.

An InsightObject, named **InsightObject**, is added to the object repository, under the test object that represents the application or window that contains the control.

## Modify an Insight test object's image

1. In the Object Repository window or Manager, select the test object whose image you want to modify.  
If you are in the Editor, double-click the test object's image in a step.
2. In the **Test object image** area, click the **Change Test Object Image**  button.

## Retrieve text from an Insight Object

Use the **Insight.GetVisibleText** test object method to retrieve text displayed on the object. UFT uses the OCR mechanism to recognize and return the text.

Use this text for verification purposes, or as a way of differentiating between objects or states of the application.

For example:

- If the text on a button in your application changes when an operation succeeds, check that text to verify success.  
Make sure to use "exclude areas" to ignore the text area in the Insight object definition.
- If you have two similar objects in your application, that are different only because of their text, learn them both as the same object, using "exclude areas" to ignore the text in the image.  
Then, use **GetVisibleText** to check the text on the object and differentiate between the two objects in your test or component.

For more details, see the **Insight** section of the *HP UFT Object Model Reference for GUI Testing*.

## Update Insight test object details

Do any of the following to improve the readability and efficiency of your test or component:

- **Rename** the test object to a name that describes the control it represents. (Recommended)
- **Move** the test object within the test object hierarchy:

If you place it under another test object...

...UFT searches for the object in the application only within its parent

	test object.
If you move the Insight test object to be a top-level object...	...UFT searches for the object anywhere on the screen.

- **Add a similarity identification property** to the test object description.

For details, see the *InsightObject* identification properties topic in the *HP UFT Object Model Reference for GUI Testing*.

- **Modify the ordinal identifier** created for the test object. For details, see "[Ordinal Identifiers](#)" on [page 230](#).
- **Define visual relation identifiers** for the test object. For details, see "[Visual Relation Identifiers](#)" on [page 234](#).

For more details , see "[How to Maintain Test Objects in Object Repositories](#)" on [page 202](#).

**Tip:** When you have finished modifying all of the Insight test objects, delete all of the snapshots to reduce the amount of disk space used. This does not delete the test object images used for object identification.

Select **Tools > Delete Insight Snapshots**.

## UI Automation in UFT

### Relevant for: GUI tests and components

Microsoft UI Automation is a framework that enables you to access, identify, and manipulate UI elements of any application by providing programmatic access to these user interface elements.

The UI Automation API enables this access by using the [IUIAutomationElement interface](#) to make each element into a separate object. You can then view the properties and operations of each of the objects in the application.

UFT uses the different parts of the framework to create both test objects based on your application, and the object's supported test object methods.

Use the following elements to understand the framework:

<b>Element tree</b>	The hierarchy of elements in the application, which displays a logical division and hierarchy of all user interface elements in the application
<b>Control Type property</b>	The appearance and functionality of the object.
<b>Control Patterns</b>	<p>These patterns also contain methods specific for the pattern. Control patterns are a way to categorize and expose a control's function independent of the control type or the appearance of the control.</p> <p>There is no one-to-one matching of the control type property to control patterns - each control type can support multiple types of patterns and each pattern can be used by multiple control types.</p>

For full details on the UI Automation framework, see the [UI Automation section on MSDN](#).

## Enable UI Automation support

Use the UFT UI Automation support with any Windows-based application that has implemented UI Automation provider interfaces. The support is loaded as with other add-ins, by selecting **UI Automation** in the Add-ins Manager when starting UFT.

**Note:** UFT support has been validated with Telerik UI for Windows Forms, Telerik UI for .NET WPF, and JavaFX.

When in use, UFT UI Automation support overrides other technology support. UI Automation support can be used with existing technology support, but not at the same time.

## How Does UFT Use the UI Automation Framework?

UFT uses the UIAutomation framework elements to:

- Create UI Automation **test objects** based on the objects in your application
- Create supported **methods** for these test objects based on the patterns supported for each object/control type
- Create the **test object hierarchy** based on the UI Automation element tree

## Control Types and UFT Test Objects

For a control that uses the UI Automation framework, the Control Type property describes the basic appearance and functionality of the control in the application.

UFT translates the Control Type property of an element/object in the application's user interface into a corresponding test object in UFT:

If the Control Type property is...	UFT creates this test object:
<b>50000</b>	<i>UIAButton</i>
<b>50001</b>	<i>UIACalendar</i>
<b>50002</b>	<i>UIACheckBox</i>
<b>50003</b>	<i>UIACComboBox</i>
<b>50004</b>	<i>UIAEdit</i>
<b>50005</b>	<i>UIAHyperLink</i>
<b>50008</b>	<i>UIAList</i>
<b>50013</b>	<i>UIARadioButton</i>

<b>50015</b>	<i>UIASlider</i>
<b>50018</b>	<i>UIATab</i>
<b>50023</b>	<i>UIATree</i>
<b>50028</b>	<i>UIATable</i>
<b>50031</b>	<i>UIASplitButton</i>
<b>50032</b>	<i>UIAWindow</i>
<b>50036</b>	<i>UIATable</i>  <b>Note:</b> This object must also have implemented the Grid pattern to be recognized as a table.

If the controls in your application use a different value for the Control Type property, or do not have a Control Type property implemented, they are identified as a UIAObject.

For a full listing of all the available values for the Control Type property, see the [Control Type Identifiers page on MSDN](#).

## Supported Patterns and Test Object Methods

UFT creates test object methods based on a control type's supported patterns. These patterns define a particular aspect of a control's functionality or feature. For a full explanation of how these patterns are used in your applications, see [https://msdn.microsoft.com/en-us/library/ms752362\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/ms752362(v=vs.110).aspx).

Each test object also supports all UFT common methods and properties, as well as additional UI Automation-specific methods, including **.Click**, **.SetFocus** and **.Type**. A number of test objects also have object-specific test object methods available for use.

For full details on these test object methods, see the UI Automation section of the *HP UFT Object Model Reference for GUI Testing*.

**Note:** The test objects and methods available are completely dependent on the properties and patterns implemented in your application. It is recommended that you familiarize yourself with the properties of your application's objects - specifically the Control Type IDs and supported patterns to understand what test objects and methods you can use.

UFT creates test object methods based on the patterns:

If a object has this pattern...	UFT has these test object methods:
<b>ExpandCollapse</b>	<ul style="list-style-type: none"> <li><i>.Expand</i></li> <li><i>.Collapse</i></li> </ul>
<b>Grid</b>	<i>.GetItem</i>
<b>Invoke</b>	<i>.Click</i>

<b>RangeValue</b>	<ul style="list-style-type: none"> <li>• <i>.Decrement</i></li> <li>• <i>.Increment</i></li> <li>• <i>.SetValue</i></li> </ul>
<b>Scroll</b>	<ul style="list-style-type: none"> <li>• <i>.Scroll</i></li> <li>• <i>.ScrollDown</i></li> <li>• <i>.ScrollLeft</i></li> <li>• <i>.ScrollRight</i></li> <li>• <i>.ScrollDown</i></li> <li>• <i>.SetScrollPercent</i></li> </ul>
<b>ScrollItem</b>	<i>.ScrollIntoView</i>
<b>Selection</b>	<ul style="list-style-type: none"> <li>• <i>.Select</i></li> <li>• <i>.AddToSelection</i></li> <li>• <i>.RemoveFromSelection</i></li> <li>• <i>.GetSelection</i></li> </ul>
<b>SelectionItem</b>	<ul style="list-style-type: none"> <li>• <i>.Select</i></li> <li>• <i>.AddToSelection</i></li> <li>• <i>.RemoveFromSelection</i></li> </ul>
<b>Table</b>	<ul style="list-style-type: none"> <li>• <i>.GetColumnHeaders</i></li> <li>• <i>.GetRowHeaders</i></li> </ul>
<b>TableItem</b>	<ul style="list-style-type: none"> <li>• <i>.GetColumnHeaderItems</i></li> <li>• <i>.GetRowHeaderItems</i></li> </ul>
<b>Text</b>	<i>.GetText</i>
<b>Transform</b>	<ul style="list-style-type: none"> <li>• <i>.Move</i></li> <li>• <i>.Resize</i></li> <li>• <i>.Rotate</i></li> </ul>
<b>Toggle</b>	<i>.Set</i>
<b>Value</b>	<i>.SetValue</i>
<b>Window</b>	<ul style="list-style-type: none"> <li>• <i>.Maximize</i></li> <li>• <i>.Minimize</i></li> <li>• <i>.Restore</i></li> <li>• <i>.Close</i></li> </ul>

## When Should You Use UFT UI Automation Support?

Use UFT UI Automation support as follows:

- Create tests exclusively of UI Automation test objects
- Mix UI Automation objects and regular test objects (such as WPF, or Windows Forms)
- Use UI Automation support only when regular object identification is no sufficient for your testing needs.

For example, use UI Automation support in the following scenarios:

**• When UFT's regular object identification support is not sufficient for testing your application**

Because UFT identifies UI Automation objects based on Control Types and supported patterns, the object identification can differ from other standard Windows-based object identification.

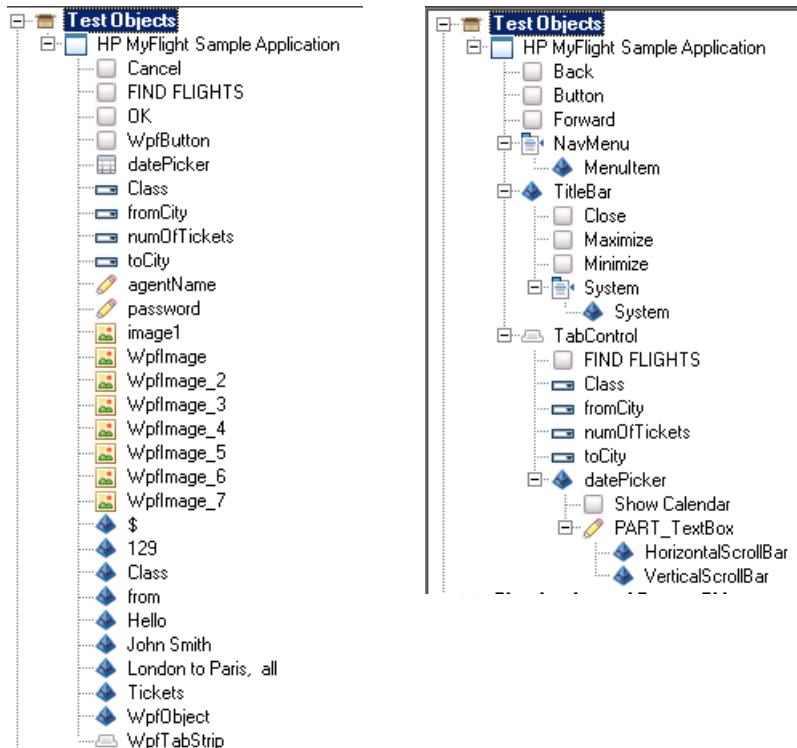
This can mean:

### The test object hierarchy might be different

Because objects are identified by UFT from a mapping of Control Type to a specific test object, the types and relations between objects can be very different.

For example, when viewing the learned hierarchy of objects in the Flight Finder page of the HP MyFlight sample application, you get very different views of the overall structure.

#### Regular WPF object identification    UI Automation object identification



### The same object might be identified completely differently

Objects can be seen as completely different types of objects.

In this example, you have an object in which an application has a corporate directory displayed in a searchable grid:

**Corporate Directory**

Drag a column header and drop it here to group by that column

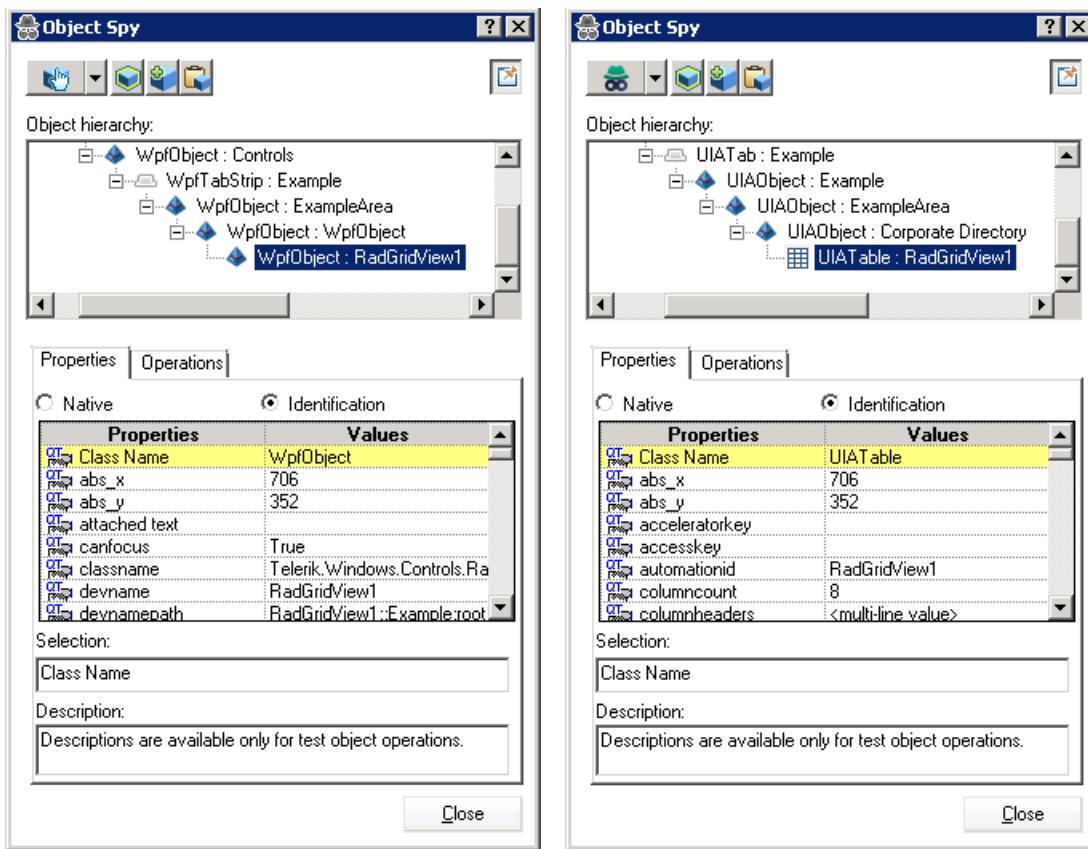
	Photo	First Name	Last Name	Title	City	Country	Phone	
		Nancy	Davolio	Sales Representative	Seattle	USA	(206) 555-9857	
+		Andrew	Fuller	Vice President, Sales	Tacoma	USA	(206) 555-9482	
+		Janet	Leverling	Sales Representative	Kirkland	USA	(206) 555-3412	
+		Margaret	Peacock	Sales Representative	Redmond	USA	(206) 555-8122	
+		Steven	Buchanan	Sales Manager	London	UK	(71) 555-4848	
+		Michael	Suyama	Sales Representative	London	UK	(71) 555-7773	
+		Robert	King	Sales Representative	London	UK	(71) 555-5598	
+		Laura	Callahan	Inside Sales Coordinator	Seattle	USA	(206) 555-1189	
+		Anne	Dodsworth	Sales Representative	London	UK	(71) 555-4444	

When UFT uses the Spy to view the main area of this window, the results are very different.

When using WPF object identification you get the general `WpfObject` for the window (which is functionally a grid control). However, UI Automation identifies this as a `UIATable` instead. In this case, the UI Automation object identification enables you to get a clearer object identification that is more in line with the functional design of the application.

#### As a WPF object

#### As a UI Automation object



Use UI Automation when regular object identification is not sufficient, and UI Automation object identification is more in line with the functional design of the application.

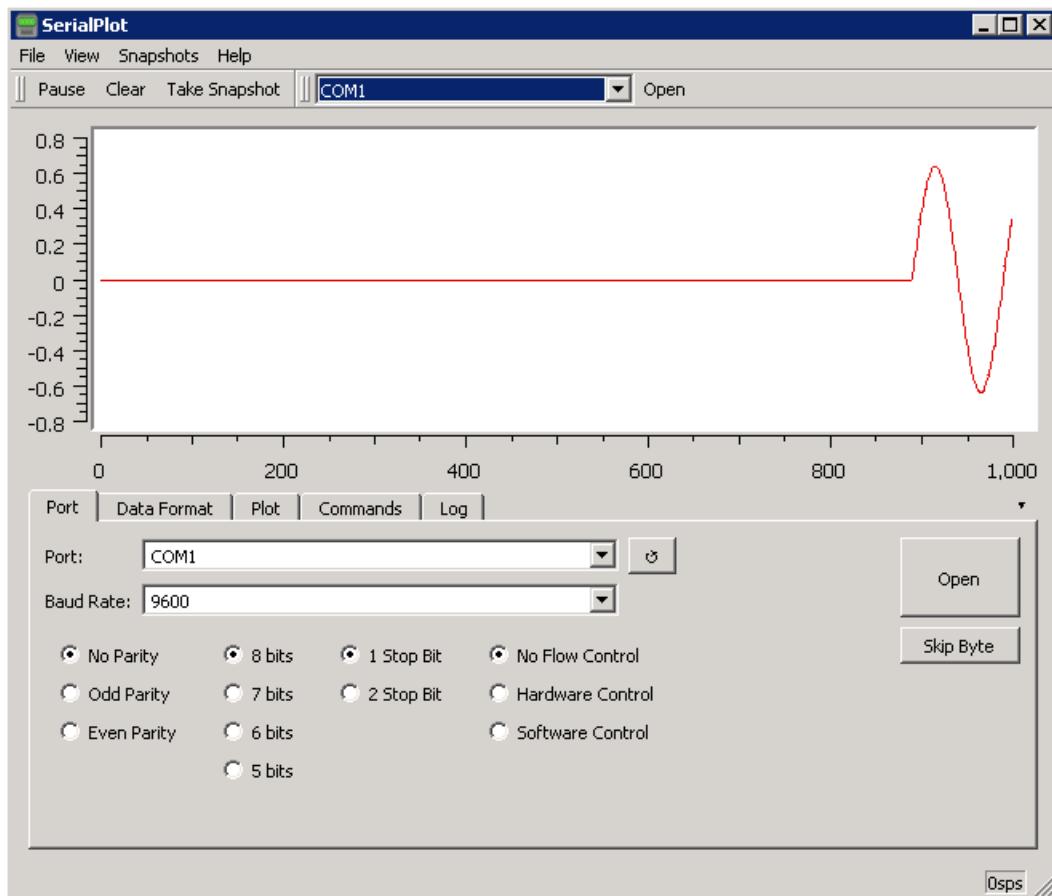
- **When UFT does not support your technology or your version of a technology**

As different technology frameworks expand their abilities and functionalities, UFT may not adequately identify the application objects, either in type or functionality. In this case, using UI Automation enables you to adequately identify and test the application.

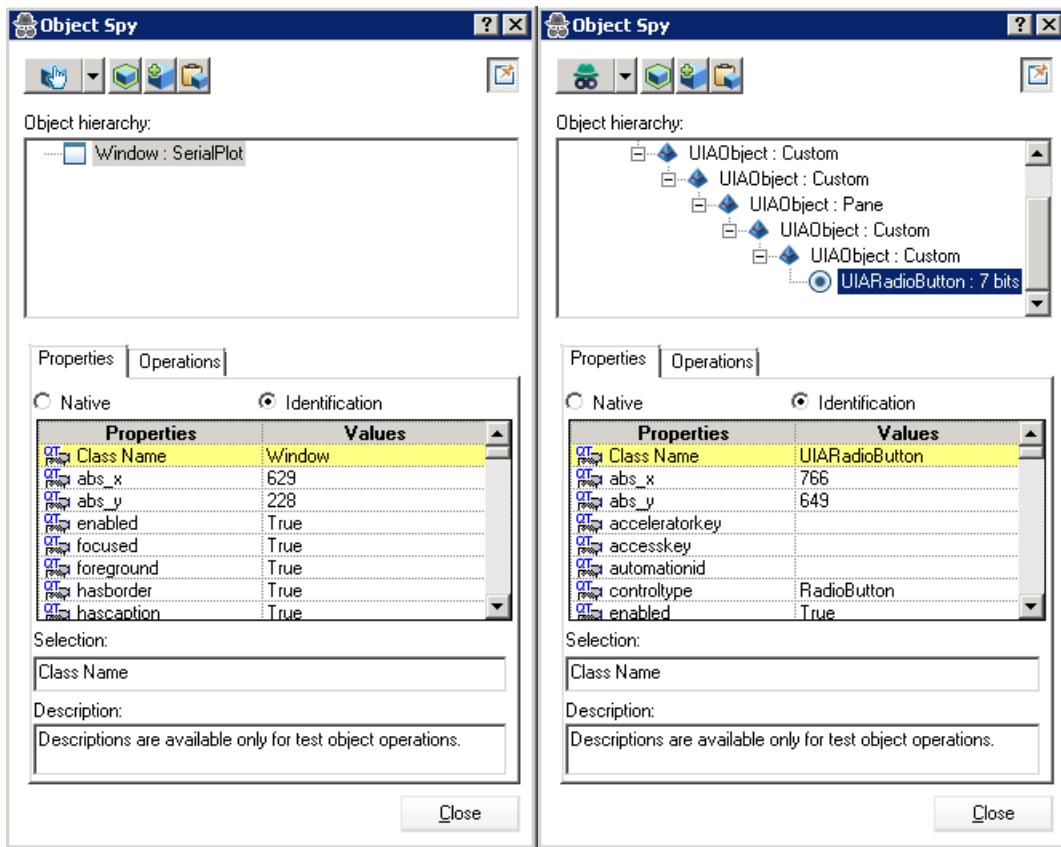
Use UI Automation when it will enable you to identify objects in your application when UFT otherwise could not.

## Example

You have an application that provides data on COM ports:



When spying on this application with regular UFT support, UFT is unable to identify or learn any of the objects. However, using UI Automation, you can identify individual objects:



## Native UI Automation Methods

### Relevant for: GUI tests and components

UFT UI Automation support provides support for a number of native methods that you can access using the **.Object** method. These methods are available for all UI Automation object.

Method	Description
<b>Compare</b>	<p>Checks if an element is equal to the selected object.</p> <p><b>Syntax</b></p> <pre>.Object.CompareElement</pre> <p><b>Parameters</b></p> <p><b>Element:</b> The element to which to compare the selected element</p>
<b>ElementFromPoint</b>	<p>Finds an object at the selected coordinates.</p> <p><b>Syntax</b></p> <pre>.Object.ElementFromPointx,y</pre>

Method	Description
	<p><b>Parameters</b></p> <ul style="list-style-type: none"> <li><b>X:</b> The x-coordinate at which to find the object.</li> <li><b>Y:</b> The y-coordinate at which to find the object.</li> </ul>
<b>GetChildren</b>	<p>Returns a list of all children of the selected object.</p> <p><b>Syntax</b></p> <pre>.Object.GetChildren</pre>
<b>GetFirstChild</b>	<p>Returns the first child object of the selected object. If there are no child objects, UFT returns a NULL value.</p> <p><b>Syntax</b></p> <pre>.Object.GetFirstChild</pre>
<b>GetLastChild</b>	<p>Returns the last child object of the selected object. If there are no child objects, UFT returns a NULL value.</p> <p><b>Syntax</b></p> <pre>.Object.GetChildren</pre>
<b>GetNextSibling</b>	<p>Returns the element which is next (in the overall hierarchy) to the selected object.</p> <p><b>Syntax</b></p> <pre>.Object.GetNextSibling</pre>
<b>GetParent</b>	<p>Returns the parent element for a selected object.</p> <p><b>Syntax</b></p> <pre>.Object.GetParent</pre>
<b>GetPreviousSibling</b>	<p>Returns the element which is prior (in the overall hierarchy) to the selected object.</p> <p><b>Syntax</b></p> <pre>.Object.GetPreviousSibling</pre>

**Note:** When using the **.Object** method for an object, UFT returns the value “as-is” (which can be complex with several different flags). You can find the value of these numerical properties on MSDN (e.g. for the State property, you can find its value [here](#)).

For additional details on how to use **.Object** with UFT, see:

- "Native Properties and Operations" on page 663
- The .Object property in the *HP UFT Object Model Reference for GUI Testing*

## How to Use UFT UI Automation Support

### Relevant for: GUI tests and components

This task describes how to properly use UFT's UI Automation support, which helps you identify objects in your application when UFT's regular object identification support is not sufficient for your needs.

**Note:** Before you use UFT's UI Automation support, you must:

- Have an application that implements Microsoft UI Automation patterns. For details on support, see the [UI Automation overview on MSDN](#).
- Load **UI Automation** in the Add-in Manager when starting UFT

UFT's UI Automation support uses existing object identification functionality (such as Object Spy, Navigate and Learn, and the like). However, each of these object identification tools must be used in UI Automation mode. To use the support, do the following:

- "[Learn objects in UI Automation mode and add them to the object repository](#)" below
- "[Record test/component steps in UI Automation mode](#)" on the next page

### Learn objects in UI Automation mode and add them to the object repository

1. Activate the UI Automation mode in UFT before learning the objects.
  - a. In the Object Repository or Object Repository Manager toolbar, click the **Object Spy** button .
  - b. In the pointing hand drop-down list, click the down arrow and select **UI Automation**. The pointing hand is changed to a hat icon .

Objects in your application are now identified as UI Automation objects, even if the object type is supported by regular UFT object identification.

Regular UFT object identification support for the loaded add-ins is disabled.
2. Click the pointing hand button (with the hat icon), and then click the object in your application. The Object Spy pauses while examining the application, and then displays the test object hierarchy.

**Note:** It may take longer than usual to display the test object hierarchy of UI Automation object. Do not interact with your application or the Object Spy during this time.

3. Close the Object Spy.
4. Add objects to your object repository using one of the following:

<b>In the Object Spy</b>	Add Object to Repository button 
<b>In the Object Repository window</b>	Add Objects to Local button 
<b>In the Object Repository Manager</b>	<a href="#">Navigate and Learn toolbar</a>

## Record test/component steps in UI Automation mode

1. In the toolbar, click the **Record** button .
2. In the Record Toolbar, from the Recording mode drop-down list, select **UI Automation Recording**. All steps performed are now recorded as UI Automation objects, even if the object type can be recognized as another regular UFT test object.

**Note:**

- If you are recording to add a checkpoint or output value, ensure that the UI Automation Recording mode is selected *before* you click the **Insert Checkpoint or Output Value** button 
- The speed of UI Automation recording can vary depending on the application.

## Identifying Unsupported Objects in Test Runtime

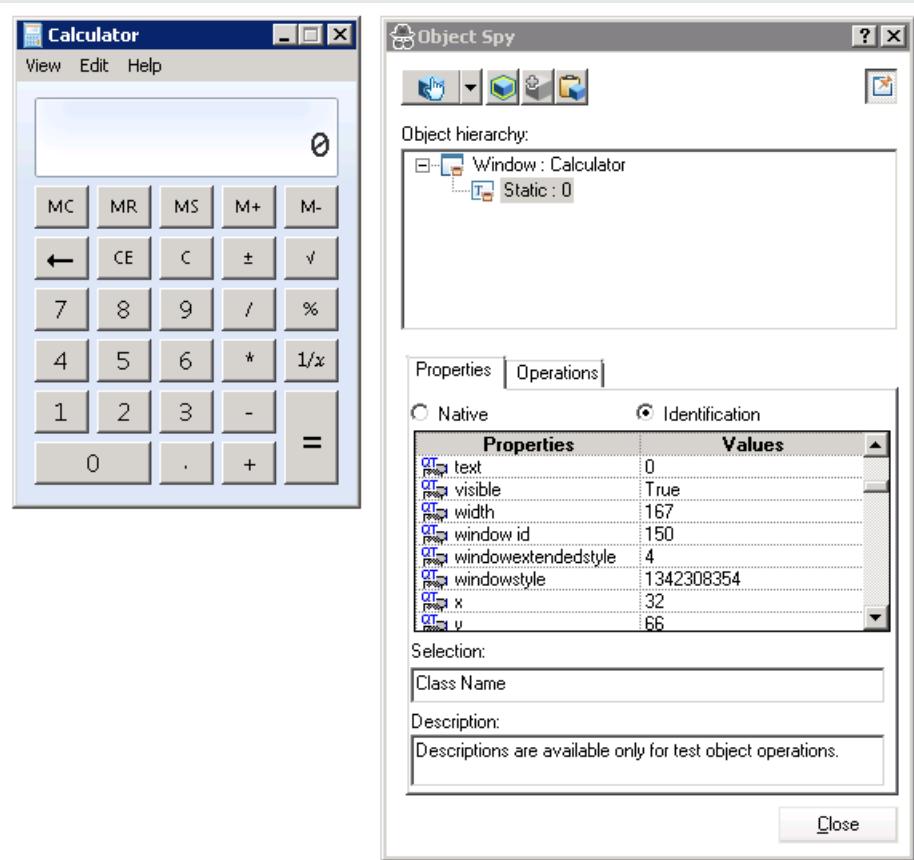
UFT may fail to find a test object during a test run because the expected properties are incorrect, and may fail to identify and learn an object because it is not a supported object for a specific technology.

In such cases, use a Static test object or static identification properties to identify the object, as follows:

- "[Create a programmatic description of the object](#)" on the next page
- "[Set the unsupported/unidentified object as Static object type](#)" on page 284
- "[Assign the unsupported/unidentified object to a supported object type](#)" on page 284
- "[Use non-test object methods](#)" on page 284

Static objects exist in the application, but typically cannot be selected or have data entered into them automatically. Since UFT does support Static objects, assigning the Static object type to an unidentifiable object enables you to select the object in your application.

The examples in this topic use the window display in a standard Windows calculator. This window is identified as a **Static** object, with a **window id** property of 150.



## Create a programmatic description of the object

1. Create a programmatic description to specify the exact property of the unknown object that you want UFT to identify.

The property you specify must be a real UFT test object identification property and must use a real value. You can find this value in the Object Spy.

2. Use a **Description.Create** statement to create a Properties collection object.
3. Set the value of the identification property using a static programmatic description, and a statement to set the value.

This Properties collection object can now be identified by UFT.

For example, with the Calculator, use a **Description.Create** statement to specify that you are looking for an object in which the **window id** property value is 150:

```
Set des = Description.Create
des="window id:= 150"
```

## Set the unsupported/unidentified object as Static object type

Once you have created a Properties collection object and specified the value of the object, assign this "object" as an object of **Static** type.

For example, with the Calculator, assign UFT to find the object with the **window id** value of 150 by assigning this "object" as a Static type:

```
Set des = Description.Create
des="window id:= 150"
Window("Calculator").Static(des).Click
```

## Assign the unsupported/unidentified object to a supported object type

Additionally, assign the unsupported or unidentified object to a supported object type to perform a specific method (such as **.Click** or **.Submit**). This enables UFT to run the step, as UFT thinks it is using a supported object type.

For example, with the Calculator, with the Properties collection object created with the **Description.Create** statement, you can assign it to a WinButton object and click the different buttons:

```
Set des = Description.Create
des="text:=1"
Window("Calculator").WinButton(des).Click

Set des1 = Description.Create
des1="text:=2"
Window("Calculator").WinButton(des1).Click
```

## Use non-test object methods

Run methods that are not supported for a specific UFT test object by assigning the Properties collection object to a supported UFT test object that supports events.

For example, with the Calculator, you can highlight the display:

```
Set des = Description.Create
des="window id:=150"
Window("Calculator").Static(des).Highlight
```

This will highlight the test object during the test run.

# UI Automation Support Limitations

<b>Unsupported features</b>	The following features are not supported together with UI Automation: <ul style="list-style-type: none"> <li>• Active Screen</li> <li>• Drag and drop operations</li> <li>• F1 support for objects and methods</li> <li>• Record and Run Settings</li> <li>• Utility objects</li> <li>• Table checkpoints</li> <li>• Using UI Automation with the UFT Add-in for ALM</li> </ul>
<b>UI Automation on Windows 10</b>	UI Automation recording and UI Automation spy mode can experience unexpected behavior when running UFT on Windows 10. (Note that these functionalities are not officially supported for UFT running on Windows 10.)
<b>Recording on environments with slow GUI refresh</b>	If you are using UI Automation for recording in an environment with a slow GUI refresh (such as a slow RDP connection), recording can experience unexpected and unstable behavior.
<b>UI Automation and Java applications</b>	Due to changes in the Java framework, when using UI Automation to test a Java application, objects added to your object repository from an application running on a computer with Java 8 installed will not be recognized on a computer with Java 6 installed.
<b>Regular expressions as property values</b>	The <b>controltype</b> identification property cannot have a value that is a regular expression.
<b>UIA container objects with large numbers of elements</b>	When spying on UIA container objects (such as lists, trees, and tables) with large numbers of elements in the object, UFT may experience slow performance and unexpected behavior. This is due to the fact that the <b>ItemContainer</b> pattern with the <b>FindItemByProperty</b> method has not been implemented for the object in the application.  <b>Workaround:</b> Do one or more of the following: <ul style="list-style-type: none"> <li>• In your application, implement the <b>ItemContainer</b> pattern with the <b>FindItemByProperty()</b> method.</li> <li>• Manually Add additional identification properties for the items in the object, such as index (as these properties are not added by default).</li> <li>• Use .Object methods to access items within the control.</li> </ul>
<b>UIA with other "spy" tools</b>	If you start other "spy" tools, such as the Microsoft Inspect tool, UFT experiences unexpected behavior.
<b>Using the Function Definition Generator with UI Automation</b>	UI Automation test objects do not have any available operations for the object's supported patterns in the Function Definition Generator. The only available operations are the UFT common methods and properties.  <b>Workaround:</b> Register a function manually (for the supported pattern methods) using the <b>RegisterUserFunc</b> statement.

# Chapter 35: Virtual Objects

**Relevant for: GUI tests and scripted GUI components**

This chapter includes:

- [Virtual Objects - Overview](#) ..... 287
- [How Virtual Objects are Defined and Recognized](#) ..... 287
- [How to Define Virtual Objects for Unsupported Objects in Your Test or Scripted Component](#) ..... 288

# Virtual Objects - Overview

## **Relevant for: GUI tests and scripted GUI components**

Your application may contain objects that behave like standard objects but are not recognized by UFT. You can define these objects as virtual objects and map them to standard classes, such as a button or a check box. UFT emulates the user's action on the virtual object during the run session. In the run results, the virtual object is displayed as though it is a standard class object.

For example, suppose you want to test a Web page containing a bitmap that the user clicks. The bitmap contains several different hyperlink areas, and each area opens a different destination page. When you create the test or scripted component, the Web site matches the coordinates of the click on the bitmap and opens the destination page.

To enable UFT to click at the required coordinates during a run session, you can define a virtual object for an area of the bitmap, which includes those coordinates, and map it to the button class. When you run the test or scripted component, UFT clicks the bitmap in the area defined as a virtual object so that the Web site opens the correct destination page.

Virtual object collections are groups of virtual objects that are stored in the Virtual Object Manager under a descriptive name.

The virtual object collections displayed in the Virtual Object Manager are stored on your computer and not with the tests or scripted components that contain virtual object steps. This means that if you use a virtual object in a step, the object is recognized during the run session only if it is run on a computer containing the appropriate virtual object definition. To copy your virtual object collection definitions to another computer, copy the contents of your <UFT installation folder>\dat\VoTemplate folder (or individual .vot collection files within this folder) to the same folder on the destination computer.

**Note:** UFT does not support virtual objects for analog or low-level recording. For details on low-level recording, see ["Creating Tests or Components" on page 1123](#).

## How Virtual Objects are Defined and Recognized

## **Relevant for: GUI tests and scripted GUI components**

UFT identifies a virtual object according to its boundaries. Marking an object's boundaries specifies its size and position on a Web page or application window. When you assign a test object as the parent of your virtual object, you specify that the coordinates of the virtual object boundaries are relative to that parent object. When you record a test or scripted component, UFT recognizes the virtual object within the parent object and adds it as a test object in the object repository so that UFT can identify the object during the run session. UFT also recognizes the virtual object as a test object when you add it manually to the object repository.

To perform an operation in the Active Screen on a marked virtual object, you must first record it, so that its properties are saved in the test object description in the object repository. If you perform an operation in the Active Screen on a virtual object that has not yet been recorded, UFT treats it as a standard object.

You can use virtual objects only when recording and running a test or scripted component. You cannot insert any type of checkpoint on a virtual object, or use the Object Spy to view its properties.

You can enable and disable recognition of virtual objects during recording, in the **General** pane of the GUI Testing tab in the Options dialog box (**Tools > Options > GUI Testing tab > General** node).

During a run session, make sure that the application window is the same size and in the same location as it was during recording, otherwise the coordinates of the virtual object relative to its parent object may be different, and this may affect the success of the run session.

## How to Define Virtual Objects for Unsupported Objects in Your Test or Scripted Component

### Relevant for: GUI tests and scripted GUI components

This task describes how to define virtual objects for objects that are not normally recognized by UFT.

This task includes the following steps:

- "Display the object containing the area you want to define as a virtual object" below
- "Use the Virtual Object wizard to define your virtual object" below

#### 1. **Display the object containing the area you want to define as a virtual object**

With UFT open (but not in record mode), open your application and display the object containing the area you want to define as a virtual object.

You can define virtual objects only for objects on which UFT records **Click** or **DblClick** methods. Otherwise, the virtual object is ignored. For example, if you define a virtual object over the **WinList** object, the **Select** operation is recorded, and the virtual object is ignored. UFT does not support virtual objects for analog or low-level recording. For details on low-level recording, see "[Frequently Asked Questions for GUI Testing](#)" on page 1122.

#### 2. **Use the Virtual Object wizard to define your virtual object**

Open the Virtual Object wizard (**Tools > Virtual Object > New Virtual Object**). The Virtual Object wizard includes the following pages:

- Map to a Standard Class Page (Virtual Object Wizard)
- Mark Virtual Object Page (Virtual Object Wizard)
- Object Configuration Page (Virtual Object Wizard)
- Save Virtual Object Page (Virtual Object Wizard)

# Chapter 36: Checkpoints in GUI Testing

**Relevant for: GUI tests and components**

This chapter includes:

• <a href="#">Checkpoints Overview</a> .....	290
• <a href="#">Adding Existing Checkpoints to a Test</a> .....	290
• <a href="#">Simple and Advanced Mode</a> .....	291
• <a href="#">Checkpoint Types</a> .....	291
• <a href="#">Standard Checkpoints Overview</a> .....	293
• <a href="#">Accessibility Checkpoints Overview</a> .....	293
• <a href="#">Bitmap Checkpoints Overview</a> .....	294
• <a href="#">Database Checkpoints Overview</a> .....	298
• <a href="#">File Content Checkpoints Overview</a> .....	299
• <a href="#">Table Checkpoints Overview</a> .....	300
• <a href="#">Page Checkpoints Overview</a> .....	300
• <a href="#">Checking Text Overview</a> .....	301
• <a href="#">XML Checkpoints Overview</a> .....	307
• <a href="#">How to Insert a Checkpoint in a GUI Test or Component</a> .....	309
• <a href="#">How to Include and Ignore Areas When Comparing a Bitmap - Use-Case Scenario</a> .....	315
• <a href="#">How to Configure Text Recognition Settings</a> .....	318
• <a href="#">Checkpoint Properties Dialog Box</a> .....	320
• <a href="#">Add Existing Checkpoint Dialog Box</a> .....	321
• <a href="#">Troubleshooting and Limitations - Using Checkpoints</a> .....	322

# Checkpoints Overview

## Relevant for: GUI tests and components

UFT enables you to add checks to your test or component. A **checkpoint** is a verification that compares the current value for specified properties or current state of other characteristics of an object with the expected value or characteristics. This helps you to identify whether your application is functioning correctly. For example, you can perform standard checkpoints to check that the actual object property values conform to the expected values, and you can perform bitmap checkpoints to check that the visible parts of your application are displayed correctly.

When you add a checkpoint, UFT inserts a checkpoint step to the current row in the Keyword View, and for tests and scripted components, also adds a **Check CheckPoint** statement in the Editor. By default, UFT names the checkpoint using the name of the test object on which the checkpoint was created. You can choose to specify a different name for the checkpoint or accept the default name.

When you run the test or component, UFT compares the expected results of the checkpoint to the current results. If the results do not match, the checkpoint fails. You can view the results of the checkpoint in the run results.

To learn more, see

- [Adding Existing Checkpoints to a Test](#) ..... 290
- [Simple and Advanced Mode](#) ..... 291

## Adding Existing Checkpoints to a Test

### Relevant for: GUI tests and scripted GUI components only

UFT enables you to reuse existing checkpoints. When you create checkpoints, consider which checkpoints can be reused in multiple locations in your test or in multiple tests. For example:

- Checkpoints that check generic content or the state of your application may be useful in multiple locations.
- Checkpoints that check the content of a specific area of your application are generally useful in only one particular place in your test.

The following examples illustrate situations in which inserting an existing checkpoint may be useful:

- If each page of your application contains your organization's logo, you can reuse a bitmap checkpoint to verify each occurrence in the application.
- If your application contains multiple edit boxes, you can reuse a checkpoint to confirm the **enabled** status of these edit boxes throughout your test.

## Simple and Advanced Mode

### Relevant for: Keyword GUI components only

UFT provides two modes of checkpoints that you can insert using the relevant Checkpoint Properties dialog box—**Simple Mode** and **Advanced Mode**. Simple Mode enables you to check a basic set of properties and values. This type of checkpoint is visible and editable in both UFT and ALM. For details on the Checkpoint Properties dialog box, see "[Checkpoint Properties Dialog Box](#)" on page 320.

In UFT, you can also view or edit advanced checkpoint properties that are not visible to users in ALM. You do this by inserting a checkpoint using Advanced Mode. If a checkpoint contains advanced properties, and an ALM user views its properties in ALM, a disclaimer opens indicating that some properties are checked but are not shown.

## Checkpoint Types

### Relevant for: GUI tests and components

You can insert the following checkpoint types to check objects in an application:

Checkpoint Type	Description
<b>Standard Checkpoint</b>	<p>Checks property values of an object in your application. For example, you can check that a radio button is activated after it is selected or you can check the value of an edit box.</p> <p>Standard checkpoints are supported for all add-in environments (see "<a href="#">Supported Checkpoints</a>" on page 1117).</p> <p>For details on standard checkpoints, see "<a href="#">Standard Checkpoints Overview</a>" on page 293.</p>
<b>Image Checkpoint (tests and scripted components only)</b>	<p>Checks the value of an image in your application. For example, you can check that a selected image's source file is correct.</p> <p>You create an image checkpoint by inserting a standard checkpoint on an image object.</p> <p>Image checkpoints are supported for the Web add-in environment (see "<a href="#">Supported Checkpoints</a>" on page 1117).</p> <p>For details on image checkpoints, see "<a href="#">Standard Checkpoints Overview</a>" on page 293.</p>
<b>Accessibility Checkpoint (tests and scripted components only)</b>	<p>Identifies areas of your Web site that may not conform to the World Wide Web Consortium (W3C) Web Content Accessibility Guidelines. For example, guideline 1.1 of the W3C Web Content Accessibility Guidelines requires you to provide a text equivalent for every non-text element. You can add an Alt property check to check whether objects that require the Alt property under this guideline, do in fact have this tag.</p> <p>Accessibility checkpoints are supported for the Web add-in environment (see "<a href="#">Supported Checkpoints</a>" on page 1117).</p> <p>For details on accessibility checkpoints, see "<a href="#">Accessibility Checkpoints Overview</a>" on page 293.</p>
<b>Bitmap Checkpoint</b>	<p>Checks an area of your application as a bitmap. For example, suppose you have a Web site that can display a map of a city the user specifies. The map has control keys for zooming. Using the bitmap checkpoint, you can check that the map zooms in correctly.</p>

Checkpoint Type	Description
	<p>You can also check that a specific bitmap exists in your application. For example, you can check that your company logo is displayed anywhere on your Web page.</p> <p>You can create a bitmap checkpoint for any area in your application.</p> <p>Bitmap checkpoints are supported for all add-in environments. For details, see "<a href="#">Supported Checkpoints</a>" on page 1117.</p> <p>For details on bitmap checkpoints, see "<a href="#">Bitmap Checkpoints Overview</a>" on page 294.</p>
<b>Database Checkpoint (tests and scripted components only)</b>	<p>Checks the contents of a database accessed by your application. For example, you can use a database checkpoint to check the contents of a database containing flight information for your Web site.</p> <p>Database checkpoints are supported for all add-in environments (see "<a href="#">Supported Checkpoints</a>" on page 1117).</p> <p>For details on database checkpoints, see "<a href="#">Database Checkpoints Overview</a>" on page 298.</p>
<b>File Content Checkpoint (tests only)</b>	<p>Checks the text in a dynamically generated (or accessed) file. For example, suppose your application generates a .pdf. You can check that the correct text is displayed on specific lines in or specific pages in that .pdf.</p> <p>File content checkpoints are supported for all add-in environments (see "<a href="#">Supported Checkpoints</a>" on page 1117).</p> <p>For details on file content checkpoints, see "<a href="#">File Content Checkpoints Overview</a>" on page 299.</p>
<b>Page Checkpoint (tests and scripted components only)</b>	<p>Checks the characteristics of a Web page. For example, you can check how long a Web page takes to load or whether a Web page contains broken links.</p> <p>You create a page checkpoint by inserting a standard checkpoint on a page object.</p> <p>Page checkpoints are supported for the Web add-in environment (see "<a href="#">Supported Checkpoints</a>" on page 1117).</p> <p>For details on page checkpoints, see "<a href="#">Page Checkpoints Overview</a>" on page 300.</p>
<b>Table Checkpoint (tests and scripted components only)</b>	<p>Checks information within a table. For example, suppose your application contains a table listing all available flights from New York to San Francisco. You can add a table checkpoint to check that the time of the first flight in the table is correct.</p> <p>You create a table checkpoint by inserting a standard checkpoint on a table object. For details on table checkpoints, see "<a href="#">Table Checkpoints Overview</a>" on page 300.</p> <p>Table checkpoints are supported for all add-in environments that have a *Table test object. Table checkpoints are also supported for some list view objects, such as WinListView and VbListView, as well as other list view objects in add-in environments. For details, see "<a href="#">Supported Checkpoints</a>" on page 1117.</p>
<b>Text Checkpoint (tests and scripted components only)</b>	<p>Checks that a text string is displayed in the appropriate place in an application. For example, suppose a Web page displays the sentence Flight departing from New York to San Francisco. You can create a text checkpoint that checks that the words "New York" are displayed between "Flight departing from" and "to San Francisco".</p> <p>Text checkpoints are supported for most add-in environments (see "<a href="#">Supported Checkpoints</a>" on page 1117).</p> <p>For details on text checkpoints, see "<a href="#">Checking Text Overview</a>" on page 301.</p>
<b>Text Area Checkpoint</b>	<p>Checks that a text string is displayed within a defined area in a Windows-based application, according to specified criteria. For example, suppose your Visual Basic application has a button that says View Doc &lt;Num&gt;, where &lt;Num&gt; is replaced by the four digit code entered in a form elsewhere in the application. You can</p>

Checkpoint Type	Description
<b>(tests and scripted components only)</b>	<p>create a text area checkpoint to confirm that the number displayed on the button is the same as the number entered in the form.</p> <p>Text area checkpoints are supported for all Windows-based environments, such as Standard Windows, Visual Basic, and ActiveX add-in environments (see "<a href="#">Supported Checkpoints</a>" on page 1117). Text area checkpoints are also supported for some other add-in environments, such as Java.</p> <p>For details on text area checkpoints, see "<a href="#">Checking Text Overview</a>" on page 301.</p>
<b>XML Checkpoint (tests and scripted components only)</b>	<p>Checks the data content of .xml documents in .xml files or .xml documents in Web pages and frames. For details on XML checkpoints, see "<a href="#">XML Checkpoints Overview</a>" on page 307</p> <p>The <b>XML Checkpoint (Web Page/Frame)</b> option is supported for the Web add-in environment. The <b>XML Checkpoint</b> option is supported for all add-in environments (see "<a href="#">Supported Checkpoints</a>" on page 1117).</p>

## Standard Checkpoints Overview

### Relevant for: GUI tests and components

You can check the object property values in your application using standard checkpoints. Standard checkpoints compare the expected values of object properties to the object's current values during a run session. You can create standard checkpoints for all supported testing environments (as long as the appropriate add-ins are loaded).

You can check that a specified object in your application has the property values you expect, by adding a standard checkpoint step to your test or component while recording or editing it. To set the options for a standard checkpoint, you use the "[Checkpoint Properties Dialog Box](#)" (described on page 320).

**For tests and scripted components:** You can also use standard checkpoints to perform checks on images, tables, Web page properties, and other objects within your application.

## Accessibility Checkpoints Overview

### Relevant for: GUI tests and scripted GUI components

You can add accessibility checkpoints to help you quickly identify areas of your Web site that may not conform to the W3C Web Content Accessibility Guidelines.

The Section 508 criteria for Web-based technology and information systems are based on access guidelines developed by the Web Accessibility Initiative of the World Wide Web Consortium (W3C).

You can add automatic accessibility checkpoints to each page in your test, or you can add individual accessibility checkpoints to individual pages or frames.

Accessibility checkpoints are not supported for keyword components.

## Automatic Accessibility Checkpoints

You can instruct UFT to create automatic accessibility checkpoints for every page in all tests. For details, see "[How to Insert a Checkpoint in a GUI Test or Component](#)" on page 309.

## Individual Accessibility Checkpoints

If you do not select to add accessibility checkpoints automatically while recording, you can add an accessibility checkpoint while recording or editing your test.

For details, see:

- "[How to Insert a Checkpoint in a GUI Test or Component](#)" on page 309
- "[Checkpoint Properties Dialog Box](#)" on page 320

## Bitmap Checkpoints Overview

### Relevant for: GUI tests and components

UFT enables you to check that the visible parts of your application are displayed correctly by comparing bitmaps of objects in your application to bitmaps captured previously and stored with the test or component.

You can create bitmap checkpoints for all supported testing environments (as long as the appropriate add-ins are loaded).

Bitmap checkpoints enable you to do the following:

- **Compare an entire object or areas within an object.** For example, suppose you have a Web site that can display a map of a city that the user specifies. The map has control keys for zooming. You can zoom in on a map, and then insert a bitmap checkpoint on the zoomed-in map to check that the map zooms in correctly.
- **Locate a specified image within an object.** For example, suppose you want to check that your company logo is displayed on your Web page. You can either select the logo in the actual Web page, or load a bitmap file containing the logo from your computer.

For details on defining checkpoint settings, see "[Checkpoint Properties Dialog Box](#)" on page 320.

## How UFT Compares Bitmaps

When you create a bitmap checkpoint, UFT captures the **visible** part of the specified object as a bitmap and inserts a checkpoint in the test or component. (UFT does not capture any part that is scrolled off the screen, or hidden by another object, for example.)

You can specify areas of the object to ignore or include in the checkpoint. For example, if your Web page includes a dynamic counter that may cause the checkpoint to fail, you can instruct UFT to ignore it during the run session by excluding the area in which it is located from the comparison.

When you run the test or component, UFT captures a bitmap of the actual object in the application and compares this runtime bitmap (or the selected areas within it) with the bitmap stored in the checkpoint. You can fine-tune this comparison by defining tolerance settings in the checkpoint. For details, see ["Fine-Tuning the Bitmap Comparison" on the next page](#).

If there are differences, UFT saves the runtime bitmap and displays it next to the expected bitmap in the run results. You can also view a bitmap that reflects the difference between the two bitmaps, to assist you in identifying the nature of the discrepancy.

## How UFT Locates Bitmaps

When you create a bitmap checkpoint, you select a bitmap that you want UFT to locate during a run session. You can either select an area within the bitmap captured from your application, or load a bitmap file from the file system and store it in the checkpoint. UFT then inserts the checkpoint in the test or component.

When you run the test or component, UFT searches for the specified bitmap within the runtime bitmap captured from the application. You can fine-tune this search by defining similarity criteria in the checkpoint. For details, see ["Fine-Tuning the Bitmap Comparison" on the next page](#).

If the specified bitmap is not found, the checkpoint fails, and UFT displays the specified bitmap next to the actual (runtime) bitmap in the run results.

The run results also display the coordinates of the nearest candidate bitmap within the actual bitmap, along with the similarity percentage used to find the candidate.

## When to Use the Option to Locate a Bitmap Within the Runtime Bitmap

Suppose your application includes a specific bitmap that you want to make sure exists when you run your steps. The bitmap you want to check may not always be located in the same place in your application.

If you attempt to compare the expected bitmap with the runtime bitmap, and the location of the expected bitmap changes, then the checkpoint may fail. Therefore, you can define the specific bitmap that you want to locate, and UFT searches for that bitmap anywhere in the object on which you define the checkpoint, relative to the top-left corner of the object.

## How UFT Displays Bitmap Checkpoint Results

By default, UFT only displays expected, actual, and difference bitmaps in the run results for checkpoints that fail. If a bitmap checkpoint is configured such that the checkpoint can pass even if the expected and actual bitmaps are not identical, you can configure the run results to display the captured actual, expected and difference bitmaps for bitmap checkpoints that pass, too.

The results of bitmap checkpoints may be affected by factors such as operating system, screen resolution, and color settings.

For details on run results of a checkpoint, see the section on [Bitmap Checkpoint Results](#).

## Fine-Tuning the Bitmap Comparison

### Relevant for: GUI tests and components

When running a bitmap checkpoint, UFT compares the area that you are checking in the application with the bitmap stored in the checkpoint, pixel by pixel. By default, if any pixels are different, the checkpoint fails. The advanced settings in the Bitmap Checkpoint Properties dialog box provides various options for fine-tuning the bitmap comparison:

- When comparing bitmaps, you can adjust the comparison to enable the checkpoint to pass even if the bitmaps are not identical by setting the **RGB tolerance** and **Pixel tolerance** options described below.
- When locating a specified bitmap within the runtime bitmap, you can use **similarity** to enable the checkpoint to pass, even if the exact bitmap is not found in your application. For details on similarity, see "[Image Similarity](#)".
- You can also use a **custom comparers** to carry out a bitmap checkpoint. A custom comparer is a COM object that you or a third party can develop to run the bitmap comparison in the checkpoint, according to a more specific algorithm. For details on using the **Comparer** option, see "[Custom Comparers](#)".

### RGB Tolerance

**Note:** This functionality is available only when comparing expected bitmaps with runtime bitmaps. It is not available when locating a specified bitmap within the runtime bitmap.

The RGB (Red, Green, Blue) tolerance determines the percent by which the RGB values of the pixels in the runtime bitmap can differ from those of the expected bitmap and allow the checkpoint to pass. (The RGB tolerance option is limited to bitmaps with a color depth of 24 bits.)

For example, a bitmap checkpoint on identical bitmaps could fail if different display drivers are used when you create your checkpoint and when you run your test. Suppose one display driver displays the color white as RGB (255, 255, 255) and another driver displays the color white as RGB (231, 231, 231). The difference between these two values is about 9.4%. By setting the **RGB tolerance** to 10%, your checkpoint will pass when running your test with either of these drivers.

**Note:** UFT applies the RGB tolerance settings when comparing each pixel in the expected and runtime bitmaps. The Red, Green, and Blue values for each pixel are compared separately. If any of the values differs more than the tolerance allows, the pixel fails the comparison.

### Pixel Tolerance

**Note:** This functionality is available only when comparing expected bitmaps with runtime bitmaps. It is not available when locating a specified bitmap within the runtime bitmap.

The pixel tolerance determines the number or percentage of pixels in the runtime bitmap that can differ from those in the expected bitmap and allow the checkpoint to pass.

For example, suppose the expected bitmap has 4000 pixels. If you define the pixel tolerance to be 50 and select the **Pixels** radio button, up to 50 pixels in the runtime bitmap can be different from those in the expected bitmap and the checkpoint passes. If you define the pixel tolerance to be 5 and select the **Percent** radio button, up to 200 pixels (5 percent of 4000) in the runtime bitmap can be different from those in the expected bitmap and the checkpoint passes.

## Using Both RGB and Pixel Tolerances

**Note:** This functionality is available only when comparing expected bitmaps with runtime bitmaps. It is not available when locating a specified bitmap within the runtime bitmap.

If you define both RGB and pixel tolerances, the RGB tolerance is calculated first. The pixel tolerance then defines the maximum number of pixels that can fail the RGB criteria and allow the checkpoint to pass.

For example, suppose you define an RGB tolerance of 10 percent and a pixel tolerance of 5 percent, for a bitmap that has 4000 pixels.

For the checkpoint to pass, each pixel in the runtime bitmap must have RGB values that are no greater than or no less than 10 percent of the RGB values of the expected bitmap. If that criterion fails, UFT checks that the number of pixels that failed are less than 200. If that criterion passes, the checkpoint passes.

## Image Similarity

**Note:** This functionality is available only when locating a specified bitmap within the runtime bitmap. It is not available when comparing expected bitmaps with runtime bitmaps.

Image similarity settings can enable a checkpoint to pass, even if the exact bitmap is not found in your application. UFT attempts to locate the specified bitmap in the runtime bitmap of the object in your application during the run session. If UFT locates an exact match to the specified bitmap, then the checkpoint passes.

If an exact match cannot be found and you specified less than 100% in the **Similarity** option in the advanced settings of the Checkpoint Properties Dialog Box, UFT adjusts the comparison according to the similarity level. If the possible candidate has a similarity that is equal to or greater than the percentage that you defined, the checkpoint passes.

## Custom Comparers

A custom comparer is a COM object that you or a third party can develop to run the bitmap comparison in the checkpoint according to a more specific algorithm. If you use a custom comparer to perform the bitmap checkpoint, UFT sends the comparer two bitmaps to compare: A screen capture of the object, created with the checkpoint and saved as the expected bitmap, and a screen capture of the object as it

appears in the application during the run session. The comparer then compares these two bitmaps according to the specifications in its algorithm. If you use a custom comparer, you cannot use the Checkpoint Properties Dialog Box to specify tolerance or similarity settings, or areas of the object to compare or ignore.

If one or more custom comparers are installed and registered on the UFT computer, the Advanced Settings Dialog Box (Bitmap Checkpoints Dialog Box) includes a **Comparer** option.

The **Comparer** option enables you to select the UFT default comparer or a custom comparer that performs the bitmap comparison according to your testing requirements. For an example of when it can be useful to create a custom comparer, see ["Custom Comparer for Images Whose Location Changes in the Application - Use-Case Scenario" on page 1093](#). For details on developing or installing custom comparers, see ["Developing Custom Comparers for Bitmap Checkpoints \(GUI Testing\)" on page 1092](#).

If you select a custom comparer, some of the options in the Bitmap Checkpoint Properties Advanced Settings dialog box are different. For details, see Advanced Settings Dialog Box (Bitmap Checkpoints Dialog Box).

## Database Checkpoints Overview

### Relevant for: GUI tests and scripted GUI components

You can use database checkpoints to check databases accessed by your application, and to detect defects. To do this, you define a query on your database. Then you create a database checkpoint that checks the results of the query.

You can define a database query in the following ways:

- Using Microsoft Query. You can install Microsoft Query from the custom installation of Microsoft Office.
- By manually defining an SQL statement.

You create a database checkpoint based on the results of the query (**result set**) you defined on a database. You can create a check on a database to check the contents of the entire result set, or a part of it. UFT captures the current data from the database, saves this information as **expected data**, and inserts a database checkpoint step.

This checkpoint is displayed in the Editor as a **DbTable.Check CheckPoint** statement and as a step in the Keyword View.

 DbTable	Check	CheckPoint("DbTable")	Check whether specified content in a database query matches
---------------------------------------------------------------------------------------------	-------	-----------------------	-------------------------------------------------------------

When you create a new database checkpoint, all cells contain a blue check mark, indicating they are selected for verification. You can select to check the entire results set, specific rows, specific columns, or specific cells. UFT checks only cells containing a check mark.

You can also specify the way UFT identifies the selected cells. For example, suppose you want to check the data that is displayed in the first row and second column in the ["Checkpoint Properties Dialog Box"](#) (described on page 320). However, you know that each time you run your test or scripted component, it

is possible that the rows may be in a different order, depending on the sorting that was performed in a previous step. Therefore, rather than finding the data based on row and column numbers, you may want UFT to identify the cell based on the column name and the row containing a known value in a **key column**.

During the run session, the database checkpoint compares the current data in the database to the expected data defined in the ["Checkpoint Properties Dialog Box" on page 320](#). If the expected data and the current results do not match, the database checkpoint fails.

**Note:**

- You can modify the expected data of a database checkpoint before a run session. You can also make changes to the query in an existing database checkpoint. This can be useful if you move the database to a new location on the network.
- You can view the results of the checkpoint in the run results. For details, see the section on [database checkpoint results](#).
- For details on creating database checkpoints, see ["How to Insert a Checkpoint in a GUI Test or Component" on page 309](#).

## File Content Checkpoints Overview

### Relevant for: GUI tests and scripted GUI components

You can use file content checkpoints to compare the textual content of a file that is generated during a run session with the textual content of a source file. This enables you to verify that the generated file contains the expected results. For example, you may want to verify that a PDF file generated during a run session displays the local corporate address at the top of every page.

You can perform a checkpoint on text in one line, multiple lines, or the entire document, as needed. You can also specify what to ignore. For example, if you expect certain lines or areas in the file to change, you can exclude them from the checkpoint.

When you select a source document to compare, UFT converts a copy of this document to a text file and displays the content in the file content editor area of the ["Checkpoint Properties Dialog Box" \(described on page 320\)](#) enabling you to configure your checkpoint. You can use parameters and regular expressions to augment your checkpoint, as needed.

You can perform a file content checkpoint for any of the following file types:

• HTML	• Microsoft Word	• Text
• PDF	• RTF	

If a file content checkpoint step fails during a run session, the step summary in the run results displays a side-by-side comparison of the generated document and the source document, enabling you to visually compare the differences between the documents, including lines or sections that were added or removed.

## Table Checkpoints Overview

### **Relevant for: GUI tests and scripted GUI components**

You can use table checkpoints to check the content of tables displayed in your application. For example, you can check that a specified value is displayed in a certain cell. For some environments, you can also check the property values of the table object. For example, you can check that a table has the expected number of rows and columns.

During a run session, the table checkpoint compares the actual data to the expected data, as defined in the checkpoint. If the results match, the checkpoint passes. For details, see the section on [table checkpoint results](#).

Different environments support different checkpoints. For details on supported checkpoints, see ["Supported Checkpoints" on page 1117](#).

### **Row Range Selection**

The tables in your application may be very large. A table checkpoint on a large table may take a long time to create and a long time to run. You can choose to include all rows in your table checkpoint or you can specify a smaller row range.

For some UFT add-ins, when creating a new table checkpoint object, you can specify the range of rows you want to include using the Define/Modify Row Range Dialog Box.

## Page Checkpoints Overview

### **Relevant for: GUI tests and scripted GUI components**

You can use page checkpoints to check statistical information about your Web pages. These checkpoints check the links and the sources of the images on a Web page. You can also instruct page checkpoints to include a check for broken links.

The following types of page checkpoints are available:

### **Individual Page Checkpoints**

You can manually add a page checkpoint to check the links and image sources on a selected Web page during a recording or editing session.

For task details, see ["How to Insert a Checkpoint in a GUI Test or Component" on page 309](#).

For user interface details, see ["Checkpoint Properties Dialog Box" on page 320](#).

### **Automatic Page Checkpoints**

You can instruct UFT to create automatic page checkpoints for every page during a recording session by selecting the **Create a checkpoint for each Web page while recording** check box in the Web > Advanced pane of the Options dialog box (**Tools > Options > GUI Testing tab > Web > Advanced** node).

By default, the automatic page checkpoint includes the checks that you select from among the available options in the Web > Advanced pane.

You can also instruct UFT not to perform automatic page checkpoints during run sessions by selecting the **Ignore automatic checkpoints while running tests** check box in the **Web > Advanced** pane of the Options dialog box (**Tools > Options > GUI Testing tab > Web > Advanced** node).

For details, see the chapter on the **Web > Advanced** pane in the GUI Testing tab of the Options dialog box in the *HP Unified Functional Testing Add-ins Guide*.

## Checking Text Overview

### Relevant for: GUI tests and scripted GUI components

You can check that a specified text string is displayed by adding one of the following checkpoints.

- **Standard Checkpoint.** Enables you to check the text property of an object. You can use standard checkpoints to check text in Windows-based and other types of applications (including Web-based applications). For details on standard checkpoints, see "[How to Insert a Checkpoint in a GUI Test or Component](#)" on page 309.
- **Text Area Checkpoint.** Enables you to check that a text string appears within a defined area in a Windows application, according to specified criteria. When checking text displayed in a Windows-based application, it is often advisable to define a text area larger than the actual text you want UFT to check. You then use the "[Checkpoint Properties Dialog Box](#)" (described on page 320) to configure the relative position of the Checked Text within the captured area. During a run session, UFT checks for the selected text within the defined area, according to the settings you configured.
- **Text Checkpoint.** Enables you to check that the text is displayed in a screen, window, or Web page, according to specified criteria. For example, suppose you want to check the third occurrence of a particular text string in a page. To check for this string, you can specify which text precedes and/or follows it and to which occurrence of the specified text string you are referring.

**Note:** Text recognition is not supported for objects in the Active Screen.

For task details, see "[How to Insert a Checkpoint in a GUI Test or Component](#)" on page 309.

## Text Recognition Overview

### Relevant for: GUI tests and components

When working with tests and scripted components, you can use the text and text area checkpoint or output value commands to verify or retrieve text in your objects.

When working with tests, keyword or scripted components, and function libraries, you can insert steps to capture the text from objects in your application using the `<testobject>.GetVisibleText`, the `testobject.GetTextLocation` test object methods, the `TextUtil.GetText` or `TextUtil.GetTextLocation` reserved object methods, or the `testobject.GetText` (for Terminal Emulator objects).

When you use one of these options, UFT identifies text in your application uses an OCR (optical character recognition) mechanism. When using this OCR engine, you can use the Abby OCR text recognition engine (the default option) or the Tesseract OCR engine.

When UFT uses the OCR mechanism, a number of factors can affect the text it retrieves. Depending on the characteristics of the text you want to retrieve, you can adjust several OCR configuration options to optimize the way the text is captured. You use the Text Recognition pane in the Options dialog box to specify the preferred text recognition mechanism and OCR-specific settings.

In addition, you can instruct UFT to preprocess images before attempting the text recognition. This enables UFT to identify the image-based elements and avoid them when trying to find the text in the selected object or image.

## Guidelines for Text Recognition

### **Relevant for: GUI tests and components**

- If your application uses small fonts (less than 10 pt.) you should use the Tesseract OCR engine with the **Preprocess the image before using text recognition** option selected.
- High contrast between the background and text is best for text recognition (for example, black text on a white background).
- The color schema of the background should be permanent and without gradient.
- Text recognition is not supported for objects in the Active Screen.
- If your text is found within an image, it is recommended to use the Preprocess the image before using text recognition option in the Text Recognition pane of the Options dialog box (**Tools > Options > GUI Testing tab > Text Recognition mode**).
- When working with text area checkpoints or specifying the text area in methods such as **GetTextLocation**, note the following:
  - Try to keep the dimensions of the selected text area as small as possible to prevent additional unwanted characters in recognized text.
  - Consider the potential movement (change of coordinates) of the object within the window. For example, the screen resolution is often different on different computers, and this can affect the coordinates of the object in the application. Also, during the design and development stages of an application, an object may be moved to make room for other objects or for aesthetic purposes.
  - Consider that the operating system, installed service packs, installed toolkits, and so on, can all affect the size and location of an object in an application. Make sure that the dimensions of the selected text area are large enough for different system configurations.
- Single text block mode and multiple text block mode sometimes result in different captured text. If you are not sure which text block mode to use, use the default multiple block mode. If the results are not what you expect, then try using the single text block mode.

**Tip:** If you want to use the text recognition mechanism for a large area containing different

fonts and backgrounds, it is recommended to create several steps to capture the text for each single text block instead of creating one step to capture a multiple text block.

- If the text recognition mechanism retrieves unwanted text information (such as hidden text and shadowed text that appears as multiple copies of the same string) when using the multiple text block mode, use the single text block mode option. To do this, in the **Text Recognition** pane of the Options dialog box (**Tools > Options > GUI Testing tab > Text Recognition** node), select **Single text block mode**.
- When checking text in an application window, it is highly recommended to check text by inserting a standard checkpoint for the object containing the desired text, using its **text** (or similar) property.
- **For tests:** If you are creating text area checkpoints, see the **Important Information** section in Checkpoint Properties Dialog Box for additional guidelines.

## Text Recognition and Development Environments

### Relevant for: GUI tests and components

The following table lists the development environments supported by UFT (via its add-ins) and specifies what is supported for text recognition.

Development Environment	Text Recognition Supported	Text Recognition Not Supported
<b>ActiveX</b>	Full text recognition support	N/A
<b>Delphi</b>	Full text recognition support	N/A
<b>Java</b>	<ul style="list-style-type: none"> <li>• Text checkpoints</li> <li>• Text output values</li> <li>• Text area checkpoints</li> <li>• Text area output values</li> </ul>	<ul style="list-style-type: none"> <li>• <b>GetTextLocation</b> method</li> <li>• <b>GetVisibleText</b> method</li> </ul>
<b>.NET WebForms</b>	<ul style="list-style-type: none"> <li>• Text checkpoints for Page object only</li> <li>• Text output values for Page object only</li> </ul>	<ul style="list-style-type: none"> <li>• Text checkpoints for all other objects</li> <li>• Text output values for all other objects</li> <li>• Text area checkpoints for all objects</li> <li>• Text area output values for all objects</li> <li>• <b>GetTextLocation</b> method</li> <li>• <b>GetVisibleText</b> method</li> </ul>
<b>.NET WinForms</b>	Full text recognition support	N/A
<b>Oracle</b>	N/A	No text recognition support

<b>Development Environment</b>	<b>Text Recognition Supported</b>	<b>Text Recognition Not Supported</b>
<b>PeopleSoft</b>	<ul style="list-style-type: none"> <li>Text checkpoints for PSFrame object only</li> <li>Text output values for PSFrame object only</li> </ul>	<ul style="list-style-type: none"> <li>Text checkpoints for all other objects</li> <li>Text output values for all other objects</li> <li>Text area checkpoints for all objects</li> <li>Text area output values for all objects</li> <li><b>GetTextLocation</b> method</li> <li><b>GetVisibleText</b> method</li> </ul>
<b>PowerBuilder</b>	Full text recognition support	N/A
<b>SAP Gui for Windows</b>	N/A	No text recognition support
<b>SAP Web</b>	<ul style="list-style-type: none"> <li>Text checkpoints</li> <li>Text output values</li> </ul>	<ul style="list-style-type: none"> <li>Text area checkpoints</li> <li>Text area output values</li> <li><b>GetTextLocation</b> method</li> <li><b>GetVisibleText</b> method</li> </ul>
<b>Siebel</b>	N/A	No text recognition support
<b>Silverlight</b>	Full support except as listed in the <b>Not Supported</b> column.	<ul style="list-style-type: none"> <li><b>GetTextLocation</b> method</li> </ul>
<b>Standard Windows</b>	Full text recognition support	N/A
<b>Stingray</b>	Full text recognition support	N/A
<b>Terminal Emulators</b>	Text output values for TeScreen and TeTextScreen objects only	<ul style="list-style-type: none"> <li>Other text checkpoints</li> <li>Other text output values</li> <li>Text area checkpoints</li> <li>Text area output values</li> <li><b>GetTextLocation</b> method</li> <li><b>GetVisibleText</b> method</li> </ul>
<b>VisualAge</b>	Full text recognition support	N/A
<b>Visual Basic</b>	Full text recognition support	N/A
<b>Web</b>	<ul style="list-style-type: none"> <li>Text checkpoints for Page object only</li> <li>Text output values for Page object only</li> </ul>	<ul style="list-style-type: none"> <li>Text checkpoints for all other objects</li> <li>Text output values for all other objects</li> <li>Text area checkpoints for all objects</li> <li>Text area output values for all</li> </ul>

Development Environment	Text Recognition Supported	Text Recognition Not Supported
		objects <ul style="list-style-type: none"> <li>• <b>GetTextLocation</b> method</li> <li>• <b>GetVisibleText</b> method</li> </ul>
<b>WPF</b>	Full support except as listed in the <b>Not Supported</b> column.	<ul style="list-style-type: none"> <li>• <b>GetTextLocation</b> method</li> </ul>

## Checking Text in an Image - Use-Case Scenario

### Relevant for: GUI tests and scripted GUI components

Ben and George are quality assurance engineers who are experienced UFT users. George is also familiar with text recognition and has a basic understanding of how text recognition mechanisms work.

Ben often uses bitmap checkpoints to check the appearance of various icons or pictures in the user interface he is testing.

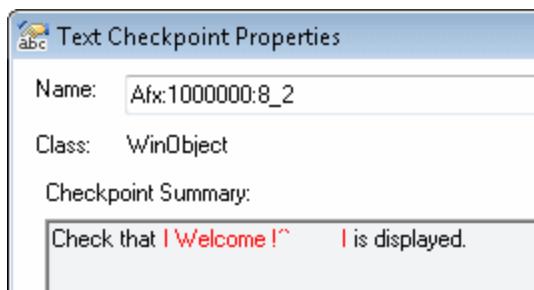
For one of his projects, Ben also needed to verify the text in the graphics, so he decided to use text checkpoints.

Ben decided to begin the verification process by inserting a text checkpoint to check that the text Welcome ! was displayed correctly in the following graphic.



Before inserting the text checkpoint, Ben opened the Text Recognition pane and configured the text recognition settings. Ben also knew that single text block mode usually works best, so he selected the **Single text block mode** option.

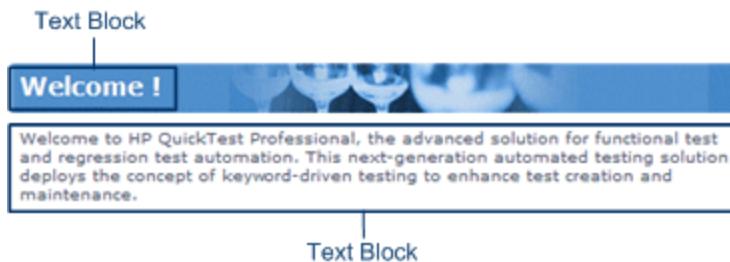
Ben then inserted a text checkpoint on the entire area shown above and received the following results in the Text Checkpoint Properties dialog box:



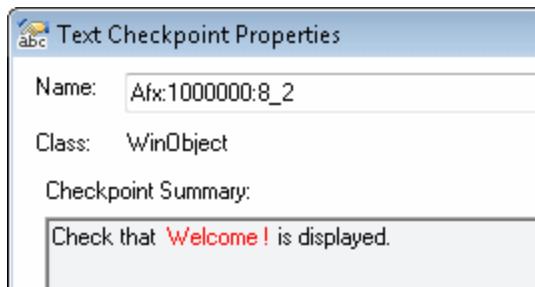
Ben noticed that there were extra characters in the **Checkpoint Summary** area of the text checkpoint, but he did not know why.

Ben asked his colleague, George, for help. George explained to him that the text recognition mechanism sometimes adds extra characters to the text checkpoint when it does not recognize the text correctly.

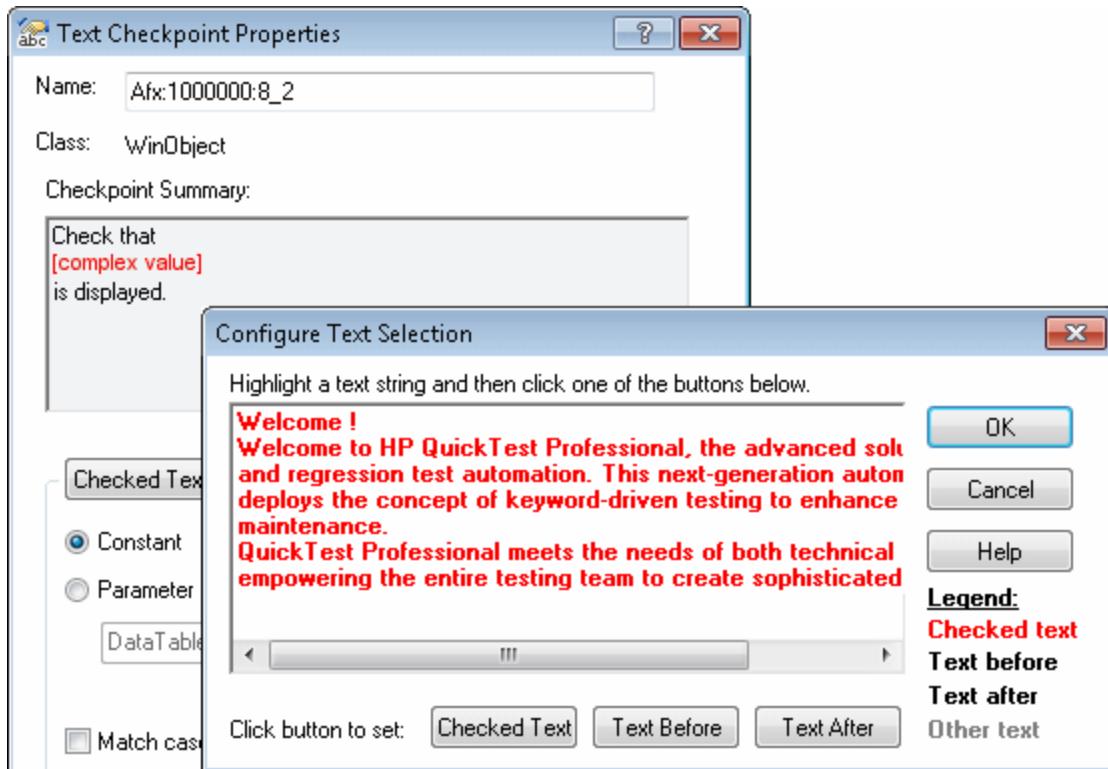
George also pointed out that the area Ben defined for the text checkpoint consisted of multiple text blocks because the text was not uniform in font size, color, or background. The title area consisted of white characters on a blue-gray background, while the remaining text was smaller and consisted of blue text on a white background.



Ben remembered that he had selected the **Single text block mode** option in the **Text Recognition** pane (**Tools > Options > GUI Testing tab > Text Recognition** node) and understood that if he wanted to use single text block mode, he would have to create a text checkpoint only on the **Welcome !** area of the graphic, and not on the entire graphic. Ben tried this, and the OCR mechanism correctly identified the text, as shown below:



Ben was pleased with the results, but he wanted to explore other possibilities, so he inserted another text checkpoint—this time on the entire graphic. He selected the **Multiple text block mode** option in the Text Recognition pane, which resulted in the following:



Ben was pleased that the OCR mechanism correctly recognized all of the text in the graphic. But he needed to check only the title, Welcome !, so he finalized this checkpoint by marking all of the text after Welcome ! as **Text After**.

Even though both checkpoints passed, Ben needed only one text checkpoint. He decided to keep the first checkpoint (that used **Single text block mode**), and he deleted the second one. He selected the **Single text block mode** option in the Text Recognition pane to help ensure that the checkpoint would pass in future run sessions.

## XML Checkpoints Overview

### Relevant for: GUI tests and scripted GUI components

You can use XML checkpoints to check the contents of individual XML data files or documents that are part of your Web application.

XML (Extensible Markup Language) is a meta-markup language for text documents that is endorsed as a standard by the W3C (World Wide Web Consortium). XML makes the complex data structures portable between different computer environments/operating systems and programming languages, facilitating the sharing of data.

XML files contain text with simple tags that describe the data within an XML document. These tags describe the data content, but not the presentation of the data. Applications that display an XML document or file use either CSS (Cascading Style Sheets) or XSL-FO (XSL Formatting Objects) to present the data.

You can perform checkpoints on XML documents contained in Web pages or frames, on XML files, and on test objects that support XML (such as **WebXML** test objects). An XML checkpoint is a verification point that compares a current value for a specified XML element, attribute and/or value with its expected value. When you insert a checkpoint, UFT adds a checkpoint step in the Keyword View and adds a `Check CheckPoint` statement in the Editor. During a run session, UFT compares the expected results of the checkpoint to the current results. If the results do not match, the checkpoint fails.

After a run session, you can view summary results of the XML checkpoint in the run results.

A few common uses of XML checkpoints are described below:

- An XML file can be a static data file that is accessed to retrieve commonly used data for which a quick response time is needed—for example, country names, zip codes, or area codes. Although this data can change over time, it is normally quite static. You can use an XML file checkpoint to validate that the data has not changed from one application release to another.
- An XML file can consist of elements with attributes and values (character data). There is a parent and child relationship between the elements, and elements can have attributes associated with them. If any part of this hierarchy (including data) changes, your application's ability to process the XML file may be affected. Using an XML checkpoint, you can check the content of an element to make sure that its tags, attributes, and values have not changed.
- XML files are often an intermediary that retrieves dynamically changing data from one system. The data is then accessed by another system using Document Type Definitions (DTD), enabling the accessing system to read and display the information in the file. You can use an XML checkpoint and parameterize the captured data values if you want to check an XML document or file whose data changes in a predictable way.
- XML documents and files often need a well-defined structure to be portable across platforms and development systems. One way to accomplish this is by developing an XML schema, which describes the structure of the XML elements and data types. You can use schema validation to check that each item of content in an XML file adheres to the schema description of the element in which the content is to be placed.

**Note:** XML checkpoints are compatible with namespace standards. A change in namespace between expected and actual values results in a failed checkpoint.

For details on XML standards, see: <http://www.w3.org/XML/>

For details on namespace standards, see: <http://www.w3.org/TR/1999/REC-xml-names-19990114/>

## XML Checkpoint Types

### **Relevant for: GUI tests and scripted GUI components**

You can create the following types of XML checkpoints:

- **XML Web Page/Frame Checkpoint.** Checks an XML document within a Web page or frame.
- **XML File Checkpoint.** Checks a specified XML file.

## Using XML Objects and Methods to Enhance Your Test or Scripted Component

### **Relevant for: GUI tests and scripted GUI components**

UFT provides several scripting methods that you can use with XML data. You can use these scripting methods to retrieve data and return new XML objects from existing XML data. You do this by using the XMLUtil, or WebXML objects to return XML data and then using the supported XMLData objects and methods to manipulate the returned data.

**Tip:** All XMLData objects and methods are compatible with namespace and XPath standards.

For details on XML standards, see: <http://www.w3.org/XML/>

For details on namespace standards, see: <http://www.w3.org/TR/1999/REC-xml-names-19990114/>

For details on XPath standards, see: <http://www.w3.org/TR/1999/REC-xpath-19991116>

For details on programming in the Editor, see "[Programming in GUI Testing Documents in the Editor](#)" on [page 647](#). For details on XML objects and methods, see the **Supplemental Objects** section of the *HP UFT Object Model Reference for GUI Testing*.

## How to Insert a Checkpoint in a GUI Test or Component

### **Relevant for: GUI tests and components**

This task describes how to insert a checkpoint step while recording or editing your test or component. You can also add an existing checkpoint to a test or scripted component. It is generally more convenient to define checkpoints after creating the initial test or component.

For details on supported checkpoints per add-in environment, see "[GUI Checkpoints and Output Values Per Add-in](#)" on [page 1116](#)

This task describes both the general process for inserting a new checkpoint step to your test or component and the prerequisites and considerations for the different types of components.

This task includes the following steps:

- "Important information before inserting the checkpoint" below
- "Define automatic page checkpoints - optional" on the next page
- "Set global accessibility checkpoint preferences" on the next page
- "Insert a checkpoint step while recording your test or component" on the next page
- "Insert a checkpoint step while editing your test or component" on page 312
- "Use programming to insert checkpoints in a test or scripted component" on page 313
- "Set options for the checkpoint" on page 313
- "Move checkpoint objects from the local object repository to a shared object repository - optional" on page 315

## Important information before inserting the checkpoint

<b>Object visibility in application</b>	<ul style="list-style-type: none"> <li>During an editing session, make sure the object is visible in your application before inserting a standard checkpoint.</li> <li><b>For bitmap checkpoints:</b> During a run session, bitmap checkpoints can capture only the visible part of an object. Therefore, confirm that the object to capture is always fully visible on the screen before a bitmap checkpoint step is performed. One way to do this is to insert a <b>MakeVisible</b> statement (for relevant environments) prior to your bitmap checkpoint step. For details on the <b>MakeVisible</b> method, see the specific object methods and properties in the <i>HP UFT Object Model Reference for GUI Testing</i>.</li> <li><b>For file content checkpoints:</b> The source file must be located on the file system.</li> </ul>
<b>Availability</b>	<ul style="list-style-type: none"> <li>Recording sessions</li> <li>Editing sessions</li> <li>Active Screen (not supported for file content checkpoints or XML checkpoints)</li> </ul>
<b>Important Information</b>	<ul style="list-style-type: none"> <li>Checkpoints can be viewed in the following modes: <ul style="list-style-type: none"> <li><b>Simple Mode:</b> Displays only the basic properties and expected values of the checkpoint.</li> <li><b>Advanced Mode:</b> Displays all supported properties and expected values of the checkpoint.</li> </ul> </li> <li>In ALM, you cannot create, edit, or rename checkpoints for keyword GUI components.</li> <li>You cannot create image, table, or (Web) page checkpoints in a keyword GUI component. These special checkpoint types are only available for tests and scripted GUI components. However, if you select a Web page or any table object when creating a standard checkpoint for your component, you can check their object properties like any other object.</li> <li><b>For bitmap checkpoints:</b> If you want to create a bitmap checkpoint that contains multiple objects, you should select the highest level object that includes all the objects to include in the bitmap checkpoint.</li> <li><b>For text or text area checkpoints:</b> <ul style="list-style-type: none"> <li>Before you create a text or text area checkpoint for a Windows-based application, make sure you configure the required capture settings in the Text Recognition pane (<b>Tools &gt; Options &gt; GUI Testing</b> tab &gt; <b>Text Recognition</b> node).</li> <li>You can also check the text property of an object in Windows-based and other types of applications (including Web-based applications) by using a standard checkpoint.</li> </ul> </li> </ul>

- **For XML checkpoints:** You can insert XML checkpoints to verify Web pages and frames and to directly access and verify specific XML files in your system.

## Define automatic page checkpoints - optional

In the **Web > Advanced** pane of the Options dialog box (**Tools > Options > GUI Testing tab > Web**)

> **Advanced** node), do one or more of the following:

- To instruct UFT to create automatic page checkpoints for every page during every recording session, select the **Create a checkpoint for each Web page while recording** check box.
- To instruct UFT not to perform automatic page checkpoints during run sessions, select the **Ignore checkpoints while running tests** check box.

## Set global accessibility checkpoint preferences

In the Web Advanced pane of the Options dialog box (**Tools > Options > GUI Testing tab > Web**)

> **Advanced** node), do one or more of the following:

- Define the checks to include in the checkpoints. All accessibility checkpoints in your test use the options that are selected in the Advanced Web Options dialog box at the time of the run session. You can also view these options in the Accessibility Checkpoint Properties dialog box.
- **(Optional)** To instruct UFT to insert an accessibility checkpoint for each page as you record, select the **Add Automatic accessibility checkpoint to each Web page while recording** check box. This creates an accessibility checkpoint for each page as you record.

## Insert a checkpoint step while recording your test or component

1. Start a recording session before inserting a checkpoint.
2. Insert a checkpoint by doing one of the following:
  - In the Record toolbar, click the **Insert Checkpoint or Output Value** button and select the type of checkpoint from the drop-down list.
  - Select **Design > Checkpoint** and choose the relevant type of checkpoint.
3. Click the **Insert Checkpoint or Output Value** button  in the toolbar and select the type of checkpoint from the drop-down list.

3. UFT is hidden, and the pointer changes to a pointing hand. In your application, click the object that you want to check.

**Note:** If the object in your application is associated with more than one location, the Object Selection dialog box opens. This dialog box enables you to select an object to check from the object tree. The objects in the tree are displayed with hierarchical order, based on the location you clicked in the Active Screen or application.

## Insert a checkpoint step while editing your test or component

1. You may need to open the application and display the relevant object before inserting a checkpoint. This depends on the environment and the object type you are checking. For details, see the prerequisite information for your specific checkpoint type.
2. Select the step where you want to add the checkpoint and do one of the following:
  - Select **Design > Checkpoint**, and then select the relevant checkpoint option.
  - Select **Design > Checkpoint > Existing Checkpoint**.
  - Right-click any object in the Active screen and select the relevant checkpoint. You can create checkpoints for any object in the Active Screen even if the object is not part of any step in the Keyword View.

If you use the Active Screen to insert a checkpoint, ensure that the Active Screen contains sufficient data for the object you want to check.

**Note:** If the object in your application is associated with more than one location, the Object Select Dialog Box opens. This dialog box enables you to select an object to check from the object tree. The objects in the tree are displayed in hierarchical order, based on the location you clicked in the Active Screen or application.

### Notes:

- **For table checkpoints:** When inserting a table checkpoint, for certain objects in certain environments, before the Table Checkpoint Properties dialog box opens, the Define/Modify Row Range Dialog Box opens. In this dialog, select the row range to check.
- **For text or text area checkpoints:**
  - To create the checkpoint, you first highlight a text string in the Active Screen then right-click the string, and select **Insert Text Checkpoint**.
  - When you create a text area checkpoint, you first define the area containing the text you want UFT to check.

When you select the Text Area Checkpoint option, the mouse turns into a crosshairs pointer. Click and drag the crosshairs point to define this area. Release the mouse button after outlining the area required.

**Tip:** Hold down the left mouse button and use the arrow keys to make precise adjustments to the defined area.

- **For file content checkpoints:** When inserting a file content checkpoint, the File Content Checkpoint Properties dialog box displays by default the option to select only **All Supported Files**. When this is selected, only files with the expected extensions are displayed (for example, .htm or .pdf files). You can also select a file that uses a non-standard extension by selecting **All**

Files in the **Files of type** box and then selecting the relevant file.

- **For database checkpoints:**

When inserting a database checkpoint, the Database Query Wizard opens.

- a. In the Database Query Wizard, define the query for your checkpoint using Microsoft Query or by manually entering a database connection and SQL statement.
- b. If you selected Microsoft Query as your data source, Microsoft Query opens, enabling you to define a query. When you are done, in the Finish page of the Query Wizard, use one of the following:
  - **Exit and return to HP Unified Functional Testing.** Exits Microsoft Query.
  - **View data or edit query in Microsoft Query.** View or edit the query prior to exiting Microsoft Query.
- c. If you selected **Specify SQL statement manually**, the Specify SQL statement page opens, enabling you to specify the connection string and the SQL statement.

## Use programming to insert checkpoints in a test or scripted component

- If you want to retrieve the return value of a checkpoint (a boolean value that indicates whether the checkpoint passed or failed), you must add parentheses around the checkpoint argument in the statement in the Editor. For example:

```
a = Browser("MyBrowser").Page("MyPage").Check (CheckPoint("MyProperty"))
```

- You can also use the **CheckProperty** method and the **CheckItemProperty** method to check specific property or item property values. For details, see the specific object methods and properties in the *HP UFT Object Model Reference for GUI Testing*.

## Set options for the checkpoint

In the Checkpoint Properties Dialog Box, specify the settings for the checkpoint object.

### For table checkpoints:

Define the cell selection for the table object in the Grid area of Table Checkpoint Properties dialog box, as follows:

To:	Do this:
Add a <b>single cell</b> to or remove it from the check	Double-click the cell
Add an <b>entire row</b> to or remove it from the check	Double-click the row header

Add an <b>entire column</b> to or remove it from the check	Double-click the column header.
Add <b>all cells</b> to or remove all cells from the check	Double-click the column header.
Add <b>a range of cells</b> to the check	Select the cells to add to the check and click the <b>Add to Check</b> button 
Remove <b>a range of cells</b> from the check	Select the cells to remove from the check and click the <b>Remove from Check</b> button 

**For database checkpoints:**

Define the cell selection for the database object in the Grid area of Database Checkpoint Properties dialog box, as follows:

To:	Do this:
Add a <b>single cell</b> to or remove it from the check	Double-click the cell
Add an <b>entire row</b> to or remove it from the check	Double-click the row header
Add an <b>entire column</b> to or remove it from the check	Double-click the column header.
Add <b>all cells</b> to or remove all cells from the check	Double-click the column header.
Add <b>a range of cells</b> to the check	Select the cells to add to the check and click the <b>Add to Check</b> button 
Remove <b>a range of cells</b> from the check	Select the cells to remove from the check and click the <b>Remove from Check</b> button 

To modify the SQL query definition, in the Keyword View or Editor, right-click the database object that you want to modify and select **Object Properties**.

**For file content checkpoints:**

In the File Content Checkpoint Properties dialog box, scroll to each line you want to compare and select it.

As you hover over a line, a checkbox and a regular expression icon are displayed in the sidebar to the left of that line.

- Click the check box to select (or clear) the line for verification.
- Click the Treat Line as Regular Expression/Plain Text button to add (or remove) backslashes prior to all special characters in that line. You can then modify any regular expressions, as needed.

**Note:**

- If the source file contains multiple pages, the File Content Editor is divided into separate pages. You can then expand or collapse the pages, select or clear entire pages for verification and so on.

## Move checkpoint objects from the local object repository to a shared object repository - optional

After you insert a checkpoint step, the checkpoint object is added to the local object repository. If you are using shared object repositories, you can move the new checkpoint object to your shared object repository.

## How to Include and Ignore Areas When Comparing a Bitmap - Use-Case Scenario

### Relevant for: GUI tests and components

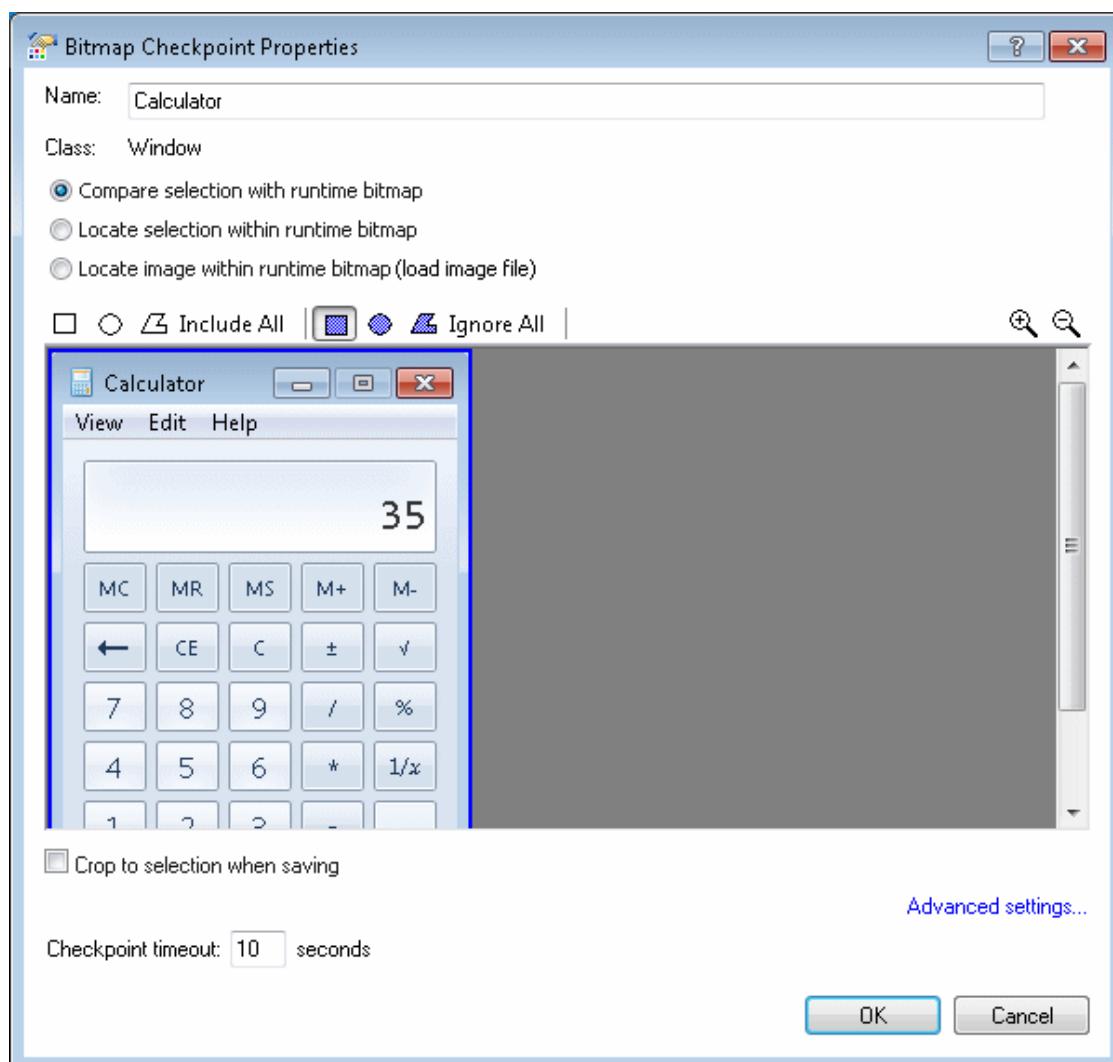
Suppose you want to compare an expected bitmap with an actual bitmap of the object in your application, but there are areas of this runtime bitmap that might be affected by dynamic values and parameters. If you include these areas in the checkpoint, running the steps with different values may cause the checkpoint to fail.

This use-case scenario describes the process you would follow to define which areas to ignore and include in a bitmap checkpoint that compares an expected bitmap with a runtime bitmap.

**Note:** For a task related to this scenario, see "[How to Insert a Checkpoint in a GUI Test or Component](#)" on page 309.

1. In UFT, start a recording session and open the Windows Calculator application (for example, **Start > Programs > Accessories > Calculator**).
2. Record steps that multiply **7** by **5** and display the result.
3. Select **Design > Checkpoint** menu, or click the **Insert Checkpoint or Output Value** button  in the toolbar, and then select **Bitmap Checkpoint**. UFT is hidden, and the pointer changes to a pointing hand.

4. Select the Calculator application. If the Object Selection Dialog Box opens, select the top-level object and click **OK**. The Checkpoint Properties Dialog Box opens.

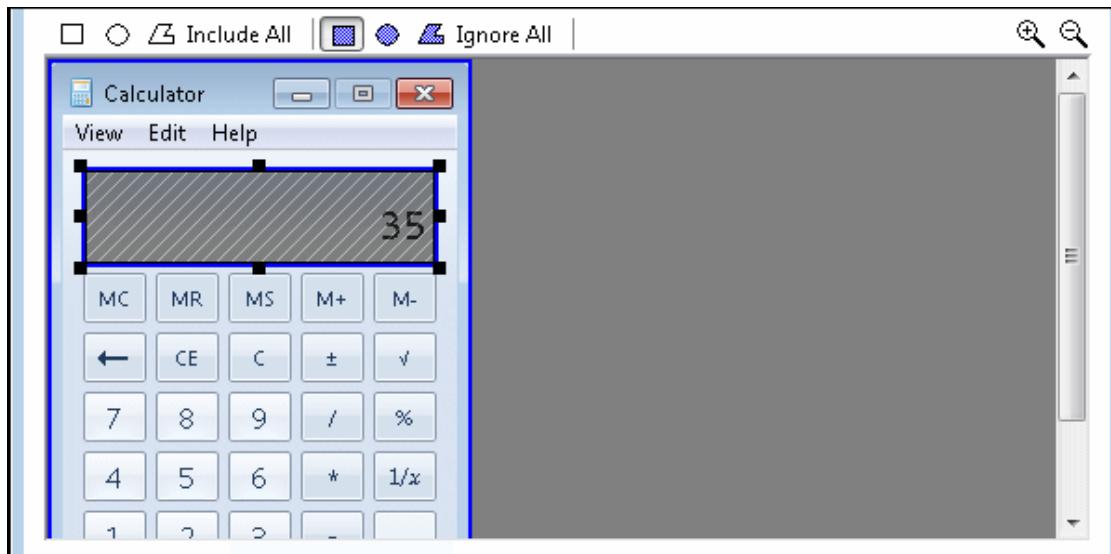


**Note:** (For tests and scripted components) For the purpose of this use-case scenario, you insert the bitmap checkpoint while recording your steps. However, the options are relevant also when inserting a bitmap checkpoint from the Active Screen.

5. In the Bitmap Checkpoint Properties dialog box, select the **Compare selection with runtime bitmap** radio button. For the purpose of this scenario, we will compare a bitmap of the entire Calculator application.
6. Because the value in the number line may change, depending on parameters that are used during the run session, we want to ignore the number line when comparing the expected bitmap with the runtime bitmap.

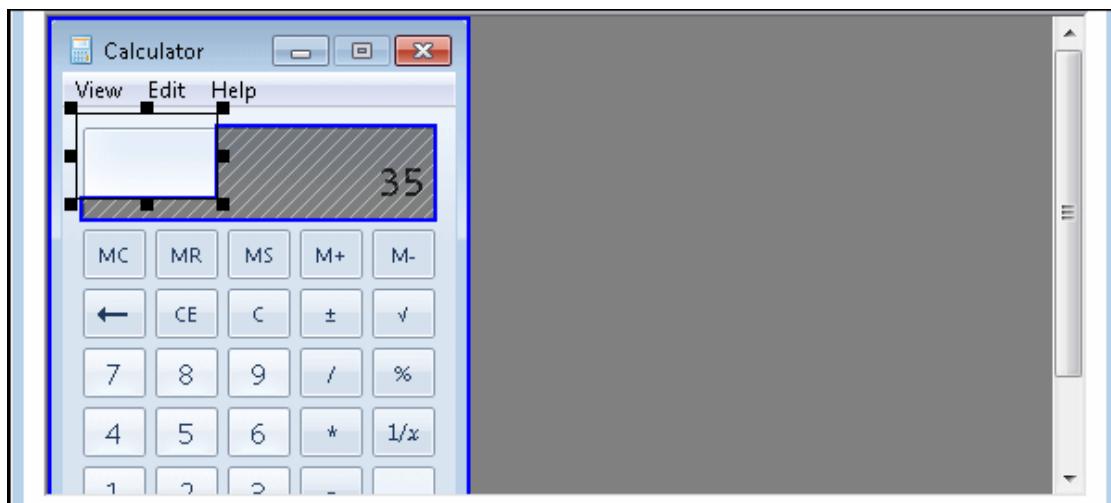
To do this, click the **Mark Rectangle to Ignore** button  in the toolbar, and then select the entire number line. Modify the size and position of your selection as needed. When you are finished, double-click to finalize the selection.

The following example shows the bitmap display area when the number line is selected to be ignored:

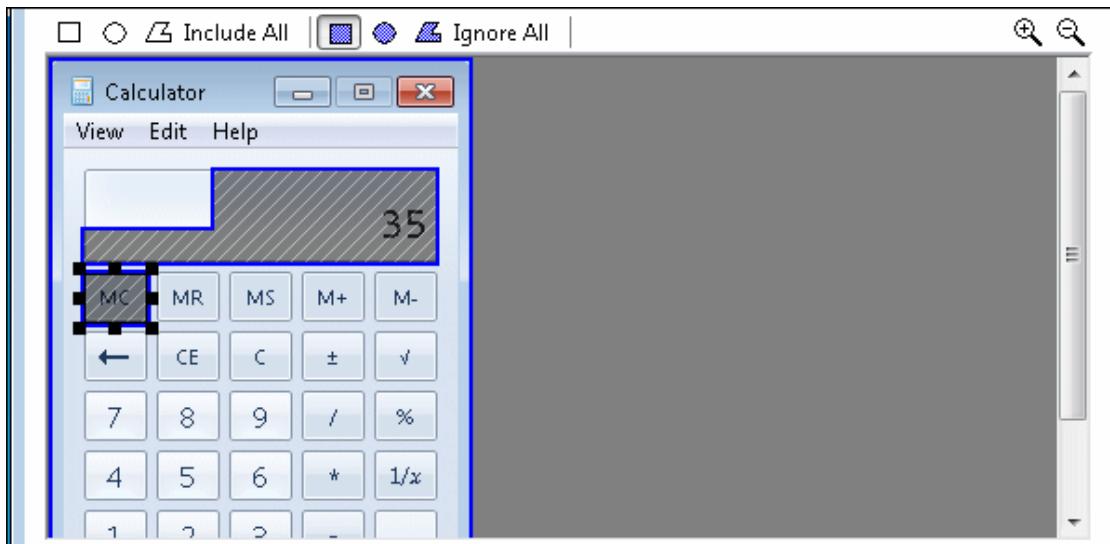


7. (Optional) If you need to modify your ignored selection after you finalize it, you can do so by clicking the **Mark Rectangle to Include** button  in the toolbar, and then selecting the areas that you want to clear. Any areas that you clear from the ignored selection are included in the comparison.

The following example shows the bitmap display area after a part of the ignored selection is selected for clearing (double-click the area to finalize):



8. (Optional) If you want additional areas to be ignored, click the **Mark Rectangle to Ignore** button  , select the relevant areas, and double-click to finalize.



9. Click **OK** to close the Bitmap Checkpoint Properties dialog box and insert the bitmap checkpoint. When you run your steps, the bitmap checkpoint ignores the area that you selected, and the checkpoint passes even if the value in the number line changes.

## How to Configure Text Recognition Settings

### Relevant for: GUI tests and components

This task describes a suggested workflow for configuring text recognition settings.

This task includes the following steps:

- ["Prerequisites" below](#)
- ["Analyze the characteristics of the text" below](#)
- ["Set the appropriate options in the Text Recognition Pane" on the next page](#)
- ["Check the text recognition settings in your test/component" on the next page](#)
- ["Adjust the settings as necessary" on page 320](#)

### 1. Prerequisites

In your application, display the text you want to capture.

### 2. Analyze the characteristics of the text

Determine whether you can capture the text using a text (or text-like) property instead of using a text recognition mechanism.

If it must use text-recognition, analyze whether the **Abby OCR** engine or the **Tesseract OCR** engine are more likely to meet your needs.

### 3. Set the appropriate options in the Text Recognition Pane

In the **Text Recognition** pane of the Options dialog box (**Tools > Options > GUI Testing tab > Text Recognition** node), set the following options:

<b>OCR engine type</b>	Select either the <b>Abby OCR</b> or <b>Tesseract OCR</b> text recognition option.  <b>Note:</b> The performance of the Tesseract OCR engine is slower than the Abby OCR engine. If your test has a significant use of text recognition steps (such as GetVisibleText), note that the total time required to run these tests will increase.
<b>Text Recognition mode</b>	<ul style="list-style-type: none"> <li><b>Single text block mode:</b> Focuses on the area and treat it as a single text block. This is especially useful when trying to capture text on small objects or in a small text area. Select this radio button if the text on the object is uniform in font, size, color, and background. For example:</li> <li><b>Multiple text block mode:</b> Instructs the OCR mechanism to handle each text area in the object that has a different background font and size. The OCR mechanism decides where to divide the text blocks according to an internal algorithm. Select this radio button only if the text on the object comprises different fonts, font sizes, colors, and/or backgrounds. For example:</li> </ul>
<b>Available languages and supported languages</b>	(For the Abby OCR engine only) From the list of selected languages, select the supported languages for text recognition.  <b>Note:</b> You can select multiple non-hieroglyphic languages (which include Chinese, Japanese, or Korean), or one of the hieroglyphic languages.
<b>Symbols for text recognition</b>	(For the Tesseract OCR engine only) Enter the list of characters you want UFT to recognize. When UFT runs the test, it will perform text recognition only on the characters specified and all others are ignored.
<b>Current language pack</b>	(For the Tesseract OCR engine only). The current language pack to use in text recognition. When using the Tesseract engine, it is possible to use only one language pack at a time.  You can download additional language packs from the Tesseract OCR engine download site: <a href="https://code.google.com/p/tesseract-ocr/downloads/detail?name=tesseract-ocr-3.02.tel.tar.gz&amp;can=2&amp;q=">https://code.google.com/p/tesseract-ocr/downloads/detail?name=tesseract-ocr-3.02.tel.tar.gz&amp;can=2&amp;q=</a> . After downloading, add the files from the language packs in the <UFT installation directory>/dat/tessdata folder.
<b>Preprocess the image before using text recognition</b>	Instructs UFT to process the background image before performing text recognition. This enables UFT to identify the image elements before using text recognition.

### 4. Check the text recognition settings in your test/component

- Create or open a test or component.

b. Do any of the following:

- Insert a text checkpoint or output value step (tests and scripted components only)
- Insert a step that uses one of the following test object methods:
  - `testobject.GetVisibleText`
  - `testobject.GetTextLocation`
  - `testobject.GetText` (for Terminal Emulator objects)
- Insert a step that uses one of the following reserved object methods (tests and scripted components only):
  - `TextUtil.GetText`
  - `TextUtil.GetTextLocation`

5. **Adjust the settings as necessary**

If the captured text is not as expected, analyze the problems and adjust the Text Recognition options to fine tune the way UFT captures your text.

## Checkpoint Properties Dialog Box

**Relevant for: GUI tests and components**

This dialog box enables you to edit your checkpoint properties for a selected checkpoint object.

<b>To access</b>	<ol style="list-style-type: none"><li>1. Ensure that a GUI action or component is in focus in the document pane.</li><li>2. Do one of the following:<ul style="list-style-type: none"><li>• <a href="#">Insert a new checkpoint step</a> and select an object from your application.</li><li>• Click in the Value cell of an existing checkpoint step, and click the checkpoint properties icon .</li><li>• <b>In an action or a scripted component:</b> Right-click an existing checkpoint step and select <b>Checkpoint Properties</b>.</li><li>• In the local or shared object repository, click an existing checkpoint object. The <b>Checkpoint</b> details are displayed on the right side of the object repository window, in the <b>Object Details</b> area.</li></ul></li></ol>
<b>Important information</b>	The advanced properties of standard checkpoint objects cannot be viewed or edited in ALM. Therefore, if one or more advanced properties are selected in a standard checkpoint object, and an ALM user views its properties in ALM, the dialog box displays text indicating that some properties are not shown.

# Add Existing Checkpoint Dialog Box

## **Relevant for: GUI tests and scripted GUI components only**

This dialog box enables you to add an existing checkpoint to your test while editing.

<b>To access</b>	<ol style="list-style-type: none"> <li>1. Ensure that a GUI action or component is the active document in the document pane.</li> <li>2. Select <b>Design &gt; Checkpoint &gt; Existing Checkpoint</b>.</li> </ol>
<b>Important information</b>	<ul style="list-style-type: none"> <li>• This option is available only if at least one of the object repositories associated with the current action (including the local object repository) contains at least one checkpoint.</li> <li>• If a test object step is highlighted in the Keyword View or the cursor is located in a step in the Editor, the Add Existing Checkpoint dialog box opens with the <b>TestObjects</b> tree hidden. The test object displayed in the <b>Test object</b> box is the object from the highlighted step in the Keyword View or the specific object where the cursor is located in the Editor.</li> </ul>
<b>See also</b>	<a href="#">"Adding Existing Checkpoints to a Test" on page 290</a>

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Elements	Description
<b>Test object</b>	The name of the test object for which you are adding a checkpoint.
<b>Test Objects tree</b>	All objects in the current action.
<b>Show/Hide Test Objects</b>	Shows or hides the <b>Test Objects</b> tree.
<b>Display only checkpoints relevant to the selected test object</b>	<p>When selected, UFT determines which checkpoints from the current action's object repositories are relevant for the selected object (based on the checkpoint type and the properties selected in the checkpoint) and displays only those checkpoints in the <b>Checkpoints</b> list.</p> <p>When using this option, open your application and display the selected object to enable UFT to accurately determine all of the checkpoints that can apply to that object.</p>
<b>Checkpoints</b>	<p>Lists the checkpoints available for insertion.</p> <p>If the <b>Display only checkpoints relevant to the selected test object</b> option is cleared, this list includes all checkpoints from all object repositories associated with the current action.</p> <p>If the <b>Display only checkpoints relevant to the selected test object</b> option is selected, this list displays only the relevant checkpoints as described above.</p>
<b>&lt;checkpoint details area&gt;</b>	Displays the settings of the selected checkpoint in read-only format.
<b>Configure value</b>	The value for the selected checkpoint in read-only mode. For details, see <a href="#">"Value Configuration and Parameterization" on page 382</a> .

# Troubleshooting and Limitations - Using Checkpoints

## Relevant for: GUI tests and components

The following sections describe troubleshooting and limitations for working with checkpoints in a GUI test or component:

### Accessibility checkpoints

In ALM, you can view a comparison of accessibility checkpoints in the Asset Comparison Tool only if both UFT and the UFT Add-in for ALM are installed on the ALM computer.

### Bitmap checkpoints

Bitmap checkpoints on objects containing text may fail if you create them using a Remote Desktop Connection and then run them locally, or if you create them locally and then run the checkpoint steps using a Remote Desktop Connection. In the run results, the image displayed when you click **View Difference** in the bitmap checkpoint results shows some text shapes.

**Workaround:** Enable the **Font smoothing** option in the Remote Desktop Connection application.

### Database checkpoints

- The format of captured values varies depending on the specific system settings. For example, date and time values may be set to different formats.  
**Workaround:** If you are running the test or scripted component on a different system than the one you used to record the test, confirm that the systems use the same format settings.
- When you create a database checkpoint on one machine and try to run it on different machine, you should have the same ODBC driver installed on both machines.

### File content checkpoints

- File content checkpoints for htm/html files generated dynamically by third-party Javascript code are not supported.
- When creating File Content checkpoints, UFT gathers information from the application under test. This can sometimes take up to two minutes.

### Text/text area checkpoints

Text and text area checkpoints are not supported if the text in the selected area is in a non-English language.

### XML checkpoints

- When executing an XML checkpoint on an XML file that contains > as a value, an error message may occur.

- When you add a new value node to an XML node, in some cases the new value may not be displayed.

**Workaround:** Close the Edit XML as Text dialog box and reopen it to display the new value node correctly.

- When inserting an XML file checkpoint on a file that cannot be loaded, or a file that is formatted incorrectly, you may receive an error message.
- Creating and running XML checkpoints for large XML documents may take a few minutes.

# Chapter 37: Output Values in GUI Testing

**Relevant for: GUI tests and components**

This chapter includes:

• <a href="#">Output Values Overview</a> .....	325
• <a href="#">Output Value Categories</a> .....	325
• <a href="#">Output Types and Settings</a> .....	327
• <a href="#">Viewing and Editing Output Values</a> .....	328
• <a href="#">Storing Output Values</a> .....	328
• <a href="#">How to Create or Modify an Output Value Step</a> .....	330
• <a href="#">Output Values for in the Keyword View</a> .....	333
• <a href="#">Output Value Properties Dialog Box</a> .....	333
• <a href="#">Add Existing Output Value Dialog Box</a> .....	334
• <a href="#">Advanced File Content Output Value Properties Dialog Box</a> .....	335

# Output Values Overview

## **Relevant for: GUI tests and components**

Your test or component can retrieve values and store them in output value objects. You can then use these values as input at a later stage in a run session.

An **output value** step is a step in which one or more values are captured at a specific point in your test or component and stored for the duration of the run session. The values can later be used as input at a different point in the run session.

You can output the property values of any object. You can also output values from text strings, table cells, databases, and .xml documents.

When you create output value steps, you can determine where the values are stored during the run session and how they can be used. During the run session, UFT retrieves each value at the specified point and stores it in the specified location. When the value is needed later in the run session, UFT retrieves it from this location and uses it as required.

Output values are stored only for the duration of the run session. When the run session is repeated, the output values are reset.

For details about using output values for each add-in environment installed with UFT, see "[Supported Output Values](#)" on page 1120.

To learn more, see:

- [Output Value Categories](#) ..... 325
- [Output Types and Settings](#) ..... 327
- [Viewing and Editing Output Values](#) ..... 328
- [Storing Output Values](#) ..... 328

## Output Value Categories

### **Relevant for: GUI tests and components**

You can create the following categories of output values:

- ["Standard Output Values" on the next page](#)
- ["File Content Output Values \(tests and scripted components only\)" on the next page](#)
- ["Table Output Values \(tests and scripted components only\)" on the next page](#)
- ["Text and Text Area Output Values \(tests and scripted components only\)" on the next page](#)
- ["Database Output Values \(tests and scripted components only\)" on page 327](#)
- ["XML Output Values \(tests and scripted components only\)" on page 327](#)
- ["Existing Output Values \(tests and scripted components only\)" on page 327](#)

## Standard Output Values

You can use standard output values to output the property values of most objects. For example, you can use standard output values to output text strings by specifying the **text** property of the object as an output value in a keyword component. In a Web-based application, the number of links on a Web page may vary based on the selections a user makes on a form on the previous page. You could create an output value in your test or scripted component to store the number of links on the page.

**Note for tests and scripted components:**

- You can also use standard output values to output the contents of table cells.
- You can use standard output values to output text strings by specifying the **text** property of the object as an output value.

For task details, see "[How to Create or Modify an Output Value Step](#)" on page 330.

## File Content Output Values (tests and scripted components only)

You can use file content output values to output the contents from any of the following file types:

• HTML	• Microsoft Word	• Text
• PDF	• RTF	

You can create output values from the entire contents of a file, or from a part of it. During the run session, UFT retrieves the current data from the file and outputs the values according to the settings that you specified.

For task details, see "[How to Create or Modify an Output Value Step](#)" on page 330.

## Table Output Values (tests and scripted components only)

Table output values are a subset of standard output values, described above. You can use table output values to output the contents of table cells. For some types of tables, you can specify a row range from which to choose the table cells. During the run session, UFT retrieves the current data from the specified table cells according to the settings that you specified and outputs the values to the Data pane.

For task details, see "[How to Create or Modify an Output Value Step](#)" on page 330.

## Text and Text Area Output Values (tests and scripted components only)

You can use text output values to output text strings displayed in an application. When creating a text output value, you can output a part of the object's text. You can also specify the text before and after the output text.

You can use text area output values to output text strings displayed within a defined area of a screen in a Windows-based application.

For example, suppose that you want to store the text of any error message that appears after a specific step in the Web application you are testing. Inside the **If** statement, you check whether a window exists with a known title bar value, for example Error. If it exists, you output the text in this window (assuming that the window size is the same for all possible error messages).

**Note:** You can create a text area output value only while recording on Windows-based applications.

For task details, see "[How to Create or Modify an Output Value Step](#)" on page 330.

### **Database Output Values (tests and scripted components only)**

You can use database output values to output the value of the contents of database cells, based on the results of a query (result set) that you define on a database. You can create output values from the entire contents of the result set, or from a part of it. During the run session, UFT retrieves the current data from the database and outputs the values according to the settings that you specified.

For task details, see "[How to Create or Modify an Output Value Step](#)" on page 330.

### **XML Output Values (tests and scripted components only)**

You can use XML output values to capture and output the values of XML elements and attributes in XML documents.

For example, suppose that an XML document in a Web page contains a price list for new cars. You can output the price of a particular car by selecting the appropriate XML element value to output.

For task details, see "[How to Create or Modify an Output Value Step](#)" on page 330.

### **Existing Output Values (tests and scripted components only)**

When you insert an existing output value in your test or scripted component, consider which output values should be used in multiple locations in your test or component. Each time an output value step is performed, the value contained in the output value is overwritten with the new output value. You should insert an existing output value only if the stored value will no longer be needed later in the run session when the output value object is used again.

For details, see "[Add Existing Output Value Dialog Box](#)" on page 334.

## Output Types and Settings

### **Relevant for: GUI tests and components**

In the Output Options Dialog Box you can set the output type and settings for each value, to determine where it is stored and how it can be used during the run session. When the output value step is reached, UFT retrieves each value selected for output and stores it in the specified location for use later in the run session.

When you create a new output value step, UFT assigns a default definition to each value selected for output. For details, see the Default Output Definitions (tests only) below.

You can change the current output definition for the selected value by selecting a different output type and/or changing the output settings in the Output Options Dialog Box.

## Default Output Definitions (tests only)

When you initially select a value for output, UFT generates a default output definition for the value.

When you output a value for a step, the following occurs:

- If at least one output parameter is defined in the action, the default output type is **Test/action parameter** and the default output name is the first output parameter displayed in the Parameters Tab (Action Properties Dialog Box).
- If no output parameters are defined in the action, the default output type is **DataTable**, and UFT creates a new DataTable output name based on the selected value.

The output value is created in the Global sheet of the Data pane.

For details on creating output parameters for actions, see "[Display / Modify Action Data](#)" on page [106](#).

## Viewing and Editing Output Values

### Relevant for: GUI tests and components

In the Keyword View, an output value step is displayed as follows:

- The **Operation** column displays Output.
- The **Value** column displays CheckPoint followed by the name assigned to the output value.

In the Editor, an output value statement is displayed with the following syntax:

```
Object.Output CheckPoint(Name)
```

You can view or edit the output value or its details in the relevant Output Value Properties dialog box, by right-clicking the step and choosing **Output Value Properties**. Alternatively, you can click the step in the **Value** column in the Keyword View and then click the **Output Properties** button .

For details on the options available in the various Output Value Properties dialog boxes, see "[Output Value Properties Dialog Box](#)" on page [333](#).

## Storing Output Values

### Relevant for: GUI tests and scripted GUI components

When you define an output value, you can specify where and how each value is stored during the run session.

You can output a value to:

- "[Test and Action Parameters \(tests only\)](#)" below
- "[Run-time Data Table](#)"
- "[Environment Variables](#)"

## Test and Action Parameters (tests only)

You can output a value to an action parameter, so that values from one part of a run session can be used later in the run session, or be passed back to the application that ran (called) the test.

For example, suppose you are testing a shopping application that calculates your purchases and automatically debits your account with the amount that you purchased. You want to test that the application correctly debits the purchase amount from the account each time that the action is run with a different list of items to purchase. You could output the total amount spent to an action parameter value, and then use that value later in your run session in the action that debits the account.

## Run-time Data Table

The option to output a value to the run-time data table is especially useful with a **data-driven** test, action, or scripted component that runs several times. In each repetition, or **iteration**, UFT retrieves the current value and stores it in the appropriate row in the run-time data table.

### Example

Suppose you are testing a flight reservation application and you design a test to create a new reservation and then view the reservation details. Every time you run the test, the application generates a unique order number for the new reservation. To view the reservation, the application requires the user to input the same order number. You do not know the order number before you run the test.

To solve this problem, you output a value to the Data pane for the unique order number generated when creating a new reservation. Then, in the View Reservation screen, you use the column containing the stored value to insert the output value into the order number input field.

When you run the test, UFT retrieves the unique order number generated by the site for the new reservation and enters this output value in the run-time data table. When the test reaches the order number input field required to view the reservation, UFT inserts the unique order number stored in the run-time data table into the order number field.

## Environment Variables

When you output a value to an internal user-defined environment variable, you can use the environment variable input parameter at a later stage in the run session.

### Example

Suppose you are testing an application that prompts the user to input an account number on a Welcome page and then displays the user's name. You can use a text output value to capture the value of the displayed name and store it in an environment variable.

You can then retrieve the value in the environment variable to enter the user's name in other places in the application. For example, in an Order Checkbook Web page, which for security reasons requires users to enter the name to appear on the checks, you could use the value to insert the user's name into the **Name** edit box.

**Note:**

- Output values are stored only for the duration of the run session, and are not saved with the test or component. If you select to output a value to an existing parameter, a data table column, or an environment variable, the existing value is overwritten when the output value step runs. When the run session ends, the original value is restored.
- You can output values only to internal user-defined environment variables and not to external or built-in environment variables, which are read-only.

## How to Create or Modify an Output Value Step

**Relevant for: GUI tests and components**

This task describes how to insert an output value step while recording or editing your test or component. You can also add an existing output value to a test or scripted component. It is generally more convenient to define output values after creating the initial test or component.

For details on supported output values per add-in environment, see "[GUI Checkpoints and Output Values Per Add-in](#)" on page 1116

This task describes both the general process for inserting a new output value step to your test or component and the prerequisites and considerations for the different types of output values.

This task includes the following steps:

- "[Important information before inserting the output value step](#)" below
- "[Insert a new standard output value step while recording your test or component](#)" on the next page
- "[Insert a new output value step using the Active Screen while editing \(tests and scripted components only\)](#)" on the next page
- "[Insert an existing output value step while editing \(tests and scripted components only\)](#)" on page 332
- "[Set the options for the output value object](#)" on page 332

### **Important information before inserting the output value step**

**Object**

- During an editing session, make sure the object is visible in your application before inserting an output value

<b>visibility in application</b>	step on it. <ul style="list-style-type: none"> <li>• <b>For file content output values:</b> The source file must be located on the file system.</li> </ul>
<b>Availability</b>	<ul style="list-style-type: none"> <li>• Recording sessions</li> <li>• Editing sessions</li> <li>• Active Screen</li> </ul>
<b>Active Screen</b>	Make sure that the Active Screen contains sufficient data for the object for which you want to define an output value.
<b>Important Information</b>	<ul style="list-style-type: none"> <li>• Output values can be viewed in the following modes: <ul style="list-style-type: none"> <li>• <b>Simple Mode:</b> Displays only the basic properties and expected values of the output value.</li> <li>• <b>Advanced Mode:</b> Displays all supported properties and expected values of the output value.</li> </ul> </li> <li>• <b>For text/text area checkpoints:</b> <ul style="list-style-type: none"> <li>• Before you create a text or text area output value for a Windows-based application, make sure you configure the required capture settings in the <b>Text Recognition</b> pane of the Options dialog box (<b>Tools &gt; Options &gt; GUI Testing tab &gt; Text Recognition node</b>).</li> <li>• When you use text-area selection to capture text displayed in a Windows application, it is often advisable to define a text area larger than the actual text you want UFT to use as an output value. During the run session, UFT outputs the selected text, within the defined area, according to the settings you configured.</li> <li>• Because text may change its position during run sessions, make sure that the area defined is large enough so that the output text is always within its boundaries. For details, see "<a href="#">Text Recognition Overview</a>" on page 301.</li> </ul> </li> </ul>

## Insert a new standard output value step while recording your test or component

1. Start a recording session.
2. Do one of the following:
  - Select **Design > Output Value > Standard Output Value**.
  - In the record toolbar, click **Insert Checkpoint or Output Value** down arrow  and select **Standard Output Value**.  
The pointer changes into a pointing hand.

If necessary, select the object or object sections you want to include in the output value. The specific dialog that opens differs depends on the type of output value selected.

## Insert a new output value step using the Active Screen while editing (tests and scripted components only)

1. Do one of the following:

- Select **View > Active Screen**, click a step whose Active Screen contains the object for which you want to specify an output value, and right-click the object for which you want to specify an output value and select **Insert Output Value**.
- Right-click the step and select **Insert Output Value**.  
If the location you clicked is associated with more than one object, the Object Selection Dialog Box opens.  
If necessary, select the object or object sections you want to include in the output value. The specific dialog that opens differs depends on the type of output value selected.

**Note if you are creating a text/text area output value:**

- To output text value while editing, you first highlight a text string in the Active Screen then right-click the string, and select **Insert Text Output Value**.
- When you output a text area value, you first define the area containing the text you want UFT to check. When you select Text Area Output Value, the mouse turns into a crosshairs pointer. Click and drag the crosshairs pointer to define this area. Release the mouse button after outlining the area required. For more details, see the section on text area output values in ["Output Value Categories" on page 325](#), and the important details in Output Value Properties Dialog Box.

**Tip:** Hold down the left mouse button and use the arrow keys to make precise adjustments to the defined area.

## Insert an existing output value step while editing (tests and scripted components only)

1. Select the step after which you want to insert the output value.
2. Select **Design > Output Value > Existing Output Value**. The Add Existing Output Value Dialog Box opens, enabling you to select the test object from which you want to output values.

## Set the options for the output value object

**For tests and scripted components:** Set the options for the output value, including the values to add to the output value step.

**For keyword components:** Specify the settings in the Output Value Properties Dialog Box.

# Output Values for in the Keyword View

## Relevant for: GUI components

When working with components, you define the output type and settings for the output value in the **Value** cell. These determine where the output value is stored and how it is used during the component run session. When the output value step is reached, UFT retrieves each value selected for output and stores it in the specified location for use later in the run session.

When you create a new output value step, UFT assigns a default definition to each value selected for output. When you output a value for a step in a business component:

- If at least one output component parameter is defined in the component, the default output type is **Component parameter** and the default output name is the first output parameter displayed in the Parameters pane of the Business Component Settings dialog box.
- If no output component parameter is defined in the component, the default output type is **Local parameter** and the default name is **p\_Local**.

You modify the output parameter, as required. If you select a local parameter, you can modify it directly in the Output Options Dialog Box. If you select a component parameter, the details for the output parameter are read-only. You can modify the parameter details in the Parameters pane of the Business Component Settings dialog box.

If, after you specify an output value, you choose not to save the output value, you can cancel it. Click in the **Output** cell. Then click the **Cancel** button  or press the **DELETE** key.

# Output Value Properties Dialog Box

## Relevant for: GUI tests and components

This dialog box enables you to define and modify properties for a selected output value.

<b>To access</b>	<ol style="list-style-type: none"><li>1. Ensure that a GUI action or component is in focus in the document pane.</li><li>2. Use one of the following:<ul style="list-style-type: none"><li>• <a href="#">Insert a new output value step</a> and select an object from your application.</li><li>• During a recording session, click the <b>Insert Checkpoint or Output Value</b> down arrow and select the relevant type.</li><li>• Right-click an existing output value step and select <b>Output Value Properties</b>.</li><li>• In the local or shared object repository, click an existing output value object. The output value details are displayed on the right side of the object repository window, in the <b>Object Details</b> area.</li></ul></li></ol>
<b>Important information</b>	<ul style="list-style-type: none"><li>• If you insert an output value on a Web page, the Page Output Value Properties dialog box opens. This dialog box is identical to the Output Value Properties dialog box, except that it contains two additional areas,</li></ul>

	<p><b>HTML verification</b> and <b>All objects in page</b>. These options are relevant only for checkpoints and are disabled when defining output values.</p> <ul style="list-style-type: none"> <li>You cannot view or edit the advanced properties of output value objects when working in ALM. Therefore, if one or more advanced properties are selected in an output value object, and an ALM user views its properties in ALM, the dialog box displays text indicating that some properties are not shown.</li> </ul>
<b>Relevant tasks</b>	<a href="#">"How to Create or Modify an Output Value Step" on page 330</a>

## Add Existing Output Value Dialog Box

### Relevant for: GUI tests and scripted GUI components

This dialog box enables you to select the test object to which you want to add an existing output value.

<b>To access</b>	<ol style="list-style-type: none"> <li>Ensure that a GUI action or component is in focus in the document pane.</li> <li><a href="#">Insert a new output value step</a> and select the <b>Existing Output Value</b> option.</li> </ol>
<b>Important information</b>	This option is available only if at least one of the object repositories associated with the current action (including the local object repository) contains at least one output object.
<b>Relevant tasks</b>	<a href="#">"How to Create or Modify an Output Value Step" on page 330</a>

### Properties Grid Area

UI Element	Description
<b>Display only output values relevant to the selected test object</b>	<p>Instructs UFT to determine which output value objects from the current action's object repositories are relevant for the selected object (based on the output value type and the properties selected to output in the output value object) and display only those output value objects in the Output Values list.</p> <p><b>Note:</b> When using this option, it is recommended to open your application and display the selected object so that UFT can accurately determine all of the output values that can apply to that object.</p>
<b>Output values</b>	<p>The output values available for insertion.</p> <p>If the <b>Display only output values relevant to the selected test object</b> option is cleared, this list includes all output value objects from all object repositories associated with the current action.</p> <p>If the <b>Display only output values relevant to the selected test object</b> option is selected, this list displays only the relevant output value objects as described above.</p>
<b>Check box</b>	Enables you to select one or more property for the object, and specify the output options for each property value you select.
<b>Type</b>	<p>The  icon indicates that the value of the property is currently a constant.</p> <p>The  icon indicates that the value of the property is currently a test or action parameter.</p>

UI Element	Description
	The  icon indicates that the value of the property is currently a DataTable parameter. The  icon indicates that the value of the property is currently an environment variable parameter. The  icon indicates that the value of the property is currently a random number parameter.
<b>Property</b>	The name of the property.
<b>Value</b>	The expected value of the property.

## Advanced File Content Output Value Properties Dialog Box

### Relevant for: GUI tests and scripted GUI components

This dialog box enables you to configure how the output value step is performed.

<b>To access</b>	In the <a href="#">"Output Value Properties Dialog Box"</a> (described on page 333), select <b>Advanced</b> in the toolbar.
<b>Important information</b>	When UFT performs a file content output value step, it locates the output value text within the line by comparing all of the text on that line. The options in this dialog box instruct UFT how to treat the text on every line in the file in which an output value is specified. (You can create multiple output values in a single file.)
<b>Relevant tasks</b>	<a href="#">"How to Create or Modify an Output Value Step" on page 330</a>
<b>See also</b>	<a href="#">"Output Value Properties Dialog Box" on page 333</a>

User interface elements are described below:

UI Element	Description
<b>Ignore spaces</b>	Ignores extraneous spaces on the same line as the output text when comparing the expected content with the actual content. This is useful when there might potentially be one or more spaces in the text preceding or following the checked text, and you do not want the checkpoint to fail because of this.  <b>Note:</b> This setting is not relevant for missing spaces.
<b>Match case</b>	Conducts a case-sensitive check of all the text on the same line as the output value text when comparing the expected content with the actual content.  <b>Default:</b> Selected

# Chapter 38: Parameterizing Object Values

**Relevant for: GUI tests and scripted GUI components**

This chapter includes:

• Parameterizing Values Overview .....	338
• Default Parameter Values .....	339
• How to Parameterize Values for Operations or Local Objects .....	340
• How to Parameterize a Checkpoint Property Value .....	341
• Data Tables and Sheets in GUI Tests and Components .....	342
• Using Different Data Tables .....	343
• Data Table Objects, Methods, and Properties .....	344
• Formulas in Data Tables .....	344
• How to Define and Manage Data Tables in a GUI Test .....	346
• How to Insert Formulas into Data Tables for Use in Checkpoints in a GUI Test .....	348
• How to Import Data Into a GUI Test Using Microsoft Query .....	349
• Data Table Parameters .....	349
• When to Choose Global or Action Data Table Parameters .....	352
• Environment Variable Parameters .....	353
• Built-in Environment Variables .....	356
• How to Use User-Defined External Environment Variables .....	357
• How to Create an External Environment Variables File .....	358
• Automatically Parameterizing Steps .....	360
• Guidelines and Considerations for Automatically Parameterizing Steps .....	360
• Data Driver .....	364
• Test and Action Parameters .....	364
• Passing Test and Action Parameter Values .....	367
• Sharing Action Information .....	368
• Using Action Parameters in Steps in the Editor .....	370
• How to Use Action Parameters .....	371
• Regular Expressions Overview .....	372
• Regular Expressions for GUI Test and Component Object Property Values .....	374
• Regular Expressions in GUI Test and Component Checkpoints .....	374

- [Regular Expression Characters and Usage Options](#) ..... 375
- [Regular Expressions in the Find and Replace Dialog Boxes](#) ..... 380

# Parameterizing Values Overview

## Relevant for: GUI tests only

You can enhance your test by parameterizing the values that it uses. A **parameter** is a variable that is assigned a value from an external data source or generator.

You can parameterize values of:

- Checkpoints.
- Object properties for a selected step.
- Operation arguments defined for a selected step.
- One or more properties of an object stored in the local object or the Object Repository Window.

## Example

Your application may include a form with an edit box into which the user types the user name. You may want to test whether your application reads this information and displays it correctly in a dialog box. You can insert a text checkpoint that uses the built-in environment variable for the logged-in user name, to check whether the displayed information is correct.

**Note:** When you parameterize the value of an object property for a local object, you are modifying the test object description in the local object repository. Therefore, all occurrences of the specified object within the action are parameterized. For details on the local object repository, see ["Test Objects in Object Repositories" on page 178](#).

You can parameterize the values in steps or the values of action parameters using one of the following parameter types:

- **Test/action parameters.** Test parameters enable you to use values passed from your test. Action parameters enable you to pass values from other actions in your test. For details, see ["Test and Action Parameters" on page 364](#).
- **DataTable parameters.** Enable you to create a data-driven test (or action) that runs several times using the data you supply. In each repetition, or iteration, UFT uses a different value from the Data pane. For details, see ["Data Table Parameters" on page 349](#).
- **Environment variable parameters.** Enable you to use variable values from other sources during the run session. These may be values you supply, or values that UFT generates for you based on conditions and options you choose. For details, see ["Environment Variable Parameters" on page 353](#).
- **Random number parameters.** Enable you to insert random numbers as values in your test. For example, to check how your application handles small and large ticket orders, you can instruct UFT to generate a random number and insert it in a **number of tickets** edit box.

## Tip:

- If you want to parameterize all the operation arguments in your test or in one or more actions of a test, consider using the Automatically parameterize steps option. For details, see ["Automatically Parameterizing Steps" on page 360](#).
- If you want to parameterize the same value in several steps in your test, consider using the Data Driver rather than adding parameters manually. For details, see ["Data Driver" on page 364](#).
- When you use the Step Generator to add new steps, you can parameterize the values for the operation you select. For details, see ["How to Insert Steps Using the Step Generator" on page 726](#).
- You can also parameterize identification property values of test objects in the object repository using repository parameters. For details, see ["Working with Repository Parameters" on page 195](#).

## Default Parameter Values

### Relevant for: GUI tests only

When you select a value that has not yet been parameterized, UFT generates a default parameter definition for the value. The following table describes how the default parameter settings are determined:

When parameterizing	Condition	Default parameter type	Default parameter name
A value for a step or a checkpoint in an action	At least one input action parameter is defined in the current action	Action parameter	The first input parameter displayed in the Parameter Values Tab of the Action Call Properties Dialog Box
An input action parameter value for a nested action	At least one input action parameter is defined for the action calling the nested action	Action parameter	The first input parameter displayed in the Parameter Values Tab of the Action Call Properties Dialog Box of the calling action
An input action parameter value for a top-level action call	At least one input parameter is defined for the test	Test parameter	The first input parameter displayed in the Parameter Values Tab of the Action Call Properties Dialog Box

If the relevant condition described above is not true, the default parameter type is Data Pane. If you accept the default parameter details, UFT creates a new data table parameter with a name based on the selected value. Data table parameters are created in the Global sheet.

# How to Parameterize Values for Operations or Local Objects

## Relevant for: GUI tests and scripted GUI components

This task describes different ways to parameterize the values for operations in a step or objects in the local object repository.

This task includes:

- "[Parameterize a value for an operation using the parameterization icon](#)" below
- "[Use the Data Driver to parameterize some or all of the occurrences of a constant value in your test \(tests only\)](#)" on the next page
- "[Enter input and output parameters as values in the Editor \(tests only\)](#)" on the next page

## Parameterize a value for an operation using the parameterization icon

1. To parameterize a value for an operation using the parameterization icon, in the Keyword View click in the **Value** column of the required step.
  - To parameterize property values for local objects, do one of the following:
    - Right-click a step and select **Object Properties**. The Object Properties Dialog Box opens.
    - Open the Object Repository Window and select the object.
2. Click the parameterization icon .
  - To parameterize a value for an operation, in the Keyword View, click the parameterization icon  for the value that you want to parameterize.
  - To parameterize property values for local objects, in the Object Repository click in the **Value** cell for the property that you want to parameterize, and click the parameterization icon .

The parameters list opens, displaying the available parameter types and parameters.

3. In the parameters list, select the type of parameter you want:
  - **Test/action** parameter
  - **Data Table** parameter
  - **Environment** parameter
  - **Random number** parameter
4. If you need to add a parameter, click the **Add New Parameter** button at the bottom of the parameter list.

5. Parameterize the value using the Value Configuration Options Dialog Box.

### Use the Data Driver to parameterize some or all of the occurrences of a constant value in your test (tests only)

Use the Data Driver Dialog Box (**Tools > Data Driver**).

### Enter input and output parameters as values in the Editor (tests only)

You can enter input and output parameters as values in the Editor using the **Parameter** utility object. For details, see "Test and Action Parameters" on page 364.

## How to Parameterize a Checkpoint Property Value

### Relevant for: GUI tests and scripted GUI components

This task describes how to parameterize the values for properties in checkpoints. You can parameterize the values for properties in checkpoints in one of the following ways:

- "Parameterize a value for a property in a checkpoint using the Checkpoint Properties dialog box" below
- "Use the Data Driver to parameterize some or all of the occurrences of a constant value in your test (tests only)" below
- "Enter input and output parameters as values in the Editor (tests only)" on the next page

### Parameterize a value for a property in a checkpoint using the Checkpoint Properties dialog box

1. Open the dialog box for the checkpoint properties.

Open the dialog box for the checkpoint properties in one of the following ways:

- Right-click the checkpoint and select **Checkpoint Properties**.
- Open the Object Repository Window and select the checkpoint.

2. Use the options in the Configure Value area to parameterize the value.

- a. Select the property whose value you want to parameterize from the table of properties.
- b. In the **Configure value** area, select **Parameter**.
- c. Click the **Parameter Options** button  . The Parameter Options Dialog Box opens:

### Use the Data Driver to parameterize some or all of the occurrences of a constant value in your test (tests only)

Select **Tools > Data Driver**.

## Enter input and output parameters as values in the Editor (tests only)

You can enter input and output parameters as values in the Editor using the **Parameter** utility object. For details, see "[Test and Action Parameters](#)" on page 364.

# Data Tables and Sheets in GUI Tests and Components

### Relevant for: GUI tests and components

In the Data pane, when you are working with GUI tests or components, UFT uses Excel data sheets to store the tests data. Depending on whether you are working with tests or components, you can use different types of data sheets:

Type of Sheet	Test or component?	Description
<b>Global sheet</b>	Tests	<p>The global sheet is the "master" sheet for the test, containing data that is available to all actions/test steps in the test.</p> <p><b>Note:</b> If you save your test in ALM or use the <a href="#">Data Awareness</a> feature, you must store the test data in the Global sheet.</p> <p>For each row in the Global data sheet, UFT can run an iteration of the entire test.</p>
<b>Action sheet</b>	Tests	<p>The action sheet contains the data relevant for a specific action. Each action in your test has its own action tab, and the data stored in this sheet is available only to the steps contained in the action.</p> <p>For each row in the Action data sheet, UFT can run an iteration of the action.</p>
<b>Component sheet</b>	Components	<p>The component sheet contains the data relevant for a specific component. When you create a component, it is created with one sheet, and the data in this component is available only to the steps contained in the component (even if the component is later added to a BPT test.)</p>

When you run the test, UFT creates a **run-time** data table—a live version of the data table associated with your test. During the run session, UFT displays the run-time data in the Data pane so that you can see any changes to the data table as they occur.

When the run session ends, the run-time data table closes, and the Data pane again displays the stored design-time data table. Data entered in the run-time data table during the run session is not saved with the test. The final data from the run-time data table is displayed in the run results (with a link to open the data table).

### To learn more, see:

- [Using Different Data Tables](#) ..... 343
- [Data Table Objects, Methods, and Properties](#) ..... 344
- [Formulas in Data Tables](#) ..... 344

## Using Different Data Tables

### Relevant for: GUI tests and scripted GUI components

When working with tests, the data table is saved with your test by default as an .xls or .xlsx file. By default, UFT uses the entered values for the test's data table when running the test.

However, when running a test, you can instruct UFT to use different data tables/sheets than the ones saved with your test. For example, you can save the entered data table in another location and instruct the test to use this data table when running a test. You can also tell UFT to use an external saved data table. You specify the data sheet name and location for the data table in the Resources pane of the Test Settings dialog box.

There are different scenarios in which you may need to save or access different data tables:

### When to save or use a data table in a different location

- If you want to run the same test with different sets of input values. For example, you can test the localization capabilities of your application by running your test with a different data table file for each language you want to test. You can also vary the user interface strings that you check in each language by using a different environment parameter file each time you run the test.
- If you need the same input information for different tests. For example, you can test a Web version and a standard Windows version of the same application using different tests, but the same data table file.

### When to save a run-time data table

If it is important for you to save the resulting data from the run-time data table, you can insert a **DataTable.Export** statement to the end of your test to export the run-time data table to a file. You can then import the data to the design-time data table using the data table's **File > Import from File** menu. Alternatively you can add a **DataTable.Import** statement to the beginning of your test to import the run-time data table that was exported at the end of the previous run session. For more details on these methods, see the **DataTable** object in the Utility Objects section of the *HP UFT Object Model Reference for GUI Testing*.

### Saving data tables in an ALM project

When working with ALM, you must save the data table file in the Test Resources module in your ALM project before you specify the data table file in the Resources pane of the Test Settings dialog box.

You can add a new or existing data table file to your ALM project. Note that adding an existing data table file from the file system to an ALM project creates a copy of the file. Thus, once you save the file to the project, changes made to the ALM data table file will not affect the data table file in the file system and vice versa.

## Data Table Objects, Methods, and Properties

### Relevant for: GUI tests and scripted GUI components

When creating or running a test, you can use a number of data table methods that enable you to retrieve information on the run-time data table and to set the value of cells in the run-time data table. You enter these statements manually in the Editor.

There are three different types of objects:

This object enables you to work with the run-time data table, including adding and deleting sheets, importing and exporting sheets, and working with the properties of the data sheet.
This object enables you to work with a specific sheet in the run-time data table. You can add sheets, get a specific sheet, see the properties of a selected data sheet, or add parameters to a sheet.
This object enables you to access a specific column in a data sheet. You can add a data table parameter or get a specific parameter.

For more details on the data table methods, see the Utility Objects section of the *HP UFT Object Model Reference for GUI Testing*.

## Formulas in Data Tables

### Relevant for: GUI tests and scripted GUI components

When, working with data tables, you can use Microsoft Excel formulas in your data table. This enables you to create contextually relevant data during the run session. You can also use formulas as part of a checkpoint to check that objects created on-the-fly (dynamically generated) or other variable objects in your Web page or application have the values you expect for a given context.

When you use formulas in a data table to compare values (generally in a checkpoint), the values you compare must be of the same type, for example integers, strings, and so forth. When you extract values from different places in your applications using different functions, the values may not be of the same type. Although these values may look identical on the screen, a comparison of them will fail, since, for example, 8.2 is not equal to "8.2".

For more details on using worksheet functions, see the Microsoft Excel documentation.

## Formulas in Data Tables for Use in Iterations

You can enter formulas rather than fixed values in the cells of a parameter column.

For example, suppose you want to parameterize the value for a WebEdit object that requires a date value no earlier than today's date. You can set the cells in the **Date** column to the date format, and enter the =NOW() Excel formula into the first row to set the value to today's date for the first iteration.

Then you can use another formula in the remaining rows to enter the above date plus one day, as shown below. By using this formula you can run the test on any day and the dates will always be valid.

A2	=A1+1	
	Date	B
1	1/3/2006	
2	1/4/2006	
3	1/5/2006	
4	1/6/2006	
5	1/7/2006	
6	1/8/2006	
7	1/9/2006	
8	1/10/2006	
9		

#### TEXT or VALUE Functions Used to Convert Values from One Type to Another

- **TEXT(value, format).** Returns the textual equivalent of a numeric value in the specified format, so that, for example the formula =TEXT(8.2, "0.00") is "8.20".
- **VALUE(string).** Returns the numeric value of a string, so that, for example, =VALUE("\$8.20") is 8.20.

For more details on using parameters, see ["Parameterizing Object Values" on page 336](#).

#### Formulas in Data Tables for Use in Checkpoints

You can use a formula in a checkpoint to confirm that an object created on-the-fly (dynamically generated) or another variable object in your Web page or application contains the value it should for a given context. For example, suppose a shopping cart Web site displays a price total. You can create a text checkpoint on the displayed total value and use a data table formula to check whether the site properly computes the total, based on the individual prices of the products selected for purchase in each iteration.

When you use the data table formula option with a checkpoint, two columns are created in the data table. The first column contains a default checkpoint formula. The second column contains the value to be checked in the form of an output parameter. The result of the formula is Boolean—TRUE or FALSE.

A1	=\$B1=337	
	Total_Price	Total_Price_out
1	TRUE	337
2		

A FALSE result in the checkpoint column during a test run causes the test to fail.

After you finish adding the checkpoint, you can modify the default formula in the first column to perform the check you need.

# How to Define and Manage Data Tables in a GUI Test

## Relevant for: GUI tests and scripted GUI components

This task describes the different options for defining a new data table for your test.

This task includes:

- "Add an external data table file" below
- "Manually enter or import information in the Data table" below
- "Import information into the data table" below
- "Add a data table file to your ALM project" on the next page
- "Define the number of iterations for an action or test" on the next page
- "Change a column name" on page 348
- "Use an autofill list" on page 348

## Add an external data table file

1. Select **File > Settings** to open the Test Settings dialog box.
2. In the Settings dialog box, select the **Resources** node.
3. In the Resources pane, in the Data Table section, select one of the following:
  - **Default location:** saved with the test in the same directory
  - **External file:** click the Browse button and navigate to the directory where the external data table is stored.

**Note:** If you select an external file as your data table, make sure that:

- The column names in the external data table match the parameter names in the test.
- The sheets in the external data table match the action names in the test.

## Manually enter or import information in the Data table

Edit information in the Data pane by typing directly into the table cells.

## Import information into the data table

Do one of the following:

- Right-click in the Data pane and select **File > Import from File** from the Data pane commands.
- Right-click in the Data pane and select **Sheet > Import > From File** from the Data pane commands.

**Note:** You can also import data saved in Microsoft Excel, tabbed text file (.txt), or ASCII format.

## Add a data table file to your ALM project

1. Prerequisite: Make sure you have an accessible Microsoft Excel file with an .xls or .xlsx extension.
2. In ALM, create a new data table resource and then upload the .xls or .xlsx file you created in the previous step to the project's **Test Resources** module. For more details, see the *HP Application Lifecycle Management User Guide*.
3. In UFT, select File > Settings to open the Test Settings dialog box.
4. In the **Resources** pane of the Test Settings dialog box, in the Data table section, select **Other location** and click the **Browse** button to locate the data table file.
5. Enter your data as needed. When you save the test, UFT saves the data table file to the ALM project.

## Define the number of iterations for an action or test

Do one of the following:

<b>For actions</b>	<ol style="list-style-type: none"> <li>1. In the canvas, right-click an action and select <b>Action Call Properties</b>. The Action Call Properties dialog box opens.</li> <li>2. In the <b>Run</b> tab of the Action Call Properties dialog box, select the appropriate option:           <ul style="list-style-type: none"> <li>• <b>Run one iteration only</b></li> <li>• <b>Run on all rows</b> - UFT runs an iteration for each row in the action's data sheet</li> <li>• <b>Run from row &lt;X&gt; to row &lt;X&gt;</b> - UFT runs an iteration for each of the defined rows</li> </ul> </li> </ol>
<b>For tests</b>	<ol style="list-style-type: none"> <li>1. Select File &gt; Settings to open the Settings dialog box.</li> <li>2. In the <b>Run</b> pane, select the appropriate option:           <ul style="list-style-type: none"> <li>• <b>Run one iteration only</b></li> <li>• <b>Run on all rows</b> - UFT runs an iteration for each row in the action's data sheet</li> <li>• <b>Run from row &lt;X&gt; to row &lt;X&gt;</b> - UFT runs an iteration for each of the defined rows</li> </ul> </li> </ol>

**Note:** If you want to prevent UFT from running an iteration on a row when the **Run on all rows** option is selected, you must delete the entire row from the data table. To do this, do one of the following:

- Select the row, right-click in the table, and select **Edit > Delete** from the data table's context menu (or use CTRL+K). This restores the bottom grid line from black to gray.
- Use the Clear option from the table's **Edit** menu (or CTRL+X).
- Select a cell and press **Delete** on the keyboard. The data is deleted from the cells, but the row is not deleted and the black line remains. This means that UFT will run an iteration for this row even though there is no data in it.

## Change a column name

In the data pane, double-click a column header and enter the new column name in the Change Parameter Name dialog box.

## Use an autofill list

1. In the Data pane, right-click a cell and select **Data > Autofill**. The Autofill Lists Dialog box opens.
2. In the AutoFill Lists dialog box, select the type of autofill list.
3. Enter the first item into a cell in the table (item names are case-sensitive).
4. Fill the cells by doing one of the following:
  - Drag the cursor, from the bottom right corner of the cell up or right to fill the next row or column in the sheet.
  - Highlight the item and press **ENTER** to automatically fill the next row of the sheet.
  - Highlight the item and press **TAB** to automatically fill the next column of the sheet.

# How to Insert Formulas into Data Tables for Use in Checkpoints in a GUI Test

### Relevant for: GUI tests and scripted GUI components

This task describes how to insert formulas into the data table for use in a checkpoint.

#### To use a formula in a checkpoint:

1. Select the object or text for which you want to create a checkpoint and open the Insert Checkpoint dialog box as described in "[How to Insert a Checkpoint in a GUI Test or Component](#)" on page 309.
2. In the **Configure value** area, click **Parameter**.
3. Set the parameter options, in the Parameter Options Dialog Box.
4. Specify your other checkpoint setting preferences as described in the Checkpoint Properties Dialog Box.
5. Highlight the value in the first (formula) column to view the formula and modify the formula to fit your needs.
6. If you want to run several iterations, add the appropriate formula in subsequent rows of the formula column for each iteration in the test or action.

**Tip:** You can encode passwords to use the resulting strings as method arguments or data table parameter values.

You can also encrypt strings in data table cells using the **Encrypt** option in the Data pane menu.

# How to Import Data Into a GUI Test Using Microsoft Query

## **Relevant for: GUI tests and scripted GUI components**

You can use Microsoft Query to choose a data source and define a query within the data source. For details on supported versions of Microsoft Query, see the *HP Unified Functional Testing Product Availability Matrix*.

### **To choose a data source and define a query in Microsoft Query:**

1. Open the Database Query Wizard and select **Create query using Microsoft Query**.
2. When Microsoft Query opens, choose a new or an existing data source.
3. Define a query.
4. In the Finish screen of the Query Wizard, select one of the following:
  - **Exit and return to HP Unified Functional Testing > Finish** to exit Microsoft Query.
  - **View data or edit query in Microsoft Query > Finish** to view the data.

After viewing or editing the data, select **File > Exit and return to HP Unified Functional Testing** to close Microsoft Query and return to UFT.

For details on working with Microsoft Query, see the Microsoft Query documentation

# Data Table Parameters

## **Relevant for: GUI tests and scripted GUI components**

You can supply the list of possible values for a parameter by creating a data table parameter. **Data table parameters** enable you to create a data-driven test, or action that runs several times using the data you supply. In each repetition, or **iteration**, UFT uses a different value from the Data pane (taken from the subsequent row in the Data pane).

When you create a new data table parameter, a new column is added at the end of the Data pane and the current value you parameterized is placed in the first row. If you parameterize a value and select an existing data table parameter, the values in the column for the selected parameter are retained and are not overwritten by the current value of the parameter.

**Note:** If you parameterize a value that is defined as a variant value, then when UFT retrieves the value from the Data pane, it retrieves it as a string. This occurs even if you enter a numeric value in the Data pane. For example, if you parameterize the argument of a step such as:  
`WpfWindow("MyWindow").WpfComboBox("cb").Select 1`  
and you enter the value 1 in the Data pane, then when the step runs, it retrieves the value as a

string: "1", and the step fails.

A data table comprises:

- **Columns.** Each column in the table represents the list of values for a single data table parameter.
- **Column headers.** The data table parameter names.
- **Rows.** Each row in the table represents a set of values that UFT submits for all the parameters during a single iteration of the test. When you run your test, UFT runs one iteration of the test for each row of data in the Data pane. For example, a test with ten rows in the Global sheet of the Data pane runs ten times.
- **Sheets.** The Data pane contains two types of sheets—Global and action-specific. For details, see ["Data Tables and Sheets in GUI Tests and Components" on page 342](#).

**Tip:** You can also create data table output values, which retrieve values during the run session and insert them into a column in the Data pane. You can then use these columns as data table parameters in your test. For details, see ["Output Values Overview" on page 325](#).

## Example 1

Suppose your application includes a feature that enables users to search for contact information from a membership database. When the user enters a member's name, the member's contact information is displayed, together with a button labelled **View <MemName>'s Picture**, where **<MemName>** is the name of the member. You can parameterize the name property of the button using a list of values so that during each iteration of the run session, UFT can identify the different picture buttons.

## Example 2

Consider the Mercury Tours sample Web site, which enables you to book flight requests. To book a flight, you supply the flight itinerary and click the **Continue** button. The site returns the available flights for the requested itinerary.

Although you could conduct the test by accessing the Web site and submitting numerous queries, this is a slow, laborious, and inefficient solution. By using data table parameters, you can run the test for multiple queries in succession.

When you parameterize your test, you first create steps that access the Web site and check for the available flights for one requested itinerary.

You then substitute the existing itinerary with a data table parameter and add your own sets of data to the relevant sheet of the Data pane, one for each itinerary.

The screenshot shows the 'Data' pane of the HP Unified Functional Testing interface. The table has columns labeled 'departure', 'arrival', 'fromMonth', 'toMonth', 'E', 'F', 'G', and 'H'. Row 1 contains values: New York, San Francisco, December, December. Row 2 contains: London, Portland, October, October. Row 3 contains: Frankfurt, Paris, November, November. Row 4 contains: Seattle, Sydney, December, December. Row 5 is empty. Rows 6 through 11 are also empty. The bottom navigation bar shows the path: Global > Login > FlightFinder > SelectFlight > BookFlight. The tabs at the bottom are: Output (selected), Errors, Tasks, Active Screen, and Data.

	departure	arrival	fromMonth	toMonth	E	F	G	H
1	New York	San Francisco	December	December				
2	London	Portland	October	October				
3	Frankfurt	Paris	November	November				
4	Seattle	Sydney	December	December				
5								
6								
7								
8								
9								
10								
11								

In this example, UFT submits a separate query for each itinerary when you run the test.

The screenshot shows the UFT interface with a flight search configuration. At the top, there are two dropdown menus: "Departing From" (containing Acapulco, Frankfurt, London, New York, Paris, Portland, San Francisco, Seattle, Sydney, Zurich, with New York selected) and "Arriving In" (containing Acapulco, Frankfurt, London, New York, Paris, Portland, San Francisco, Seattle, Sydney, Zurich, with San Francisco selected). Below these are two large rectangular boxes labeled "Data". The left box contains a table with columns "departure", "arrival", "fromMonth", and "toMonth". The right box contains a table with columns "E", "F", "G", and "H". The bottom of the interface shows a navigation bar with tabs: Global, Login, FlightFinder, SelectFlight, BookFlight, Output, Errors, Tasks, Active Screen, and Data. The "SelectFlight" tab is currently active.

	departure	arrival	fromMonth	toMonth	E	F	G	H
1	New York	San Francisco	December	December				
2	London	Portland	October	October				
3	Frankfurt	Paris	November	November				
4	Seattle	Sydney	December	December				
5								
6								
7								
8								
9								
10								
11								

## When to Choose Global or Action Data Table Parameters

### Relevant for: GUI tests only

When you parameterize a step in a test using the Data pane, you must decide whether you want to make it a **global data table parameter** or a **local data table parameter**.

You use local data table parameters when you want the data to be used only for a single action. You use global data table parameters when you want the data to be available to other actions, and when you want subsequent iterations to use different data for a particular parameter (each time the test repeats or each time the action repeats within the test).

If you have multiple rows in the Global data sheet, the entire test runs multiple times. If you have multiple rows in a local data sheet, the corresponding action runs multiple times before running the next action in the test. If you have multiple rows in both Global and local data sheets, each single test iteration runs all iterations of each action before running the next iteration of the test.

- **Global data table parameters** take data from the Global sheet in the Data pane. The Global sheet contains the data that replaces global parameters in each iteration of the test.

By default, the test runs one iteration for each row in the Global sheet of the Data pane. You can also set the test to run only one iteration. You can also set the test to run iterations on specified rows within the Global sheet of the Data pane.

You can use the parameters defined in the Global data sheet in any action.

**Tip:** By outputting values to the global Data pane sheet from one action and using them as input parameters in another action, you can pass values from one action to another. For details, see "[Output Values Overview](#)" on page 325.

- **Local data table parameters** take data from the action's sheet in the Data pane. The data in the action's sheet replaces the action's data table parameters in each iteration of the action. By default, actions run only one iteration.

You can also set a particular call of the action to run iterations for all rows in the action's sheet or to run iterations on specified rows within the action's sheet. When you set your action properties to run iterations on all rows, UFT inserts the next value from the action's data sheet into the corresponding action parameter during each **action iteration**, while the values of the global parameters stay constant.

After running a parameterized test, you can view the actual values taken from the Data pane in the run results run-time data table.

## Environment Variable Parameters

### Relevant for: GUI tests and scripted GUI components

UFT can insert a value from the Environment variable list, which is a list of variables and corresponding values that can be accessed from your test or component. Throughout the run session, the value of an environment variable remains the same, regardless of the number of iterations, unless you change the value of the variable programmatically.

- When you add an environment variable to a GUI test, it is available to all actions in the test and all steps in the test.
- If you add an environment variable to a component, it is available only to that component, even if the component is part of a BPT test. Environment variables cannot be used to pass data from one component to another in a BPT test.

## Example

You can instruct UFT to read all the values for filling in a Web form from an external file, or you can use one of UFT's built-in environment variables to insert current information about the computer running the test or scripted component.

**Tip:** Environment parameters are especially useful for localization testing, when you want to test an application where the user interface strings change according to the selected language. Environment parameters can be used for testing the same application on different browsers. You can also vary the input values for each language by selecting a different data table file each time you run the steps.

There are several types of environment variables:

### User-Defined Internal Environment Variables (GUI tests and components only)

User-defined internal environment variables are defined within the test or component. These variables are saved with the test or component and are accessible only within the test or component in which they were defined.

You can create or modify internal, user-defined environment variables for your test or component in the:

- Environment pane of the Test or Component Settings dialog box.
- Parameter Options dialog box.

**Tip:** You can also create environment output values, which retrieve values during the test run and output them to internal environment variable parameters for use in your test or component. For details, see "[Output Values Overview](#)" on page 325.

### User-Defined External Environment Variables (GUI tests and components only)

User-defined external environment variables are predefined in the active external environment variables file. You can create as many files as you want and select an appropriate file for each test or component, or change files for each run. External environment variable values are designated as read-only within the test or component.

External environment variable files are comprised of a list of variable-value pairs in .xml format. You select the active external environment variable file for a test or component in the Environment pane of the Test Settings or Component Settings dialog box. Then you can use the variables from the file as parameters.

You can set up your environment variable XML files manually, or you can define the variables as internal environment variables in the Environment pane of the Test Settings dialog box and use the **Export** button to create the .xml file with the correct structure.

For details on creating and using user-defined external environment variable files see, "[How to Use User-Defined External Environment Variables](#)" on page 357.

**Note:**

- You can also store environment variable files in ALM. For details, see "[Environment Variable Files and ALM \(tests only\)](#)" below.
- You can create several external variable files with the same variable names and different values and then run the test several times, using a different file each time. This is especially useful for localization testing.

## Built-in Environment Variables

Variables that represent information about the test or scripted component and the computer on which they are run, such as Test path and Operating system. These variables are accessible from all tests and scripted components, and are designated as read-only.

ALM provides a set of built-in variables that enable you to use current information about the test or scripted component and the UFT computer running your test or component. These can include the test or component name and path, the operating system type and version, and the local host name.

For example, you may want to perform different checks in your test or component based on the operating system being used by the computer that is running the test or component. To do this, you could include the OSVersion built-in environment variable in an If statement.

You can also select built-in environment variables when parameterizing values.

**Note:** UFT also has a set of predefined environment variables that you can use to set the values of the Record and Run Settings dialog options. You should not use the names of these variables for any other purpose. For details, see the sections on how to define record and run settings for Windows-based and Web-based applications in the *HP Unified Functional Testing Add-ins Guide*.

## Environment Variable Files and ALM (tests only)

When working with ALM and environment variable files, you must save the environment variable file in the Test Resources module in your ALM project before you specify the file in the Environment pane of the Test Settings dialog box.

You can add a new or an existing environment variable file to your ALM project. Adding an existing file from the file system to an ALM project creates a copy of the file in ALM. Therefore, changes made to the ALM environment variable file do not affect the file system file and vice versa.

## Built-in Environment Variables

### Relevant for: GUI tests and scripted GUI components

The following built-in environment variables are available:

Name	Description											
<b>ActionIteration (GUI tests only)</b>	The action iteration currently running.											
<b>ActionName</b>	The action or component name.											
<b>ControllerHostName</b>	The name of the controller's computer. This variable is relevant only when running as a GUI Vuser from the LoadRunner controller.											
<b>GroupName</b>	The name of the group in the running scenario. This variable is relevant only when running as a GUI Vuser from the LoadRunner controller.											
<b>LocalHostName</b>	The local host name.											
<b>OS</b>	The operating system.											
<b>OSVersion</b>	The operating system version.											
<b>ProductDir</b>	The folder path where the product is installed.											
<b>ProductName</b>	The product name.											
<b>ProductVer</b>	The product version.											
<b>ResultDir</b>	The path of the folder in which the current run results are located.  <b>Note:</b> You cannot use the <b>ResultDir</b> environment variable when running a test from Business Availability Center, LoadRunner, or the Silent Test Runner in UFT.											
<b>Scenarioid</b>	The identification number of the scenario. This variable is relevant only when running as a GUI Vuser from the LoadRunner controller.											
<b>SystemTempDir</b>	The system temporary folder.											
<b>TestDir</b>	The path of the folder in which the test or scripted component is located.											
<b>TestIteration (GUI tests only)</b>	The test or iteration currently running.											
<b>TestName</b>	The name of the test or component.  <table border="1"> <thead> <tr> <th>When you run a ...</th> <th>In ...</th> <th>TestName contains ...</th> </tr> </thead> <tbody> <tr> <td>UFT test</td> <td>UFT</td> <td>Test name</td> </tr> <tr> <td>Component</td> <td>UFT</td> <td>Component name</td> </tr> </tbody> </table>			When you run a ...	In ...	TestName contains ...	UFT test	UFT	Test name	Component	UFT	Component name
When you run a ...	In ...	TestName contains ...										
UFT test	UFT	Test name										
Component	UFT	Component name										

Name	Description		
	When you run a ...	In ...	TestName contains ...
	BPT test	UFT	BPT test name (use the <b>ActionName</b> variable for the component name)
	BPT test	ALM Test Lab	BPT test name (use the <b>ActionName</b> variable for the component name)
	BPT test	ALM Test Plan	BPT test name (use the <b>ActionName</b> variable for the component name)
<b>UpdatingActiveScreen</b>	Indicates whether the Active Screen images and values are being updated during the update run process.		
<b>UpdatingCheckpoints</b>	Indicates whether checkpoints are being updated during the update run process.		
<b>UpdatingTODescriptions</b>	Indicates whether the set of properties used to identify test objects are being updated during the update run process.		
<b>UserName</b>	The Windows login user name.		
<b>VuserId</b>	The Vuser identification under load. This variable is relevant only when running as a GUI VUser from the LoadRunner controller.		

For more details, see the section on built-in environment variables in "["Environment Variable Parameters" on page 353](#)".

## How to Use User-Defined External Environment Variables

### Relevant for: GUI tests only

This task describes how to create an external variable file and use it in your test.

This task includes the following steps:

- ["Create an external environment variables file" below](#)
- ["\(ALM users only\) Upload the environment resource file to your ALM project" on the next page](#)
- ["Use environment variables in your test" on the next page](#)
- ["Results" on the next page](#)

#### 1. [Create an external environment variables file](#)

You can create an external environment variable file using the Test Settings dialog box or manually. For details, see "["Create an external environment variable file manually" on page 359](#)".

#### 2. [\(ALM users only\) Upload the environment resource file to your ALM project](#)

- a. In the ALM Test Resources module, create a new environment variable resource and then upload the .xml environment variable file you want to use with your test. For details, see the *HP Application Lifecycle Management User Guide*.
- b. In UFT, connect to the ALM project.

### 3. Use environment variables in your test

- a. Open the Environment pane in the Test Settings dialog box (**File > Settings > Environment** node).
- b. Select **User-defined** from the **Variable type** list.
- c. Select the **Load variables and values from external file (reloaded each run session)** check box.
- d. Use the browse button to the right of the **File** edit box, or enter the full path of the external environment variables file you want to use with your test. If your test is stored in ALM, you must select a file that is stored with your ALM project.

The variables defined in the selected file are displayed in the list of user-defined environment variables.

### 4. Results

You can now select the variables in the active file as external user-defined environment parameters in your test.

## How to Create an External Environment Variables File

**Relevant for:** GUI tests only

**Note:** This task is part of a higher-level task. For details, see "[How to Use User-Defined External Environment Variables](#)" on the previous page.

You create an external environment variables file in one of the following ways:

- "[Create an external environment variable file using the Test Settings dialog box](#)" below
- "[Create an external environment variable file manually](#)" on the next page

### Create an external environment variable file using the Test Settings dialog box

1. Define the variables in the Environment pane of the Test Settings dialog box (**File > Settings > Environment** node).
2. Click **Export** to export the user-defined environment variables to an .xml file.

### Create an external environment variable file manually

1. Create an .xml file using the editor of your choice.

You can use the UFT environment variable file schema in:<Unified Functional Testing installation folder>\help\QTEvironment.xsd or follow the formatting instructions below.

2. Enter <Environment> on the first line.
3. Enter each variable name-value pair in between <Variable> elements using the following format:

```
<Variable>
 <Name>This is the first variable's name</Name>
 <Value>This is the first variable's value</Value>
 <Description> This text is optional and can be used to add comments.
 It is shown only in the XML and not in UFT</Description>
</Variable>
```

4. Enter </Environment> on the last line.

#### Example

```
<Environment>
 <Variable>
 <Name>Address1</Name>
 <Value>25 Yellow Road</Value>
 </Variable>
 <Variable>
 <Name>Address2</Name>
 <Value>Greenville</Value>
 </Variable>
 <Variable>
 <Name>Name</Name>
 <Value>John Brown</Value>
 </Variable>
 <Variable>
 <Name>Telephone</Name>
 <Value>1-123-12345678</Value>
 </Variable>
</Environment>
```

5. Save the file in a location that is accessible from the UFT computer. The file must be in .xml format with an .xml file extension.

## Automatically Parameterizing Steps

### Relevant for: GUI tests only

You can instruct UFT to automatically parameterize the steps in your test actions at the end of a recording session.

This enables you to create actions that can be used for a variety of different purposes or scenarios by referencing different sets of data.

To activate this option, select the **Automatically parameterize steps** option in the **General** node of the GUI Testing tab of the Options dialog box (**Tools > Options > GUI Testing tab > General** node). You can set the option to use **Global Data Table Parameters** or **Test Parameters**.

When you stop a recording session while this option is selected, UFT replaces the constant values in the test object operation arguments of your steps with either Data pane parameters or action parameters, based on your selection of global Data pane parameters or test parameters in the Options dialog box.

UFT performs this automatic parameterization for all relevant steps in any action in your test, in which you recorded one or more steps during that recording session.

- When you select to use **Global Data Table Parameters**, the generated parameters are added to the Global sheet of the Data pane.

If you work with ALM, you can map these parameters to the column names of a data resource and then use different configurations in your test sets. For details on ALM configurations and mapping data table parameters to ALM data resources, see "[Data Awareness in ALM](#)" on page 928.

- When you select to use **Test Parameters**, UFT parameterizes the step with a newly created action parameter. It also creates a corresponding test parameter.

For details on how UFT creates these parameters, which types of operation arguments are included and excluded from the parameterization, and other important criteria to consider, see "[Guidelines and Considerations for Automatically Parameterizing Steps](#)" below.

To learn more, see: "[Guidelines and Considerations for Automatically Parameterizing Steps](#)" below.

## Guidelines and Considerations for Automatically Parameterizing Steps

### **Relevant for: GUI tests only**

Before you begin a recording session with the **Automatically parameterize steps** option activated, make sure you are aware of the following guidelines and considerations:

#### **General Guidelines**

- The automatic parameterization process runs on all relevant steps in all local actions in which you recorded one or more steps during the recording session. It also parameterizes the action that is displayed when you stop the recording session, even if you did not record any steps in that action.
  - All relevant steps in any relevant action are parameterized, whether or not those steps were added in the current recording session. Therefore, you can perform automatic parameterization on an existing action by starting and stopping a recording session without adding any steps.
- Automatic parameterization is not performed on external actions that are called from your test, nor for actions called using the **LoadAndRunAction** statement.

- In general, simple, constant (string, number, boolean) test object and utility object operation arguments are parameterized. Therefore, if the following are contained in a method argument, they are not parameterized:
  - arguments that are already parameterized
  - variables
  - enumeration constants, such as **micLeftbtn**
  - expressions, such as: `x = 3`
  - Assignments of values, such as: `Window("Notepad").WinMenu("Menu").ExpandMenu = True`
  - mathematical or other concatenation operations, such as `"Hello World" & micCtrl & "S"`
  - VBScript statements, such as `msgbox "hello"`
  - VBScript language statements such as `For`, `If`, `Do`, `While`
  - steps inside called functions from function libraries or in functions or sub-routines defined directly within the action
- In addition to the above general rule, operation arguments in the following scenarios are also not parameterized:
  - **SAPGuiTable.Input**, **SAPGuiGrid.Input**, or **SAPGuiAPOGrid.Input** steps (inserted using the **Auto-parameterize table and grid controls** option). For details, see the auto-parameterize table topic in the SAP section of the *HP Unified Functional Testing Add-ins Guide*.
  - Native methods and properties accessed by the **Object** property.
  - Steps for test objects that are stored in variables. For example, the "text" constant is not parameterized in:

```
Set MyEditBox = Browser("x").Page("x").WebEdit("myedit")
MyEditBox.Set "text"
```

- Steps containing programmatic descriptions, such as:

```
Browser("x").Page("x").WebEdit(MyDescription).Set "text"
```

or

```
Browser("x").Page("x").WebEdit("prop:=value", "prop2:=value2").Set "text".)
```

- User-defined functions that are registered to test objects using a **RegisterUserFunc** statement.

However, built-in UFT test object operations that were overridden using a **RegisterUserFunc** statement are parameterized.

- Operation arguments of type variant, when the value is a number.

For example, in the following statement, the variant argument of the Select method is not parameterized, because it is a number.

```
WpfWindow("MyWindow").WpfComboBox("cb").Select 1
```

However, in the following statement, the variant argument of the Select method is parameterized, because it is a string.

```
WpfWindow("MyWindow").WpfComboBox("cb").Select "item1"
```

- When working with Siebel applications, only operation arguments of **Sbl\*** test objects, which represent standard interactivity (SI) objects are parameterized. The operation arguments of **Sieb\*** test objects, which represent Siebel High Interactivity (HI) API test objects, are not parameterized. For details on these types of objects, see the **Siebel** section of the *HP Unified Functional Testing Add-ins Guide*.
- The name of the parameter that UFT creates for each method argument is in the format **TestObjectName\_ArgumentName**.
  - If a parameter with this name already exists for the relevant parameter type, UFT appends an underscore and a number to the end of the new parameter to create a unique name.
  - If the test object name contains characters that are not valid for parameter names or if it contains multi-byte characters, then the test object class is used instead of the test object name.
- The automatic parameterization option is relevant only for tests. UFT does not automatically parameterize steps after you record steps in a component.
- If you select the **Automatically generate With statements after recording** option in addition to the **Automatically parameterize steps** option, then when you stop a recording session, UFT converts the steps to **With** statements and then parameterizes the operation arguments within them.

## Guidelines for Automatic Parameterization of Data Table Parameters

- When you use **Global Data Table Parameters** for the automatic parameterization option, the arguments are parameterized using Global Data Table parameters. The constant value that the parameter replaces is stored in the created column in the Global data sheet.  
If the Global data sheet already contains more than one row, the value is entered in all rows of the created Data Table column.
- Data pane sheets can contain 256 columns. At the end of a recording session, UFT parameterizes the relevant operation arguments until it reaches the 256th column. If UFT reaches the maximum

number of data table parameters, it stops parameterizing at that point and a message informs you that not all relevant steps were parameterized.

- If the test uses an external data table, and the data table file is read-only, locked by another user, or checked in to ALM version control, the automatic parameterization is not performed and a message is displayed when you finish your recording session.

## Guidelines for Automatic Parameterization of Test Parameters

- When you use **Test Parameters** for the automatic parameterization option:
  - Each relevant method argument is converted to a new action parameter.
  - A corresponding test parameter is also created.
  - The constant value that the parameter replaces is stored as the default value for the action parameter (in the Action Properties dialog box) as well as for the test parameter (in the Parameters pane of the Test Settings dialog box).

For **top-level actions**, UFT also promotes (parameterizes) the action parameter value (in the Action Call Properties Dialog Box) to use the corresponding test parameter for you.

For **nested actions**, UFT creates an action parameter and a corresponding test parameter. However, the action parameter value (in the Action Call Properties Dialog Box) is not promoted (parameterized) to use a parent action parameter. It is set to use the constant value that it replaced.

When working with nested actions, if you want to use the created test parameter in order to provide the actual values from an external application that calls your test, such as from SAP eCATT, you must manually create corresponding action parameters for all actions that call your action, and then pass (promote) the action parameter values up to each parent action and then to the test parameter level. For details on passing values between action and test parameters, see ["How to Use Action Parameters" on page 371](#).

## Data Driver

### Relevant for: GUI tests only

The Data Driver enables you to quickly parameterize several (or all) property values for test objects, checkpoints, and/or method arguments containing the same constant value within a given action.

You can choose to replace all occurrences of a selected constant value with a parameter, in the same way that you can use a **Find and Replace All** operation instead of a step-by-step **Find and Replace** process.

UFT can also show you each occurrence of the constant so that you can decide whether or not to parameterize the value.

**Note:**

- When finding multiple occurrences of a selected value, UFT conducts a search that is case sensitive and searches only for exact matches. (It does not find values that include the selected value as part of a longer string.)
- You cannot use the Data Driver to parameterize the values of arguments for user-defined methods or VBScript functions.

## Test and Action Parameters

### Relevant for: GUI tests only

You can parameterize a value in a step using a test or action parameter (both input and output parameters). You can create parameters for:

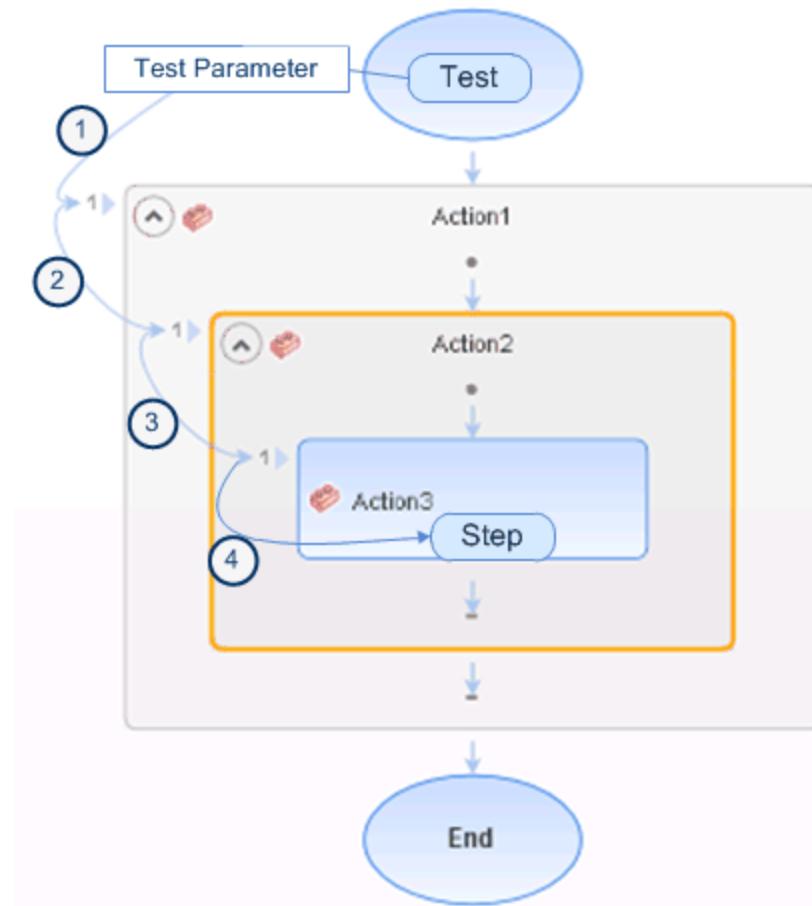
<b>The test</b>	<p>Test parameters enable you to share parameter and its value with each action (and subsequently each step in the action) in your test. Once you have created a test parameter, any action parameter can take its value from the test parameter.</p> <p>Test parameters are created in the <b>Parameters</b> tab of the Properties pane.</p>
<b>The action</b>	<p>Action parameters enable you specify parameters that are available for all steps contained in the action.</p> <p>Action parameters are stored with the action and are maintained to all calls to the action. You can provide the action parameter value from any of the following:</p> <ul style="list-style-type: none"><li>• A test parameter</li><li>• The parameters of a parent action (if the action is nested)</li><li>• The output of a previous action call (for sibling actions)</li></ul> <p>You create parameters in the <b>Parameters</b> tab of the Properties pane or the <b>Parameters</b> tab of the Action Properties dialog box. You set the actual value of the action parameter in the <b>Parameter Values</b> tab of the Action Call Properties dialog box.</p>

### Considerations for Using Action Parameters

- Input action parameter values can be used only within the steps of the current action. You can use an action input value from another action (or from the test) only if you pass the value from action to action down the test hierarchy to the action in which you want to use it.

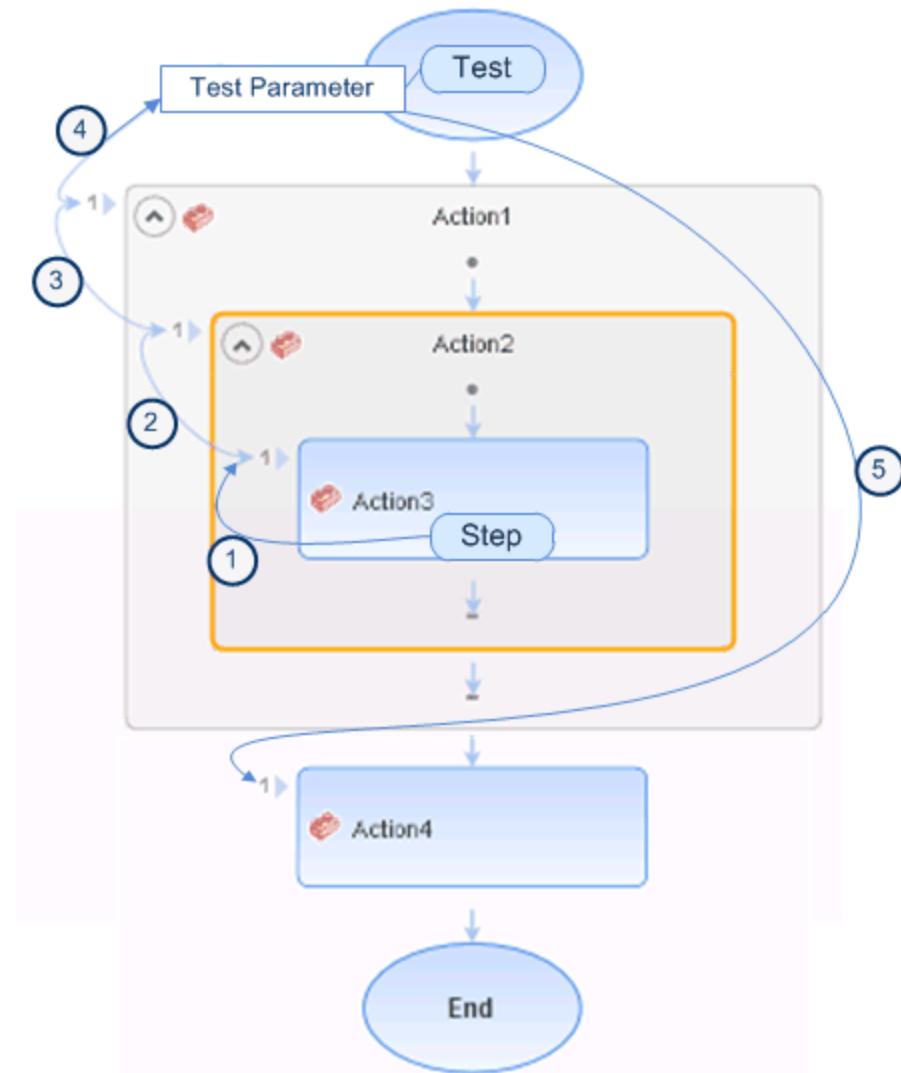
- In subsequent steps of a calling action, you can use any type of action output value as a variable, if the value was retrieved from the called action. For example, if ActionA calls ActionB and specifies MyBVar as the variable in which to store ActionB's output parameter, then steps in ActionA after the call to ActionB can use the MyBVar as a value (just as you would use any other variable). For example:

Test -> Action1 -> Action2 -> Action3 -> (Action3) Step 1



- Output action parameter values can be retrieved from a previous action at the same hierarchical level, from a parent action, or from the current action. You can use an action output value from one action within the step of another action if:
  - You pass the value from action to action up the test hierarchy to the action in which you want to use it. For example:

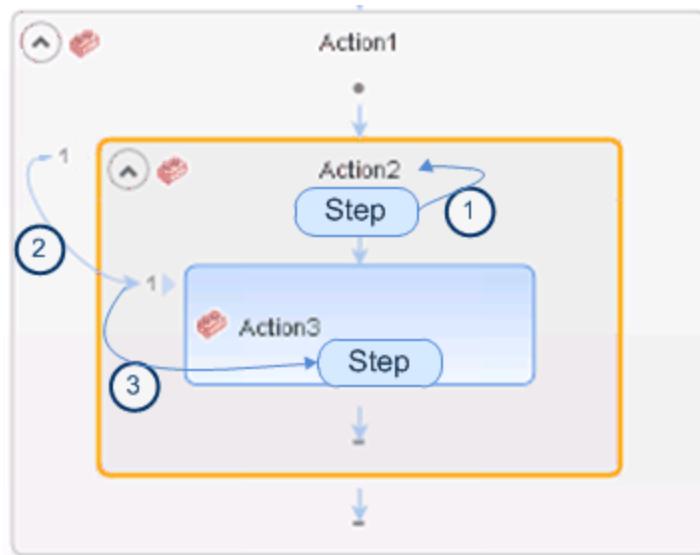
(Action3) Step 1-> Action3 -> Action2 -> Action1 -> Test -> Action4



In this example, any step in Action 1, Action 2, or Action 3 can potentially use the output value from (Action3) Step 1, even though the example shows that the output value is used by steps in Action4.

- You pass the value from a previous action to the sibling action in which you want to use it. For example:

(Action2) Step 1 -> Action2 -> Action3 -> (Action3) Step 1



In this example, any step in Action 2 or Action 3 can potentially use the output value from (Action2) Step 1, even though the example shows that the output value is used by (Action3) Step 1.

For task details on using test and action parameters, see ["How to Use Action Parameters" on page 371](#).

## Passing Test and Action Parameter Values

When you create test and action parameter values, you can use the parameters and values in numerous places throughout the test. For example, the value of an action parameter can be taken from the test parameter value. Likewise, a test step object value can be taken from an action parameter.

However, there will be times where you want to use the value of a test parameter as the parameter value for a step, or use a parameter of a parent action in a nested (child action). In these cases, you need to pass the parameter from the test to the action, or through the action hierarchy, to enable the step to access the parameter.

### Example

Suppose that Action3 is a nested action of Action1 (a top-level action), and you want to parameterize a value in Action3 using a value that is passed into your test from the external application that runs (calls) the test.

You can pass the value from the test level to Action1, then to Action3, and then parameterize the required value using this action input parameter value (that was passed through from the external application).

Alternatively, you can pass an output action parameter value from an action step to a later sibling action at the same hierarchical level. For example, suppose that Action2, Action3, and Action4 are sibling actions at the same hierarchical level, and that these are all nested actions of Action1.

You can parameterize a call to Action4 based on an output value retrieved from Action2 or Action3. You can then use these parameters in your action step.

For details, see ["Considerations for Using Action Parameters" on page 365](#).

For details on how to create and use test and action parameters in your test, see ["How to Use Action Parameters" on page 371](#).

## Sharing Action Information

### **Relevant for: GUI tests only**

There are several ways to share or pass values from one action to other actions:

### **Sharing Values Using the Global Data Table**

You can share a value that is generated in one action with other actions in your test, by storing the value in the data table. You can choose to place the value in either the **Global** sheet or the **Action** sheet. If you put the value in the Global sheet, other actions can then use the value in the Data pane as an input parameter. You can store a value in the Data pane by outputting the value to the global data table, or by using **Data Table**, **Sheet** and **Parameter** objects and methods in the Editor to add or modify a value.

**Note:** If you create data table parameters or output value steps in your action and select to use the **Current action sheet (local)** option, be sure that the relevant run settings for your action are set in the Run tab of the Action Call Properties Dialog Box.

### **Example**

Suppose you are testing a flight reservation application. When a user logs into the application, the user's full name is displayed on the top of the page. Later, when the user purchases the tickets, the user must enter the name that is listed on his or her credit card.

Suppose your test contains three actions—Login, SelectFlight, and PurchaseTickets and the test is set to run multiple iterations with a different login name for each iteration. In the Login action, you can create a text output value to store the displayed name of the user. In the PurchaseTickets action, you can parameterize the value that is set in the Credit Card Owner edit box using the Data pane column containing the user's full name.

### **Sharing Values Using Environment Variables**

If you do not need to run multiple iterations of your test, or if you want the value you are sharing to stay

constant for all iterations, you can use an internal, user-defined environment variable, which can be accessed by all local actions in your test.

For example, suppose you want to test that your flight reservation application correctly checks the credit card expiration date that the user enters. The application should request a different credit card if the expiration date that was entered is earlier than the scheduled flight departure date. In the `SelectFlight` action, you can store the value entered in the departure date edit box in an environment variable. In the `PurchaseTickets` action, you can compare the value of the expiration date edit box with the value stored in your environment variable.

For more information on environment variables, see ["Parameterizing Object Values" on page 336](#). For information on the **Environment** object, see the **Utility Objects** section of the *HP UFT Object Model Reference for GUI Testing*.

## Sharing Values Using the Dictionary Object

As an alternative to using environment variables to share values between actions, you can use the Dictionary object. The Dictionary object enables you to assign values to variables that are accessible from all actions called from the test in which the Dictionary object is created, including both local and external actions.

To use the Dictionary object, you must first add a reserved object to the registry (in **HKEY\_CURRENT\_USER\Software\Mercury Interactive\QuickTest Professional\MicTest\ReservedObjects\**) with `ProgID = "Scripting.Dictionary"`.

```
HKEY_CURRENT_USER\Software\Mercury Interactive\QuickTest
Professional\MicTest\ReservedObjects\GlobalDictionary
```

After you add the reserved Dictionary object to the registry and restart UFT, you can add and remove values to and from the Dictionary in one action, and retrieve the values in another action called from the same test.

For more information on the Dictionary object, see the VBScript Reference documentation (**Help > HP Unified Functional Testing Help > VBScript Reference > Script Runtime**).

### Example

Suppose you want to access the departure date set in the `SelectFlight` action from the `PurchaseTickets` action. You can add the value of the `DepartDate` WebEdit object to the dictionary in the `SelectFlight` action as follows:

```
GlobalDictionary.RemoveAll
```

Then you can retrieve the date from the `PurchaseTickets` action as follows:

```
Dim CompareDate
CompareDate=GlobalDictionary("DateCheck")
```

## Sharing Values Using Output Values

You can create a output value step in your action to pass the output of a step. Then in the Output Options, you can pass the output value to an action output parameter, which is then accessible to all subsequent or sibling actions.

## Using Action Parameters in Steps in the Editor

Instead of selecting input (or output) parameters from the appropriate dialog boxes while parameterizing steps or inserting output value steps, you can enter input and output parameters as values in the Editor using the **Parameter** utility object in the format:

Parameter("ParameterName") for the current action

or

Parameter("ActionName", "ParameterName") to use the output parameter from a previous action as an input parameter in the current action.

### Example:

Suppose you have test steps that enter information in a form to display a list of purchase orders in a table, and then return the total value of the orders displayed in the table.

You can define input parameters, called **SoldToCode** and **MaterialCode**, for the codes entered in the **Sold to** and **Materials** edit boxes of the form so that the Orders table that is opened is controlled by the input parameter values passed when the test is called.

You can define an output parameter, for example **TotalValue**, to store the returned value. The output value (**TotalValue**) can then be returned to the application that called the test.

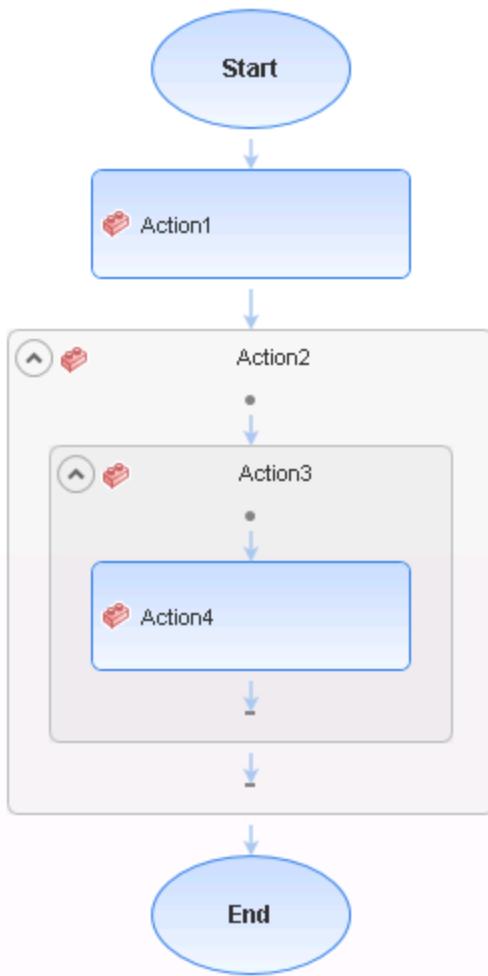
The example described above might look something like this (parameters are in bold font):

```
Browser("Mercury").Page("List Of Sales").WebEdit("Sold to").Set Parameter
("SoldToCode")
Browser("Mercury").Page("List Of Sales").WebEdit("Materials").Set Parameter
("MaterialCode")
Browser("Mercury").Page("List Of Sales").WebButton("Enter").Click
NumTableRows = Browser("Mercury").Page("List Of Sales").WebTable
("Orders").RowCount
Parameter("TotalValue") = Browser("Mercury").Page("List Of Sales").WebTable
("Orders").GetCellData(NumTableRows, "Total")
```

# How to Use Action Parameters

## Relevant for: GUI tests only

Suppose you want to take a value from the external application that runs (calls) your test and use it in an action within your test. If your test has an **Action4** that is nested within **Action3**, and **Action3** is further nested within **Action2**, you would need to pass the input test parameter from the external application, through **Action2** and **Action3**, to the required step in **Action4**.



You would do this as follows:

1. Define the input test parameter (**File > Settings > Parameters** node) with the value that you want to use later in the test.
2. Define an input action parameter for Action2 (right-click an action and select **Action Properties > Parameters** tab) with the same value type as the input test parameter.
3. Parameterize the input action parameter value (right-click an action and select **Action Call Properties > Parameter Values** tab) using the input test parameter value you specified above.

4. Define an input action parameter for Action3 (right-click an action and select **Action Properties > Parameters** tab) with the same value type as the input test parameter.
5. Parameterize the input action parameter value.
  - Right-click an action and select **Action Call Properties > Parameter Values** tab and select the input action parameter value you specified for Action2.
  - Use the **Parameter** utility object to specify the action parameter as the *Parameters* argument for the **RunAction** statement in the Editor.
6. Define an input action parameter for Action4 (right-click an action and select **Action Properties > Parameters** tab) with the same value type as the input test parameter.
7. Parameterize the input action parameter value.
  - Right-click an action in the canvas or Keyword View and select **Action Call Properties > Parameter Values** tab and select the input action parameter value you specified for Action3.
  - Use the **Parameter** utility object to specify the action parameter as the *Parameters* argument for the **RunAction** statement in the Editor.
8. Parameterize the value in the required step in Action4.
  - Click the parameterization icon  and specify the parameter in the Value Configuration Options Dialog Box using the input action parameter you specified for Action 4.
  - Use the **Parameter** utility object in the Editor to specify the value to use for the step. For more information, see "[Test and Action Parameters](#)" on page 364.

**Note:** You can also modify many of the action properties from the Properties pane.

## Regular Expressions Overview

### Relevant for: GUI tests and components and API testing

A **regular expression** is a string that specifies a complex search phrase. By using special characters, such as a period (.), asterisk (\*), caret (^), and brackets ([ ]), you can define the conditions of a search.

Regular expressions are used to identify objects and text strings with varying values. You can use regular expressions to instruct UFT to find a value that matches a particular pattern or condition instead of a specific hard-coded value.

You can use regular expressions only for values of type **string**.

## Use case examples:

- Suppose the name of a window's title bar changes according to a file name. You can use a regular expression in a test object description to identify a window whose title bar has the specified product name, followed by a hyphen, and then any other text. When the relevant step runs, UFT compares the regular expression that you provide with the corresponding value in your application.
- Suppose the text property of an object is a date value, but the displayed date changes according to the current date. You can define a regular expression for the date, so that UFT can identify the object that contains text with the expected date format, rather than the exact date value.

Whenever a UFT feature supports regular expressions, the relevant dialog box includes a **Regular Expression** check box. Selecting this check box instructs UFT to treat the provided value as a regular expression. Some dialog boxes that contain a **Regular Expression** check box, also contain a right arrow adjacent to the text box for the value. Clicking this arrow enables you to select regular expression characters from a drop-down list, and to test your regular expression to make sure it suits your needs.

For example, there are a number of scenarios in which you could use regular expressions for a GUI test or component:

- When defining the property values of an object in dialog boxes or in programmatic descriptions used within a function library
- When defining expected values for checkpoints in tests
- When defining pop-up window conditions in a recovery scenario

There are also places you could use a regular expressions in an API test:

- When defining XML checkpoint property values where the property is a string
- HTTP Request test step checkpoint values, if the HTTP response is text

For details on defining regular expressions, including regular expression syntax, see ["Regular Expression Characters and Usage Options" on page 375](#).

To learn more, see:

- [Regular Expressions for GUI Test and Component Object Property Values](#) ..... 374
- [Regular Expressions in GUI Test and Component Checkpoints](#) ..... 374
- [Regular Expression Characters and Usage Options](#) ..... 375
- [Regular Expressions in the Find and Replace Dialog Boxes](#) ..... 380

## Regular Expressions for GUI Test and Component Object Property Values

### **Relevant for: GUI tests and components**

If you expect the value of an object property in your application to change in a predictable way during each run session, you can use regular expressions when defining identification property values, for example, in the Object Repository window, or in [programmatic descriptions](#).

For example, your Web site may include a form in which the user inputs data and clicks the **Send** button to submit the form. When a required field is not completed, the form is displayed again for the user to complete. When resubmitting the form, the user clicks the **Resend** button. You can define the value of the button's **name** property as a regular expression, so that this variation in the button name is ignored when identifying the button in your application.

**Tip:** For tests and scripted components, UFT provides a tool to test your regular expressions to help you make sure that they suit your needs. UFT provides a list of commonly used regular expression characters that you can select when creating a regular expression.

## Regular Expressions in GUI Test and Component Checkpoints

### **Relevant for: GUI tests and components**

When creating a standard checkpoint to verify the property values of an object, you can set the expected value of an object's property as a regular expression so that an object with a varying value can be verified.

For example, suppose you want to check that every window and dialog box in your application contains the name of your application followed by a hyphen (-) and a descriptive title. You can add a checkpoint for each dialog box object in your test to check that the first part of the title contains the name of your application followed by a hyphen.

When working with the different types of checkpoints available for GUI tests, you can use regular expressions in a number of ways:

- When creating a text checkpoint to check that a varying text string is displayed on your application, you can define the text string as a regular expression.

For example, when booking a flight in the Mercury Tours sample Web site, the total cost charged to a credit card number should not be less than \$300. You define the amount as a regular expression, so that variations in the text string will be ignored as long as the value is not less than \$300.

- For table checkpoints you can set cell values as regular expressions
- For XML checkpoints you can set attribute or element values as regular expressions.

You can apply the same principles to any checkpoint type whose dialog box contains a **Configure Value** area similar to that described in [Configure Value Area](#).

**Tip:** For tests and scripted components, UFT provides a tool to test your regular expressions to help you make sure that they suit your needs. UFT provides a list of commonly used regular expression characters that you can select when creating a regular expression.

## Regular Expression Characters and Usage Options

### Relevant for: GUI tests and components and API testing

This section describes some of the more common options that can be used to create regular expressions:

### Using the Backslash Character (\)

A backslash (\) can serve two purposes. It can be used in conjunction with a special character to indicate that the next character be treated as a literal character. For example, \. would be treated as period (.) instead of a wildcard. Alternatively, if the backslash (\) is used in conjunction with some characters that would otherwise be treated as literal characters, such as the letters n, t, w, or d, the combination indicates a special character. For example, \n stands for the newline character.

If the backslash character is not used for either of these purposes, it is ignored.

For example:

- w matches the character w
- \w is a special character that matches any word character including underscore
- \\" matches the literal character \
- \( matches the literal character (
- one\two matches the string onetwo

For example, if you were looking for a Web site called:

newtours.demoaut.com

the period would be mistaken as an indication of a regular expression. To indicate that the period is not part of a regular expression, you would enter it as follows:

newtours\.demoaut\.com

### Matching Any Single Character (.)

A period (.) instructs UFT to search for any single character (except for \n). For example:

welcome.

matches welcomes, welcomed, or welcome followed by a space or any other single character. A series of periods indicates the same number of unspecified characters.

To match any single character including \n, enter:

(.|\\n)

For more details on the **( )** regular expression characters, see "[Grouping Regular Expressions \( \)](#)" on the next page. For more details on the **|** regular expression character, see "[Matching One of Several Regular Expressions \( | \)](#)" on the next page.

### Matching Any Single Character in a List ( [xy] )

Square brackets instruct UFT to search for any single character within a list of characters. For example, to search for the date 1967, 1968, or 1969, enter:

196[789]

### Matching Any Single Character Not in a List ( [^xy] )

When a caret (^) is the first character inside square brackets, it instructs UFT to match any character in the list except for the ones specified in the string. For example:

[^ab]

matches any character except a or b

**Note:** The caret has this special meaning only when it is displayed first within the brackets.

### Matching Any Single Character within a Range ( [x-y] )

To match a single character within a range, you can use square brackets ([ ]) with the hyphen (-) character. For instance, to match any year in the 1960s, enter:

196[0-9]

A hyphen does not signify a range if it is displayed as the first or last character within brackets, or after a caret (^).

For example, [-a-z] matches a hyphen or any lowercase letter.

**Note:** Within brackets, the characters ".", "\*", "[" and "\" are literal. For example, [.\*] matches . or \*. If the right bracket is the first character in the range, it is also literal.

### Matching Zero or More Specific Characters ( \* )

An asterisk (\*) instructs UFT to match zero or more occurrences of the preceding character. For example:

ca\*r

matches car, caaaaaar, and cr

## Matching One or More Specific Characters (+)

A plus sign (+) instructs UFT to match one or more occurrences of the preceding character. For example:

ca+r

matches car and caaaaaar, but not cr

## Matching Zero or One Specific Character (?)

A question mark (?) instructs UFT to match zero or one occurrences of the preceding character. For example:

ca?r

matches car and cr, but nothing else

## Grouping Regular Expressions (())

Parentheses (()) instruct UFT to treat the contained sequence as a unit, just as in mathematics and programming languages.

Using groups is especially useful for delimiting the arguments to an alternation operator (|) or a repetition operator: (\*, +, ?, { })

## Matching One of Several Regular Expressions (|)

A vertical line (|) instructs UFT to match one of a choice of expressions. For example:

foo|bar

causes UFT to match either foo or bar

fo(o|b)ar

causes UFT to match either foobar or fobar

## Matching the Beginning of a Line (^)

A caret (^) instructs UFT to match the expression only at the start of a line, or after a newline character.

For example:

book

matches book **within** the lines—book, my book, and book list, while

^book

matches book **only** in the lines—book and book list

## Matching the End of a Line ( \$ )

A dollar sign (\$) instructs UFT to match the expression only at the end of a line.

For example:

book

matches book within the lines—my book, and book list, while a string that is followed by (\n), (\r), or (\$), matches only lines ending in that string. For example:

book\$

matches book only in the line—my book

## Matching a Newline or Carriage Return Character ( \n ) or ( \r )

\n or \r instruct UFT to match the expression only when followed by a newline or carriage return character.

- \n instructs UFT to match any newline characters.
- \r instructs UFT to match any carriage return characters.

For example:

book

matches book within the lines—my book, and book list, while a string that is followed by (\n) or (\r) matches only lines that are followed by a newline or carriage return character. For example:

book\r

matches book only when book is followed by a carriage return

## Matching Any AlphaNumeric Character Including the Underscore ( \w )

\w instructs UFT to match any alphanumeric character and the underscore (A-Z, a-z, 0-9, \_).

For example:

\w\* causes UFT to match zero or more occurrences of the alphanumeric characters—A-Z, a-z, 0-9, and the underscore (\_). It matches Ab, r9Cj, or 12\_uYLgeu\_435.

For example:

\w{3} causes UFT to match 3 occurrences of the alphanumeric characters A-Z, a-z, 0-9, and the underscore (\_). It matches Ab4, r9\_, or z\_M.

## Matching Any Non-AlphaNumeric Character ( \W )

\W instructs UFT to match any character other than alphanumeric characters and underscores.

For example:

\W

matches &, \*, ^, %, \$, and #

### Matching a Decimal Digit ( \d )

\d instructs UFT to match any decimal digit.

For example:

\d

matches 1, 2, 4, and 5

### Matching an Integer ( \D )

\D instructs UFT to match any whole integer.

For example:

\D

matches 145643, 20, 3426767, 4, and 5

## Combining Regular Expression Operators

You can combine regular expression operators in a single expression to achieve the exact search criteria you need.

For example, you can combine the '.' and '\*' characters to find zero or more occurrences of any character (except \n).

For example,

start.\*

matches start, started, starting, starter

You can use a combination of brackets and an asterisk to limit the search to a combination of non-numeric characters. For example:

[a-zA-Z]\*

To match any number between 0 and 1200, you need to match numbers with 1 digit, 2 digits, 3 digits, or 4 digits between 1000-1200.

The regular expression below matches any number between 0 and 1200.

([0-9]?[0-9]?[0-9]|1[01][0-9][0-9]|1200)

**Note:** For a complete list and explanation of supported regular expressions characters, see the Regular Expressions section in the Microsoft VBScript documentation (select **Help > HP Unified Functional Testing Help** to open the UFT Help. Then select **VBScript Reference > VBScript > VBScript User's Guide > Introduction to Regular Expressions**).

## Regular Expressions in the Find and Replace Dialog Boxes

### Relevant for: GUI actions, scripted GUI components, function libraries, and user code files

You can use regular expressions in the **Find text** fields to enhance your search in both the Find Dialog Box and Replace Dialog Box.

You can insert a regular expression into your search string by selecting one from a predefined list, or by entering one manually. For a general understanding of regular expressions in UFT, see "[Regular Expressions Overview](#)" on page 372.

For more details about find and replace functionality, see "[Searching and Replacing in the Editor](#) " on page 633.

**Note:** Not all the regular expressions listed for other parts of UFT are supported for the search strings in the Find and Replace dialog boxes.

The following table lists the regular expressions supported for search strings, as well as descriptions of each regular expression. For details, see "[Regular Expression Characters and Usage Options](#)" on page 375.

Character	Description	Reference
.	(period) Matches any single character	<a href="#">"Matching Any Single Character ( . )"</a> on page 375
*	(asterisk) Matches zero or more specific characters	<a href="#">"Matching Zero or More Specific Characters ( * )"</a> on page 376
+	(plus sign) Matches one or more specific characters	<a href="#">"Matching One or More Specific Characters ( + )"</a> on page 377
?	(question mark) Matches zero or one specific character	<a href="#">"Matching Zero or One Specific Character ( ? )"</a> on page 377
^	(caret) Matches the beginning of a line	<a href="#">"Matching the Beginning of a Line ( ^ )"</a> on page 377
\$	(dollar sign) Matches the end of a line	<a href="#">"Matching the End of a Line ( \$ )"</a> on page 378
\n	Matches a line break	<a href="#">"Matching a Newline or Carriage Return Character ( \n ) or ( \r )"</a> on page 378
\r	Matches a carriage return	<a href="#">"Matching a Newline or Carriage Return Character ( \n ) or ( \r )"</a> on page 378
[]	Matches any single character in a list	<a href="#">"Matching Any Single Character in a List ( [xy] )"</a> on page 376
[^]	Matches any single character not in a list	<a href="#">"Matching Any Single Character Not in a List ( [^xy] )"</a> on page 376

Character	Description	Reference
\w	Matches any alphanumeric character including the underscore	<a href="#">"Matching Any AlphaNumeric Character Including the Underscore ( \w )" on page 378</a>
\W	Matches any non-alphanumeric character	<a href="#">"Matching Any Non-AlphaNumeric Character ( \W )" on page 378</a>
\d	Matches a decimal digit	<a href="#">"Matching a Decimal Digit ( \d )" on page 379</a>
\D	Matches an integer	<a href="#">"Matching an Integer ( \D )" on page 379</a>
	Matches one of several regular expressions	<a href="#">"Matching One of Several Regular Expressions (   )" on page 377</a>
\	Matches a special character treated as a literal character, or a literal character treated as a special character	<a href="#">"Using the Backslash Character ( \ )" on page 375</a>

# Chapter 39: Value Configuration and Parameterization

**Relevant for: GUI tests and components**

This chapter includes:

- [Value Configuration Overview](#) ..... 383
- [Working with Local and Component Parameters](#) ..... 383
- [How to Configure Constant and Parameter Values](#) ..... 384
- [How to Parameterize Input Values \(Keyword Components\)](#) ..... 385

# Value Configuration Overview

## Relevant for: GUI tests and scripted GUI components

UFT enables you to configure the values for properties and other items by defining a value as a constant or a parameter. You can also use regular expressions for some values to increase the flexibility and adaptability of your tests. For details on regular expressions, see "[Regular Expressions Overview](#)" on [page 372](#).

Some dialog boxes, such as the Checkpoint Properties dialog boxes, include a **Configure value** area, in which you can define the value for a selected item as a constant or a parameter. In other contexts, such as the Keyword View, Step Generator, and Object Repository window, you can select a value directly and parameterize it or define it as a constant.

- **Constant.** A manually defined value that remains unchanged for the duration of the test. In certain contexts, you can define a constant value using a regular expression.
- **Parameter.** A value that is defined or generated externally and is retrieved during a run session. For example, a parameter value may be defined in an external file or generated by UFT.

When you define a value as a parameter, you can also specify other settings according to the parameter type. For details on using parameters in your tests, see "[Parameterizing Object Values](#)" on [page 336](#).

# Working with Local and Component Parameters

## Relevant for: Keyword GUI components

You can define input parameters that pass values into your keyword component, and output parameters that pass values from your component to external sources or from one step to another step. You can then use these parameters to parameterize input and output values in steps.

You can define two types of parameters—**local parameters** and **component parameters**:

Parameter Type	Description
<b>Local parameter</b>	Variable values defined within a component for use within the same component. Local input parameter values can be received and used by a later parameterized step in the same component. You define local input parameters in the Keyword View using the Value Configuration Options Dialog Box and the Parameters Tab of the Properties Pane. Local output parameters can be returned by an operation or component step for use within the same component. Local output parameter values can be viewed in the business process run results. You define these local output parameters using the Output Options dialog box. You cannot delete local parameters, but you can cancel the input or output to them.
<b>Component parameter</b>	Variable values defined within a component for use in the same component or later components in the business process test.

Component input parameter values can be received and used as the values for specific, parameterized steps in the component.

Component output parameters can be returned as input parameters in components that are used later in the test. These values can also be viewed in the business process test run results.

You define component input and output parameters in the Parameters pane of the Component Settings dialog box, or in the ALM Business Components module.

**Tip:** Additionally, if you are working with a business process test or flow, you can define parameters in the Properties pane.

After you define a parameter you can use it to parameterize a value. Alternatively, you can apply a constant value to the parameter by typing it directly in the **Value** cell.

For instructions on how to parameterize input values, see "[How to Parameterize Input Values \(Keyword Components\)](#)" on the next page.

For instructions on how to parameterize output values, see "[How to Create or Modify an Output Value Step](#)" on page 330.

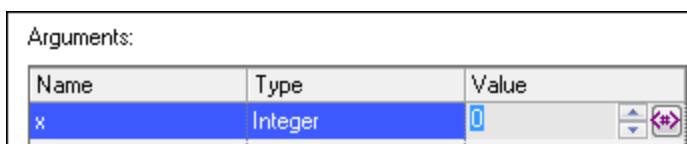
## How to Configure Constant and Parameter Values

### Relevant for: GUI tests and components

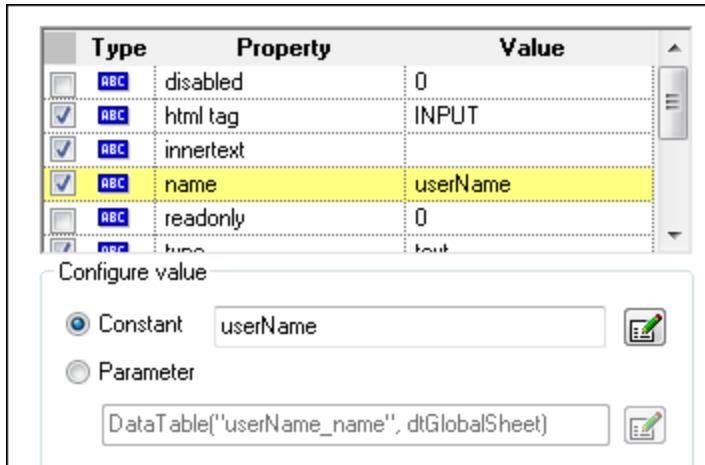
This task describes how to define a value as a constant or a parameter:

Do one of the following:

1. In the **Value** area (available from the Keyword View, Step Generator, Object Repository window, and so on), click the parameterization button  for a selected value to open the Value Configuration Options Dialog Box. For example:



2. Above the **Configure value** area of a dialog box, select a property or argument, for example:



Then, in the **Configure value** area, select the **Constant** or **Parameter** radio button and click the **Constant Value Options** or **Parameter Options** button .

## How to Parameterize Input Values (Keyword Components)

### Relevant for: Keyword GUI components

This task describes how to parameterize input values for a step using a local parameter or a component parameter.

This task includes the following steps:

- "Open the Value Configuration Options dialog box or Parameter Options dialog box" below
- "Select the type of parameter and define its details" on the next page
- "Results" on page 387

### Open the Value Configuration Options dialog box or Parameter Options dialog box

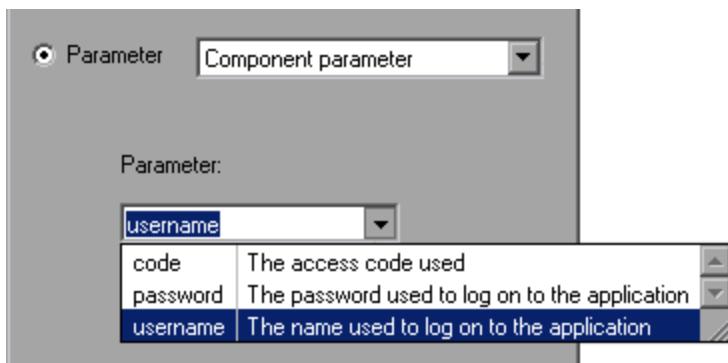
Do one of the following:

- In the Keyword View, click the parameterization button for a selected value.
- In the Checkpoint Properties Dialog Box, click the **Parameter Options** button .
- In the Parameterization / Properties Dialog Box (Checkpoints), select the **Parameter** radio button. If a parameter is already defined, you must then click the **Parameter Options** button to open this dialog box.

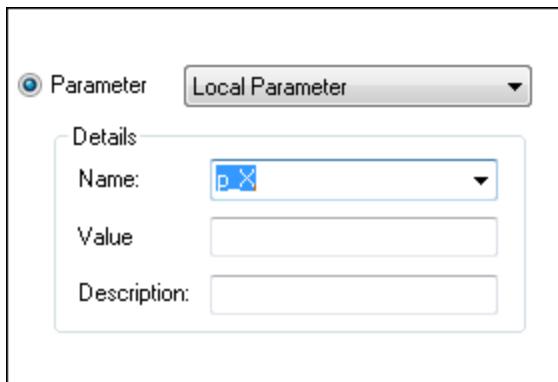
## Select the type of parameter and define its details

1. In the **Parameter** box, select one of the following:

- **Component parameter.** Select the component parameter you want to use for the parameterized value. The names and full descriptions of the available component parameters are displayed as read-only. You can resize the display, as needed, and, if the list of parameters is long, you can scroll through the list.



- **Local parameter.** The details for the local parameter type are displayed.



2. Specify the property details for the local parameter:

- **Name.** Enter a meaningful name (case-sensitive) for the parameter or choose one from the list.
- **Value.** Enter an input value for the parameter.
- **Description.** Enter a brief description for the parameter.

**Tip:**

You can cancel the parameterization of a value by selecting the **Constant** option in the Value Configuration Options Dialog Box and entering a constant value. The Constant box offers the same editing options as the Value cell in which you clicked the parameterization button `<#p>` to open this dialog box.

## Results

The local or component parameter is displayed in the **Value** cell of your step. When the component runs, it will use the value specified in the parameter for the step.

# Part 6: API Testing Design with the UFT IDE

# Chapter 40: API Test Creation Overview

**Relevant for: API testing only**

This chapter includes:

- [UFT API Testing Overview](#) ..... 390
- [Automated Testing Tool Integration](#) ..... 391
- [Automatically Generating API Tests - Overview](#) ..... 392
- [How to Create an API Test](#) ..... 393
- [How to Create an API Test - Use-Case Scenario](#) ..... 397
- [How to Automatically Generate API Tests](#) ..... 403

# UFT API Testing Overview

## Relevant for: API testing only

API Testing is how UFT enables you to test your non-GUI applications to see how your application's API processes performs. Using UFT API testing, you create a test that represents the activities that your application performs (even using actual and output data from your application), and use checkpoints to assess the success or failure of the test.

You design an API test by dragging activities from the Toolbox pane onto the canvas. The Toolbox pane includes a collection of standard activities and technology-specific activities such as HTTP, Java, JMS, IBM WebSphere MQ, and SAP. You can add additional activities to the Toolbox pane by importing WSDLs and WADLs, creating REST Services, or using services from the file system or ALM. You can also create new custom activities using the built-in Activity Wizard as described in "["API Testing Extensibility" on page 603.](#)

UFT provides a number of standard activities that test common application processes, including:

<b>Standard Activities, such as</b>	<ul style="list-style-type: none"><li>• <b>Flow Control</b> activities, such as <b>Wait</b>, <b>Break</b>, and <b>Conditional</b> steps. These activities enable you to customize your test to match any special application workflows.</li><li>• <b>String Manipulation</b> activities, such as <b>Concatenate Strings</b> and <b>Replace String</b>.</li><li>• <b>File system</b> activities, which enable you to test your application's interaction with the file system.</li><li>• <b>Database</b> activities, which enable you to test your application's ability to connect and communicate with a database.</li><li>• <b>FTP</b> activities, which enable you to test your application's ability to perform FTP-related procedures.</li><li>• <b>Network</b> activities, such as <b>HTTP Request</b> and <b>SOAP Request</b>, which enable you to test your application's connection with a network or web-based server.</li><li>• <b>JSON</b> and <b>XML</b> activities, which enable you to test your application's ability to convert XML and JSON data to text and vice versa.</li><li>• <b>Math</b> and <b>Date/Time</b> activities, which enable you to perform math operations or perform tasks using the system date and time.</li><li>• Other <b>Miscellaneous</b> activities, including <b>Custom Code</b> activities, <b>Run</b> and <b>End</b> program activities, and a <b>Report</b> activity.</li></ul>
<b>Technology-specific activities, such as:</b>	<ul style="list-style-type: none"><li>• A <b>Call Java Class</b> activity, which enables you to test Java-based processes that run in your application</li><li>• <b>JMS</b> (Java Message Service) activities, which enable you to test your application's ability to communicate, publish, browse, receive, and check messages from a JMS queue.</li><li>• <b>IBM WebSphere MQ</b> activities, which enable you to test your application's ability to communicate with, publish, browse, receive, and check messages from the IBM WebSphere MQ queue or topic.</li><li>• <b>SAP</b> activities, which enable you to test your application's ability to communicate with an SAP server using IDOCS and RFCS.</li></ul>

	<ul style="list-style-type: none"> <li>• <b>Load Testing</b> activities, which enable you to add steps for your test to be run as a load test in HP LoadRunner.</li> <li>• <b>HP Automated Testing Tools</b> activities, which enable you to call a GUI test or action, API test or action, or Virtual User Generator script from UFT, QuickTest Professional, Service Test, or LoadRunner to use as part of your test.</li> </ul>
<b>Custom activities, based on your services, such as:</b>	<ul style="list-style-type: none"> <li>• <b>Web Service</b> activities imported from a WSDL file. For details, see "<a href="#">How to Import a WSDL-Based Web Service</a>" on page 512.</li> <li>• <b>Web Application</b> activities imported from a WADL file. For details, see "<a href="#">How to Import a Web Application Service</a>" on page 522,</li> <li>• <b>REST Service</b> activities created in UFT using the Add REST Service dialog box. For details, see "<a href="#">How to Create a REST Service</a>" on page 515.</li> </ul>

If the built-in activities are not sufficient for your needs, you can:

- **Use custom code activities that seamlessly integrate with UFT.** Using customized code, you can also customize the behavior of existing activities using event handlers. For details on creating and using custom code with your test, see "[Writing Event Handlers for API Test Steps](#)" on page 751.
- Create custom test activities with the UFT Activity Wizard (**Start > All Programs > HP Software > HPUnified Functional Testing > Tools > Activity Wizard** or <UFT installation folder>\Tools\ActivityWizard\bin\ActivityWizard.exe). The Activity Wizard enables you to specify the activity type and properties. It then exports the activity to the Toolbox pane for use in future testing sessions.

For details on adding new activities and custom code, see "[API Testing Extensibility](#)" on page 603.

## Automated Testing Tool Integration

### Relevant for: API testing only

APITesting integrates with the following other HP products:

- **HP Application Lifecycle Management (ALM).** You can save tests, business component, and test resources on ALM, enabling multiple users to store and access a shared repository of tests and test resources. You can also use ALM's defect tracking abilities to record and manage application defects when running your tests.
- **HP Service Virtualization.** In order to mimic services that your application may use, UFT integrates with HP Service Virtualization. After creating your service models in Service Virtualization, you then run them as a service for your test.

For more details on using Service Virtualization in UFT, see "[Running Tests with Virtualized Services - Overview](#)" on page 877. For more details on HP Service Virtualization, see the *Service Virtualization User Guide*.

- In addition, by using **HP Automated Testing Tools** activities you can integrate with additional HP functional testing products by creating test steps that call a GUI test or action, API test or action, or a LoadRunner script. These tests or scripts are created in the original application and call from within

your test flow.

For task details, see "[How to Call External Tests or Actions](#)" on page 426.

## Automatically Generating API Tests - Overview

In order to create API tests using the full IDE functionality, you must fully understand your application's service layer: what processes are used, what parameters are passed between layers of the service, and other similar details. This can sometimes be very difficult.

However, if you want to quickly create a test of your application, UFT provides tools to automatically generate an API test, including test steps and step property values:

API Test Generator Wizard	<p>This wizard takes the content from a WSDL file and creates a full API test, including all the steps in the test, and all the property values entered. After you select the appropriate file (which contains the meta description of your service), you select the methods to include in the generated test.</p> <p>In addition, UFT can create multiple and separate tests for a number of different testing aspects:</p> <ul style="list-style-type: none"><li>• <b>Positive Testing:</b> A full positive test that checks that the service's operations work as expected. This adds relevant checkpoints for each step (although these are not enabled by default).</li><li>• <b>Standard Compliance:</b> Checks the service compliance with industry standards such as WS-I and SOAP.</li><li>• <b>Security Testing.</b> Tests the security of the service. You can select one of the following types of security testing:<ul style="list-style-type: none"><li>• <b>SQL Injection Vulnerability.</b> This checks if the service is vulnerable to SQL injections by injecting SQL statements and errors into relevant parameters.</li><li>• <b>Cross-site Scripting (XSS).</b> Attempts to hack the service by injecting code with relevant parameters that will disrupt its functionality.</li></ul></li><li>• <b>Boundary Testing.</b> Using the negative testing technique, creates tests to manipulate data, types, parameters, and the actual SOAP message to test the component to its limits. You can select one of the following types of boundary testing:<ul style="list-style-type: none"><li>• <b>Extreme Values:</b> Provides invalid data types to the services and verifies they are not accepted.</li><li>• <b>Null Values.</b> Provides NULL parameters to the components to verify they are not accepted.</li></ul></li></ul>
<b>SOAPUI to Test Converter</b>	If you have previously created tests in SOAPUI, you can convert these tests into API tests. The test is automatically generated, containing the steps and the test properties.

For task details on how to automatically generate your tests, see "[How to Automatically Generate API Tests](#)" on page 403.

# How to Create an API Test

## Relevant for: API testing only

This task describes the workflow and methodology to follow when you create and build a test.

This task includes the following steps:

- "Prerequisite - Analyze your application" below
- "Prerequisite - Configure UFT according to your testing needs" below
- "Prerequisite - Prepare the service references (optional)" below
- "Build your test structure" on the next page
- "Enhance your test steps" on page 395
- "Results" on page 396

## Prerequisite - Analyze your application

Before creating a test, you need to analyze your application and determine your testing needs. You need to:

- **Determine the functionality you want to test.** To do this, you consider the various activities that the application performs. What business processes run? What activities are most relevant for the business processes that you want to test?
- **Identify any processes that run repeatedly.** Plan to create actions in your test for such processes.

As you plan, try to keep the number of steps that you plan to include in each action to a minimum. Creating small, modular actions helps make your tests easier to read, follow, and maintain.

## Prerequisite - Configure UFT according to your testing needs

This can include:

- Setting up your **global testing preferences**.  
For details, see "[Global Options](#)" on page 80.
- Setting up **test-specific preferences**, including global properties, test input and output properties and parameters, or user profiles and variables.  
For details, see "[How to Define API Test Properties or User/System Variables](#)" on page 558.
- Configuring your **run session preferences**.

## Prerequisite - Prepare the service references (optional)

- **Import or build the set of resources** to be used by your tests, including:
  - **Importing WSDL or WADL files**, which define the methods used for a Web service or Web application.

For details, see ["How to Import a WSDL-Based Web Service" on page 512](#) or ["How to Import a Web Application Service" on page 522](#).

- **Creating REST service resources and methods** if your application is based upon REST services.  
For details, see ["How to Create a REST Service" on page 515](#).
- **Importing .NET assemblies** referenced by your test.  
For details, see ["How to Import and Create a .NET Assembly API Test Step" on page 526](#).
- Adding any **additional reference files** that will be referenced by your test.

**Note:** Skip this step when you will be using only the built-in operations. These activities are available in the Toolbox pane under the Standard Activities section.

## Build your test structure

**Note:** API tests cannot be created in a path containing an "equal sign" ('=') character.

To build your basic test structure, do the following:

### 1. Create additional test flow steps- optional

Expand the Toolbox pane nodes and drag **Flow Control** activities onto the canvas:

- **Loop.** Enables you to add another loop (the **Test Flow** loop is always part of a test, and cannot be removed). You specify the loop behavior in the loop's input properties.
- **Condition.** Enables you to define conditional branches.
- **Sleep.** Indicates a time delay in milliseconds.

### 2. Add activities to the test to create test steps

Expand the nodes of the Toolbox pane and drag activities into the **Text Flow** or a **Loop** box within the canvas to create test steps. If you added a **Condition** step, drag activities into the condition branches. For a list of the available activities, see the ["Standard Activity Types" on page 434](#).

### 3. Provide the step properties

Enter the input, output, and checkpoint properties (as needed) for each activity. For details on the input, output, and checkpoint properties available for each activity, see ["Standard Activities" on page 406](#).

**Note:** If you have a number of activities/steps that are repeated often in your test, consider creating an action and adding the steps to this action. After creating the action, you can call the action each time you need to repeat these steps in your test instead of adding the activities and setting the activity's properties repeatedly.

## Enhance your test steps

To enhance your test steps, do any of the following:

- **Define data sources for the test**

For details, see ["How to Assign Data to API Test/Component Steps" on page 551](#).

- **Create a Custom Code activity**

- Select the **Custom Code** activity from the **Miscellaneous** category and drag it into a loop.
- Click the Input/Checkpoints tab  in the Properties pane.
- Click **Add Properties** and create the required input and output properties.
- Open the **Events** tab  in the Properties pane.
- Double-click the **Handler** column of the **ExecuteEvent** row. UFT opens a new tab `TestUserCode.cs`.
- Locate the **Todo** section and enter your custom code. Follow the sample code in the comments and use autocompletion to write your code.
- Click **File > Save All** to save the custom code and the test.

For details and examples, see ["Writing Event Handlers for API Test Steps" on page 751](#).

- **Add input attachments to the test - optional**

For Web Service activities that support input attachments, add an input attachment.

- Select the Web Service call step in the canvas and open the **Attachments**  tab in the Properties pane.
- In the upper pane, select an attachment Type: **DIME** or **MIME**.
- Click in the **Attachments** row and click the **Add** button  to add an array element.
- Select a file as the **Origin** of the attachment using the **Browse** button .
- Select a **Content Type**. Specify a **Content ID** or keep the default value, **Auto**.

- **Add/validate an output attachment- optional**

- Select the Web Service call step in the canvas and open the **Attachments**  view in the Properties pane.
- Click in the **Attachments** row in the Checkpoints pane, and click **Add**  to add an array element (it may be necessary to expand the column).
- Select the check box adjacent to each item that you want to validate. Specify values for the elements being validated: **Content Type** and/or **Content ID**.
- To validate content, click the **Calculate the file checksum**  button icon in the **Content** row. UFT calculates the specified file's checksum using the MD5 Hash function.

You can also check for received attachments in the test's folder, stored as files with a .bin extension.

- **Add an event handler - optional**

For any activities, you can define default event handlers for checkpoints, and before and after step executions.

To add an event handler for a step:

- Select a step in the canvas and open the Events tab  in the Properties pane.
- In the row containing the event execution point you want (before, after, etc.), select **Create a default handler**.
- Edit the code in the TestUserCode.cs tab. Locate the **Todo** section and add your custom code. Follow the sample code in the comments and use statement completion to create an expression.
- To access the properties of an activity, cast it before the activity name. For example:

```
ConcatenateStringsActivity cat = args.Activity as
ConcatenateStringsActivity;
args.Checkpoint.Assert.Equals(cat.Prefix+cat.Suffix, cat.Result);
```

- Click **File > Save All** to save the TestUserCode.cs file and the test.

For details and examples, see "[Writing Event Handlers for API Test Steps](#)" on page 751.

**Tip:** To ignore an event handler for a step, select a handler and press **DELETE** to clear the field in the Events tab. To completely remove the event handler, you must also delete the code manually in the TestUserCode.cs tab.

## Results

After you create your test, you can perform different types of runs to achieve different goals. You can:

- **Run your test to check your application.**

The test starts running from the Start step in the canvas and stops at the end of the test. While running, UFT performs each step in your test, including any checkpoints.

If you parameterized the test using data from a data source stored in the Data pane, UFT repeats the test (or test flow loop, if needed) using the data as defined in the Input tab for the **Test Flow** or test flow steps.

- **Run your test to debug it.**

**Note:** Before running a debugging session, make sure to enable debugging capabilities by selecting the **Run test in debugging mode** option in the General Pane of the API Testing tab of the Options dialog box.

For general details on debugging, see "[Debugging Tests and Components](#)" on page 887.

- **Run an individual step.**

Select the step in the canvas and choose **Run Step** from the context menu to run the step with its property values. Note the results in the **Run Step Results** pane, in the lower section of the main window. If something needs to be modified, do so at this point.

**Note:** The Run Step command, for steps whose properties are assigned an XML data source, does not retrieve the latest data shown in the user interface.

**Workaround:** Save the test before activating the Run Step command.

## How to Create an API Test - Use-Case Scenario

### Relevant for: API testing only

Creating a test is comprised of several stages. This section walks you through the stages you might perform when preparing a test for the Flight API application.

This scenario consists of the following steps:

- ["Analyze the Flight API application" below](#)
- ["Create or import your test resources" on page 399](#)
- ["Create your test steps" on page 400](#)
- ["Enhance your test steps" on page 402](#)
- ["Run the test" on page 403](#)

### 1. [Analyze the Flight API application](#)

When analyzing the application to determine what application processes you may want to test, you can consider the existing business processes that run in the application.

The business processes that should be tested for the Flight API application include:

- Connecting to the user database and confirming login information
- Retrieving the list of flights
- Retrieving a flight order based on user input
- Creating a flight order
- Updating a flight order
- Deleting a flight order
- Deleting all flight orders
- Logging a user out of the application

Although the first and last items above have not yet been implemented in the Flight API application that you want to test, it is important to take them into account in the planning stage.

Now that you have determined the primary application processes, you should analyze each one to determine the breakdown of these application processes into smaller, testable parts.

A logical breakdown of the above business processes could be:

- **Connecting to the user database and confirming login information**

- Sending a request to connect to the database
- Receiving confirmation or failure of database connection
- Checking the customer login details database based on user input
- Returning authentication results (permission or rejection) for user input
- Return results to application user interface

- **Retrieving a list of flights**

- Connecting to the flight information database
- Receiving confirmation or failure of database connection
- Searching the flight information database for all available flights
- Returning the search results

- **Retrieving a flight order based on user input**

- Connecting to the flight information database
- Receiving confirmation or failure of the database connection
- Searching the flight information database using the user-specified criteria
- Returning the search results

- **Creating a flight order**

- Connecting to the flight database for the specific flight based on user input
- Receiving confirmation or failure of the database connection
- Reserving a place on the selected flight in the database
- Returning the order confirmation

- **Updating a flight order**

- Connecting to the flight database and finding the selected flight based on user input
- Receiving confirmation or failure of the database connection
- Retrieving the flight order details
- Updating the flight information in the database based on user input
- Returning the flight order update confirmation

- **Deleting a flight order**

- Connecting to the flight database and finding the selected flight based on user input
- Receiving confirmation or failure of the database connection
- Retrieving the flight order details
- Deleting the flight order from the database
- Returning confirmation of the flight order deletion

By comparing the sub-steps in each of the business processes, you can see what specific steps you need to design in your test. In addition, you can also see the steps that are repeated and can be combined in a reusable action that can be called in different parts of your test.

## 2. **Create or import your test resources**

Some of your application processes for the Flight API application require custom activities not included in the standard collection of API activities provided in the Toolbox pane. For these activities, you must import or create the activities into your tests.

At this stage we can import the following Web services methods:

- CreateFlightOrder
- GetFlights
- GetFlightOrders
- UpdateFlightOrder
- DeleteFlightOrder
- DeleteAllFlightOrders

For details on how to import Web Service methods, see "[How to Import a WSDL-Based Web Service](#)" on page 512.

You can also create the following REST Service resources and methods:

- Flights Get
- Flight Get
- FlightOrders Get
- FlightOrders ReserveOrder
- FlightOrder Get
- FlightOrder Update

- FlightOrder Delete
- FlightOrder DeleteAll

For details on how to create REST Service methods, see ["How to Create a REST Service" on page 515.](#)

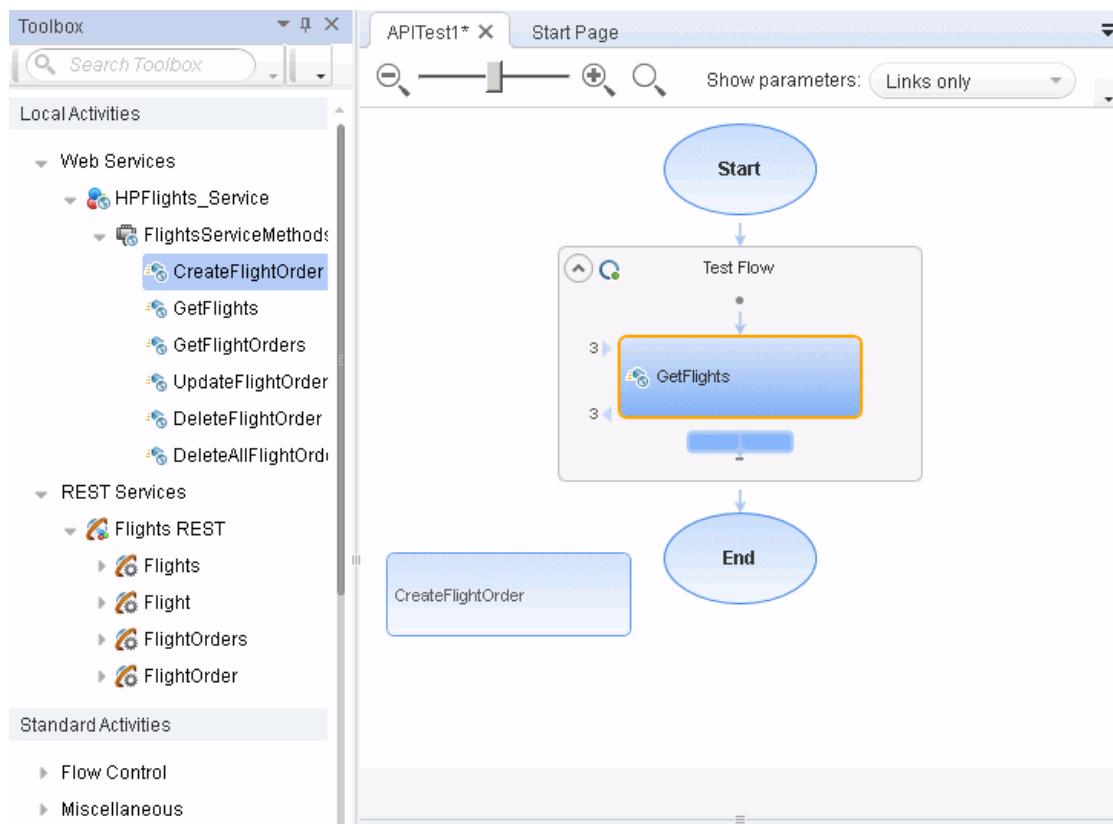
### 3. **Create your test steps**

Now that you have planned and prepared all of the required resources for your test, you are ready to create test steps that represent the steps a real application would perform on the Flight API application.

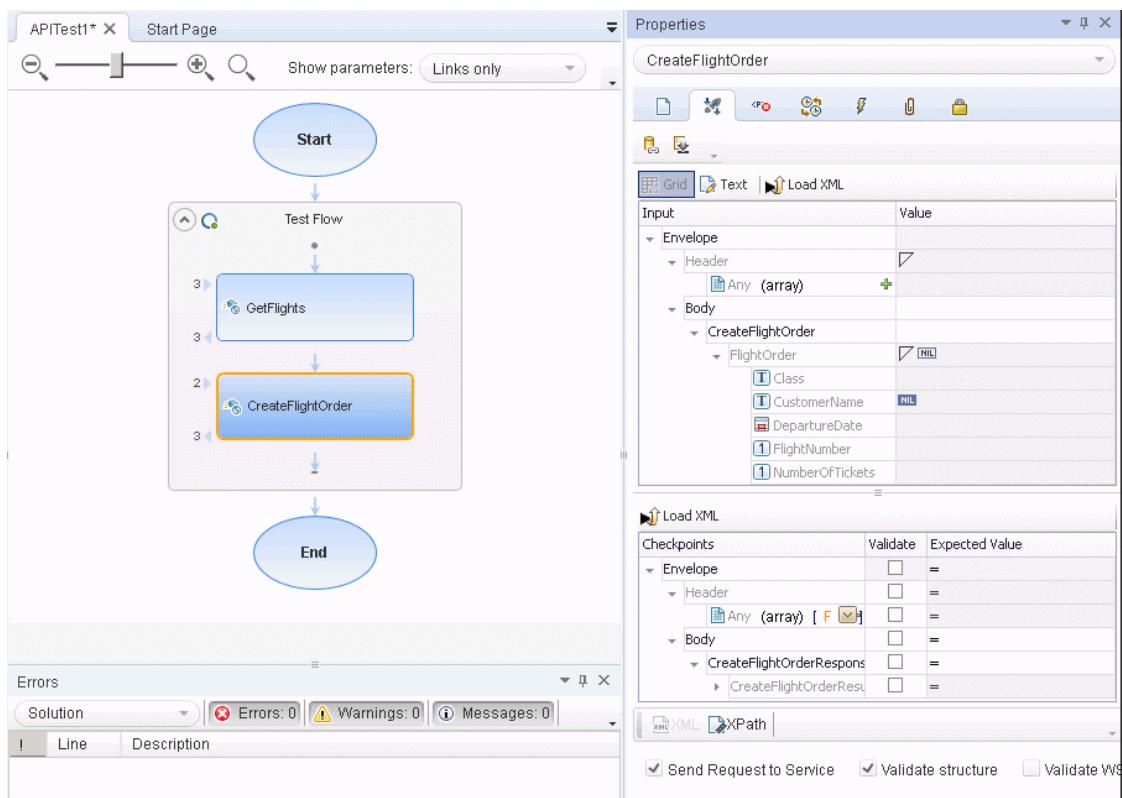
The activities that are available for your test steps are stored in the Toolbox pane. This includes custom methods that were imported or created for your test:



To create test steps, you select activities from the Toolbox pane and either drag them to the canvas or double-click them to add them to the canvas:



You then set input, output, or checkpoint property values for the activities to mimic your application:



If you have steps that appeared multiple times, you can create an action which combines the repeated steps and call this action instead of the repeated steps.

For task details on creating test steps, see "[How to Create an API Test](#)" on page 393.

#### 4. Enhance your test steps

Once you have put the appropriate steps for the Flight API application in the canvas, you can add additional enhancements for these test steps:

- You can set checkpoint properties for a test step, providing the expected output values for each of the methods.
- You can link the test steps to a data source, such as an Excel file containing different values for the step input properties. This enables you see how the test steps run with different input values.
- You can link one step to another. For example, you can link the GetFlights Web Service method to the CreateFlightOrder method to pass the flight information between the two methods.
- You can add additional event handlers to enhance the step function before the test step runs, during the running of the test step, and after the step's execution.

## 5. Run the test

After you add your test steps, and provide the input, output, or checkpoint property values, you run your test and observe the results. In the run results, display the Test Flow and observe the results and response for each of the test steps.

If you want to test each individual step after entering its properties, you can right-click the step name in the canvas and select Run Step. UFT runs only that step and displays the step results in the Run Step Results pane.

# How to Automatically Generate API Tests

## Relevant for: API tests only

This task describes how to automatically generate API tests from external documents. This can be helpful if you have existing API resources (such as WSDL documents or other service model description documents) that need to be made into a functional test of your application's API.

**Note:** This task is part of a higher-level task. For details, see "[How to Create an API Test](#)" on page [393](#).

This task includes the following steps:

- "[Generate your test from a WSDL file](#)" below
- "[Generate your test from a SOAPUI test file](#)" on the next page

## Generate your test from a WSDL file

Using the API Test Generator Wizard, you can create full tests (with all steps and step property values entered) from a WSDL file:

1. From the Start Menu, open the API Test Generator Wizard (**Start > All Programs > HP Software > HP Unified Functional Testing > Tools > API Test Generator Wizard**) and click **Next**.
2. In the Select Service window, select from where to import your WSDL file:
  - **URL:** The URL in which this file is stored. Make sure that the URL is accessible when you try to select the service.
  - **File:** A location on the file system.
3. Click **Next**.
4. In the Select Methods window, select the methods to use in the test. These methods are created from the metadata specified in your WSDL file.
5. In the Select Aspects window, select the types of tests you would like to create:

<b>Positive Testing</b>	A full positive test that checks that the service's operations work as expected. This adds relevant checkpoints for each step (although these are not enabled by default).
<b>Standard Compliance</b>	Checks the service compliance with industry standards such as WS-I and SOAP.
<b>Security Testing - SQL Injection Vulnerability</b>	Checks if the service is vulnerable to SQL injections by injecting SQL statements and errors into relevant parameters.
<b>Security Testing - Cross-site Scripts (XSS)</b>	Attempts to hack the service by injecting code with relevant parameters that will disrupt its functionality.
<b>Boundary Testing - Extreme Values</b>	Provides invalid data types to the services and verifies they are not accepted.
<b>Boundary Testing - Null Values</b>	Provides NULL parameters to the components to verify they are not accepted.

6. In the bottom of the Select Aspects window, specify where to save the created API test. By default, this folder is **C:\GeneratedAPITests**.

7. Click **Next**.

UFT automatically generates the test. Generation progress and error details are displayed in the Generate window.

If you need to view log details on the test generation, or open the test folder, you can access these from the Generate window.

## Generate your test from a SOAPUI test file

If you previously had tests created using SOAPUI, you can automatically import these files into UFT to create Web service tests.

- From the Start Menu, open the SOAPUI to API Test Converter (**Start > All Programs > HP Software > HP Unified Functional Testing > Tools > SOAPUI to API Test Converter**).
- In the SOAPUI to API Test Converter window, navigate to the directory in which your SOAPUI test was saved.
- Select a destination directory for the converted test and click **Convert**. A full API test (with a .st extension) is created in the specified folder.

You can also use the following command line options to convert a test:

```
soapUI2APITestCMD.exe /<source soapUI_file> /destination <destination directory>/
logs <log_directory>
```

Command Line Switch	Description
<b>/source</b>	The absolute path to the soapUI file with an .xml extension, to be converted.

Command Line Switch	Description
<b>/destination</b>	The absolute path of the folder to where the created API tests will be written.
<b>/logs (optional)</b>	The absolute path of the folder in which to write the log file. If this option is omitted, the log file is written to the destination folder.
<b>-? or /?</b>	Show the parameters and their usage. For example: soapUI2APITestCMD.exe -?

# Chapter 41: Standard Activities

**Relevant for: API testing only**

This chapter includes:

• <a href="#">Activity Overview</a> .....	408
• <a href="#">Checkpoint Validation for Test Steps</a> .....	408
• <a href="#">XPath Checkpoints</a> .....	409
• <a href="#">How to Set Array Checkpoints for Test Steps</a> .....	410
• <a href="#">How to Set XPath Checkpoints for Test Steps</a> .....	412
• <a href="#">How to Use Flow Control Activities</a> .....	413
• <a href="#">How to Execute Database Commands or Retrieve Data</a> .....	416
• <a href="#">How to Send a Multipart HTTP or REST Service Request</a> .....	418
• <a href="#">How to Create a Call to a Java Class</a> .....	420
• <a href="#">How to Use JMS Activities</a> .....	422
• <a href="#">How to Create an SAP API Test Step</a> .....	425
• <a href="#">How to Call External Tests or Actions</a> .....	426
• <a href="#">How to Prepare and Run a Load Test</a> .....	429
• <a href="#">How to Test Web Sockets Communication</a> .....	431
• <a href="#">Standard Activity Types</a> .....	434
• <a href="#">Flow Control Activities</a> .....	437
• <a href="#">Miscellaneous Activities</a> .....	439
• <a href="#">String Manipulation Activities</a> .....	442
• <a href="#">System Activities</a> .....	444
• <a href="#">Math Activities</a> .....	444
• <a href="#">Date and Time Activities</a> .....	444
• <a href="#">File System Activities</a> .....	447
• <a href="#">Database Activities</a> .....	452
• <a href="#">FTP Activities</a> .....	457
• <a href="#">Network Activities</a> .....	465
• <a href="#">JSON Activities</a> .....	470
• <a href="#">Java Activities</a> .....	471
• <a href="#">JMS Activities</a> .....	472

• <a href="#">Load Testing Activities</a> .....	<a href="#">478</a>
• <a href="#">IBM WebSphere MQ Activities</a> .....	<a href="#">479</a>
• <a href="#">HP Automated Testing Tools Activities</a> .....	<a href="#">492</a>
• <a href="#">SAP Activities</a> .....	<a href="#">492</a>
• <a href="#">XML Activities</a> .....	<a href="#">494</a>
• <a href="#">Web Sockets Activities</a> .....	<a href="#">497</a>
• <a href="#">Troubleshooting and Limitations - Standard Activities</a> .....	<a href="#">500</a>

# Activity Overview

## **Relevant for: API testing only**

You create a test by double-clicking or dragging **activities** from the Toolbox pane into a canvas. The Toolbox pane provides a collection of built-in standard activities for functional testing in areas such as file and string manipulation, and messaging through HTTP, FTP, and JMS.

The activities are divided into several categories:

- **Standard Activities.** This category includes the built-in activities, such as **String Manipulation**, **Database**, **Network**, **File System**. For a complete list of the standard activities, see "[Standard Activity Types](#)" on page 434.
- **Local Activities.** This category includes the activities that are stored as part of the test or business component. These can be imported services such as Web or REST service operations, and .NET assemblies. For details, see "[Custom Activities](#)" on page 502.
- **File System Activities.** This category includes activities that reside in the file system on either local or network drives. These are Local activities that you moved into the file system repository that can be shared between tests.
- **ALM Activities.** This category includes the activities that reside in the ALM repository. These are Local activities that you moved into the ALM repository that can be shared between tests. This category only appears when a connection to an ALM server is open.

You can also create new custom activities, using the extensibility API provided with the product. These will be stored under the Standard Activities. For details, see "[API Testing Extensibility](#)" on page 603.

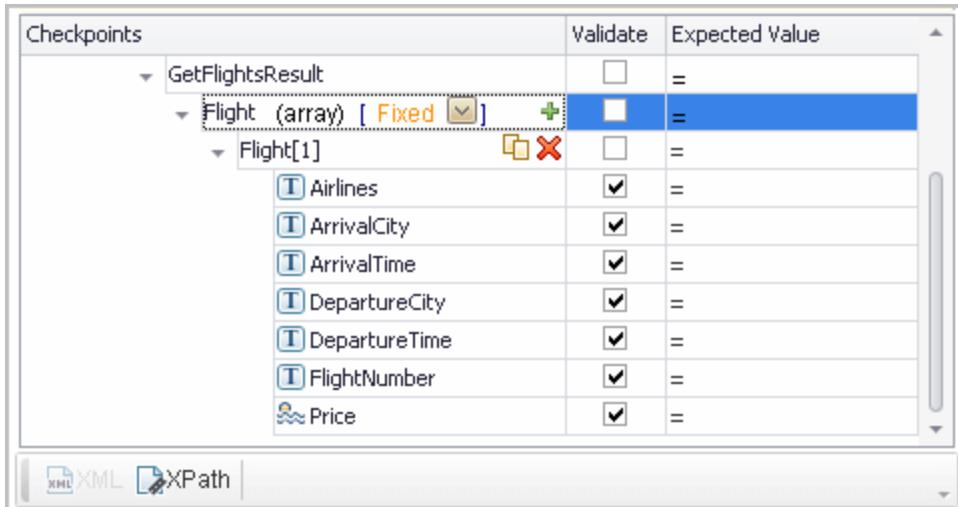
For general information about the API interface and creating tests, see "[API Test Creation Overview](#)" on page 389.

# Checkpoint Validation for Test Steps

## **Relevant for: API testing only**

In functional testing, it is necessary to check the results of test steps to confirm that the activity performed its expected functionality. The response can contain several properties, each containing several data items. The **Checkpoints** section is a central point for defining the expected values of the properties.

Before replaying a test, you set the expected output property values. You can enter the values manually, link them to a data source, or load values that were captured during a prior replay. This is useful when you have many argument values—instead of manually entering values, you automatically load them.



To include a checkpoint in the test, you select its check box. If you loaded replay values, you can select only those properties that you loaded.

For WSDL-based Web Services and SOAP Requests, UFT includes two built-in checkpoints for the purpose of validation. One checkpoint validates the XML structure and the other checks its compliance with WS-I.

Additional checkpoint settings let you trim the string, ignore case inconsistencies, and indicate whether to stop on failed checkpoints.

After the test run, the run results displays the checkpoint status in its own sub-node.

## XPath Checkpoints

### Relevant for: API testing only

For steps with XML output properties, such as **Web Service** and **SOAP Request, String to XML**, and so forth, you can validate the test results against XPath expressions.

You can specify a fully qualified XPath expression, or you can instruct UFT to ignore the namespaces and prefixes during the test run. By ignoring the namespaces, you can use a simpler expression.

In the following example, to retrieve the contents of the second node, B, you would need to write an expression that also indicates the namespace, such as `//*[@local-name(.)='Node' and namespace-uri(.)='ns2']`.

```
<Root>
 <Body>
```

```
<Node xmlns="ns1">A </Node>
<Node xmlns="ns2">B </Node>
</Body>
</Root>
```

When working with simple XPath expressions, you can further simplify the XPath expression by selecting **Ignore namespaces**. In the above example, the expression `//Node[2]` is sufficient to evaluate the value B in the second node.

You can type in the XPath expressions manually, or use the shortcut menu to retrieve the simplified or fully qualified XPath.

UFT also enables you to evaluate XML with namespace prefixes. For example, if the XML contains the prefix definition `xmlns:T="ns1"`, you can specify the prefix in the XPath expression: `//T:NodeName`. To evaluate namespace prefixes, disable the **Ignore namespaces** option.

XPath checkpoints can only be used when the XPath query returns a scalar value—not XML.

For task details, see "[How to Set XPath Checkpoints for Test Steps](#)" on page 412.

## How to Set Array Checkpoints for Test Steps

### **Relevant for:** API testing only

This task describes how to manually set checkpoints for elements of an array. If you want to learn about data driving array checkpoints, see "[How to Data Drive Array Checkpoints](#)" on page 562.

This task includes the following steps:

- "[Enable active content on your computer](#)" below
- "[Add a step with an array output](#)" on the next page
- "[Select an array validation method](#)" on the next page
- "[Add array elements](#)" on the next page
- "[Provide values](#)" on the next page
- "[Validate the element count - optional](#)" on the next page
- "[Set the number of iterations and run the test](#)" on page 412

### **1. [Enable active content on your computer](#)**

To enable this view, modify your browser settings as follows:

- a. In Internet Explorer, select **Tools > Options**.
- b. Select the **Advanced** tab.
- c. Enable the option **Allow active content to run in files on My Computer** in the **Security** section.
- d. Click **OK** and close the browser.

## 2. Add a step with an array output

Add a test step with output properties in the form of an array.

## 3. Select an array validation method

Expand the drop down adjacent to the name of the parent array node. Select one of the following:

- **Fixed.** Checks that each of the returned array elements matches its corresponding array element in the **Checkpoints** pane. Each array is marked by an index number, as it checks the arrays by their index.
- **All.** Checks that all of the returned array elements match the array element in the **Checkpoints** pane. In this mode, arrays are not marked by an index number. For example, if a property in the first array is marked **>= 2** and the same property in another array element is set to **<=10**, the test run will check that all returned values are between 2 and 10.
- **Contains.** Checks that at least one of the returned array elements matches the value of the property in the **Checkpoints** pane. In this mode, arrays are not marked by an index number.

## 4. Add array elements

Use the plus button **+** in the row of the parent node to add the desired number of array elements.

## 5. Provide values

Select the **Validate** box for each value that you want to validate and provide values for those properties.

Checkpoints	Validate	Expected Value
Body	<input type="checkbox"/>	=
GetFlightsResponse	<input type="checkbox"/>	=
GetFlightsResult	<input type="checkbox"/>	=
Flight (array) [ Fixed ] <b>+</b>	<input type="checkbox"/>	=
Flight[1]	<input checked="" type="checkbox"/>	=
Flight[2]	<input checked="" type="checkbox"/>	=
Airlines	<input checked="" type="checkbox"/>	= Blue Skies
ArrivalCity	<input checked="" type="checkbox"/>	= Denver
ArrivalTime	<input type="checkbox"/>	=
DepartureCity	<input checked="" type="checkbox"/>	= Los Angeles
DepartureTime	<input type="checkbox"/>	=

## 6. Validate the element count - optional

Click in the parent row of the array and enter a desired count number in the **Expected Value** column and the evaluation expression, such as **=**, **>**, and so forth. During the test run, the validation will check the number of returned array elements against this value.

## 7. Set the number of iterations and run the test

Click in the **Test Flow** or **Loop** box and open the Properties pane. Set the test flow properties, such as the number of iterations or loop type. Click the **Run** button and provide a results location.

# How to Set XPath Checkpoints for Test Steps

### Relevant for: API testing only

This task describes how to validate your test results against XPath expressions.

This task includes the following steps:

- "Add a step with XML output" below
- "Set the namespace setting" below
- "Copy the XPath" below
- "Add an XPath checkpoint" below
- "Provide the XPath expression" on the next page
- "Set up the validation" on the next page

### 1. Add a step with XML output

- a. From the Toolbox pane, add a test step with XML output, such as **String to XML**.
- b. In the Properties pane, select the **Input/Checkpoints** tab .
- c. In the Value column for a step property, paste a source string or use the **Link to data source** button to link to a value.

### 2. Set the namespace setting

Click the **XPath** tab, and indicate whether or not to ignore namespaces. By default, the **Ignore namespaces** option is enabled. If you plan to validate against a fully qualified expression, or if you need to specify a namespace prefix, disable the option.

### 3. Copy the XPath

Copy an XPath expression to the clipboard. If you intend to enter the XPath expression manually, skip this step.

- To retrieve a simple XPath, click in the **Value** column, and select **Copy XPath** from the shortcut menu.
- To retrieve a complete qualified XPath, click in the **Value** column, and select **Copy Fully Qualified XPath** from the shortcut menu.

### 4. Add an XPath checkpoint

- a. In the Checkpoints section of the Input/Checkpoints tab, open the **XPath** tab in the Checkpoints section.
- b. In the XPath tab, click the **Add** button .

## 5. Provide the XPath expression

Type or paste the expression into the **XPath Checkpoints** column. You can also use the **Link to data source** button to link to a value.

## 6. Set up the validation

Select the check box in the **Validate** column, select a comparison operator, and provide an expected value.

XPath Checkpoints	Validate	=	Value
//Node[1]	<input checked="" type="checkbox"/>	=	A
//Node[2]	<input checked="" type="checkbox"/>	=	B

# How to Use Flow Control Activities

## Relevant for: API testing only

This task describes how to use Flow Control activities in an API test.

This task includes the following steps:

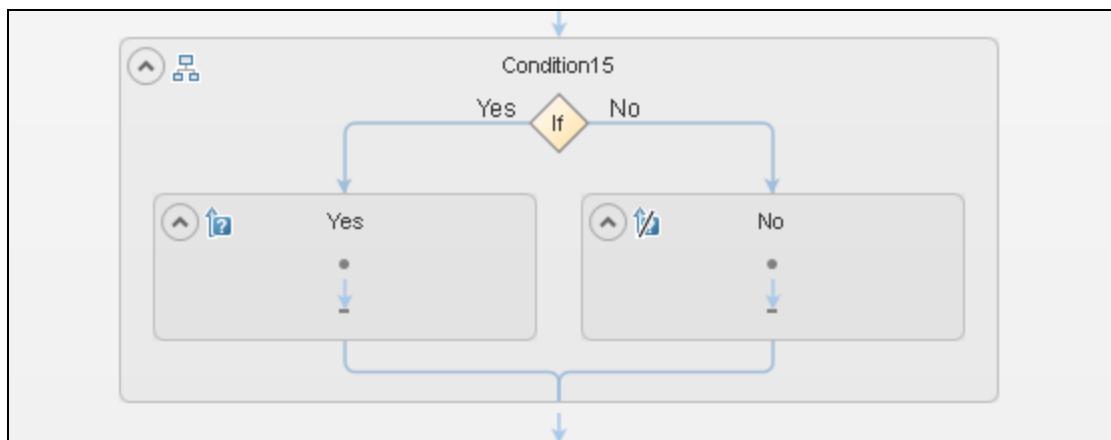
- "Add a conditional step" below
- "Add a loop" on page 415
- "Add steps to pause and restart the test" on page 416
- "Add a wait step" on page 416

## Add a conditional step

Conditional steps enable you to specify alternate test paths depending on the outcome of a previous step. This is the equivalent of an If...Else function in an application's code.

1. In the Toolbox pane, expand the **Flow Control** activities node and drag a **Condition** step to the canvas.

The canvas displays a branched test step, with **Yes** and **No** branches, as see in the example below:



**2. Set the trigger for each branch of the step.**

You can set the trigger for the **Yes** or **No** branches in the following ways:

<b>Specify a condition for the output of a test step</b>	<ol style="list-style-type: none"> <li>In the <b>Condition</b> tab  in the Properties pane, select the <b>Use condition</b> option. The condition properties are displayed.</li> <li>In the <b>Variable</b> field, click the <b>Link to data source</b> button .</li> <li>In the <b>Select Link Source</b> dialog box, link to the output of a previous step.</li> </ol> <p><b>Note:</b> You must link the <b>Variable</b> field to a previous step to set the value to meet to trigger the condition.</p> <ol style="list-style-type: none"> <li>Specify the expected <b>Value</b> and the <b>Operator</b> for the step selected in the <b>Variable</b> field.</li> </ol>
<b>Use an event handler</b>	<ol style="list-style-type: none"> <li>In the <b>Condition</b> tab  in the Properties pane, select the <b>Use event</b> option.</li> <li>In the <b>Events</b> tab , in the <b>Condition</b> row, click the drop-down arrow and select <b>Create a default handler</b>. The <b>TestUserCode.cs</b> file opens in the document pane.</li> <li>In the <b>IfElse&lt;#&gt;_OnCondition</b> section of the <b>TestUserCode.cs</b> file, replace the <b>TODO</b> section with your event handler code:</li> </ol> <pre> 34: 35: 36:     /// &lt;summary&gt; 37:     /// Handler for the IfElse9 Activity's Condition event. 38:     /// &lt;/summary&gt; 39:     /// &lt;param name="sender"&gt;The activity object that raised the event.&lt;/param&gt; 40:     /// &lt;param name="args"&gt;The event arguments passed to the event handler.&lt;/param&gt; 41:     /// Use this.IfElse9 to access the IfElse9 Activity's condition. 42:     public bool IfElse9_OnCondition(object sender, STActivityEventArgs args) 43:     { 44:         //TODO: Add your code here... 45:         return false; 46:     } 47: </pre>

**3. Drag activities to the **Yes** and **No** branches of the Condition step to create the steps to run based on the results of the condition.**

## Add a loop

1. In the Toolbox pane, expand the **Flow Control** activities node and drag a **Loop** step to the canvas.
2. In the **Input** tab of the Properties pane, select the Loop type and define the loop properties.

Select one of the following loop types:

- **'For' Loop**

This loop runs the included steps the specified number of times. Set the number of iterations to run the loop.

**Note:** If the number of iterations is defined elsewhere (the result of a previous step or a data table), link to the property by clicking the **Link to data source** button  in the **Value** cell for the **Number of Iterations** property.

- **'Do While' Loop**

This loop runs indefinitely until a specific condition is met. When the condition is met, the test advances to the steps following the loop.

You can set the loop run properties in the following ways:

<b>Specify a condition</b>	<ol style="list-style-type: none"> <li>i. In the <b>Condition</b> tab  in the Properties pane, select the <b>Use condition</b> option. The condition properties are displayed.</li> <li>ii. In the <b>Variable</b> field, click the <b>Link to data source</b> button .</li> <li>iii. In the Select Link Source dialog box, link to the output of a previous step.</li> </ol> <p><b>Note:</b> You must link the <b>Variable</b> field to a previous step to set the value to meet to trigger the condition.</p> <ol style="list-style-type: none"> <li>iv. Specify the <b>Value</b> and the <b>Operator</b> for the value of the step selected in the Variable field.</li> </ol>
<b>Use an event handler</b>	<ol style="list-style-type: none"> <li>i. In the <b>Condition</b> tab  in the Properties pane, select the <b>Use event</b> option.</li> <li>ii. In the <b>Events</b> tab , in the Condition row, click the drop-down arrow and select <b>Create a default handler</b>. The TestUserCode.cs file opens in the document pane.</li> <li>iii. In the <b>Loop_OnCondition</b> section of the TestUserCode.cs file, replace the <b>TODO</b> section with your event handler code.</li> </ol>

- **'For Each Loop**

This loop runs one time for each item in a selected collection (usually an array). To link this loop to a collection from a different step, click the **Link to data source** button  and select the collection in the Select Link Source dialog box.

3. Drag activities inside the loop.
4. Associate data sources with the loop.

You can assign a specific data source to use with the current loop. The data from this data source is then available for all steps included in the loop. For details, see "[Add a data source to the Test Flow or test loop](#)" on page 566.

## Add steps to pause and restart the test

You can add any of the following steps to your test to pause and restart the test:

- **Break.** This step stops the test. You insert a **Continue** step to resume the test.
- **Continue.** This step resumes the test after a **Break** step stops it.
- **Sleep.** This step temporarily pauses the test for a specified number of milliseconds. Enter the number of milliseconds to wait in the **Input/Checkpoints** tab for the step.

## Add a wait step

Wait steps are mandatory when your application uses an asynchronous service call. After the call to the service, you insert the Wait step. While the test is waiting for the response from the service call, the test waits the amount of time specified in this step.

1. In the Toolbox pane, expand the **Flow Control** activities node and drag a **Wait** step to the canvas.
2. In the **Input/Checkpoints** tab in the Properties pane, Set the step properties:
  - **Timeout:** the number of milliseconds to pause the test.
  - **Start timeout from step:** the step for which UFT is waiting for a response.
  - **Action on Timeout:** what happens when the end of the timeout is reached and the step's response is not received.
  - **Completion events:** the events that signify completion of the timeout.

# How to Execute Database Commands or Retrieve Data

## Relevant for: API testing only

The following describes how to use database activities.

This task includes:

- "[Open a connection](#)" on the next page
- "[Add a Select Data step](#)" on the next page
- "[Add an Execute Command step](#)" on the next page
- "[Add database transaction activities](#)" on page 418
- "[Add a Close Connection step](#)" on page 418

## Open a connection

This step is mandatory for all of the following steps.

1. From the Toolbox pane, add a **Database > Open Connection** activity to the canvas.
2. In the Properties pane, select the **Input/Checkpoints** tab .
3. In the Input/Checkpoints tab, select the **Connection string** input property.
4. In the Value column, click the **Connection Builder** button . The Connection Builder dialog box opens.
5. Paste in an existing string into the text area or click the  **Connection Builder** button to open Microsoft's Data Link Properties dialog box.  
If necessary, in the Data Link Properties dialog box, provide the required information and click **OK**.

**Note:** The database connection must be an OLE DB type.

## Add a Select Data step

1. From the Toolbox pane, add **Database > Select Data** activity to the canvas below an **Open connection** step.
2. In the Properties pane, select the **Input/Checkpoints** tab .
3. In the Input/Checkpoints tab, select the **Connection** input property.
4. In the **Value** column of the Connection property, click the **Link to a data source** button .
5. In the Select Link Source dialog box, select an earlier **Open connection** step in the left pane. In the right pane, select the step's result.
6. In the Input/Checkpoints property, select the Query string property.
7. In the Value column of the Query string property, click the **Browse** button  to open the Query Builder.
8. In the Query Builder dialog box, paste in a query or use the Query Designer.
9. Optionally, set the **Timeout**, the maximum time allowed for the database response. To allow an unlimited amount of time, enter 0.

## Add an Execute Command step

1. Drag the **Database > Execute Command** activity onto the canvas, below the **Open Connection** step.
2. Click the Input/Checkpoints tab in the Properties pane.
3. Set the Input property—**Connection**. Click the **Link to a data source** button by the right side of the **Value** column . In the Select Link Source dialog box, select an earlier **Open connection** step in the left pane. In the right pane, select the step's result.
4. Set the Input property—**Command**. Click the arrow to open an edit box, Paste in a command and click **OK**.

5. Optionally, set the **Timeout**, the maximum time allowed for the database response.

## Add database transaction activities

You can optionally add transaction activities to your test.

1. From the Toolbox pane, add a **Database > Begin Transaction** activity to the canvas after an **Open Connection** step and before the steps to be included in the transaction.
2. Add a **Database > Commit Transaction** activity to the canvas after the database steps that make up the transaction.
3. Drag a **Database > Rollback Transaction** activity to the canvas after the database steps that make up the transaction.

**Tip:** A common use is to insert a **Condition** step to check a value after the database commands. Set one branch with **Commit Transaction** and the other branch to **Rollback Transaction**.

## Add a Close Connection step

1. From the Toolbox pane, add a **Database > Close Connection** activity on to the canvas, after all of the database steps.
2. In the Properties pane, select the **Input/Checkpoints** tab 
3. In the Input/Checkpoints tab, select the **Connection** input property.
4. In the Value cell of the Connection input property, click the **Link to a data source** button .
5. In the Select Link Source dialog box, select the **Available steps** option.
6. In the left pane, select the earlier step, **Open Connection**. Then, In the right pane, select the **Connection** node.

# How to Send a Multipart HTTP or REST Service Request

### Relevant for: API testing only

The following describes how to send an HTTP Request or REST Service method request that uses multiple parts.

This task includes:

- "[Add an HTTP or REST Service step](#)" on the next page
- "[Set the General properties](#)" on the next page
- "[Set the properties for the first part](#)" on the next page
- "[Set the properties for the other parts](#)" on the next page

## 1. Add an HTTP or REST Service step

Do one of the following:

- Drag the **Network > HTTP Request** activity onto the canvas.
- Drag a REST Service method onto the canvas.

## 2. Set the General properties

- a. In the Properties pane, open the **General** tab .
- b. Set the step's general properties.

**Note:** The same properties are also used for REST Service steps.

## 3. Set the properties for the first part

Set the input and checkpoint properties for the first part of the request:

- For an **HTTP Request** step, these properties are found in the **Input/Checkpoints** tab .
- For a REST Service step, these properties are found in the **HTTP Input/Checkpoints** tab .

## 4. Set the properties for the other parts

- a. Open the **Multipart** tab  in the Properties pane.
- b. Select the **Enable Multipart** option.
- c. Set the multipart type.
- d. Provide the details for the general header:
  - i. Click the **Add** button  to add a header.
  - ii. Expand the **Headers (array)** element.
  - iii. In the Headers[<number>] array property, enter the **Name** and **Value** for the general header.
- e. Add additional arrays to the Parts array corresponding to the number of parts. Click the **Add** button  and add additional parts for each part in the multipart request.
- f. Set the properties for the parts:
  - i. In the **Headers (array)** cell, click the **Add** button  to add an request header.
  - ii. Expand the **Headers[<number>]** array
  - iii. In the Headers array, enter the value for the **Name** and **Value** properties for the part request header.
  - iv. In the **Value** column for the **Path** property, enter the path to the request header or click the **Browse** button  and navigate to the reader header.

- g. To set a checkpoint for the parts, select the box in the **Validate** column and provide the expected values as described above.

## How to Create a Call to a Java Class

### Relevant for: API testing only

The Call Java Class activity lets you add Java steps to your test script. This feature enables you to incorporate existing Java code into your test.

- "Set the global Java settings - optional" below
- "Implement the Java interface" below
- "Compile the Java source code" below
- "Package your custom step - optional" on the next page
- "Set up the Java environment - optional" on the next page
- "Add a Call Java Class activity" on the next page
- "Set the Java step property values" on the next page

#### 1. Set the global Java settings - optional

- a. In the canvas, select the Start or End steps in the canvas
- b. In the Properties pane, open the **Test Settings** tab  . Set the required values as needed in this tab.

#### 2. Implement the Java interface

Open the <installation\_folder>\addins\ServiceTest\JavaCall\Java Interface\src\hp\st\ext\java folder and create an implementation for the java interface. For an example, see the sample subfolder.

This interface includes the essential information for the Java call, such as input properties, output properties, and a point of entry. The following methods are included:

- **getInputProperties.** Returns a mapping of the input property names and their Java class.
- **getOutputProperties.** Returns a mapping of the output property names and their Java class.
- **Execute.** A method that receives the mapping of the input property names and their actual values i.e. their object instance. In this method, you process input properties and delegate them to your own Java artifacts. Afterward you process the output properties and send their mappings and their actual values as the method's output.

#### 3. Compile the Java source code

Compile the java files located in the <installation\_folder>\addins\ServiceTest\JavaCall\Java Interface\src\hp\st\ext\java folder.

**Tip:** To determine which JDK to use for, check the version of Java JRE installed with UFT. Open the <installation\_folder>/jre/bin folder and right-click the java.exe file. Select **Properties** and open the **Version** tab.

#### 4. Package your custom step - optional

Package your java classes into a .jar file.

#### 5. Set up the Java environment - optional

- a. In the canvas, select a Start or End step.
- b. In the Properties pane, open the **Test Settings**  tab in the Properties pane. In the Test Settings tab, set the values for the VM and JMS properties.
- c. In the Toolbox pane, expand the **JMS** node in the Toolbox pane and drag a **JMS** activity onto the canvas.
- d. In the Input/Checkpoints tab  of the Properties pane, set the step's properties.  
You must enter a value for the following properties:
  - **Queue**
  - **Subscription**
  - **Topic name**
- e. If you are using a **Send** activity, specify a message.
- f. If you are using a **Receive** activity, in the Checkpoints section of the Input/Checkpoints tab, select the output properties you want to validate and specify the expected values..

#### 6. Add a Call Java Class activity

In the toolbox pane, expand the **Java** node, and drag the **Call Java Class** activity onto the canvas.

#### 7. Set the Java step property values

- a. In the canvas, select the Java step in the canvas
- b. In the Properties, open the Input/Checkpoints tab .
- c. In the Input/Checkpoints tab, click the **Java Class** button to open the Java Class Dialog Box.
- d. In the Java Class dialog box, do the following:
  - i. **Provide a classpath.** If you packaged your Java step, click the **Browse** button adjacent to the **Jar** field and point to a .jar file. Alternatively, click the **Browse** button adjacent to the **Package root** field and point to a package root folder.

**Note:** To embed the jar file and save it with the test, select **Embed Jar in Test**. Due to a technological limitation, if you intend to specify a class file, you must select the

**Embed Jar in Test** option before you browse for the class file.

- ii. For the **Class file** field, click the **Browse** button adjacent to the **Class file** field to locate the class within the .jar file or the folder. Make sure it is a class that implements the **ServiceTestCall** interface.
- iii. To provide additional classpaths, click the **Jar** or **Folder** buttons in the **Additional Classpaths** section and browse to a .jar file or classpath folder. Click **Add** to move the contents into the list.
- iv. Click **OK** to save the Java Call settings.

## How to Use JMS Activities

### Relevant for: API testing only

This task describes how to send, receive and browse JMS messages on a JMS queue.

This task includes the following steps:

- "Define the global JMS settings" below
- "Prepare a message to send to the queue" below
- "Add a Send Message step to your test" below
- "Add a Send Message with security and attachments - optional" on the next page
- "Add a Receive Message step to your test - optional" on the next page
- "Browse the message queue - optional" on the next page

### 1. Define the global JMS settings

- a. In the canavs, select the **Start** or **End** step.
- b. In the Properties pane, open the **Test Settings**  tab and set the values for the properties in the JMS section.

**Note:** If necessary, you can link each of these properties to a test input variable or a test variable.

### 2. Prepare a message to send to the queue

Prepare the message you want to send to the queue in one of the following ways:

- Prepare an expression and type it into the **Message** property area.
- Use other activities to generate the string, for example **Concatenate String**, **Trim String**, and so forth.

### 3. Add a Send Message step to your test

- a. In the Toolbox pane, expand the **Standard Activities > JMS** node.
- b. Drag a **Send Message to JMS Queue** or a **Send and Receive Message from JMS Queue** activity onto the canvas.

- c. In the **Input/Checkpoints** tab  of the Properties pane, set the send-related input properties:
  - **Send queue name**
  - **Message**

**Note:** If you used another step to generate the message text, link its output to the **Message** property.

- **JMS send properties**

#### 4. **Add a Send Message with security and attachments - optional**

- a. If necessary, add a Web service step to the canvas.
- b. In the canvas, select the Web service step, and in the Properties pane, disable its **Send Request to Service** option.
- c. In the Security Settings tab, set the security options for the service.
- d. In the Attachments tab , add any attachments to send with the request.
- e. From the Toolbox pane, add a **Send Message to JMS Queue** activity to the canvas.
- f. In the Properties pane's Input/Checkpoints tab, select the link icon  for the **Message** property. The Select Link Source dialog box opens.
- g. In the Select Link Source dialog box, select the Web service step in the left pane. In the Output property section, double-click on the **Raw Request** property. This attaches the security and attachment data to the message.
- h. Define any other JMS properties as you normally would. For property details, see "JMS Activities" on page 472.

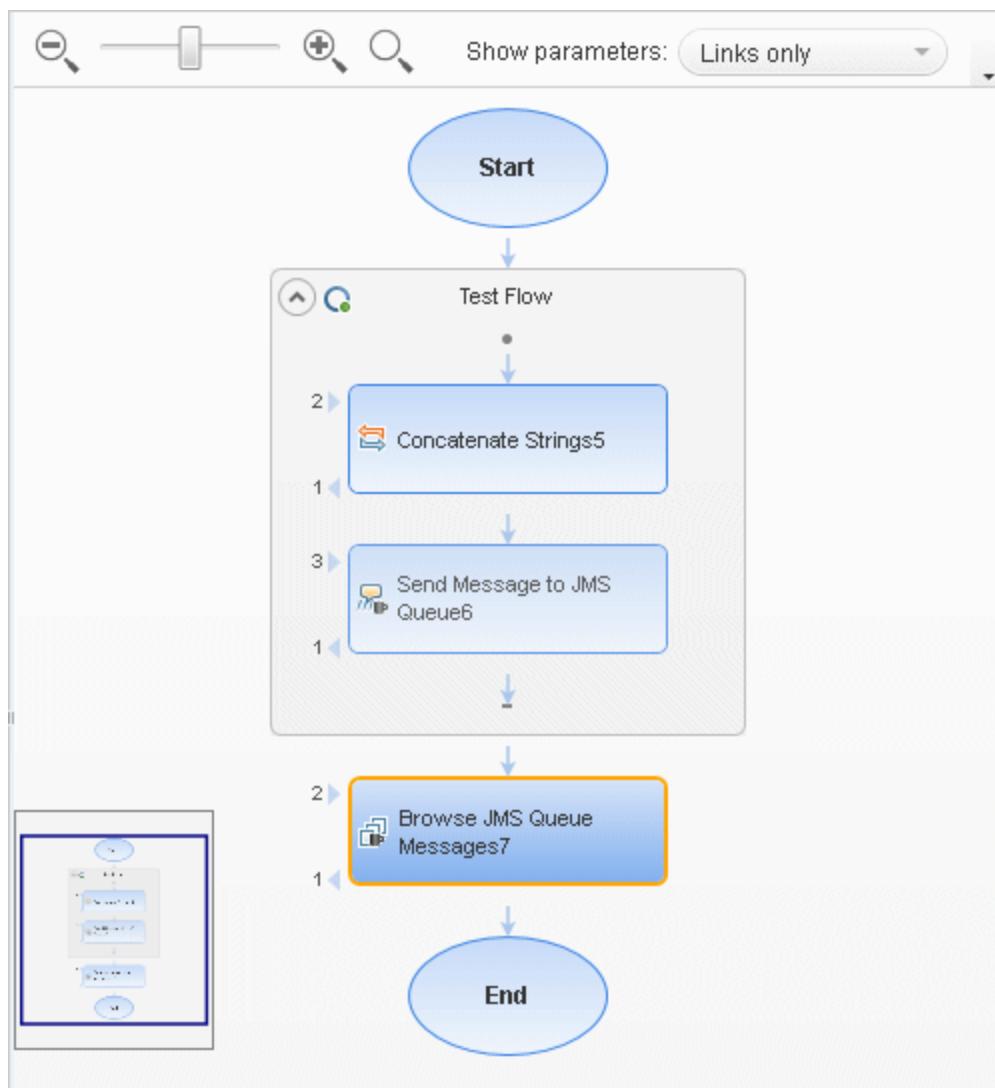
#### 5. **Add a Receive Message step to your test - optional**

- a. If necessary, to retrieve a message from the queue, drag a **Receive Message from JMS Queue** activity onto the canvas.
- b. In the canvas, select the **Receive Message from JMS Queue** step and then select the Properties pane's Input/Checkpoints tab .
- c. In the Input/Checkpoints tab, set the receive-related input properties:
  - **Receive queue name**
  - **JMS receive message selector**

#### 6. **Browse the message queue - optional**

- a. From the Toolbox pane, add a **Browse JMS Queue Messages** activity onto the canvas, outside

of the Test Flow. Make sure that a **Send (and Receive) Message from JMS Queue** step precedes the **Browse JMS Queue Messages** step.



- b. In the Properties pane, select the **Input/Checkpoints** tab and set the browse-related input properties:
- **Queue name.** Provide a queue name or link to the queue name from an earlier step. For IBM Websphere's MQ, if you specify a queue name that does not exist, a new queue will be created during the test run.
  - **JMS receive message selector.** The selector lets you filter the messages list. For details about these properties, see "[JMS Activities](#)" on page 472.

## 7. [Subscribe to a topic](#)

Add a **Subscribe to JMS Topic** step and provide values for its input properties as described in "[JMS Activities](#)" on page 472.

## 8. Publish a message to the topic

Add a **Publish Message to JMS Topic** step and provide values for its input properties.

## 9. Include Web service security and attachments - optional

To publish a Web service message via JMS with attachments and security setting, such as tokens and signatures:

- a. Drag in or select the Web service step in the canvas and disable its **Send Request to Service** option in the Properties pane.
- b. Set the security options for the service.
- c. Include any attachments.
- d. Drag a **Publish Message to JMS Topic** step onto the canvas.
- e. In the Properties pane's Input/Checkpoints tab, select the link icon  for the **Message** property. The Select Link Source dialog box opens.
- f. In the Select Link Source dialog box, select the Web service step in the left pane. In the Output property section, double-click on the **Raw Request** property. This attaches the security and attachment data to the message.
- g. Define any other JMS properties as you normally would. For property details, see "JMS Activities" on page 472.

## 10. Receive the message through the topic

Drag a **Receive Message from JMS Topic** activity onto the canvas. Use the topic and subscription name that you defined in the previous steps.

# How to Create an SAP API Test Step

## Relevant for: API testing only

This task describes how to create a test step for an SAP IDoc or RFC.

This task includes the following steps:

- "Prerequisite" below
- "Define an SAP Connection" on the next page
- "Select an iDoc from your SAP server." on the next page

### 1. Prerequisite

You must have the SAP .NET Connector installed on your machine. The installation is available on the SAP Help portal, at [http://help.sap.com/saphelp\\_NW04/helpdata/en/e9/23c80d66d08c4c8c044a3ea11ca90f/content.htm](http://help.sap.com/saphelp_NW04/helpdata/en/e9/23c80d66d08c4c8c044a3ea11ca90f/content.htm).

## 2. Define an SAP Connection

In the SAP Connections pane of the Options dialog box **Tools > Options > API Testing tab > SAP Connections** define one or more SAP connections.

## 3. Select an iDoc from your SAP server.

- a. Select **Tools > Import** from SAP.
- b. In the SAP dialog box, select one of the connections that you added in the SAP Connections pane of the Options dialog box.
- c. (Optional) Accept the default credentials entered in the SAP Connections pane or click **Override connection** to provide different credentials.
- d. Select **IDoc** or **RFC** and specify a search string using an asterisk (\*).
- e. Click **Search**.
- f. After the search is completed, select the necessary items and then click **Import Selected**.

These items are added in the Toolbox pane, to the Local Activities node, with a separate node for SAP. You can now add them to the test as individual steps.

# How to Call External Tests or Actions

## Relevant for: API testing only

This task describes how to incorporate tests from other HP applications. You can call any of the following types of tests:

- Unified Functional Testing tests (both GUI and API tests)
- QuickTest Professional tests
- Service Test tests
- VuGen (Virtual User Generator) scripts from HP LoadRunner

For details on integration between UFT API Testing and other HP products, see "[Automated Testing Tool Integration](#)" on page 391.

This task includes the following steps:

- "[Prerequisites](#)" on the next page
- "[Call an API Test or Action or Service Test test](#)" on the next page
- "[Call a GUI Test or QuickTest action or test](#)" on page 428
- "[Add a LoadRunner script activity](#)" on page 428

## 1. Prerequisites

Make sure you have installed the application whose test/script you want to call on the same computer with UFT or have access to the directory containing the tests or scripts.

### Notes:

- If you are calling a test last modified in a version of Service Test prior to your present version, make sure the test was modified with Service Test 11.10 or higher or UFT.
- To call a test created in QuickTest Professional, ensure that the test was created with QuickTest 11.00. The first time you run the step, you may need to wait several seconds for UFT to invoke and load the test.
- If you are calling a test created in HP LoadRunner, ensure that you created or opened the test in LoadRunner version 11.00 or later.

## 2. Call an API Test or Action or Service Test test

- a. Make sure the action or test you want to call has been saved and run successfully at least once.
- b. In the **Standard Activities** section of the Toolbox pane, expand the **HP Automated Testing Tools** node.
- c. Add a **Call API Action or Test** activity to the canvas.
- d. In the **Input/Output Properties** tab  in the Properties pane, click the **Select Action or Test** button.
- e. In the Select Action or Test Dialog Box, select a test last modified with Service Test 11.10 or higher, or with UFT.
- f. In the Input/Output Properties tab, edit the property values as needed.

### Note:

- The property list remains empty until you select a test.
- If the test you are calling has no input or output parameters, the Input/Output Properties tab will be empty.

- g. Add other relevant steps to your test. You can link input properties of subsequent step to the output properties of the step containing the called API test or action..
- h. If the value of the input parameter for step containing the API test/action call must be a string (such when the result of a previous step was XML), add an **XML to String** activity before the step containing the call to the action or test.
- i. Optional - To specify a custom directory for the results, in the General tab of the Properties, click the **Browse** button  in the **Results directory** row in the **General** view tab.

### 3. Call a GUI Test or QuickTest action or test

- a. Make sure the action or test you want to call has been saved and run successfully at least once.
- b. In the Toolbox pane, in the **Standard Activities** section, expand the **HP Automated Testing Tools** node.
- c. Drag the **Call GUI Action or Test** activity onto the canvas.

**Note:** This activity is only available when working with an HP Unified Functional Testing license.

**Tip:** Do not insert a call to a QuickTest or GUI action or test that contains a call to an API Test action or test, as this can cause unexpected behavior.

- d. In the **Input/Output Properties** tab  in the Properties pane, click the **Select Action or Test** button. In the Select Action or Test Dialog Box, select an action or a test.
- e. In the Select Action or Test Dialog Box, select an action or a test created in QuickTest 11.00 or UFT.

**Note:** If you want to use data from a GUI test in your API test, the GUI test or action that is called must have test or action parameters saved with the test or action.

- f. In the Input/Output Properties tab, edit the property values as needed. If you want to use parameters from the called GUI test, in the Input/Checkpoints tab, click the **Link to data source** button in subsequent test steps. In the Select Link Source dialog box, link the selected property to a GUI test or action parameter.

**Note:**

- The property list remains empty until you select a test.
- If the test you are calling has no input or output parameters, the Input/Output Properties tab will be empty.

### 4. Add a LoadRunner script activity

- a. Make sure the action or test you want to call has been saved and run successfully at least once.
- b. In the Toolbox pane, in the **Standard Activities** section, expand the **HP Automated Testing Tools** node.
- c. Add a **Call Virtual User Generator Script** activity to the canvas.
- d. In the Properties pane, select the General tab, and click the script selection button .
- e. Navigate to the directory where your VuGen script file (.usr) is saved.

# How to Prepare and Run a Load Test

## **Relevant for: API testing only**

This task describes how to prepare a test for load testing in LoadRunner.

This task includes the following steps:

- "Prerequisite" below
- "Create a load-enabled API test" below
- "Add test steps" below
- "Prepare for load testing - optional" below
- "How to Prepare and Run a Load Test" above
- "Set the data retrieval properties - optional" on the next page
- "Set the run configuration to Release" on the next page
- "Run in Load Testing mode to validate the test" on the next page
- "Incorporate the test into LoadRunner" on the next page

### **1. Prerequisite**

- Make sure that HP LoadRunner or a standalone version of VuGen (HP Virtual User Generator) is installed. Without this installation, the Load Testing template will not be available.
- If you are running the test from a Performance Center host, ensure that UFT is installed on the same machine as the Performance Center host.

### **2. Create a load-enabled API test**

In the New dialog box, in the **Select type** section, choose **API Load Test**.

If you have a test that was created with the standard API testing template, select **Design > Operation > Enable Test for Load Testing** or click the **Enable Test for Load Testing** button .

### **3. Add test steps**

Drag activities from the Toolbox pane onto the canvas to add steps to the test.

### **4. Prepare for load testing - optional**

To measure the performance of a group of steps, define a transaction.

- a. Mark the beginning of a transaction.
  - From the Toolbox pane, in the Load Testing node, add a **Start Transaction** activity to the canvas. Place it before the first step of the group of steps that you want to measure.
  - In the Properties, select the Input/Checkpoints tab  and enter a **Transaction name** in the **Input** section of the tab. This name will be used in LoadRunner Analysis.
- b. Mark the end of a transaction.
  - From the Load Testing node, add a **End Transaction** activity to the end of the group of steps you want to measure.
  - In the Input/Checkpoints tab in the Properties pane, type a transaction name. The name must be one that was already used for a prior **Start Transaction** step.
  - In the End Transaction's **Input** properties, select a **Status** for reporting: PASS, FAIL, AUTO, or STOP.

**Note:** The End Transaction status is only the LoadRunner transaction's status—not the status of step in UFT. For example, if you assign a **Failed** status to the transaction, UFT can still issue a **Passed** status for the test step.

- c. Set the think time.
  - If you want to emulate think time, add a **Load Testing > Think Time** activity between the relevant steps.
  - In the Input/Checkpoints tab, in the **Duration (sec)** row, specify a think time in seconds.

## 5. Set the data retrieval properties - optional

If you have data in the Data Pane, set the data retrieval properties. Click the **Link to a data source** button .

## 6. Set the run configuration to Release

In the General pane of the API Testing tab in the Options dialog (**Tools > Options > API Testing tab > General** node), in the **Run Sessions** options, select **Release**. The **Release** mode conserves resources, thus enhancing the load testing capabilities.

## 7. Run in Load Testing mode to validate the test

Expand the toolbar **Run** button and select **Run Test in Load Testing Mode**. This run is only for debugging purposes, to verify that the test is functional.

**Note:** When you run a test in Load Testing mode, the Output pane does not contain data and the run results do not open. To view the results, select **Run** to run the test in functional mode.

## 8. Incorporate the test into LoadRunner

Add the test to the LoadRunner Controller console to include it in a load test.

# How to Test Web Sockets Communication

Relevant for: API testing only

The following task explains how to test web socket communication in your application using the API test Web Socket activities.

This task includes the following:

- "Open a Web socket connection" below
- "Send a message to another Web socket" below
- "Receive a message from another Web socket" on the next page
- "Close the Web socket connection" on page 433

## Open a Web socket connection

In order to test the communication between Web sockets, you must first open a connection to the Web socket. This step is mandatory for testing the sending/receiving of messages from a Web socket connection.

1. Do one of the following:
  - From the **Web Sockets** section of the Toolbox pane, drag an **OpenSocket** activity to the Test Flow.
  - In the canvas, select an **OpenSocket** activity.
2. In the Input/Checkpoints tab  in the Properties pane, in the **Value** cell for the **URL** property, enter the URL for the Web socket connection.

## Send a message to another Web socket

1. Do one of the following:
  - From the **Web Sockets** section of the Toolbox pane, drag an **SendMessage** activity to the Test Flow.
  - In the canvas, select an **SendMessage** activity.
2. In the Input/Checkpoints tab  in the Properties pane, in the **Value** cell for the **SocketID** property, click the **Link to a data source** button . The Select Link Source dialog box opens.
3. In the Select Link Source dialog box, select the **Available steps** option.
4. In the list of available steps, select the **OpenSocket** activity. A list of available properties is displayed in the right pane.
5. In the right pane, select the **General** tab .

6. In the General tab, select the **SocketID** property and click **OK** to link the ReceiveMessage step to the OpenSocket step.

7. In the Properties pane, select the **HTTP** tab .

8. In the HTTP tab, in the **Request Body** section, from the drop-down list, select the format for your message body. You can send a message with **Text**, **XML**, or **JSON**.

9. In the Request Body section, in the Text Editor area, enter the body of your message to send.

**Note:** You can load the XML or JSON for the sent message body from an external file by click the **Load** button in the text editor area.

10. In the Toolbox pane, expand the **Flow Control** activities section.

11. From the Flow Control activities, drag a **Wait** activity to the canvas. The Input/Checkpoints tab  opens in the Properties pane.

12. In the Input/Checkpoints tab, in the **Value** cell for the Completion event property, click the **Link to a data source** button . The Select Link Source dialog box opens.

13. In the Select Link Source dialog box, select the **Available steps** option. A list of available steps is displayed in the **Select a step:** (left) pane.

14. In the Select a step: pane, select the **ReceiveMessage** activity. A list of available properties is displayed in the **Select a property:** (right) pane.

15. In the Select a property pane, select the **General** tab .

16. In the General tab, select the **Completion event name** property and click **OK**. UFT links the ReceiveMessage step to the Wait step, instructing the test to wait to proceed until the message is received from the Web socket in the ReceiveMessage step.

## Receive a message from another Web socket

1. Prerequisite: Create an **OpenSocket** step in your test.

2. Do one of the following:

- From the **Web Sockets** section of the Toolbox pane, drag an **ReceiveMessage** activity to the Test Flow.
- In the canvas, select an **ReceiveMessage** activity.

3. In the Input/Checkpoints tab  in the Properties pane, in the **Value** cell for the **SocketID** property, click the **Link to a data source** button . The Select Link Source dialog box opens.

4. In the Select Link Source dialog box, select the **Available steps** option.

5. In the list of available steps, select the **OpenSocket** activity. A list of available properties is displayed in the right pane.

6. In the right pane, select the **General** tab .

7. In the General tab, select the **SocketID** property and click **OK** to link the SendMessage step to the OpenSocket step.
8. In the Properties pane, select the **HTTP** tab .
9. In the HTTP tab, in the **Received Message Body** section, from the drop-down list, select the format for your message text. You can receive a message body with **Text**, **XML**, or **JSON**.
10. In the Received Message Body section, in the Text Editor area or the regular expression grid area, enter the body of the expected message or a regular expression representing the received message body.

**Note:** You can load the XML or JSON for the received message body from an external file by click the **Load** button in the text editor area.

## Close the Web socket connection

**Note:** This step is optional. You should use this step if you want to send or receive messages from a different Web socket in later test steps.

1. Prerequisite: Create an **OpenSocket** step in your test.
2. Do one of the following:
  - From the **Web Sockets** section of the Toolbox pane, drag an **CloseSocket** activity to the Test Flow.
  - In the canvas, select an **CloseSocket** activity.
3. In the Input/Checkpoints tab  in the Properties pane, in the **Value** cell for the **SocketID** property, click the **Link to a data source** button . The Select Link Source dialog box opens.
4. In the Select Link Source dialog box, select the **Available steps** option.
5. In the list of available steps, select the **OpenSocket** activity. A list of available properties is displayed in the right pane.
6. In the right pane, select the **General** tab .
7. In the General tab, select the **SocketID** property and click **OK** to link the **CloseSocket** step to the **OpenSocket** step.
8. (Optional)- In the **Checkpoints** section of the Input/Checkpoints tab, select the **Validate** checkbox in the **Result** row to set a checkpoint to check if the Close operation succeeds.

# Standard Activity Types

## **Relevant for: API testing only**

This section describes the Standard activity groups:

• <a href="#">Flow Control Activities</a> .....	437
• <a href="#">Condition Activity</a> .....	437
• <a href="#">Loop Activity</a> .....	438
• <a href="#">Wait Activity</a> .....	439
• <a href="#">Miscellaneous Activities</a> .....	439
• <a href="#">Run Program Activity</a> .....	439
• <a href="#">End Program Activity</a> .....	440
• <a href="#">Report Message Activity</a> .....	441
• <a href="#">String Manipulation Activities</a> .....	442
• <a href="#">Replace String Activity</a> .....	442
• <a href="#">Substring Activity</a> .....	443
• <a href="#">Trim String Activity</a> .....	443
• <a href="#">System Activities</a> .....	444
• <a href="#">Math Activities</a> .....	444
• <a href="#">Date and Time Activities</a> .....	444
• <a href="#">Increment Date Activity</a> .....	445
• <a href="#">Date/Time Difference Activity</a> .....	445
• <a href="#">Format Date/Time Activity</a> .....	446
• <a href="#">File System Activities</a> .....	447
• <a href="#">File Copy Activity</a> .....	447
• <a href="#">File Move Activity</a> .....	448
• <a href="#">Create File Activity</a> .....	449
• <a href="#">File Rename Activity</a> .....	449
• <a href="#">Write to File Activity</a> .....	450
• <a href="#">Read From File Activity</a> .....	451
• <a href="#">Get Folder Contents Activity</a> .....	451
• <a href="#">Database Activities</a> .....	452
• <a href="#">Open Connection/Close Connection Activity</a> .....	453

• Select Data Activity .....	454
• Execute Command Activity .....	455
• Begin Transaction Activity .....	455
• Commit Transaction/Rollback Transaction Activity .....	456
• FTP Activities .....	457
• FTP Download Activity .....	457
• FTP Upload Activity .....	459
• FTP Rename Activity .....	460
• FTP File Delete Activity .....	461
• FTP File Get Size Activity .....	462
• FTP Directory Delete Activity .....	462
• FTP Directory Create Activity .....	463
• FTP Directory Get Content Activity .....	464
• Network Activities .....	465
• HTTP Request Activity .....	466
• HTTP Receiver Activity .....	468
• SOAP Request Activity .....	469
• JSON Activities .....	470
• String to JSON Activity .....	471
• JSON to String Activity .....	471
• Java Activities .....	471
• JMS Activities .....	472
• Publish Message to JMS Topic Activity .....	472
• Receive Message from JMS Queue Activity .....	473
• Receive Message from JMS Topic Activity .....	474
• Send Message to JMS Queue Activity .....	475
• Send/Receive Messages from JMS Queue Activity .....	475
• Subscribe to JMS Topic Activity .....	477
• Browse JMS Queue Activity .....	477
• Load Testing Activities .....	478
• IBM WebSphere MQ Activities .....	479
• Connect to MQ Queue Manager Activity .....	480
• Disconnect from MQ Queue Manager Activity .....	481

• Commit/Backout Pending MQ Messages Activity .....	481
• Browse Messages in MQ Queue Activity .....	482
• Put Message to MQ Queue Activity .....	484
• Get Message from MQ Queue Activity .....	485
• Put and Get Message from MQ Queue .....	486
• Subscribe to MQ Topic Activity .....	488
• Unsubscribe from MQ Topic Activity .....	489
• Publish Message to MQ Topic Activity .....	489
• Receive Message from MQ Topic Activity .....	490
• HP Automated Testing Tools Activities .....	492
• SAP Activities .....	492
• XML Activities .....	494
• Transform XML Activity .....	495
• Compare XML Activity .....	495
• XML to String Activity .....	496
• String to XML Activity .....	496
• Validate XML Activity .....	497
• Web Sockets Activities .....	497
• Open Socket Activity .....	497
• Close Socket Activity .....	498
• Send Message Activity .....	498
• Receive Message Activity .....	499

## Flow Control Activities

### Relevant for: API testing only

The following activities allow you to control the flow of the test using loops, conditions, breaks, and delays:

- **Condition.** Enables you to use a branching mechanism to determine alternate test flows based on the value of a property. For details, see "[Condition Activity](#)" below.
- **Loop.** A loop frame whose properties can be set independently of the Test Flow. The loop types are **For loop**, **Do While loop**, and **For Each loop**. For details, see "[Loop Activity](#)" on the next page.
- **Break.** Halts the test execution.
- **Continue.** Resumes the test execution after a break.
- **Sleep.** Pauses the test execution for a specified amount of time.
- **Wait.** Waits a specified amount of time for a trigger event before releasing a step for execution. For details, see "[Wait Activity](#)" on page 439.

## Condition Activity

### Relevant for: API testing only

This activity enables you to branch your test flow depending on a conditional outcome.

The following properties are available for this activity:

Property	Description
<b>Use condition</b>	<p>Instructs UFT to use a conditional outcome to proceed with the test flow.</p> <p>If the current state of the application matches the defined condition, then the condition proceeds with the test steps defined in the <b>Yes</b> branch. If the current state of the application does not match the defined condition, the test proceeds with the steps in the <b>No</b> branch.</p> <p>You define the following settings for the condition activity:</p> <ul style="list-style-type: none"><li>• <b>Variable:</b> the process for which you want to determine the value</li><li>• <b>Operator:</b> the basis for comparison for the variable. You can select one of the following:<ul style="list-style-type: none"><li>• =</li><li>• !=</li><li>• &gt;</li><li>• &gt;=</li><li>• &lt;</li><li>• &lt;=</li></ul></li></ul>

	<ul style="list-style-type: none"> <li><b>Value:</b> the expected value for the variable</li> </ul> <p>You can link the variable and value to data sources or other test steps. For details, see "<a href="#">How to Assign Data to API Test/Component Steps</a>" on page 551.</p>
<b>Use event</b>	<p>Instructs UFT to use the event handler specified in the <b>Events</b> tab  to determine on which conditional branch to continue to run the test.</p> <p><b>Note:</b> You can create custom properties to use in the event handler by clicking the <b>Add</b> button .</p>

## Loop Activity

### Relevant for: API testing only

This activity enables you to use additional test flow loops independently of the test flow. This also enables you to set individual data flow patterns for each loop independently of the main **Test Flow** data usage properties.

The following iteration settings are available for the Test Flow or other custom loops:

Property	Description
<b>Do While Loop - Use condition</b>	<p>Repeats the loop as long as the specified condition is met:</p> <ul style="list-style-type: none"> <li><b>Variable.</b> The variable to evaluate. Use the Select Link Source button  to enter a data source expression.</li> <li><b>Operator.</b> A comparison operator relevant to the variable such as <b>=</b>, <b>!=</b>, <b>Contains</b>, <b>Starts</b> and <b>Regex</b>.</li> <li><b>Value.</b> The value with which to compare the result.</li> </ul>
<b>Do While Loop - Use event</b>	<p>Performs iterations until the event returns <b>True</b>. This mode is useful for complex conditions that can be defined in an event handler. When you select this option, the Input Properties grid opens.</p> <ul style="list-style-type: none"> <li>Click  to define properties.</li> <li>Click  to open the <b>Events</b> view. Click <b>Create a default handler</b> in the <b>Handler</b> column.</li> <li>Modify or add code to the event handler method that defines your condition.</li> </ul>
<b>For Each Loop</b>	<p>Performs an iteration for each element in the associated array or collection of objects. Data is selected using the <b>Select Link Source</b> button .</p> <p><b>Note:</b> When you delete data from a data table, it continues to run an iteration for that row, with empty values. To remove an entire row of a data, click the row and select <b>Delete</b> from the shortcut menu (only with Excel installations).</p>
<b>For Loop</b>	Performs the Test Flow or custom loop the number of times that you specify in the <b>Number of Iterations</b> box.

## Wait Activity

### Relevant for: API testing only

This activity enables you to temporarily pause the test flow for a specified amount of time.

When running asynchronous service calls as part of your test, the **Wait** activity waits a specified amount of time for a trigger event before releasing the associated step. This activity is used in conjunction with the **HTTP Receiver** steps and Web Service calls imported as server response steps. For details, see ["Asynchronous Service Calls" on page 597](#).

The following properties are available:

Property	Description
<b>Timeout</b>	The timeout in milliseconds. Negative values indicate that there is no timeout.
<b>Start timeout after step</b>	The time after step execution from when to begin measuring the timeout interval.
<b>Action on timeout</b>	The action to take when the timeout is reached without the completion events: <b>Fail</b> or <b>Continue</b> .
<b>Completion event names</b>	A list of the completion events that were configured in prior receiver steps.

## Miscellaneous Activities

### Relevant for: API testing only

This activity group includes the following activities:

- **Custom Code.** Enables you to program a step to execute your own activity. Use the **Add** button to define new input or output properties.
- **Set Test Variable.** Defines a global variable for your test.

**Note:** This adds an additional test variable to the built-in variables included in every test.

- **Run Program.** Invokes an application on the current machine. For details, see ["Run Program Activity" below](#).
- **End Program.** Closes an application that is running. For details, see ["End Program Activity" on the next page](#).
- **Report Message.** Sends a custom message to the Run Results and/or the Output pane. For details, see ["Report Message Activity" on page 441](#).

## Run Program Activity

### Relevant for API testing only

This activity enables you to run a program from your computer as part of the test.

The following properties are available for this activity:

### Input Properties

Property	Description
<b>Target</b>	The name of the program to run.  <b>Tip:</b> You can click the <b>Browse for file</b> button  to navigate to the file.
<b>Command line arguments</b>	The arguments to enter in the Windows command line to run the program selected for the <b>Target</b> property.
<b>Working directory</b>	The file system location of the program defined in the <b>Target</b> property.  <b>Note:</b> If you do not remember the exact path to the Target program, click the <b>Browse for file</b> button  to find the correct directory.
<b>Run as user</b>	The user with which to run the program specified in the <b>Target</b> property.
<b>Password</b>	The password of the user specified in the <b>Run as user</b> property.
<b>Domain</b>	The password to use when starting the application.
<b>Timeout</b>	The amount of time (in milliseconds) to wait for the application to exit.

### Checkpoint Properties

Property	Description
<b>Exit code</b>	The value returned by the program specified in the <b>Target</b> property when the program closed.  <b>Tip:</b> You can choose an operator to expand the possible values if the output of the program varies. Select the appropriate operator from the dropdown menu.
<b>Process terminated</b>	Specifies whether the program completed its run correctly or timed out. Select the appropriate value from the drop-down menu in the <b>Value</b> column.

## End Program Activity

### Relevant for API testing only

This activity enables you to stop and close a program that is running as part of your test. You should use this step in conjunction with a **Run Program** activity.

The following properties are available for this activity:

## Input Properties

When entering the input properties, you can choose between ending the program by specifying the path to the program, by specifying the window to close, or by specifying the process ID of the program.

**Note:** You can only choose one of the options to close the program.

Property	Description
<b>Executable path</b>	The path to file to close.  <b>Note:</b> You can click the <b>Browse for file</b> button  to navigate the file.
<b>Window Title</b>	Enables you to specify a specific program window to close. In order to close the window, you must define the following properties: <ul style="list-style-type: none"><li>• <b>Title:</b> The window title of the program to close</li><li>• <b>Use RegEx:</b> Instructs UFT to treat the window title as a regular expression. Select <b>true</b> or <b>false</b> from the drop-down list to enable or disable this attribute.</li></ul>
<b>Process ID</b>	The Windows process ID of the program to close.

## Checkpoint Properties

Property	Description
<b>Exit Code</b>	The value provided by the program upon its completion.

## Report Message Activity

### Relevant for: API testing only

This activity enables you to send a message to the Output pane during a test run or to the run results.

The following properties are available for this activity:

Property	Description
<b>Status</b>	The status to report in the message. By default, UFT includes statuses of Done, Pass, and Fail. To define other statuses, click the <b>Link to data source</b> button in the <b>Value</b> cell.  <b>Note:</b> This status is different from the run status of the step displayed for each test step in the step's captured data in the run results.
<b>Message</b>	The message to report. You can manually enter the message or link this message to a data source or other test steps. For details on linking this property value, see " <a href="#">How to Assign Data to API Test/Component Steps</a> " on <a href="#">page 551</a> .

<b>Destination</b>	The place to report the message. You can report this message in the run results only or in the run results and the Output pane (displayed during test runs).
--------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------

The message generated during the test run of this step, including the **Status** and **Message** property values, is displayed in the step's captured data in the run results.

## String Manipulation Activities

### Relevant for: API testing only

This activity group provides access to the following string related activities:

- **Concatenate Strings.** Joins two strings.
- **Replace String.** Replaces part of a string with an alternate string. You can provide an array of replacement strings for use with iterations. For details, see "[Replace String Activity](#)" below.
- **Substring.** Extracts the characters between two specified locations of a string. For details, see "[Substring Activity](#)" on the next page.
- **Trim String.** Removes whitespace characters from the beginning or end of a string. For details, see "[Trim String Activity](#)" on the next page.

## Replace String Activity

### Relevant for: API testing only

This activity enables you to replace a specified string with another string.

The following properties are available for this activity:

### Input Properties

Property	Description
<b>Source string</b>	The name of the string to replace.
<b>Search and replace array</b>	<p>The array containing the strings to find and replace. You must provide the values for the following:</p> <ul style="list-style-type: none"> <li>• <b>Search string:</b> The string to replace.</li> <li>• <b>Replace String:</b> The string which replaces the string found in the <b>Search String</b> property.</li> </ul> <p><b>Tip:</b> If you need to replace multiple parts of the source string, add an additional array by clicking the <b>Add</b> button  in the <b>Search and Replace (array)</b> property row.</p>
<b>Case-sensitive</b>	Indicates that UFT should search for the <b>Search String</b> property by locating the matching case for the string specified in the Search String value cell. Select True or False from the drop-down list to set this property.

## Checkpoint Properties

Property	Description
<b>Result</b>	<p>The expected result for the search and replace operation.</p> <p><b>Note:</b> Make sure to include spaces where needed in both the <b>Search</b> and <b>Replace</b> strings as well as the <b>Result</b> output string or the checkpoint will fail.</p>

## Substring Activity

### Relevant for: API testing only

This activity enables you to extract a section of a specified string for use elsewhere in your test.

The following properties are available for this activity:

## Input Properties

Property	Description
<b>Source string</b>	The string from which you are extracting a section.
<b>Substring start</b>	The character position (numerically) in the string defined in the <b>Source string</b> property to begin extracting.
<b>Substring length</b>	The number of characters to extract from the string specified in the <b>Source string</b> property.

## Checkpoint Properties

Property	Description
<b>Result</b>	The expected string that is extracted.

## Trim String Activity

### Relevant for: API testing only

This activity enables you to trim extra whitespaces from a selected string.

The following properties are available for this activity:

## Input Properties

Property	Description
<b>Source string</b>	The string from which to trim.

<b>Trim start</b>	Enables you to trim the whitespace from the beginning of the string. Choose True or False from the drop-down list to enable or disable trimming.
<b>Trim end</b>	Enables you to trim the whitespace from the end of the string. Choose True or False from the drop-down list to enable or disable trimming.

## Checkpoint Properties

Property	Description
<b>Result</b>	The result of the trim operation.  <b>Note:</b> Make sure to include spaces where needed in both the <b>Search</b> and <b>Replace</b> strings as well as the <b>Result</b> output string or the checkpoint will fail.

## System Activities

### Relevant for: API testing only

This activity group provides access to the following operating system activities:

- **Set OS Environment Variable.** Sets the value of an operating system variable.
- **Get OS Environment Variable.** Retrieves the value of an operating system variable.

## Math Activities

### Relevant for: API testing only

This activity group performs the following math related activities:

- **Add.** Adds two numbers.
- **Subtract.** Subtracts one number from another.
- **Multiply.** Multiplies two numbers.
- **Divide.** Divides one number by another.

## Date and Time Activities

### Relevant for: API testing only

This activity group performs the following date and time related activities:

- **Increment Date.** Adds date or time units to a date/time string. For details, see "["Increment Date Activity" on the next page](#).
- **Date/Time Difference.** Calculates the difference between two date/time strings. For details, see "["Date/Time Difference Activity" on the next page](#).

- **Format Date/Time.** Formats a date/time string. For details, see "Format Date/Time Activity" on the next page.

## Increment Date Activity

### Relevant for: API testing only

This activity enables you to change a date and time by a measured amount.

The following properties are available for this activity:

### Input Properties

Property	Description
<b>Original Date/Time</b>	The source date and time to modify.  <b>Tip:</b> You can manually enter the date and time or choose a date and time from the drop-down calendar.
<b>Original Date/Time Format</b>	The format of the date and time provided in the Original Date/Time property. Choose one of the formats from the drop-down list.  <b>Note:</b> You only need to enter this information when you provide a string expression for the <b>Original Date/Time</b> property.
<b>Units</b>	The unit to add to the date and time specified in the <b>Original Date/Time</b> property.  <b>Note:</b> You can only increment one unit at a time in a test step. If you need to increment multiple date and time units, add additional <b>Increment Date</b> activities to your test.
<b>Units</b>	The number of units (as specified in the <b>Units</b> property value) to add to the <b>Original Date/Time</b> property.

### Checkpoint Properties

Property	Description
<b>Result</b>	The new date and time after the addition is made.  <b>Note:</b> Make sure to enter the date and time with exactly the same format as specified in the <b>Original Date/Time Format</b> property.

## Date/Time Difference Activity

### Relevant for: API testing only

This activity enables you compare different dates and times.

The following properties are available for this activity:

### Input Properties

Property	Description
<b>Date/Time (A,B)</b>	<p>The date and times to compare.</p> <p><b>Tip:</b> You can manually enter the date and time or choose a date and time from the drop-down calendar.</p>
<b>Date/Time Format (A,B)</b>	<p>The format of the date and time provided in the <b>Date/Time</b> property. Choose one of the formats from the drop-down list.</p> <p><b>Note:</b> You only need to enter this information when you provide a string expression for the <b>Date/Time</b> property.</p>
<b>Units</b>	<p>The unit to compare between the date and times specified in the <b>Date/Time</b> properties.</p> <p><b>Note:</b> You can only increment one unit at a time in a test step. If you need to increment multiple date and time units, add additional <b>Increment Date</b> activities to your test.</p>

### Checkpoint Properties

Property	Description
<b>Result</b>	The numeric difference between the compared dates specified in the <b>Date/Time</b> properties.

## Format Date/Time Activity

### Relevant for: API testing only

This activity enables you to format a selected date and time with a given date and time format.

The following properties are available for this activity:

### Input Properties

Property	Description
<b>Input Date/Time</b>	<p>The date and time to format.</p> <p><b>Tip:</b> You can manually enter the date and time or choose a date and time from the drop-down calendar.</p>
<b>Input Date/Time Format</b>	<p>The format of the date and time specified in the Input Date/Time property.</p> <p><b>Note:</b> You only need to enter this information when you provide a string expression for the <b>Input Date/Time</b> property.</p>

<b>Format</b>	The format to which to change the date and time specified in the <b>Input Date/Time</b> property.
---------------	---------------------------------------------------------------------------------------------------

## Checkpoint Properties

Property	Description
<b>Result</b>	<p>The date and time after the new format is applied.</p> <p><b>Note:</b> Make sure to enter the date and time with exactly the same format as specified in the <b>Format</b> property.</p>

## File System Activities

### Relevant for: API testing only

This activity group provides access to the following file system related activities:

- **File Exists.** Searches for a specific file, or group of files, if you use a wildcard expression. It returns true if the file exists.
- **Folder Exists.** Searches for a specific folder. It returns true if the folder exists.
- **File Copy.** Copies a file to a new destination. For details, see "[File Copy Activity](#)" below.
- **File Move.** Moves a file to another destination. For details, see "[File Move Activity](#)" on the next page.
- **File Create.** Creates a new file. For details, see "[Create File Activity](#)" on page 449.
- **File Delete.** Deletes a file.
- **File Rename.** Renames a file. For details, see "[File Rename Activity](#)" on page 449.
- **Write to File.** Writes to an existing or new file. For details, see "[Write to File Activity](#)" on page 450.
- **Read from File.** Retrieves text from a file. For details, see "[Read From File Activity](#)" on page 451.
- **Get Folder Contents.** Gets the contents of a folder—file names, folder names, or both. For details, see "[Get Folder Contents Activity](#)" on page 451.

## File Copy Activity

### Relevant for: API testing only

This activity enables you to copy a selected file to another specified location.

The following properties are available for this activity:

### Input Properties

Property	Description
<b>File source path</b>	The path of the file to copy.

	<b>Tip:</b> You can click the <b>Browse for file</b> button  to navigate to the file.
<b>Target folder</b>	The folder into which to copy the file specified in the File source path property.  <b>Tip:</b> If you do not remember the exact path in which you want to copy the file, you can click the <b>Browse for folder</b> button  to find the directory.
<b>Target file name</b>	The name of the file after it is copied.
<b>Overwrite</b>	Instructs UFT to overwrite or not overwrite an already existing file of the same name. Select True or False from the drop-down list to enable or disable the setting.

## Checkpoint Properties

Property	Description
<b>Result</b>	Indicates whether the file copy succeeded or failed.

## File Move Activity

### Relevant for: API testing only

This activity enables you to move a file to a selected location.

The following properties are available for this activity:

## Input Properties

Property	Description
<b>File source path</b>	The path of the file to copy.  <b>Tip:</b> You can click the <b>Browse for file</b> button  to navigate to the file.
<b>Target folder</b>	The folder into which to copy the file specified in the File source path property.  <b>Tip:</b> If you do not remember the exact path in which you want to copy the file, you can click the <b>Browse for folder</b> button  to find the directory.
<b>Target file name</b>	The name of the file after it is copied.
<b>Overwrite</b>	Instructs UFT to overwrite or not overwrite an already existing file of the same name. Select True or False from the drop-down list to enable or disable the setting.

## Checkpoint Properties

Property	Description
<b>Result</b>	Indicates whether the file copy succeeded or failed.

## Create File Activity

### Relevant for: API testing only

This activity enables you to create a new file.

**Note:** This activity creates a new file but does not put content into the file. If you need to also add content to the file, use a **Write to File** activity in conjunction with this activity.

The following properties are available for this activity:

## Input Properties

Property	Description
<b>Target folder</b>	The folder into which to copy the file specified in the File source path property.  <b>Tip:</b> If you do not remember the exact path in which you want to copy the file, you can click the <b>Browse for folder</b> button  to find the directory.
<b>Target file name</b>	The name of the file after it is copied.
<b>Overwrite</b>	Instructs UFT to overwrite or not overwrite an already existing file of the same name. Select True or False from the drop-down list to enable or disable the setting.

## Checkpoint Properties

Property	Description
<b>Result</b>	Indicates whether the file copy succeeded or failed.

## File Rename Activity

### Relevant for: API testing only

This activity enables you to rename a selected file.

The following properties are available for this activity:

### Input Properties

Property	Description
<b>File source path</b>	The path of the file to copy.  <b>Tip:</b> You can click the <b>Browse for file</b> button  to navigate to the file.
<b>Target file name</b>	The name of the file after it is copied.
<b>Overwrite</b>	Instructs UFT to overwrite or not overwrite an already existing file of the same name. Select True or False from the drop-down list to enable or disable the setting.

### Checkpoint Properties

Property	Description
<b>Result</b>	Indicates whether the file copy succeeded or failed.

## Write to File Activity

### Relevant for: API testing only

This activity enables you to add content to a new or existing file.

You can use this activity in conjunction with the **Create File** activity to add content to a created file.

The following properties are available for this activity:

### Input Properties

Property	Description
<b>Content</b>	The information to add to the file specified in the <b>File path</b> property.  If you have a lot of content to add to the file, click the down arrow to display the extended
<b>File path</b>	The directory containing the file, including the file name.
<b>Mode</b>	The manner of editing the specified file. You can select one of the following: <ul style="list-style-type: none"> <li>• <b>Append to existing file:</b> Adds content to a file that already exists.</li> <li>• <b>Create new file:</b> Creates a new file with the name specified in the <b>File path</b> property and with the content indicated in the <b>Content</b> property.</li> </ul>
<b>Encoding</b>	The manner of encoding the content added in the file. Select one of the following modes of encoding: <ul style="list-style-type: none"> <li>• ASCII</li> <li>• UTF-7</li> <li>• UTF-8</li> </ul>

	<ul style="list-style-type: none"> <li>• UTF-32</li> <li>• Unicode</li> </ul>
<b>Append new line</b>	Instructs UFT to add a new line for your content to the file selected (if you are writing to an existing file). Select True or False to enable or disable the setting.

## Checkpoint Properties

Property	Description
<b>Result</b>	Indicates whether the file copy succeeded or failed.

## Read From File Activity

### Relevant for: API testing only

This activity enables you to check and read the content in a specific file.

The following properties are available for this activity:

## Input Properties

Property	Description
<b>File path</b>	The directory containing the file, including the file name.
<b>Encoding</b>	The manner of encoding the content added in the file. Select one of the following modes of encoding: <ul style="list-style-type: none"> <li>• ASCII</li> <li>• UTF-7</li> <li>• UTF-8</li> <li>• UTF-32</li> <li>• Unicode</li> </ul>

## Checkpoint Properties

Property	Description
<b>Content</b>	The content read from the file.

## Get Folder Contents Activity

### Relevant for: API testing only

This activity enables you to check the contents of a specified folder.

The following properties are available for this activity:

## Input Properties

Property	Description
<b>Folder source path</b>	The folder in which to search.
<b>Search pattern</b>	The criteria by which to search the folder's content. By default, UFT searches for any file having an extension (indicated by the *.* criteria).
<b>Content type</b>	The type of content for which to search. You can choose to search only files, only folders, or both files and folders.
<b>Sort by</b>	Sorts the folder content results. You can sort by a number of criteria: <ul style="list-style-type: none"> <li>• <b>Name</b></li> <li>• <b>Creation Time</b> (of the file/folder)</li> <li>• <b>Date Modified</b> (of the file/folder)</li> <li>• <b>None</b></li> </ul>
<b>Sort type</b>	Displays the results in ascending or descending order.
<b>Include subfolders</b>	Searches the subfolders included in the folder specified in the <b>Folder source path</b> property.

## Checkpoint Properties

The checkpoint properties for this activity are displayed as an array, To add an array (or multiple arrays if needed), click the **Add Array** button .

Property	Description
<b>Name</b>	The name of a file expected to be found.
<b>Path</b>	The directory of the expected result.
<b>Full path</b>	The full path to the directory of the expected result.
<b>Content type</b>	The expected content of the search.
<b>Creation time</b>	The creation time of the expected result.
<b>Date modified</b>	The modification date of the expected result.

## Database Activities

### Relevant for: API testing only

This activity group performs the following database related activities:

- **Open Connection.** Opens a connection to a database using the specified connection string. For details, see "["Open Connection/Close Connection Activity" on the next page.](#)

- **Close Connection.** Closes an open database connection. For details, see "[Open Connection/Close Connection Activity](#)" below.
- **Select Data.** Executes a SELECT type SQL statement that retrieves data. For details, see "[Select Data Activity](#)" on the next page.
- **Execute Command.** Executes an SQL statement that does not retrieve data from the database, such as UPDATE, DELETE, and so forth. For details, see "[Execute Command Activity](#)" on page 455.
- **Begin Transaction.** Begins a database transaction. For details, see "[Begin Transaction Activity](#)" on page 455.
- **Commit Transaction.** Commits a database transaction. For details, see "[Commit Transaction/Rollback Transaction Activity](#)" on page 456.
- **Rollback Transaction.** Rolls back a database transaction from a pending state. For details, see "[Commit Transaction/Rollback Transaction Activity](#)" on page 456.

## Open Connection/Close Connection Activity

### **Relevant for: API testing only**

This activity enables you to open or close a connection to a database. You should use both of these activities together.

The **Open Connection** activity is required before other database connection activities.

The following properties are available for these activities:

### **Input Properties**

Property	Description
<b>Connection String</b> (for Open Connection activities only)	An OLE DB database connection string. You can provide a value in one of the following ways: <ul style="list-style-type: none"> <li>• Type it in manually.</li> <li>• Open the <b>Connection Builder</b> </li> <li>• Use the <b>Link to a data source</b> button  to create a link to the string.</li> </ul>
<b>Connection</b> (for Close Connection activities)	A link to the output of an <b>Open Connection</b> step. Use the <b>Link to a data source</b> button  to create a link.

### **Checkpoint Properties**

Property	Description
<b>Result</b>	Indicates whether the connection is active (true) or closed (false).
<b>Result Message</b>	A message indicating success or an informational message from the database.

## Select Data Activity

### Relevant for: API testing only

This activity enables you to specify the data from a database user source.

You must put an **Open Connection** step in your test before this activity.

The following properties are available for this activity:

### Input Properties

Property	Description
<b>Connection</b>	A link to the output of a previously opened connection string. Use the <b>Link to a data source</b> button  to create the link.
<b>Query string</b>	An SQL SELECT statement. You can provide a string in one of the following ways: <ul style="list-style-type: none"> <li>Type it in manually.</li> <li>Open the <b>Query Builder</b> .</li> <li>Use the <b>Link to a data source</b> button  to create a link to the string.</li> </ul>
<b>Timeout</b>	The maximum time to allow for executing the database command. A value of zero indicates no time limit. <b>Default:</b> 30 seconds

### Checkpoint Properties

Property	Description
<b>Count</b>	The number of rows returned by the command.
<b>Result</b>	Indicates whether the operation has been successfully completed: <code>true</code> or <code>false</code> .
<b>Result table</b>	An array of returned rows. The row elements are column names. You can set an expected value for each column of each row: <ul style="list-style-type: none"> <li>The columns changed due to parameterization.</li> <li>A column was removed since the original query.</li> <li>You don't have access to database when designing the test.</li> </ul> <p><b>Note:</b> In order to see the table column, you need to generate the output at least once. You can do this by selecting the <b>Generate output</b> option in the Query Builder, or by clicking the <b>Generate Output</b> button in the <b>Checkpoints</b> pane.</p> <p><b>Tip:</b> Use the <b>Manage Columns</b> button to remove or replace columns when:</p>

## Database Property Options

Option	Description
<b>Generate Output</b>	Retrieves table data from the database and constructs an array with the current table structure.
<b>Manage Columns</b>	If you enabled <b>Generate output</b> in the Query Builder, you do not need to generate the output again.
<b>Manage Columns</b>	If the table structure changed since you generated the output, the Manage Columns dialog box lets you manage the columns. This is common when columns names were a parameter, or if a column was deleted from the table.

## Execute Command Activity

### Relevant for: API testing only

This activity enables you to execute a command for your database data.

You must put an **Open Connection** step in your test before this activity.

The following properties are available for this activity:

### Input Properties

Property	Description
<b>Connection</b>	A link to the output of a previously opened connection string. Use the <b>Link to a data source</b> button  to create the link.
<b>Command</b>	An SQL statement that does not retrieve data, such as UPDATE or DELETE. You can provide a string in one of the following ways: <ul style="list-style-type: none"> <li>Type it in manually in the drop-down edit box.</li> <li>Use the <b>Link to a data source</b> button  to create a link to the string.</li> </ul>
<b>Timeout</b>	The maximum time to allow for executing the database command. A value of zero indicates no time limit. <b>Default:</b> 30 seconds

### Checkpoint Properties

Property	Description
<b>Affected rows</b>	The number of rows affected by the command.
<b>Result</b>	Indicates whether the command executed successfully: true or false.
<b>Result Message</b>	Success or upon failure, a message issued by the database indicating the nature of the failure.

## Begin Transaction Activity

### Relevant for: API testing only

This activity enables you to start a database transaction.

You must put an **Open Connection** step in your test before this activity.

The following properties are available for this activity:

### Input Properties

Property	Description
<b>Connection</b>	A link to the output of a previously opened connection string. Use the <b>Link to a data source</b> button  to create the link.
<b>Isolation level</b>	The isolation level of the transaction: <ul style="list-style-type: none"> <li>• Default</li> <li>• Chaos</li> <li>• Read Committed</li> <li>• Read Uncommitted</li> <li>• Repeatable Read</li> <li>• Serializable</li> <li>• Snapshot.</li> </ul> For details, see <a href="http://msdn.microsoft.com/en-us/library/system.data.isolationlevel.aspx">http://msdn.microsoft.com/en-us/library/system.data.isolationlevel.aspx</a> .

### Checkpoint Properties

Property	Description
<b>Result</b>	Indicates whether the connection is active ( <code>true</code> ) or closed ( <code>false</code> ).
<b>Result Message</b>	A message indicating success or an informational message from the database.

## Commit Transaction/Rollback Transaction Activity

### Relevant for: API testing only

This activity enables you to commit the results of a transaction to your database or roll back a selected transaction.

You must put a **Begin Transaction** step before this activity.

The following properties are available for this activity:

### Input Properties

Property	Description
<b>Transaction</b>	A link to an open transaction. Use the <b>Link to a data source</b> button  to create a link to the output of a <b>Begin Transaction</b> step.

## Checkpoint Properties

Property	Description
<b>Result</b>	Indicates whether the connection is active (true) or closed (false).
<b>Result Message</b>	A message indicating success or an informational message from the database.

## FTP Activities

### Relevant for: API testing only

This activity group provides activities that allow you to perform FTP operations, such as **FTP Upload** and **FTP Download**.

- **FTP Download**. Downloads a file or directory from an FTP server. For details, see "[FTP Download Activity](#)" below.
- **FTP Upload**. Uploads a file or directory to an FTP server. For details, see "[FTP Upload Activity](#)" on page 459.
- **FTP Rename**. Renames an existing file or directory on an FTP server. For details, see "[FTP Rename Activity](#)" on page 460.
- **FTP File Delete**. Deletes a file on an FTP server. For details, see "[FTP File Delete Activity](#)" on page 461.
- **FTP File Get Size**. Gets the size of a file on an FTP server. For details, see "[FTP File Get Size Activity](#)" on page 462.
- **FTP Directory Delete**. Deletes an empty directory on an FTP server. For details, see "[FTP Download Activity](#)" below.
- **FTP Directory Create**. Creates a new directory on an FTP server. For details, see "[FTP Directory Create Activity](#)" on page 463.
- **FTP Directory Get Content**. Lists the content of a directory on an FTP server. For details, see "[FTP Directory Get Content Activity](#)" on page 464.

## FTP Download Activity

### Relevant for: API testing only

This activity enables you to download an FTP file from a specified directory.

The following properties are available for this activity:

## General Properties

To view the General properties, click the **General** view button in the Properties pane.

Property	Description
<b>Enable SSL</b>	Indicates whether to establish a secure FTP connection: <b>true</b> or <b>false</b> .
<b>Proxy</b>	The proxy server information: <b>Server</b> , <b>Username</b> and <b>Password</b> .
<b>Timeout</b>	The FTP request timeout in milliseconds.
<b>Client certificate</b>	<ul style="list-style-type: none"> <li>•  <b>Browse Certificates</b>. Opens the Select Certificate dialog box. To see the icon, click in the top level row of the <b>Client certificate</b> section, in the <b>Value</b> column.</li> <li>• A Choice element indicating the source of the client certificate for secure FTP connections: <ul style="list-style-type: none"> <li>• <b>Use file system certificate</b>: The expanded node provides the full path of the certificate file.</li> <li>• <b>Use Windows store certificate</b>: The expanded node lists the following properties <b>Store location</b>, <b>Store name</b>, <b>X509 find type</b>, and <b>X509 find value</b>.</li> <li>• <b>Password</b>. The certificate's password.</li> </ul> </li> </ul>

## Input Properties

Property	Description
<b>URL</b>	The URL location of the file to download.
<b>User ID</b>	The user name to access the FTP server.
<b>Password</b>	The password to access the FTP server.
<b>Local Path</b>	The location to which to download the file.
<b>Overwrite</b>	Enables UFT to overwrite or not overwrite an existing file by the same name. Choose <b>True</b> to overwrite a file of the same name or <b>False</b> to make another copy.
<b>Transfer mode</b>	The method of file transfer: <b>ASCII</b> or <b>binary</b> .

## Checkpoint Properties

Property	Description
<b>Result</b>	An indicator if the step succeeded. Select <b>True</b> to add a checkpoint confirming the successful completion, or <b>False</b> to confirm an unsuccessful attempt to download the selected file.

## FTP Upload Activity

### Relevant for: API testing only

This activity enables you to upload a file to specified FTP directory.

The following properties are available for this activity:

### General Properties

To view the General properties, click the **General** view button in the Properties pane.

Property	Description
<b>Enable SSL</b>	Indicates whether to establish a secure FTP connection: <b>true</b> or <b>false</b> .
<b>Proxy</b>	The proxy server information: <b>Server</b> , <b>Username</b> and <b>Password</b> .
<b>Timeout</b>	The FTP request timeout in milliseconds.
<b>Client certificate</b>	<ul style="list-style-type: none"> <li>•  <b>Browse Certificates</b>. Opens the Select Certificate dialog box. To see the icon, click in the top level row of the <b>Client certificate</b> section, in the <b>Value</b> column.</li> <li>• A Choice element indicating the source of the client certificate for secure FTP connections: <ul style="list-style-type: none"> <li>• <b>Use file system certificate</b>: The expanded node provides the full path of the certificate file.</li> <li>• <b>Use Windows store certificate</b>: The expanded node lists the following properties <b>Store location</b>, <b>Store name</b>, <b>X509 find type</b>, and <b>X509 find value</b>.</li> <li>• <b>Password</b>. The certificate's password.</li> </ul> </li> </ul>

### Input Properties

Property	Description
<b>Directory file path</b>	The URL and directory in which to upload the file.
<b>User ID</b>	The user name to access the FTP server.
<b>Password</b>	The password to access the FTP server.
<b>Local Path</b>	The location from which to upload the file.
<b>Overwrite</b>	Enables UFT to overwrite or not overwrite an existing file by the same name. Choose <b>True</b> to overwrite a file of the same name or <b>False</b> to make another copy.
<b>Transfer mode</b>	The method of file transfer: ASCII or binary.

## Checkpoint Properties

Property	Description
<b>Result</b>	An indicator if the step succeeded. Select <b>True</b> to add a checkpoint confirming the successful completion, or <b>False</b> to confirm an unsuccessful attempt to download the selected file.

## FTP Rename Activity

### Relevant for: API testing only

This activity enables you to rename a specified file in an FTP directory.

The following properties are available for this activity:

### General Properties

To view the General properties, click the **General** view button in the Properties pane.

Property	Description
<b>Enable SSL</b>	Indicates whether to establish a secure FTP connection: <b>true</b> or <b>false</b> .
<b>Proxy</b>	The proxy server information: <b>Server</b> , <b>Username</b> and <b>Password</b> .
<b>Timeout</b>	The FTP request timeout in milliseconds.
<b>Client certificate</b>	<ul style="list-style-type: none"> <li>•  <b>Browse Certificates</b>. Opens the Select Certificate dialog box. To see the icon, click in the top level row of the <b>Client certificate</b> section, in the <b>Value</b> column.</li> <li>• A Choice element indicating the source of the client certificate for secure FTP connections: <ul style="list-style-type: none"> <li>• <b>Use file system certificate</b>: The expanded node provides the full path of the certificate file.</li> <li>• <b>Use Windows store certificate</b>: The expanded node lists the following properties <b>Store location</b>, <b>Store name</b>, <b>X509 find type</b>, and <b>X509 find value</b>.</li> <li>• <b>Password</b>. The certificate's password.</li> </ul> </li> </ul>

### Input Properties

Property	Description
<b>URL</b>	The URL location of the file to rename.
<b>User ID</b>	The user name to access the FTP server.
<b>Password</b>	The password to access the FTP server.
<b>New file name</b>	The name of the file after renaming.

## Checkpoint Properties

Property	Description
<b>Result</b>	An indicator if the step succeeded. Select <b>True</b> to add a checkpoint confirming the successful completion, or <b>False</b> to confirm an unsuccessful attempt to download the selected file.

## FTP File Delete Activity

### Relevant for: API testing only

This activity enables you to delete a selected file from an FTP directory.

The following properties are available for this activity:

### General Properties

To view the General properties, click the **General** view button in the Properties pane.

Property	Description
<b>Enable SSL</b>	Indicates whether to establish a secure FTP connection: <b>true</b> or <b>false</b> .
<b>Proxy</b>	The proxy server information: <b>Server</b> , <b>Username</b> and <b>Password</b> .
<b>Timeout</b>	The FTP request timeout in milliseconds.
<b>Client certificate</b>	<ul style="list-style-type: none"> <li>•  <b>Browse Certificates</b>. Opens the Select Certificate dialog box. To see the icon, click in the top level row of the <b>Client certificate</b> section, in the <b>Value</b> column.</li> <li>• A Choice element indicating the source of the client certificate for secure FTP connections: <ul style="list-style-type: none"> <li>• <b>Use file system certificate</b>: The expanded node provides the full path of the certificate file.</li> <li>• <b>Use Windows store certificate</b>: The expanded node lists the following properties <b>Store location</b>, <b>Store name</b>, <b>X509 find type</b>, and <b>X509 find value</b>.</li> <li>• <b>Password</b>. The certificate's password.</li> </ul> </li> </ul>

### Input Properties

Property	Description
<b>URL</b>	The URL location of the file to delete.
<b>User ID</b>	The user name to access the FTP server.
<b>Password</b>	The password to access the FTP server.

## Checkpoint Properties

Property	Description
<b>Result</b>	An indicator if the step succeeded. Select <b>True</b> to add a checkpoint confirming the successful completion, or <b>False</b> to confirm an unsuccessful attempt to download the selected file.

## FTP File Get Size Activity

### Relevant for: API testing only

This activity enables you to check the size of a file on a specified FTP directory.

The following properties are available for this activity:

### Input Properties

Property	Description
<b>URL</b>	The URL location of the file to check.
<b>User ID</b>	The user name to access the FTP server.
<b>Password</b>	The password to access the FTP server.

## Checkpoint Properties

Property	Description
<b>Result</b>	An indicator if the step succeeded. Select <b>True</b> to add a checkpoint confirming the successful completion, or <b>False</b> to confirm an unsuccessful attempt to download the selected file.
<b>File size</b>	The expected size of the file.

## FTP Directory Delete Activity

### Relevant for: API testing only

This activity enables you to delete a selected FTP directory.

The following properties are available for this activity:

### General Properties

To view the General properties, click the **General** view button in the Properties pane.

Property	Description
<b>Enable SSL</b>	Indicates whether to establish a secure FTP connection: <b>true</b> or <b>false</b> .
<b>Proxy</b>	The proxy server information: <b>Server</b> , <b>Username</b> and <b>Password</b> .
<b>Timeout</b>	The FTP request timeout in milliseconds.
<b>Client certificate</b>	<ul style="list-style-type: none"> <li>•  <b>Browse Certificates</b>. Opens the Select Certificate dialog box. To see the icon, click in the top level row of the <b>Client certificate</b> section, in the <b>Value</b> column.</li> <li>• A Choice element indicating the source of the client certificate for secure FTP connections: <ul style="list-style-type: none"> <li>• <b>Use file system certificate</b>: The expanded node provides the full path of the certificate file.</li> <li>• <b>Use Windows store certificate</b>: The expanded node lists the following properties <b>Store location</b>, <b>Store name</b>, <b>X509 find type</b>, and <b>X509 find value</b>.</li> <li>• <b>Password</b>. The certificate's password.</li> </ul> </li> </ul>

## Input Properties

Property	Description
<b>URL</b>	The URL location of the file to download.
<b>User ID</b>	The user name to access the FTP server.
<b>Password</b>	The password to access the FTP server.

## Checkpoint Properties

Property	Description
<b>Result</b>	An indicator if the step succeeded. Select <b>True</b> to add a checkpoint confirming the successful completion, or <b>False</b> to confirm an unsuccessful attempt to download the selected file.

## FTP Directory Create Activity

### Relevant for: API testing only

This activity enables you to create an FTP directory.

The following properties are available for this activity:

### General Properties

To view the General properties, click the **General** view button in the Properties pane.

Property	Description
<b>Enable SSL</b>	Indicates whether to establish a secure FTP connection: <b>true</b> or <b>false</b> .
<b>Proxy</b>	The proxy server information: <b>Server</b> , <b>Username</b> and <b>Password</b> .
<b>Timeout</b>	The FTP request timeout in milliseconds.
<b>Client certificate</b>	<ul style="list-style-type: none"> <li>•  <b>Browse Certificates</b>. Opens the Select Certificate dialog box. To see the icon, click in the top level row of the <b>Client certificate</b> section, in the <b>Value</b> column.</li> <li>• A Choice element indicating the source of the client certificate for secure FTP connections: <ul style="list-style-type: none"> <li>• <b>Use file system certificate</b>: The expanded node provides the full path of the certificate file.</li> <li>• <b>Use Windows store certificate</b>: The expanded node lists the following properties <b>Store location</b>, <b>Store name</b>, <b>X509 find type</b>, and <b>X509 find value</b>.</li> <li>• <b>Password</b>. The certificate's password.</li> </ul> </li> </ul>

## Input Properties

Property	Description
<b>URL</b>	The URL location of the directory to create.
<b>User ID</b>	The user name to access the FTP server.
<b>Password</b>	The password to access the FTP server.
<b>Directory Path</b>	The FTP location in which to create the new directory.

## Checkpoint Properties

Property	Description
<b>Result</b>	An indicator if the step succeeded. Select <b>True</b> to add a checkpoint confirming the successful completion, or <b>False</b> to confirm an unsuccessful attempt to download the selected file.

## FTP Directory Get Content Activity

### Relevant for: API testing only

This activity enables you retrieve the content from a specified FTP directory.

The following properties are available for this activity:

### General Properties

To view the General properties, click the **General** view button in the Properties pane.

Property	Description
<b>Enable SSL</b>	Indicates whether to establish a secure FTP connection: <b>true</b> or <b>false</b> .
<b>Proxy</b>	The proxy server information: <b>Server</b> , <b>Username</b> and <b>Password</b> .
<b>Timeout</b>	The FTP request timeout in milliseconds.
<b>Client certificate</b>	<ul style="list-style-type: none"> <li>•  <b>Browse Certificates</b>. Opens the Select Certificate dialog box. To see the icon, click in the top level row of the <b>Client certificate</b> section, in the <b>Value</b> column.</li> <li>• A Choice element indicating the source of the client certificate for secure FTP connections: <ul style="list-style-type: none"> <li>• <b>Use file system certificate</b>: The expanded node provides the full path of the certificate file.</li> <li>• <b>Use Windows store certificate</b>: The expanded node lists the following properties <b>Store location</b>, <b>Store name</b>, <b>X509 find type</b>, and <b>X509 find value</b>.</li> <li>• <b>Password</b>. The certificate's password.</li> </ul> </li> </ul>

## Input Properties

Property	Description
<b>URL</b>	The URL location of the directory to check.
<b>User ID</b>	The user name to access the FTP server.
<b>Password</b>	The password to access the FTP server.

## Checkpoint Properties

Property	Description
<b>Result (array)</b>	An array of possible property values for the content in the directory. Possible values include: <ul style="list-style-type: none"> <li>• <b>Name</b></li> <li>• <b>Type</b></li> <li>• <b>Flags</b></li> <li>• <b>Owner</b></li> <li>• <b>Group</b></li> <li>• <b>Size</b></li> <li>• <b>Date Created</b></li> </ul>

## Network Activities

### Relevant for: API testing only

This activity group provides activities that enable you to send and receive HTTP and SOAP requests.

This activity group contains the following activities:

- **HTTP Request**. This activity enables you to send an HTTP request over the network. For details, see "[HTTP Request Activity](#)" below.
- **HTTP Receiver**. This activity enables you to receive an HTTP response from a server. For details, see "[HTTP Receiver Activity](#)" on page 468.
- **SOAP Request**. This activity enables you to send a SOAP request over the network. For details, see "[SOAP Request Activity](#)" on page 469.

## HTTP Request Activity

### Relevant for: API testing only

This activity enables you to add an HTTP request to a network-based service to your test.

The following properties are available for these activities:

### General Properties

To view the General properties, click the **General** view button in the Properties pane.

**Note:** These properties are also relevant for Web Service request steps.

Property	Description
<b>Proxy</b>	The settings for the proxy server hosting the service contract: <b>Server</b> (URL and port when required), <b>Username</b> , and <b>Password</b> .
<b>Authentication</b>	The user credentials settings for obtaining a service contract: <b>Username</b> and <b>Password</b> .
<b>Use preemptive authentication</b>	Enables you to send authentication for your Web service call without waiting for the authentication challenge from the Web service/server.
<b>Connection type</b>	The type of connection: <b>Keep-Alive</b> or <b>Close</b> . <b>Default:</b> Keep - Alive
<b>Timeout</b>	The HTTP timeout in milliseconds.
<b>Client certificate</b>	A Choice element indicating the source of the client certificate: <ul style="list-style-type: none"><li>• <b>Use file system certificate</b>: The expanded node provides the full path of the certificate file.</li><li>• <b>Use Windows store certificate</b>: The expanded node lists the following properties <b>Store location</b>, <b>Store name</b>, <b>X509 find type</b>, and <b>X509 find value</b>.</li><li>• <b>Password</b>: The certificate's password.</li></ul> <p><b>Note:</b> To use a certificate, make sure to the <b>Use Client Certificate</b> property to true.</p>
<b>Use Client Certificate</b>	Enables the use of a client certificate. <b>Default:</b> false

Property	Description
<b>Maximum automatic redirections</b>	The number of times to attempt accessing a page, including redirection. <b>Default:</b> 3
<b>Allow redirections</b>	Indicates whether to allow the step to redirect to another URL. <b>Default:</b> true
<b>Reuse cookies</b>	Enables the reusing of cookies for the current step. <b>Default:</b> false.
<b>Save request body with this extension</b>	Instructs UFT to save the body of the request for the HTTP Request step with the specified extension.
<b>Save response body with this extension</b>	Instructs UFT to save the body of the response for the HTTP Request step with the specified extension.
<b>Expect server error</b>	Instructs UFT to anticipate a server error when performing the HTTP Request.

## Input Properties

Property	Description
<b>URL</b>	The URL to which to make the HTTP Request.
<b>HTTP Method</b>	The method by which to send the request. You can select one of the following: <ul style="list-style-type: none"> <li>• GET</li> <li>• POST</li> <li>• PUT</li> <li>• DELETE</li> <li>• TRACE</li> <li>• OPTIONS</li> <li>• HEAD</li> </ul>
<b>HTTP Version</b>	The HTTP version to use in the request. Choose from version 1.0 or version 1.1.
<b>Request Headers</b>	<p>The request header to send with the HTTP request.</p> <p>This property is displayed as an array. You must provide the following details for the request:</p> <ul style="list-style-type: none"> <li>• <b>Name</b></li> <li>• <b>Value</b></li> </ul> <p><b>Note:</b> To send multiple request headers, click the <b>Add Array Element</b> button  in the <b>RequestHeaders</b> row.</p>

## Checkpoint Properties

Property	Description

<b>HTTP Version</b>	The HTTP version of the response to the HTTP request.
<b>Status code</b>	The expected code of the response.
<b>Status description</b>	The expected description of the response.
<b>Response body as UTF-8 string</b>	The expected response in UTF-8 form.
<b>Response body as Base64 string</b>	The expected response in Base64 form.

## HTTP Receiver Activity

### Relevant for: API testing only

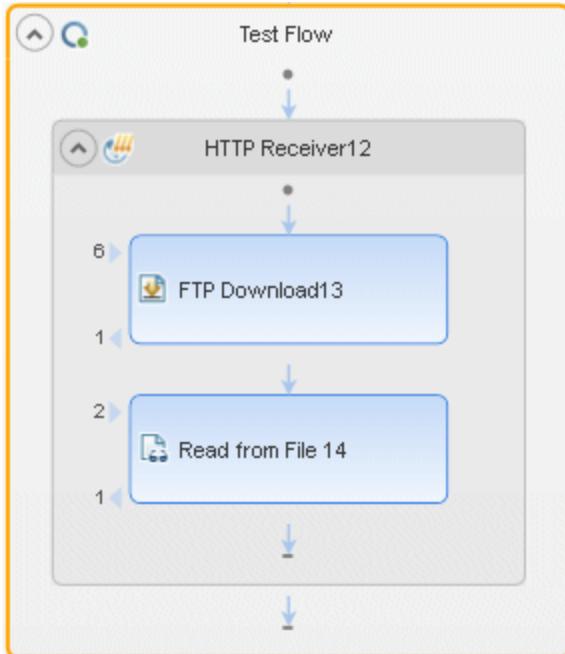
**Note:** To run an HTTP Receiver step, you must be logged in with administrator privileges.

The **HTTP Receiver** activity receives an HTTP message from a server. This activity is used with asynchronous messaging. For details, see "[Asynchronous Service Calls](#)" on page 597.

Receiver activities are activities that act as receiver for a server response. This category includes the built-in **HTTP Receiver** activity and Web Service operations that were imported as a server response.

All receiver activities are composite activities. This means that they serve as a wrapper for all activities inside the container whose responses are sent to the HTTP Receiver step—not the Test Flow.

For example, if an HTTP Receiver step contains the **FTP Download** and **Read from File** steps, these internal steps send their response to the HTTP Receiver step—not to the Test Flow loop.



The following properties are available for these activities:

## General Properties

To view the General properties, click the **General** view button in the Properties pane.

Property	Description
<b>Listen on port</b>	The port upon which to listen for the server request.
<b>Completion event name</b>	The name of the event associated with a following <b>Wait</b> step, allowing the step to receive the server request.
<b>Use SSL</b>	Secures the connection with SSL: <b>true</b> or <b>false</b> .
<b>SSL Certificate</b>	Information about the SSL certificate: <ul style="list-style-type: none"> <li>• <b>Subject.</b> The certificate's subject string.</li> <li>• <b>Store location.</b> The location of the certificate—LocalMachine or Windows store.</li> <li>• <b>Store name.</b> The name of the store hosting the certificate.</li> <li>• <b>Thumbprint.</b> The certificate's thumbprint or hash code.</li> </ul>
<b>Save response body with this extension</b>	Saves the response body for messages with the specified extension.

## Output Properties

You can link to the **HTTP Receiver** output property, its response body, through the Select Link Source dialog box.

Property	Description
<b>Response body as a UFT-8 string</b>	The body of the HTTP response in UFT-8 format.

## SOAP Request Activity

### Relevant for: API testing only

The **SOAP Request** activity sends a manual SOAP request to the server, usually in XML format. You can import a schema and load XML for this activity.

The following properties are available for these activities:

## General Properties

To view the General properties, click the **General** button in the Properties pane.

**Note:** These properties are also relevant for REST Service method steps.

Property	Description
<b>Transport</b>	The type of transport for the SOAP request: <b>HTTP</b> or <b>JMS</b> .
<b>Transport &gt; HTTP</b>	<p>Use the HTTP transport method for this call with the following properties:</p> <ul style="list-style-type: none"> <li>• <b>Endpoint Address.</b> The endpoint location of the service as derived from the WSDL</li> <li>• <b>SoapAction.</b> An expression indicating the SOAP action</li> </ul> <p style="background-color: #f0f0f0; padding: 5px;"><b>Example:</b> HP.SOAQ.SampleApp/IHPFlights_Service/DeleteFlightOrder.</p> <ul style="list-style-type: none"> <li>• <b>ContentType.</b> The type of content: For example <code>text/xml; charset=utf-8</code>.</li> <li>• <b>Timeout.</b> The maximum time in milliseconds to wait for the response. Enter <code>-1</code> to disable the timeout and wait for the response as long as required.</li> </ul>
<b>Transport &gt; JMS</b>	<p>Use the JMS transport method for this call with the following properties:</p> <ul style="list-style-type: none"> <li>• <b>Send queue name.</b> The name of the queue from which to send the message.</li> <li>• <b>Receive queue name.</b> The name of the queue from which to receive the message.</li> <li>• <b>JMS send properties.</b> JMS properties with the key/value form. You can use the key to modify JMS header or message properties.</li> <li>• <b>JMS receive message selector.</b> An SQL expression to filter the received message. For more information on the syntax for defining selectors, see the <b>Message Properties</b> section in <a href="http://docs.oracle.com/javaee/1.3/api/javax/jms/Message.html">http://docs.oracle.com/javaee/1.3/api/javax/jms/Message.html</a>. For example, a key <code>JMSCorrelationID</code>, with a value of <code>4566636</code>, receives only messages whose correlation ID is <code>4566636</code>.</li> </ul>
<b>IsOneWay</b>	Indicates whether the service is a one way service: <code>true</code> or <code>false</code> . A one way service only sends a request. A non-one way service sends a request and receives a response.

The input and checkpoint properties for the SOAP Request activity are dependent on the file you load containing your Request and Response information.

## JSON Activities

### Relevant for: API testing only

This activity group contains activities related to JSON files:

- **JSON to String.** Converts a JSON file to a text string. For details, see "[JSON to String Activity](#)" on the next page.
- **String to JSON.** Converts a string to a JSON structure. For details, see "[String to JSON Activity](#)" on the next page.

This **JSON to String** and **String to JSON** activities are useful when you need to access the output in a specific format—either as string or JSON.

For example, suppose your test contains a Web Service call whose output is JSON. However, you need to use the output as an input for the next step, such as an API test step, which requires a textual string—not JSON.

You can use the **JSON to String** activity to create a string that is compatible with the API test step.

After the API test step, you can use a **String to JSON** activity to convert the output to JSON, making it available for linking and checkpoints.

## String to JSON Activity

### **Relevant for: API testing only**

This activity enables you to convert a selected string to JSON format.

The following properties are available for this activity:

### **Input Properties**

Property	Description
Source string	The string to convert into JSON text.

The checkpoints are dependent upon the JSON text contained in the file you load in the checkpoints section.

## JSON to String Activity

### **Relevant for: API testing only**

This activity enables you to take JSON text and convert it into a string.

The following properties are available for this activity:

The input properties are dependent upon the JSON text contained in the file you load in the checkpoints section.

### **Checkpoint Properties**

Property	Description
Source string	The string to convert into JSON text.

## Java Activities

### **Relevant for: API testing only**

This activity group contains the **Call Java Class** activity. It enables you to set up and configure a call to a Java class.

The input and checkpoint properties for this activity are dependent on the information in your Java class file.

## JMS Activities

### Relevant for: API testing only

The JMS transport method is a J2EE standard for sending messages—either text or Java objects—between Java clients. UFT supports the sending of text messages between Java clients. There are two scenarios for communication:

- **Peer-to-Peer.** Also known as **Point-to-Point**. JMS implements point-to-point messaging by defining a message queue as the target for a message. Multiple senders send messages to a message queue, and the receiver gets the message from the queue.
- **Publish-Subscribe.** Each message is sent from one publisher to many subscribers through a designated topic. The subscribers only receive messages sent after they have subscribed.

To interpret Topic and Queue names, UFT calls a lookup method on a JNDI context, defined in the Properties pane's Test Settings.

For task details, see "[How to Use JMS Activities](#)" on page 422.

This activity group provides steps that handle JMS messages, and contains the following activities:

- **Publish Message to JMS Topic.** Publishes a message to multiple subscribers using a JMS topic.
- **Receive Message from JMS Queue.** Receives a message from a JMS queue.
- **Receive Message from JMS Topic.** Receives published messages to a specific JMS topic for a subscription. Place this step after **Subscribe to JMS Topic**.
- **Send Message to JMS Queue.** Sends a message to a JMS queue.
- **Send and Receive Message from JMS Queue.** Sends a message to a specified queue and receives a message from a specified queue.
- **Subscribe to JMS Topic.** Creates a subscription for a JMS topic. Use this step before any **Receive Message from Topic** steps for the subscription.
- **Browse JMS Queue Messages.** Retrieves messages from the JMS queue without consuming them.

### Publish Message to JMS Topic Activity

#### Relevant for: API testing only

This activity enables you to publish a message to a specified JMS topic.

The following properties are available for these activities:

#### Input Properties

Property	Description
<b>Topic name</b>	The name of the queue from which to receive the message.

Property	Description
<b>Message</b>	The message to publish.
<b>JMS send properties</b>	JMS properties with the key/value form. You can use the key to modify JMS header or message properties.

## Checkpoint Properties

Property	Description
<b>Result</b>	A boolean variable indicating whether the step succeeded.

## Receive Message from JMS Queue Activity

### Relevant for: API testing only

This activity enables you receive a message from a selected JMS Queue.

The following properties are available for this activity:

### Input Properties

Property	Description
<b>JMS receive message selector</b>	An SQL expression to filter the received message. For details on the syntax for defining selectors, see the <b>Message Properties</b> section in <a href="http://docs.oracle.com/javaee/1.3/api/javax/jms/Message.html">http://docs.oracle.com/javaee/1.3/api/javax/jms/Message.html</a> . For example, a key JMSCorrelationID, with a value of 4566636, receives only messages whose correlation ID is 4566636.
<b>Receive queue name</b>	The name of the queue from which to receive the message. If the value is empty, it uses a temporary queue.

## Checkpoint Properties

Property	Description
<b>JMSMessageID</b>	A unique ID for the message.
<b>JMSTimeStamp</b>	The time and date of the message.
<b>JMSCorrelationID</b>	The message's correlation ID.  <b>Tip:</b> Use in conjunction with the JMS receive message selector.
<b>JMSReplyTo</b>	The contents of the <b>Reply To</b> field.
<b>JMSDestination</b>	The message's destination.
<b>JMSDeliveryMode</b>	The message's delivery mode. Use the scroller to select a value.

Property	Description
<b>JMSRedelivered</b>	Indicates whether the message was redelivered. <b>False</b> by default.
<b>JMSType</b>	A string describing the JMS type.
<b>JMSExpiration</b>	The message's expiration date.
<b>JMSPriority</b>	The message's priority in comparison to other messages. Use the scroller to select a value.
<b>JMS properties</b>	An array of message properties in the structure of keys and values.  <b>Tip:</b> Use the <b>Number of elements</b> drop-down to specify the number of properties, or click the <b>Add</b> button to add elements.
<b>Message body</b>	The message content.

## Receive Message from JMS Topic Activity

### Relevant for: API testing only

This activity enables you to receive a message from a selected JMS topic.

The following properties are available for this activity:

### Input Properties

Property	Description
<b>Topic name</b>	The name of the topic from which to receive the message.
<b>Subscription name</b>	The name of the subscription. Use the same subscriber name value from the previously-called <b>Subscribe to JMS Topic</b> step.

### Checkpoint Properties

Property	Description
<b>JMSMessageID</b>	A unique ID for the message.
<b>JMSTimeStamp</b>	The time and date of the message.
<b>JMSCorrelationID</b>	The message's correlation ID.  <b>Tip:</b> Use in conjunction with the JMS receive message selector.
<b>JMSReplyTo</b>	The contents of the <b>Reply To</b> field.
<b>JMSDestination</b>	The message's destination.
<b>JMSDeliveryMode</b>	The message's delivery mode. Use the scroller to select a value.
<b>JMSRedelivered</b>	Indicates whether the message was redelivered. <b>False</b> by default.

Property	Description
<b>JMSType</b>	A string describing the JMS type.
<b>JMSExpiration</b>	The message's expiration date.
<b>JMSPriority</b>	The message's priority in comparison to other messages. Use the scroller to select a value.
<b>JMS properties</b>	An array of message properties in the structure of keys and values.  <b>Tip:</b> Use the <b>Number of elements</b> drop-down to specify the number of properties, or click the <b>Add</b> button to add elements.
<b>Message body</b>	The message content.

## Send Message to JMS Queue Activity

### Relevant for: API testing only

This activity enables you to send a message to a selected JMS queue.

The following properties are available for this activity:

### Input Properties

Property	Description
<b>Send queue name</b>	The name of the queue from which to send the message.
<b>Message</b>	The message to send.
<b>JMS send properties</b>	JMS properties with the key/value form. You can use the key to modify JMS header or message properties.

### Checkpoint Properties

Property	Description
<b>Result</b>	A boolean variable indicating whether the step succeeded.

## Send/Receive Messages from JMS Queue Activity

### Relevant for: API testing only

This activity enables you to send or receive messages from a selected JMS queue activity.

The following properties are available for this activity:

## Input Properties

Property	Description
<b>Send queue name</b>	The name of the queue to which to send the message.
<b>Receive queue name</b>	The name of the queue from which to receive the message. If the value is empty, it uses a temporary queue.
<b>Message</b>	The message to send.
<b>JMS send properties</b>	JMS properties with the key/value form. You can use the key to modify JMS header or message properties.
<b>JMS receive message selector</b>	An SQL expression to filter the received message. For details on the syntax for defining selectors, see the <b>Message Properties</b> section in <a href="http://docs.oracle.com/javaee/1.3/api/javax/jms/Message.html">http://docs.oracle.com/javaee/1.3/api/javax/jms/Message.html</a> . For example, a key <b>JMSCorrelationID</b> , with a value of 4566636, receives only messages whose correlation ID is 4566636.  <b>Tip:</b> To automatically generate a selector, use the <b>Automatically generate selector</b> option in the Properties pane's <b>Test Settings</b> tab. This selector uses a correlation ID that corresponds to the request's message ID.

## Checkpoint Properties

Property	Description
<b>JMSMessageID</b>	A unique ID for the message.
<b>JMSTimeStamp</b>	The time and date of the message.
<b>JMSCorrelationID</b>	The message's correlation ID.  <b>Tip:</b> Use in conjunction with the JMS receive message selector.
<b>JMSReplyTo</b>	The contents of the <b>Reply To</b> field.
<b>JMSDestination</b>	The message's destination.
<b>JMSDeliveryMode</b>	The message's delivery mode. Use the scroller to select a value.
<b>JMSRedelivered</b>	Indicates whether the message was redelivered. <b>False</b> by default.
<b>JMSType</b>	A string describing the JMS type.
<b>JMSExpiration</b>	The message's expiration date.
<b>JMSPriority</b>	The message's priority in comparison to other messages. Use the scroller to select a value.
<b>JMS properties</b>	An array of message properties in the structure of keys and values.

Property	Description
JMSMessageID	A unique ID for the message.
	<p><b>Tip:</b> Use the <b>Number of elements</b> drop-down to specify the number of properties, or click the <b>Add</b> button to add elements.</p>
Message body	The message content.

## Subscribe to JMS Topic Activity

### Relevant for: API testing only

This activity enables you to subscribe to a selected JMS topic.

The following properties are available for this activity:

### Input Properties

Property	Description
Topic name	The name of the topic from which to receive the message.
Subscription name	The name of the subscription. To retrieve messages on the subscription created by this step, insert a <b>Receive Message from Topic</b> step using the same <b>Subscription name</b> value.
JMS receive message selector	An SQL expression to filter the received message. For details on the syntax for defining selectors, see the <b>Message Properties</b> section in <a href="http://docs.oracle.com/javaee/1.3/api/javax/jms/Message.html">http://docs.oracle.com/javaee/1.3/api/javax/jms/Message.html</a> .

### Checkpoint Properties

Property	Description
Result	A boolean variable indicating whether the step succeeded.

## Browse JMS Queue Activity

### Relevant for: API testing only

This activity enables you to browse a selected JMS queue.

The following properties are available for this activity:

### Input Properties

Property	Description
Queue	The name of the queue upon which to browse for a collection of messages. This field is mandatory.

Property	Description
<b>name</b>	<b>Note:</b> When using IBM ActiveMQ, you must use the format dynamicQueues / <queue_name>. If the specified queue does not exist, it will be created dynamically.
<b>Messages selector</b>	An optional SQL expression to filter the received messages. For details on the syntax for defining selectors, see the <b>Message Properties</b> section in <a href="http://www.oracle.com/technetwork/java/jms-101-spec-150080.pdf">http://www.oracle.com/technetwork/java/jms-101-spec-150080.pdf</a> .

## Checkpoint Properties

Property	Description
<b>JMSMessageID</b>	A unique ID for the message.
<b>JMSTimeStamp</b>	The time and date of the message.
<b>JMSCorrelationID</b>	The message's correlation ID.  <b>Tip:</b> Use in conjunction with the JMS receive message selector.
<b>JMSReplyTo</b>	The contents of the <b>Reply To</b> field.
<b>JMSDestination</b>	The message's destination.
<b>JMSDeliveryMode</b>	The message's delivery mode. Use the scroller to select a value.
<b>JMSRedelivered</b>	Indicates whether the message was redelivered. <b>False</b> by default.
<b>JMSType</b>	A string describing the JMS type.
<b>JMSExpiration</b>	The message's expiration date.
<b>JMSPriority</b>	The message's priority in comparison to other messages. Use the scroller to select a value.
<b>JMS properties</b>	An array of message properties in the structure of keys and values.  <b>Tip:</b> Use the <b>Number of elements</b> drop-down to specify the number of properties, or click the <b>Add</b> button to add elements.
<b>Message body</b>	The message content.

## Load Testing Activities

### Relevant for: API testing only

Load testing enable you to measure the performance of your application or service with a defined user load.

When you have a full LoadRunner installation on the same machine, UFT provides a custom template, **API Load Test**.

For tests created with the standard **API Test** template, UFT provides a conversion utility which converts the script into a LoadRunner compatible script, with a **.usr** extension.

**Note:** Tests created in UFT and converted into LoadRunner scripts, cannot be edited in LoadRunner's Virtual User Generator (VuGen). To edit a test, modify it in UFT.

When working with data tables, you can use the standard data retrieval methods that are available in LoadRunner.

For task details, see "[How to Prepare and Run a Load Test](#)" on page 429.

This activity group contains the following activities that allow you prepare the test for load testing:

- **Start Transaction.** Marks the beginning of a LoadRunner transaction.
- **End Transaction.** Marks the end of a LoadRunner transaction.
- **Think Time.** Emulates a time delay, in seconds, between steps.

For details, see "[Prepare for load testing - optional](#)" on page 429.

## IBM WebSphere MQ Activities

### **Relevant for: API testing only**

This activity group enables you to perform actions on an IBM Websphere MQ application server.

The activities in this group are:

- **Connect to MQ Queue Manager.** Connects to a specific MQ Queue Manager and maintains an open connection with the server. For details, see "[Connect to MQ Queue Manager Activity](#)" on the next page.
- **Disconnect from MQ Queue Manager.** Disconnects from the specific MQ Queue Manager and closes the connection with the MQ server. For details, see "[Disconnect from MQ Queue Manager Activity](#)" on page 481.
- **Commit MQ Pending Messages.** Commits the last changes made for all messages since the last point of synchronization. For details, see "[Commit/Backout Pending MQ Messages Activity](#)" on page 481.
- **Backout MQ Pending Messages.** Performs a Backout operation from the last point of synchronization. All PUT messages will be cancelled and GET messages will be reinstated to the queue. For details, see "[Commit/Backout Pending MQ Messages Activity](#)" on page 481.
- **Browse Messages in MQ Queue.** Browses the messages on the MQ Queue via the Queue Manager. For details, see "[Browse Messages in MQ Queue Activity](#)" on page 482.
- **Put Message to MQ Queue.** Puts a message on the MQ queue via the Queue Manager. For details, see "[Put Message to MQ Queue Activity](#)" on page 484.
- **Get Message from MQ Queue.** Gets the most recent message from the MQ queue via the Queue Manager. For details, see "[Get Message from MQ Queue Activity](#)" on page 485.
- **Put and Get Message from MQ Queue.** Puts a message on the MQ queue and then gets a message

from the MQ queue via the Queue Manager. For details, see "[Put and Get Message from MQ Queue](#)" on page 486.

- **Subscribe to MQ Topic.** Creates a subscription for an MQ topic. From this point on, a subscriber can receive messages from a specified topic. For details, see "[Subscribe to MQ Topic Activity](#)" on page 488.
- **Unsubscribe to MQ Topic.** Cancels a subscription for the specified MQ topic. For details, see "[Unsubscribe from MQ Topic Activity](#)" on page 489.
- **Publish Message to MQ Topic.** Publishes a message to subscribers using an MQ topic. For details, see "[Publish Message to MQ Topic Activity](#)" on page 489.
- **Receive Message from MQ Topic.** Receives messages for the active subscription, that were published to a specific MQ topic. For details, see "[Receive Message from MQ Topic Activity](#)" on page 490.

UFT enables you to set the data format of the message for both queues and topics. The **String** format indicates Unicode and non- Unicode strings without a prefix. The **UTF** format represents a UTF string with a 2-byte prefix.

## Connect to MQ Queue Manager Activity

### Relevant for: API testing only

This activity enables you to connect to the IBM Websphere MQ Queue Manager.

The properties available for this activity include:

### Input Properties

Property	Description
<b>Host name</b>	The name or IP address of the Queue Manager's host machine.
<b>Port</b>	The port through which to connect, by default <b>1414</b> .
<b>Channel</b>	The channel through which to connect. For most configurations, use SYSTEM.DEF.SVRCONN.
<b>Queue manager name</b>	An property indicating the name of the Queue Manager.
<b>User ID</b>	An optional identification property for connecting to the host machine.
<b>Password</b>	An optional password for connecting to the host machine.
<b>SSL</b>	Enables or disables SSL. When you enable SSL, you can set values for the following properties: <ul style="list-style-type: none"><li>• <b>SSLCipherSpec.</b> The SSL Cipher Specification, SSLCIPH. This specification indicates which data encryption algorithm and key size to use, such as DES_SHA_EXPORT or RC2_MD5_EXPORT.</li><li>• <b>SSLKeyRepository.</b> The path of the SSL key repository on the client machine.</li></ul>

## Checkpoint Properties

Property	Description
<b>MQManager</b>	An automatically generated connection expression.  <b>Note:</b> <ul style="list-style-type: none"> <li>Although this does not appear in the <b>Checkpoints</b> grid, it is visible as an output property in the Select Link Source dialog box.</li> <li>When you add steps, they automatically use the connection expression from the most recent connection step.</li> </ul>
<b>Result</b>	A boolean variable indicating whether the step succeeded.

## Disconnect from MQ Queue Manager Activity

### Relevant for: API testing only

This activity enables you to disconnect from the IBM Websphere MQ Queue Manager.

The properties available for this activity include:

## Input Properties

Property	Description
<b>MQManager</b>	A connection expression linked to a <b>Connect to MQ Queue Manager</b> step.  <b>Note:</b> When you add a <b>Disconnect to MQ Queue Manager</b> step, it automatically creates a link to the most recent connection step.

## Checkpoint Properties

Property	Description
<b>Result</b>	A boolean variable indicating whether the step succeeded.

## Commit/Backout Pending MQ Messages Activity

### Relevant for: API testing only

This activity enables you to commit or roll back pending messages from a MQ Queue.

You must put a **Connect to MQ Queue Manager** step before this step.

The properties available for this activity include:

### Input Properties

Property	Description
<b>MQManager</b>	A connection expression linked to a <b>Connect to MQ Queue Manager</b> step.  <b>Note:</b> When you add this step, it automatically creates a link to the most recent connection step.

### Checkpoint Properties

Property	Description
<b>Result</b>	A boolean variable indicating whether the step succeeded.

## Browse Messages in MQ Queue Activity

### Relevant for: API testing only

This activity enables you to browse all the messages in a selected MQ Queue.

You must put a **Connect to MQ Queue Manager** step before this step.

The following properties are available for this activity:

### Input Properties

Property	Description
<b>MQManager</b>	A connection expression linked to the most recent <b>Connect to MQ Queue Manager</b> step.
<b>Queue name</b>	The name of the MQ queue to browse.
<b>Queue Access Options</b>	A tree with several built-in MQ queue access options.  <b>Note:</b> To access additional options, use an event handler. For details, see " <a href="#">Writing Event Handlers for API Test Steps</a> " on page 751.
<b>Message Get Options</b>	A tree with several built-in MQ Get options.  <b>Note:</b> To access additional options, use an event handler. For details, see " <a href="#">Writing Event Handlers for API Test Steps</a> " on page 751.
<b>Wait interval</b>	The maximum time in milliseconds to wait for the message to arrive, using the following values: <ul style="list-style-type: none"> <li>• -1: Unlimited wait time</li> <li>• 0: No wait</li> <li>• A positive number: The time to wait in milliseconds.</li> </ul>
<b>Message</b>	The format of the message being retrieved: <b>String</b> or <b>UTF</b> .

Property	Description
<b>MQManager</b>	A connection expression linked to the most recent <b>Connect to MQ Queue Manager</b> step.
<b>Queue name</b>	The name of the MQ queue to browse.
<b>Queue Access Options</b>	A tree with several built-in MQ queue access options.  <b>Note:</b> To access additional options, use an event handler. For details, see " <a href="#">Writing Event Handlers for API Test Steps</a> " on page 751.
<b>Message Get Options</b>	A tree with several built-in MQ Get options.  <b>Note:</b> To access additional options, use an event handler. For details, see " <a href="#">Writing Event Handlers for API Test Steps</a> " on page 751.
<b>Wait interval</b>	The maximum time in milliseconds to wait for the message to arrive, using the following values: <ul style="list-style-type: none"> <li>• -1: Unlimited wait time</li> <li>• 0: No wait</li> <li>• A positive number: The time to wait in milliseconds.</li> </ul>
<b>data format</b>	
<b>Match Conditions</b>	An array of match conditions in a key / value format. Select a key from the drop down box in the Key's <b>Value</b> column and provide a value. MQMO_NONE does not require a value.

## Checkpoint Properties

Property	Description
<b>JMSMessageID</b>	A unique ID for the message.
<b>JMSTimeStamp</b>	The time and date of the message.
<b>JMSCorrelationID</b>	The message's correlation ID.  <b>Tip:</b> Use in conjunction with the JMS receive message selector.
<b>JMSReplyTo</b>	The contents of the <b>Reply To</b> field.
<b>JMSDestination</b>	The message's destination.
<b>JMSDeliveryMode</b>	The message's delivery mode. Use the scroller to select a value.
<b>JMSRedelivered</b>	Indicates whether the message was redelivered. <b>False</b> by default.
<b>JMSType</b>	A string describing the JMS type.
<b>JMSExpiration</b>	The message's expiration date.
<b>JMSPriority</b>	The message's priority in comparison to other messages. Use the scroller to select a value.
<b>JMS properties</b>	An array of message properties in the structure of keys and values.

Property	Description
<b>JMSMessageID</b>	A unique ID for the message.
	<b>Tip:</b> Use the <b>Number of elements</b> drop-down to specify the number of properties, or click the <b>Add</b> button to add elements.
<b>Message body</b>	The message content.

## Put Message to MQ Queue Activity

### Relevant for: API testing only

This activity enables you to add a message to a selected MQ Queue.

You must put a **Connect to MQ Queue Manager** step before this step.

The following properties are available for this activity:

### Input Properties

Property	Description
<b>MQManager</b>	A connection expression linked to the most recent <b>Connect to MQ Queue Manager</b> step.
<b>Put queue name</b>	The name of the MQ queue on which to put the message.
<b>Queue Access Options</b>	A drop down of built-in MQ queue access options.  <b>Note:</b> To access additional options, use an event handler. For details, see " <a href="#">Writing Event Handlers for API Test Steps</a> " on page 751.
<b>Message Put Options</b>	A drop down of built-in message put options.  <b>Note:</b> To access additional options, use an event handler. For details, see " <a href="#">Writing Event Handlers for API Test Steps</a> " on page 751.
<b>Priority</b>	The priority of the message rated from 0 through 9 (highest). A message with a higher priority will be taken from the queue before messages with a lower priority.
<b>Correlation ID</b>	The correlation ID of the message.
<b>Message format</b>	The format of the message being sent: <b>String</b> or <b>UTF</b> .
<b>Character Set</b>	The character set code to use when sending a message.  <b>Note:</b> This property is only relevant when the message format is set to <b>String</b> .
<b>Message body</b>	The body of the message to put on the queue.

Property	Description
<b>Message Properties</b>	An array of Put message properties in a Key / Value format.

## Checkpoint Properties

Property	Description
<b>MessageId</b>	A message identifier of the message being sent.
<b>PutTime</b>	The date and time the message was put.
<b>UserId</b>	The sender's user ID.

## Get Message from MQ Queue Activity

### Relevant for: API testing only

This activity enables you to receive a message from a selected MQ Queue.

You must put a **Connect to MQ Queue Manager** step before this step.

The following properties are available for this activity:

### Input Properties

Property	Description
<b>MQManager</b>	A connection expression linked to the most recent <b>Connect to MQ Queue Manager</b> step.
<b>Get Queue Name</b>	The name of the MQ queue from which to get the message.
<b>Queue Access Options</b>	A drop down of built-in MQ queue access options.  <b>Note:</b> To access additional options, use an event handler. For details, see " <a href="#">Writing Event Handlers for API Test Steps</a> " on page 751.
<b>Message Get Options</b>	A collection of built-in Message get options.  <b>Note:</b> To access additional options, use an event handler. For details, see " <a href="#">Writing Event Handlers for API Test Steps</a> " on page 751.
<b>Wait interval</b>	The maximum time in milliseconds to wait for the message to arrive, using the following values: <ul style="list-style-type: none"> <li>• -1: Unlimited wait time</li> <li>• 0: No wait</li> <li>• A positive number: The time to wait in milliseconds.</li> </ul>
<b>Message data format</b>	The format of the message being retrieved: <b>String</b> or <b>UTF</b> .

Property	Description
<b>MQManager</b>	A connection expression linked to the most recent <b>Connect to MQ Queue Manager</b> step.
<b>Match Condition</b>	An array of match conditions in a key / value format. Select a key from the drop down box in the Key's <b>Value</b> column and provide a value. MQMO_NONE does not require a value.

## Checkpoint Properties

Property	Description
<b>MessageId</b>	The message identifier of the retrieved message.
<b>CorrelationId</b>	The correlation identifier of the retrieved message.
<b>GroupId</b>	An identifier of the message group to which the physical message belongs.
<b>MsgSeqNumber</b>	The sequence number of a logical message within a group.
<b>PutTime</b>	The date and time the PUT action was executed for the message.
<b>Priority</b>	The priority of the message rated from 0 through 9 (highest). A message with a higher priority will be taken from the queue before messages with a lower priority.
<b>UserId</b>	The User ID of the user who submitted the message.
<b>MessageBody</b>	A string representing the UTF content of the retrieved message.

## Put and Get Message from MQ Queue

### Relevant for: API testing only

This activity enables you to add and receive a message from a selected MQ Queue.

You must put a **Connect to MQ Queue Manager** step before this step.

The following properties are available for this activity:

### Input Properties

Property	Description
<b>MQManager</b>	A connection expression linked to the most recent <b>Connect to MQ Queue Manager</b> step.
<b>Auto Match Correlation ID</b>	Correlates the sent and received messages by automatically adding a match condition. This condition indicates that the <b>Correlation ID</b> of the received message should equal the <b>Message ID</b> of the sent message.
<b>Put Queue Name</b>	The name of the MQ queue upon which to put the message.
<b>Put Queue Access</b>	A collection of built-in MQ queue access options.

Property	Description
<b>MQManager</b>	A connection expression linked to the most recent <b>Connect to MQ Queue Manager</b> step.
<b>Options</b>	<p><b>Note:</b> To access additional options, use an event handler. For details, see "<a href="#">Writing Event Handlers for API Test Steps</a>" on page 751.</p>
<b>Message Put Options</b>	<p>A collection of built-in message put options.</p> <p><b>Note:</b> To access additional options, use an event handler. For details, see "<a href="#">Writing Event Handlers for API Test Steps</a>" on page 751.</p>
<b>Put priority</b>	The priority of the message to be submitted.
<b>Put correlation ID</b>	A correlation identifier for the message to be submitted.
<b>Put Message Format</b>	The format of the message: being sent: <b>String</b> or <b>UTF</b> .
<b>Character Set</b>	
<b>Message Body</b>	A string representing the UTF content of the submitted message.
<b>Message properties</b>	An array of Put message properties in a key / value format.
<b>Get Queue Name</b>	The name of the MQ queue from which to get the message.
<b>Get Queue Access Options</b>	<p>A collection of built-in MQ queue access options.</p> <p><b>Note:</b> To access additional options, use an event handler. For details, see "<a href="#">Writing Event Handlers for API Test Steps</a>" on page 751.</p>
<b>Message Get Options</b>	<p>A collection of built-in options that control the Get operation.</p> <p><b>Note:</b> To access additional options, use an event handler. For details, see "<a href="#">Writing Event Handlers for API Test Steps</a>" on page 751.</p>
<b>Wait interval</b>	<p>The maximum time in milliseconds to wait for the message to arrive, using the following values:</p> <ul style="list-style-type: none"> <li>• -1: Unlimited wait time</li> <li>• 0: No wait</li> <li>• A positive number: The time to wait in milliseconds.</li> </ul>
<b>Get Message Format</b>	The format of the received message: <b>String</b> or <b>UTF</b> .
<b>Match Condition</b>	An array of Get Message match conditions in a key / value format. Select a key from the drop down box in the Key's <b>Value</b> column and provide a value. MQMO_NONE does not require a value.

## Checkpoint Properties

Property	Description
<b>MessageId</b>	The message identifier of the retrieved message.
<b>CorrelationId</b>	The correlation identifier of the retrieved message.
<b>GroupId</b>	An identifier of the message group to which the physical message belongs.
<b>MsgSeqNumber</b>	The sequence number of a logical message within a group.
<b>PutTime</b>	The date and time the PUT action was executed for the message.
<b>Priority</b>	The priority of the message rated from 0 through 9 (highest). A message with a higher priority will be taken from the queue before messages with a lower priority.
<b>UserId</b>	The User ID of the user who submitted the message.
<b>MessageBody</b>	A string representing the UTF content of the retrieved message.

## Subscribe to MQ Topic Activity

### Relevant for: API testing only

This activity enables you to subscribe to a selected MQ Queue topic.

You must put a **Connect to MQ Queue Manager** step before this step.

The following properties are available for this activity:

## Input Properties

Property	Description
<b>MQManager</b>	A connection expression linked to the most recent <b>Connect to MQ Queue Manager</b> step.
<b>Topic name</b>	The name of the MQ topic from which subscribers can access the message.
<b>Subscription name</b>	An automatically generated subscription name.
<b>Topic Access Options</b>	A collection of built-in MQ topic access options.  <b>Note:</b> To access additional options, use an event handler. For details, see " <a href="#">Writing Event Handlers for API Test Steps</a> " on page 751.

## Checkpoint Properties

Property	Description
<b>Result</b>	The Subscription name entered manually or generated automatically.

Property	Description
	<p><b>Notes:</b></p> <ul style="list-style-type: none"> <li>Although this does not appear in the <b>Checkpoints</b> grid, it is visible as an output property in the Select Link Source Dialog Box.</li> <li>This property can be used as an input value for the subscription name in the <b>Unsubscribe from MQ Topic</b> and <b>Receive Message from MQ Topic</b> steps.</li> </ul>

## Unsubscribe from MQ Topic Activity

### Relevant for: API testing only

This activity enables you to unsubscribe from a previous subscription to an MQ Queue topic.

You must put a **Connect to MQ Queue Manager** step before this step.

The following properties are available for this activity:

### Input Properties

Property	Description
<b>MQManager</b>	A connection expression linked to the most recent <b>Connect to MQ Queue Manager</b> step.
<b>Subscription name</b>	The subscription from which to unsubscribe. This can be a literal string or an expression linked to the output of a <b>Subscribe to MQ Topic</b> step.

### Checkpoint Properties

Property	Description
<b>Result</b>	A boolean variable indicating whether the step succeeded.

## Publish Message to MQ Topic Activity

### Relevant for: API testing only

This activity enables you to publish a message to a selected MQ Queue topic.

You must put a **Connect to MQ Queue Manager** step before this step.

The following properties are available for this activity:

## Input Properties

Property	Description
<b>MQManager</b>	A connection expression linked to the most recent <b>Connect to MQ Queue Manager</b> step.
<b>Topic Name</b>	The name of the topic on which to publish the message.
<b>Topic Access Options</b>	A drop down of the built-in options that control the opening of the topic.  <b>Note:</b> To access additional options, use an event handler. For details, see " <a href="#">Writing Event Handlers for API Test Steps</a> " on page 751.
<b>Message Put Options</b>	A drop down of built-in options that control the Put operation, such as MQPMO_SYNCPOINT.  <b>Note:</b> To access additional options, use an event handler. For details, see " <a href="#">Writing Event Handlers for API Test Steps</a> " on page 751.
<b>Priority</b>	The priority of the message rated from 0 through 9 (highest). A message with a higher priority will be taken from the queue before messages with a lower priority.
<b>Correlation ID</b>	The UserId of the user who submitted the message.
<b>Message Format</b>	The format of the message being published: <b>String</b> or <b>UTF</b> .
<b>Character Set</b>	The character set to use when sending this message.  <b>Note:</b> This property is only relevant when the Message Format is set to String.
<b>Message body</b>	The message to publish.
<b>Message Properties</b>	An array of message properties in a Key / Value format.

## Checkpoint Properties

Property	Description
<b>MessageId</b>	A message identifier of the submitted message.
<b>PutTime</b>	The date and time the message was put.
<b>UserId</b>	The UserId of the user who submitted the message.

## Receive Message from MQ Topic Activity

### Relevant for: API testing only

This activity enables you to receive a message from a selected MQ Queue topic.

You must put a **Connect to MQ Queue Manager** step before this step.

The following properties are available for this activity:

### Input Properties

Property	Description
<b>MQManager</b>	A connection expression linked to the most recent <b>Connect to MQ Queue Manager</b> step.
<b>Subscription Name</b>	The subscription through which to receive topic messages. This can be a literal string or an expression linked to the output of a <b>Subscribe to MQ Topic</b> step.
<b>Message Get Options</b>	A drop down of built-in message get options.  <b>Note:</b> To access additional options, use an event handler. For details, see " <a href="#">Writing Event Handlers for API Test Steps</a> " on page 751.
<b>Wait interval</b>	The maximum time in milliseconds to wait for the message to arrive, using the following values: <ul style="list-style-type: none"><li>• -1: Unlimited wait time</li><li>• 0: No wait</li><li>• A positive number: The time to wait in milliseconds.</li></ul>
<b>Message Format</b>	The format of the message being retrieved: <b>String</b> or <b>UTF</b> .
<b>Match Condition</b>	An array of match conditions in a key / value format. Select a key from the drop down box in the Key's <b>Value</b> column and provide a value. MQMO_NONE does not require a value.

### Checkpoint Properties

Property	Description
<b>MessageId</b>	The message identifier of the retrieved message.
<b>CorrelationId</b>	The correlation identifier of the retrieved message.
<b>GroupId</b>	An identifier of the message group to which the physical message belongs.
<b>MsgSeqNumber</b>	The sequence number of a logical message within a group.
<b>PutTime</b>	The date and time the message was put.
<b>Priority</b>	The priority of the message rated from 0 through 9 (highest). A message with a higher priority will be taken from the queue before messages with a lower priority.
<b>UserId</b>	The User ID of the user who submitted the message.
<b>MessageBody</b>	A string representing the UTF content of the retrieved message.

## HP Automated Testing Tools Activities

### Relevant for: API testing only

This activity group contains activities that allow you implement unified testing plan. For task details, see ["How to Call External Tests or Actions" on page 426](#).

- **Call API Action or Test.** Calls an API or Service Test test or action when the current Test Flow or loop reaches this step.
- **Call GUI Action or Test.** Calls a GUI test or action when the current Test Flow or loop reaches this step. Do not insert a call to a GUI Test action or test that contains a call to a API Test action or test, as this can cause unexpected behavior.
- **Call Virtual User Generator script.** Runs a LoadRunner Virtual User Generator (VuGen) script.

### General Properties

The following general properties are available for these activities:

Property	Activity	Description
<b>Test path</b>	<b>Call API Action or Test</b>	
	<b>Call GUI Action or Test</b>	
<b>Action name</b>	<b>Call API Action or Test</b> <b>Call GUI Action or Test</b>	The name of the action being called. This field is read-only. To select an action, click the <b>Select Action or Test</b> button in the <b>Input/Checkpoints</b> tab.
<b>Description</b>	<b>All</b>	An editable field describing the call.
<b>Result Directory</b>	<b>Call API Action or Test</b>	The location in which to store the run results. This field is editable.
<b>Script path</b>	<b>Call Virtual User Generator Script</b>	The path of the Virtual User Generator-LoadRunner script. This field is read-only. To select a script, click the <b>Select Virtual User Generator Script</b> button in the Input/Checkpoint view.

## SAP Activities

### Relevant for: API testing only

This activity group contains the activity related to SAP IDoc status.

- **Get IDOC Status.** Retrieves the status of an IDoc.

This section describes the built-in SAP activity. For information about imported SAP activities representing RFCs and IDocs, see ["How to Create an SAP API Test Step" on page 425](#).

## General Properties

To view these properties, click the **General** tab in the Properties pane.

Property	Description
<b>Timeout</b>	The maximum time allowed for the function to respond in milliseconds. <b>Default:</b> 100000
<b>ConnectionInfo</b>	The SAP Connection information for the selected step: <ul style="list-style-type: none"> <li><b>Server.</b> The name or IP address of the server.</li> <li><b>System number.</b> The server's System Number.</li> <li><b>Client.</b> The client for this SAP connection.</li> <li><b>Username.</b> The username for this SAP connection.</li> <li><b>Password.</b> The password for this SAP connection.</li> </ul>
<b>Function name</b>	The name of the RFC (for RFCs only).
<b>Expect Exception</b>	Indicates whether or not to expect an exception when running the function: true or false (for RFCs only).
<b>IDocController</b>	IDoc Controller properties (for IDocs only): <ul style="list-style-type: none"> <li><b>Message Type, Basic Type, and Extension.</b></li> <li><b>Sender:</b>Port, Partner number, and Partner type.</li> <li><b>Receiver:</b>Port, Partner number, and Partner type.</li> </ul>
<b>Transaction</b>	Indicates whether or not the RFC is part of a transaction or not: true or false (for RFCs only).

## Input Properties

Property	Description
<b>IDoc Number</b>	The IDoc number for the IDoc whose status you want to retrieve.

## Checkpoint Properties

The checkpoint name corresponds to the SAP property ID.

Property	Description
<b>MANDT</b>	The client.
<b>DOCNUM</b>	The IDoc number.
<b>LOGDAT</b>	The date of the status information.
<b>LOGTIM</b>	The time of the status information.
<b>COUNTR</b>	The IDoc status counter.

Property	Description
<b>CREDAT</b>	The creation date of the status record.
<b>CRETIM</b>	The creation time of the status record.
<b>STATUS</b>	The status of the IDoc.
<b>UNAME</b>	The user name.
<b>REPID</b>	The program name.
<b>ROUTID</b>	The name of the subroutine or function module.
<b>STACOD</b>	The status code.
<b>STATXT</b>	The status code textual representation.
<b>SEGNUM</b>	The SAP segment number.
<b>SEGFLD</b>	The field name in the SAP segment.
<b>STAPA1,2,3,4</b>	Status parameters 1, 2, 3, and 4.
<b>STATYP</b>	The system message type: A, W, E, S, or I.
<b>STAMQU</b>	The status message qualifier.
<b>STAMID</b>	The status message ID.
<b>STAMNO</b>	The status message number.
<b>TID</b>	The transaction ID.
<b>APPL_LOG</b>	The application log.

## XML Activities

### Relevant for: API testing only

This activity group contains activities related to XML files:

- **Transform XML.** Converts an XML file to another structure based on the specified XSLT. For details, see "[Transform XML Activity](#)" on the next page.
- **Compare XML.** Performs a comparison between two XML strings. For details, see "[Compare XML Activity](#)" on the next page.
- **XML to String.** Converts an XML file to a text string. For details, see "[XML to String Activity](#)" on page 496.
- **String to XML.** Converts a string to XML structure. For details, see "[String to XML Activity](#)" on page 496.
- **Validate XML.** Validates an XML file against an XSD schema. For details, see "[Validate XML Activity](#)" on page 497.

The **XML to String** and **String to XML** activities are useful when you need to access the output in a specific format—either as string or XML.

For example, suppose your test contains a Web Service call whose output is XML. However, you need to use the output as an input for the next step, which requires a textual string—not XML.

You can use the **XML to String** activity to create a string that is compatible with the API test step.

After the test step, you can use a **String to XML** activity to convert the output back to XML, making it available for linking, and checkpoints.

## Transform XML Activity

### **Relevant for: API testing only**

This activity enables you to transform a selected XML string.

The following properties are available for this activity:

### **Input Properties**

Property	Description
<b>XML String</b>	The XML to transform.
<b>Import XSLT to test folder</b>	Copies the file containing the transformed XML (the .xslt file) to the test folder.
<b>XSLT folder</b>	The full path of the folder in which to place the file containing the transformed XML,

### **Checkpoint Properties**

Property	Description
<b>Transformed XML</b>	The expected XML string after its transformation.

## Compare XML Activity

### **Relevant for: API testing only**

This activity enables you to compare two selected XML strings.

The following properties are available for this activity:

### **Input Properties**

Property	Description
<b>XML String 1</b>	The first XML string to compare.
<b>XML String 2</b>	The second XML string to compare
<b>Ignore</b>	Instructs UFT to ignore the order of XML elements when doing the comparison. Select True to ignore the

<b>element order</b>	elements or False to notice the element order during the comparison.
<b>Ignore namespaces</b>	Instructs UFT to ignore the XML namespaces of XML elements in the strings. Select True to ignore the namespaces or False to consider the namespaces during the comparison.
<b>Ignore XPaths</b>	A list of all the XPaths that UFT ignores during the comparison. Click the <b>Add Array</b> button  to add XPath elements to ignore. You can add as many XPath elements as needed.

## Checkpoint Properties

Property	Description
<b>Are Equal</b>	Checks if the values of the <b>XML String 1</b> and <b>XML String 2</b> properties are equal. Select True if the strings are expected to be equal, or False if they are expected to be different.

## XML to String Activity

### Relevant for: API testing only

This activity enables you to convert a selected XML string into a regular text string.

The following properties are available for this activity:

### Input Properties

Property	Description
Source string	The source XML to convert into a string

The available checkpoint properties depend on the XML loaded in the Checkpoints section.

## String to XML Activity

### Relevant for : API testing only

This activity enables you to convert a selected text string into XML.

The following properties are available for this activity:

The available input properties depend on the XML loaded in the Input section.

## Checkpoint Properties

Property	Description
Source string	The expected XML after the conversion.

## Validate XML Activity

### Relevant for: API testing only

This activity enables you to validate a selected XML string against a specified XML schema and the types of values returned in the XML schema.

The following properties are available for this activity:

### Input Properties

Property	Description
<b>XML string</b>	The XML string to validate
<b>Import XSD to test folder</b>	Instructs UFT to import the file (the .xsd file) containing the transformed XML to the folder containing your test.
<b>XSD file</b>	The .xsd file against which the validation is performed.

### Checkpoint Properties

Property	Description
<b>Valid</b>	Indicates whether the XML string specified in the <b>XML string</b> property is valid according to the validation performed against the file specified in the <b>XSD file</b> property.
<b>Errors</b>	The expected error in the validation.

## Web Sockets Activities

This group contains activities related to Web Socket communication. This enables you to test application processes that use communication between two Web sockets.

- [OpenSocket Activity](#)
- [CloseSocket Activity](#)
- [SendMessage Activity](#)
- [ReceiveMessage Activity](#)

## Open Socket Activity

This activity enables you to open a connection to a designated Web socket.

The following properties are available for this activity:

## Input Properties

Property	Description
URL	The URL of the Web socket to access. The format of the URL is ws://.

## Checkpoint Properties:

Property	Description
Result	Indicates whether the connection is active (true) or closed (false).

## Close Socket Activity

This activity enables you to close a connection to a designated Web socket.

The following properties are available for this activity:

## Input Properties

Property	Description
SocketID	<p>The unique ID of the Web Socket with which you are communicating.</p> <p>This ID is provided by linking to the <b>SocketID</b> property (found in the <b>General</b> tab  of an <b>OpenSocket</b> activity) of an OpenSocket activity.</p>

## Checkpoint Properties:

Property	Description
Result	Indicates whether the connection is active (true) or closed (false).

## Send Message Activity

This activity enables you to send a message to another Web socket.

The following properties are available for this activity:

## General Properties

Property	Description
SocketID	<p>The unique ID of the Web Socket with which you are communicating.</p> <p>This ID is provided by linking to the <b>SocketID</b> property (found in the <b>General</b> tab  of an <b>OpenSocket</b> activity) of an OpenSocket activity.</p>

## HTTP properties

Property	Description
Request Body	The formatted request body that you want to send to the other Web socket. You can send a message as plain <b>Text</b> , <b>XML</b> , or <b>JSON</b> .

There are no checkpoint properties for the Send Message Activity.

## Receive Message Activity

This activity enables you to receive a message from another Web socket.

The following properties are available for this activity:

### General Properties

Property	Description
SocketID	The unique ID of the Web Socket with which you are communicating.  This ID is provided by linking to the <b>SocketID</b> property (found in the <b>General</b> tab  of an <b>OpenSocket</b> activity) of an OpenSocket activity.
<b>Response Body Extension</b>	The file extension of the expected response.

## HTTP Properties

Property	Description
<b>Received Message Body</b>	The formatted response that you want to receive from the other Web socket. You can receive a message as plain <b>Text</b> , <b>XML</b> , or <b>JSON</b> .

# Troubleshooting and Limitations - Standard Activities

**Relevant for: API testing only**

Activity Type	Activity and Limitation
<b>System</b>	<ul style="list-style-type: none"><li>• <b>End Program.</b> You cannot specify <b>Window Title</b> as an input method for a windowless process, even if you are using a wildcard expression.</li><li>• <b>End Program.</b> If you are running on a 64-bit machine, the <b>End Program</b> activity will not be able to terminate 64-bit applications. However, it can terminate other 32-bit applications.</li></ul>
<b>Java</b>	<ul style="list-style-type: none"><li>• <b>Call Java Class.</b> Supports only Java primitive types.</li><li>• <b>Call Java Class.</b> Once you select a Java file for the call, the <b>Java Class</b> button is disabled. As a result, you cannot replace or update the Java file. <b>Workaround:</b> Remove the step containing the <b>Call Java Class</b> step and add a new one using the new Java file.</li><li>• <b>Call Java Class.</b> Java code loaded by the JVM (Java Virtual Machine) cannot be modified or updated when the JVM is running.</li></ul>
<b>Network</b>	<ul style="list-style-type: none"><li>• <b>HTTP Request/Receiver.</b> Nested transactions are not supported. <b>Workaround:</b> Add a new loop activity within an existing transaction. Add the new transaction steps to the newly created loop. Make sure to set the loop iteration to 1.</li><li>• <b>HTTP Request/SOAP Request.</b> XML files that use XSL are not supported.</li><li>• <b>SOAP Request.</b> Switching between <b>Text</b> and <b>Grid</b> views, may cause the grid to display an element added below the Body's <b>Any</b> node, under the Header's <b>Any</b> node. <b>Workaround:</b> Open the <b>Text</b> view to view the correct XML..</li><li>• <b>SOAP Request.</b> By default, UFT validates the request received in a SOAP Request step using the SOAP 1.1 schema. If you expect your response to use the SOAP 1.2 schema, the validation will fail. <b>Workaround:</b> Import the SOAP 1.2 schema for your SOAP Request step's response body. This schema is available with the UFT installation at &lt;UFT installation folder&gt;\Addins\ServiceTest\WSImportTechnology\envelop1.2.xsd.</li></ul>
<b>Database</b>	<ul style="list-style-type: none"><li>• Nested transactions are not supported in database transaction activities. <b>Workaround:</b> Add a new loop activity within an existing transaction. Add the new transaction steps to the newly created loop. Make sure to set the loop iteration to 1.</li><li>• Databases which are supported by ODBC, but not by OLEDB, cannot be accessed by UFT.</li></ul>
<b>FTP</b>	<ul style="list-style-type: none"><li>• <b>FTP:</b> When working with FTP activities that specify paths, you need to enter the full path.</li><li>• <b>FTP Download:</b> When downloading from Linux servers, you cannot download an empty folder.</li></ul>
<b>IBM Websphere MQ</b>	<ul style="list-style-type: none"><li>• <b>Get Message from MQ Queue.</b> The MQGMO_MARK_SKIP_BACKOUT option is not supported.</li><li>• <b>Put Message to MQ Queue, Publish Message to MQ Topic.</b> The message body should not exceed 64K bytes. If it exceeds this size, the execution issues a MQRC_CONVERTED_STRING_TOO_BIG status. This is a limitation of IBM MQ.</li><li>• <b>MQ Steps.</b> Automatic linking of IBM Websphere MQ steps to the most recent <b>Connect to MQ Queue</b></li></ul>

	<b>Manager</b> connection, is supported only when the steps are on the same level in the container, or if the connection step is in a parent container. If the connection step is in a leaf container and the step using the connection is in a parent container or in another leaf, UFT does not create an automatic link.  <b>Workaround:</b> Manually link to a MQManager property using the Select Link Source dialog box.
<b>JSON</b>	If your JSON string contains non-ASCII characters, you should save this file with UFT-8 encoding. Otherwise, the characters in your file may not display correctly in UFT.
<b>Load Testing</b>	<ul style="list-style-type: none"> <li>Related data mapping in a load-enabled test is not supported for the LoadRunner parameter advance policy of <b>Each Occurrence</b>.</li> <li>Tests created as Business Process Testing (BPT) components, cannot be used in load testing.</li> <li>If your tests use IBM's MQ client, make sure to install the MQ client on all machines running these tests.</li> <li>You cannot run tests containing actions or calls to other tests on a remote load generator. This limitation does not apply when running the test on a local load generator.</li> <li>The data assignment method, <b>Use a unique value for each Virtual User when load testing</b> is not supported in all environments.</li> </ul>
<b>Web Sockets</b>	Using a web socket open between actions in a test is not supported.
<b>XPath Activity Checkpoints</b>	XPath aggregate functions are not supported.

# Chapter 42: Custom Activities

**Relevant for: API testing only**

This chapter includes:

• <a href="#">Custom Activity Overview</a> .....	.503
• <a href="#">Activity Sharing</a> .....	.506
• <a href="#">How to Perform Activity Sharing</a> .....	.506
• <a href="#">Negative Testing of Web Services</a> .....	.507
• <a href="#">Passing REST Service Properties</a> .....	.508
• <a href="#">Exposing REST Service Properties</a> .....	.511
• <a href="#">How to Import a WSDL-Based Web Service</a> .....	.512
• <a href="#">How to Create a REST Service</a> .....	.515
• <a href="#">How to Send and Receive a JSON Request for a REST Service</a> .....	.521
• <a href="#">How to Import a Web Application Service</a> .....	.522
• <a href="#">How to Import a Network Capture File</a> .....	.525
• <a href="#">How to Import and Create a .NET Assembly API Test Step</a> .....	.526
• <a href="#">Troubleshooting and Limitations - Custom Activities</a> .....	.529

# Custom Activity Overview

## Relevant for: API testing only

In addition to the **Standard** activities, you can create or import other services and activities. These activities, displayed under the Toolbox pane's **Local Activities** node, include Web Services, REST Services, Web Application Services, .NET assembly operations, SAP IDocs/RFCs whose contract/document or DLLs you import, or custom activities created from a Network Capture.

This section includes:

- "[Web Service Activities](#)" below
- "[REST Service Activities](#)" below
- "[Swagger API REST Services](#)" on the next page
- "[Web-Application Services](#)" on the next page
- "[Network Capture Activities](#)" on the next page
- "[.NET Assembly Activities](#)" on page 505
- "[SAP Activities](#)" on page 505

## Web Service Activities

To create Web service activities, you must import a WSDL file. This file provides a structure for the test, by describing the service in terms of its elements, argument values, and properties.

The WSDL import supports both Document/Literal and RPC type Web services.

In **Document/Literal** Web services, the client sends standard XML documents to the server. The server application is responsible for mapping the server objects (properties, method calls, and so forth) and the values in XML documents. The data is serialized according to a given schema, so it can be validated against the schema. For Document/Literal type Web services, the Properties pane displays the input and output properties in a grid, allowing you to assign values for each property independently.

In **RPC** type Web services, the WSDL file and SOAP body contain the complete operation name, its input and output properties, and their values. There is no schema for this type of service and it is not supported by the WS-I conformance standard. The Properties pane does not display the input and output properties for RPC type services.

If your service document is unique and cannot be imported in the normal way, you can use the SOAP Request activity to send a manual SOAP request to the server.

For details about importing a Web service, see "[How to Import a WSDL-Based Web Service](#)" on page 512.

## REST Service Activities

To create REST services, resources, and methods you must define the metadata manually using the REST Service Editor. Based on how you describe the metadata, UFT creates a hierarchy of **Service**,

**Resource, and Method.** The services, resources, and methods are then stored as a prototype activity within your test and can be used as test steps (like any other Toolbox pane activity).

UFT also enables you to update REST service definitions and resolve service definition conflicts through the Resolve REST Method Steps Wizard.

In addition, you can also define properties and parameters for your REST service at all levels of the hierarchy. You can then pass these properties or parameters from the service and resource levels to the resource and method levels. For details, see "["Passing REST Service Properties" on page 508](#)".

For task details, see "[How to Create a REST Service](#)" on page 515.

## Swagger API REST Services

You can import your REST APIs created with Swagger. You define the service's metadata using Swagger tools. Once you have created it, UFT can then import the definition of your service, either from a locally saved JSON file or a URL.

UFT automatically imports the service's definition, and creates the service model in the same manner as a REST service (with the Service/Resource/Method hierarchy). The service model, including its resources and methods, are added to the Local Activities node of the Toolbox pane, and then can be used in any test.

## Web-Application Services

Web Application services provide a description of a HTTP-based Web application in an XML format, saved in a Web Application Description Language (WADL) file. The WADL file describes the resources provided by a service and the methods used to access the service.

Like a Web Service, you import a Web application service into UFT. The resources and methods are then displayed in a hierarchy like a REST service, in a **Service/Resource/Method** hierarchy.

The URL for a Web application is defined in the XML of the WADL file. You can, however, define other HTTP properties and add input and output parameters for an activity's methods.

**Note:** If you import a WADL from a URL, you cannot edit the WADL's properties manually.

Like REST services, you can define parameters and their values at all levels of the Web Application hierarchy. You can then pass these parameter values to lower levels of the hierarchy.

The imported Web Application service methods serve as a prototype for test steps. You can modify the parameter values of a method after dragging a method to the canvas.

## Network Capture Activities

Network capture activities enable you to create test steps by recording network traffic. Importing a network capture file provides you with another way to create test steps measuring the network activity of your application or Web service. Instead of using the standard Network activities to design steps for your application's network processes, you can perform a network capture and use the captured

information as a basis for your test.

Using a network capture program, you capture the network traffic for your application or Web service. The network capture program then saves the file containing the network traffic. You then import this file into UFT.

UFT takes the TCP network stream and creates test steps based on the request and response information for each TCP stream in the network traffic capture.

Based on the request and response information, UFT creates a test activity differently:

- If the TCP stream request and response is compatible with or matches an already existing Web service, UFT creates a **Web Service** step.
- If the TCP stream request has a SOAP request structure, UFT creates a **SOAP Request** step.
- If the TCP stream is not similar to an existing Web service method or a SOAP request transaction, UFT creates a **HTTP Request** step.

These activities are not stored in the Toolbox pane. If you need to reuse the steps in your test, you can reimport the network capture file or cut and paste the existing steps into your test.

For task details, see "[How to Import a Network Capture File](#)" on page 525.

## .NET Assembly Activities

The .NET importer lets you create activities for testing APIs in the form of .NET assemblies. You can interface with the types defined in the assembly.

You begin by importing the .NET assembly into your test. The Toolbox pane then displays the assembly as an activity and you add a .NET activity on to the canvas.

When you import a .NET assembly, it saves a local copy of the assembly with the test. This makes the test portable and allows you to copy it to another machine. If the assembly calls other assemblies, the test may not run until you copy the additional assemblies to the new machine.

For task details, see "[How to Import and Create a .NET Assembly API Test Step](#)" on page 526.

## SAP Activities

UFT enables you to create SAP activities by importing SAP Intermediate Documents (IDoc) and Remote Function Calls (RFC).

These activities can be useful for testing the SAP server response in several common scenarios:

- Sending an IDoc to an SAP server, and confirming that the IDoc was sent
- Checking an IDoc's status on the SAP server
- Calling an RFC in SAP and ensure that it returned the expected results

These activities can also be useful when upgrading a system to verify that the integration patterns are still functional, such as Aggregator, Enricher, Router, Translation, Bridge, Splitter, and so forth.

For task details, see "[How to Create an SAP API Test Step](#)" on page 425.

# Activity Sharing

## **Relevant for: API testing only**

The activity sharing feature enables you to save locally stored services to a repository, so that they will be available for other tests.

You can specify a repository on the file system or in ALM. The next time you create a test, you can access the service's activities from the repository instead of reimporting or recreating them.

If the resource (WSDL or REST service) becomes unavailable, the canvas displays alerts on the steps that use the resource. If you run the test when its resource is not available from its original source, it uses a copy of the test stored in its cache. By clicking the alert, you can reload the steps when the resource becomes available again.

The Toolbox pane detects version updates for activities stored in a repository. When it detects a discrepancy between the step and the Toolbox pane activity, it displays an alert for the step. By clicking the alert, you can automatically update the resource from its source.

## How to Perform Activity Sharing

### **Relevant for: API testing only**

This task describes how to save activities to a repository, update them, and view their properties.

This task includes the following steps:

- "[Connect to ALM](#)" below
- "[Set up the repository paths](#)" below
- "[Import a WSDL or create a REST service](#)" on the next page
- "[Move the activity to a repository](#)" on the next page
- "[Refresh the activity - optional](#)" on the next page
- "[Update the activity - optional](#)" on the next page

### **1. Connect to ALM**

If you want to work with a repository on ALM, connect to the desired ALM server.

### **2. Set up the repository paths**

- a. Select **Tools > Options > API Testing tab > General node**.
- b. In the Activity Repositories section, click the **Browse** button and navigate to the location in the file system or in ALM.
- c. Click **OK**.

### 3. Import a WSDL or create a REST service

Import a WSDL file or create a REST service method. The Toolbox pane shows the services in the **Local Activities** node. By default, the test stores these activities in its `EmbeddedJavaResources` subfolder.

### 4. Move the activity to a repository

Right-click the service node, and select **Move to > File System Activities** or **ALM Activities**. This moves the service from Local Activities to **File System Activities** or **ALM Activities** in the Toolbox pane. The service is also removed from the test's `EmbeddedJavaResources` subfolder and placed in the repository folder.

The next time you create a test, these activities will be accessible from the **File System Activities** or **ALM Activities** nodes.

### 5. Refresh the activity - optional

To reload the WSDL from its stored location, right-click the service node, and select **Refresh** . This is useful in cases when the WSDL is stored in a shared location and may have been updated by another user.

### 6. Update the activity - optional

To reimport the WSDL from its original location, for example to revert back to the original version, right-click the service node and select **Update WSDL**, or click the **Update WSDL** button in the Toolbox pane .

## Negative Testing of Web Services

### Relevant for: API testing only

When performing a functional test for your Web Service, you should approach the testing in a variety of ways. The most common type of testing is called **Positive Testing**—checking that the service does what it was designed to do.

In addition, you should perform **Negative Testing**, to confirm that the application did not perform a task that it was not designed to perform. In those cases, you need to verify that the application issued an appropriate error—a SOAP Fault.

To illustrate this, consider a form accepting input data—you apply positive testing to check that your Web Service has properly accepted the name and other input data. You apply negative testing to make sure that the application detects an invalid character, for example a letter character in a telephone number.

When your service sends requests to the server, the server responds in one of the following ways:

- **SOAP Result.** A SOAP response to the request.
- **SOAP Fault.** A response indicating that the SOAP request was invalid. Negative Testing applies only to SOAP faults.
- **HTTP Error.** An HTTP error, such as Page Not Found, unrelated to Web Services.

UFT can check for a standard SOAP result or a SOAP fault response. For example, if your Web Service attempts to access a Web page that cannot be found, it will issue a 404 HTTP error. Using negative testing you indicate that you expect a SOAP fault. In this case, the test run will fail if the service accesses a *valid* Web page.

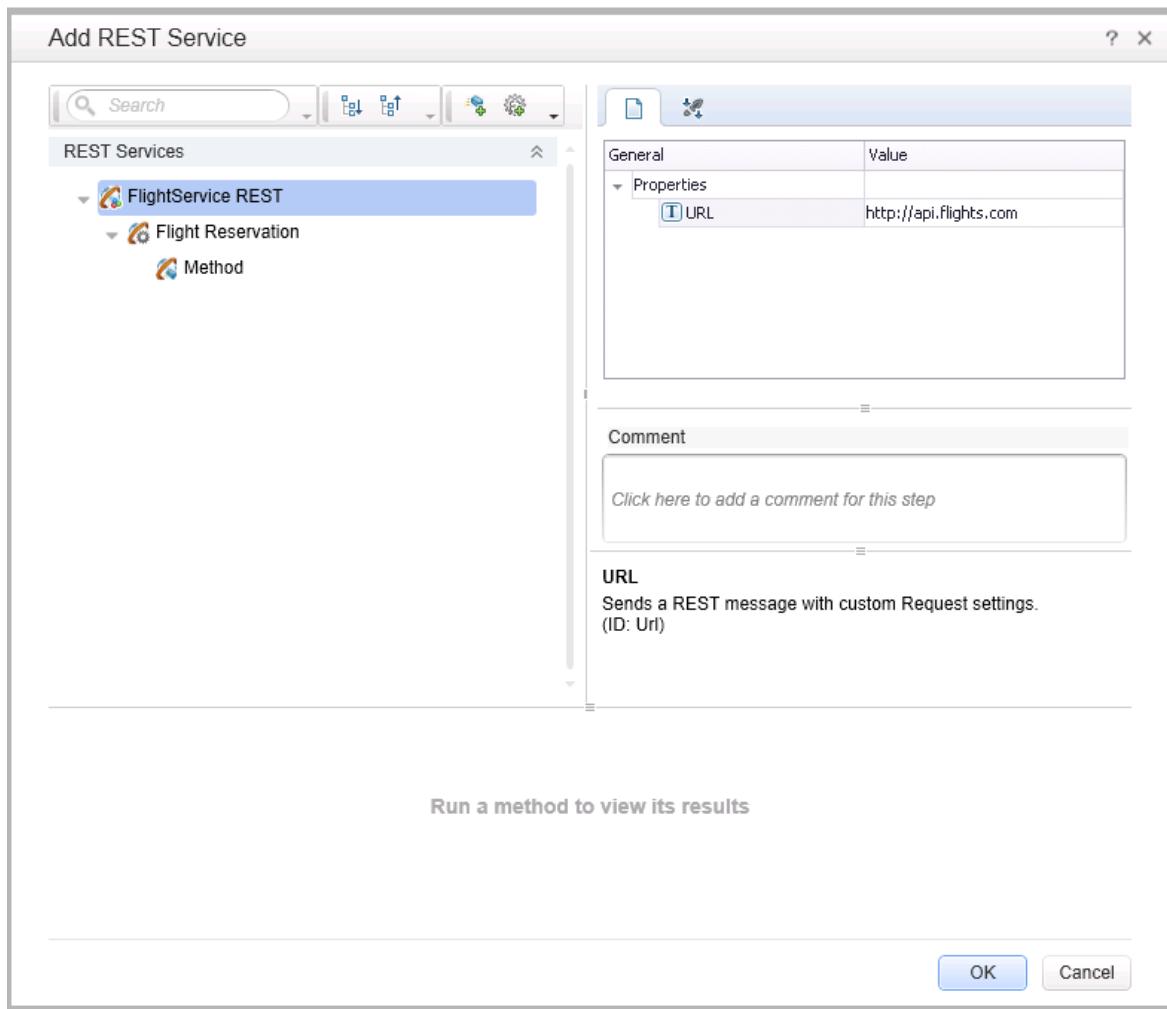
The Properties pane lets you provide values for the SOAP fault header and body. You can enter **faultcode**, **faultstring**, and **faultactor** values as well as custom properties using **Any** type parameters. Using the Checkpoint mechanism, you can validate these values and view the results in the run results.

## Passing REST Service Properties

### **Relevant for: API testing only**

When creating or editing a REST service, you may want to define the value of a URL property or a custom input or output property at the service or resource level in order to make this value available for all resources and methods included in the service or resource. For example, if your REST service resources/methods all reference the same URL prefix, you can define the value of URL property at the service level and pass the URL property value to all resources and methods.

You can also define custom input and output property names and values at all levels of the hierarchy and pass these input and output properties and their values through the REST service hierarchy. For details on creating custom input and output properties, see ["Define custom properties - optional" on page 517.](#)



After a URL property value is defined at a higher level, you can define other (relative) additions to the URL property value for any of the resources and methods included in the service. These adjusted relative URL values are concatenated to the URL property value received from higher levels of the hierarchy and the adjusted values are passed to all levels below it. For example, if you add a URL property value at the service level, you can append relative URL paths for a selected resource or method. The URL property value passed from the service level is then concatenated with the relative URL property value added at the resource or method levels.

**Note:** You cannot modify URL property values passed down from a higher level of the REST service hierarchy. You can only add to them with a relative URL value.

After you add additional relative URL property values at lower levels of the hierarchy, such as at the resource or method level, you must assure that the full URL is a proper URL. For example, if you define

the URL property value at the resource level with `http://`, the relative URL property value appended at the method level should not also use a `http://` prefix.

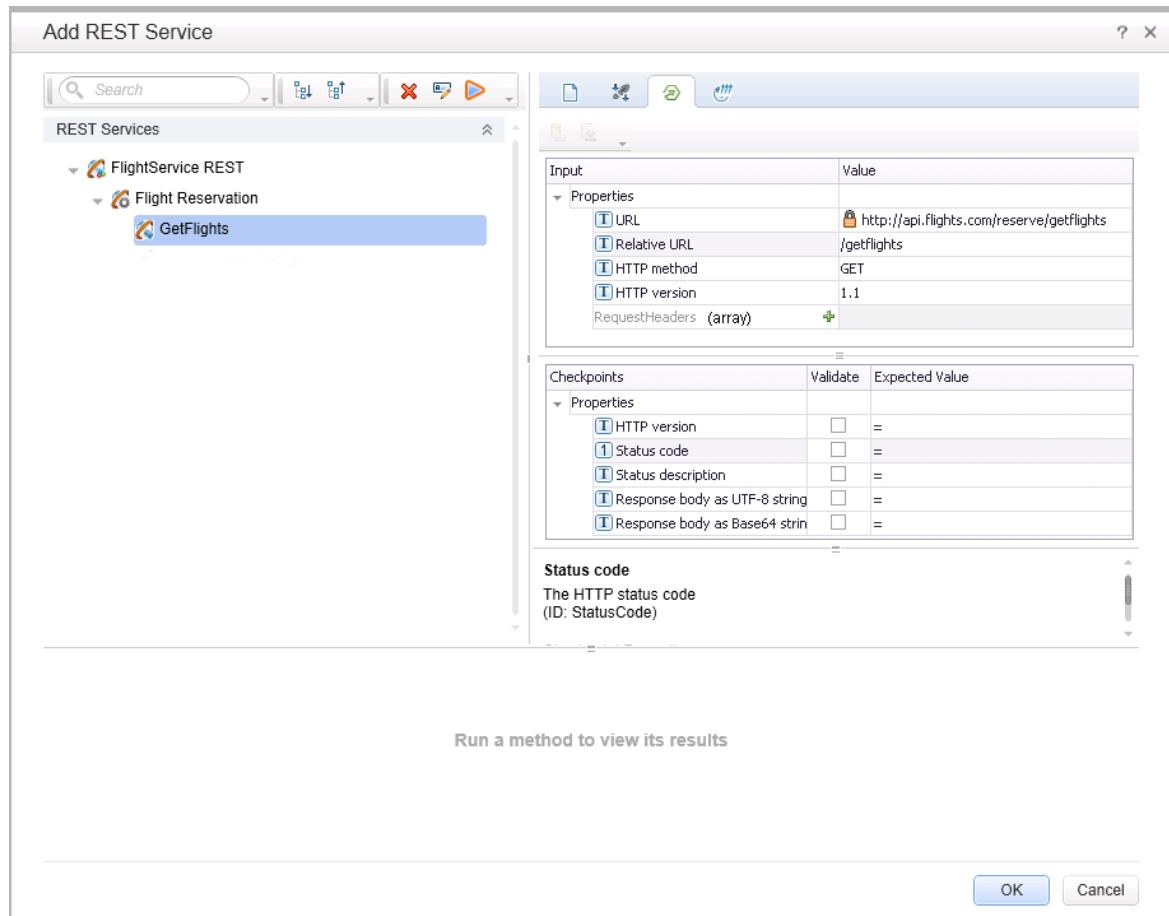
**Example**

You define a URL property value for the REST service at the service (top) level: `http://flights.api.com`. This value is then passed to any resources and methods within this REST service.

You also create a resource for this service, called **Flight Reservation**. You define the relative URL value as `/reserve`. This relative URL property value is then concatenated with the URL property value passed from the service level to create a URL for the resource: `http://flights.api.com/reserve`. This URL property value can also pass to any methods created for the resource.

You then create a method for the Flight Reservation resource, called **GetFlights** and define the relative URL property value for this method as `/getflights`. This value is then further concatenated with the URL passed from the resource to make a complete URL for the method: `http://flights.api.com/reserve/getflights`.

The example below shows the URL property value passed from the service level (<http://api.flights.com>) and the relative URL property value defined at the resource level (/reserve), with the relative URL property value for the method (/getflights) also defined. These URL parts are then concatenated in the **URL** property field to make the complete URL for the method.



These properties and custom input or output properties then serve as a prototype template for the REST service, and you can edit the URL property values or custom input and output property values after adding the method activities to the canvas.

**Note:** The relative URL is not displayed on the REST method when it is included in a test in the canvas. Only the full, concatenated URL for the method displayed in the Add REST Service dialog box is displayed in the Properties pane for the selected method.

## Exposing REST Service Properties

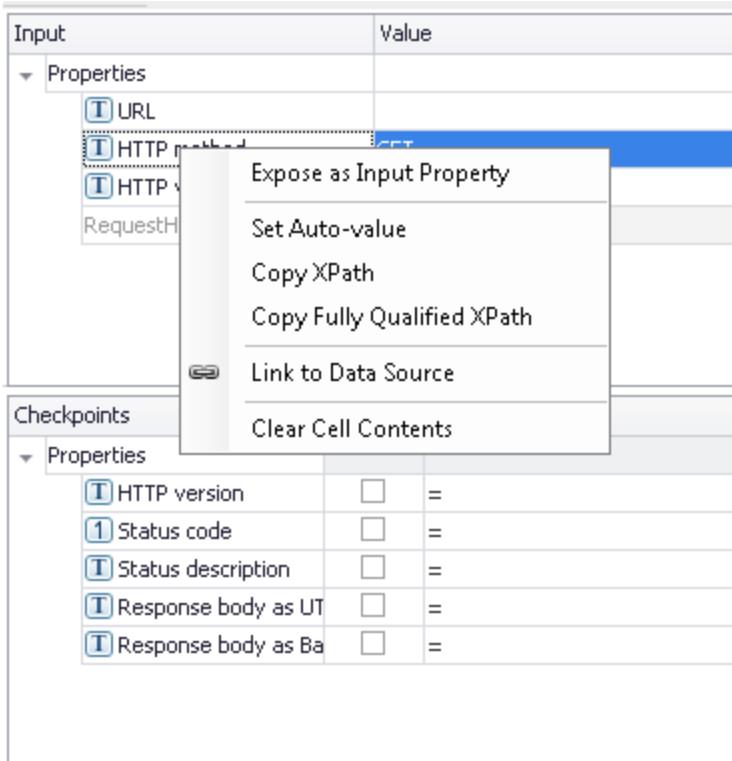
### Relevant for: API testing only

When working with REST service methods saved in API tests created in UFT 11.51 or earlier or Service Test 11.51 or earlier, the Properties pane enables you to expose input and output HTTP properties.

Exposing HTTP properties means that you make them available at the REST method wrapper level instead of just the inner HTTP level. You can expose properties that are available from the **General**, **Input/ Checkpoint**, **HTTP**, and **Multipart** views.

**Note:** You can only expose properties if you are working with API tests created in UFT 11.51 or earlier or Service Test 11.51 or earlier.

The following example shows the shortcut menu item, **Expose as an input property**. This option prompts you to provide a name for the property as it should appear in the REST wrapper level.



The property, **HTTP method**, will be available in the REST method wrapper. To see the exposed property, select the REST method wrapper in the canvas —not the inner HTTP Request.

**Note:**

- When exposing a property with incoming links, the links are redirected to the newly created property.
- When exposing a complex property, the new property will be created as a **String** type.

## How to Import a WSDL-Based Web Service

### Relevant for: API testing only

This task describes how to import a WSDL file into your test.

This task includes the following steps:

- "Open the Import WSDL dialog box" below
- "Select a source" below
- "Set connection settings for URL/UDDI imports - optional" below
- "Mark the WSDL as a server response - optional" on the next page
- "Complete the import" on the next page
- "Validate the WSDL file - optional" on the next page
- "Update your WSDL information - optional" on the next page
- "Configure SOAP Fault information - optional" on page 515

### 1. Open the Import WSDL dialog box

- To import a WSDL from the file system or ALM, in the toolbar, click the **Import WSDL** button  and select **Import WSDL from File or ALM Application Component**.
- To import a WSDL from a Web site or UDDI repository, in the toolbar, click the Import WSDL button and select **Import WSDL from URL or UDDI**.

### 2. Select a source

<b>For File System or ALM Application Components imports</b>	In the Import WSDL dialog box, navigate to the location of the WSDL file and select it.
<b>For URL imports</b>	<ol style="list-style-type: none"><li>In the Import WSDL dialog box, select <b>URL</b>.</li><li>Click the <b>Browse</b> button adjacent to the <b>Address</b> field.</li><li>In the browser window that opens, navigate to the URL containing the WSDL file.</li><li>Close the browser window. The URL is automatically entered in the <b>Address</b> field.</li></ol>
<b>For UDDI imports</b>	<ol style="list-style-type: none"><li>In the Import WSDL dialog box, select <b>UDDI</b>.</li><li>Click the <b>Browse</b> button adjacent to the <b>Address</b> field.</li><li>In the <b>Select Service from UDDI</b> dialog box that opens, specify the UDDI address and click <b>Search</b>. A list of all WSDL files saved in this UDDI location is displayed.</li><li>From the list of WSDL files, select the file(s) to import and click <b>OK</b>. The address and file is automatically added in the <b>Address</b> field.</li></ol>

### 3. Set connection settings for URL/UDDI imports - optional

For URL or UDDI imports, if your WSDL must be accessed through a secure server or proxy machine you may need to set the connection information.

- In the Import WSDL from URL or UDDI dialog box, click **Advanced Settings** to expand the dialog box.

- b. Enter the authentication information or the proxy server details as needed.

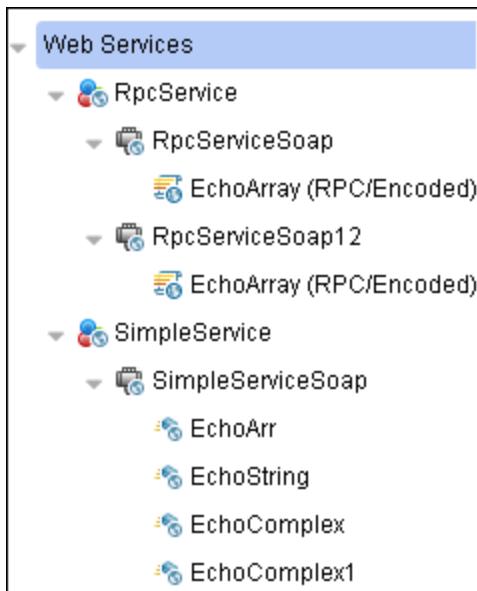
#### 4. **Mark the WSDL as a server response - optional**

You can mark the imported WSDL as a server response method. This is useful when working with asynchronous Web services.

- a. In the Import WSDL from URL or UDDI dialog box, click **Advanced Settings** to expand the dialog box.
- b. At the bottom of the dialog box, select the **Import as server** option.

#### 5. **Complete the import**

Click **OK**. If the import succeeds, the **Toolbox** displays the service, its ports, and operations under the **Web Services** node. The Toolbox pane differentiates between Document/Literal and RPC encoded services with different icons.



#### 6. **Validate the WSDL file - optional**

To check the WSDL's compliance with the WS-I standard, in the Toolbox pane, right-click the service node and select **Validate WS-I Compliance**. The Output window shows the WS-I validation results.

**Note:** These validations apply only to Document/Literal type services, but not RPC.

#### 7. **Update your WSDL information - optional**

- a. If the URL for your imported Web Service changes, you can update the URL after importing without the need to reimport the service and recreate the tests using the imported service methods.

In the Toolbox pane, right-click the top-level service node and do one of the following:

<b>Update WSDL:</b>	This updates the service details from the source provided when you first imported the WSDL file.
<b>Update WSDL from:</b>	<p>This updates the service URL and details from one of the following sources:</p> <ul style="list-style-type: none"> <li>○ <b>URL or UDDI:</b> Enables you to reimport the WSDL file (containing the service model details) from a different URL or UDDI source.</li> <li>○ <b>File or ALM Component:</b> Enables you to reimport the WSDL file from the file system or ALM.</li> </ul>

The service model details are updated in the Toolbox pane and the steps using the service model methods are also updated.

**IMPORTANT:** The service or service location URL must be accessible when updating the service.

## 8. Configure SOAP Fault information - optional

To apply negative testing to a Web service:

- a. In the canvas, select the step to which to apply the SOAP fault.
- b. In the Properties pane, open the **SOAP Fault** tab .
- c. Select **Fault is expected**.
- d. Provide SOAP Fault checkpoint values for the negative testing:

<b>To work in the XML layout:</b>	Expand the SOAP nodes and define <b>Any</b> elements for the SOAP Header and Body. If relevant, provide values for <b>faultcode</b> , <b>faultstring</b> , or <b>faultactor</b> .
<b>To work with XPath expressions:</b>	Click the <b>XPath</b> tab and use the Add button to add new XPath entries. Copy the XPATH entry into the cell.

# How to Create a REST Service

## Relevant for: API testing only

This task describes how to set up a REST service with its resources and methods. You define the request body and properties in order to create a prototype method that could be reused by multiple test steps.

This task includes the following steps:

- "Prerequisite" on the next page
- "Create the service model hierarchy." on the next page
- "Import the service model from a Swagger REST API (optional)" on the next page
- "Set the General properties" on page 517
- "Set the URL property values" on page 517
- "Define the method's HTTP properties" on page 517

- "Define custom properties - optional" on the next page
- "Enter the request body directly - optional" on page 518
- "Link the body request to a data source - optional" on page 518
- "Test the method" on page 519
- "Save the service model to your test" on page 519
- "Use the REST methods in your test" on page 520
- "Expose input and output properties - optional (for API tests created in previous versions)" on page 520

### 1. Prerequisite

Study the structure of the REST Service body and determine which resources and methods you need to define.

### 2. Create the service model hierarchy.

In order for UFT to create and use the service model, you must define the hierarchy of the service, including its resources and methods:

- a. In the toolbar, click the Add REST Service button.
- b. In the Add REST Service editor, give the service a name.
- c. Add resources and methods as needed:

<b>For new resources</b>	Click the <b>Add Resource</b> toolbar button  and provide a meaningful name for the resource.
<b>For new methods</b>	Click the <b>Add Method</b> toolbar button  and provide a meaningful name for the method.

### 3. Import the service model from a Swagger REST API (optional)

You can automatically import the entire REST service structure from a Swagger REST API. UFT will then create the entire Service/Resource/Method hierarchy from the service's details.

- a. In the toolbar click the **Add REST Service** drop-down button  and select **Import Swagger Service from URL** or **Import Swagger Service from File**.
- b. Do one of the following:

<b>When importing from a file</b>	Navigate to the JSON file containing the service description.
<b>When importing from a URL</b>	Enter the full URL to the file, including the file name.

UFT parses the file for a few seconds, then adds the service's resources and methods to the Local Activities node in the Toolbox.

**Note:** If the schema you import from the Swagger API is missing values for some

elements, UFT automatically generates values for these elements, depending on the values set in the **Autovalues** pane of the Options dialog box (**Tools > Options > API Testing tab > Autovalues pane**)

#### 4. Set the General properties

- a. Select the service, resource or method for which you want to set its properties.
- b. In the right pane's **Properties** list, open the **General**  tab.
- c. Enter values for the properties.

#### 5. Set the URL property values

- a. In the REST Service editor, select a REST service, resource, or method.
- b. In the General tab, enter the URL property value:
  - If you are entering a URL property value for a REST service, enter the URL prefix in the **URL** property row, beginning with a `http://`
  - If you are entering a URL property value for a REST resource or method, enter the URL property value in the Relative URL property row.

**Note:** If you enter a URL property value for the service or resource of your REST service, the URL property values are passed to all resources or methods included in the service or resource. For details, see "[Passing REST Service Properties](#)" on page 508

#### 6. Define the method's HTTP properties

- a. Select a REST service method.
- b. In the REST service editor's right pane, click the **HTTP Input/Checkpoints** tab.
- c. For each method, modify the **HTTP method** and **HTTP version**.
- d. Use the plus sign  in the **RequestHeaders** parent node to add name and value pairs for the request header array.

#### 7. Define custom properties - optional

For methods that require input, such as PUT or POST, you can add custom input properties. For methods that provide an output such as GET, you add the required output properties. To create these properties:

- a. In the right pane's **Properties** list, click the **Custom Input/Checkpoints** tab .
- b. Expand the plus button  and select **Add Input/Output Property**.
- c. Provide a name, data type, and description (optional) for each property.
- d. Enter the property values in the **Value** column.

## 8. Enter the request body directly - optional

**Note:** This step describes how to enter the request body directly. (For details on linking to a data source, see the next step.)

- a. Return to the design document for the REST service and copy the Request body onto the clipboard.
- b. In the right pane of the Add REST Service window, open the **HTTP** tab. Make sure the **Body** type in the drop down is set to **Text**, and click within the **Body** section. Press **CTRL+V** to paste the contents of the request into the pane. Make sure that there are no extra spaces before or after the body text.
- c. Modify the element values within the XML body as required.

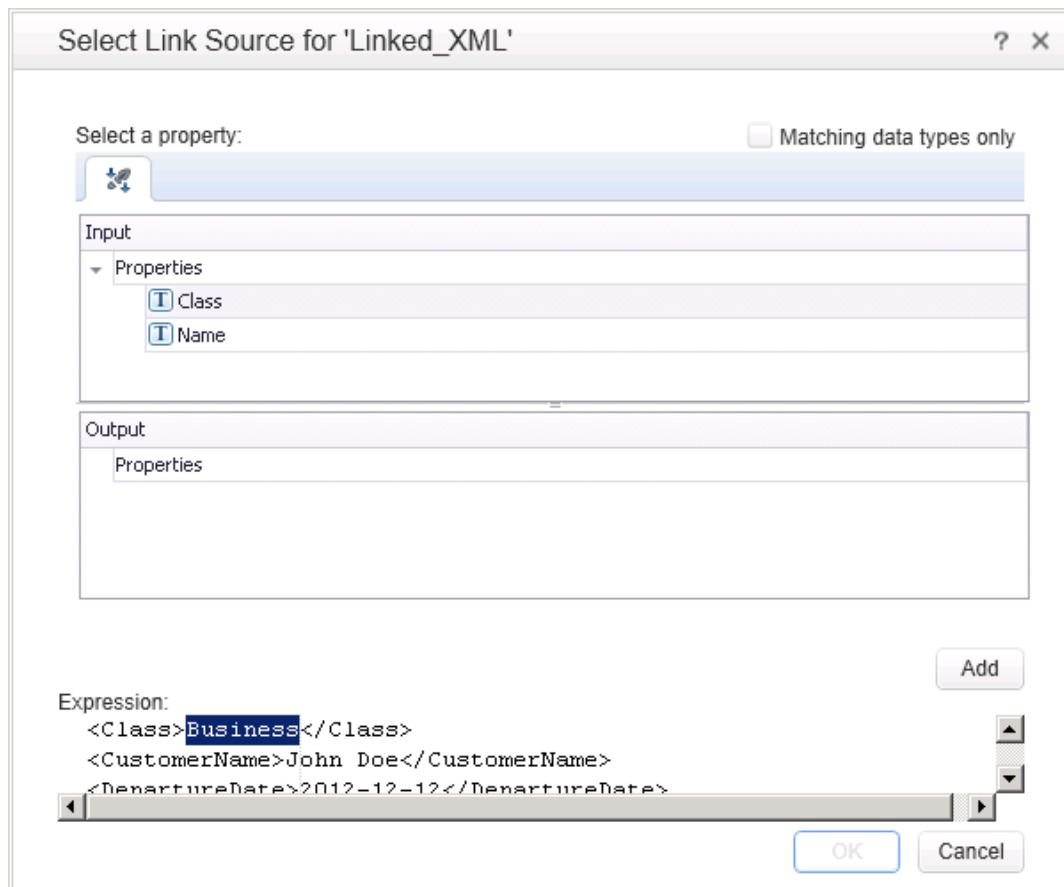
**Note:** If you need to use a multipart request with your REST Service methods, this is done after dragging the REST method to the canvas. For details, see "How to Send a Multipart HTTP or REST Service Request" on page 418.

## 9. Link the body request to a data source - optional

**Note:** This step describes how to enter the Request body through a data source. (For details on entering the Request body directly, see the previous step.)

- a. Return to the design document for the REST service and copy the request body to the clipboard.
- b. In the right pane of the Add REST Service dialog box, open the **HTTP** tab . Make sure the **Body** type in the drop down is set to **Text**, and click within the **Body** section.
- c. Click the **Link Body** button  to the right of the pane, to open the Select Link Source dialog box.
- d. In the Select Link Source dialog box, click **Custom Expression** to expand the dialog box. Paste the clipboard contents, the Request body, into the **Expression** area.

- e. In the **Expression** area, highlight the value of the element you want to link. In the upper pane, double-click on the custom property you defined earlier in the properties list. The property is now linked to a value.



- f. The modified expression appears in the **Expression** area. In the following example the custom Class property is linked to Class element in the REST document.

```
<Class>{Step.InputProperties.RestMethod.Class}</Class>
```

- g. Click **OK**. UFT places the data in the **Body** area.

If you need to use a multipart request with your REST Service methods, this is done after dragging the REST method to the canvas. For details, see "[How to Send a Multipart HTTP or REST Service Request](#)" on page 418.

## 10. Test the method

Click the **Run Method** button on the toolbar to test the method with its property values. The results are displayed in the lower section of the window.

## 11. Save the service model to your test

In the REST Service editor, click **OK** in the Design REST Service dialog box. UFT adds the REST service along with its resources and methods to the Toolbox pane, under the **Local Activities** category.

## 12. Use the REST methods in your test

From the Toolbox pane Drag the prototype methods to the canvas and edit the step's properties if needed.

**Note:** The default properties entered after you drag the method to the canvas are the properties you created when designing the Service, Resources, and Methods for the REST Service.

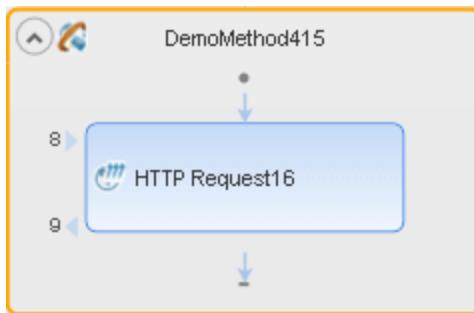
If you modify a step property after dragging the step to the canvas, UFT reports a conflict. You can use the Resolve REST Method Steps Wizard to resolve the conflicts.

## 13. Expose input and output properties - optional (for API tests created in previous versions)

You can forward built-in HTTP properties to the REST method wrapper. This is useful for making specific HTTP properties available at the wrapper level.

**Note:** If you created a REST method in UFT 11.52 or Service Test 11.52 or later, all REST Service method properties are incorporated into the REST activity step itself without a wrapper.

- Double-click a REST method from the Toolbox pane to add it to the canvas. Expand the method to show the **HTTP Request** frame.



- Click in the **HTTP Request** frame. Select a property in the Properties pane and select **Expose as Input Property** or **Expose as Output Property** from the right-click menu.
- Provide a name for the property in the New Exposed Property dialog box.
- Click the REST method wrapper in the canvas, and view the newly exposed property in the Properties pane's **Custom Input/Checkpoints** tab

For details, see "["Exposing REST Service Properties"](#) on page 511.

# How to Send and Receive a JSON Request for a REST Service

## Relevant for: API testing only

This task describes how to send a JSON request for a REST service method that has been added to your test.

This task includes the following steps:

- "Set the HTTP properties" below
- "Prepare to load the request body" below
- "Load the request body" below
- "Modify the JSON body - optional" on the next page
- "Data drive or link to fields - optional" on the next page

**Note:** The following tasks show how to send a single JSON request to a REST service method step (after it has been added to the canvas). To create a reusable model, create a prototype. For details, see "[How to Create a REST Service](#)" on page 515.

## 1. Set the HTTP properties

**Note:** If you are working with an API test created in UFT 11.51 or earlier or Service Test 11.51 or earlier, you must expand the REST activity's wrapper and enter these properties in the HTTP Request step found inside the REST activity wrapper.

In the Properties pane's Input/Checkpoints tab , set the destination **URL** and the **HTTP method**, usually POST or PUT.

## 2. Prepare to load the request body

- a. In the Properties pane, open the **HTTP** tab .
- b. Select **Body type** **JSON**.

## 3. Load the request body

Click the **Load JSON** button and navigate to the .json file.

**Note:** If your JSON file contains non-ASCII characters, you should save this file with UFT-8 encoding. Otherwise, the characters in your file may not display correctly in UFT.

## 4. Add a request header - optional

**Note:** If you are working with an API test created in UFT 11.51 or earlier or Service Test 11.51 or earlier, you must expand the REST activity's wrapper and enter these properties in the HTTP Request step found inside the REST activity wrapper.

If your server requires you to specify JSON during content negotiation, you need to set the request header.

- a. In the Properties pane, open the Input/Checkpoints tab .
- b. In the Input/Checkpoints tab, click the plus sign to add a **RequestHeader** array element.
- c. Add a custom request header named Accept with the value application/json.

## 5. Modify the JSON body - optional

If you intend to dynamically assign values to the JSON body from a data source, you need to add escape characters. Open the **Text** view of the JSON body and add the escape character, \, for each occurrence of a square or curly bracket ({, }, [ , and ] ). Do not use an escape character for link expressions enclosed by curly brackets. For example:

```
\{"results":
[\n {"name": "John", "id": 873829904, location: "NY"},
 {"name": "Linda", "id": 726371109, location: "LA"},
 {"name": "Mike", "id": 711029345, location: "NY"},
]
```

When no links are used, this is not required.

## 6. Data drive or link to fields - optional

To data drive or link to specific JSON fields, highlight the value in the body text, click the **Link Body** button , and select a data source.

# How to Import a Web Application Service

## Relevant for: API testing only

This task describes how to import a Web Application service (WADL) into your test.

This task includes the following steps

- ["Prerequisite" on the next page](#)
- ["Import the WADL document" on the next page](#)

## 1. Prerequisite

Before importing, study the structure of your WADL document, as specific elements from the document are imported into the WADL hierarchy inside UFT.

```
<?xml version="1.0" encoding="utf-8" ?>
<application xmlns="http://wadl.dev.java.net/2009/02">
 <resources base="http://example.com/api">
 <resource path="books">
 <method name="GET"/>
 <resource path="{bookId}">
 <param required="true" style="template" name="bookId"/>
 <method name="GET"/>
 <method name="DELETE"/>
 <resource path="reviews">
 <method name="GET">
 <request>
 <param name="page" required="false" default="1" style="query"/>
 <param name="size" required="false" default="20" style="query"/>
 </request>
 </method>
 <resource path="{index}">
 <method name="GET" id="get index"/>
 <param name="index" style="template"/>
 </resource>
 </resource>
 </resource>
 <resource path="readers">
 <method name="GET"/>
 </resource>
 </resources>
</application>
```

For details on WADL elements, see <http://www.w3.org/Submission/wadl/>.

## 2. Import the WADL document

- In the toolbar, click on the **Add REST Service** button and select **Import WADL from File** or **Import WADL from URL** or select **Tools > Add REST Service > Import WADL from File** or **Import WADL from URL**.
- Do one of the following:
  - In the Choose WADL file dialog box, navigate to the directory in which the WADL is saved, and select the WADL file.
  - In the Import WADL from URL dialog box, enter the URL for the WADL or click **Browse** and search for the URL.

The WADL service is imported to the **Local Activities** node of your test with its resources and methods. The WADL service, resource, and method hierarchy is created based on the XML description provided in the WADL file, described below:

XML Element	UFT WADL Activity Hierarchy Element
doc xml:lang="en" title="RestService"	<b>WADL service name</b>
resource path=<resource name>"	<b>WADL resource</b> <b>Note:</b> If multiple resources have the same name, UFT numbers the resources sequentially to differentiate between resources.
method name=<method name>"	<b>WADL method</b> <ul style="list-style-type: none"> <li>• UFT assigns the WADL hierarchy name using the following criteria: If a method name has an "id" attribute, the name is taken from the value of the "id" attribute. If a method name does not have an "id" attribute, then the name is defined as the value of the "method_name". For example, if the "method_name" is defined as &lt;"method_name="GET"/&gt;, UFT defines the method name in the WADL hierarchy as GET Method. The method name always contains the HTTP method as part of its value. This method (GET, POST, PUT, DELETE, TRACE, OPTIONS, or HEAD) is also used as the HTTP method in the HTTP tab of the WADL service.</li> </ul> <b>Note:</b> If there are multiple methods of the same name using the default values, then the methods are defined with increasing sequential numbers.
param name=<parameter name>"	<b>WADL resource or method parameters.</b> These parameters are displayed until the Custom Input/Checkpoints tab in the Edit REST Service dialog and in Input/Checkpoints tab for methods on the canvas. If a "param_name = <name>" string also contains a "default=<value>" string, the value defined in the XML is displayed with the parameter.
resources base="http://example.com/api"	<b>WADL Service URL.</b> This is displayed on the HTTP tab for the service. The URL is also passed to all resources and methods included in the service. For details, see " <a href="#">Passing REST Service Properties</a> " on page 508. <b>Note:</b> You cannot change the URL property for an individual resource or method.

# How to Import a Network Capture File

## Relevant for: API testing

This task describes how to import a network capture file to your test to create test steps.

This task includes the following steps:

- "Create a capture file" below
- "Prerequisite - study the structure of your network capture file" below
- "Import the network capture file" on the next page

### 1. Create a capture file

Use a network capture program (also known as a sniffer) to create a capture file containing a log of network activity for your application or Web service.

**Note:** It is strongly recommended that you capture network traffic only on your application or Web service during the network capture session to prevent the creation of invalid or unneeded activities in your test. Many network capture tools capture all network traffic on the computer where they are installed, including network traffic unrelated to your application or Web service.

### 2. Prerequisite - study the structure of your network capture file

The structure of your file depends on the type of file you import. UFT supports .libpcap/pcap and .har network capture files.

- **For .pcap files:** UFT creates test steps based on the TCP network traffic. You can see the input and checkpoint values by viewing the TCP request and response information for a TCP stream.

**Note:** You must use a network capture program to view the network capture traffic for your .pcap file.

- **For .har files:** Using a text editor, you can view the Request and Response information in the JSON hierarchy of the file.

**Note:** UFT creates a test activity based on its recognition of the TCP stream in the following ways:

- If UFT recognizes the TCP stream as being compatible with or matching an already existing Web Service method, UFT creates a **Web Service** step.
- If UFT recognizes the TCP stream response as a SOAP network transaction, it creates a **SOAP Request** step.
- If UFT does not recognize the TCP stream as being similar to an existing Web Service step or a SOAP request network transaction, it creates an **HTTP Request** step.

### 3. Import the network capture file

- a. In UFT, with an API test open or selected, select **Tools > Import Network Capture File**. The Import Network Capture Dialog Box opens.
- b. In the Import Network Capture Dialog Box, select the file containing your network capture data.

**Note:** UFT supports only .pcap and .har network capture files.

- c. View the file details in the **Info Pane** of the Import Network Capture Dialog Box. If there are errors in your file, return to your network capture tool and fix the errors.
- d. If you want UFT to create checkpoints for your test steps based on the response sections of the network capture file, select the **Create checkpoints from response** option.
- e. Click **Import**. If your network capture file contains many transactions, UFT displays the progress of your import.

**Note:** To stop an import, click **Cancel** in the import progress window. All test steps created prior to your cancellation are removed from the test.

UFT creates Web Service test steps, SOAP Request test steps, or HTTP Request test steps for each of the transactions contained in your network capture file.

## How to Import and Create a .NET Assembly API Test Step

### Relevant for: API testing only

This task describes how to create a test step for verifying the functionality of a .NET assembly.

This task includes the following steps:

- "Prerequisite " on the next page
- "Import a .NET assembly" on the next page

- "Select a GAC assembly - optional" below
- "Add an ExecuteEvent event handler" below
- "Add custom input and output properties - optional" on the next page

## 1. Prerequisite

Prepare a .dll assembly file and store it in a location accessible from your machine.

## 2. Import a .NET assembly

- a. In the toolbar, click the **Import .NET Assembly** button .
- b. In the Import .NET Assembly dialog box, select the **.NET Assembly Browser** tab.
- c. Navigate to the direction containing the assembly .dll or .exe file and select the file.
- d. Click **Select** to add it to the **Selected References** list.
- e. Click **OK** to add the .NET assemblies to the Toolbox pane.

**Tip:** After importing the .NET assembly, you should save the test. Not saving the test leads to unexpected behavior with the autocomplete functionality when adding an event handler to a .NET assembly test step.

## 3. Select a GAC assembly - optional

Each computer where the common language runtime is installed has a machine-wide code cache called the GAC (Global Assembly Cache). The GAC stores assemblies specifically designated to be shared by several applications on the computer.

- a. In the Import .NET Assembly dialog box, select the **GAC** tab.
- b. In the list of references, select one or more GAC assemblies and click **Select** to add them to the **Selected References** list.

## 4. Add an ExecuteEvent event handler

- a. In the Properties pane, open the **Events** tab .
- b. In the **ExecuteEvent** row, in the Handler column, click the down arrow and select Create a default handler.
- c. In the **TestUserCode.cs** file that opens, locate the **CodeActivity<#>\_ExecuteEvent** section of the code.
- d. Under the **CodeActivity>#>\_ExecuteEvent**, edit the TODO area. You can access input and output properties that you defined earlier, using autocompletion.

```
/// Use this.CodeActivity4 to access the CodeActivity4
/// Activity's context, including input and output properties.
public void CodeActivity4_OnExecuteEvent(object sender,
```

```
STActivityEventArgs args)
{
 this.CodeActivity4.Input.Property1...
 ...
}
```

## 5. Add custom input and output properties - optional

By default, the .NET Assembly step contains no input or output properties. You must add the necessary properties:

- a. After adding a step, in the Properties pane, open the Input/Output Properties tab .
- b. In the Input/Output Properties tab, click the **Add Input/Output Property** button and add input and output properties.

# Troubleshooting and Limitations - Custom Activities

## Relevant for: API testing only

Note the following troubleshooting and limitations for working with custom activities:

<b>Web services</b>	<ul style="list-style-type: none"><li>You cannot import WSDL files with names that are restricted by the Windows operating system. This list includes: CON, PRN, AUX, NUL, COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, and LPT9.</li></ul> <p><b>Workaround:</b> Rename the WSDL file before the import.</p> <ul style="list-style-type: none"><li>The following WSDLs are not supported:<ul style="list-style-type: none"><li>WSDL version 2.0</li><li>WSDLs containing a &lt;appInfo&gt; element.</li><li>RPC Encoded WSDLs configured as Asynchronous Web services.</li><li>WSDLs authenticated by HTTP digest on Apache servers.</li></ul></li><li>When opening many tests in the same solution using one or more of the same Web services, UFT may develop a memory leak.</li></ul> <p><b>Workaround:</b> Move the Web service to the file system or ALM repository. To do this, import a Web service and then, in the <b>Toolbox</b> pane, right-click the Web Service activity and select <b>Move to &gt; File System Activities or ALM Activities</b>.</p> <ul style="list-style-type: none"><li>For a Web service imported as a server response:<ul style="list-style-type: none"><li>RPC type WSDLs cannot be imported as a server response. If you attempt to update a service from an RPC encoded WSDL, it will create a duplicate entry.</li><li>If you end the UFT .exe process during the listening stage, after the binding was added, the binding will not be removed from the system.</li></ul></li></ul> <p><b>Workaround:</b> Remove the binding manually using a utility such as <code>httpcfg.exe</code> or <code>netsh.exe</code>.</p> <ul style="list-style-type: none"><li>When working with SSL, certificates from a file are not supported. If you move the test to another machine, the certificate will not be saved with the test.</li></ul> <p><b>Workaround:</b> Add the certificate to the local machine store before the listener starts, and remove it at the end of the listening process.</p>
<b>REST services</b>	<ul style="list-style-type: none"><li>Importing a schema to a REST, HTTP, or SOAP checkpoint may remove the links of the input properties.</li><li>XPath checkpoints are not supported for HTTP and REST activities.</li><li>If you link between input and output properties in the same REST method and then delete the input property without removing the link, the link expression still remains in the output property. If you save the test in this state, you may be unable to reopen it.</li></ul> <p><b>Workaround:</b> Delete the link explicitly either before or immediately after deleting its source property.</p> <ul style="list-style-type: none"><li>When defining a REST method prototype in the Add/Edit REST Service dialog box - in order to use the Trim, Ignore, or Stop test options, select the check box in the <b>Validate</b> column, in the Checkpoints pane.</li><li>When running a REST method using the <b>Run Step</b> command, checkpoints of dynamic property values in the</li></ul>

	method linked to other property values (input or output) are ignored.
<b>Web Application Services</b>	<p>The following elements are not supported when importing your WADL. UFT does not import these elements into the WADL hierarchy inside your test:</p> <ul style="list-style-type: none"><li>• Grammars element</li><li>• Resource_type element</li><li>• Link element</li></ul> <p><b>Note:</b> Any child elements of these elements are also ignored by UFT and not added to your WADL inside the test.</p> <p>In addition, if you use the href attribute to link to other elements, you must refer to an element in the same WADL file. Linking between WADL files is not supported.</p>
<b>Network Capture Activities</b>	Steps added to your test from a network capture file do not include: <ul style="list-style-type: none"><li>• Security settings for Web service calls and SOAP requests</li><li>• Web authentication information for any type of step</li><li>• Cookies data for HTTP request</li><li>• Attachments sent as part of the Web Service call</li></ul>
<b>.NET Assembly Activities</b>	Importing or adding references to 64-bit .NET assemblies is not supported.

# Chapter 43: Actions for API Tests

**Relevant for: API testing only**

This chapter includes:

- [Actions in API Testing - Overview](#) ..... 532
- [How to Use Actions in an API Test](#) ..... 533
- [Actions Using the Data Table](#) ..... 534

# Actions in API Testing - Overview

## Relevant for: API testing only

Actions are sequences of operations that you perform within the context of a test, consisting of either one or multiple steps. Actions also allow you use a repeated activity, such as logging in to a Web site multiple times. Each time the action is called, the test runs the action steps, its properties, and its data. Instead of adding the necessary test steps again each time you need to perform an action in the application, you can call the action and UFT runs its steps instead.

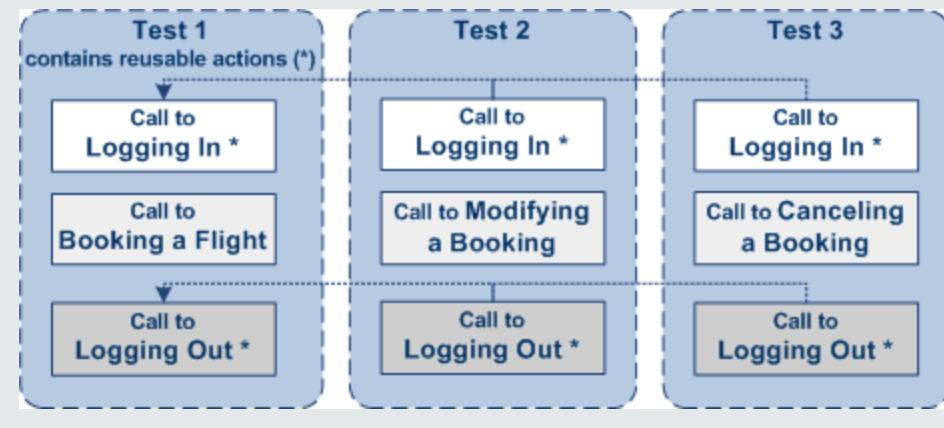
You can call both actions defined within your current test or defined in other tests. When you call an action from another test, by default its data is read-only. If you call an action from an external test that has data assigned to its properties, you can indicate whether you want the data from action's data sheet to be imported as a local, editable copy, or whether you want to use the (read-only) data from the original action.

**Note:** To modify the steps of an action from an external test, you must open the test in which the action is stored and make your modifications there. The modifications apply to all tests that call that action.

## Example

Suppose you want to create the following three tests for flight reservation--booking a flight, modifying a reservation, and deleting a reservation. While planning your tests, you realize that for each test, you need to log in and log out of the site.

Test 1 would contain three actions (Logging In, Booking a Flight, and Logging Out). Test 2 would contain three actions (Logging In, Modifying a Booking, and Logging Out). Test 1 would contain three actions (Logging In, Cancelling a Booking, and Logging Out). The Logging In and Logging Out actions are defined in the same test, and called as external actions by all three tests.



Actions can also be saved as business components to be used for BPT.

# How to Use Actions in an API Test

## Relevant for: API testing only

This task describes the different operations you can perform to use actions in your test.

This task includes the following steps:

- "[Insert a call to a new action](#)" below
- "[Call an existing action or a test](#)" below
- "[Set action properties](#)" below
- "[How to Use Actions in an API Test](#)" above
- "[Enable an action's data for editing when called by another test](#)" on the next page
- "[Override data from an action called by another test](#)" on the next page

## Insert a call to a new action

1. Select **Design > Call to New Action**.
2. Specify the action name and description. Click **OK** to add the call to the action to the canvas.
3. In the Solution Explorer, double-click the action to open its canvas.
4. Add steps to the action. Drag activities from the Toolbox onto the canvas.

**Note:** You can insert a call to another action, from an existing action call.

## Call an existing action or a test

1. Drag an **HP Automated Testing Tools > Call API Action or Test** or **Call GUI Action or Test** activity onto the canvas.
2. In the Properties Pane, Open the **Input/Checkpoints** tab .
3. In the Input/Checkpoints tab, click the **Select Action or Test** button.
4. Select the desired test and action. Click **OK**.

## Set action properties

To set an action call's properties, you must set the values for each of the steps contained in the action separately.

1. Select an activity within the action whose properties you want to set.
2. In the Properties tab, provide values or data drive the properties.
3. The property values that you set will be the default values used when you add a call.

**Note:** If you reload the service using the **Refresh** button, the new version may add or remove properties. Properties that remain unchanged, will retain the default values.

## Enable an action's data for editing when called by another test

To view or override an external action's data, you must expose it in its original location. This only applies to Excel type data sources, and only available for tests—not business components.

1. Define data for the action that you will be calling. Assign the data manually or use data driving. For details, see "[How to Assign Data to API Test/Component Steps](#)" on page 551.
2. In the Data Pane, select the data source's node.
3. In the Properties pane, open the Data Source Properties tab  and select the **Allow other tools to override the data** option. The Data pane creates a local copy of the data which can be edited.

## Override data from an action called by another test

1. Make sure the action that you want to call has its data exposed. For details, see the above step.
2. Open the test in which you want to add the call to the action.
3. From the Toolbox pane, expand the **HP Automated Testing Tools** node and add a **Call API Action or Test** or a **Call GUI Action or Test** activity onto the canvas.
4. In the **Input/Checkpoints** tab  in the Properties pane, click the **Select Action or Test** button.
5. In the Select Action or Test dialog box, navigate to the test and select the desired action. Click **OK**.
6. In the Data pane, select the data source node.
7. In the Properties pane, open the **Data Source Properties** tab  and select the **Use a local, editable copy** option.
8. Edit the data tables in the Data pane.

# Actions Using the Data Table

## Relevant for: API testing only

You can use the Data pane tables to provide property values for your actions. Each action uses its own data sources—not the test's data source.

**Tip:** If you want to use the same data for the test and in a specific action, you must define the data source path twice, once for the test and once for the action. See "[Using Data in Your API Test or Component Steps](#)" on page 537.

By default, data from actions called from other tests, are not visible in your current test. You can expose the data by enabling the **Allow other tools to override the data** option in the action's test. As a result, you can view the data in the Data Pane. It is marked with an icon  indicating that this is data from an action.

An **Update** option allows you to reload the action's data in your test when it changes in its original location. This only applies to data that you did not make editable. For details, see "[How to Use Actions in an API Test](#)" on page 533.

**Note:** You can switch data from read-only to editable or vice versa. However, when changing from editable to read-only, any changes made to the data are lost.

If an action is called repeatedly in the test, only one set of the action's data is displayed in the Data Pane. If you make changes to the data at the action's original location and it is not marked as editable, then the changes are applied to all calls to this action.

# Chapter 44: Data Usage in API Tests

**Relevant for: API testing only**

This chapter includes:

• Using Data in Your API Test or Component Steps .....	537
• Populating Property Values Manually .....	537
• Linking Property Values to a Data Source .....	539
• Linking Property Values to Other Test Steps .....	541
• Linking Property Values to Multiple Sources of Data .....	544
• Data Assignment in Arrays .....	545
• How to Add Data Sources to an API Test .....	548
• How to Assign Data to API Test/Component Steps .....	551
• How to Assign Data to API Test Steps - Tutorial .....	555
• How to Define API Test Properties or User/System Variables .....	558
• How to Parameterize XML Data .....	560
• How to Data Drive Array Checkpoints .....	562
• Navigating Within a Data Source .....	564
• Parent/Child Data Source Relations .....	565
• How to Set the Data Source Navigation Properties .....	566
• Data Keywords .....	567
• Troubleshooting and Limitations - Using Data in API Tests and Components .....	569

# Using Data in Your API Test or Component Steps

## **Relevant for: API testing only**

When you create test steps, you must assign values for input and checkpoint properties contained in your test steps.

In UFT, you can link property values to data in a number of ways:

<b>Link to a data source.</b>	UFT enables you to use an Excel file, an XML data source, a database, or a locally-created data table as possible data sources. After the data source is added to the test, you associate property values to the associated data source. For more details on linking your test steps to a data source, see " <a href="#">Linking Property Values to a Data Source</a> " on page 539. For details on associating a data source with your test using the Data pane, see " <a href="#">How to Add Data Sources to an API Test</a> " on page 548.
<b>Link to another test step:</b>	In some cases, your application passes a value between multiple processes. Therefore, in your test, you can link the output of one step to the input of another step. For details, see " <a href="#">Linking Property Values to a Data Source</a> " on page 539.
<b>Link to multiple sources.</b>	You can also link your property values to multiple sources by creating a custom expression that combines manual property value input, values from a data source, and input from multiple test steps. As a result, if an application process receives its input from both a data source and the previous step, your test can reflect the same property/parameter input. For more details, see " <a href="#">Linking Property Values to Multiple Sources of Data</a> " on page 544.
<b>Link to a test variable value.</b>	In addition to linking to a data source value or another test step, you can provide the property values by linking to a test variable value or a user-defined variable. For task details on using data with test steps, see " <a href="#">How to Assign Data to API Test/Component Steps</a> " on page 551.

In addition, if you would like to use data for all the properties in your test, you can data-drive all the properties of a selected step. Data-driving is the mechanism by which UFT automatically creates data expressions that refer to data in a new, editable table.

## Populating Property Values Manually

## **Relevant for: API testing only**

When you enter property values for your test step, the basic way to do this is to provide manual values for each property.

In the Parameters/Checkpoints Tab, you can view and edit the values of each input and checkpoint property, both for simple properties and arrays.

The screenshot shows the 'Input' tab of the property grid. The structure is as follows:

- Envelope** (selected)
  - Header**
  - Body**
    - GetFlights**
      - DepartureCity**: Value is 'Denver'
      - ArrivalCity**: Value is 'Los Angeles'

For test steps that use XML and JSON (such as Web Service calls and XML/JSON activities), you can also manually edit the values in the XML/JSON text mode.

```

<?xml version="1.0" encoding="utf-8"?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
 <Body>
 <GetFlightOrders xmlns="HP.SOAP.SampleApp">
 <FlightsDetails>
 <CustomerName>John Smith</CustomerName>
 <FlightDate>2011-12-22T00:00:00+02:00</FlightDate>
 <OrderNumber>123</OrderNumber>
 </FlightsDetails>
 </GetFlightOrders>
 </Body>
</Envelope>

```

Any changes you make are then reflected in the Input/Checkpoints property grid:

The screenshot shows the 'Input' tab of the property grid after manual edits:

- Envelope** (selected)
  - Header**
  - Body**
    - GetFlightOrders**
      - FlightsDetails**
        - CustomerName**: Value is 'John Smith'
        - FlightDate**: Value is '2011-12-22T00:00:00.000'
        - OrderNumber**: Value is '123'

In addition, you can manually populate the checkpoint values by loading values that were obtained from an earlier run using the **Load from Replay** mechanism.

## Linking Property Values to a Data Source

### **Relevant for: API testing only**

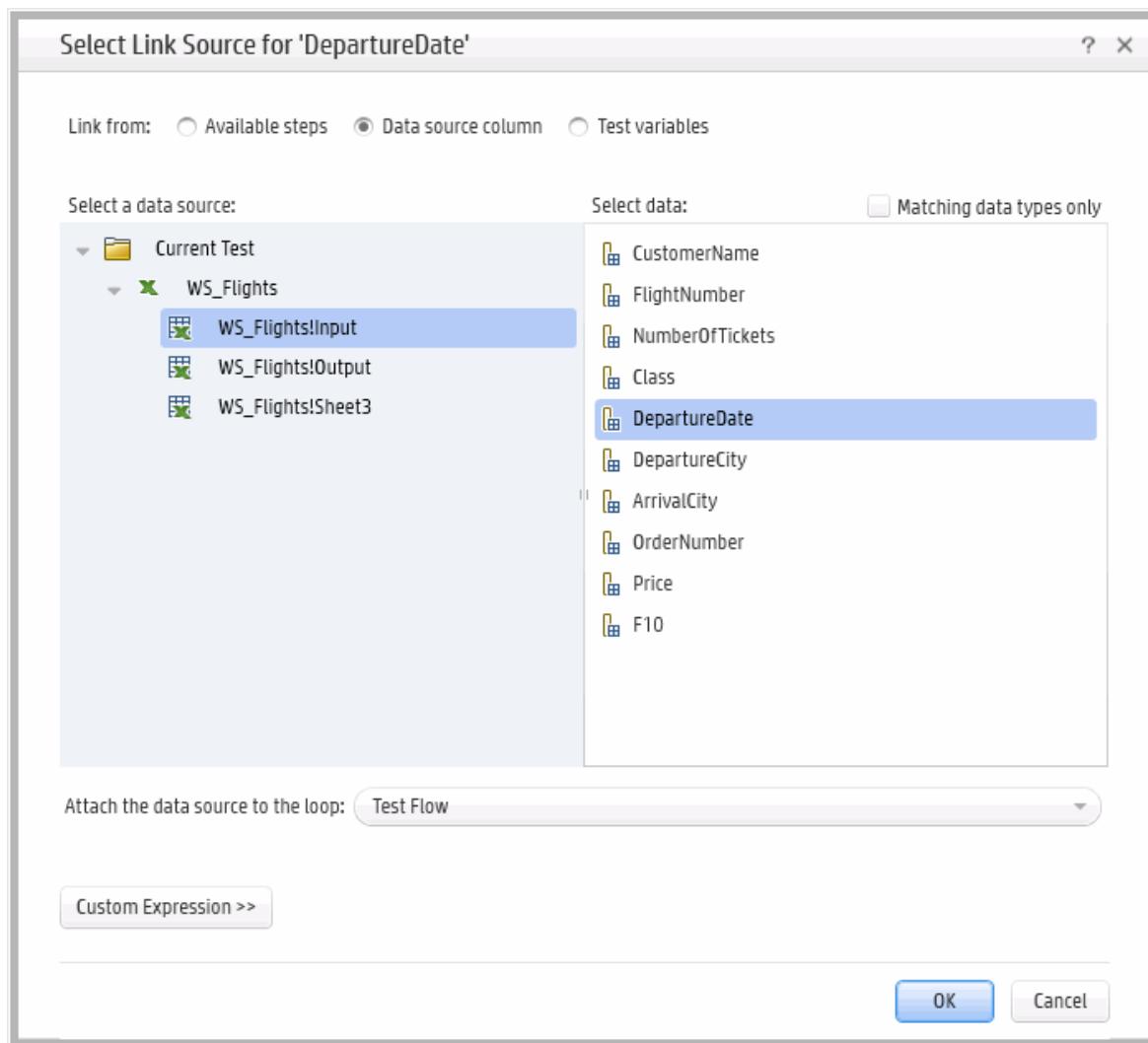
You can link the test input and checkpoint property values to a data source. In this way, you can test how a particular test runs with varying input and checkpoint properties.

**Note:** Before you can link your property values to a data source, you must import the data source into your test in the Data pane.

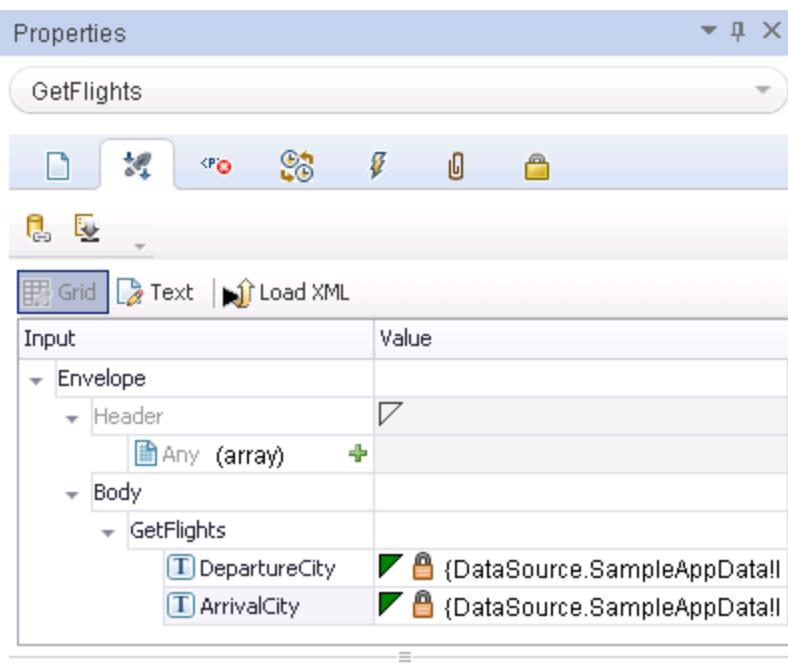
You can link your test steps to multiple types of data:

- Excel files
- XML files
- Database values
- Local table values

The image below shows an example of connecting a test step to an Excel data source using the Select Link Dialog Box:



Once you have linked the step to a specific value in the data source, the Value column for the property name in the Properties pane reflects this link:



When the test runs, the property values are provided from the associated source.

For task details on how to link property values to a data source, see "[Link your test step to a data source](#)" on page 552.

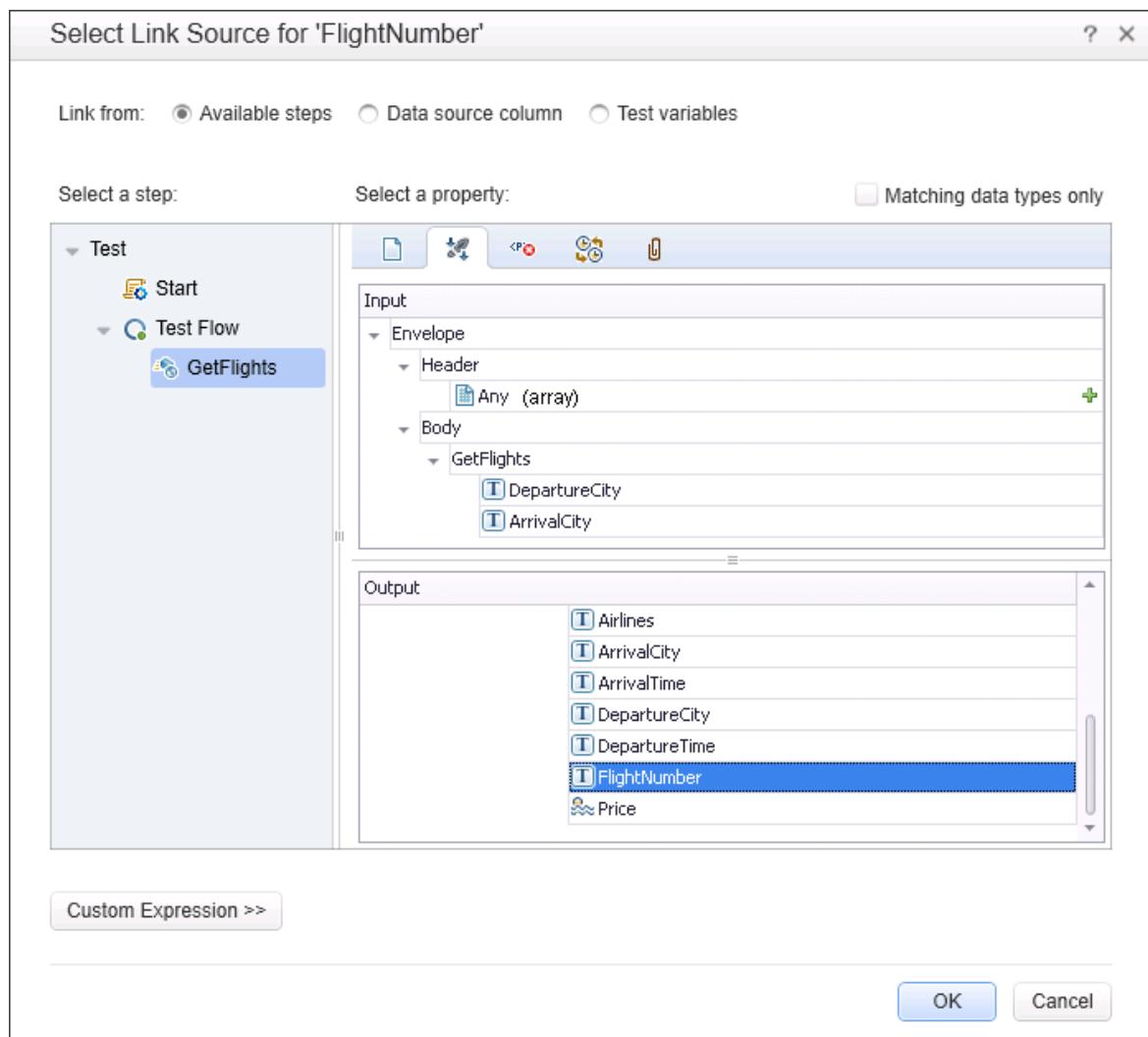
## Linking Property Values to Other Test Steps

### **Relevant for: API testing only**

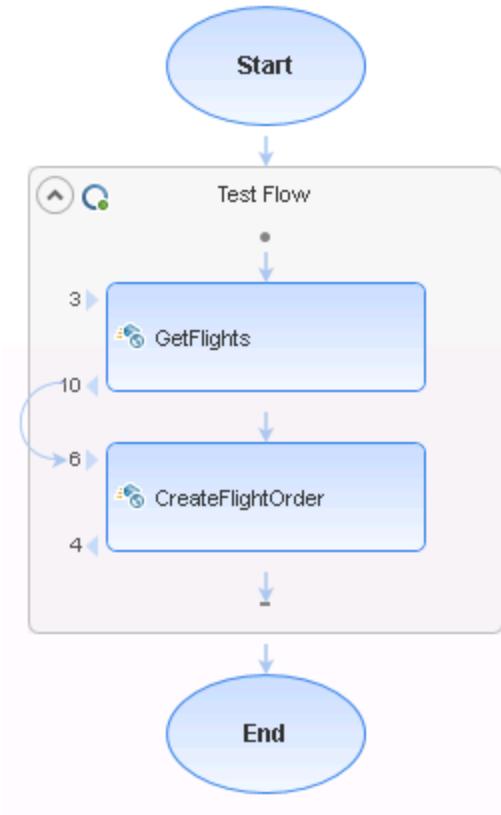
In some instances, your application passes data from one application process to another. Thus, the input value that one process is calling is reliant upon the output value of another process.

Therefore, when testing the behavior and performance of your application in a test, you must be able to mimic this passing of data. UFT enables you to do this through linking the property values of one step to the property values of another step.

Linking step property values is much the same as linking a step's property values to a data source. You use the Select Link Source dialog box and select the option **Available Steps**.



After you link the steps together, the canvas indicates the connection between the test steps:



For task details, see ["Link your test step to another step" on page 552](#).

## Outgoing Links

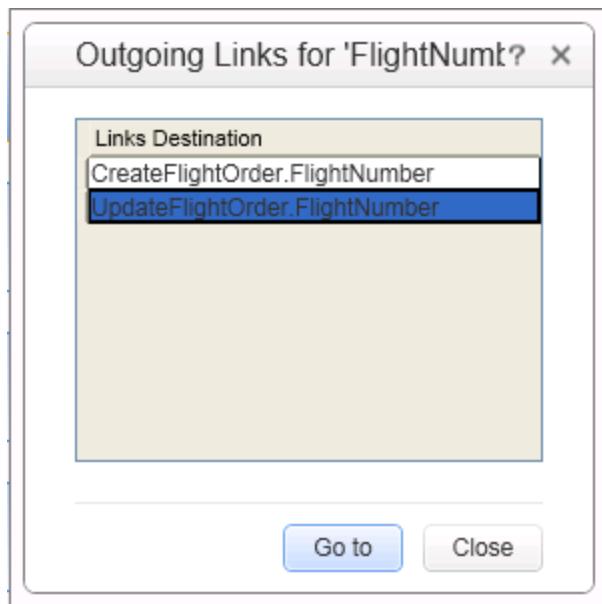
### Relevant for: API testing only

When you link a property to another step's property, UFT indicates this with a **Outgoing Links** button in the property's value column in the Properties pane's **Input/Checkpoints** tab.

Checkpoints	Validate	Expected Value
Properties		
Result	<input checked="" type="checkbox"/>	= <input type="button" value="▼"/> <input type="button" value="▼"/>

By clicking on the button, you can view a list of the properties that link to this output property. This is especially helpful when linking other steps to an output property multiple times to enable you to view the flow of data throughout your test. Likewise, this helps you ensure that the test steps match the flow of properties/parameters as they do in your application.

When you click the **Outgoing Links** button, UFT opens a list of the target properties that link to this value. Using the **Go to** button, you can navigate directly to the linked property. This selects the target step on the canvas and the target property in the Properties pane.



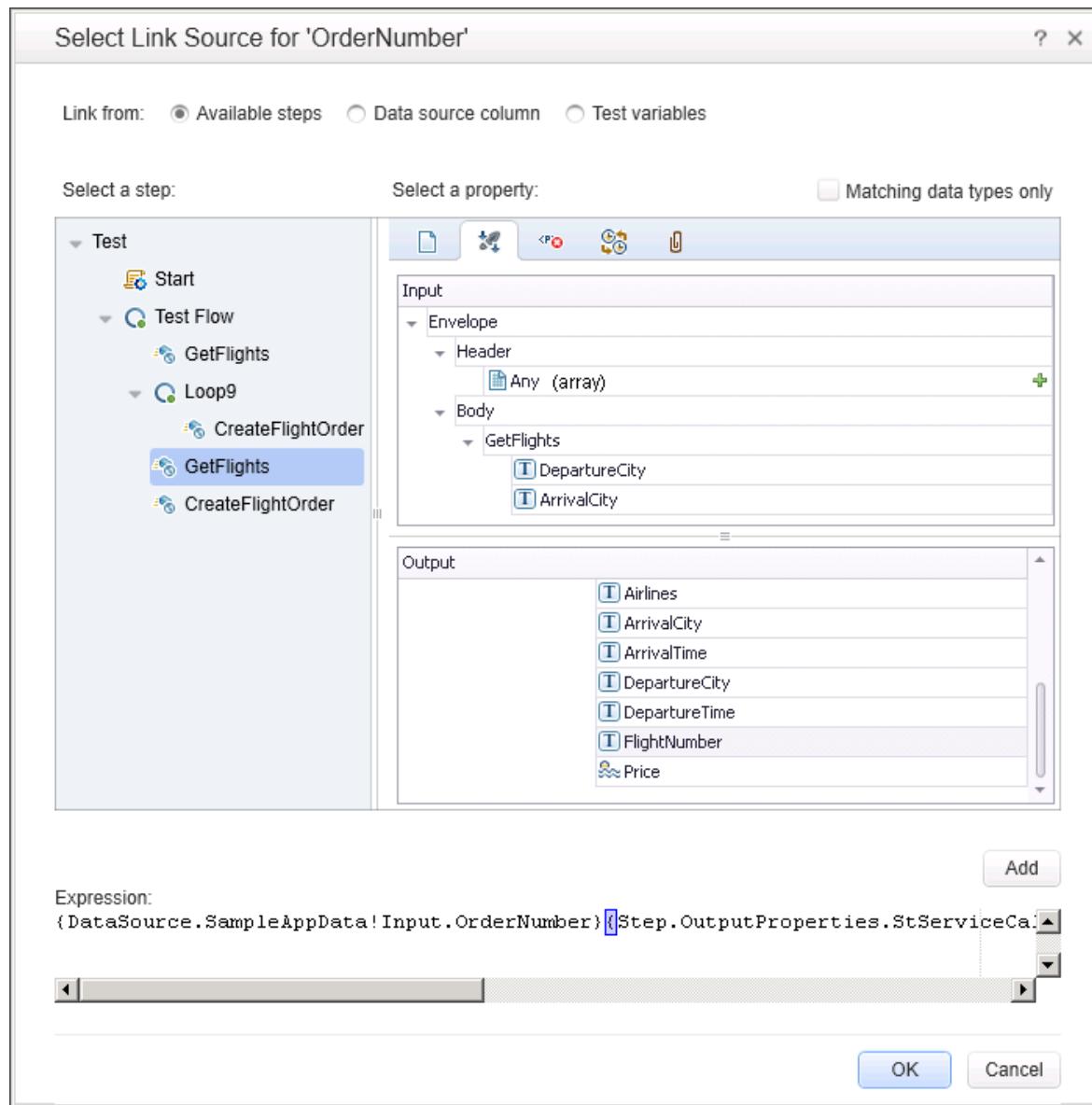
## Linking Property Values to Multiple Sources of Data

### **Relevant for: API testing only**

If the input of a particular application process in your application is linked to both a data source and the result of a previous application process, you can create property value expressions in UFT that reflect this process.

You do this by linking the property value of a test step to multiple sources of data and then creating custom expressions for the property value using a combination of the linking methods.

Using the Select Link Source dialog box, you create these custom expressions in any combination:



The custom expression is then added as the property value for your test step and runs accordingly when you run your test.

For task details, see "[Link your test step to multiple sources](#)" on page 553.

## Data Assignment in Arrays

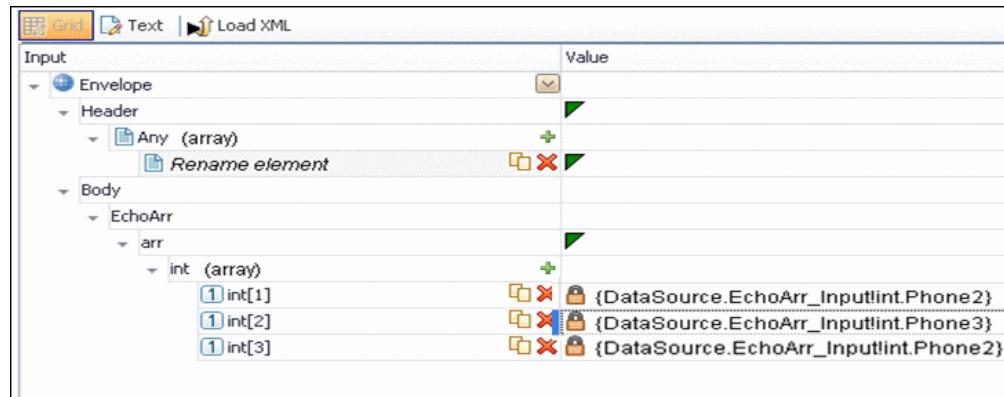
### **Relevant for: API testing only**

When your step has properties that are arrays, you can assign them data as a fixed-size array or through data relations:

**Fixed-Size  
Array  
Assignment**

In the fixed-size method, you assign each element of a fixed-size array to any column in a data table. You can assign each array element to a different data table column.

The following example shows three array elements assigned to different columns of a data table.

**Through  
Data  
Relations**

When you have one or more data relations defined, as described in ["Create a new child relation" on page 567](#), UFT assigns data from a single column. You link the first element of the array to a column in a data source.

In order for UFT to assign data based on a data relation, the following conditions must be present:

- The data source in the link is defined in a data relation.
- The parent data source is attached to the loop containing the step.
- Only the first element of the array is mapped to the child data source. If you map another array element to a different column, simple based assignment will be used.

The following example shows a single array element linked to a child data source. In this example, all elements of the array will take values from the same column, using the data-relation based assignment.

The screenshot shows the Data pane with the following structure:

- Input** tab selected.
- Envelope** node expanded, showing **Header** and **Body**.
- Header** node expanded, showing **Any (array)**.
- Body** node expanded, showing **EchoArr**, which has **arr** and **int (array)** children.
- int (array)** node expanded, showing **int[1]**.
- A link icon (blue square with a white number 1) is positioned next to the **int[1]** node, indicating it is linked to a data relation.
- A red X icon is positioned next to the link icon.
- A green checkmark icon is positioned next to the link icon.
- A lock icon is positioned next to the link icon.
- The link is labeled **{DataSource.EchoArr\_Input!int}**.

The link to the first element indicates the column from which the values for all of the array elements will be taken.

The screenshot shows the Data pane with the following structure:

- Data** tab selected.
- Current Test** node expanded, showing **EchoArr\_Input**, which has **EchoArr\_Input!int** and **EchoArr\_Input>MainDetails** children.
- EchoArr\_Input>MainDetails** node is highlighted with a yellow background.
- MainDetailsKey** table is displayed with the following data:

	A1	MainDetailsKey
1	MainDetailsKey	
2		1
3		
4		
5		
6		
7		
8		
9		
10		
11		

If you create a simple link to data, and the data source is already designated as a child in a data relation, the behavior will be relation-based.

Data-driven array elements will always be assigned using the data relation based assignment, using the column assigned to the first element.

#### Data Assignment for Array Checkpoints

For relation-based data assignment, the drop down list adjacent to the name of the parent array node provides the following options for checkpoint validation:

- **Fixed**. Checks that each of the returned array elements matches the corresponding array element in the data table in the Data Pane. Each array is marked by an index number, as it checks the arrays by their index.
- **All**. Checks that all of the returned array elements match the array element in the Data Pane. In this mode, arrays are not marked by an index number. For example, if a property in the first array is marked  $\geq 2$  and the

	<p>same property in another array element is set to &lt;=10, the test run will check that all returned values are between 2 and 10.</p> <ul style="list-style-type: none"><li>• <b>Contains.</b> Checks that at least one of the returned array elements matches the value of the property in the Checkpoints pane. In this mode, arrays are not marked by an index number.</li><li>• <b>Fixed.</b> Checks that each of the returned array elements matches the corresponding array element in the data table in the Data Pane. Each array is marked by an index number, as it checks the arrays by their index.</li><li>• <b>All.</b> Checks that all of the returned array elements match the array element in the Data Pane. In this mode, arrays are not marked by an index number. For example, if a property in the first array is marked &gt;= 2 and the same property in another array element is set to &lt;=10, the test run will check that all returned values are between 2 and 10.</li><li>• <b>Contains.</b> Checks that at least one of the returned array elements matches the value of the property in the Checkpoints pane. In this mode, arrays are not marked by an index number.</li></ul>
--	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## How to Add Data Sources to an API Test

For details about data handling for GUI tests and the GUI testing Data Pane, see "["Data Pane" on page 63](#).

### Relevant for: API testing only

The following steps provide you with different options for creating a data source.

- "["Add an Excel data source" below](#)
- "["Add an XML data source" on the next page](#)
- "["Add a database data source" on the next page](#)
- "["Add a local data table" on page 551](#)

### Add an Excel data source

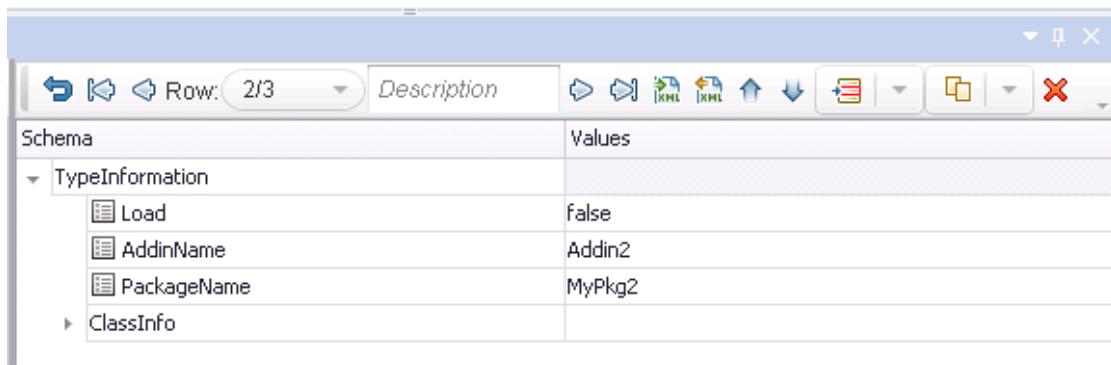
1. In the Data pane, click the **New Data Source** down arrow  and select **Excel**. The New/Change Excel Data Source Dialog Box opens.
2. Browse to the file and indicate if the first row is a header row.
3. Specify a name for the data source. If you do not provide a name, UFT sets it to the Excel file name after you navigate to the file.
4. Select whether to link to the Excel in its original location or create a local copy. The advantage of making a local copy is that it becomes portable with the test. However, any updates to the data at its original location will not be reflected in the migrated test.
5. Select **Allow other tools to override the data** option to enable ALM to run the test with different values. This allows you to overwrite the data from the imported Excel file with values from an ALM Data Resource.  
In addition, if you call a test or action with data assigned to its steps, this option allows you to edit the data in the called test or action.
6. Click **OK**.

- In the Data pane, select the data source under the **Excel** node. In the right pane, you can view and edit the values.

**Caution:** When entering values into the Excel sheet, you should use the Excel General format for values.

## Add an XML data source

- In the Data pane, click the **New Data Source** down arrow  and select **XML**. The New XML Data Source Dialog Box opens.
- Provide a **Data Source name**.
- Select the entity upon which to base the data source:
  - Schema file:** Browse to an .xsd file.
  - XML file:** Browse to an .xml file.
  - Use existing:** Browse to an existing XML-based data source from another test.
- Click **OK**.
- Select the data source's node in the Data pane.



Schema	Values
TypeInformation	
Load	false
AddinName	Addin2
PackageName	MyPkg2
ClassInfo	

- Edit the XML data in the grid. Click the **Add rows** button  to add row values.
- To load XML values, click the **Load data from an XML file** button .

## Add a database data source

- In the Data pane, click the **New Data Source** down arrow  and select **Database**. The Add New Database Data Source Wizard opens. The **Set Database Connection** page opens.
- In the Set Database Connection pane, add the database connection information as follows:

- |                             |                                                                                                                      |
|-----------------------------|----------------------------------------------------------------------------------------------------------------------|
| For an<br><b>OleDb</b> type | a. Select <b>OleDb</b> as the <b>Connection type</b> .<br>b. Set the connection string in one of the following ways: |
|-----------------------------|----------------------------------------------------------------------------------------------------------------------|

connection:	<ul style="list-style-type: none"> <li>◦ Paste an OLE DB connection string into the text area.</li> <li>◦ Select a previously defined connection string from the drop down.</li> <li>◦ Click the <b>Build Connection String</b> button  to use Microsoft's Data Link Properties dialog box.</li> </ul> <p>c. Click <b>Test Connection</b> to verify that the connection is valid.</p> <p>d. Click <b>Next</b>. This button is only available for valid connections.</p>
<b>For an ODBC type connection:</b>	<p>a. Select <b>ODBC</b> as the <b>Connection type</b>.</p> <p>b. Click the <b>Build Connection String</b> button  to select a DSN type.</p> <p>c. Select a DSN type and provide the necessary credentials. To create or edit a DSN entry, click the <b>Manage ODBC Data Sources</b> button. For details, click the <b>Help</b> button in the ODBC Data Source Administrator dialog box. After you close the administrator dialog box, click the <b>Reload DNS list</b> button to refresh the list.</p> <p>d. Test the connection. After the confirmation, click <b>OK</b>.</p>

3. In the **Set SQL Statement** page:

- a. Provide a unique name for the data source, not used by another data source. If you do not provide a name, the data source is automatically assigned the table name after you build the query.
- b. Provide an SQL statement in one of the following ways:

<b>Paste an existing statement into the text area</b>	Take your preexisting statement and enter it.
<b>Select a previously defined statement</b>	From the drop-down list, select a previously-used statement.
<b>Use the Query Designer</b>	<ol style="list-style-type: none"> <li>i. Click the <b>Build Query Statement</b> button  to open the Query Designer Dialog Box.</li> <li>ii. Select a table or view. The Query Designer lists all of the rows in the pane below and displays the corresponding SQL statement in the preview pane.</li> <li>iii. Enable <b>Auto execute</b> to view the results of your query as you make changes in the upper pane.</li> <li>iv. By default, the Query Designer selects all columns in the table. To remove a column from the query, clear the <b>Output</b> check box for that column.</li> <li>v. To set a sorting order, select <b>Ascend</b> or <b>Descend</b> from the <b>Sort</b> column drop-down. If you have multiple rows that you are sorting, provide the <b>Sort Order</b> beginning with 1. The Query Designer adds an <b>ORDER BY</b> clause to the preview pane.</li> <li>vi. To add a <b>GROUP BY</b> or <b>DISTINCT</b> clause to your statements, select the appropriate check box.</li> <li>vii. Click <b>OK</b> to accept the query and return to the wizard.</li> </ol>

- c. In the wizard's **Set SQL Statement** page, click the **Check SQL Statement** button.
- d. After the Query Preview is finished, click **Close** to return to the wizard.
- e. Click **Finish**. UFT shows the query details in the Properties pane's "Database Data Source Properties Tab".

**Note:** By default the Data pane shows the first ten rows of the query data. To change the number of displayed rows, use the General Pane.

## Add a local data table

1. In the Data pane, click the **New Data Source** down arrow  and select **Local Table**. The New Local Table Data Source Dialog Box opens.
2. Click the **Add**  button to create a column in the data table.
3. In the columns list, specify a column name and description. Select a data type from the drop down list.
4. Repeat steps **3** and **4** for each column you want to create.
5. When you are finished, click **OK**.
6. In the Data pane, select the table's node in the left pane and move within the table using the keyboard arrows. Type within the cells to manually set values.

# How to Assign Data to API Test/Component Steps

## Relevant for: API testing only

This task describes the different ways to link test step properties (both for input properties and checkpoint properties) to data sources :

- "Manually enter the data for your test step properties" below
- "Link your test step to a data source" on the next page
- "Link your test step to another step" on the next page
- "Link your test step to multiple sources" on page 553
- "Link your test steps to a test or user-defined variable" on page 554
- "Data drive the test step" on page 554
- "How to Assign Data to API Test/Component Steps" above

## Manually enter the data for your test step properties

1. In the canvas, select the test step to assign property values.
2. In the Properties pane, open the **Input/Checkpoints** tab .

3. In the **Value** cell for the property, enter the value.

**Note:** If you are working with a step that uses XML or JSON input, you can also enter the property values using the **Text** view in the Input section.

**Tips:**

- If the property value is a string, enter it exactly as you expect it appear (including spaces and special characters.)
- If you want to use the values from the previous test run, click the **Load from Replay** button.

## Link your test step to a data source

1. In the Data pane, associate a data source with your test.
2. In the canvas, select the test step for which you want to assign property values.
3. In the Properties pane, open the **Input/Checkpoints** tab .
4. In the **Value** cell for the property, click the **Link to data source** button . The Select Link Source dialog box opens.
5. In the Select Link Source dialog box, select the **Data source column** option. A list of all data sources is displayed.
6. Select the data source containing your property's data value. A list of all available data columns is displayed.
7. In the data columns list, select the corresponding column name for your test step property and click **OK**.

The property name in the Input/Checkpoints tab is now displayed with the associated data source name and data source value.

**Note:**

- You can only link properties to data sources only if the data source is attached in a loop, such as **Test Flow** or custom loops.
- Linking a leaf node to a complex XML node is supported only for string type data.

## Link your test step to another step

1. In the canvas, select the step for which you want to link a property value.
2. In the Properties pane, open the **Input/Checkpoints** tab .
3. In the **Value** cell for the property, click the **Link to data source** button . The Select Link Source dialog box opens.
4. In the Select Link Source dialog box, select the **Available steps** option. A list of all steps preceding

the selected step is displayed.

5. In the list of available steps, select the step you want to link to the selected property value. A list of available properties is displayed in the right pane.
6. In the right pane, open the tab containing the property to which to link.

**Note:** If you are linking to the output of a previous step, open the **Input/Checkpoints** tab .

7. Select the property to which to link and click **OK**.

The property value is displayed in the **Value** column of the input property name with an expression representing the output of the linked step. The canvas also displays an arrow connecting the step property values.

**Note:** The Properties pane displays a down arrow next to any property that is set to be used as output data for another step. Click the icon to display a list of all steps linked to the selected property. For details, see "[Outgoing Links](#)" on page 543.

## Link your test step to multiple sources

1. If necessary, add a data source. For details on adding data sources, see "[How to Add Data Sources to an API Test](#)" on page 548.
2. In the canvas, select the step for which you want to link a property value.
3. In the Properties pane, open the **Input/Checkpoints** tab .
4. In the **Value** cell for the property, click the **Link to data source** button . The Select Link Source dialog box opens.
5. In the Select Link Source dialog box, click the **Custom Expression** button to expand the expression area.
6. Create your expression, with any combination of methods. Click **Add** after linking to each source to add the value to your custom expression:
  - Manually enter the expression
  - Link to data source values, as described in "[Link your test step to a data source](#)"

**Note:** You can make a custom expression that includes multiple links to data sources.

- Link to another step's property, as described in "[Link your test step to another step](#)"

**Caution:** If you are entering a custom string, make sure to add the string exactly as you want it to appear, including spaces and special characters.

You can use these methods in any order and in any manner as needed to create your expression.

7. When you are finished entering all values, click **OK**.

The custom expression is displayed in the **Value** column for the property name in the **Input/Checkpoints** tab exactly as you entered it in the expression area.

## Link your test steps to a test or user-defined variable

1. In the canvas, select the step you want to link to a test or user-defined variable.
2. In the Properties pane, open the **Input/Checkpoints** tab .
3. In the **Value** cell for the property you want to link to a variable value, click the **Link to data source** button . The Select Link Source dialog box opens.
4. In the Select Link Source dialog box, select the **Test variables** option. The list of available variables is displayed.
5. In the list of available variables, select the variable for the link and click **OK**.

The value for the selected property is displayed as the expression of the test variable.

## Data drive the test step

1. In the Input/Checkpoints tab, click the **Data Drive**  button. The Data Driving Dialog Box opens.
2. In the Data Driving dialog box, choose a Data provider: **Excel** or **XML**.
3. Select the properties upon which you want to apply data driving: **input properties**, **checkpoints**, or both.
4. If you specified an **Excel** provider and data-driving for both input properties and checkpoints, specify whether to place the data for both sections in the same Excel worksheet or different ones.
5. Indicate whether you want to **Configure '<loop name>' as a For Each loop using the new data source**. This option repeats the steps in the loop frame according to the number of data rows in the Excel or XML data source. If you disable this option, you can manually set the number of iterations via the loop properties. This setting overrides the general loop properties. For details, see "["Loop Activity" on page 438](#)".
6. Click **OK**.
7. UFT prompts you to confirm the action. Click **OK**.

UFT populates the value column with data expressions. The expressions are preceded by informational icons, indicating read-only status or unmatched data types.

**Note:** If your properties are within an array, you must create at least one array element to allow data driving. To add an array element, select the array's parent node and click the **Add** button . If you do not add at least one element, then the array will not be data driven.

# How to Assign Data to API Test Steps - Tutorial

## Relevant for: API testing only

This tutorial teaches you how to assign data to your test steps. The test steps are based upon the sample Flight API application provided with UFT.

**Note:** For the task related to this scenario, see ["How to Assign Data to API Test/Component Steps" on page 551](#).

This tutorial includes the following steps:

- "Prerequisite - import the Web Service methods for the sample application" below
- "Associate a data source with your test" below
- "Create your test steps" on the next page
- "Manually enter the input properties for the GetFlights step" on the next page
- "Link the CreateFlightOrder input properties to the data source" on the next page
- "Link the Report step input properties to the CreateFlightOrder step" on page 557
- "View the run results" on page 557

## 1. Prerequisite - import the Web Service methods for the sample application

For this scenario, you use the **GetFlights** and **CreateFlights** methods from the sample Flight API application.

To import the service, do the following:

- a. Open the Flight API application from the **Start** menu or screen or the file system (<UFT installation folder>\samples\flight\_service\HPflights\_Service.exe).
- b. Type help to display the service details.
- c. Copy the URL for the Web Service of the Flight API Application.
- d. Import the Web Service into UFT. For details, see ["How to Import a WSDL-Based Web Service" on page 512](#).

## 2. Associate a data source with your test

Add the sample application Excel data source to your test. This file (SampleAppData.xlsx) is found in the <UFT installation>\samples\flight\_service directory.

- a. In the Data pane, click the **New Data Source** button  and select **Excel**.
- b. In the New/Change Excel Data Source Dialog Box, navigate to sample application Excel file. Click **OK** to add the Excel file to the test.

The data source is displayed in the **Data** pane.

### 3. Create your test steps

From the Toolbox pane, drag the following steps to the canvas in order:

- **GetFlights** (found in the **Local Activities > Web Services** node)
- **CreateFlightOrder** (found in the **Local Activities > Web Services** node)
- **Report Message** (found in the **Miscellaneous** node)

### 4. Manually enter the input properties for the GetFlights step

To provide property values for the **GetFlights** step, use the first method for providing data by manually entering the property values.

To provide the property values, do the following:

- a. In the canvas, make sure that the **GetFlights** step is selected.
- b. In the Properties pane, open the **Input/Checkpoints** tab .
- c. In the Input/Checkpoints tab, manually select the following values from the property value drop-down lists:
  - **DepartureCity:** Denver
  - **Arrival City:** Los Angeles

### 5. Link the CreateFlightOrder input properties to the data source

When the **GetFlights** method runs in the Flight API application, it automatically creates a number of output properties, including the airline number, price, a flight number, and so on.

In this step, you link the input property values of the **CreateFlightOrder** step both to the output of the **GetFlights** step and to the sample Excel file you associated with your test.

- a. In the canvas, select the **CreateFlightOrder** step.
- b. In the Properties pane, open the **Input/Checkpoints** tab .
- c. In the Input/Checkpoints tab, in the **Value** cell for the **FlightNumber** input property, select the **Link to data** source button .
- d. In the Select Link Source dialog box, link the **FlightNumber** property to the **GetFlights** step output property of the same name. When prompted if you would like to link the selected property as part of a loop, select **No**.

For details on linking to another test step property, see "[Link your test step to another step](#)" on page 552.

**Note:** By default, the **FlightNumber** property is not visible. Expand the **GetFlightsResult** node and click the **Add** button  to expand all the output properties.

- e. In the Input/Checkpoints tab, link the remaining input properties to the relevant columns in the

Excel data source.

**Note:** Make sure to clear the NIL property on any property values.

## 6. Link the Report step input properties to the CreateFlightOrder step

For the final step of this tutorial, you will create a custom expression to mimic the Flight API application's passing of flight data to a flight booking Web site page. For this, create the custom message that the results display.

- a. In the canvas, select the **Report Message** step.
- b. In the Properties pane, open the **Input/Checkpoints** tab .
- c. In the Input/Checkpoints tab, in the **Value** cell of the **Message** property, select the **Link to data source** button .
- d. In the Select Link Source dialog box, click the **Custom Expression** button to display the expression area.
- e. In the expression area, enter the text "Your flight order number is (with an extra space at the end).
- f. In the Select Link Source dialog box, select the Available step option.
- g. In the list of available steps (left pane), select the **CreateFlightOrder** step.
- h. In CreateFlightOrder properties list in the right pane, in the **Output Properties** section, expand the **CreateFlightOrderResult** node.
- i. In the output properties list, select the **OrderNumber** property.
- j. Click **Add** to add this to the custom expression.
- k. Click **OK** in the Select Link Source dialog box to add this expression.
- l. The expression Your flight order number is  
`{Step.OutputProperties.StServiceCallActivity4.Body.CreateFlightOrderResponse.CreateFlightOrderResult.OrderNumber}` appears in the Value column of the Message property

## 7. View the run results

Run the test. When the run results open, explore the nodes in the Test Flowtree and check the results of your steps in step summary.

In the captured data for the CreateFlight step and the Report Message step, you should see the data values passed between steps.

# How to Define API Test Properties or User/System Variables

## Relevant for: API testing only

The following steps describe how to define test properties, user variables, and operating system variables. These settings are optional.

- "Define test parameters" below
- "Define user variables" below
- "Set user variable values" below
- "Define user variable profiles" on the next page
- "Set OS variable values for the test - optional" on the next page

## Define test parameters

This step applies if you want to define custom parameters that can be used by all steps in the test, with the ability to assign data from a data source.

1. Click in a blank area of the canvas. In the Properties pane, open the **Test Input Parameters** view .
2. Click the **Add** button to define a new input or output parameter.
3. In the Add Input/Output Property/Parameter Dialog Box, specify a parameter name and data type. All types that were defined in any reference file, are available.
4. Click in the **Default Value** column and specify a value.

## Define user variables

You can create user variables and set their values. You can define multiple profiles for the variable values. You select a profile to be active before the test run. For details, see "Set user variable values" below.

1. Click in a blank area of the canvas. In the Properties pane, open the **Test Variables** tab .
2. Click the **Add User Variable** button  to define a new user variable.

## Set user variable values

You can set values for variables in one of the following ways:

- In the Properties pane's **Test Variables** view, click in the **Profile** column and manually enter values.
- Open the Toolbox pane and drag the **Set Test Variable** activity from the **Miscellaneous** category into the **Test Flow** (or loop). In the Properties pane, define the variable key and value. To obtain values,

click the Link to a Data Source button in the row. In the Select Link Source Dialog Box, select the **Test Variables** option. Select a name and value for the **Variable key** and **Variable value**.

- Open the Properties pane's **Events** view for the step or define a Custom Code activity. Edit the event handler code and assign a value using the TestProfile object. The following example sets the value of the Region user variable to NE.

```
activity.Context.TestProfile.SetVariableValue("Region", "NE");
```

Repeat this step for each variable for which you want to set values.

## Define user variable profiles

This step applies only if you created user variables as described in the above steps.

1. Define one or more user variables as described above.
2. Click the **Add New Profile** button .
3. In the New Test Profile Dialog Box, specify a name and indicate the profile (if any) from which to copy the properties. If you do not copy the properties from an existing profile, UFT copies the user variables from the active profile without its values.
4. To rename or delete a profile, click the **Manage Profiles** button . Click **Remove** or **Rename**.
5. Click the **Compare** button  to display the profiles side-by-side.
6. Only the active profile is accessed during the test run. To make a profile active, select it from the **Active Profile** list.

## Set OS variable values for the test - optional

This step lets you set global operating system variables that will apply to all steps in the current test run.

1. From the Toolbox pane and drag the **Set OS Environment Variable** activity from the **System** category into the Test Flow or another custom loop.
2. In the Properties pane's Input/Checkpoints tab, define the variable key and value or click the **Link to Source** button  to specify a data source.

**Tip:** To view a list of the test variables, click in a blank area of the canvas. In the Properties pane, open the **Test Variables** tab . The System variables are listed in the lower pane.

# How to Parameterize XML Data

## Relevant for: API testing only

This task describes how to parameterize XML data. Suppose you pasted or loaded an XML file as the request body for your step. Using the parameterization capabilities, you can replace a constant value with a data source expression.

### 1. Prerequisites

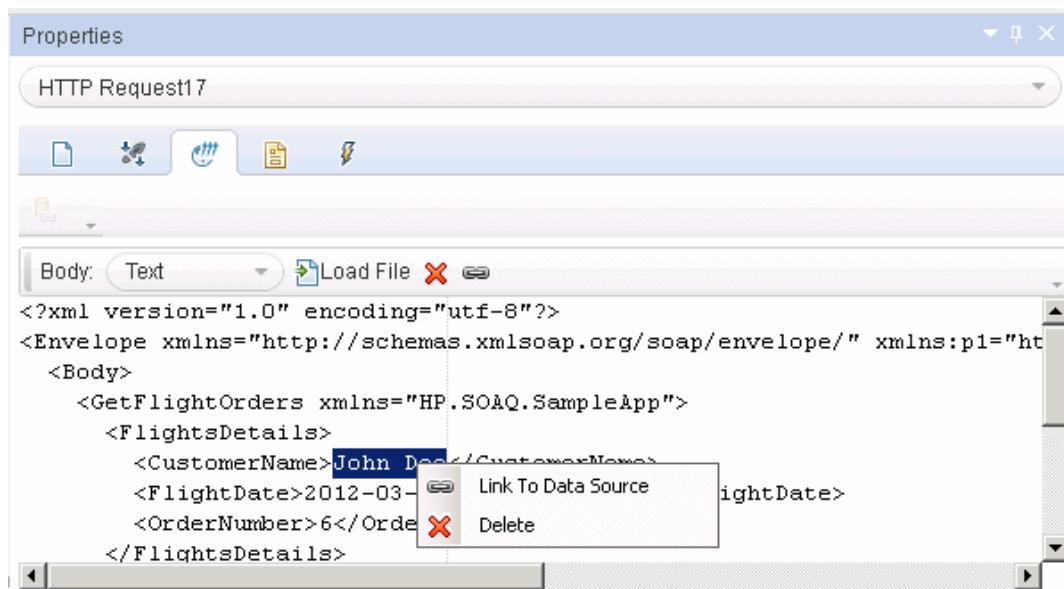
Create a test step that uses XML text in the request body. This applies to Network and XML type activities. If you intend to parameterize the data with a data source, import or create the data source. For details, see ["How to Add Data Sources to an API Test" on page 548](#).

### 2. Add the XML body text

- a. Using the Properties pane, open the **HTTP** tab .
- b. Click the **Load File** button and load an XML file or paste XML code directly into the **Body** area.

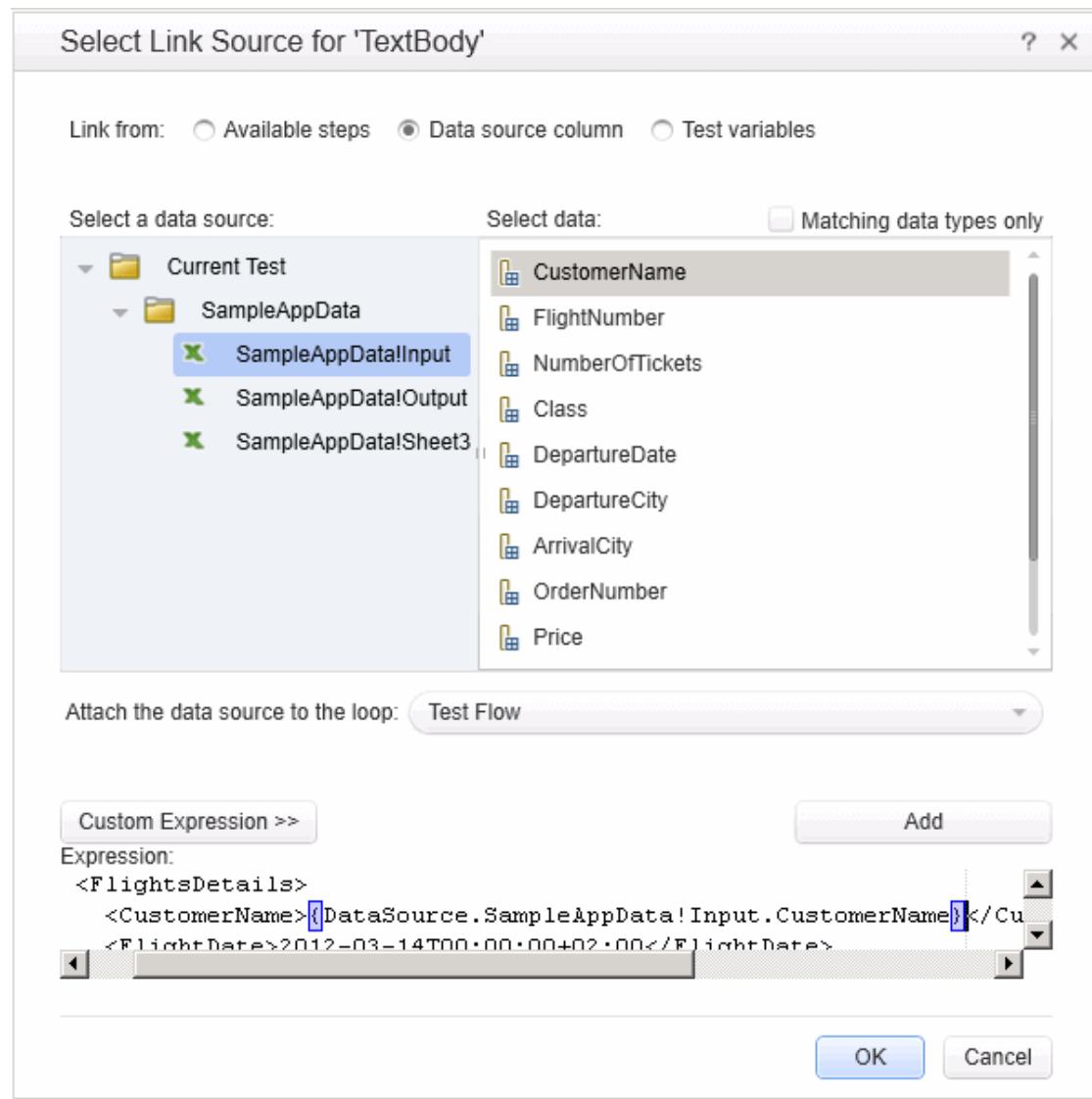
### 3. Select the text to parameterize

- a. In the **HTTP** tab, select **Text** from the **Body** type drop-down list.
- b. In the Body area, select the text value that you want to replace and select **Link to Data Source** from the right-click menu.



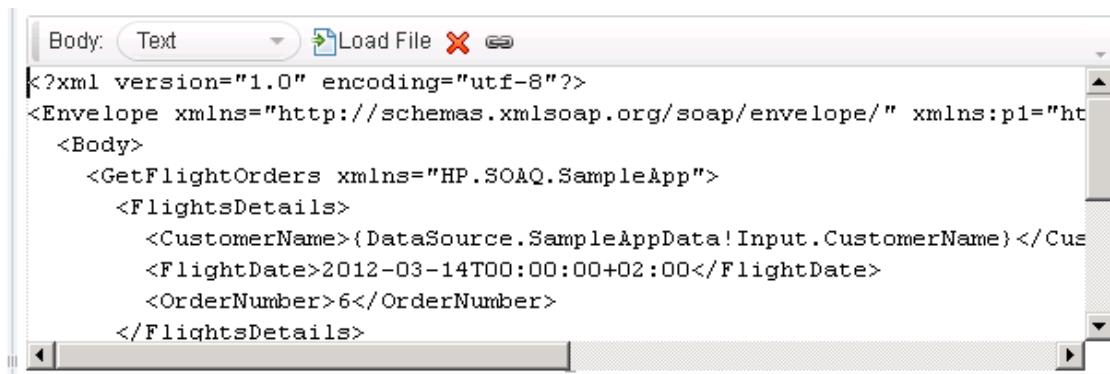
### 4. Select a link source

In the Select Link Source dialog box, select a data source type, such as **Available steps** or **Data source column**. Select the node that you want to use for parameterization. Click **Add**. Note that the expression changed.



##### 5. Verify the value in the Properties pane

In the Properties pane, check the Body area, and verify that the constant value was replaced with the data source expression.



The screenshot shows the Properties pane with the 'Body' tab selected. The pane displays an XML snippet:

```
<?xml version="1.0" encoding="utf-8"?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/" xmlns:p1="ht
<Body>
 <GetFlightOrders xmlns="HP.SO&Q.SampleApp">
 <FlightsDetails>
 <CustomerName>(DataSource.SampleAppData!Input.CustomerName)</Cus
 <FlightDate>2012-03-14T00:00:00+02:00</FlightDate>
 <OrderNumber>6</OrderNumber>
 </FlightsDetails>
```

## How to Data Drive Array Checkpoints

### Relevant for: API testing only

This task describes how to set checkpoints for elements of an array through data driving.

This task includes the following steps:

- "Enable active content on your computer" below
- "Add a step with an array output" below
- "Add an array element" below
- "Data Drive the array" on the next page
- "Set the evaluation expression" on the next page
- "Provide data for the array" on the next page
- "Select an array validation method" on the next page
- "Set the number of iterations - optional" on the next page

#### 1. **Enable active content on your computer**

The run results show array checkpoint results in a tree. To enable this view, configure your browser to allow active content as described in ["How to Set Array Checkpoints for Test Steps" on page 410](#).

#### 2. **Add a step with an array output**

Add a test step with output properties in the form of an array.

#### 3. **Add an array element**

- a. In the Properties pane, open the **Input/Checkpoints** tab .
- b. In the Checkpoints area, use the plus button  in the row of the parent node, to add an array element.

- 
- 
- 
- c. Provide property values. These values are transferred to the Data pane during data driving. If you do not provide data, include the array element by selecting the triangle icon in the parent node.

#### 4. **Data Drive the array**

- a. Select the array's parent node and click the **Data Drive** button .
- b. In the Data Driving Dialog Box, select **Only Input, Checkpoints** or **Both Input and Checkpoints** as a **Data Driven Section** option.
- c. Click **OK**. The data driving mechanism informs you that it succeeded.

#### 5. **Set the evaluation expression**

In the Properties pane, adjacent to the data driving expression, select an evaluation operator, such as `=`, `<`, and so forth.

#### 6. **Provide data for the array**

- a. Once you data drive an array, you do not set property values from the **Checkpoints** pane. Instead, you edit the table in the Data pane. In the Data pane, which opens automatically, click the node in the left pane, corresponding to the array element that is to be validated.
- b. Enter the iteration number to which to checkpoint should be applied in the `MainDetails` column of the `<array_name>` and `MainDetails` tables. A "1" indicates the first iteration. If you have several columns with "1" as the iteration number, the checkpoints will be validated against all of those values.

#### 7. **Select an array validation method**

In the Input/Checkpoints tab, expand the drop down adjacent to the name of the parent array node. Select one of the following:

- **Fixed.** Checks that each of the returned array elements matches the corresponding array element in the data table in the Data pane. Each array is marked by an index number, as it checks the arrays by their index.
- **All.** Checks that all of the returned array elements match the array element in the Data pane. In this mode, arrays are not marked by an index number. For example, if a property in the first array is marked `>= 2` and the same property in another array element is set to `<=10`, the test run will check that all returned values are between 2 and 10.
- **Contains.** Checks that at least one of the returned array elements matches the value of the property in the **Checkpoints** pane. In this mode, arrays are not marked by an index number.

#### 8. **Set the number of iterations - optional**

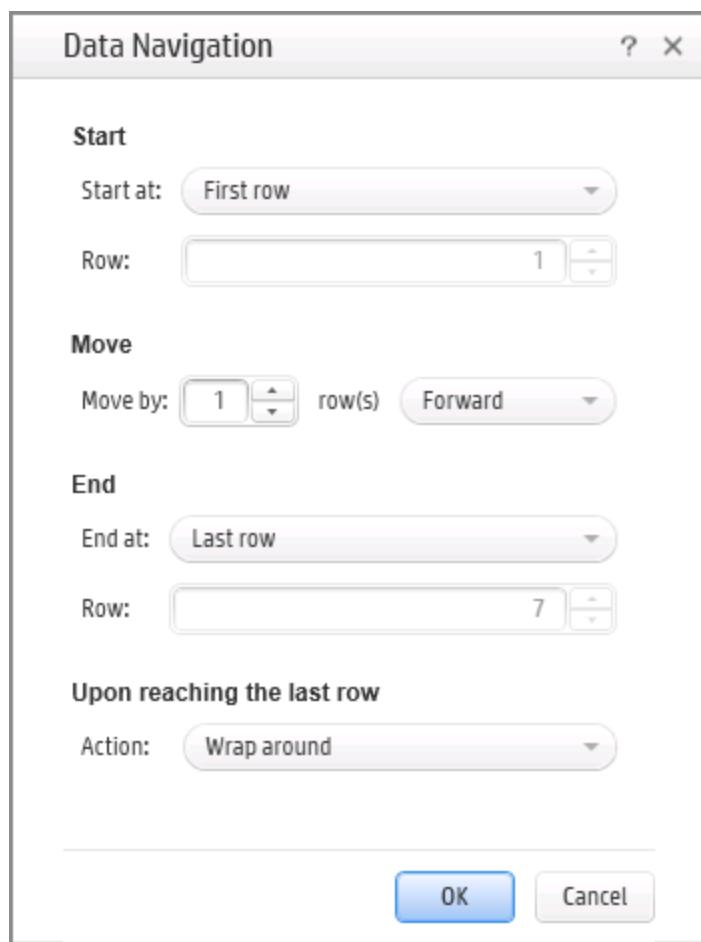
If you selected the **Configure Test Flow as a ForEach loop using the new data source option** in the Data Driving Dialog Box, you can skip this step. If not, click in the **Test Flow** or **Loop** box and

open the Properties pane. Set the test flow properties, such as the number of iterations or loop type.

## Navigating Within a Data Source

### Relevant for: API testing only

Sometimes, when you are running your test, your data is stored in multiple sources and called by many different step properties in the test. In these cases, you can instruct UFT how to use the data sources, including where in the data source to begin calling data values, how to move through the data source, and where to stop using values from the data source. You specify these settings for each of the data sources associated with your test in the Data Navigation Dialog Box



This is useful to test your application's performance with constantly varying sets of data. In addition, this enables you to maintain your data source while still using it in a test. For example, if you are adding new data to a data source that is being used in a test, but the new data is not complete, you can instruct UFT to begin at the row containing the old data and stop before calling the new data values currently under construction.

The Data Navigation settings you create work with the loop settings for the Test Flow or selected test loop:

- If the loop is configured as a **ForEach** type, and you assign a specific data source as the loop's collection, then the Data Navigation settings affect the number of iterations of the loop. Because you specify the number of rows to use from the selected data source, the test runs the same number of iterations as the number of rows specified.
- For data sources that are not designated as the loop collection but whose values are called by steps within the loop, the Data Navigation policy affects the data differently. It indicates the order in which the values are called from the data source and assigned to steps within the loop. For example, if you specify in the Data Navigation dialog box to begin in the second row of your Excel data source, UFT populates the property value with the values from the second row (and so forth, as specified in the Data Navigation dialog box).

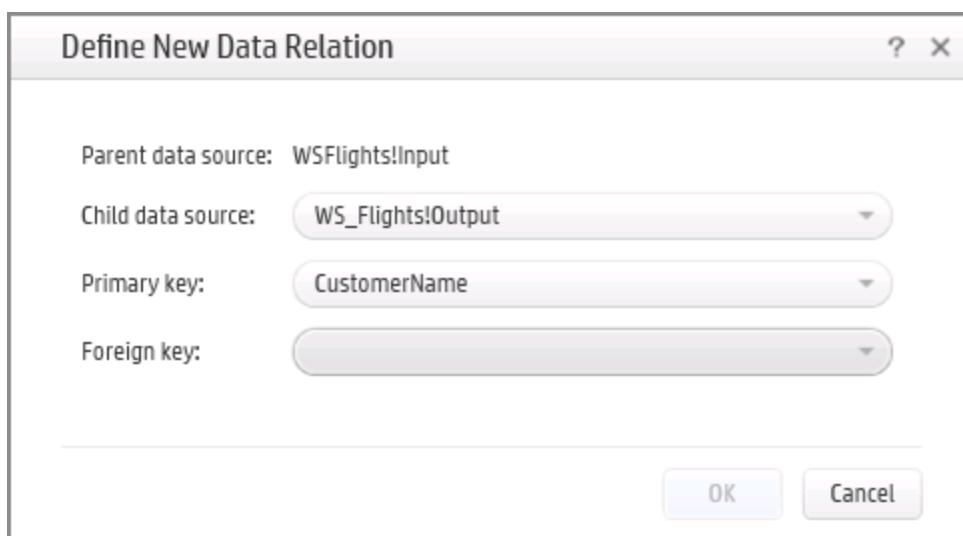
For task details, see "[How to Set the Data Source Navigation Properties](#)" on the next page

## Parent/Child Data Source Relations

### **Relevant for: API testing only**

If your test uses multiple data sources to provide data for your test step properties, you can create a hierarchy of relations between data sources that describes how to use the data sources in your test. You use data relations when you use one specific data source as the primary data source for your test but require other data sources to populate property values within the same test.

When you use data relations, you assign the primary data source to the Test Flow or selected test loop. Then, using the Define New Data Relation dialog box, you specify any necessary child data sources and map the columns of the parent data source to the columns of the child data source.



When the test runs, UFT substitutes the child data as specified.

For task details, see "[Create a new child relation](#)" on page 567.

# How to Set the Data Source Navigation Properties

## Relevant for: API testing only

This task describes how to set the navigation preferences for data stored in tables. You can set different navigation properties for each data source. For details, see ["Navigating Within a Data Source" on page 564](#).

This task includes the following steps:

- ["Add a data source to the Test Flow or test loop" below](#)
- ["Specify the navigation properties for the data source" below](#)
- ["Create a new child relation" on the next page](#)

## Add a data source to the Test Flow or test loop

1. Associate the necessary data sources with your test.
2. In the canvas, select the **Test Flow** or another test loop.
3. In the Properties pane, select the **Data Sources** tab . The list of all currently associated data sources is displayed.
4. In the Data Sources tab, click **Add**. The Attach Data Source to Loop Dialog Box.
5. In the Attach Data Source to Loop dialog box, select the data source to add to the loop and click **OK**.  
The selected data source is now added to the loop and you can use it to link property values.

## Specify the navigation properties for the data source

1. In the canvas, select the **Test Flow** or other test loop.
2. In the Properties pane, open the **Data Sources** tab . The list of data sources associated with the current loop is displayed.
3. In the Data Sources tab, click **Edit**. The Data Navigation Dialog Box opens.
4. In the Data Navigation dialog box, specify the **Start** and **End** rows.
5. Indicate the direction in which to move when retrieving data from the table, **Forward** or **Backward**, and the number of rows by which to advance for each iteration. Alternatively, you can indicate to move to a random row.
6. Specify the action to do when reaching the last row- **Wrap around** or **Keep using that row**. Click **OK**.

## Create a new child relation

This step enables you to specify parent-child data source relations between multiple data sources. You can use parent-child relations for **Excel**, **Local Table**, and **Database** type data sources.

1. Add at least two or more Excel, Local Table, or Database data sources.

**Note:** This can also be a single Excel file with two worksheets.

2. In the Data pane, select the data source that you want to designate as a parent.

3. In the Properties, open the Data Source Properties tab .

4. Click the **Add** button. The Define New/Edit Data Relation Dialog Box opens.

5. Specify the details of the new relation:

- The data source to use as the child data source
- The primary key - the data column in the parent data source
- The foreign key - the data column in the child data source. This column is used in place of the primary key column in the parent data source when running the test.

Click **OK** after specifying the data source details. The data relation is displayed in the Data Source Properties tab for the parent data source.

## Data Keywords

### Relevant for: API testing only

You can use keywords to customize the test run and validation. For example, **SKIP** omits a property in the request or in the validation.

UFT provides keywords for both input properties and checkpoints. You can set keywords in the following ways:

- Select the property and choose **Insert Keyword** from the right-click menu.
- Link to a data source containing the keyword.
- Type to keyword into the **Expected Value** column (checkpoints only).

For the manual options, make sure to use the required format by enclosing the keyword with hash (#) signs. Keywords are not case-sensitive.

## Input Keywords

The following keywords are supported for input properties:

Keyword	Description
#SKIP#	Omits this element from the XML of the request. This is useful for elements of SOAP requests, for which <code>minOccurs = 0</code>
#NIL#	Adds a <code>nil=true</code> attribute to the property's XML in the request.  <b>Example:</b> <code>&lt;name John Doe nil="true"&gt;&lt;/name&gt;</code>  If the XML element is not nillable, this is reported to the log.

## Checkpoint Keywords

The following keywords are supported for checkpoints:

Keyword	Description
#EXISTS#	Verifies that the element is present in the XML response. In this evaluation, the actual value is ignored—it only checks for the presence of a value. This is useful for SOAP response elements, for which <code>minOccurs = 0</code> .
#NOT_FOUND#	Verifies that the element is not present in the XML response. In this evaluation, the value is ignored. This is useful for SOAP response elements, for which <code>minOccurs = 0</code> .
#SKIP#	Informs the run engine to ignore this value when evaluating checkpoints.

To check if an element has a `nil="true"` attribute in its response, select or clear its NIL icon in the user interface.

# Troubleshooting and Limitations - Using Data in API Tests and Components

## **Relevant for: API testing only**

This section describes troubleshooting and limitations for working with data.

<b>General</b>	<ul style="list-style-type: none"><li>• If a specified data source is or becomes inaccessible, the test will not fail. The Errors pane and report, however, indicate that there was an error in retrieving the data.</li><li>• Keywords such as <b>#SKIP#</b>, and so forth, are not supported in the Input property grid. <b>Workaround:</b> Link to a data source that contains the keyword.</li><li>• When adding a referenced Excel data source, if the file requires special credentials (for example, a location on another domain or a drive requiring authentication), you must verify that the operating system will allow access to the file.</li><li>• Data sources with parent-child relationships, should not be accessed together in the same loop, unless the child data source is used for data-driving array elements. Accessing parent-child data sources in the same loop, may corrupt the test.</li><li>• Linking to file names is not supported for <b>HTTP Request</b> and <b>HTTP Receiver</b> activities that use the <b>File</b> type message body.</li><li>• For data sources with child relations: If you change the name the Key column in the child sheet, the Define New/Edit Data Relation dialog box does not reflect the new column name in the drop down lists.</li></ul>
<b>Data driving</b>	<ul style="list-style-type: none"><li>• Data driving for JSON requests or responses, is only supported for Excel.</li><li>• Data driving for an Excel data source is only effective for the first 254 properties of a step. If a step has more than 254 properties, they will not be data-driven.</li><li>• Data driving for an Excel data source is only supported for property values consisting of 255 characters or less. If a property value has more than 255 characters, the data driving mechanism offers to truncate the string.</li><li>• When data driving a test step that has an array with two or more nested levels, the data driving engine only copies the first element of each array to the Excel data tables.</li><li>• Keyword data driving to an XML data source is not supported.</li></ul>

# Chapter 45: Updating Services and Assemblies

**Relevant for: API testing only**

This chapter includes:

- [Updating Services](#) ..... 571
- [How to Update a Web Service](#) ..... 572
- [How to Update a .NET Assembly](#) ..... 574
- [How to Resolve Conflicts in a REST Service Test Step](#) ..... 574
- [How to Update an SAP RFC or IDoc](#) ..... 575
- [Update Port Security Wizard](#) ..... 577
- [Update Step/Activity Wizard](#) ..... 577
- [Resolve REST Method Steps Wizard](#) ..... 578
- [Troubleshooting and Limitations - Updating](#) ..... 580

# Updating Services

## Relevant for: API testing only

You can update services that exist in the Toolbox pane, from their original locations or from alternate locations.

<b>For Web Services</b>	<p>When you update a service, UFT first compares the service and port names. If the port names match, UFT transfers all of the data from the updated service, including new security information. If the new version of the service contains a port that was not present in the original service, it adds it to the Toolbox pane, as an additional port for the service.</p> <p>If the port paths (&lt;service name:port name&gt; combination) do not match, and security was set for on the port level, UFT opens the <b>Update Port Security Wizard</b>.</p> <p>When you update an RPC Web service, as long as the operation names match, the properties are marked as resolved. If the operation names cannot be resolved, UFT opens the wizard. If there is no operation available to map to the original, the wizard suggests that you delete the affected step.</p> <p>For task details, see "<a href="#">How to Update a Web Service</a>" on the next page.</p>
<b>For REST services</b>	<p>If you modify a REST step's properties after incorporating it into your test, it will no longer match the method in the Toolbox. You may want to keep the change, or you may want to remove the conflicts so that your step will match the method defined in the toolbox.</p> <p>You may encounter conflicts in the following areas:</p> <ul style="list-style-type: none"><li>• Adding or removing an input or output property</li><li>• Renaming an input or output property</li><li>• A change in the REST service URL</li><li>• A change in the HTTP request/response body type or schema</li><li>• Internal links between a REST property and its inner HTTP elements (if you are working with an API test created in UFT 11.51 or earlier or Service Test 11.51 or earlier)</li><li>• HTTP methods</li></ul> <p>The Resolve REST Method Steps wizard enables you to view the conflicts and decide what to do with the conflicts in your test step—keep, remove, or map them. The wizard lets you resolve property-related conflicts. Other conflicts, such as discrepancies in the URL values, the HTTP body, and so forth, are resolved automatically.</p>
<b>For SAP IDocs or RFCs</b>	<p>You can update SAP RFCs and IDocs from their original location or from another connection or location.</p> <p>The <b>Update</b> option automatically updates the item from its original location. UFT loads the information for the RFC/IDoc with the same names, from the original connection.</p> <p>The <b>Update from</b> option lets you specify different connection information, or a name of a different RFC/IDoc on the same server.</p> <p>If UFT detects an inconsistency between the original and updated RFC/IDoc names or their properties, it enables the Update Step wizard. This wizard lets you map the original and new items. For details, see "<a href="#">How to Update an SAP RFC or IDoc</a>" on page 575.</p>

For task details, see "[How to Update a Web Service](#)" on the next page, "[How to Resolve Conflicts in a REST Service Test Step](#)" on page 574, or "[How to Update an SAP RFC or IDoc](#)" on page 575.

# How to Update a Web Service

## Relevant for: API testing only

This task describes how to update a WSDL-Based Web Service that was already imported and incorporated into a test.

This task includes the following steps:

- "Prerequisites" below
- "Update the service" below
- "Run the Update Port Security wizard - optional" below
- "Run the Update Step wizard" on the next page

### 1. Prerequisites

If you want to update the WSDL from ALM, make sure you have an open connection to the ALM server.

### 2. Update the service

- To update a service from its original location, select its main node in the Toolbox pane. Choose **Update WSDL** from the context menu.
- To update the service from a different location, or if the **Update WSDL** option is not available:
  - i. Select the service's main node in the Toolbox pane.
  - ii. Choose **Update WSDL from > URL or UDDI** or **Update WSDL from > File or ALM Application Component** from the right-click menu.
  - iii. Navigate to the WSDL and click **OK**.

**IMPORTANT:** The service or service location URL must be accessible when updating the service.

- iv. Accept any warning or informational pop-ups.

**Note:** The **Update WSDL** option may not be available if the user credentials changed or if the service was imported using Service Test or an earlier version of UFT.

### 3. Run the Update Port Security wizard - optional

If UFT detects a change in the port path (<service name:port name> combination) the **Update Port Security** Wizard opens. The wizard only opens if you configured security settings for the original service. Follow the steps of the wizard to resolve all of the Service/Port conflicts.

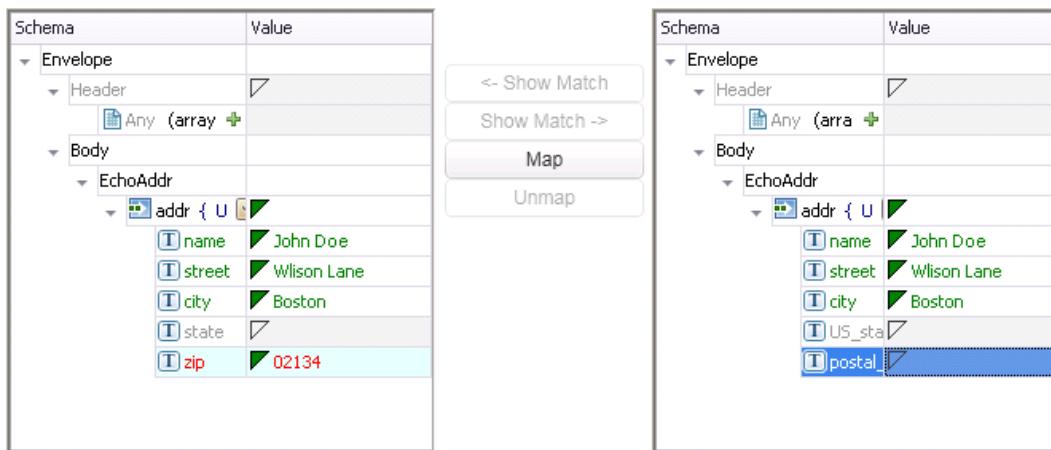
**Note:**

- The wizard imports all of the services defined in the WSDL, even those deleted manually before the update. To remove them from the Toolbox pane, delete them again manually, after the update.

#### 4. Run the Update Step wizard

If a test step became invalid because an operation name changed or properties with values were changed, then the canvas marks the step with a warning marker.

- Click the warning marker to display the message **The step must be resolved. Resolve step.**. Click on the message text. The **Update Step Wizard** window opens.  
Note that the step's properties become read-only, until you resolve them with the help of the wizard.
- In the **Select Operation** page, click in the **New Operation** pane and select the operation that corresponds to the one in the **Original Operation** pane. Click **Next**. Properties for which conflicts were detected are highlighted in red.
- In the page, in the **Original Properties** pane, select a property for which there is a conflict **Update Input Properties**, highlighted in red. In the right pane, **New Properties**, select a property to map.



Click **Map**. The wizard adds the mapping to the list in the bottom pane. Repeat this for all properties that you want to map. Click **Next**.

- In the **Update Output Properties** page, select a property for which there is a conflict, highlighted in red. In the **New Properties** pane select the property to which you want to map. Click **Map**. The wizard adds the mapping to the list of mappings in the bottom pane. Repeat this for all properties that you want to map. Click **Next**.
- Click **Finish** to save your changes and close the wizard.

# How to Update a .NET Assembly

## Relevant for: API testing only

This task describes how to update a .NET assembly with a newer version.

This task includes the following steps:

- "[Select the assembly to update](#)" below
- "[Import a new .NET assembly](#)" below
- "[Handle warnings - optional](#)" below
- "[Modify custom code - optional](#)" below

### 1. [Select the assembly to update](#)

In the Toolbox pane, expand the **.NET Assemblies** node and select the assembly you want to update.

### 2. [Import a new .NET assembly](#)

- a. In the Toolbar, click the Import .NET Assembly button .
- b. In the Import .NET Assembly dialog box, click the **.NET Assembly Browser** tab.
- c. In the .NET Assembly Browser tab, click **Browse** and navigate to the the .dll or .exe file.
- d. Click **Open** to add it to the **Selected References** list.
- e. Click **OK** to begin the update.

### 3. [Handle warnings - optional](#)

If the updated .NET assembly is missing types that are in use by existing activities, you will need to modify the step properties. The canvas displays warning icons adjacent to the steps whose properties use the missing type.

### 4. [Modify custom code - optional](#)

If you wrote custom code in an event handler, you may need to modify the step properties. If the updated .NET assembly is missing types that are used by the custom code, make sure to modify the code to use only the types that are available.

# How to Resolve Conflicts in a REST Service Test Step

## Relevant for: API testing only

This task describes how to update a REST method step that was changed.

This task includes the following steps:

- "Prerequisites" below
- "Modify the step as required" below
- "Open the wizard" below
- "Run the wizard" below

## 1. Prerequisites

Create one or more prototype methods for REST services. Drag them onto the canvas to create REST method steps. For details, see "[How to Create a REST Service](#)" on page 515.

## 2. Modify the step as required

Modify the REST service step as required: add or remove properties, change property values, and so forth. If there are conflicts, the canvas will display warning icons next to the relevant steps.

## 3. Open the wizard

- To resolve conflicts for the selected step, click the warning icon in the top right corner of the REST method step in the canvas. Select **This step should be resolved. Resolve step.** The Resolve REST Method Steps wizard opens.
- To resolve conflicts for all steps created with the prototype, right-click the method in the Toolbox pane and select **Apply Changes to all Steps**.

## 4. Run the wizard

- Select the steps that you want to resolve (only relevant when opening the wizard through the Toolbox pane).  
The left pane, **Before Changes**, shows the step's properties before they were resolved. The right pane, **After Changes**, shows the step's properties after they were resolved.
- Proceed with the wizard, resolving or keeping the detected differences.

For details about the wizard, see the "[Resolve REST Method Steps Wizard](#)" on page 578.

# How to Update an SAP RFC or IDoc

## Relevant for: API testing only

This task describes how to update an SAP RFC or IDoc from its original or from a different location.

This task includes the following steps:

- "[Update the RFC/IDoc from its original location](#)" on the next page
- "[Update the RFC/IDoc from a different location](#)" on the next page
- "[Run the Update Step wizard](#)" on the next page

## Update the RFC/IDoc from its original location

To update an item from its original location, drill down to the actual RFC or IDoc in the Toolbox pane. This assumes that the location and name of the RFC or IDoc did not change. Choose **Update** from the context menu.

## Update the RFC/IDoc from a different location

To update an RFC or IDoc from a different server, or from the same server but a different RFC or IDoc:

1. Select the RFC or IDoc in the Toolbox pane and choose **Update from** from the context menu. The **Update <Item\_Name>** dialog box opens.
2. To change the connection or server information, select **Override connection** and specify the details. Alternatively, you can change the default connection information in the SAP Connections Pane.
3. To select a different RFC or IDoc, search for it and select it in the bottom pane.
4. Click **Update**. Accept any warning or informational pop-ups.

## Run the Update Step wizard

If a test step contains conflicts because an RFC or IDoc name or property changed, then the canvas marks the step with a warning marker.

1. Click the warning marker to display the message **The step must be resolved. Resolve step.** Click on the message text. The **Update Step Wizard** window opens.  
Note that the step's properties become read-only, until you resolve them with the help of the wizard.
2. In the wizard's **Select Activity** page, click in the **New item** area and select the operation that corresponds to the one in the **Original item** pane. Click **Next**. Properties for which conflicts were detected are highlighted in red.
3. If there are conflicted input properties, the **Update Input Properties** page opens. In the **Original Properties** pane, select a property for which there is a conflict, highlighted in red. In the right pane, **New Properties**, select a property to map. Click **Map**. The wizard adds the mapping to the list in the bottom pane. Repeat this for all properties that you want to map. Click **Next**.
4. If there are conflicted output properties, the **Update Output Properties** page opens. Select a property for which there is a conflict, highlighted in red. In the **New Properties** pane select the property to which you want to map. Click **Map**. The wizard adds the mapping to the list of mappings in the bottom pane. Repeat this for all properties that you want to map. Click **Next**.
5. Click **Finish** to save your changes and close the wizard.

## Update Port Security Wizard

### Relevant for: API testing only

This wizard enables you to update ports that became invalid as a result of an update action. The steps became invalid because the port path (<service name>:port name combination) changed. The wizard enables you to map old port names to new ones.

<b>To access</b>	<ol style="list-style-type: none"><li>1. Do one of the following:<ul style="list-style-type: none"><li>• Ensure that an API test or component is in focus in the document pane.</li><li>• In the Solution Explorer, select an API test or component.</li></ul></li><li>2. Import a WSDL using the <b>Import WSDL</b> toolbar button.</li><li>3. In the Toolbox pane, click on one of the service's ports and select <b>Security Settings</b> from the right-click menu.</li><li>4. In the Security Settings for Port dialog box, configure the port's security.</li><li>5. Select the service's parent node in the Toolbox pane and select <b>Update</b> (or <b>Update from</b>) from the right-click menu.</li></ol>
<b>Relevant tasks</b>	<a href="#">"How to Update a Web Service" on page 572</a>
<b>See also</b>	<a href="#">"Updating Services" on page 571</a>

This wizard contains:

- Map Services and Ports Page
- Finish Page

## Update Step/Activity Wizard

### Relevant for: API testing only

This wizard enables you to update test steps that became invalid as a result of an **Update from** action. The steps became invalid because the step/activity was removed or changed, or property names were modified. The wizard enables you to map old operations/activities and properties to new ones.

<b>To access</b>	<ol style="list-style-type: none"><li>1. Do one of the following:<ul style="list-style-type: none"><li>• Ensure that an API test or component is in focus in the document pane.</li><li>• In the Solution Explorer, select an API test or component.</li></ul></li></ol>
------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	<ol style="list-style-type: none"> <li>2. Use the <b>Tools</b> menu to import a Web service or an SAP RFC or IDoc.</li> <li>3. Drag an operation/activity from the Toolbox pane onto the canvas and set input values.</li> <li>4. Select the parent node and select <b>Update</b> or <b>Update from</b> in the shortcut menu. Accept any warning popup messages.</li> <li>5. Click on the alert adjacent to the step .</li> <li>6. Click on the text <b>The step must be resolved. Resolve step.</b></li> </ol> <p><b>Note:</b> The wizard will open only when there is a change in property names that were assigned values.</p>
<b>Relevant tasks</b>	<ul style="list-style-type: none"> <li>• <a href="#">"How to Update a Web Service" on page 572</a></li> <li>• <a href="#">"How to Update an SAP RFC or IDoc" on page 575</a></li> </ul>
Important Information	The service or service location URL must be accessible when updating the service.
<b>See also</b>	<a href="#">"Updating Services" on page 571</a>

This wizard contains:

- Select Operation/Activity Page
- Update Input Properties Page
- Update Output Properties Page
- Finish Page

## Resolve REST Method Steps Wizard

### Relevant for: API testing only

This wizard enables you to resolve differences between REST service steps and the prototype method upon which they were based. The wizard detects changes such as added, removed, or renamed properties and helps you resolve them.

<b>To access</b>	<p>Do one of the following:</p> <ul style="list-style-type: none"> <li>• Ensure that an API test or component is in focus in the document pane.</li> <li>• In the Solution Explorer, select an API test or component.</li> </ul> <p>As a prerequisite:</p> <ol style="list-style-type: none"> <li>1. Create a prototype for a REST service method. For details, see <a href="#">"How to Create a REST Service" on page 515</a>.</li> <li>2. Drag one or more REST service methods from the Toolbox pane to the canvas.</li> <li>3. Select a REST method in the Toolbox pane and choose <b>Edit Service</b> from the right-click menu. In the Edit REST Service dialog box, modify the method's properties and/or values.</li> </ol> <p>To start the wizard:</p> <ul style="list-style-type: none"> <li>• <b>For the selected step only:</b> Click on the alert in the bottom right corner of the REST method step  and</li> </ul>
------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	<p>click the text <b>The step should be resolved. Resolve step.</b></p> <ul style="list-style-type: none"><li>• <b>For all steps using the prototype method:</b> In the Toolbox pane, right-click the method and select <b>Apply Changes to all Steps</b>.</li></ul>
<b>Relevant tasks</b>	<a href="#">"How to Resolve Conflicts in a REST Service Test Step" on page 574</a>

This wizard contains:

- Select Steps Page
- Resolve Conflicts Page
- Finish Page

# Troubleshooting and Limitations - Updating

## **Relevant for: API testing only**

When comparing a REST step to its prototype - a difference in the request/response body contents is not considered a conflict. Therefore, when resolving a REST step whose contents do not match the body contents of the prototype, the body contents will not be affected.

# Chapter 46: Web Service Security

**Relevant for: API testing only**

This chapter includes:

• Setting Security Overview .....	582
• Security Scenarios Overview .....	582
• Web Service Scenario Overview .....	583
• WCF Service Scenarios Overview .....	584
• How to Set Security for a Standard Web Service .....	586
• How to Customize Security for WCF Type Web Services .....	590
• How to Set up Advanced Standards Testing .....	594
• Troubleshooting and Limitations - Web Service Security .....	596

# Setting Security Overview

## Relevant for: API testing only

When building Web Service applications, there is a challenge in building scalable applications that are secure. You can secure Web Services by having the message sent over a secure transport, such as Secure Sockets Layer (SSL), or by applying security at the message level, also known as **WS-Security**.

For testing a secured service, answering the following questions will help you define your security scenario.

- Is there transport security, such as SSL? What is the HTTPS URL?
- Is basic authentication required?
- Is mutual authentication required?
- What type of security is required in the SOAP header?

UFT lets you set the security for a service on two levels—**port** or **operation**. If you define a security for a port, all of its methods use these settings, by default. When working in the canvas, you can override the default port security for a given test step and customize the security for a particular operation.

**Note:** You set the basic authentication information for your Web Service calls (including REST Service methods, HTTP Request, and SOAP Request steps) in the General tab of the Properties pane (for HTTP and REST method steps), the HTTP tab of the Security Settings dialog box, or Security tab in the Properties pane (for Web Service and SOAP request steps).

# Security Scenarios Overview

## Relevant for: API testing only

UFT provides several built-in scenarios for configuring security in Web Service calls.

A security scenario represents a typical security implementation for a Web Service. It contains information such as authentication, encoding, proxy, certificates, and so forth.

You can select one of the following types of Web Service security scenarios:

<b>Default Web Service Scenario</b>	A default <b>Web Service</b> scenario can be used for most Web services. It enables you to configure both transport and message-level security. UFT support for message-level security lets you manually configure the security elements such as tokens, message signatures, and encryption. For details, see " <a href="#">Web Service Scenario Overview</a> " on the <a href="#">next page</a> .
	<p>You use the <b>default Web Service scenario</b> for:</p> <ul style="list-style-type: none"><li>• Simple Web Services where no advanced standards are required.</li><li>• Web services using HTTP transport level security.</li><li>• Web services using message level security (WS-Security) for SOAP 1.1</li></ul>

<b>WCF-type scenario</b>	<p>WCF scenarios enables you to configure security for HTTP or custom bindings and work with advanced specifications, such as <b>WS-SecureConversation</b>.</p> <p>You use a <b>WCF scenario</b> for:</p> <ul style="list-style-type: none"> <li>• WCF Services that utilize advanced security and WS-Specifications.</li> <li>• Web services using message level security (WS-Security) for SOAP 1.2</li> </ul>
--------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Such services can be written in various platforms such as WCF (Windows Communication Foundation), Metro (WSIT), and Axis2. UFT also supports proprietary standards and transports.

## Learn more!

- "[Web Service Scenario Overview](#)" below
- "[WCF Service Scenarios Overview](#)" on the next page

## Web Service Scenario Overview

### Relevant for: API testing only

The default Web Service scenario is based on the WS-Security specification. This scenario lets you place security credentials in the actual SOAP message.

When a SOAP message sender sends a request, the security credentials, known as **tokens**, are placed in the SOAP message. When the Web server receives the SOAP request, it does not need to send additional requests to verify the integrity of the sender. The server verifies that the credentials are authentic before letting the Web Service execute the application. By not having to go back to the source of the credentials, the application performance improves significantly.

To further secure Web Services, it is common to use digital signatures or encryption for the SOAP messages. Digitally signing a SOAP message verifies that the message has not been altered during transmission. Encrypting a SOAP message helps secure a Web Service by making it difficult for anyone other than the intended recipient to read the contents of the message.

Use the following types of security:

<b>Transport Level Security</b>	The transport level security includes the authentication and proxy server information. You can also specify Keep Alive preferences and connection timeout.
<b>Message Level Security</b>	<p>The <b>WS-Security</b> tab lets you set the message level security through tokens, signatures and encryption. To support WS-Security, UFT enables you to create security tokens for your script. You can create multiple tokens and set their properties. After creating a token, you use it to sign or encrypt a SOAP message.</p> <p>The Web Services security mechanism associates security tokens with messages. This mechanism supports several security token formats to accommodate a variety of authentication requirements. For example, a client might need to provide a proof of identity or a security certificate.</p> <p>The available tokens are: <b>UserName</b>, <b>X509 Certificate</b>, <b>Kerberos</b>, <b>Kerberos2</b>, and <b>SAML</b>:</p> <ul style="list-style-type: none"> <li>• <b>UserName</b>: The <b>User Name</b> token contains user identification information for the purpose of authentication:</li> </ul>

	<p><b>User Name and Password.</b> You can also specify Password Options, indicating how to send the password to the server for authentication: <b>Text</b>, <b>None</b>, or <b>Hash</b>, and indicate whether to include a timestamp.</p> <ul style="list-style-type: none"><li>• <b>X509 Certificate:</b> This token is based on an X.509 certificate. To obtain a certificate, you can either purchase it from a certificate authority, such as VeriSign, Inc. or set up your own certificate service to issue a certificate. Most Windows servers support the public key infrastructure (PKI), which enables you to create certificates. You can then have it signed by a certificate authority or use an unsigned certificate.</li><li>• <b>Kerberos/Kerberos 2:</b> The Kerberos protocol is used to mutually authenticate users and services on an open and unsecured network. Using shared secret keys, it encrypts and signs user credentials. A third party, known as a KDC (Kerberos Key Distribution Center), authenticates the credentials. After authentication, the user may request a service ticket to access one or more services on the network. The ticket includes the encrypted, authenticated identity of the user. The tickets are obtained using the current user's credentials.</li></ul> <p>The primary difference between the Kerberos and Kerberos2 tokens, is that Kerberos2 uses the Security Support Provider Interface (SSPI), so it does not require elevated privileges to impersonate the client's identity. In addition, the Kerberos2 security token can be used to secure SOAP messages sent to a Web Service running in a Web farm.</p> <ul style="list-style-type: none"><li>• <b>SAML Token:</b> SAML is an XML standard for exchanging security-related information, called assertions, between business partners over the Internet. The assertions can include attribute statements, authentication, decision statements, and authorization decision statements.</li></ul> <p>SAML uses brokered authentication with a security token issued by STS (Security Token Service). The STS is trusted by the client and the Web Service to provide interoperable security tokens. SAML tokens are important for Web Service security because they provide cross-platform interoperability and a means of exchanging information between clients and services that do not reside within a single security domain.</p>
--	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

When you add a security token to a SOAP message, it is added to the SOAP message in the form of an XML element in the WS-Security SOAP header. The message, however, is exposed and therefore requires additional security. This is especially true when the credentials, including the password, are sent in plain text as it is with role-based security.

The two methods used to secure the data are message signatures and message encryption:

- **Message Signatures.** Message Signatures are used by the recipients to verify that messages were not altered since their signing. The signature is in the form of XML within the SOAP message. The recipient checks the signature to make sure it is valid.
- **Message Encryption.** Although the XML message signature offers a mechanism for verifying that the message has not been altered since it was signed, it does not encrypt the SOAP message—the message is still plain text in XML format. To secure the message in order that it should not be exposed, you encrypt it, making it difficult for an intruder to view and obtain a user's password.

## WCF Service Scenarios Overview

In addition to the default Web service scenario, you can select from one of the following WCF-type service scenarios:

<b>WCF Service (CustomBinding) Scenario</b>	The <b>WCF Service (CustomBinding)</b> scenario enables the highest degree of customization. Since it is based upon the WCF CustomBinding standard, it enables you to test most WCF services, along with services on other platforms such as Java-based services that use the WS - <spec_name> specifications.
---------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	<p>Use the WCF Service (CustomBinding) scenario to configure a scenario that does not comply with any of the predefined security scenarios. You can customize the transport and encoding settings:</p> <ul style="list-style-type: none"> <li>• <b>Transport.</b>HTTP, HTTPS, TCP, or NamedPipe</li> <li>• <b>Encoding.</b>Text, MTOM, or WCF Binary</li> </ul> <p>You can also provide additional security information:</p> <ul style="list-style-type: none"> <li>• <b>Authentication mode.</b> The type of authentication, such as None, AnonymousForCertificate, and so forth. The options are available from the drop down list.</li> <li>• <b>Bootstrap policy.</b> For the SecureConversation authentication mode, you can specify a bootstrap policy.</li> <li>• <b>Net Security.</b> The network security for TCP and NamedPipe type transports. Typical values are <b>None</b>, <b>Windows stream security</b>, or <b>SSL stream security</b> available from the field's drop down list. For services with HTTP transport, you should set the value to <b>None</b>. To enable SSL for HTTP, select <b>HTTPS transport</b>.</li> </ul> <p>For task details, see "<a href="#">How to Customize Security for WCF Type Web Services</a>" on page 590.</p> <p><b>Note:</b> For WSE3 security configurations, use the WCF Service(CustomBinding) Scenario. For details, see "<a href="#">How to Customize Security for WCF Type Web Services</a>" on page 590.</p>
<b>WCF Service (Federation) Scenario</b>	<p>In the <b>WCF Service (Federation)</b> scenario, the client authenticates against the STS (Security Token Service) to obtain a token. The client uses the token to authenticate against the application server. Therefore, two bindings are needed, one against the STS and another against the application server. You define the bindings in two stages:</p> <ul style="list-style-type: none"> <li>• Provide security details for the application server's security scenario in the following areas: <ul style="list-style-type: none"> <li>• <b>Server.</b> The transport and encoding methods.</li> <li>• <b>Security.</b> The authentication mode, such as IssuedToken, SecureConversation, and so forth.</li> <li>• <b>Identity.</b> Information about the server certificate and expected DNS.</li> <li>• <b>STS.</b> Settings related to the STS, such as the endpoint address and binding.</li> </ul> </li> <li>• Define an STS binding in the <b>Referenced binding</b> field.</li> </ul> <p>For task details, see "<a href="#">How to Customize Security for WCF Type Web Services</a>" on page 590.</p>
<b>WCF Services (WSHttpBinding) Scenario</b>	<p>In the <b>WCFService (WSHttpBinding)</b> scenario, you can select from several types of authentication: None, Windows, Certificate, or Username (message protection).</p> <p><b>No Authentication (Anonymous)</b></p> <p>In this scenario, the client uses the server's certificate to encrypt a message; there is no client authentication. Use this scenario to test Web Services where the:</p> <ul style="list-style-type: none"> <li>• Client uses the server's X.509 certificate for encryption.</li> <li>• Client is not authenticated.</li> <li>• Communication may utilize advanced standards such as secure conversation or MTOM.</li> </ul> <p><b>Windows Authentication</b></p>

	<p>This scenario uses Windows Authentication. If you are testing a WCF service that has not been customized and uses the default configuration, use this type of scenario.</p> <p>Use this scenario to test Web Services where the:</p> <ul style="list-style-type: none"><li>• Client and server use Windows authentication.</li><li>• Security is based on Kerberos or SPNEGO negotiations.</li><li>• Communication may utilize advanced standards such as secure conversation or MTOM.</li></ul> <p><b>Certificate Authentication</b></p> <p>In this <b>WCF WSHttpBinding</b> scenario, the client uses the server's X.509 certificate to encrypt the message and its own certificate for a signature.</p> <p>Use this scenario to test Web Services where the:</p> <ul style="list-style-type: none"><li>• Client uses the server's X.509 certificate for encryption.</li><li>• Client uses its own X.509 certificate for signatures.</li><li>• Communication may utilize advanced standards such as secure conversation or MTOM.</li></ul> <p><b>Username Authentication (Message Protection)</b></p> <p>In the <b>WCF WSHttpBinding</b> scenario, the client uses the server's X.509 certificate to encrypt the message, and sends a user name and password to authenticate itself.</p> <p>Use this scenario to test Web Services where the:</p> <ul style="list-style-type: none"><li>• Client uses the server's X.509 certificate for encryption.</li><li>• Client is authenticated with a username and password.</li><li>• Communication may utilize advanced standards such as secure conversation or MTOM.</li></ul>
--	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## How to Set Security for a Standard Web Service

### Relevant for: API testing only

This task describes how to configure security settings for a standard Web Service. This mode lets you define the HTTP transport information and security elements such as tokens.

This task includes the following steps:

- "[Create a Web Service scenario](#)" on the next page
- "[Configure the HTTP settings](#)" on the next page
- "[Add message level security with a Username Token](#)" on the next page
- "[Add message level security by signing with an X.509 Certificate](#)" on the next page
- "[Encrypt a Web service message using a Certificate](#)" on page 588
- "[Send a username token and encrypt the token with an X.509 Certificate](#)" on page 589
- "[Sign and encrypt a Web service message](#)" on page 589
- "[Configure the WS-Addressing \(optional\)](#)" on page 590

## 1. Create a Web Service scenario

- a. Open the Security Settings dialog in one of the following ways:
  - To set security on a port level, right-click a Web Service port in the Toolbox and choose **Security Settings**.
  - To set security for a specific Web Service step already on the canvas, select the step and open the **Security Settings** tab in the Properties pane. Clear the **use the port's security settings** option.
  - For a SOAP Request step, click the **Security Settings** tab in the Properties pane.
- b. In the Security Settings dialog box, select **Web Service** from the **Service Details** dropdown list (default).

## 2. Configure the HTTP settings

In the main window of the Security Settings dialog box, select the **HTTP** tab, and set the transport and proxy information.

## 3. Add message level security with a Username Token

To send a message level username/password token (a UserName token):

- a. In the Security Settings dialog box, from the Service Details dropdown list, select the **Web Service** scenario.
- b. In the main part of the Security Settings dialog box, click the **WS-Security** tab.
- c. In the WS-Security tab, click the **Add Token**  button and add a **Username** token.
- d. In the lower pane, customize the token details, such as username and password.

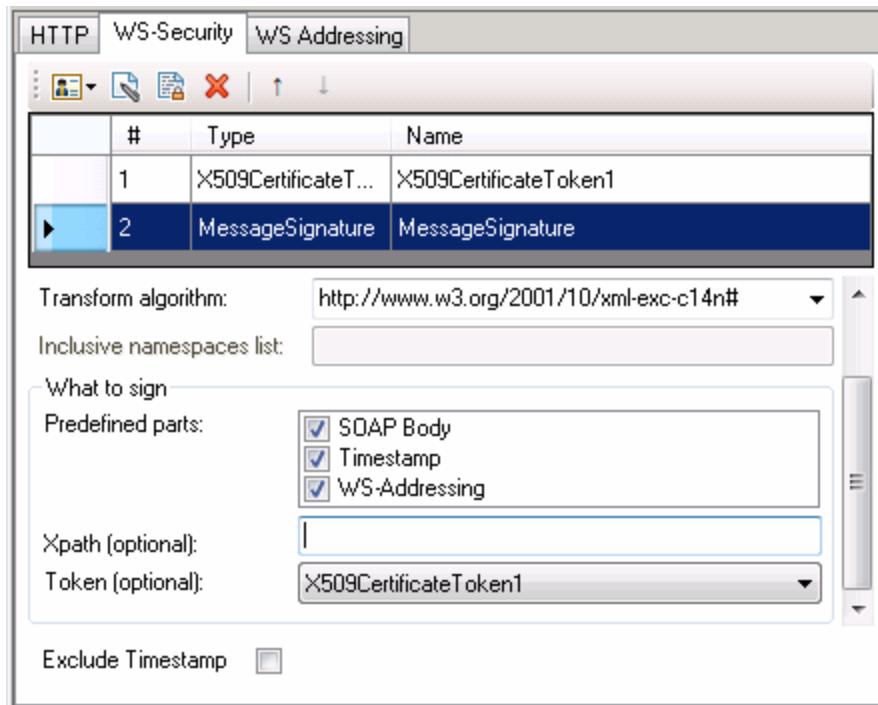
## 4. Add message level security by signing with an X.509 Certificate

- a. In the Security Settings dialog box, select the **Web Service** scenario from the **Service Details** dropdown list.
- b. In the main part of the Security Settings dialog box, click the **WS-Security** tab.
- c. In the WS-Security tab, click the **Add Token** button  and select **X509 Certificate** from the dropdown list
- d. In the lower pane, enter the token name.
- e. Click the **Browse** button to navigate to your certificate file. The certificate must be installed in the Windows certificate store.
- f. Select a **Reference type**. Since this token is used for a signature, the most common type is **BinarySecurityToken**.
- g. Click the **Add Message Signature** button .
- h. In the **Signing Token** dropdown list, select the token you entered in the previous steps.

- i. To sign a specific element with the certificate, scroll down to the **XPath** field and provide an XPath expression.

You cannot use an XPath expression to sign a timestamp or token that is under the security element of a SOAP request.

- To sign a **SOAP Body**, **Timestamp**, or **WS-Addressing**, select the check box in the **Predefined parts** area.
- To sign tokens within the security element, select a token in the **Token (optional)** field, in the **What to sign** area.



**Note:** The certificate needs to be installed in the Windows certificate store. In the example above, you need to set the actual store name, store location, and subject name of your certificate.

## 5. Encrypt a Web service message using a Certificate

To encrypt a message using the service's X.509 certificate:

- a. In the Security Settings dialog box, select the **Web Service** scenario from the **Service Details** dropdown list,
- b. In the main part of the Security Settings dialog box, select the **WS-Security** tab.
- c. In the WS-Security tab, click the **Add Token** button  and select **X509 Certificate** token from the **Security Token** drop down list.
- d. Enter token name.
- e. Since this token is used for encryption, use Reference as the **Reference type**.

- f. Click the **Add Message Encryption** button  . In the drop down list, select the token you created in the previous steps.
- g. Scroll down to the **XPath** field. Enter an XPath expression to the elements to encrypt, for example: // \*[local-name(.)='Body'].

## 6. Send a username token and encrypt the token with an X.509 Certificate

The following section describes how to send a **Username** token to the service and encrypt it with the server's **X.509** certificate:

- a. In the Security Settings dialog box, select the **Web Service** scenario from the **Service Details** dropdown list.
- b. In the main part of the Security Settings dialog box, select the **WS-Security** tab.
- c. In the WS-Security tab, click the **Add Token** button  and select **Username Token** from the **Security Token** drop down list.
- d. In the lower pane, Provide the token details for the Username token.
- e. Click the **Add Token** button  again and select **X509 Certificate Token** from the **Security Token**  drop down list.
- f. In the lower pane, enter the token details to reference the server's public certificate. Since this token is used for encryption, use **Reference** as the **Reference type**.
- g. Click the **Add Message Encryption** button  . In the drop down list, select the X.509 token you created in the previous steps.
- h. To encrypt a specific message, scroll down to the **XPath** field. Enter an XPath expression, for example: // \*[local-name(.)='Body'].

## 7. Sign and encrypt a Web service message

This example shows how to sign a message using a private key and then encrypt it using the service's public key.

- a. In the Security Settings dialog box, select the **Web Service** scenario from the **Service Details** list.
- b. In the main part of the Security Settings dialog box, select the **WS-Security** tab.
- c. In the WS-Security tab, click the **Add Token** button  and select **X509 Certificate Token** from the **Security Token** drop down list.
- d. Enter the token details to reference your private certificate. Select **BinarySecurityToken** as a **Reference type**, since this token is used for a signature.
- e. Click the **Add Token** button  again and select **X509 Certificate Token** from the **Security Token** drop down list to add another X.509 token.
- f. Enter the token details to reference your private certificate. Select **Reference** as a **Reference type**, since this token is used for a encryption.

- g. Click the **Add Message Signature** button  . In the drop down list, select the first X.509 token you created.
  - h. Click the **Add Message Encryption** button  . In the drop down list, select the second token you created.
8. **Configure the WS-Addressing (optional)**
- a. In the main pane of the Security settings dialog box, click the **WS-Addressing** tab.
  - b. In the WS-Addressing tab, select the relevant version or **None** if WS-Addressing is not used.
  - c. In the **Reply to** field, provide an alternate destination.

## How to Customize Security for WCF Type Web Services

### Relevant for: API testing only

This section describes how to customize the security settings for Web services using WCF.

This section describes:

- "Create a WCF scenario" below
- "Configure the settings for a Web Service using WSHTTPBinding" on the next page
- "Configure the settings for a Web Service using CustomBinding" on the next page
- "Configure the settings for a WCF Federation Web service" on the next page
- "Configure the settings for a WCF service using netTcp or namedPipe transport" on page 592
- "Configure the settings for a Web service using WSE3 security configuration with a server certificate" on page 592
- "Configure the settings for WCF service using mutual certificate authentication" on page 593
- "Configure the settings for a WCF scenario using binding with TCP transport to require an X.509 client certificate" on page 593

### Create a WCF scenario

1. Open the Security Settings dialog box in one of the following ways:
  - For port level security, right-click a service's port in the Toolbox pane and select **Security Settings**.
  - For step level security, open the **Security Settings** tab in the Properties pane. Clear the **Use the port's security settings** option.
2. In the Security Settings dialog box, select the type of **WCF Service** from the **Service Details** dropdown list.

## Configure the settings for a Web Service using WSHTTPBinding

1. At the top of the Security Settings dialog box, in the dropdown list, select **WCF Service (WSHttpBinding)**.
  2. In the Client authentication type dropdown list, choose a client credential type to use in your binding—Windows, Certificate, or Username. This value corresponds to the **MessageClientCredentialType** property of the WCF's WSHttpBinding parameter.  
**Windows** authentication is the most common value for a WCF services. If you are using the WCF default settings for your service, use this option.
  3. Define the security settings for your authentication type. The available options differ per authentication type.
- Note:** For some scenarios you should indicate whether to use the WCF proprietary negotiation mechanism to get the service credentials.
4. Click **Advanced** to control the usage of a secure session.

## Configure the settings for a Web Service using CustomBinding

1. In the Security Settings dialog box, in the dropdown list, select the **WCFSERVICE (Custom Binding)** scenario.
2. In the main pane of the Security Settings dialog box, set the Web service security options, including:
  - **Transport** type
  - **Encoding**
  - **Authentication mode** for the Web service
  - **Net security** type
  - The **identities** for the custom bindings and authentication certificate
  - The **client user** information for the "user" who would access the Web service

## Configure the settings for a WCF Federation Web service

1. In the Security Settings dialog box, in the dropdown list, select the **WCFSERVICE Federation** scenario.
2. Provide the service and security transport details, including:
  - **Transport** type
  - **Encoding**

- **Authentication mode** for the Web service
- **Bootstrap policy** for the Web service
- The **identities** for the custom bindings and authentication certificate
- **STS** (Security Token Service) settings

**Note:** You must define the communication properties for both the STS and the application server

### Configure the settings for a WCF service using netTcp or namedPipe transport

1. In the Security Settings dialog box, from the dropdown list, select the **WCFService (Custom Binding)** scenario.
2. Set the **Transport** option to **TCP** or **NamedPipe**.
3. Set the other security settings as described in "[Configure the settings for a Web Service using CustomBinding](#)" on the previous page.

### Configure the settings for a Web service using WSE3 security configuration with a server certificate

1. Create a new test and import a WSDL containing the WSE3 service.
2. Add a method from the Web service to the canvas.
3. Select the **Security Settings** tab  in the Properties pane or right-click the Web service node in the Toolbox and select **Security Settings**.
4. In the Security Settings dialog box, from the dropdown list, select the **WCFService (Custom Binding)** scenario.
5. In the main pane of the Security Settings dialog box, set the **Transport** option to **HTTP**, and the Encoding to **Text**.
6. In the Identities section, enter a username and password.
7. Click the **Browse** button adjacent to the Server Certificate field and specify the **Store Location**, **Store Name** and **Search text** (optional). Click **Find**, select the certificate, and click **Select**.
8. Provide the **Expected DNS**.
9. Click the **Advanced** button and configure the following settings in the Advanced Settings dialog box:
  - a. In the **Encoding** tab: Set the **WS-Addressing** version appropriately
  - b. In the **Security** tab, set the following options:
    - **Enable secure session:** Enabled
    - **Negotiate service credentials:** Enabled

- **Protection level:** Encrypt and Sign
  - **Message protection order:** Sign Before Encrypt
  - **Message security version:**  
WSSecurity11WSTrustFebruary2005WSSecureConversationFebruary2005 (first entry)
  - **Require Derived keys:** Enabled
- For all other fields, use the default settings.

## Configure the settings for WCF service using mutual certificate authentication

The following procedure describes how to set up a security scenario for mutual certificates and how to comply with a WSE3 security configuration.

1. In the Security Settings dialog box, from the dropdown list, select the **WCFSERVICE (CustomBinding)** scenario.
2. Set the **Transport** option to HTTP, and the **Encoding** to Text.
3. Set the authentication mode to MutualCertificate.
4. In the **Identities** section, select the server and client certificates.
5. Provide the **Expected DNS**.
6. Click the **Advanced** button and configure the following settings in the Advanced Settings dialog box:
  - a. **Encoding tab—WS-Addressing:** WSA 04/08 (for a WSE3 security configuration).
  - b. **Security tab—Require Derived keys:** Disabled

For all other fields, use the default settings.

## Configure the settings for a WCF scenario using binding with TCP transport to require an X.509 client certificate

The following procedure describes how to configure a WCF custom scenario to require an X.509 client certificate in **nettcp**.

1. In the Security Settings dialog box, from the dropdown list, select the **WCFSERVICE (Custom Binding)** scenario.
2. Set the **Transport** to **TCP** and the **Net Security** to **SSL stream security**.
3. In the Properties pane, open the **Events** tab .
4. In the events list, select the **BeforeApplyProtocolSettings** event. Click in the **Handler** column and select **Create a default handler** from the drop-down.
5. In the **TestUserCode.cs** file, locate the **TODO** section of the code and add the following definitions.

```
var wcf = (HP.ST.Ext.CommunicationChannels.Models.WcfChannelBinding)args[1];
```

```
var ssl =
 (HP.ST.Ext.CommunicationChannels.Models.WcfSslStreamSecurityChannel)wcf.
 Protocols.Channels[1];
ssl.RequireClientCertificate = true;
```

For all other fields, use the default settings.

## How to Set up Advanced Standards Testing

### **Relevant for: API testing only**

This section provides guidelines for using UFT in advanced standards testing.

This section includes:

- "[Test a Web Service using MTOM](#)" below
- "[Change the WS-Addressing version of a service](#)" below
- "[Enable support for a service or activity that uses 256-bit SSL encoding](#)" below

### **Test a Web Service using MTOM**

1. Select the **WCFService (Custom Binding)** scenario from the **Service Details** list.
2. Configure the **Encoding** to MTOM.

If your service requires advanced settings, click the **Advanced** button.

### **Change the WS-Addressing version of a service**

1. Select the **Web Service** scenario from the **Service Details** list.

**Note:** If your service uses WCF, use the appropriate scenario and configure the addressing version from the Advanced Settings dialog box's **Encoding** tab.

2. Click the **WS-Addressing** tab and select a version.

### **Enable support for a service or activity that uses 256-bit SSL encoding**

Change the SSL cipher order so that AES256 precedes AES128 in the cipher list.

**Tip:** Check with an IT professional before performing the following actions.

To change the cipher order:

1. Type `gpedit.msc` at a command prompt to open your group policy editor.
2. Choose **Computer Configuration > Administrative Templates > Network > SSL Configuration Settings**.
3. Open the only item—**SSL Cipher Suite Order**.

4. Select **Enabled**.
5. The first item in the list is TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA  
The second item is TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA
6. Change the first 128 to 256. Then move the cursor forward and change the 256 to 128.
7. Move the cursor through the list and change the cipher priorities as in the above step.
8. Close the group policy editor and reboot.

# Troubleshooting and Limitations - Web Service Security

## **Relevant for: API testing only**

This section describes troubleshooting and limitations for working with Web services security.

- Authentication and proxy security are not supported for Web Services imported from a UDDI.
- For Web Service configured with WCF settings: Configuring different security settings for operations residing on the same port is not supported.
- For Web Service configured with WCF settings, some of the user event handlers (such as the **AfterProcessRequestSecurity**, **BeforeProcessResponseSecurity**, **OnSendRequest**, and **OnReceiveResponse** events) will not be invoked.
- When testing Web Services that require message-level security, the Web Service security scenario only supports SOAP version 1.1. For SOAP 1.2 use a WCF type scenario.
- When using a SAML security token for Web services security, user-provided content may contain creation and expiration timestamps. To extend the life of the test, we recommend that you hard-code an expiration date in the distant future. In this is not possible, change the timestamp by implementing the **OnBeforeApplyProtocolSettings** event.
- When using a SAML security token for Web services security, if you edit the values in Grid mode, they may not be updated in UFT.

**Workaround:** To update the values, switch to **Text** mode and save the test.

- Web Service steps are not supported when using a SAML token with a certificate from the file system.

**Workaround:** Install the certificate to the Windows store and select the certificate from the store.

- When working with Federation type scenarios that use STS (Security Token Service), you cannot change the SOAP version.
- If you are using SOAP version 1.2:
  - You can choose only UserName or X509 tokens when configuring the message level security.
  - When configuring the canonicalization algorithm and the transform algorithm for the message signature, you cannot use the following format: <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform>.

# Chapter 47: Asynchronous Service Calls

**Relevant for: API testing only**

This chapter includes:

- [Asynchronous Services](#) ..... 598
- [How to Create an API Test for an Asynchronous Web Service](#) ..... 599
- [Troubleshooting and Limitations - Asynchronous Testing](#) ..... 602

# Asynchronous Services

## Relevant for: API testing only

You can use UFT to emulate asynchronous services, such as Web Services, REST services, HTTP requests, JMS/MQ-based services, and so forth.

In synchronous messaging, the replay engine blocks step execution until the server responds. The client sends a request and receives a response immediately, using the same connection. During the waiting time, the replay engine is blocked and does not perform any other activity. If the timeout was reached without a response from the server, the client returns an error.

In asynchronous mode, the replay engine executes the step without waiting for server's response from previous requests.

UFT provides a solution for the following asynchronous patterns:

<b>WS-Addressing</b>	<p><b>WS-Addressing</b> is a specification that enables Web services to communicate addressing information. You can instruct the server to respond to any location, and not necessarily to the machine that issued the request. To do this, you use the WS-Addressing <b>replyTo</b> attribute.</p> <p>In this implementation, UFT pauses the test and uses a listener mechanism to verify that the response arrived at the specified address. After the listener acknowledges that the server responded to the address or if it reaches the timeout, the test resumes. Upon the completion of the test, you can validate the response with the standard API testing checkpoints.</p>
<b>HTTP Receiver</b>	<p>In the <b>HTTP Receiver</b> pattern, the <b>server</b> sends an HTTP request to the client, reversing the typical roles of the client and server.</p> <p>This is useful, for example, if you want to test a service which publishes information over HTTP to a client. You define a receiver, which waits for a request from the server, sent over HTTP.</p> <p>After a trigger, the receiver captures the request. The trigger can be an HTTP client request, a call to a Web service, an email, or any other event that will trigger the server. If there are inner steps, the receiver waits for them to finish and only then is the receiver activity considered complete.</p> <p>Using the API testing interface, you can insert the necessary logic and validate the checkpoints in the captured request.</p> <p>The response from the receiver should wait for the inner steps to complete and link to them.</p> <p>The completion event name fired for the receiver should only be fired AFTER the inner steps are done.</p> <p>For details about the properties, see the "<a href="#">HTTP Receiver Activity</a>" on page 468.</p>
<b>Web Service Publish Subscribe</b>	<p>In the <b>Web Service Publish Subscribe</b> pattern, the <b>server</b> sends an HTTP request to the client, reversing the typical roles of the client and server. It is similar to the <b>HTTP Receiver</b>, except that the request is sent to the client through a Web Service call instead of exclusively via HTTP.</p> <p>Using UFT, you test the publishing of messages to the client. You set up a receiver, which waits for a server request, sent from the server as a Web service call.</p> <p>After a trigger, the receiver captures the request. The trigger can be an HTTP client request, a call to a Web service, an email, or any other event that will trigger the server.</p> <p>Using the API testing interface, you can validate the response with standard API testing checkpoints.</p>

<b>Web Service Solicit Response</b>	<p>The <b>Web Service Solicit Response</b> pattern is a variation of the <b>Web Service Publish Subscribe</b> pattern. It enables you test a service which publishes information through a Web Service to a client.</p> <p>In this pattern, however, the client is expected to send a response to the server request. The response can be a simple acknowledgment or a full SOAP message.</p> <p>You set up a receiver activity, which waits for a server request. This server request is sent from the server as a Web service call. The receiver then sends a client response back to the server.</p> <p>After the trigger, the receiver captures the request. The trigger can be an HTTP client request, a call to a Web service, an email, or any other event that will trigger the server.</p> <p>Using the API testing interface, you can validate the response with standard API testing checkpoints.</p>
<b>Dual WSDL Files</b>	<p>The Dual WSDL technique is a standard request-response pattern. In this pattern, however, the client request is defined by one WSDL, and the server response is defined by another WSDL.</p> <p>You implement this scenario in two stages:</p> <ul style="list-style-type: none"> <li>Import the Request WSDL file, using the <b>Import WSDL from</b> import command.</li> <li>Import the Response WSDL file, using the <b>Import WSDL from</b> import command, enabling the <b>Import as Server Response</b> option..</li> </ul>

For task details, see "[How to Create an API Test for an Asynchronous Web Service](#)" below.

## How to Create an API Test for an Asynchronous Web Service

### Relevant for: API testing only

This task describes how to create an API test for testing an asynchronous Web service.

The following section describes the following tasks:

- ["Create a test for WS-Addressing" below](#)
- ["Create a test for HTTP Receiver" on the next page](#)
- ["Create a test for a Web service publish subscribe pattern" on page 601](#)
- ["Create a test for Dual WSDL Files" on page 601](#)

### Create a test for WS-Addressing

1. In the toolbar, click the **Import WSDL** button  and select **Import WSDL from URL or UDDI** or **Import WSDL from File or ALM Application Component**.
2. In the Import WSDL dialog box, navigate to the location of your WSDL file and press Import.
3. From the Toolbox pane, add a Web service step to the canvas.
4. In the Properties pane, open the Asynchronous tab  and select **This is an asynchronous call** box.

5. Specify a value for the **Listen for response on** property. This is the port to which you expect the server to respond.
6. In the Properties pane, open the **Security Settings** tab  in the Properties pane.
7. In the Security Settings tab, clear the **Use the port's security settings** option (if necessary).
8. Select the **WS Addressing** tab.
9. Select a WS-Addressing **Version** and provide and a URL and port (same port as defined for the **Listen for response on** property) in the **Reply to** box, to indicate the destination of the server response.

## Create a test for HTTP Receiver

This test enables you to have an HTTP Receiver in which the client receives the response:

1. In the Toolbox pane, expand the Network node and add a **HTTP Receiver** step to the canvas.

**Note:** Make sure you are logged in as an administrator. Administrator privileges are required to run **HTTP Receiver** steps.
2. In the Properties pane, open the **General** tab  and set the property values for the HTTP Receiver activity.
3. In the **HTTP Receiver** tab , set the values.
4. In the **Filter** tab , set a filter for the HTTP message received from the server.
5. From the Toolbox pane, expand the **Flow Control** node and add a **Wait** step onto the canvas.
6. In the Properties pane, open the **Input/Checkpoints** tab .
7. In the Input/Checkpoints tab, specify values for the timeout properties and add one or more completion events. You can link to the completion event from a prior **HTTP Receiver** step.
8. If required, add additional activities from the Toolbox pane into the **HTTP Receiver** flow.

**Tip:** If your test needs to listen to more than one message, receiver steps (such as **HTTP Receiver** or Web Service calls set up as receivers) can be data driven and placed inside a loop. The placement of the **Wait** step—inside or outside of the loop—depends on whether the send order matters:

- If the messages to be sent to the receiver are expected in a specific order, you must place the **Wait** step inside the receiver step's frame. All steps that are contained within the receiver can be data driven using this loop.
- If however, the messages are expected in a random order, place the **Wait** step outside the receiver step. Steps that are contained within the receiver should not be data driven using the same loop as the receiver step and should not link to other steps outside the receiver.

## Create a test for a Web service publish subscribe pattern

This test enables you to check that messages are properly published to the client:

1. In the toolbar, click the **Import WSDL** button  and select **Import WSDL from URL or UDDI** or **Import WSDL from File or ALM Application Component**.
2. In the Import WSDL dialog box, navigate to the location of your WSDL file and press **Import**. Make sure to select the **Import as server** option when importing.
3. From the Toolbox pane, add a Web service step to the canvas.
4. In the Properties pane, open the **Input/Checkpoints** tab  and set the input or output property values.
5. In the Toolbox pane, expand the **Flow Control** node and add a **Wait** step to the canvas.
6. In the Input/Checkpoints tab, add values for the timeout properties and add one or more completion events.

## Create a test for Dual WSDL Files

This test enables you to use one WSDL for the request and another for the response:

1. In the toolbar, click the **Import WSDL** button  and select **Import WSDL from URL or UDDI** or **Import WSDL from File or ALM Application Component**.
2. In the Import WSDL dialog box, navigate to the location of your WSDL file and press **Import**. Make sure to select the **Import as Server Response** option when importing.
3. From the Toolbox pane, add a Web service step to the canvas.
4. Drag a Web service operation with the response onto the canvas, after the request step.

# Troubleshooting and Limitations - Asynchronous Testing

## **Relevant for: API testing only**

For a Web service imported as a server response:

- When enabling the SSL option, UFT temporarily binds the SSL certificate to the specified port on a system, **http.sys**, level. If you end the UFT.exe process from the Task Manager during the listening stage after the binding was added, the binding will not be automatically removed from the system.

**Workaround:** Remove the binding manually using a utility such as httpcfg.exe or netsh.exe.

- When working with SSL, certificates from a file are not supported. If you move the test to another machine, the certificate will not be available.

**Workaround:** Add the certificate to the local machine store before running the test. If desired, remove it after you finish working with the test.

# Chapter 48: API Testing Extensibility

## Relevant for: API testing only

The HP UFT installation includes the capabilities to create custom activities for the **Toolbox** pane. After defining a custom activity, you can drag it onto the canvas as you would with any other built-in activity.

This chapter includes:

- [Creating New Activities - Overview](#) ..... 604
- [Custom Activity Files](#) ..... 604
- [Runtime Files](#) ..... 605
- [Signature Files](#) ..... 605
- [Addin Files](#) ..... 610
- [Resource Files](#) ..... 612
- [How to Use the Wizard to Create a Custom Activity - C#](#) ..... 612
- [How to Use the Wizard to Create an Activity - Java](#) ..... 614
- [How to Manually Create a Custom Activity in C#](#) ..... 616
- [How to Create a Runtime File](#) ..... 619
- [Troubleshooting and Limitations - Extensibility \(API Testing\)](#) ..... 624

# Creating New Activities - Overview

## **Relevant for: API testing only**

UFT enables you to create custom API testing activities to extend the capabilities of the product.

Once you create a new activity through this mechanism, it is available in the **Toolbox** pane for all future tests.

For most custom activities, you can use the Activity Wizard. For C# users, the wizard creates a Visual Studio project into which you can add your own C# code. Java users can edit .java files which will be compiled into .class files. You then deploy the new activity into UFT. For details, see "[How to Use the Wizard to Create a Custom Activity - C#](#)" on page 612.

Advanced users can build custom activities manually, without the wizard. For details, see "[How to Manually Create a Custom Activity in C#](#)" on page 616.

## Custom Activity Files

## **Relevant for: API testing only**

This section describes the structure and content of the files required to manually define a new activity in UFT. The following information is not relevant if you are using the Activity wizard.

To create a custom activity, you need to define the following files on all machines upon which you intend to run the test:

<a href="#">"Runtime Files"</a>	The DLL that UFT invokes to run the activity.
<a href="#">"Signature Files"</a>	An XML file that defines the input and output properties, events, and the runtime class that executes the activity.
<a href="#">"Addin Files"</a>	An XML file that references all of the activity component data.
<a href="#">"Resource Files " on page 612</a>	Files to store text strings used by your activity. These files are optional.

All of the custom files—Signature, Addin, and Runtime—should be stored in the <Installation\_folder>\addins\CustomerAddins\<addin\_name> folder. This enables UFT to load them during startup.

The product's installation includes a sample project in the ExtensibilitySamples folder. Use this sample as a basis for a new activity.

For more details, see "[How to Manually Create a Custom Activity in C#](#)" on page 616.

**Note:** This is a preliminary version of the SDK (Software Development Kit). It enables you to extend the capabilities of the product. However, this SDK is subject to change in a future release, and these changes might require you to update any code that uses this preliminary version. Although

HP endeavors to keep these changes to a minimum, we cannot guarantee that extensions created using the preliminary version of the SDK will continue to work without modification when upgrading to a new version of HP UFT.

## Runtime Files

### **Relevant for: API testing only**

When creating a custom activity, you must provide a DLL to run when executing the activity.

Note the following when creating your activity's code:

- When creating a custom activity, you must use methods that are included in the **STActivityBase** class.
- The **STExecutionResult** object is the return value of the **ExecuteStep** method. It receives the parameters: **STExecutionStatus Status** and optionally, **string msg**.
- **STExecutionStatus** is an **enum** type that can be set to the following values:
  - ActivityFailure
  - ActivityStopTest
  - ApplicationUnderTestFailure
  - Equals
  - ReferenceEquals
  - Success
  - TestStopped
- Place your executable code within the **ExecuteStep** function.
- You must compile the DLL with a **Target Framework** of Framework 4.0, available in Microsoft Visual Studio 10.

You can use the sample `ReportMessageActivitySample.sln` located in the product's `ExtensibilitySamples` folder as a basis for your project.

For task details and an example, see "[How to Create a Runtime File](#)" on page 619.

## Signature Files

### **Relevant for: API testing only**

The signature file describes the activity to UFT. It typically has a **Resource** element followed by the following sections: **GeneralProperties**, **InputProperties**, **OutputProperties**, **Tabs**, and **Events**. The signature file must have an `.xml` extension.

The signature file can contain the following information:

Element	Attributes	Description
<b>Resource Element</b>	<b>type</b>	The type of entity. In this case, the type is <b>Activity</b> .
	<b>id</b>	A unique string that identifies the activity. This string is used when writing event handler code for the activity.
	<b>version</b>	The version of the current addin mechanism, for example 1.0.0
	<b>group</b>	The parent group (in the Toolbox) pane to which to add the activity.
	<b>shortname</b>	The short name of the activity displayed in the hint area of the Toolbox pane.
	<b>description</b>	The description of the activity displayed in the hint area of the Toolbox pane.
	<b>assembly</b>	The DLL file to call when running the activity. This .dll file is stored in the same folder as the signature and addin files.
	<b>className</b>	The class implemented by the activity. This class must inherit from the STActivityBase class.
	<b>image</b>	An image file for the icon representing the activity. This image is stored in the same folder as the signature file.
	<b>visible</b>	A boolean value indicating whether to display the activity in the Toolbox pane
<b>Section Element</b>	<b>xmlns</b>	The namespace defining the schema for the signature file. Keep the default value.
	<b>xmlns:xsi</b>	The schema instance used for the signature file. Keep the default value.
	<b>xmlns:Location</b>	The URL of the signature file's schema (Signature.xsd), referenced by the name space. Keep the default value for this.
	<b>name</b>	The internal name of the section. If you use a sub-element, it is recommended that you set the value of the name of the sub-element, for example name="Tab" or name="Alerts".
	<b>source</b>	A boolean value indicating the source of the section.
	<b>dest</b>	A boolean value indicating the destination path of the section.
	<b>checkpoint</b>	A boolean value indicating whether to display a checkpoint checkbox in the Validate column for the activity.
<b>Section Element - Sub Attributes</b>	<b>isSharedMetadata</b>	A boolean value indicating whether to share the section's meta data with other sections.
	<b>propertiesType</b>	The type of the properties in the section, for example, "XML"
	<b>showXmlControls</b>	A boolean value indicating whether to display the Text and XPath tabs in the Input/Checkpoints tab.
	<b>displayName</b>	The name of the section as it will be displayed in the Properties pane.
	<b>Tab</b>	The tabs to display in the Properties pane for the activity. This sub-element can include the following attributes: <ul style="list-style-type: none"> <li>• <b>name</b>. The internal name of the tab. Some of the built-in ones are General, InputOutput, Events, Attachments, and SOAPFault.</li> </ul>

	<ul style="list-style-type: none"> <li><b>id.</b> The id of the tab referred to be the API. The id usually uses the name with an added suffix, "Tab". For example, GeneralTab, InputOutputTab, and EventsTab.</li> <li><b>CanBeInToolbox.</b> A boolean value indicating if the activity can be shown in the Toolbox pane's toolbar</li> <li><b>CanBeInPropertySheet.</b> A boolean value that indicates if the activity's tab is displayed in the Properties pane.</li> <li><b>CanBeInDataLinkDialog.</b> A boolean value that indicates if the activity can be displayed in the Select Link Source dialog box.</li> </ul> <p><b>Note:</b> To use the default tabs - <b>General</b>, <b>Input/Checkpoints</b>, and <b>Events</b>, you do not need to include this element. If you want to omit one of the tabs or add extra ones, then you need to include the <b>Tabs</b> sub-element and specify the desired tabs.</p>
<b>Alert</b>	<p>Enables you to use alerts for the properties in the section. This sub-element can include any of the following attributes:</p> <ul style="list-style-type: none"> <li><b>constraint.</b> The reason to show the alert, for example, NullValueConstraint.</li> <li><b>target.</b> The Xpath of the property to which to apply the constraint.</li> <li><b>section.</b> The internal name of the section containing the properties.</li> <li><b>type.</b> The type of alert, such as error or warning</li> </ul>
<b>Events</b>	<p>The events that are available for event handler code in this activity. This sub-element can include any of the following attributes:</p> <ul style="list-style-type: none"> <li><b>name.</b> The internal name of the event. Use one of the built-in names or define a custom one. <ul style="list-style-type: none"> <li><b>CodeCheckpointEvent.</b> Enables you to create an event handler to run when the test is verifying checkpoints.</li> <li><b>BeforeExecuteStepEvent.</b> Enables you to create an event handler to run before executing the activity.</li> <li><b>AfterExecuteStepEvent.</b> Enables you to create an event to run after executing the activity.</li> <li><b>&lt;custom event&gt;.</b> A custom event that you define.</li> </ul> </li> <li><b>description.</b> A textual description of the event.</li> <li><b>eventArgs.</b> The source of the arguments for the event. The standard argument for <b>BeforeExecuteStepEvent</b> and <b>AfterExecuteStepEvent</b> event is STActivityBaseEventArgs. The built-in value for the <b>CodeCheckpointEvent</b> is CheckpointEventArgs.</li> </ul> <p><b>Note:</b> To access the default events: <b>CodeCheckpoint</b>, <b>BeforeExecute</b>, and <b>AfterExecute</b>, you need to include only the <b>Events</b> tab in the <b>Tab</b> sub-element, but you do not need to use the <b>Events</b> sub-element. If you want to omit one of the events or add custom events, then you need to include this sub-element and specify the desired events.</p>

## Property Definitions

### Relevant for: API testing only

For all sections that use properties, you can define property definitions for these sections. The built-in sections that use properties are:

- **GeneralProperties.** Defines the properties in the **General** tab of the Properties pane, for example **Step ID** and **Name**.
- **InputProperties.** Defines the input properties located in the **Input** pane of the Properties pane's **Input/Checkpoints** tab.
- **OutputProperties.** Defines the output properties located in the **Checkpoints** pane of the Properties pane's **Input/Checkpoints** tab.

Property Definitions can contain the following:

Type	Name	Description
<b>Elements/subelements</b>  <b>Note:</b> The elements and attributes are defined in the standard XML schema file, <a href="http://www.w3.org/2001/XMLSchema">http://www.w3.org/2001/XMLSchema</a> , or the built-in types.xsd schema, located in the <Installation_Folder>/dat/schema folder. The level number indicates the level of the element or sub-element in the hierarchy.	<b>xs:schema</b>	The schema namespaces for the properties, as described by the <b>xml:ns</b> attribute. Keep the default value for this attribute.
	<b>xs:import</b>	The namespace to import using the <b>namespace</b> and <b>schemaLocation</b> attributes. Keep the default value for this attribute.
	<b>xs:element</b>	The element to define, using the attributes described in the table below.
	<b>xs:simpleType</b>	A tag indicating the beginning of definitions of a simple type property.  <b>Note:</b> You only need to enclose a simple type element with this tag, if you want to do enumeration with a drop-down list. For example, the following definition does not require an <b>xs:simpleType</b> tag. <pre>&lt;xs:element             name="ClientCertificate"             type="types:Certificate"             types:displayName="Client             certificate" /&gt;</pre>
	<b>xs:complexType</b>	A tag indicating the beginning of definitions for a node of multiple properties.
	<b>xs:sequence</b>	A tag indicating the beginning of a list of properties in a complex type property.
	<b>xs:restriction</b>	A tag restricting the value of the enumeration values of a property, using the <b>base</b> attribute.

		To restrict <b>String</b> type values, use <code>base="xs:string"</code> .
	<b>xs:enumeration</b>	A tag indicating the beginning of list of values in the drop-down list for a property, using the <b>value</b> attribute.
	<b>xs:annotation</b>	An annotation for the element Use an <b>xs:documentation</b> sub-element to compose text that will appear below the properties grid in the Properties pane.
<b>Element Attributes</b>  The following table describes the primary attributes of the <b>xs:element</b> . For attributes in the standard XML schema, use an <b>xs:</b> prefix in the value, for example standard types use <code>type=xs:string</code> or <code>type=xs:int</code> .  For types defined in the <b>Types.xsd</b> schema, use a <b>types:</b> prefix in the attribute name. For example <code>types:displayName</code> .	<b>name</b>	The internal name of the property or grid in the Properties pane. This is the name referenced by other calls and by the event handlers code. This is not the name displayed in the Properties pane's <b>Name</b> column.
	<b>type</b>	The type of property. Some common values are:  <code>xs:string</code> , <code>xs:int</code> , <code>xs:boolean</code> , Multipart, Header, Part. For a value defined in the Types schema, use the <b>types:</b> prefix. For example <code>type="types:filePath"</code> .
	<b>minOccurs</b>	The minimum number of array elements for which the user must provide. For none, specify "0".
	<b>maxOccurs</b>	The maximum number of array elements the user may provide. To allow an unlimited amount, specify "unbounded".
	<b>types:visible</b>	When <b>true</b> , enables the parameter to be visible even before being expanded by the Add Array Element command.
	<b>types:argType</b>	The type of the property: "XML" or "Object".
	<b>types:displayName</b>	The property name as it will appear in the Properties pane.

## Simple Elements with Enumeration

The following **ReportMessageActivitySample** example defines an input parameter, **Status**, with an enumeration attribute. This code creates a drop-down list of values in the Properties pane's input property grid.

```
<xs:element name="Status" default="Done" types:displayName="Status">
 <xs:simpleType>
 <xs:restriction base="xs:string">
```

```

 <xss:enumeration value="Done"/>
 <xss:enumeration value="Pass"/>
 <xss:enumeration value="Fail"/>
 </xss:restriction>
</xss:simpleType>
</xss:element>

```

## Complex Array Elements

The following sample defines a complex property, with a Key and Value pair of values.

```

<xss:complexType name="NameValueType">
 <xss:sequence>
 <xss:element name="Key" type="xs:string"/>
 <xss:element name="Value" type="xs:string"/>
 </xss:sequence>
</xss:complexType>

```

## Addin Files

### Relevant for: API testing only

The addin file provides the references for the activity you are defining. The file is in XML format and contains information such as activity names, dependencies, and runtime DLLs.

The addin file should be located in the installation directory under the `addons\CustomerAddins\<addin_name>` folder, together with the signature file. The addin file must have an .addin extension.

Each addin file should contain the following sections:

Section	Element	Description
<b>Addin Section</b>	<b>&lt;Addin&gt;</b>	The basic details about the section, including any of the following attributes: <ul style="list-style-type: none"> <li><b>name.</b> the name of the addin.</li> <li><b>author.</b> the creator of the activity.</li> <li><b>copyright.</b> the full path of a text file with the copyright information.</li> <li><b>description.</b> a textual description of the activity.</li> <li><b>version.</b> The addin file version, set to 1.0.</li> </ul>
	<b>Path</b>	Details about the location of the activity, including any of the following attributes: <ul style="list-style-type: none"> <li><b>name.</b> The logical path scanned by the framework, to identify addins. The physical location of this folder is <code>addons\CustomerAddins\&lt;addin_name&gt;</code></li> </ul>
	<b>Activity</b> (sub-element of the Path)	Additional information about the activity, including any of the following attributes:

	attribute)	<ul style="list-style-type: none"> <li><b>id.</b> An identifying string corresponding to the ID in the signature file.</li> <li><b>displayName.</b> The activity's display name in the <b>Toolbox</b> pane.</li> <li><b>signatureFile.</b> The name of the XML signature file.</li> </ul>
<b>Manifest Section</b>	<b>Identity</b>	<p>Basic details about the addin, including any of the following attributes:</p> <ul style="list-style-type: none"> <li><b>name.</b> The activity name corresponding to the <b>ID</b> in the signature file. When referring to this addin as a dependency, use this name.</li> </ul>
	<b>Dependency</b>	<p>The activity on which the current activity is dependent, including any of the following attributes:</p> <ul style="list-style-type: none"> <li><b>addin.</b> The identity name of the dependent activity, from the name attribute in the <b>Manifest</b> section of its addin file.</li> <li><b>requirePreload.</b> A boolean value indicating whether to preload the dependent addin before loading the current one.</li> </ul>
Runtime Section	Import	<p>An assembly to import when running the activity.</p> <p>You can use the <b>assembly</b>, which is the name of an assembly. Use the DLL name without the DLL extension.</p> <p><b>Note:</b> To import an addin from another activity, precede the addin name with a colon. For example, :HP . ST . Fwk . DesignerModel imports the DesignerModel addin.</p>

The following example shows the **ReportMessageActivitySample.addin** file. For multiple activities, use unique **Addin** files.

```
<?xml version="1.0" encoding="utf-8"?>
<AddIn name = "HP Report Message Activity Sample"
 author = "John Doe"
 copyright = "C:\Copyrights\copyright.txt"
 description = "Extensibility Sample - Report Message Activity"
 version="1.0">
 <Manifest>
 <!--Must be unique -->
 <Identity name = "ReportMessageActivitySample"/>
 </Manifest>
 <Runtime>
 <Import assembly=":HP . ST . Fwk . DesignerModel"/>
 </Runtime>
 <Path name = "/ST/Activities">
 <!--Misc Activities -->
 <Activity id = "ReportMessageActivitySample"
 displayName = "ReportMessageSample"
 signatureFile = "ReportMessageActivitySample.xml"
 assembly="ReportMessageActivitySample.dll"/>
 </Path>
</AddIn>
```

## Resource Files

### Relevant for: API testing only

You can use resource files to retrieve values for elements and attributes. The **fromResource** function lets you name the resource containing the values.

In the following example, the signature file retrieves the **shortName** and **description** from a resource file.

```
<Resource
 type="Activity"
 id="ReportMessageActivitySample"
 version="1.0.0"
 group="Miscellaneous"
 shortName="fromResource(conc_str_short_name)"
 description="fromResource(conc_str_description)"
```

The resources are defined in a standard Microsoft ResX Schema version 2.0 Resource file.

```
<data name="conc_str_description" xml:space="preserve">
 <value>Reports an activity's run status to the log</value>
</data>
<data name="conc_str_short_name" xml:space="preserve">
 <value>Report Message</value>
</data>
```

The resource reference must be a compiled file with a **.resources** extension, compiled from the ResX source file and stored in the same folder as the signature file.

You can generate the compiled file as a post-build operation using the **resgen** utility. For example:

```
resgen STBasicActivity.resx STBasicActivity.resources.
```

## How to Use the Wizard to Create a Custom Activity - C#

### Relevant for: API testing only

This task describes how to create a new activity, using C#, and deploying it in UFT.

This task includes the following steps:

- ["Run the Activity Wizard" on the next page](#)
- ["Add execution code" on the next page](#)
- ["Add Logger code - optional" on the next page](#)

- "Add a Report statement - optional" below
- "Compile the project into a DLL" on the next page
- "Deploy the activity in UFT" on the next page

## 1. Run the Activity Wizard

- a. Open the Activity Wizard (**Start > All Programs > HP Software > HP Unified Functional Testing > Tools > Activity Wizard** or <UFT installation folder>\bin\ActivityWizard.exe).
- b. In the wizard's **General Properties** pane, select the **C#** as the **Language**.
- c. Follow the steps of the wizard to create your activity.

## 2. Add execution code

- a. In the final screen of the wizard, click **Open Folder** to open the <Activity Name> folder, corresponding to the activity name you specified in the wizard. Navigate to the SourceCode subfolder and locate the <Activity Name>.cs file.

**Caution:** Do not close the Activity Wizard after this step.

- b. In the <Activity Name>.cs file, add your execution code to the **ExecuteStep** function, as follows:

```
protected override STExecutionResult ExecuteStep()
{
 try
 {
 //*****
 // Execution code goes here //*****
 ...
 }
}
```

## 3. Add Logger code - optional

In the <Activity Name>.cs file, add information for the log using the **LogInfo**, **LogDebug**, or **LogError** statements. For example:

```
protected override STExecutionResult ExecuteStep()
{
 try
 {
 LogInfo("Log Message 1");
 LogDebug("Log Message 2");
 LogError("Log Message 3");
 ...
 }
}
```

## 4. Add a Report statement - optional

In the <Activity Name>.cs file, add a **Report** statement. For example:

```
protected override STExecutionResult ExecuteStep()
{
try
{
 DetailsReport = DetailsReport.Replace("\\n", "
");
 this.Report("Message", DetailsReport); ;
...
}
```

## 5. **Compile the project into a DLL**

In your IDE, build the project and make sure the current <Activity Name> .dll file is in the new activity folder that you specified in the wizard.

## 6. **Deploy the activity in UFT**

- a. In the final wizard screen, click **Deploy in UFT**.
- b. Click **Finish** to close the wizard and restart UFT.

# How to Use the Wizard to Create an Activity - Java

## Relevant for: API testing only

This task describes how to create a new activity using Java code, and deploy it in UFT.

This task includes the following steps:

- "Prerequisite" below
- "Run the Activity Wizard" below
- "Edit the code" on the next page
- "Add Logger code - optional" on the next page
- "Add a Report statement - optional" on the next page
- "Compile the Java into a class" on page 616
- "Deploy the activity" on page 616
- "How to Use the Wizard to Create an Activity - Java" above

### 1. **Prerequisite**

Make sure you have a JAVA\_HOME environment variable defined on your machine indicating the parent JDK folder.

### 2. **Run the Activity Wizard**

- a. Open the Activity Wizard from the product's start menu (**Start > All Programs > HP Software**

> **HP Unified Functional Testing > Tools >Activity Wizard** or <UFT installation folder>\bin\ActivityWizard.exe).

- b. In the wizard's **General Properties** pane, select the **Java** as the **Language**.
- c. Follow the steps of the wizard to create your activity.

### 3. **Edit the code**

- a. On the final screen of the wizard, click **Open Folder** to open the <**Activity Name**> folder, corresponding to the activity name you specified in the wizard. Navigate to the <**Activity Name**>\hp\st\ext\java subfolder and locate the MyLogic.java file.

**Caution:** Do not close the Activity Wizard after this step.

- b. Edit the ExecuteLogic function inside the MyLogic.java file. Make sure to keep the **Properties** definition.

```
public Properties Props = new Properties();
public ExecutionResult ExecuteLogic()
{
 try{
 //*****
 // Execution code goes here
 //*****
 return ExecutionResult.Success;
 }
 ...
}
```

### 4. **Add Logger code - optional**

In the MyLogic.java file, add information for the log using the **Logger.LogInfo**, **Logger.LogDebug**, or **Logger.LogError** statements. For example:

```
try{
 ...
 Logger.LogInfo("Log Message 1");
 Logger.LogDebug("Log Message 2");
 Logger.LogError("Log Message 3");
 ...
 return ExecutionResult.Success;
}
```

### 5. **Add a Report statement - optional**

In the MyLogic.java file, add a Report statement, **Reporter.Report**, using key value combinations. For example:

```
try{
...
 Reporter.Report{"Name", "John");
...
return ExecutionResult.Success;
}
```

## 6. **Compile the Java into a class**

- a. In your design IDE, add the ServiceTestCall.jar file to the build path.
- b. In UFT, run the CompileJavaFiles batch file in the <Activity Name>\hp\custom\java\activity folder to compile all java files into a class. This utility only compiles the files in its folder.

## 7. **Deploy the activity**

- a. In the final wizard screen, click **Deploy in UFT**.
- b. Click **Finish** to close the wizard and restart UFT.

# How to Manually Create a Custom Activity in C#

## **Relevant for: API testing only**

This task describes how to create a new activity and implement it into UFT.

To run a test with the custom activity on another machine, you need to copy all of the custom files to its <Installation\_Folder>\addins\CustomerAddins\<addin\_name> folder.

This task includes the following steps:

- "[Prerequisite - create a runtime file](#)" below
- "[Create a signature file](#)" on the next page
- "[Create an addin file](#)" on page 618
- "[Provide a graphic for your activity - optional](#)" on page 618
- "[Check the implementation](#)" on page 619

## 1. **Prerequisite - create a runtime file**

Create a C# project that implements your activity's actions in the addins\CustomerAddins\<addin\_name> folder. For task details, see "[How to Create a Runtime File](#)" on page 619.

## 2. Create a signature file

- a. Create a new signature file with an .xml extension in the addins\CustomerAddins\<addin\_name> folder, together with the runtime file. Use the sample project in the <installation\_folder>\ExtensibilitySamples folder as a basis for your custom signature file.
- b. Customize the **Resource** section or copy the code provided below, modifying the bolded text for your needs.

```
<Resource
 type="Activity"
 id="ReportMessageActivitySample"
 version="1.0.0"
 group="Miscellaneous"
 shortName="ReportMessageActivitySample"
 description="ReportMessageActivitySample allows you to send a custom
message to the report and/or log. "
 assembly="ReportMessageActivitySample.dll"
 className="ReportMessageActivitySample.ReportMessageActivitySample"
 image="toolbox_ReportMessageActivitySample.png"
 visible="true"
 xmlns="http://hp.st.schemas/signature/v1.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://hp.st.schemas/signature/v1.0
 ../../dat/schemas/Signature.xsd"
 >
```

- c. Add the required sections, such as GeneralProperties, InputProperties, Tabs, Events and so forth
- d. Add properties to the relevant sections.
  - **GeneralProperties.** Properties displayed in the Properties pane's **General** tab. In most cases you can use the section as it appears in the sample file, without any modifications. By default, it will provide the **Step ID** and **Name** properties.
  - **InputProperties.** Properties displayed in the Properties pane's **Input/Checkpoints** tab, in the **Input** pane.
  - **OutputProperties.** Properties displayed in the Properties pane's **Input/Checkpoints** tab, in the **Checkpoints** pane.
- e. Specify any external resource files.
- f. Close the file with the </Resource> tag.

```
</Resource>
```

For more details about the structure of the signature file, see "[Signature Files](#)" on page 605.

### 3. Create an addin file

- a. Create a new file with an .addin extension in the <installation directory>\addons\CustomerAddins\<addin\_name> folder, together with the signature file.
- b. Use the sample addin file in the <installation folder>\ExtensibilitySamples folder as a basis, or copy the code provided below, modifying the bolded text for your needs.

```
<?xml version="1.0" encoding="utf-8"?>
<AddIn name = "HP Report Message Activity Sample"
 author = "John Doe"
 copyright = "prj:///doc/copyright.txt"
 description = "Extensibility Sample - Report Message Activity"
 version="1.0">
<Manifest>
 <!--Must be unique -->
 <Identity name = "ReportMessageActivitySample"/>
</Manifest>
<Runtime>
 <Import assembly=":HP.ST.Fwk.DesignerModel"/>
</Runtime>
<Path name = "/ST/Activities">
 <!--Misc Activities -->
 <Activity id = "ReportMessageActivitySample"
 displayName = "ReportMessageSample"
 signatureFile = "ReportMessageActivitySample.xml"
 assembly="ReportMessageActivitySample.dll"/>
</Path>
</AddIn>
```

- c. Create a unique Addin file for each activity—do not define multiple activities in a single Addin file.
- d. Define post-build tasks such as resgen.
- e. Compile the project and copy the DLL to the <Installation\_Folder>\addons\CustomerAddins\<addin\_name> folder.

For additional details, see ["Runtime Files" on page 605](#).

### 4. Provide a graphic for your activity - optional

- a. Copy an icon image for your activity into the <Installation\_Folder>\addons\CustomerAddins\<addin\_name> folder. This file should meet the following requirements:
  - a .png extension
  - sized at 16 x 16 pixels
  - 8-bit color depth

- b. Specify the name of the image file in the signature file's **Resource Element**.

## 5. **Check the implementation**

- a. Reopen the application and drag the new activity into the Test Flow. Verify that the activity and its properties appear as expected.
- b. Provide property values.
- c. Run the test and observe the Output log and the run results.
- d. Enable checkpoints to verify the results and rerun the test.

# How to Create a Runtime File

## **Relevant for: API testing only**

This section contains the following topics:

- ["Add Using statements" below](#)
- ["Specify the namespace and class" below](#)
- ["Set the internal logging" on the next page](#)
- ["Initialize the properties" on the next page](#)
- ["Retrieve the property values" on the next page](#)
- ["Define events" on page 621](#)
- ["Execute the step" on page 622](#)
- ["Set the status" on page 622](#)
- ["Compile the runtime file" on page 623](#)

**Note:** You must compile the DLL with a **Target Framework** of Framework 4.0.

## 1. **Add Using statements**

In your runtime file, provide the mandatory **using** statements. In your solution, you must also add a reference to the .dll files. The .dll files are located in the products installation's /bin folder. You must always add a reference to HP.ST.Fwk.RunTimeFWK.dll. If you are using internal logging, you must also add a reference to log4net.dll.

The following example shows the **Using** statements in the sample .cs file.

```
using HP.ST.Fwk.RunTimeFWK;
// If you need to implement Internal Logging
using log4net;
```

## 2. **Specify the namespace and class**

Define the namespace and provide the activity's runtime code. The class you define for your custom activity must inherit from the `STActivityBase` class. For example:

```
namespace ReportMessageActivitySample
{
 [Serializable]
 public class ReportMessageActivitySample : STActivityBase
 {
```

### 3. Set the internal logging

Use the built-in logger manager to instruct the activity to create an internal log during runtime. This example gets the property values of the input properties and sends the output to either the run results only or to the run results and Output window. For example:

```
/// <summary> /// Internal log
/// </summary>
private static readonly ILog log =
 LogManager.GetLogger(typeof(ReportMessageActivitySample));
const string runResults = "Run Results";
const string runResultsAndOutputWindow = "Run Results and Output Window";
```

For details about other logging options, see ["Assert Object" on page 804](#).

### 4. Initialize the properties

Initialize the custom Input and Output properties that you define in the signature file. The following example initializes the three input properties: **Status**, **Message**, and **Destination**. For example:

```
/// <summary>
/// Initializes properties.
/// </summary>
/// <param name="ctx">The runtime context</param>
public ReportMessageActivitySample(ISTRuntimeContext ctx, string name)
: base(ctx, name)
{
 this.Status = String.Empty;
 this.Message = String.Empty;
 this.Destination = String.Empty;
}
```

### 5. Retrieve the property values

This section retrieves or sets the input property values. For example:

```
/// <summary>
```

```
/// Gets or sets the status of the message to report.
/// </summary>
public string Status { get; set; }
/// <summary>
/// Gets or sets the details of the message to report.
/// </summary>
public string Message { get; set; }
/// <summary>
/// Gets or sets the destination where the message should be reported to.
/// </summary>
public string Destination { get; set; }
```

If you have array type properties that are not described by a schema, for example, key/value pairs, you must initialize all the members of the array explicitly, and indicate the actual number of elements.

The following example initializes 40 elements for the `MyArrayName` property. It contains 40 key and value pairs.

```
this. MyArrayName = new MyPair[40];
for (int i=0; i<40; i++)
{
 this. MyArrayName [i] = new MyPair();
}
public MyPair[] MyArrayName;
public class MyPair
{
 string Key;
 string Value;
}
```

For arrays defined by a schema or WSDL, you can use the standard Select Link Source Dialog Box and link directly to the array element.

## 6. Define events

Define one or more custom events, that you will invoke later. For example:

```
public event EventHandler CustomerEvent;
private void InvokeCustomerEvent(EventArgs MyArg)
{
 EventHandler handler = this.CustomerEvent;
 if (handler != null)
 {
 handler(this, MyArg);
 }
```

```
}
```

## 7. Execute the step

Execute the step and send the runtime information to the log. Use the **STExecutionResult** data type and its **ExecuteStep** function defined in the **STActivityBase** class. For example:

```
protected override STExecutionResult ExecuteStep()
{
 string DetailsReport;
 if (this.Destination == runResultsAndOutputWindow)
 {
 LogInfo("\n" + this.Message.Replace("\\n", "\n"));
 }
 /// <summary>
 /// Reports message to test results and output window.
 /// </summary>
 // The line-breaks replacements allow the printing of multiple lines in the
 // report
 DetailsReport = this.Message;
 DetailsReport = DetailsReport.Replace("\\n", "
");
 DetailsReport = DetailsReport.Replace("\n", "
");
 this.Report("Message", DetailsReport);
```

If you defined a custom event, invoke it after the call to ExecuteStep.

```
...
protected override STExecutionResult ExecuteStep()
{
 InvokeCustomerEvent();
}
```

## 8. Set the status

Set the Status of the test run. The `ReportMessageActivitySample.cs` sample file uses enumeration to set the status, based on the **STExecutionResult** value. For example:

```
switch (this.Status)
{
 case "Done":
 this.Report(ReportKeywords.StatusKeywordTag, ReportKeywords.DoneValueTag);
 return new STExecutionResult(STExecutionStatus.Success);
 case "Pass":
 this.Report(ReportKeywords.StatusKeywordTag,
 ReportKeywords.SuccessValueTag);
```

```
 return new STExecutionResult(STExecutionStatus.Success);
 case "Fail":
 this.Report(ReportKeywords.StatusKeywordTag,
ReportKeywords.FailureValueTag);
 return new STExecutionResult(STExecutionStatus.Success);
 default:
 return new STExecutionResult(STExecutionStatus.Success);
 }
```

## 9. **Compile the runtime file**

After you customize the code, you compile the .dll. The .dll name should be the same as the name of the addin file. For example, the runtime file, ReportMessageActivitySample.dll corresponds to the ReportMessageActivitySample.addin file.

After you create the runtime file in your development environment, you reference the .dll from the signature file, and the signature file from the addin file.

# Troubleshooting and Limitations - Extensibility (API Testing)

## **Relevant for: API testing only**

This section describes troubleshooting and limitations for creating and using custom activities.

- You must have Visual Studio 2012 installed on the same machine as UFT to create custom activities using the Activity Wizard.
- The following error may occur if you place the signature file beneath a sub-folder of the **addin** folder.

```
ServiceTest was unable to drag and drop the activity: Type
'http://hp.vtd.schemas/types/v1.0:GeneralPropertiesType' is not declared.
```

**Workaround:** Modify the relative path of the Types schema. For example:

```
.schemaLocation="../../dat/schemas/Types.xsd".
```

- If you modify the activity structure in the signature file, you will be unable to open tests using that activity. To modify an activity structure, create a new activity with the new structure, replace all of the test steps using the old activity, and then remove the old activity implementation.
- Custom Java activities created in versions of UFT prior to 12.00 will fail when you run your test.

**Workaround:**

- a. Remove the ServiceTestCall.java interface from the project in which you created the custom Java activity..
- b. Change the project package name to something other than hp.st.ext.java.
- c. Add the ServiceTestCall.jar file to your classpath. This file is located in <UFT installation path>\Addins\ServiceTest\JavaCall\Java Interface\bin.
- d. Recompile your Java project.

# Part 7: Creating and Enhancing UFT Tests with Code

# Chapter 49: Using the UFT Editor

## **Relevant for GUI tests and components**

This chapter includes:

• <a href="#">Editing Text and Code Documents</a> .....	627
• <a href="#">Statement Completion in the Editor</a> .....	627
• <a href="#">Learn more!</a> .....	628
• <a href="#">Statement Completion Options</a> .....	628
• <a href="#">Statement Completion Considerations</a> .....	631
• <a href="#">Automatic Code Completion for GUI Testing</a> .....	632
• <a href="#">Searching and Replacing in the Editor</a> .....	633
• <a href="#">File and Item Types Included in String Searches</a> .....	635
• <a href="#">How to Use Code Snippets and Templates</a> .....	635
• <a href="#">How to Search for References or Classes in Documents in the Editor (API Testing Only)</a> .....	638
• <a href="#">Editor User Interface</a> .....	640
• <a href="#">Class and Function Browser Examples</a> .....	643
• <a href="#">Editor Icons</a> .....	644
• <a href="#">Troubleshooting and Limitations - Statement Completion</a> .....	646

# Editing Text and Code Documents

## **Relevant for: GUI actions, scripted GUI components, function libraries, and user code files**

You can write customized code for your tests by modifying actions, function libraries, and user code files in the Editor, as well as enhance your tests and function libraries using a few simple programming techniques.

The Editor supports common text and code editing features, including:

- **Statement completion.** For details, see "[Statement Completion in the Editor](#)" below and "[Automatic Code Completion for GUI Testing](#)" on page 632.
- **Bookmarks.**
- **Search and Replace.** For details, see "[Searching and Replacing in the Editor](#)" on page 633
- **Collapsible code sections.**
- **Zoom in/zoom out of code documents using the mouse wheel.**
- **Code templates.**

You can also define personalized options for using the Editor. For details on the available options, see the options in the Text Editor Tab of the Options Dialog Box. For a user interface description of the Editor, see "[Editor User Interface](#)" on page 640.

For a description of using the editor for **GUI tests and components**, see "[Programming in GUI Testing Documents in the Editor](#)" on page 647. For a description of using the editor for **API tests**, see "[Writing Event Handlers for API Test Steps](#)" on page 751.

## Statement Completion in the Editor

### **Relevant for: GUI actions, scripted GUI components, function libraries, and user code files**

Statement completion, which is similar to Microsoft's IntelliSense functionality, enables you to increase programming speed and accuracy by providing dynamic lists of items, in the form of tooltips, drop-down lists, or popup windows, while writing statements in the Editor.

As you type in the Editor, UFT displays items you can add to your statement, as well as the syntax relevant to what you are typing. UFT provides this type of statement completion information, when available, for:

- Activity-specific properties (API testing only)
- Methods or objects you can use in your API test step event handlers
- Function and method syntax
- Objects you create in your code
- Operations
- Properties or operations that return objects

- Structured parameters
- Variable definitions and methods
- Variables to which classes, objects or test objects are assigned
- VBScript classes, user defined functions, or constants (GUI actions and scripted components only)
- Reserved objects
- Test objects and collections (GUI actions and scripted components only)
- Utility objects

**Note: (for GUI testing only)** The list of available statements is displayed even if you typed an object that has not yet been added to the object repository. If the action contains a function, or the action or component is associated with a function library, the functions are also displayed in the list.

## Learn more!

- "[Statement Completion Options](#)" below
- "[Statement Completion Considerations](#)" on page 631

## Statement Completion Options

### Relevant for: GUI actions, scripted GUI components, function libraries, and user code files

The following table summarizes some statement completion options available when you enter specific items and keystrokes.

If you enter:	Followed by:	UFT displays:
An operation name	SPACE or Open parenthesis "("	The operation syntax, including its mandatory and optional arguments. When you add a step that uses an operation, you must define a value for each mandatory argument associated with the operation.
An argument	Comma (,)	The operation syntax, bolding the next argument for which you need to enter a value. Relevant if you enter a comma after any argument value other than the last one in a step.  <b>Note:</b> For certain operations, when you type the space or comma before an argument that has a predefined list of values, UFT displays the list of possible values.
An operation or function name	CTRL+SHIFT+SPACE or <b>For GUI testing:</b> Select <b>Edit &gt; Format &gt; Argument Info</b>	The statement completion (argument syntax) tooltip for that item.

If you enter:	Followed by:	UFT displays:
CTRL+SPACE or <b>When editing a GUI test: Edit &gt; Format &gt; Complete Word</b>		<p>A dynamic list of the relevant:</p> <ul style="list-style-type: none"> <li>• operations</li> <li>• properties</li> <li>• user-defined functions</li> <li>• constants and local variables relevant to the current programming scope</li> <li>• functions and methods relevant to the current programming scope</li> <li>• options relevant to your GUI test or component, including:           <ul style="list-style-type: none"> <li>• test objects</li> <li>• utility objects</li> <li>• collections</li> </ul> </li> </ul> <p><b>Note:</b> If there is only one relevant item defined, its name is automatically entered in the step, without opening the list. For example, if you typed the beginning of the item name before pressing CTRL+SPACE, and only one item matches the text you typed.</p>
The beginning of an item in the statement completion list		The list of items, highlighting the first item (alphabetically) that matches the text you typed. Pressing ENTER or SPACE adds the highlighted word to the step.

## Statement Completion Options for GUI Testing Only

The following table summarizes some additional statement completion options available in GUI tests only, when you enter specific items and keystrokes.

If you enter:	Followed by:	UFT displays:
<ul style="list-style-type: none"> <li>• A test object type</li> <li>• "Environment"</li> <li>• "Parameter"</li> <li><b>(for tests and scripted components)</b></li> </ul>	Open parenthesis "("	<p>Respectively, an alphabetically sorted list of the relevant available:</p> <ul style="list-style-type: none"> <li>• Test objects (for example, <code>Page()</code> displays a list of all the Page test objects in the object repository)</li> <li>• Environment variables (built-in, user-defined, and external)</li> <li>• Action parameters (output and input)</li> </ul> <p>If there is only one selection available, UFT automatically enters its name in quotes after the open parenthesis.</p>
<ul style="list-style-type: none"> <li>• A test object</li> <li>• A collection object</li> <li>• A variable that was assigned an object</li> <li>• An object that you created in the test or component (using <code>CreateObject</code>, for</li> </ul>	Period (.)	<p>Depending on the type of object and its hierarchy, a dynamic list of the relevant:</p> <ul style="list-style-type: none"> <li>• test objects (Editor only)</li> <li>• collections</li> <li>• operations</li> <li>• properties</li> <li>• registered functions</li> </ul>

If you enter:	Followed by:	UFT displays:
example)		<p><b>Example:</b></p> <ul style="list-style-type: none"> <li>The list displayed for a container object includes the objects it contains.</li> <li>The list displayed for a collection object includes collection methods.</li> <li>The list displayed for a variable that was assigned an Excel worksheet (Set var = CreateObject ("excel.application")) includes worksheet methods and properties.</li> </ul>
A variable that was assigned a VBScript class	Period (.)	A list of the relevant VBScript class methods, properties, and variables
A <b>With</b> statement	Period (.)	<p>A list of the operations and properties available for the relevant object.</p> <p><b>Note:</b> If you manually type a <b>With</b> statement (as opposed to using a menu command to create it), you must use the <b>Edit &gt; Format &gt; Apply "With" to Script</b> command (or press <b>CTRL+W</b>) to enable statement completion within the <b>With</b> statement.</p>
The <b>Object</b> property	Period (.)	<p>A list of the object's native operations and properties.</p> <p>Available <b>only if</b> the object data is currently available in the Active Screen (tests and scripted components only) or the open application.</p> <p>For details related to tests and scripted components, see "<a href="#">Statement Completion Considerations</a>" on the next page.</p>
A structured parameter	Colon (:)	<p>A list of the elements available, according to the structure definition.</p> <p><b>Example:</b></p> <p>If you define a ZMOVIE structure, with the elements TITLE, DIRECTOR, GENRE, and STARRING, and then map a parameter named Movie to a structure of this type, then when you type:</p> <pre>Parameter ("Movie:</pre> <p>the displayed list includes the elements of a ZMOVIE structure: TITLE, DIRECTOR, GENRE, STARRING.</p>
A structured parameter's element or sub-element	Period (.)	<p>A list of the sub-elements available, according to the structure definition.</p> <p><b>Example:</b></p> <p>If you type:</p> <pre>Parameter ("Movie:Starring.item[1].</pre> <p>the displayed list includes the sub-elements of a Starring element: FIRST_NAME, LAST_NAME.</p>

# Statement Completion Considerations

## Relevant for: GUI actions, scripted GUI components, function libraries, and user code files

When working with UFT's statement completion feature, consider the following:

<b>General Statement Completion Considerations</b>	<ul style="list-style-type: none"><li>To close the statement completion drop-down list without selecting from it, press <b>Esc</b>.</li><li>If you resize the frame in which the statement completion drop-down list is displayed, UFT subsequently uses the new size when it displays statement completion drop-down lists.</li><li>UFT might not display statement completion information if the statement is typed incorrectly and contains syntax errors. In many cases, you can view such errors in the "<a href="#">Errors Pane</a>" when you save your changes.</li></ul>
<b>Statement Completion Considerations for GUI Testing</b>	<ul style="list-style-type: none"><li>Although statement completion in function library documents is supported to help generate test object statements, it is generally not recommended to include a full object hierarchy statement in a function. It is preferable to make your functions generic so that they can be used with different objects.</li><li>In some cases, UFT needs to retrieve statement completion information from an object in the application. In such cases, you may experience a delay while typing in the Editor.</li><li>When you record in a browser other than Internet Explorer, UFT captures the dynamic HTML from the relevant browser. However, the Active Screen uses an Internet Explorer engine and thus renders Web-based pages as static HTML with the Internet Explorer DOM exposed.</li></ul> <p>Therefore, if you insert the <b>Object</b> property for a Web object when only Active Screen object data is available (the object is not displayed in an open application), then UFT statement completion displays the available native operations and properties for the Internet Explorer DOM, even if you recorded the object in another browser.</p> <ul style="list-style-type: none"><li>In the Editor, when working with Java or ActiveX objects, UFT dynamically retrieves the list of possible values for certain arguments from the object in the application. For UFT to retrieve the possible values, the application must be open and the relevant object must be visible. For example, UFT can retrieve the list of items in a specific Java list object, and display them as the possible values for the Item argument of the Select method.</li></ul> <p><b>Note:</b> When you edit a test during a recording session, UFT does not retrieve the possible argument values from the application.</p> <ul style="list-style-type: none"><li>If you assign an object to a variable, and then type the name of the variable followed by a period, UFT displays a list of the operations and properties available for the object.</li></ul> <p>In some cases, the value of a variable cannot be determined while editing the test (for example, if the value is set by a conditional assignment or returned by another function). In this case, UFT provides statement completion information according to the most recent line of code in which the value of the variable could be evaluated, if any.</p> <p>The following examples illustrate this:</p> <p><b>Example 1:</b></p> <pre>Line 1: Set x = CreateObject("Excel.Application") Line 2: z = GetValueFromUser() Line 3: If z = 2 Then Line 4:     Set x = CreateObject("Word.Application")</pre>

	<p>Line 5: End If Line 6: x.</p> <p>While editing this test, UFT cannot determine which object will actually be assigned to x in line 6. However, because the value of x can be evaluated independently in line 4, UFT displays the statement completion information relevant to the object "Word.Application" for the variable x in line 6.</p>
	<p><b>Example 2:</b></p> <pre>Line 1: Set x = CreateObject("Excel.Application") Line 2: Set x = MyGetObject() Line 3: x.</pre> <p>While editing this test, UFT cannot determine the type of object that the <b>MyGetObject</b> function returns (line 2). Therefore, in line 3 in the example above, UFT displays the statement completion information relevant to the object "Excel.Application", because line 1 is the most recent line of code in which the value of x could be evaluated. However, if line 2 were not preceded by a line in which the value could be evaluated, UFT would not display any statement completion information for x in line 3.</p>
<b>Statement Completion Considerations for API Testing</b>	<ul style="list-style-type: none"><li>After creating an event handler for a test step, you cannot access activity-specific properties until you save the entire test. Therefore, if you create the event handler and immediately begin entering code, properties relevant to your test step are not available.</li><li>When working with event handler code, only properties, parameters, and objects relevant to the current test step and event context are available in the statement completion drop-down menu. For example, if you are trying to create code for a BeforeExecuteStepEvent event, the properties and objects for the step's checkpoints are not available in the statement completion drop-down list.</li><li>When working with a Custom Code step, the statement completion drop-down list contains properties, parameters, and objects from any step preceding the Custom Code step as well as any parent activity.</li><li>When entering code for an API test (whether an event handler or a Custom Code step), the statement completion drop-down list contains numerous items that are not applicable to your test or usable in your code. You should only use documented methods (either documented in the UFT Help or the Microsoft C# reference). For details for UFT-specific objects and methods, see <a href="#">"API Test Event Coding Common Objects" on page 802</a> and <a href="#">"API Test Event Coding Common Methods" on page 815</a>.</li></ul>

## Automatic Code Completion for GUI Testing

### Relevant for: GUI actions, scripted GUI components, and function libraries

UFT provides automatic code completion features, to facilitate coding in the Editor. This includes some basic, static code snippets that you can insert automatically into your document, as well as templates that you can use to insert additional code snippets by typing specific keywords.

For example, if you enter the letters `if` at the beginning of an empty line in a GUI action, followed by a SPACE, UFT automatically enters:

```
If True Then
End If
```

The word `True` is highlighted, reminding you to replace it with the relevant condition.

You can also modify the templates provided, or build your own customized templates as needed, and define the keywords used to invoke the use of each template.

For example, you might repeat a complicated **If...Then** statement many times your document. You can use an existing template for the **If...Then** statement to create your own customized template with your more complicated code. You can also create templates from scratch, such as a comment block template, which might include information such as programmer identification, date added, or other details you want included in all comments.

Code templates are defined in the Code Templates Pane, and are supported for the following file types, used for actions and function libraries:

- .txt files
- .mts files
- .qfl files
- .vbs files

For details, see "[How to Use Code Snippets and Templates](#)" on page 635.

If you enter two characters that are the initial characters of multiple VBScript keywords, a drop-down menu appears with all of the relevant keywords, and you can select the one you want. For example, if you enter the letters `pr` and then enter a space, the drop-down menu is displayed, containing the keywords `preserve`, `private`, and `property`. You can then select a keyword from the list and press `ENTER` to insert it into your script.

## Searching and Replacing in the Editor

### **Relevant for: GUI actions, scripted GUI components, function libraries, and user code files**

When searching in GUI tests you can search in the Editor for text strings only. When searching API tests, you can search in the Editor for text strings, as well as references, derived classes, base classes, or overriding methods, for the current method, function or class.

**Note:** Search and replace functionality is not available in the Keyword View, the canvas, or an application area.

When performing text searches or replacements you can search throughout an entire solution, test, or folder (even if the file is currently closed). However, the specific file and item types searched within the solution, test, or folder are defined by the search algorithm and cannot be modified by users.

Search strings can be either standard text or regular expressions. For details on using regular expressions, see "[Regular Expressions in the Find and Replace Dialog Boxes](#)" on page 380.

**Note:** The search is limited to the text in the file at the time that the Find or Replace dialog box was opened. Any changes made after opening the dialog box are not included in the search.

When searching for text, you can use standard text or regular expressions in your search strings, and you can perform string replacements. You can also search in documents that are closed but accessible by the search functionality, either by searching an entire solution, or specifying a search folder.

The manner in which UFT searches differs for GUI tests and components and API tests:

### **Searches for strings in GUI tests and scripted components**

When you search for text strings in GUI tests, you can search in actions, function libraries, \*.vbs files, or \*.txt files. The search is performed in the action scripts, for each action defined in the test.

When you search in scripted components, the search is performed in any defined function libraries.

### **Searches for strings in API tests**

When you search for text strings in API tests, you can search in actions, \*.cs files, or \*.txt files. The search is performed in each source code module and in the test flow.

When you search in a specific C# source code file, the search is performed throughout all user code in the API test, as a single text file.

You can search for the following types of items in API tests:

- Activity or event display names or event handles
- Global environment variable display names and values
- Link expressions
- Loaded XML or schema files
- Test setting definitions
- Visible checkpoint or property display names and values
- X-paths

For task details on searching and replacing, see "[How to Search for References or Classes in Documents in the Editor \(API Testing Only\)](#)" on page 638.

## File and Item Types Included in String Searches

### **Relevant for: GUI actions, scripted GUI components, function libraries, and user code files**

When performing text searches or replacements you can search throughout an entire solution, test, or folder. However, the specific file and item types searched within the solution, test, or folder are defined by the search algorithm and cannot be modified by users.

**Note:** The search is limited to the text in the file at the time that the Find or Replace dialog box was opened. Any changes made after opening the dialog box are not included in the search.

### **Searches for strings in GUI tests and scripted components**

When you search for text strings in GUI tests, you can search in actions, function libraries, \*.vbs files, or \*.txt files. The search is performed in the action scripts, for each action defined in the test.

When you search in scripted components, the search is performed in any defined function libraries.

### **Searches for strings in API tests**

When you search for text strings in API tests, you can search in actions, \*.cs files, or \*.txt files. The search is performed in each source code module and in the test flow.

When you search in a specific C# source code file, the search is performed throughout all user code in the API test, as a single text file.

You can search for the following types of items in API tests:

- Activity or event display names or event handles
- Global environment variable display names and values
- Link expressions
- Loaded XML or schema files
- Test setting definitions
- Visible checkpoint or property display names and values
- X-paths

## How to Use Code Snippets and Templates

### **Relevant for: GUI actions, scripted GUI components, and function libraries**

This task describes how to insert pre-designed code snippets or blocks of text into your document, as well as how to manage templates for such snippets in the Code Templates Pane. For details on code snippets and templates, see "[Automatic Code Completion for GUI Testing](#)" on page 632.

This task includes the following steps:

- "Insert code snippets into your document in the Editor" below
- "Modify an existing list of code templates" below
- "Remove an existing list of code templates" on the next page
- "Add a new list of code templates" on the next page

## Insert code snippets into your document in the Editor

1. Create or open an action, scripted GUI component, or function library.  
(If you are editing a GUI action or scripted GUI component, open it in the Editor.)
2. Place your cursor at the point in the document that you want to insert the code snippet and then enter text as described in the following table.

If you enter:	Followed by:	UFT does the following:
The keyword for a code snippet listed in the <b>Edit &gt; Code Snippet</b> menu (GUI tests and scripted components)	<b>SPACE</b>	<p>Inserts the first VBScript snippet defined for the keyword you entered, as listed in the <b>Edit &gt; Code Snippet</b> menu.</p> <p><b>Note:</b> You can also insert these snippets by selecting them from the <b>Edit &gt; Code Snippet</b> menu.</p>
The keyword for a code template defined in the Options dialog box	<b>TAB</b>	<p>Inserts the code snippet defined in the relevant template in the Code Templates Pane.</p> <p><b>Note:</b> If multiple templates are defined for the keyword you entered, the first template in the list is inserted.</p>
Part of a code template keyword	<b>CTRL+SPACE</b>	<p>Displays a list of the code snippets and templates whose keywords match the characters you entered.</p> <p>The list includes the static VBScript snippets listed in the <b>Edit &gt; Code Snippet</b> menu, as well as the code templates defined for the relevant file type in the Code Templates Pane.</p> <p>In the drop-down list, browse to the keyword for the template you want to insert, and press Tab to insert its snippet into your document.</p>

## Modify an existing list of code templates

1. Open the Code Templates Pane.
2. From the **File Types** drop-down list, select the item associated with the list of templates you want to modify. The table lists all code templates defined for the selected file type or types.
3. To edit the file types associated with the selected list, click **Edit List**. In the Edit List dialog box, enter the file type or types that should be supported by the selected list, separated by semi-colons (;).

4. To edit the list of code templates, select the table row for the code template you want to modify and do one of the following:
  - **Add a new template in the list:** Click the empty space below the last row in the table, above the syntax area.
  - **Edit the template name:** Double-click the cell in the Template column, and update the name.
  - **Edit the keyword:** Double-click the cell in the Keyword column, and update the keyword. The keyword is the text that you enter in the Editor to insert the template.
  - **Edit the code template description:** Double-click the cell in the Description column and update the description text.
  - **Edit the code syntax inserted into your code:** Click inside the code syntax area below the table and update the code.

**Note:** Note that changes made here do not affect the code snippets available from the **Edit > Code Snippet** menu, which are static and cannot be modified.

## Remove an existing list of code templates

1. Open the Code Templates Pane.
2. From the **File Types** drop-down list, select the item associated with the list of templates you want to remove. The table lists all code templates defined for the selected file type or types.
3. Click **Remove List**. All code templates associated with the selected file types are removed, as well as the **File Types** item.

**Caution:** This action is irreversible.

## Add a new list of code templates

1. Open the Code Templates Pane.
2. Click **Add List**.
3. In the Add List dialog box, enter the file types you want to associate with your new list of templates, separated by semi-colons (:). A blank list is added to the table.
4. In each of the cells in the rows, enter text to add template names, keywords to be entered in the code, and descriptions of each template. In the syntax area below the table, enter the code template to be inserted in your code.
5. To add a new row in the list, click the empty space below the last row in the table, above the syntax area.

# How to Search for References or Classes in Documents in the Editor (API Testing Only)

## Relevant for: User code files

This task describes how to search for references to functions or method definitions, or base or descending classes, and includes the following steps:

- "Search for references to the currently selected function or method" below
- "Search for classes derived from the currently selected class" below
- "Search for methods that override a virtual method" below
- "Search for the base class of the current class" on the next page

## Search for references to the currently selected function or method

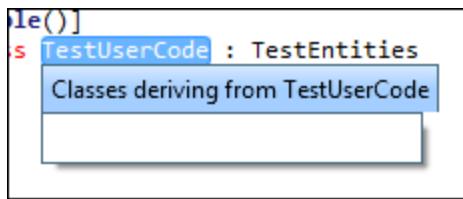
1. Create a new user code file, or open an existing one.
2. Select a function or method definition, and then select **Search > Find References**.

The search results found are displayed in the Search Results Pane.

## Search for classes derived from the currently selected class

1. Create a new user code file, or open an existing one. For details, see "[Add an event handler - optional](#)" on page 396 or "[How to Open a Window for Writing Custom Code](#)" on page 765.
2. Select a class and then select **Search > Find Derived Symbols**.

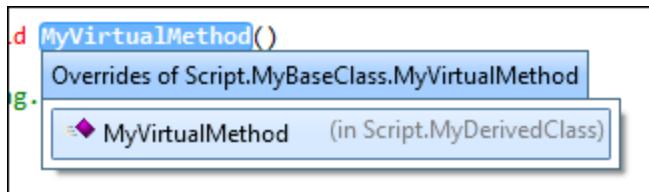
The derived classes are displayed in a small drop-down box under the selected class. For example:



## Search for methods that override a virtual method

1. Create a new user code file, or open an existing one. For details, see "[Add an event handler - optional](#)" on page 396 or "[How to Open a Window for Writing Custom Code](#)" on page 765.
2. Select a class and then select **Search > Find Derived Symbols**.

The overriding methods are displayed in a small drop-down box under the selected class. For example:



### Search for the base class of the current class

1. Create a new action or user code file, or open an existing one. For details, see "[Add an event handler - optional](#)" on page 396, "[How to Open a Window for Writing Custom Code](#)" on page 765, or "[How to Use Actions in an API Test](#)" on page 533.
2. Select a class and then select **Search > Find Base Classes**.

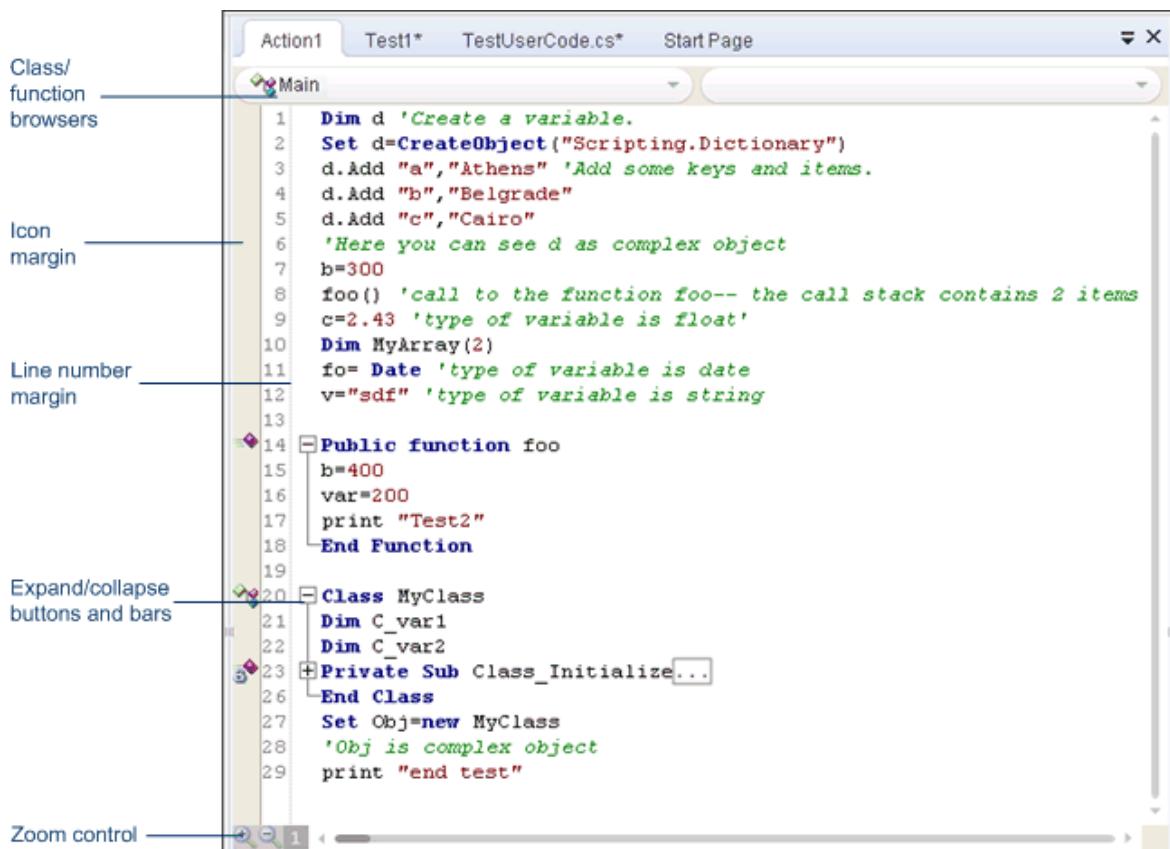
The base classes are displayed in a small drop-down box under the selected class. For example:

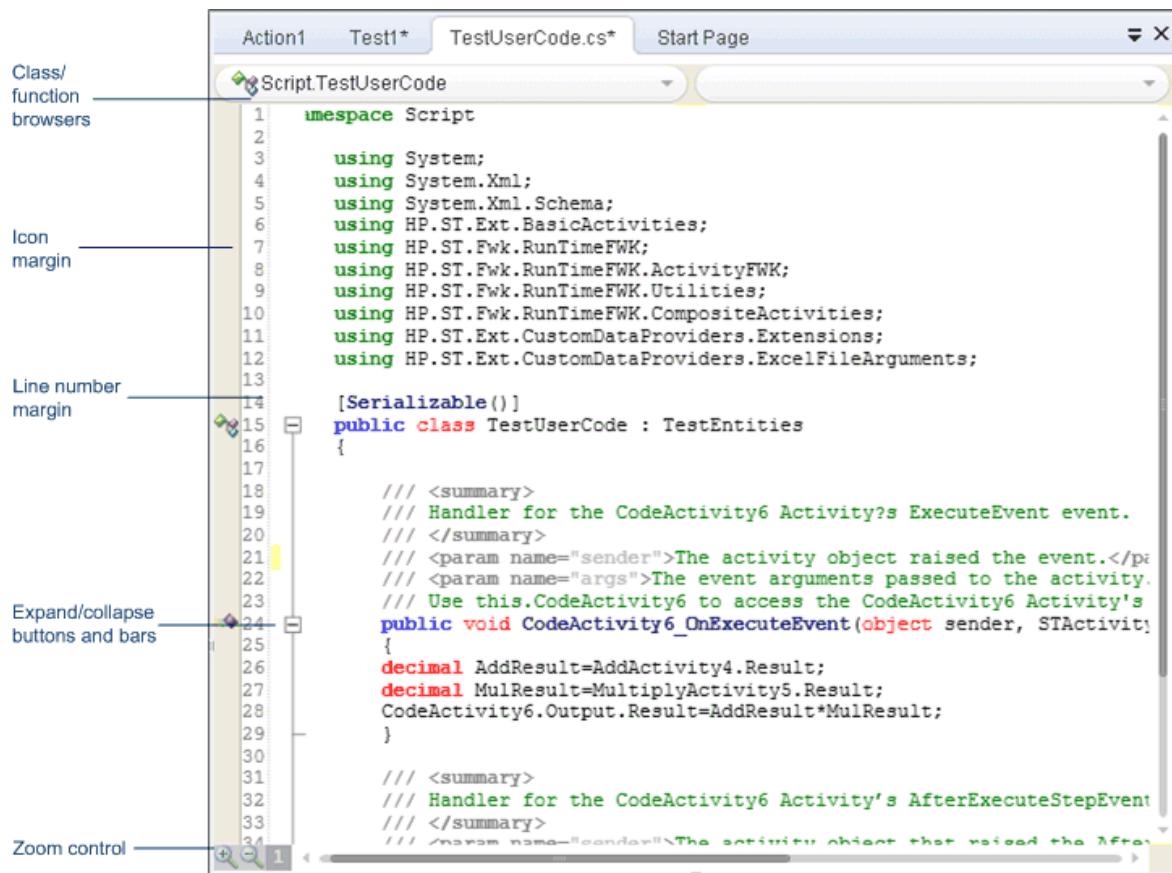


# Editor User Interface

**Relevant for: GUI actions, scripted GUI components, function libraries, and user code files**

This view enables you to edit text and code for GUI actions, scripted GUI components, function libraries, user code files, and text files.





<b>To access</b>	Create a new action, scripted GUI component, function library, or user code file, or open an existing one. (If you are editing a GUI action or scripted GUI component, open it in the Editor.)
<b>Important information</b>	<p>To open the help topic for a specific object or method from the Editor, do the following:</p> <ol style="list-style-type: none"> <li>Verify that you do not have any text highlighted.</li> <li>Place the cursor inside the word and press <b>F1</b>.</li> </ol> <p><b>Note:</b></p> <ul style="list-style-type: none"> <li>For many test objects this mechanism only works if the object exists in the object repository.</li> <li>You can also use these steps to find VBScript items referenced in the Microsoft VBScript help, included in the UFT Help.</li> <li>If you place the cursor in an item that cannot be located this way and press <b>F1</b>, the UFT Help opens and searches for the item.</li> </ul>
<b>Relevant tasks</b>	<ul style="list-style-type: none"> <li><a href="#">"How to Use Code Snippets and Templates" on page 635</a></li> <li><a href="#">"How to Search for References or Classes in Documents in the Editor (API Testing Only)" on page 638</a></li> </ul>
<b>See also</b>	<ul style="list-style-type: none"> <li><a href="#">"Editing Text and Code Documents" on page 627</a></li> <li>API tests only: <a href="#">"Writing Event Handlers for API Test Steps" on page 751</a></li> <li>GUI tests only: <a href="#">"Programming in GUI Testing Documents in the Editor" on page 647</a></li> </ul>

User interface elements are described below (unlabeled elements are shown in angle brackets):

UI Element	Description
<class and function browsers>	<p>Drop-down menus that enable you to quickly browse to a class and function by selecting from those available in the current document. These drop-down menus are most helpful in documents that include many function definitions, such as function libraries.</p>
	<p>To navigate to a class or function definition, first select the class from the left drop-down menu, and then select the function from the right drop-down menu.</p>
	<p><b>Note:</b> In a <b>GUI action</b>, the class browser:</p> <ul style="list-style-type: none"> <li>• Displays only those functions actually defined in the action, and does not display all functions defined in the function libraries associated with the test.</li> <li>• Always displays <b>Main</b> as the root menu item. In a document that includes function definitions, the functions will be listed as sub-items in the menu. In a document that includes no function definitions (such as most actions), this menu will include only the <b>Main</b> menu item. For example, see "<a href="#">Class and Function Browser Examples</a>" on the next page.</li> </ul>
	<p>If the class and function browsers are not displayed, you can select the <b>Show class/function browser</b> option in the General pane of the Text Editor pane (<b>Tools &gt; Options &gt; Text Editor tab &gt; General</b> node).</p>
<Icon margin>	<p>Display icons for lines with specific types of code, such as classes, events, or fields. For details, see "<a href="#">Editor Icons</a>" on page 644.</p>
<Line number margin>	<p>Displays the line numbers for each line of code. You can use the "Go To Dialog Box" to navigate to a specific line in the code.</p> <p>If the line numbers are not displayed, you can select the <b>Show line numbers</b> option in the General pane of the Text Editor pane (<b>Tools &gt; Options &gt; Text Editor tab &gt; General</b> node).</p>
⊕ ⊖	<p><b>&lt;Expand and collapse&gt;</b>. Expands or collapses folded code. Folded code is indicated by an ellipsis at the end of the first line of the folded code.</p> <p><b>Tip:</b> Hover over the ellipses to display a snapshot of the hidden code.</p> <p>If the expand and collapse buttons are not displayed, you can select the <b>Enable folding</b> option in the General pane of the Text Editor pane (<b>Tools &gt; Options &gt; Text Editor tab &gt; General</b> node).</p>
 1	<p><b>&lt;Zoom control bar&gt;</b>. Displayed in the lower left corner of the Editor, only after having zoomed in or out of the text for the first time in a session.</p> <ul style="list-style-type: none"> <li>• Click the <b>Zoom in</b>  icon to zoom in.</li> <li>• Click the <b>Zoom out</b>  icon to zoom out.</li> <li>• Click the <b>Default zoom</b>  icon to return the text to the default size.</li> </ul>

## Class and Function Browser Examples

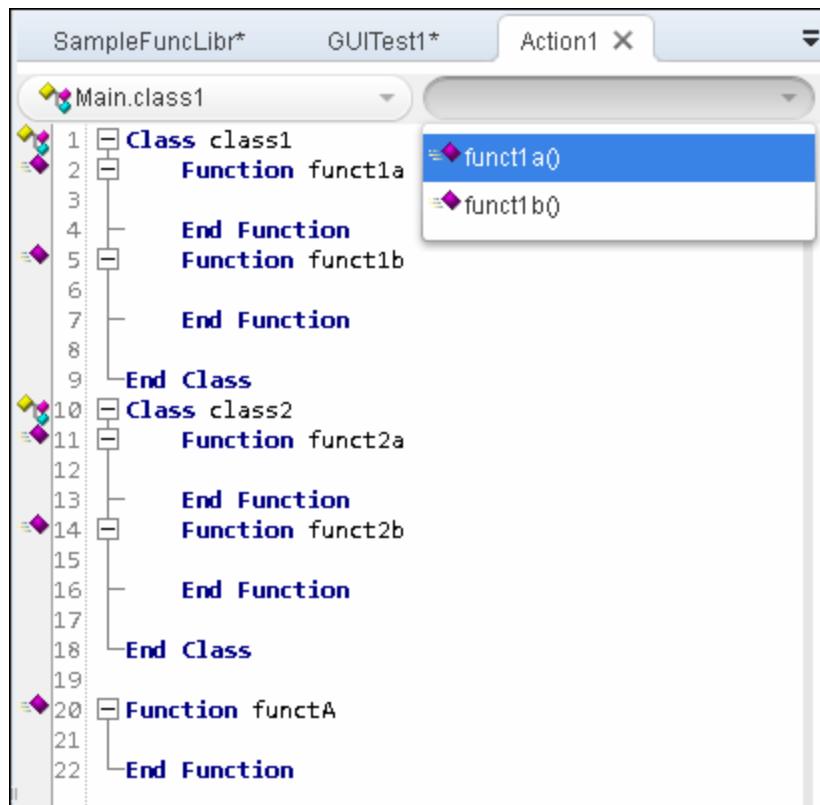
### Relevant for: GUI actions, scripted GUI components, function libraries, and user code files

The following image shows an example of an action containing two classes and a function. Each of the classes contains two functions.

The functions defined in a specific class are displayed in the function browser only when you select that class in the class browser. If you select one of these functions, the cursor jumps to that function definition. If you select a different class from the class browser, the cursor jumps to that class definition.

In this image, the **Main.class1** class is selected in the class browser, and the function browser is expanded, displaying the functions defined in the **class1** class (**fucnt\_1a** and **fucnt\_1b**).

In the class browser, the **Main** item represents the context of the document. In this image, if you select **Main** from the class browser, the function browser displays only **funct\_A**.



## Editor Icons

### **Relevant for: GUI actions, scripted GUI components, function libraries, and user code files**

Icons used in the Editor to indicate the different types of code are described below:

Type	Icon	Description
<b>Class</b>		Class
		Internal class
		Private class
		Protected class
<b>Delegate</b>		Delegate
		Internal delegate
		Private delegate
		Protected delegate
<b>Enumeration definition</b>		Enumeration definition
		Internal enumeration definition
		Private enumeration definition
		Protected enumeration definition
<b>Event</b>		Event
		Internal event
		Private event
		Protected event
<b>Extension method</b>		Extension method
		Internal extension method
		Private extension method
		Protected extension method

Type	Icon	Description
<b>Field</b>		Field
		Internal field
		Private field
		Protected field
<b>Indexer</b>		Indexer
		Internal indexer
		Private indexer
		Protected indexer
<b>Interface</b>		Interface
		Internal interface
		Private interface
		Protected interface
<b>Keyword</b>		Keyword
<b>Literal</b>		Literal
<b>Local</b>		Local
<b>Method</b>		Method
		Internal method
		Private method
		Protected method
<b>Namespace</b>		Namespace
<b>Operator</b>		Operator
<b>Parameter</b>		Parameter

Type	Icon	Description
<b>Property</b>		Property
		Internal property
		Private property
		Protected property
<b>Reference</b>		Reference
<b>Structure</b>		Structure
		Internal structure
		Private structure
		Protected structure

## Troubleshooting and Limitations - Statement Completion

### **Relevant for: GUI actions, scripted GUI components, and function libraries**

This section describes troubleshooting and limitations for the document pane.

Statement completion is not available for the following types of code:

- **Variables.** For example:

```
Set x = CreateObject("Application.Excel")
x
```

- **Class methods.** For example:

```
class fooClass
publicfunctionfoo
sin(45)
end function
End Class
Set x = New fooClass
x
```

- **Data tables and checkpoints.**

# Chapter 50: Programming in GUI Testing Documents in the Editor

**Relevant for: GUI tests, scripted GUI components, and function libraries**

This chapter includes:

• Programming in GUI Testing Documents in the Editor - Overview .....	649
• Programmatic Descriptions .....	649
• Static Programmatic Descriptions .....	651
• Dynamic Programmatic Descriptions .....	655
• Retrieving Child Objects .....	657
• Programmatic Description Checks .....	658
• Opening and Closing Applications Programmatically .....	660
• Comments, Control-Flow, and Other VBScript Statements .....	661
• Retrieving and Setting Identification Property Values .....	661
• Checkpoint and Output Statements .....	662
• Native Properties and Operations .....	663
• Running DOS Commands .....	664
• Filtering the GUI Steps Reported in the Run Session .....	664
• Using the Windows API in GUI Testing .....	665
• How to Enhance Your Actions, Scripted Components, and Function Libraries Using the Windows API	665
• Basic VBScript Syntax .....	667
• General VBScript Syntax Rules and Guidelines .....	668
• Handling VBScript Syntax Errors .....	669
• Formatting VBScript Text .....	670
• Comments in VBScript .....	671
• Parameter Indications in VBScript .....	672
• Parentheses in VBScript .....	673
• Calculations in VBScript .....	674
• Variables in VBScript .....	675
• Do...Loop Statement .....	676
• For...Each Statement .....	677

• <a href="#">For...Next Statement</a> .....	<a href="#">678</a>
• <a href="#">If...Then...Else Statement</a> .....	<a href="#">678</a>
• <a href="#">While...Wend Statement</a> .....	<a href="#">680</a>
• <a href="#">With Statement</a> .....	<a href="#">680</a>
• <a href="#">Report Modes</a> .....	<a href="#">681</a>

# Programming in GUI Testing Documents in the Editor - Overview

## **Relevant for: GUI actions, scripted GUI components, and function libraries**

GUI test actions and scripted GUI components are composed of statements coded in the Microsoft VBScript programming language. The Editor provides an alternative to the Keyword View for testers who are familiar with VBScript. In the Editor, you can view an action or scripted component in VBScript.

When working with GUI tests, and both scripted and keyword GUI components, you can also create and work with function libraries in UFT using VBScript. This enables you to increase the power and flexibility of your tests and components.

If you are familiar with VBScript, you can add and update statements and enhance your tests, components, and function libraries with programming. This enables you to increase a test or component's power and flexibility.

This chapter provides a brief introduction to VBScript, and shows you how to enhance your actions, scripted components, and function libraries using a few simple programming techniques.

**Note:** For details about using the text and code editor abilities available in the Editor, see "[Editing Text and Code Documents](#)" on page 627.

To learn about working with VBScript, you can view the VBScript documentation available from the UFT Help menu (**Help > HP Unified Functional Testing Help > VBScript Reference**).

You can add statements that perform operations on objects or retrieve information from your application. For example, you can add a step that checks that an object exists, or you can retrieve the return value of an operation.

You can add steps to your action, component, or function library either manually or using the Step Generator. For details on using the Step Generator, see "[How to Insert Steps Using the Step Generator](#)" on page 726.

## Programmatic Descriptions

## **Relevant for: GUI actions, scripted GUI components, and function libraries**

When UFT learns an object in your application, it adds the appropriate test object to the object repository. After the object exists in the object repository, you can add statements in the Editor to perform additional operations on that object. To add these statements, you usually enter the name (not case sensitive) of each of the objects in the object's hierarchy as the object description, and then add the appropriate operation.

Because each object in the object repository has a unique name, the object name is all you need to specify. During the run session, UFT finds the object in the object repository based on its name and parent objects, and uses the stored test object description for that test object to identify the object in your application.

You can also instruct UFT to perform operations on objects without referring to the object repository or to the object's name. To do this, you provide UFT with a list of properties and values that UFT can use to identify the object or objects on which you want to perform an operation. You can also describe an Insight test object, by providing a file containing an image of the control as a description property.

Such a **programmatic description** can be very useful in a number of scenarios:

<b>The object is not stored in the object repository</b>	Sometimes, your object may not be stored in the object repository, but still needs to be recognized during a test run. In this case, you can use descriptive programming to help UFT find this object in run-time by describing the object's properties instead of using the object name itself.
<b>A number of objects have common identical properties</b>	Normally, when UFT identifies an object, it uses the identification properties for that object to help find the object in the application. In some applications, the application objects have unique identification properties. However, in other applications, many objects may have the same identification properties, making it much more difficult for UFT to find the object in the application. In this case, you can substitute the object's properties using a description instead of relying upon the identification properties of the object stored in the object repository.
<b>The object is created dynamically during the run session</b>	In some applications, you have objects that are created dynamically depending on user input. In applications such as these, it is difficult or impossible to add these to an object repository as they do not "exist" in the application when UFT is learning it. Therefore, using programmatic descriptions to identify these objects in run time makes it possible for UFT to find and identify the objects in the application.
<b>The object differs between different versions of the application</b>	Especially when working with Web applications, objects have different properties depending on the browser in which the application is displayed. As a result, even if an object is added to the object repository for the application, UFT may have trouble identifying the object due to how each browser type renders the object. Using descriptive programming instead of static object identification properties makes your test objects more flexible in many different situations or browser versions, and enables UFT to find the object regardless of the environment in which the object is found.

### Use-case scenario:

Suppose you are testing a Web site that generates a list of potential employers based on biographical information you provide, and offers to send your resume to the employer names you select from the list. You want your test to select all the employers displayed in the list, but when you design your test, you do not know how many check boxes will be displayed on the page, and you cannot, of course, know the exact object description of each check box. In this situation, you can use a programmatic description to instruct UFT to perform a Set "ON" method for all objects that fit the description: `HTML TAG = input, TYPE = check box`.

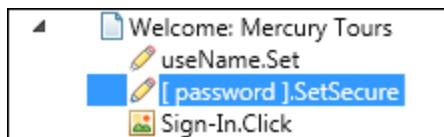
## Programmatic Description Types

There are two types of programmatic descriptions:

- **Static.** You list the set of properties and values that describe the object directly in a VBScript statement. For details, see "[Static Programmatic Descriptions](#)" below.
- **Dynamic.** You add a collection of properties and values to a Description object, and then enter the Description object name in the statement. For details, see "[Dynamic Programmatic Descriptions](#)" on page 655.

Using the **Static** type to enter programmatic descriptions directly into your statements may be easier for basic object description needs. However, in most cases, using the **Dynamic** type provides more power, efficiency, and flexibility.

In the Run Results, square brackets around a test object name indicate that the test object was created dynamically during the run session using a programmatic description or the **ChildObjects** method.



To learn more, see:

- [Static Programmatic Descriptions](#) ..... 651
  - [Guidelines for Using Static Programmatic Descriptions](#) ..... 652
- [Dynamic Programmatic Descriptions](#) ..... 655
- [Retrieving Child Objects](#) ..... 657
- [Programmatic Description Checks](#) ..... 658

## Static Programmatic Descriptions

### Relevant for: GUI actions, scripted GUI components, and function libraries

You can describe an object directly in a statement by specifying **property:=value** pairs describing the object instead of specifying an object's name.

The general syntax is:

```
TestObject("PropertyName1:=PropertyValue1", "...",
 "PropertyNameX:=PropertyValueX")
```

The method parts include:

<b>TestObject</b>	The test object class.
<b>PropertyName:=PropertyValue</b>	<p>The identification property and its value. Each <b>property:=value</b> pair should be separated by commas and quotation marks.</p> <p><b>Note:</b> To describe an Insight test object, specify the <b>ImgSrc</b> property, with the <b>PropertyValue</b> providing the file system path or ALM path to an image of the control. (To specify an ALM path to a file located in the ALM Test Resources module, type: [QualityCenter\Resources] Subject\&lt;folder and file name&gt;).</p>

The statement below specifies a WebEdit test object in the Mercury Tours page with the Name author and an index of 3. During the run session, UFT finds the WebEdit object with matching property values and enters the text Mark Twain.

```
Browser("Mercury Tours").Page("Mercury Tours").WebEdit
 ("Name:=Author", "Index:=3").Set "Mark Twain"
```

The statement below specifies an InsightObject test object in the Calculator window, with the image in the C:\AllMyFiles\Button6.bmp file. This file contains an image of the **6** button. During a run session, UFT finds the area on the calculator that looks like this image, and clicks its center.

```
Window("Calculator").InsightObject("ImgSrc:=C:\AllMyFiles\Button6.bmp").Click
```

For more details, see ["Using Programmatic Descriptions for Insight Test Objects" on page 655](#).

## Guidelines for Using Static Programmatic Descriptions

When working with static programmatic descriptions, be aware of the following guidelines:

- ["Regular Expressions in Programmatic Descriptions" on the next page](#)
- ["Variables in Programmatic Descriptions" on the next page](#)
- ["Programmatic Descriptions for Parent Test Objects" on the next page](#)
- ["Reuse of Static Programmatic Descriptions" on page 654](#)
- ["Copying Programmatic Description Data from the Object Spy" on page 654](#)
- ["Using Programmatic Descriptions for Insight Test Objects" on page 655](#)

## Regular Expressions in Programmatic Descriptions

UFT evaluates all property values in programmatic descriptions as regular expressions. Therefore, if you want to enter a value that contains a special regular expression character (such as \*, ?, or +), use the \ (backslash) character to instruct UFT to treat the special characters as literal characters. For details on regular expressions related to tests and scripted components, see ["Regular Expressions Overview" on page 372](#).

## Variables in Programmatic Descriptions

You can enter a variable name as the property value if you want to find an object based on property values you retrieve during a run session. For example:

```
MyVar="some text string"
Browser("Hello").Page("Hello").Webtable("table").Webedit("name:=" & MyVar)
```

## Programmatic Descriptions for Parent Test Objects

When using programmatic descriptions from a specific point within a test object hierarchy, you must continue to use programmatic descriptions from that point onward within the same statement. If you specify a test object by its object repository name after parent objects in the hierarchy have been specified using programmatic descriptions, UFT cannot identify the object.

### Examples:

- You can use the following statement, which uses object repository names for the parent objects and a programmatic description for the object on which the operation is performed:

```
Browser("Mercury Tours").Page("Mercury Tours").
 WebEdit("Name:=Author", "Index:=3").Set "Mark Twain"
```

- You can use the following statement since it uses programmatic descriptions throughout the entire test object hierarchy:

```
Browser("Title:=Mercury Tours").Page("Title:=Mercury Tours").
 WebEdit("Name:=Author", "Index:=3").Set "Mark Twain"
```

- You can also use the statement below, since it uses programmatic descriptions from a certain point in the description (starting from the Page object description):

```
Browser("Mercury Tours").Page("Title:=Mercury Tours").
 WebEdit("Name:=Author", "Index:=3").Set "Mark Twain"
```

- However, you cannot use the following statement, since it uses programmatic descriptions for the Browser and Page objects but then attempts to use an object repository name for the WebEdit test object:

```
Browser("Title:=Mercury Tours").Page("Title:=Mercury Tours").
 WebEdit("Author").Set "Mark Twain"
```

In this case, UFT tries to locate the WebEdit object based on its name, but cannot locate it in the repository because the parent objects were specified using programmatic descriptions.

For details on working with test objects, see "[Test Objects in Object Repositories](#)" on page 178.

## Reuse of Static Programmatic Descriptions

If you want to use the same static programmatic description several times in one action, scripted component, or function library, you may want to assign the object you create to a variable or use a **With** statement.

### Example

Instead of entering:

```
Window("Text:=Myfile.txt - Notepad").Move 50,
Window("Text:=Myfile.txt - Notepad").WinEdit("AttachedText:=Find what:").Set
"hello"
Window("Text:=Myfile.txt - Notepad").WinButton("Caption:=Find next").Click
```

You can enter:

```
Set MyWin = Window("Text:=Myfile.txt - Notepad")
MyWin.Move 50,
MyWin.WinEdit("AttachedText:=Find what:").Set "hello"
MyWin.WinButton("Caption:=Find next").Click
```

Alternatively, when working with actions or scripted components, you can use a **With** statement:

```
With Window("Text:=Myfile.txt - Notepad")
 .Move 50, 50
 .WinEdit("AttachedText:=Find what:").Set "hello"
 .WinButton("Caption:=Find next").Click
End With
```

For details on the **With** statement related to actions and scripted components, see "[With Statement](#)" on page 680.

## Copying Programmatic Description Data from the Object Spy

You can use the **Copy Identification Properties to Clipboard** option in the Object Spy Dialog Box to copy all identification properties and values for a selected object to the Windows Clipboard. The copied values are formatted in standard programmatic description syntax with line breaks between each property-value pair. For example:

```
"Class Name:=Image",
"abs_x:=585",
"abs_y:=573",
```

```
"alt:=Specials",
....
```

You can paste the copied data to any document and then copy selected lines (remove the line breaks) into a programmatic description.

## Using Programmatic Descriptions for Insight Test Objects

To describe an Insight test object, specify the **ImgSrc** property, with the **PropertyValue** providing the file system path or ALM path to an image of the control. (To specify an ALM path to a file located in the ALM Test Resources module, type: [QualityCenter\Resources] Subject\<folder and file name>).

When using programmatic descriptions for Insight test objects, consider the following:

- The description can contain only the **ImgSrc** property (mandatory) and ordinal identifier properties (optional).
- The description cannot contain regular expressions.
- The file containing the image of the control (specified in the **ImgSrc** property):
  - must be a non-compressed image file that supports 24 or 32 bits-per-pixel (JPEG, BMP or PNG).
  - must be accessible from any computer that runs the test or component.
- When running the **Click** method on an Insight test object defined using a programmatic description, UFT clicks in the center of the control that matches the specified image.

## Dynamic Programmatic Descriptions

### Relevant for: GUI actions, scripted GUI components, and function libraries

You can use the **Description** object to return a **Properties** collection object containing a set of **Property** objects. A **Property** object consists of a property name and value. You can then specify the returned **Properties** collection in place of an object name in a statement. (Each property object contains a property name and value pair.)

To create the **Properties** collection, you enter a **Description.Create** statement using the following syntax:

```
Set MyDescription = Description.Create()
```

After you have created a **Properties** object (such as `MyDescription` in the example above), you can enter statements to add, edit, remove, and retrieve properties and values to or from the **Properties** object during the run session. This enables you to determine which, and how many properties to include in the object description in a dynamic way during the run session.

After you fill the **Properties** collection with a set of **Property** objects (properties and values), you can specify the **Properties** object in place of an object name in a statement.

For example, instead of entering:

```
Window("Error").WinButton("text:=OK", "width:=50").Click
```

you can enter:

```
Set MyDescription = Description.Create()
MyDescription("text").Value = "OK"
MyDescription("width").Value = 50
Window("Error").WinButton(MyDescription).Click
```

When working with **Properties** objects, you can use variable names for the properties or values to generate the object description based on properties and values you retrieve during a run session.

You can create several **Properties** objects if you want to use programmatic descriptions for several objects.

For details on the **Description** and **Properties** objects and their associated methods, see the **Utility Objects** section of the *HP UFT Object Model Reference for GUI Testing*.

## Considerations for Description Objects

- By default, the value of all **Property** objects added to a **Properties** collection are treated as regular expressions. Therefore, if you want to enter a value that contains a special regular expression character (such as \*, ?, +), use the \ (backslash) character to instruct UFT to treat the special characters as literal characters. For details on regular expressions, see "[Regular Expressions Overview](#)" on page 372.

You can set the **RegularExpression** property to False to specify a value as a literal value for a specific **Property** object in the collection. For details, see the **Utility Objects** section of the *HP UFT Object Model Reference for GUI Testing*.

- When using programmatic descriptions from a specific point within a test object hierarchy, you must continue to use programmatic descriptions from that point onward within the same statement. If you specify a test object by its object repository name after other objects in the hierarchy have been described using programmatic descriptions, UFT cannot identify the object.

For example, you can use `Browser(Desc1).Page(Desc1).Link(desc3)`, since it uses programmatic descriptions throughout the entire test object hierarchy.

You can also use `Browser("Index").Page(Desc1).Link(desc3)`, since it uses programmatic descriptions from a certain point in the description (starting from the Page object description).

However, you cannot use `Browser(Desc1).Page(Desc1).Link("Example1")`, since it uses programmatic descriptions for the Browser and Page objects but then attempts to use an object repository name for the Link test object (UFT tries to locate the Link object based on its name, but cannot locate it in the repository because the parent objects were specified using programmatic descriptions).

## Retrieving Child Objects

### Relevant for: GUI actions, scripted GUI components, and function libraries

You can use the **ChildObjects** method to retrieve all objects located inside a specified parent object, or only those child objects that fit a certain programmatic description. To retrieve this subset of child objects, you first create a description object, and then you add the set of properties and values that you want your child object collection to match using the **Description** object.

**Note:** You must use the **Description** object to create the programmatic description for the **ChildObjects** description argument. You cannot enter the programmatic description directly into the argument using the **property:=value** syntax.

After you build a description in your description object, use the following syntax to retrieve child objects that match the description:

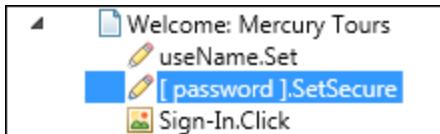
```
SetMySubSet=TestObject.ChildObjects(MyDescription)
```

### Example

The statements below instruct UFT to select all of the check boxes on the Itinerary Web page:

```
Set MyDescription = Description.Create()
MyDescription("html tag").Value = "INPUT"
MyDescription("type").Value = "checkbox"
Set Checkboxes = Browser("Itinerary").Page("Itinerary").ChildObjects
(MyDescription)
NoOfChildObjs = Checkboxes.Count
For Counter=0 to NoOfChildObjs-1
 Checkboxes(Counter).Set "ON"
Next
```

In the Run Results, square brackets around a test object name indicate that the test object was created dynamically during the run session using the **ChildObjects** method or a programmatic description.



For details on the **ChildObjects** method, see the **Common Methods and Properties** section in the *HP UFT Object Model Reference for GUI Testing*.

### Using the Index Property in Programmatic Descriptions

The index property can sometimes be a useful identification property for uniquely identifying an object.

The **index** identification property identifies an object based on the order in which it appears within the source code, where the first occurrence is 0.

Index property values are object-specific. Thus, if you use an index value of 3 to describe a WebEdit test object, UFT searches for the fourth WebEdit object in the page.

If you use an index value of 3 to describe a WebElement object, however, UFT searches for the fourth Web object on the page regardless of the type, because the WebElement object applies to all Web objects.

For example, suppose you have a page with the following objects:

- An image with the name Apple
- An image with the name UserName
- A WebEdit object with the name UserName
- An image with the name Password
- A WebEdit object with the name Password

The description below refers to the third item in the list above, which is the first WebEdit object on the page with the name UserName:

```
WebEdit("Name:=UserName", "Index:=0")
```

The following description, however, refers to the second item in the list above, which is the first object of any type (WebElement) with the name UserName:

```
WebElement("Name:=UserName", "Index:=0")
```

**Note:** If there is only one object, using index=0 does not retrieve it. You should not include the index property in the object description.

## Programmatic Description Checks

### Relevant for: GUI actions, scripted GUI components, and function libraries

You can compare the run-time value of a specified object property with the expected value of that property using either programmatic descriptions or user-defined functions.

Programmatic description checks are useful in cases in which you cannot apply a regular checkpoint, for example, if the object whose properties you want to check is not stored in an object repository. You can then write the results of the check to the run results.

For example, suppose you want to check the run-time value of a Web button. You can use the **GetROProperty** or **Exist** operations to retrieve the run-time value of an object or to verify whether the object exists at that point in the run session.

## Example

The following examples illustrate how to use programmatic descriptions to check whether the **Continue** Web button is disabled during a run session:

Using the **GetROProperty** operation:

```
ActualDisabledVal = Browser(micClass:="Browser").Page
(micClass:="Page").WebButton
 (alt:="Continue").GetROProperty("disabled")
```

Using the **Exist** operation:

```
While Not Browser(micClass:="Browser").Page(micClass:="Page").WebButton
 (alt:="Continue").Exist(30)
Wend
```

By adding **Report.ReportEvent** statements, you can instruct UFT to send the results of a check to the run results:

```
If ActualDisabledVal = True Then
Reporter.ReportEvent micPass, "CheckContinueButton = PASS", "The Continue button
is disabled, as expected."
Else
Reporter.ReportEvent micFail, "CheckContinueButton = FAIL", "The Continue button
is enabled, even though it should be disabled."
```

You can also create and use user-defined functions to check whether your application is functioning as expected. The following example illustrates a function that checks whether an object is disabled and returns **True** if the object is disabled:

```
'@Description Checks whether the specified test object is disabled
'@Documentation Check whether the <Test object name> <test object type> is
enabled.
Public Function VerifyDisabled (obj)
 Dim enable_property
 ' Get the disabled property from the test object
 enable_property = obj.GetROProperty("disabled")
 If enable_property = 1 Then ' The value is True (1)—the object is disabled
 Reporter.ReportEvent micPass, "VerifyDisabled Succeeded", "The test object
is disabled, as expected."
 VerifyDisabled = True
 Else
 Reporter.ReportEvent micFail, "VerifyDisabled Failed", "The test object is
enabled, although it should be disabled."
 VerifyDisabled = False
 End If
End Function
```

**Note:** For details on using the **GetROProperty** operation, see "Retrieving Native Properties" on page 663. For details on using **While...Wend** statements, see "While...Wend Statement" on page 680. For details on specific test objects, operations, and properties, see the *HP UFT Object Model Reference for GUI Testing*.

## Opening and Closing Applications Programmatically

### Relevant for: GUI actions and function libraries

In addition to using the Record and Run Settings Dialog Box (for tests) to instruct UFT to open a new application when a run session begins, or manually opening the application you want to test, you can insert statements into your test or component that open and close the applications you want to test.

You can run any application from a specified location using a `SystemUtil.Run` statement. You can add these statements directly in your GUI test action or in a function library.

You can specify an application and pass any supported parameters, or you can specify a file name and the associated application starts with the specified file open.

This is especially useful in the following situations:

- **For tests:** If your test includes more than one application, and you selected the **Record and run test on any application** check box in the Record and Run Settings Dialog Box.
- **For components:** If you want to provide an operation (function) that opens an application from within a component.

You can close most applications using the `Close` method. You can also use `SystemUtil` statements to close applications.

For example, you could use the following statements to open a file named `type.txt` in the default text application (Notepad), type `happy days`, save the file using shortcut keys, and then close the application:

```
SystemUtil.Run "C:\type.txt", "", "", ""
Window("Text:=type.txt - Notepad").Type "happy days"
Window("Text:=type.txt - Notepad").Type micAltDwn & "F" & micAltUp
Window("Text:=type.txt - Notepad").Type micLShiftDwn & "S" & micLShiftUp
Window("Text:=type.txt - Notepad").Close
```

### Note:

- When you specify an application to open using the Record and Run Settings Dialog Box (for tests only), UFT does not add a `SystemUtil.Run` statement to your test.
- The `InvokeApplication` method can open only executable files and is used primarily for backward compatibility.

For more details, see the *HP UFT Object Model Reference for GUI Testing*.

## Comments, Control-Flow, and Other VBScript Statements

### **Relevant for: GUI tests and components**

UFT enables you to incorporate decision-making into your test or component by adding conditional statements that control its logical flow. You can add the conditional statements directly in your action or component, or in the function libraries that they use.

In addition, you can define messages in your action or component that UFT sends to your run results.

To improve the readability of your actions and function libraries, you can also add comments to them.

For details on how to use these programming concepts in the Keyword View, see "[Keyword View](#)" on [page 109](#).

**Note:** The **VBScript Reference** (available from **Help > HP Unified Functional Testing Help**) contains Microsoft VBScript documentation, including VBScript, Script Runtime, and Windows Script Host.

For details, see:

- ["Comments in VBScript" on page 671](#)
- ["Calculations in VBScript" on page 674](#)
- ["For...Next Statement" on page 678](#)
- ["For...Each Statement" on page 677](#)
- ["Do...Loop Statement" on page 676](#)
- ["While...Wend Statement" on page 680](#)
- ["If...Then...Else Statement" on page 678](#)
- ["With Statement" on page 680](#)

## Retrieving and Setting Identification Property Values

### **Relevant for: GUI tests and components**

Identification properties are the set of properties defined by UFT for each object. You can set and retrieve a test object's identification property values, and you can retrieve the values of identification properties from a run-time object.

When you run your test or component, UFT creates a temporary version of the test object that is stored in the test object repository. You can use the `GetT0Property`, `GetT0Properties`, and `SetT0Property`

methods in your action, component, or function library to set and retrieve the identification property values of the test object.

- The **GetT0Property** and **GetT0Properties** methods enable you to retrieve a specific property value or all the properties and values that UFT uses to identify an object.
- The **SetT0Property** method enables you to modify a property value that UFT uses to identify an object.

**Note:** Because UFT refers to the temporary version of the test object during the run session, any changes you make using the **SetT0Property** method apply only during the course of the run session, and do not affect the values stored in the test object repository.

For example, the following statements would set the **Submit** button's name value to `my_button`, and then retrieve the value `my_button` to the `ButtonName` variable:

```
Browser("QA Home Page").Page("QA Home Page").WebButton
("Submit").SetT0Property "Name", "my_button"
ButtonName=Browser("QA Home Page").Page("QA Home Page").WebButton
("Submit").GetT0Property("Name")
```

- You use the **GetR0Property** method to retrieve the current value of an identification property from a run-time object in your application.

For example, you can retrieve the target value of a link during the run session as follows:

```
link_href = Browser("HP Technologies").Page("HP Technologies").Link
("Jobs").GetR0Property("href")
```

**Tip:** If you do not know the identification properties of objects in your application, you can view them using the Object Spy.

For a list and description of identification properties supported by each object, and for more details on the **GetR0Property**, **GetT0Property**, **GetT0Properties**, and **SetT0Property** methods, see the **Common Methods and Properties** section of the *HP UFT Object Model Reference for GUI Testing*.

## Checkpoint and Output Statements

### Relevant for: GUI actions and scripted GUI components

In UFT, you can create checkpoints and output values on pages, text strings, tables, and other objects. When you create a checkpoint or output value in the Keyword View, UFT creates a corresponding line in VBScript in the Editor. It uses the **Check** method to perform the checkpoint, and the **Output** method to perform the output value step.

For example, in the following statement UFT performs a check on the words New York:

```
Browser("Mercury Tours").Page("Flight Confirmation").Check Checkpoint("New York")
```

The corresponding step in the Keyword View is displayed as follows:

	Operation	Value	Documentation
 Flight Confirmation	Check	CheckPoint("New York")	Check whether text in the "Flight Confirmation:" Web page

**Note:**

- The details about a checkpoint are set in the relevant Checkpoint Properties dialog box. The details about an output value step are set in the relevant Output Value Properties dialog box. The statement displayed in the Editor is a reference to the stored information. Therefore, you cannot insert a checkpoint or output value statement in the Editor manually.
- For details on inserting and modifying checkpoints, see "[Checkpoints Overview](#)" on page 290. For details on inserting and modifying output values, see "[Output Values Overview](#)" on page 325.

## Native Properties and Operations

### Relevant for: GUI tests and components

If the test object operations and identification properties available for a particular test object do not provide the functionality you need, you can access the native operations and properties of any run-time object in your application using the **Object** property.

You can view the native properties and their values, or the identification properties, of the run-time object associated with an object in your application, in the Object Spy Dialog Box.

### Retrieving Native Properties

You can use the **Object** property to access the native properties of any run-time object. For example, you can retrieve the current value of the ActiveX calendar's internal **Day** property as follows:

```
Dim MyDay
Set MyDay=Browser("index").Page("Untitled").ActiveX
("MSCAL.Calendar.7").Object.Day
```

For more details on the **Object** property, see your object's properties in the *HP UFT Object Model Reference for GUI Testing*.

## Activating Native Operations

You can use the **Object** property to activate the internal operations of any run-time object. For example, you can activate the native **focus** method of the edit box as follows:

```
Dim MyWebEdit

Set MyWebEdit=Browser("Mercury Tours").Page("Mercury Tours").WebEdit
("username").Object

MyWebEdit.focus
```

For more details on the **Object** property, see your object's properties in the *HP UFT Object Model Reference for GUI Testing*.

You can use the statement completion feature with object properties to view a list of the available native operations and properties of an object.

**Tip:** If the object is a Web object, you can also reference its native properties in programmatic descriptions using the **attribute/property** notation. For details, see the *HP Unified Functional Testing Add-ins Guide*.

## Running DOS Commands

### Relevant for: GUI actions, scripted GUI components, and function libraries

You can run standard DOS commands in your action, scripted component, or function using the VBScript Windows Scripting Host Shell object (WScript.shell). For example, you can open a DOS command window, change the path to C:\, and run the DIR command using the following statements:

```
Dim oShell
Set oShell = CreateObject ("WScript.shell")
oShell.run "cmd /K CD C:\ & Dir"
Set oShell = Nothing
```

For more details, see the Microsoft VBScript Language Reference (select **Help > HP Unified Functional Testing Help > VBScript Reference > VBScript**).

## Filtering the GUI Steps Reported in the Run Session

### Relevant for: GUI actions, scripted GUI components, and function libraries

You can use the **Report.Filter** method to determine which steps or types of steps are included in the Run Results. You can completely disable or enable reporting of steps following the statement, or you

can indicate that you only want subsequent failed or failed and warning steps to be included in the report. You can also use the **Report.Filter** method to retrieve the current report mode.

For details, see "[Report Modes](#)" on page 681.

## Using the Windows API in GUI Testing

### **Relevant for: GUI actions, scripted GUI components, and function libraries**

Using the Windows API, you can extend testing abilities and add usability and flexibility to your tests components. The Windows operating system provides a large number of functions to help you control and manage Windows operations. You can use these functions to obtain additional functionality.

The Windows API is documented in the Microsoft MSDN Web site, which can be found at:

<http://msdn.microsoft.com/en-us/library/ee663266>

A reference to specific Windows API functions can be found at: <http://msdn2.microsoft.com/en-us/library/Aa383749>

For details on how to use the Windows API, see "[How to Enhance Your Actions, Scripted Components, and Function Libraries Using the Windows API](#)" below.

## How to Enhance Your Actions, Scripted Components, and Function Libraries Using the Windows API

### **Relevant for: GUI actions, scripted GUI components, and function libraries**

1. In MSDN, locate the function you want to use.
2. Read its documentation and understand all required parameters and return values.
3. Note the location of the Windows API function. Windows API functions are located inside Windows DLLs. The name of the .dll in which the requested function is located is usually identical to the Import Library section in the function's documentation. For example, if the documentation refers to User32.lib, the function is located in a .dll named User32.dll, typically located in your System32 library.
4. Use the UFT **Extern** object to declare an external function. For details, see the **Utility Objects** section of the *HP UFT Object Model Reference for GUI Testing*.

The following example declares a call to a function called **GetForegroundWindow**, located in user32.dll:

```
extern.declare micHwnd, "GetForegroundWindow", "user32.dll",
"GetForegroundWindow"
```

5. Call the declared function, passing any required arguments, for example:

```
hwnd = extern.GetForegroundWindow()
```

In this example, the foreground window's handle is retrieved. This can be useful if the foreground window is not in the object repository or cannot be determined beforehand (for example, a window with a dynamic title). You may want to use this handle as part of a programmatic description of the window, for example:

```
Window("HWND:=" &hwnd).Close
```

In some cases, you may have to use predefined constant values as function arguments. Since these constants are not defined in the context of your action, scripted component, or function, you need to find their numerical value to pass them to the called function. The numerical values of these constants are usually declared in the function's header file. A reference to header files can also be found in each function's documentation under the Header section. If you have Microsoft Visual Studio installed on your computer, you can typically find header files under X:\Program Files\Microsoft Visual Studio\VC98\Include.

For example, the GetWindow function expects to receive a numerical value that represents the relationship between the specified window and the window whose handle is to be retrieved. In the MSDN documentation, you can find the constants: GW\_CHILD, GW\_ENABLEDPOPUP, GW\_HWNDFIRST, GW\_HWNDLAST, GW\_HWNDNEXT, GW\_HWNDPREV and GW\_HWNDPREV.

If you open the WINUSER.H file, mentioned in the **GetWindow** documentation, you will find the following flag values:

```
/*
 * GetWindow() Constants
 */
#define GW_HWNDFIRST 0
#define GW_HWNDLAST 1
#define GW_HWNDNEXT 2
#define GW_HWNDPREV 3
#define GW_OWNER 4
#define GW_CHILD 5
#define GW_ENABLEDPOPUP 6
#define GW_MAX 6
```

## Example

The following example retrieves a specific menu item's value in the Notepad application:

```
' Constant Values:
const MF_BYPOSITION = 1024
```

```
' Windows API Functions Declarations
Extern.Declare micHwnd,"GetMenu","user32.dll","GetMenu",micHwnd
Extern.Declare
micInteger,"GetMenuItemCount","user32.dll","GetMenuItemCount",micHwnd
Extern.Declare micHwnd,"GetSubMenu","user32.dll","GetSubMenu",micHwnd,micInteger
Extern.Declare
micInteger,"GetMenuString","user32.dll","GetMenuString",micHwnd,micInteger,
micString+micByRef,micInteger,micInteger
' Notepad.exe
hwin = Window("Notepad").GetROProperty ("hwnd")' Get Window's handle
MsgBox hwin
' Use Windows API Functions
men_hwnd = Extern.GetMenu(hwin)' Get window's main menu's handle
MsgBox men_hwnd
item_cnt = Extern.GetMenuItemCount(men_hwnd)
MsgBox item_cnt
hSubm = Extern.GetSubMenu(men_hwnd,0)
MsgBox hSubm
rc = Extern.GetMenuString(hSubm,0,value,64,MF_BYPOSITION)
MsgBox value
```

## Basic VBScript Syntax

### **Relevant for: GUI actions, scripted GUI components, and function libraries**

You use VBScript in UFT to develop actions, scripted components, or function libraries in the Editor that you can use in your GUI tests and components.

VBScript is an easy-to-learn, yet powerful scripting language. You can use VBScript to develop scripts to perform both simple and complex object-based tasks, even if you have no previous programming experience.

This section provides some basic guidelines to help you use VBScript statements to enhance your action, scripted component, or function library. For more detailed information on using VBScript, you can view the VBScript documentation from the **UFTHelp** menu (**Help > HP Unified Functional Testing Help > VBScript Reference**).

Each VBScript statement has its own specific syntax rules. If you do not follow these rules, errors will be generated when you run the problematic step.

Additionally, if you try to move to the Keyword View from the Editor, UFT lists any syntax errors found in the document in the Errors pane. You cannot switch to the Keyword View without fixing or eliminating the syntax errors.

**Tip:** You can check the syntax of your documents at any time by selecting **Design > Check Syntax**. In the Errors pane, you can view all of the syntax errors in the whole solution, or only the ones found

in a selected test's actions and associated function libraries.

To learn more, see:

- [General VBScript Syntax Rules and Guidelines](#) ..... 668
- [Handling VBScript Syntax Errors](#) ..... 669
- [Formatting VBScript Text](#) ..... 670
- [Comments in VBScript](#) ..... 671
- [Parameter Indications in VBScript](#) ..... 672
- [Parentheses in VBScript](#) ..... 673
- [Calculations in VBScript](#) ..... 674
- [Variables in VBScript](#) ..... 675
- [Do...Loop Statement](#) ..... 676
- [For...Each Statement](#) ..... 677
- [For...Next Statement](#) ..... 678
- [If...Then...Else Statement](#) ..... 678
- [While...Wend Statement](#) ..... 680
- [With Statement](#) ..... 680

## General VBScript Syntax Rules and Guidelines

### Relevant for: GUI actions, scripted GUI components, and function libraries

When working with actions, scripted components, or function libraries in the Editor, you should consider the following general VBScript syntax rules and guidelines:

<b>Case-sensitivity</b>	<p>By default, VBScript is not case sensitive and does not differentiate between upper-case and lower-case spelling of words, for example, in variables, object and operation names, or constants.</p> <p>For example, the two statements below are identical in VBScript:</p> <pre>Browser("Mercury").Page("Find a Flight:").WebList("toDay").Select "31" browser("mercury").page("find a flight:").weblist("today").select "31"</pre>
<b>Text strings</b>	<p>When you enter a value as a text string, you must add quotation marks before and after the string. For example, in the above segment of script, the names of the Web site, Web page, and edit box are all text strings surrounded by quotation marks.</p> <p>Note that the value 31 is also surrounded by quotation marks because it is a text string that represents a number and not a numeric value.</p> <p>In the following example, only the property name (first argument) is a text string and is in quotation marks. The second argument (the value of the property) is a variable and therefore does not have quotation marks. The third</p>

	argument (specifying the timeout) is a numeric value, which also does not need quotation marks.
	<pre>Browser("Mercury").Page("Find a Flight:").WaitProperty("items count", Total_Items, 2000)</pre>
<b>Variables</b>	You can specify variables to store strings, integers, arrays and objects. Using variables helps to make your script more readable and flexible. For details, see " <a href="#">Variables in VBScript</a> " on page 675.
<b>Parentheses</b>	To achieve the desired result and to avoid errors, it is important that you use parentheses () correctly in your statements. For details, see " <a href="#">Parentheses in VBScript</a> " on page 673.
<b>Indentation.</b>	You can indent or outdent your script to reflect the logical structure and nesting of the statements. For details, see " <a href="#">Formatting VBScript Text</a> " on the next page.
<b>Comments.</b>	You can add comments to your statements using an apostrophe (), either at the beginning of a separate line, or at the end of a statement. It is recommended that you add comments wherever possible, to make your scripts easier to understand and maintain. For details, see " <a href="#">Formatting VBScript Text</a> " on the next page, and " <a href="#">Comments in VBScript</a> " on page 671.
<b>Spaces.</b>	You can add extra blank spaces to your script to improve clarity. These spaces are ignored by VBScript.
<b>Reserved Words.</b>	Certain words are reserved by UFT or VBScript. You cannot use these words as variables, constants, or procedure names. <ul style="list-style-type: none"> <li>Reserve words in UFT include the names of all UFT test object classes, methods, and properties, as well as F-keys (F1, F2, and so on).</li> <li>VBScript reserved words can be found in various online VBScript guides.</li> </ul> For example, a run error occurs if you try to run either of the following statements, which use a test object class name as a variable: <pre>Set Window = Window("Calculator") or WinButton = Window("Calculator").GetROProperty("hwnd")</pre> A run error also occurs when running the following statement because it uses a reserved F-key as a variable: <pre>Set F1 = CreateObject("Scripting.FileSystemObject")</pre>

For details on using specific VBScript statements to enhance your tests or components, see "[Comments, Control-Flow, and Other VBScript Statements](#)" on page 661.

## Handling VBScript Syntax Errors

### Relevant for: GUI actions, scripted GUI components, and function libraries

The Errors pane lists the syntax errors found in your document, and enables you to locate each syntax error so that you can correct it.

You can view a description of each of the VBScript errors in the VBScript Reference. For details, select **Help > HP Unified Functional Testing Help > VBScript Reference > VBScript > Reference > Errors > VBScript Syntax Errors.**

**Tip:** You can check the syntax of your documents at any time by selecting **Design > Check Syntax**.

In the Errors pane, you can view all of the syntax errors in the whole solution, or only the ones found in a selected test's actions and associated function libraries.

When you switch to the Keyword View from the Editor, UFT verifies the syntax. If a new or updated VBScript statement contains syntax errors, an error message is displayed in the Keyword View and the syntax error is displayed in the Errors pane. UFT is unable to display the document in the Keyword View until you have fixed all the syntax errors.

**Tip:** The Microsoft VBScript Language Reference defines VBScript syntax errors as: "errors that result when the structure of one of your VBScript statements violates one or more of the grammatical rules of the VBScript scripting language."

To learn about working with VBScript, you can view the VBScript Reference from the **UFT Help** menu (**Help > HP Unified Functional Testing Help > VBScript Reference**).

## Formatting VBScript Text

### **Relevant for: GUI actions, scripted GUI components, and function libraries**

When working with actions, scripted components, or function libraries in the Editor, it is important to follow accepted VBScript practices for comments and indentation.

#### **Comments**

Use comments to explain sections of an action, a scripted component, or a function library. This improves readability and makes your scripts easier to maintain and update. For details, see "[Comments in VBScript](#)" on the next page.

- **Adding Comments.** You can add comments to your statements by adding an apostrophe ('), either at the beginning of a separate line, or at the end of a statement.

**Tip:** You can comment a selected block of text by choosing **Edit > Format > Comment**. Each line in the block is preceded by an apostrophe.

- **Removing Comments.** You can remove comments from your statements by deleting the apostrophe ('), either at the beginning of a separate line, or at the end of a statement.

**Tip:** You can remove the comments from a selected block or line of text by choosing **Edit > Format > Uncomment**.

## Indentation

Use indentation to reflect the logical structure and nesting of your statements.

- **Indenting Statements.** You can indent your statements by selecting the text and choosing **Edit > Format > Indent**. If you want to indent multiple lines, you can also select the text and press the TAB key.

The text is indented according to the **Tab spacing** selected in the **General** pane of the Text Editor tab in the Options dialog box (**Tools > Options > Text Editor tab > General** node).

**Note:** The **Indent selected text when using the Tab key** check box must be selected in the Editor Options dialog box, otherwise pressing the TAB key deletes the selected text.

- **Outdenting Statements.** You can outdent your statements by selecting **Edit > Format > Outdent** or by deleting the space at the beginning of the statements.

For more detailed information on formatting in VBScript, you can view the VBScript documentation from the **UFT Help** menu (**Help > HP Unified Functional Testing Help > VBScript Reference**).

## Comments in VBScript

### Relevant for: GUI actions, scripted GUI components, and function libraries

A comment is a line or part of a line in a script that is preceded by an apostrophe ('). During a run session, UFT does not process the comments. You can use comments to explain sections of an action, a scripted component, or a function library, to improve readability, and to make your scripts easier to maintain and update.

The following example shows how a comment describes the purpose of the statement below it:

```
'Sets the word "mercury" into the "username" edit box.
Browser("Mercury Tours").Page("Mercury Tours").WebEdit("username").Set "mercury"
```

By default, comments are displayed in green inside your VBScript code. You can customize the appearance of comments in the **Fonts and Colors** pane of the **Text Editor** tab in the Options dialog box (**Tools > Options > Text Editor tab > General** node).

### Tips:

- You can comment a block of text by selecting **Edit > Format > Comment**.
- To remove the comment, select **Edit > Format > Uncomment**.

**Note:** You can also add a comment line using the VBScript **Rem** statement. For details, see the Microsoft VBScript Language Reference (select **Help > HP Unified Functional Testing Help > VBScript Reference > VBScript**).

## Parameter Indications in VBScript

### Relevant for: GUI actions and scripted GUI components

You can use UFT to enhance your tests by parameterizing values. A **parameter** is a variable that is assigned a value from an external data source or generator.

When you create a parameter in the Keyword View, UFT creates a corresponding line in VBScript in the Editor.

For example, if you define the value of a method argument as a Data pane parameter, UFT retrieves the value from the Data pane using the following syntax:

```
Object_Hierarchy.Method DataTable (parameterID, sheetID)
```

Item	Description
<i>Object_Hierarchy</i>	The hierarchical definition of the test object, consisting of one or more objects separated by a dot.
<i>Method</i>	The name of the method that UFT executes on the parameterized object.
<i>DataTable</i>	The reserved object representing the data table.
<i>parameterID</i>	The name of the column in the data table from which to take the value.
<i>sheetID</i>	The name of the sheet in which the value is stored. If the parameter is a global parameter, <code>dtGlobalSheet</code> is the sheet ID.

### Example

Suppose you are creating a test for the Mercury Tours site, and you select San Francisco as your destination. The following statement would be inserted into an action in your test in the Editor:

```
Browser("Welcome: Mercury").Page("Find a Flight:").WebList("toPort").Select "San Francisco"
```

Now suppose you parameterize the destination value, and you create a **Destination** column in the Data pane. The previous statement would be modified to the following:

```
Browser("Welcome: Mercury").Page("Find a Flight:").WebList("toPort").Select
DataTable("Destination",dtGlobalSheet)
```

In this example, `Select` is the method name, `DataTable` is the object that represents the data table, `Destination` is the Data pane parameter (column name), and `dtGlobalSheet` indicates the Global sheet in the Data pane.

In the Keyword View, this step is displayed as follows:



For more details on using and defining parameter values, see ["Parameterizing Object Values" on page 336](#).

## Parentheses in VBScript

### Relevant for: GUI actions, scripted GUI components, and function libraries

When programming in VBScript, it is important that you follow the rules for using or not using parentheses **0** in your statements.

You must use parentheses around method arguments if you are calling a method that returns a value and you are using the return value.

For example, use parentheses around method arguments if you are returning a value to a variable, if you are using the method in an **If** statement, or if you are using the **Call** keyword to call an action or function. When working with actions, you also need to add parentheses around the name of a checkpoint if you want to retrieve its return value.

**Tip:** If you receive an **Expected end of statement** error message when running a step, it may indicate that you need to add parentheses around the arguments of the step's method.

## Example

Following are several examples showing when to use or not use parentheses.

The following example requires parentheses around the method arguments for the **ChildItem** method because it returns a value to a variable:

```
Set WebEditObj = Browser("Mercury Tours").Page("Method of Payment").WebTable
("FirstName").ChildItem (8, 2, "WebEdit", 0)
WebEditObj.Set "Example"
```

The following example requires parentheses around the method arguments because **Call** is being used:

```
Call RunAction("BookFlight", oneIteration)
```

or

```
Call MyFunction("Hello World")
```

...

...

The following example requires parentheses around the **WaitProperty** method arguments because the method is used in an **If** statement:

```
If Browser("index").Page("index").Link("All kinds of").WaitProperty("attribute/readyState",
"complete", 4)
Then
 Browser("index").Page("index").Link("All kinds of").Click
End If
```

The following example requires parentheses around the **Check** method arguments, since it returns the value of the checkpoint:

```
a = Browser("MyBrowser").Page("MyPage").Check (CheckPoint("MyProperty"))
```

The following example does not require parentheses around the **Click** method arguments because it does not return a value:

```
Browser("Mercury Tours").Page("Method of Payment").WebTable("FirstName").Click
3,4
```

## Calculations in VBScript

### **Relevant for: GUI actions, scripted GUI components, and function libraries**

You can write statements that perform simple calculations using mathematical operators. For example, you can use a multiplication operator to multiply the values displayed in two text boxes in your Web site. VBScript supports the following mathematical operators:

Operator	Description
+	addition
-	subtraction
-	negation (a negative number)
*	multiplication
/	division
^	exponent

In the following example, the multiplication operator is used to calculate the maximum luggage weight of the passengers at 100 pounds each:

```
'Retrieves the number of passengers from the edit box using the GetROProperty
method
passenger = Browser ("Mercury_Tours").Page ("Find_Flights").WebEdit
("numPassengers").GetROProperty("value")
'Multiplies the number of passengers by 100
```

```
weight = passenger * 100
'Inserts the maximum weight into a message box.
msgbox("The maximum weight for the party is "& weight &"pounds.")
```

## Variables in VBScript

### Relevant for: GUI actions, scripted GUI components, and function libraries

You can specify variables to store test objects or simple values in your action, scripted component, or function library. When using a variable for a test object, you can use the variable instead of the entire object hierarchy in other statements. Using variables in this way makes your statements easier to read and to maintain.

To specify a variable to store an object, use the **Set** statement, with the following syntax:

```
Set ObjectVar = ObjectHierarchy
```

In the example below, the **Set** statement specifies the variable **UserEditBox** to store the full **Browser.Page.WebEdit** object hierarchy for the **username** edit box. The **Set** method then enters the value John into the **username** edit box, using the **UserEditBox** variable:

```
Set UserEditBox = Browser("Mercury Tours").Page("Mercury Tours").WebEdit
("username")
UserEditBox.Set "John"
```

**Note:** Do not use the **Set** statement to specify a variable containing a simple value (such as a string or a number). The example below shows how to define a variable for a simple value:

```
MyVar = Browser("Mercury Tours").Page("Mercury Tours").WebEdit
("username").GetTOProperty("type")
```

You can also use the **Dim** statement to declare variables of other types, including strings, integers, and arrays. This statement is not mandatory, but you can use it to improve the structure of your action, scripted component, or function library.

## Examples

The examples below demonstrate using the **Dim** statement to declare a variable:

**In an action or scripted component** (using the **passengers** variable):

```
Dim passengers
passengers = Browser("Mercury Tours").Page("Find Flights").WebEdit
```

```
("numpassengers").GetROProperty("value")
```

**In a function libraries** (using the `actual_value` variable):

```
Dim actual_value
 ' Get the actual property value
 actual_value = obj.GetROProperty(PropertyName)
```

These variables can then be used in different statements within the action, scripted component, or function library.

## Do...Loop Statement

**Relevant for: GUI actions, scripted GUI components, and function libraries**

The **Do...Loop** statement instructs UFT to perform a statement or series of statements while a condition is true or until a condition becomes true. It has the following syntax:

```
Do [{while} {until} condition]
 statement
Loop
```

Item	Description
<i>condition</i>	A condition to be fulfilled.
<i>statement</i>	A statement or series of statements to be performed during the loop.

### Example

In the following example, UFT calculates the factorial value of the number of passengers using the **Do...Loop**:

```
passengers = Browser("Mercury Tours").Page("Find Flights").
 WebEdit("numPassengers").GetROProperty("value")
total = 1
i = 1
Dowhile i <= passengers
 total = total * i
 i = i + 1
Loop
MsgBox "!" & passengers & "=" & total
```

## For...Each Statement

### Relevant for: GUI actions, scripted GUI components, and function libraries

A **For...Each** loop instructs UFT to perform one or more statements for each element in an array or an object collection. It has the following syntax:

```
For Each item In array
 statement
Next
```

Item	Description
<i>item</i>	A variable representing the element in the array.
<i>array</i>	The name of the array.
<i>statement</i>	A statement, or series of statements, to be performed during the loop.

### Example

The following example uses a **For...Each** loop to display each of the values in an array:

```
MyArray = Array("one", "two", "three", "four", "five")
For Each element In MyArray
 msgbox element
Next
```

**Note:** During a run session, if a **For Each** statement iterates on the **ParameterDefinitions** collection, the run may fail if the collection was retrieved directly before using the **For Each** statement. To prevent this, use other VBScript loop statements, such as **For** or **While**.

## For...Next Statement

### Relevant for: GUI actions, scripted GUI components, and function libraries

A **For...Next** loop instructs UFT to perform one or more statements a specified number of times. It has the following syntax:

```
For counter = start to end [Step step]
 statement
Next
```

Item	Description
<i>counter</i>	The variable used as a counter for the number of iterations.
<i>start</i>	The start number of the counter.
<i>end</i>	The last number of the counter.
<i>step</i>	The number to increment at the end of each loop. <b>Default = 1.</b> <b>Optional.</b>
<i>statement</i>	A statement, or series of statements, to be performed during the loop.

### Example

In the following example, UFT calculates the factorial value of the number of passengers using the **For** statement:

```
passengers = Browser("Mercury Tours").Page("Find Flights").
 WebEdit("numPassengers").GetROProperty("value")
total = 1
For i=1 To passengers
 total = total * i
Next
MsgBox "!" & passengers & "=" & total
```

## If...Then...Else Statement

### Relevant for: GUI actions, scripted GUI components, and function libraries

The **If...Then...Else** statement instructs UFT to perform a statement or a series of statements based on specified conditions. If a condition is not fulfilled, the next **Elseif** condition or **Else** statement is examined. It has the following syntax:

```
If condition Then
```

```
 statement
ElseIf condition2 Then
 statement
Else
 statement
End If
```

Item	Description
<i>condition</i>	Condition to be fulfilled.
<i>statement</i>	Statement to be perform.

## Example

In the following example, if the number of passengers is fewer than four, UFT closes the browser:

```
passengers = Browser("Mercury Tours").Page("Find Flights").WebEdit
("numpassengers").GetROProperty("value")
If (passengers < 4) Then
 Browser("Mercury Tours").Close
Else
 Browser("Mercury Tours").Page("Find Flights").Image("continue").Click 69,5
End If
```

The following example uses **If**, **ElseIf**, and **Else** statements to check whether a value is equal to 1, 2, or a different value:

```
value = 2
If value = 1 Then
 msgbox "one"
ElseIf value = 2 Then
 msgbox "two"
Else
 msgbox "not one or two"
End If
```

## While...Wend Statement

### Relevant for: GUI actions, scripted GUI components, and function libraries

A **While...Wend** statement instructs UFT to perform a statement or series of statements while a condition is true. It has the following syntax:

```
While condition
 statement
Wend
```

Item	Description
<i>condition</i>	A condition to be fulfilled.
<i>statement</i>	A statement or series of statements to be executed during the loop.

### Example

In the following example, UFT performs a loop using the **While** statement while the number of passengers is fewer than ten. Within each loop, UFT increments the number of passengers by one:

```
passengers = Browser("Mercury Tours").Page("Find Flights").WebEdit
("numpassengers").GetROProperty("value")
While passengers < 10
 passengers = passengers + 1
Wend
msgbox("The number of passengers in the party is " & passengers)
```

## With Statement

### Relevant for: GUI actions and scripted GUI components

With statements make your script more concise and easier to read and write or edit by grouping consecutive statements with the same parent hierarchy.

In addition, using With statements might help your script run faster. When running a With statement, UFT identifies the object in the application before running the first statement, but does not re-identify it before running each statement.

On the other hand, this can affect the running of your test if the object referenced by the With statement is refreshed, redrawn, or changed in some way in the application while running the With statement. To instruct UFT to re-identify the object in the application before running the next statement, add a statement that calls the RefreshObject test object operation. For details on the **RefreshObject** method, see the **Common Methods and Properties** section of the *HP UFT Object Model Reference for GUI Testing*.

The With statement has the following syntax:

```
With object
 statements
End With
```

Item	Description
object	An object or a function that returns an object.
statements	One or more statements to be performed on an object.

## Example

You could replace this script:

```
Window("Flight Reservation").WinComboBox("Fly From:").Select "London"
Window("Flight Reservation").WinComboBox("Fly To:").Select "Los Angeles"
Window("Flight Reservation").WinButton("FLIGHT").Click
Window("Flight Reservation").Dialog("Flights Table").WinList("From").
 Select "19097 LON"
Window("Flight Reservation").Dialog("Flights Table").WinButton("OK").Click
```

with the following:

```
With Window("Flight Reservation")
 .WinComboBox("Fly From:").Select "London"
 .WinComboBox("Fly To:").Select "Los Angeles"
 .WinButton("FLIGHT").Click
 With .Dialog("Flights Table")
 .WinList("From").Select "19097 LON"
 .WinButton("OK").Click
 End With 'Dialog("Flights Table")
End With 'Window("Flight Reservation")
```

Note that entering With statements in the Editor does not affect the Keyword View in any way.

**Note:** In addition to entering With statements manually, you can also instruct UFT to automatically generate With statements as you record or to generate With statements for an existing test. For more details, see ["How to Generate With Statements for Your Test" on page 727](#).

## Report Modes

**Relevant for: GUI actions, scripted GUI components, and function libraries**

For details on using report modes, see ["Filtering the GUI Steps Reported in the Run Session" on page 664](#).

The following report modes are available:

Mode	Description
<b>0 or rfEnableAll</b>	All events are displayed in the Run Results. <b>Default.</b>
<b>1 or rfEnableErrorsAndWarnings</b>	Only events with a warning or fail status are displayed in the Run Results.
<b>2 or rfEnableErrorsOnly</b>	Only events with a fail status are displayed in the Run Results.
<b>3 or rfDisableAll</b>	No events are displayed in the Run Results.

To use report mode settings, you can enter the following:

<b>Disable reporting of subsequent steps</b>	Reporter.Filter = rfDisableAll
<b>Re-enable reporting of subsequent steps</b>	Reporter.Filter = rfEnableAll
<b>Instruct UFT to include only subsequent failed steps in the run results</b>	Reporter.Filter = rfEnableErrorsOnly
<b>Instruct UFT to include only subsequent failed or warning steps in the run results</b>	Reporter.Filter = rfEnableErrorsAndWarnings
<b>Retrieve the current report mode</b>	MyVar=Reporter.Filter

For more details, see the **Reporter** object in the **Utility Objects** section of the *HP UFT Object Model Reference for GUI Testing*.

# Chapter 51: User-Defined Functions and Function Libraries

## Relevant for GUI tests and components

**Note:** The terms function, method, and operation are used interchangeably in this chapter.

This chapter includes:

• <a href="#">Function Library Overview</a> .....	684
• <a href="#">Associated Function Libraries</a> .....	684
• <a href="#">Working with Associated Function Libraries in ALM</a> .....	685
• <a href="#">User-Defined Functions</a> .....	685
• <a href="#">User-Defined Function Storage and Access</a> .....	687
• <a href="#">Registered User-Defined Functions</a> .....	688
• <a href="#">Preparing the User-Defined Function for Registration</a> .....	689
• <a href="#">Registering User-Defined Functions as Test Object Methods</a> .....	689
• <a href="#">Unregistering User-Defined Test Object Methods</a> .....	690
• <a href="#">Running an Overriding User-Defined Test Object Method</a> .....	691
• <a href="#">Loading Function Libraries During a Run Session</a> .....	693
• <a href="#">How to Create and Manage Function Libraries</a> .....	693
• <a href="#">How to Edit a Function Library</a> .....	696
• <a href="#">How to Manage Function Library Associations</a> .....	697
• <a href="#">How to Create and Work with a User-Defined Function</a> .....	700
• <a href="#">How to Create and Register a User-Defined Function Using the Function Definition Generator</a> .....	703
• <a href="#">Function Definition Generator Dialog Box</a> .....	708
• <a href="#">Troubleshooting and Limitations - Function Libraries</a> .....	715

# Function Library Overview

## Relevant for: GUI tests and components

In addition to the test objects, methods, and built-in functions supported by the UFT Test Object Model, you can define your own function libraries containing VBScript functions, subroutines, statement, and so on, and then call their functions from your test or use their functions as operations in your test or component.

A **function library** is a separate document that contains Visual Basic script. Any text file written in standard VBScript syntax can be used as a function library.

Your function libraries can contain:

- **Function definitions (function signature and code).** You can call these functions from other functions, from actions in your test, or from your component. To call a function from a test or component, you must first associate the function library with the test or with the component's application area.
- **VBScript statements.** These are statements that are not contained within function definitions (for example, RegisterUserFunc statements). UFT runs all of these statements when it loads the function library.

At the beginning of a run session, UFT loads all of the function libraries associated with your test or with your component's application area. If you need to dynamically load a function library during the test run, you can also use the **LoadFunctionLibrary** statement.

For task details, see "[How to Create and Manage Function Libraries](#)" on page 693.

## Associated Function Libraries

## Relevant for: GUI tests and components

After you create your function libraries, you can associate them with your test or application area. This enables you to insert a call to a public function or subroutine in the associated function library from that test or any component associated with that application area.

At the beginning of a run session, UFT loads the function libraries associated with the test or application area, and can then access their functions during the run session. Public functions stored in function libraries can be called from any associated test or component, whereas private functions can be called only from within the same function library.

The order in the list of associated function libraries determines the order in which UFT searches for a function or subroutine that is called from a step in your test or component. If there are two functions or subroutines with the same name, UFT uses the first one it finds.

When UFT searches for the function, it searches from the bottom of the list upwards to find the function.

You can:

- Edit the list of associated function libraries for an existing test or application area
- Specify default function libraries for all new tests (tests only)

For details, see "[How to Manage Function Library Associations](#)" on page 697.

To use a function library without associating it to your test or application area, you can load the function library dynamically during a run session using the **LoadFunctionLibrary** statement. For details, see the **Utility Statements** section of the *HP UFT Object Model Reference for GUI Testing*.

If you dynamically load a function library during a run session, and a later step calls a function that has the same name as a function in an associated function library, the function in the dynamically loaded function library is used.

## Working with Associated Function Libraries in ALM

### **Relevant for: GUI tests and components**

You can associate a function library with a test, regardless of whether the function library is stored in the file system or your ALM project. However, if you are planning to use the function library in a business process test or if you are working with the Resources and Dependencies model, you must save it in your ALM project.

When working with ALM and associated function libraries, you must save the function library in the Test Resources module in your ALM project before you can associate it with the test or application area. You can add a new or existing function library to your ALM project.

If you add an existing function library from the file system to an ALM project, you are actually adding a copy of that file to the project. Therefore, if you later make modifications to either of these function libraries (in the file system or in your ALM project), the other function library remains unaffected.

A component accesses the functions that are associated with its application area. Therefore, any changes you make to a function library that is stored in your ALM project and associated with an application area may affect its associated components. When making changes to a function library that is stored in your ALM project and associated with an application area, consider the effect of the changes on the components that use this application area. In ALM, you can view the list of components using the application area in the Dependencies tab of the Business Components module.

## User-Defined Functions

### **Relevant for: GUI tests and components**

For tests or scripted components, if you have segments of code that you need to use several times in your tests, you may want to create a **user-defined function**. For keyword components, you can create user-defined functions to provide additional functionality.

You create the functions in VBScript. For details on using VBScript, see "[Handling VBScript Syntax Errors](#)" on page 669 and "[Basic VBScript Syntax](#)" on page 667.

A user-defined function encapsulates an activity (or a group of steps that require programming) into a keyword (also called an operation). By using user-defined functions, your tests or components are shorter, and easier to design, read, and maintain. You or a Subject Matter Expert can then call user-defined functions from an action or a component by inserting the relevant keywords (or operations) into that action or component.

**For tests:** Before you can call a function from an action, you must add the function definition to the action or to a function library that is associated with the action's test.

**For components:** Before you can call a function from a component, you must add the function definition to a function library that is associated with the component's application area.

There are different kinds of user-defined functions:

## Global Functions

A user-defined function is automatically defined as a global function. You can call global functions by typing them in the step or selecting them from the lists displayed in the following locations:

- The **Operation** box in the Step Generator, when the **Functions** category is selected (for function libraries)
- The **Operation** column in the Keyword View, when the **Operation** item is selected from the **Item** list
- The Editor, when using the statement completion feature

## Functions registered to test objects

You can also register a user-defined function as a method for a UFT test object class (type). A registered method can either override the functionality of an existing test object method for the duration of a run session, or be registered as a new method for a test object class. You can call the test object method by typing it in the step or selecting it from the list of operations available for the test object.

For more details, see "[Registered User-Defined Functions](#)" on page 688 and "[How to Create and Register a User-Defined Function Using the Function Definition Generator](#)" on page 703.

**Note:** When deciding the name for your function, consider the following:

- During run-time, UFT searches the function libraries for the specified function in the order in which they are listed in the Solution Explorer. This order determines the function library priority.

For tests, UFT searches for the specified function in the action **before** searching the function libraries.

If UFT finds more than one function that matches the function name in a specific action or function library, it uses the last function it finds in that action or function library.

If UFT finds two functions with the same name in two different function libraries, it uses the function from the function library that has the higher priority. To avoid confusion, it is recommended that you verify that within the resources associated with a test or application area, each function has a unique name.

- When you create a user-defined function, do not give it the same name as a built-in function (for example, **GetLastError**, **MsgBox**, or **Print**). Similarly, do not use VBScript registered words (for example, **cStr**, **F1**, **ESC**) for function names. Built-in functions take priority over user-defined functions, so if you call a user-defined function that has the same name as a built-in function, the built-in function is called instead. For a list of built-in functions, see the **Built-in functions** list in the Step Generator (**Design > Step Generator**).

For details on naming conventions for function names, see "[Troubleshooting - Naming Conventions](#)" on page 1144.

## User-Defined Function Storage and Access

### Relevant for: GUI tests and components

Using UFT, you can define and store your user-defined functions either in a function library (saved as a .qfl file, by default) or directly in an action within a test.

When you store a public function in a function library and associate the function library with a test or application area, the test or any component associated with that application area can call the public functions in that function library. For details, see "[Associated Function Libraries](#)" on page 684.

You can access the functions that are stored in an associated function library from the Step Generator (for function libraries), the **Operation** column in the Keyword View, or the Toolbox pane, or you can enter these functions manually in a function library or an action.

You can also define private functions and store them in a function library. **Private functions** are functions that can be called only by other functions within the same function library. This is useful if you need to reuse segments of code in your public functions.

By implementing user-defined functions in function libraries and associating them with your test or component (via the component's application area), you and other users can choose functions that perform complex operations, such as adding if/then statements and loops, or working with utility objects—without adding the code directly to the test or needing any programming knowledge. In addition, you save time and resources by implementing and using reusable functions.

### For tests:

- When you store a function in an action, it can be called only from within that action—the function cannot be called from any other action or test. This is useful if you do not want the function to be available outside of a specific action.
- If the same function name exists locally within your action and within an associated function library, the function defined in the action is used.

# Registered User-Defined Functions

## Relevant for: GUI tests and components

You can register a public user-defined function to a test object to instruct UFT to use your user-defined function as a method of a specified test object class for the duration of a test or component run, or until you unregister the method.

A registered method applies only to the run session in which you register it. All function registrations are cleared at the beginning of each run session.

When you register a function to a test object class, you can register the function as a new operation for the test object class, or you can choose to override the functionality of an existing operation. You can unregister the function to disable new operations or to return existing operations to their original UFT behavior.

**For tests:** If you call an external action that registers a method (and does not unregister it at the end of the action), the method registration remains in effect for the remainder of the test that called the action.

The availability of registered user-defined functions differs for tests and components:

## Availability of Registered User-Defined Functions (for tests)

After you register a function to a test object class, it can be called as a method of that test object class, in addition to being available as a global function.

UFT displays the function in the general **Operation** list in the Step Generator and in the list of operations available for the test object displayed in the following locations:

- The **Operation** box in the Step Generator, when a test object of the relevant class is selected.
- The **Operation** column in the Keyword View, when a test object of the relevant class is selected from the **Item** list.
- The Editor, when you type the name of a test object of the relevant class and use the statement completion feature.

When you register a function to a test object class, you can optionally define it as the default operation for that test object class. This instructs UFT to use the function as the test object operation by default, in the following situations:

- In the **Operation** column in the Keyword View, when you choose a test object of the relevant class in the **Item** list.
- In the **Operation** box in the Step Generator, when you choose a test object of the relevant class from the **Object** list.
- In the Editor, when you drag in a test object of the relevant class from the object repository.

## Availability of Registered User-Defined Functions (for components)

After you register a function to a test object class, it can be called as a method of that test object class, in addition to being available as a global function.

UFT therefore displays the function in the Keyword View **Operation** list when a test object of that class is selected from the **Item** list, as well as in the general **Operation** list in the Step Generator (for function libraries).

When you register a function to a test object class, you can optionally define it as the default operation for that test object class. This instructs UFT to display the function in the **Operation** column in the Keyword View, by default, when you or a Subject Matter Expert choose a test object from the relevant class in the **Item** list.

To learn more, see:

- [Preparing the User-Defined Function for Registration](#) ..... 689
- [Registering User-Defined Functions as Test Object Methods](#) ..... 689
- [Unregistering User-Defined Test Object Methods](#) ..... 690
- [Running an Overriding User-Defined Test Object Method](#) ..... 691

## Preparing the User-Defined Function for Registration

### Relevant for: GUI tests and components

When you run a statement containing a registered method, UFT sends the test object as the first argument. For this reason, your user-defined function must have at least one argument. Your user-defined function can have any number of arguments, or it can have only the test object argument.

If you register a user-defined function to override an existing test object method, then after the test object argument, the function must have the same number of arguments as the method it overrides.

**Tip:** You can use the **parent** identification property to retrieve the parent of the object represented by the first argument in your function. For example:

```
ParentObj = obj.GetROProperty("parent")
```

## Registering User-Defined Functions as Test Object Methods

### Relevant for: GUI tests and components

To register a user-defined function as a test object method, you enter a **RegisterUserFunc** statement in an action or function library. The **RegisterUserFunc** statement specifies the test object class, the name of your function, and the name of the test object method that should call your function.

In this statement, you can also instruct UFT to use the function as the default operation for the test object class.

You can register the same function to more than one test object class, using the same operation name for different test object classes, or different names.

After the **RegisterUserFunc** statement runs, your method becomes a recognized method of the specified test object class for the remainder of the run session, or until you unregister the method.

When UFT loads a function library it runs all the statements in the function library. Therefore, if the function you are registering is defined in a function library, it is recommended to include the **RegisterUserFunc** statement in the function library as well so that the method is immediately available for use in any test or component using that function library.

For task details, see "[Register the function to a test object class - optional](#)" on page 702.

## Unregistering User-Defined Test Object Methods

### **Relevant for: GUI tests and components**

When you register a method using a **RegisterUserFunc** statement, your method becomes a recognized method of the specified test object class for the remainder of the run session, or until you unregister the method.

If your method overrides a UFT method, unregistering the method resets the method to its normal behavior. Unregistering other methods removes them from the list of methods supported by the test object class.

For task details, see "[Unregister the function - optional](#)" on page 703.

In certain situations, you must pay special care when unregistering user-defined methods:

### **Unregistering functions for reusable actions**

Unregistering methods is especially important when a reusable action contains registered methods that override UFT methods. For example, if you do not unregister a method that uses a function defined directly within a called action, then the calling test will fail if the registered method is called again in a later action, because it will not be able to find the function definition.

If you register a method within a reusable action, you should unregister the method at the end of the action (and then re-register it at the beginning of the next action if necessary), so that tests calling your action are not affected by the method registration.

If the registered function was defined in a function library, then the calling test may succeed (assuming the function library is associated with the calling test). However, unexpected results may be produced as the author of the calling test may not realize that the called action contained a registered function, and therefore, may use the registered method in later actions, expecting normal UFT behavior.

## Unregistering Functions That Were Registered More Than Once

You can re-register the same method to use different user-defined functions without first unregistering the method. However, when you do unregister the method, it resets to its original UFT functionality (or is cleared completely if it was a new method), and not to the previous registration.

### Example:

Suppose you enter the following statements:

```
RegisterUserFunc "Link", "Click", "MyClick"
RegisterUserFunc "Link", "Click", "MyClick2"
UnRegisterUserFunc "Link", "Click"
```

After running the **UnRegisterUserFunc** statement, the **Click** method stops using the functionality defined in the **MyClick2** function, and returns to the original UFT **Click** functionality, and not to the functionality defined in the **MyClick** function.

## Running an Overriding User-Defined Test Object Method

### Relevant for: GUI tests and components

You can register a user-defined function to (temporarily) override the functionality of an existing test object method for a test object class.

When a user-defined function runs instead of the test object method it overrides, if it calls any overridden test object methods, the standard functionality of those methods is used.

When you call the user-defined function directly, if it calls any overridden test object methods, their overriding user-defined functions are used.

## Examples

The following scenarios demonstrate various situations that are affected by this functionality:

### A Registered User Function That Calls the Test Object Method It Overrides

Suppose you want to report the current value of a Web edit box to the run results before you set a new value for it. You can override the standard UFT Set method with a function that retrieves the current value of an edit box, reports that value to the run results, and then sets the new value of the edit box using the standard Set method.

The function (and its registering line) would look something like this:

```
Function MySet (obj, x)
 dim y
```

```
y = obj.GetROProperty("value")
Reporter.ReportEvent micDone, "previous value", y
obj.Set (x)
End Function
RegisterUserFunc "WebEdit", "Set", "MySet"
```

When a test or component step uses the WebEdit.Set method, the overriding MySet function runs, and in turn, calls the original UFT WebEdit Set method.

However, when a test or component step uses the MySet function, the function runs and calls the overridden WebEdit.Set method, running the MySet function once more. This time, MySet calls the original UFT WebEdit Set method.

## A Registered User Function That Calls a Test Object Method That Is Overridden by Another Function

Suppose you want to override the VbButton's standard Click method to always perform a double click. In addition, you want to override the standard UFT DblClick method with a function that retrieves the text of the button and reports it to the run results before double-clicking the button.

The functions (and their registering lines) would look something like this:

```
Function MyDblClick (obj, x, y, button)
 dim button_name
 button_name = obj.GetROProperty("text")
 Reporter.ReportEvent micDone, "Clicking", button_name
 obj.DblClick x, y, button
End Function
RegisterUserFunc "VbButton", "DblClick", "MyDblClick"
Function MyClick (obj, x, y, button)
 obj.DblClick x, y, button
End Function
RegisterUserFunc "VbButton", "Click", "MyClick"
```

When a test or component step uses the VbButton.Click method, the overriding MyClick function runs. In this situation, MyClick will then run the original UFT VbButton DblClick method.

When a test or component step uses the MyClick function, the function runs and calls the overridden VbButton.DblClick method, running MyDblClick. MyDblClick reports the button text to the run results and then calls the original UFT VbButton DblClick method.

To ensure that the MyClick function always runs the overridden behavior for DblClick method, you could call MyDblClick directly within MyClick.

# Loading Function Libraries During a Run Session

## Relevant for: GUI tests and components

If you decide not to associate a function library (any VBScript file) with a test, but do want to be able to call its functions, subroutines, and so forth, you can do so by loading the function library during the run session.

Similarly, if you want to call a function that is not stored in an action in your test or in an associated function library, store it in an independent VBScript file, and load that function library during the run session.

To load a function library during a run session, insert a **LoadFunctionLibrary** statement or **ExecuteFile** statement in your action, scripted component, or function library. When you run the test, this statement runs all global code in the specified function library, making all definitions in the file available for use.

For details on these statements, see the **Utility Objects** section of the *HP UFT Object Model Reference for GUI Testing*.

The following table describes the differences between using each of these statements:

LoadFunctionLibrary	ExecuteFile
<b>In a test:</b> After you run a <b>LoadFunctionLibrary</b> statement, the functions in the file are available to your entire test, until the end of the run session.	After you run an <b>ExecuteFile</b> statement, you can call the functions in the loaded file only within the scope of the calling action or component.
<b>In a component:</b> <b>LoadFunctionLibrary</b> works in the same way as <b>ExecuteFile</b> . After you run the statement, the functions in the file are available only within the scope of the calling component.	
<b>LoadFunctionLibrary</b> enables you to debug the functions in the function library during run-time.	You cannot debug a file that is called using an <b>ExecuteFile</b> statement, or any of the functions contained in the file. In addition, when debugging a test or component that contains an <b>ExecuteFile</b> statement, the execution marker may not be correctly displayed.

If you want functions in a function library (VBScript file) to always be available to your test or component, associate the function library with your test or application area. For details, see "[Associated Function Libraries](#)" on page 684.

# How to Create and Manage Function Libraries

## Relevant for: GUI tests and components

This task describes the different activities you can perform to manage function libraries in UFT. If a function is stored in a function library, you must associate the function library with a test or a component's application area before you can call the function from the test or component.

This task includes the following steps:

- "[Create a function library](#)" below
- "[Open a function library](#)" below
- "[Edit a function library](#)" on the next page
- "[Enable editing for a read only function library](#)" on the next page
- "[Debug a function in a function library](#)" on the next page
- "[Associate a function library with a test or an application area](#)" on the next page
- "[Load a function library dynamically during a run session](#)" on page 696

## Create a function library

Do one of the following:

- Click the **New** button down arrow and select **Function Library**. The Open dialog box opens, enabling you to specify the location to save the function library.

**Note:** A newly created function library is not automatically associated to the current test or component. Therefore, the new function library is not displayed in the Solution Explorer.

- Create a new function library file from the Application Area (for components only). This also associates the function library with the application area, displaying it in the Solution Explorer.
- Create a new function library file from the Test Resources module in ALM. For details, see the *HP Application Lifecycle Management User Guide*.
- Create VBScript function libraries outside of UFT in any editor and save them with a .qfl, .vbs, or .txt extension.

## Open a function library

Do one of the following:

- Click the **Open** button down arrow and select **Function Library**.
- If the function library was recently created or opened, select it from the **Recent Files** list in the **File** menu.
- If the function library is associated with the solution, open test, component, or application area, you can also open it as follows:
  - In the Solution Explorer, double-click the function library, or right-click the function library and select **Open Function Library**.
  - In the Toolbox pane, double-click the function library, or right-click the function library and select **Open Resource**.

You can choose to open a function library in edit mode or read-only mode.

**Tip:** To open a function library, you must have read or read-write permissions for the file.

## Edit a function library

For details, see "[How to Edit a Function Library](#)" on the next page.

### Enable editing for a read only function library

Right-click the function library tab in the document pane and select **Enable Editing**. You can now edit the function library.

**Note:**

- You cannot enable editing if the function library is locked by another user or checked in to an ALM project.
- During a debug session, all documents (such as tests or components and function libraries) are read-only. To edit a document during a debug session, you must first stop the debug session.

## Debug a function in a function library

1. Associate the function library with a test or component (for a component, you do this via its application area).
2. In your test or component, insert a call to a function defined in the function library.
3. Place a breakpoint at the beginning of the function in the function library.
4. Run the test or component. When the function is called, UFT pauses the run session at the beginning of the function, enabling you to debug it.

For details, see "[Debugging Tests and Components](#)" on page 887.

**Note:**

- During a debug session, all documents are read-only and cannot be edited. To edit a document during a debug session, you must first stop the debug session.
- You cannot debug a file that is called using an **ExecuteFile** statement, or any of the functions contained in the file. In addition, when debugging a test that contains an **ExecuteFile** statement, the execution marker may not be correctly displayed.

To debug a dynamically loaded function library, use a **LoadFunctionLibrary** statement to load it instead of an **ExecuteFile** statement.

## Associate a function library with a test or an application area

For details, see "[How to Manage Function Library Associations](#)" on page 697.

## Load a function library dynamically during a run session

Add a the **LoadFunctionLibrary** statement to your action, scripted component, or associated function library. For details, see the **Utility Objects** section of the *HP UFT Object Model Reference for GUI Testing* and "Loading Function Libraries During a Run Session" on page 693.

**Tip:** To include the same **LoadFunctionLibrary** statement in every action you create, you can add the statement to an action template. For details, see "[Create an action template](#)" on page 106.

# How to Edit a Function Library

### Relevant for: GUI tests and components

This task describes how to edit a function library using the UFT editing features that are available in the Editor.

**Note:** This task is part of a higher-level task. For details, see "[How to Create and Manage Function Libraries](#)" on page 693.

This task includes the following steps:

- "[Add steps manually](#)" below
- "[Add steps using the Step Generator \(tests and scripted components only\)](#)" below
- "[Drag and drop a function](#)" on the next page
- "[Check the syntax of the code in your function library](#)" on the next page

### Add steps manually

When writing the steps in your function library, you can use standard VBScript statements as well as any UFT reserved objects or test objects, and the methods available for these objects.

UFT applies the same editing settings to your function library as it does to content in the Editor of an action. For details, see "[Editing Text and Code Documents](#)" on page 627.

Statement completion functionality is available for all functions defined in your action or component and for public functions defined in associated function libraries.

**Note:** In function libraries, the statement completion feature does not enable you to view test object names or collections because function libraries are not directly associated with object repositories.

### Add steps using the Step Generator (tests and scripted components only)

The "[Step Generator Dialog Box](#)" (described on page 732) enables you to add steps that contain **reserved objects** (the objects that UFT supplies for enhancement purposes, such as utility objects),

VBScript functions (such as **MsgBox**), utility statements (such as **Wait**), and user-defined functions that are defined in the same function library.

## Drag and drop a function

You can drag and drop a function (or part of it) within the same document, or from one document to another.

### To drag and drop a function from one document to another:

1. Separate the tabbed documents into separate document panes by dragging one document by its tab. You can then optionally dock the document in another location in UFT.
2. Select the relevant lines and drag and drop them from one document to another.

## Check the syntax of the code in your function library

Select **Design > Check Syntax**. UFT checks the syntax of all the code in all open function libraries and in function libraries associated with your tests. This may include function definitions and other VBScript statements.

**Tip:** For details on using VBScript, see "Basic VBScript Syntax" on page 667.

# How to Manage Function Library Associations

### Relevant for: GUI tests and components

This task describes the different ways that you can associate a function library with a test or application area or modify existing associations.

**Note:** This task is part of a higher-level task. For details, see "How to Create and Manage Function Libraries" on page 693.

This task contains the following steps:

- "View the list of function libraries associated with a test or component" on the next page
- "Associate the currently active function library with a solution, test, or an application area" on the next page
- "Associate a function library with a test using the Test Settings dialog box (tests only)" on the next page
- "Associate a function library with a test using the Solution Explorer pane (tests only)" on page 699
- "Associate a function library with an application area using the Function Libraries pane (components only)" on page 699
- "Modify the priority of an associated function library" on page 699
- "Remove a function library association" on page 699
- "Specify default function libraries for all new tests (tests only)" on page 700

## View the list of function libraries associated with a test or component

Do one of the following:

- In the Solution Explorer pane, expand the **Function Libraries** node within the relevant test or component's node.
- Select the test or component whose associated function libraries you want to view, and then select **File > Settings > Resources**. The Resources pane of the Test/Business Component dialog box opens.

## Associate the currently active function library with a solution, test, or an application area

1. Make sure that the test or application area with which you want to associate the function library is included in your open solution.
2. Create or open a function library in UFT. For details, see "[How to Create and Manage Function Libraries](#)" on page 693.
3. Save the function library in the file system (for tests only) or in your ALM project.
4. In UFT, do one of the following:
  - right-click the function library document tab and select **Associate Library '<Function Library>' with '<Solution/Test/Application Area>'**.
  - right-click the test or application area name node in the Solution Explorer and select **Associate Function Library**.

## Associate a function library with a test using the Test Settings dialog box (tests only)

1. Create or open a test.

**Note:** You can access or create a test using a relative path. If you enter a relative path, UFT searches for the test in the folders listed in the Folders pane of the Options dialog box (**Tools > Options > GUI Testing tab > Folders** pane).

2. In the Test Settings dialog box (**File > Settings**), click the **Resources** node.
3. In the **Associated function libraries** list, click the **Add** button  , UFT displays a browse button enabling you to browse to a function library in the file system. If you are connected to an ALM project, UFT also adds [ALM] to the file path, indicating that you can browse to a function library either in your ALM project or in the file system.

**Tip:** If you want to add a file from your ALM project but are not connected to ALM, press and hold the **SHIFT** key and click the **Add** button  . UFT adds [ALM], and you can enter the path manually. If you do, make sure there is a space after [ALM]. For example: [ALM] Subject\Tests

Note that UFT searches ALM project folders only when you are connected to the corresponding ALM project.

4. Select the function library you want to associate with your test and click **Open**.

## Associate a function library with a test using the Solution Explorer pane (tests only)

In the Solution Explorer pane, do one of the following:

- Right click a **GUI<test name>** node and select **Associate Function Library**.
- Right-click the **Function Libraries** node within the relevant test's node in the tree and select **Associate Function Library**.

The Open dialog box opens.

The function library that you select is associated with the test and displayed as a node under the **Function Libraries** node in the tree.

**Note:** A test node contains the **Function Libraries** node only if at least one function library is associated with the test. If the relevant test does not contain a **Function Libraries** child node, use one of the other association methods described in this task to associate the first function library.

## Associate a function library with an application area using the Function Libraries pane (components only)

1. In UFT, open your application area and click the **Function Libraries** button  on the sidebar.
2. In the **Associated function libraries** list, click the **Add** button . UFT displays a browse button enabling you to browse to a function library in your ALM project.
3. Select the function library you want to associate with your application area and click **Open**.

## Modify the priority of an associated function library

For tests, do one of the following:

- In the Solution Explorer, expand the Function Libraries node for your test or application area, right-click the function library you want to prioritize and select **Move up** or **Move down**.
- In the list of associated function libraries in the Resources pane of the Test Settings dialog box (for tests) or the Function Libraries pane (for application areas), select the function library you want to prioritize and use the **Up** and **Down** arrows  .

## Remove a function library association

Do one of the following:

- In the Solution Explorer, expand the Function Libraries node for your test or application area, right-click the function library and select **Remove Function Library from List**, or select the function library

and press the **DELETE** key.

- In the list of associated function libraries in the Resources pane of the Test Settings dialog box (for tests) or the Function Libraries pane (for application areas), select the function library you want to remove and click the **Remove** button .
- Open an associated function library in UFT. For details, see "[How to Create and Manage Function Libraries](#)" on page 693.

Right-click the function library document tab and select **Dissociate Library '<Function Library>' from '<Test/Application Area>'**.

## Specify default function libraries for all new tests (tests only)

In the Resources pane of the Settings dialog box, create the list of associated function libraries that you want to use for every newly created test, and click the **Set as Default** button. (This does not affect existing tests.)

# How to Create and Work with a User-Defined Function

### Relevant for: GUI tests and components

This task describes how to create and work with a user-defined function.

This task includes the following steps:

- "[Prerequisites - Open the function library or test](#)" below
- "[Create the function](#)" on the next page
- "[Register the function to a test object class - optional](#)" on page 702
- "[Associate the function library with a test or application area](#)" on page 702
- "[Call the function](#)" on page 703
- "[Navigate to the function's definition - optional](#)" on page 703
- "[Unregister the function - optional](#)" on page 703

### 1. [Prerequisites - Open the function library or test](#)

- a. **For tests:** Determine whether you want to store the function in an action or in a function library.
  - If you insert the function in a function library, the function will be accessible to any associated test.
  - If you insert the function directly in an action in the Editor, it can be called only from within the specific action.
- b. Create a new function library or action, open an existing one, or click on the tab of an open function library or action to bring it into focus.

## 2. Create the function

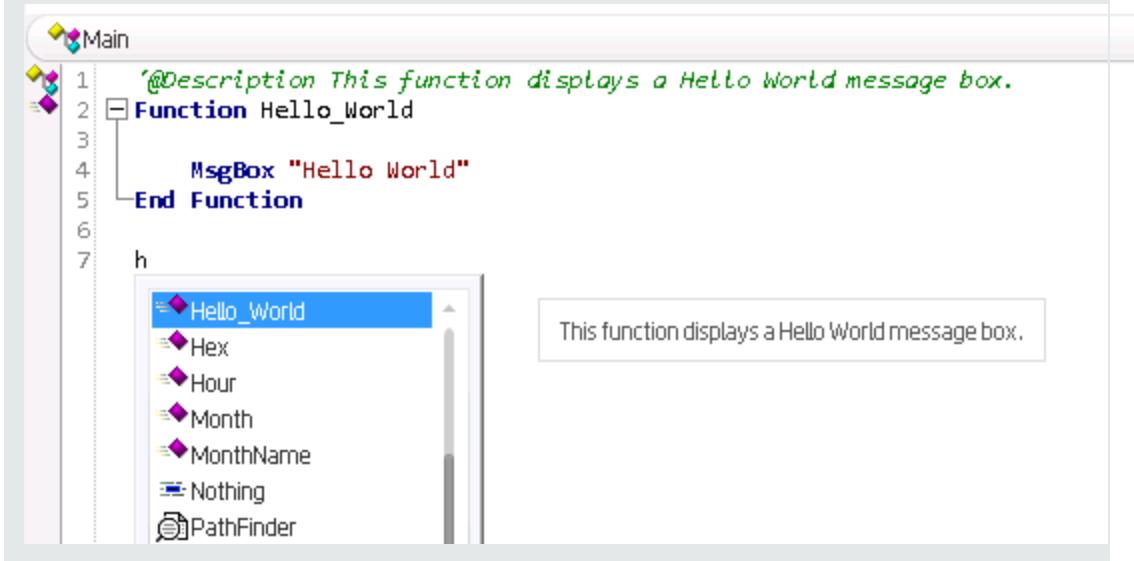
You can define functions manually or using the Function Definition Generator, which creates the basic function definition for you automatically.

Even if you prefer to define functions manually, you may still want to use the Function Definition Generator to view the syntax required to add header information, register a function to a test object class, or set the function as the default method for that test object class. For an in-depth view of the required syntax, you can define a function using the Function Definition Generator and experiment with the various options.

**Tip:** If you want to add a comment about your function, you can add a comment immediately above the function name with @description line and the string describing the function. This description is displayed as a custom tooltip in UFT's autocomplete window. For example

```
'@Description This function displays a Hello World message box.
Function Hello_World
 MsgBox "Hello world"
End Function
```

Then, when you use UFT's autocomplete menu, you can see the tooltip:



For details, see "[Function Definition Generator Dialog Box](#)" on page 708.

When writing the code for your function, consider the issues listed in "[Troubleshooting and Limitations - Function Libraries](#)" on page 715. For details on using VBScript, see the Microsoft VBScript Language Reference ([Help > HP Unified Functional Testing Help > VBScript Reference > VBScript](#)).

**Note:**

- If you want to register the function to a test object class, define it as a public function, and make sure that it expects the test object as the first argument.

- If you want to override an existing test object method, make sure that after the test object argument, your function accepts the same number of arguments as the method it overrides.

### 3. Register the function to a test object class - optional

You can register your function as a new method for the test object class, or you can register it using an existing method name to (temporarily) override the existing functionality of the specified method.

You can perform this step manually, or using the ["Function Definition Generator Dialog Box"](#) (described on page [708](#)):

- Add a **RegisterUserFunc** statement in your action or function library. The name of the test object operation you register cannot contain spaces. In this statement, you can also instruct UFT to use the function as the default operation for the test object class.

#### **Example:**

```
RegisterUserFunc "WebEdit", "MySet", "MySetFunc", True
```

After this statement runs (during the run session), the MySet method (operation) is added to the WebEdit test object class using the MySetFunc user-defined function, and defined to be the default operation (as specified in the last argument of the statement).

If you or the Subject Matter Expert choose the WebEdit test object from the **Item** list in the Keyword View, the MySet operation is selected automatically in the **Operation** column. It is also displayed in the **Operation** list together with other registered and out of the box operations for the WebEdit test object.

For syntax and other examples, see the **Utility Statements** section of the *HP UFT Object Model Reference for GUI Testing*, available from the Unified Functional Testing Help.

- If you use the ["Function Definition Generator Dialog Box"](#) (described on page [708](#)) to create your function definition, a RegisterUserFunc statement is automatically added immediately after the definition if you select the **Register to a test object** option.

You can add additional RegisterUserFunc statements manually to register the function to additional test object classes.

**Tip:** If the function you are registering is defined in a function library, it is recommended to include the RegisterUserFunc statement in the function library as well so that the method will be immediately available for use in any test or component using that function library.

### 4. Associate the function library with a test or application area

If you inserted the code in a function library, you must associate the function library with a test or

application area to enable tests and components to access to the user-defined functions. For details, see "[How to Manage Function Library Associations](#)" on page 697.

Alternatively, you can add a **LoadFunctionLibrary** statement to your test or component to load the function library during the run session and access its functions. For details, see the **Utility Statements** section of the *HP UFT Object Model Reference for GUI Testing*.

## 5. Call the function

In your test, component, or function library, do one or both of the following:

- Create steps that call your user-defined function as a global function.
- Run the user-defined function by calling the test object method to which it is registered.

## 6. Navigate to the function's definition - optional

You can navigate directly from a function call to the function's definition.

- a. In the Editor, in an action, click in the step containing the relevant function.

- b. Perform one of the following:

- Select **Search > Go to > Definition**.
- Right-click the step and select **Go to Definition** from the context menu.

UFT activates the relevant document (if the function definition is located in a function library) and positions the cursor at the beginning of the function definition.

## 7. Unregister the function - optional

If you do not want your function to remain registered until the end of the run session, add an **UnregisterUserFunc** statement in your test or function library. For syntax and an example, see the **Utility Statements** section of the *HP UFT Object Model Reference for GUI Testing*.

**For tests:** If you register a method within a reusable action, you should unregister the method at the end of the action (and then re-register it at the beginning of the next action if necessary), so that tests calling your action are not affected by the method registration.

# How to Create and Register a User-Defined Function Using the Function Definition Generator

## Relevant for: GUI tests and components

This task provides a general description of each step required to create a user-defined function and register it to a test object class using the "[Function Definition Generator Dialog Box](#)" (described on page 708). This task also includes examples for some of the steps.

**Note:** This task is part of a higher-level task. For details, see "[How to Create and Work with a User-Defined Function](#)" on page 700.

This task includes the following steps:

- "Prerequisite - Open the function library or action" below
- "Open the Function Definition Generator" below
- "Specify the details for the function definition" below
- "Register the function to a test object class - optional" below
- "Add arguments to the function - optional" on the next page
- "Add documentation details to the function - optional" on the next page
- "Preview the function definition before finalizing it" on page 706
- "Insert the function in your active document" on page 706
- "Add the content (code) of the function" on page 706
- "Results" on page 707

## 1. Prerequisite - Open the function library or action

Make sure that the function library or action in which you want to insert the function definition is the active document. (You can click the function library or action document's tab to bring it into focus.) This is because the Function Definition Generator inserts the function in the currently active document after you finish defining it.

## 2. Open the Function Definition Generator

Select **Design > Function Definition Generator**. The "Function Definition Generator Dialog Box" (described on page 708) opens.

## 3. Specify the details for the function definition

Specify the function's name, type, and scope in the "Function Definition Area" on page 710.

### Example:

If you want to define a function that verifies the value of a specified property, you might name it VerifyProperty and define it as a public function so that it can be called from any associated test or component. (If you define it as private, the function can be called only from elsewhere in the same function library. Private functions cannot be registered to a test object class.)

## 4. Register the function to a test object class - optional

If you defined a public function and you want to register the function as a test object operation, do the following in the "Registration Area" (described on page 711):

- a. Select the **Register to a test object** check box.
- b. Select the test object from the list of available objects.

**Example:**

For the sample **VerifyProperty** function, you might want to register it to the **Link** test object.

- c. Enter the name of a new operation that you want to add to the test object class, or select an existing operation to specify the operation that you want to override its standard functionality. The name of the method cannot contain spaces.

**Example:**

For the sample **VerifyProperty** function, you might want to define a new **VerifyProperty** operation.

- d. Optionally, specify that this operation is the default operation for test objects of this type.

**Tip:** If you choose not to register your function at this time, you can manually register it later by adding a **RegisterUserFunc** statement as described in "[Register the function to a test object class - optional](#)" on page 702. You can also add additional **RegisterUserFunc** statements, to register the function to additional test object classes.

## 5. Add arguments to the function - optional

Specify any arguments that are required for your function to run correctly. For details, see "[Arguments Area](#)" on page 712.

**Example:**

For the **VerifyProperty** function, registered to a test object class in the example for the previous step, you may want to assign the arguments `prop_name` (the name of the property to check) and `expected_value` (the expected value of the property), in addition to the first argument, `test_object`.

## 6. Add documentation details to the function - optional

Define **Description** and **Documentation** strings for the test object operation. For details, see "[Additional Information Area](#)" on page 713.

**Example:**

- For the sample **VerifyProperty** function, you may want to provide the following description:

Checks whether a property value matches the actual value.

- If you were checking a link to "HP" from a search engine, you might define the following documentation using the Function Definition Generator:

```
'@Documentation Check if the <Test object name> <Test object type> <prop_name> value matches the expected value: <expected_value>.
```

Additional information	
Description:	Checks if a property matches its expected value
Documentation:	Check if the <Test object name><Test object type><prop

In the Keyword View, after you create a step that calls the **VerifyProperty** operation and choose values for the arguments, the above documentation might be displayed as follows:

Check if the "Management Software" link "text" value matches the expected value: "Business Technology Optimization (BTO) Software".

## 7. Preview the function definition before finalizing it

As you add information to the Function Definition Generator, the **Preview** area displays the emerging function definition. You can review your function and make any changes, as needed, in the various areas of the dialog box. For details, see ["Bottom Area" on page 714](#).

### Example:

After defining the **VerifyProperty** function as described in the previous steps, the **Preview** area displays the following code:

```
'@Description Checks whether a property matches its expected value
'@Documentation Check whether the <Test object name> <Test object type>
<prop_name> value matches the expected value: <expected_value>.
Public Function VerifyProperty (test_object, prop_name, expected_value)
 'TODO: add function body here
End Function
RegisterUserFunc "Link", "VerifyProperty", "VerifyProperty"
```

## 8. Insert the function in your active document

- To generate an additional function definition after inserting this one, select **Insert another function definition**. The **OK** button changes to **Insert**.
- Click **OK** or **Insert**. UFT inserts the generated VBScript code in the active document.

## 9. Add the content (code) of the function

To finalize the function, add content to the function code, as required, replacing the TODO comment.

When writing the code for your function, consider the issues listed in ["Troubleshooting and Limitations - Function Libraries" on page 715](#). For details on using VBScript, see the Microsoft VBScript Language Reference (**Help > HP Unified Functional Testing Help > VBScript Reference > VBScript**).

**Example:**

For example, if you want the function to verify whether the expected value of a property matches the actual property value of a specific test object, you might add the following to the body of the function:

```
Dim actual_value
 ' Get the actual property value
 actual_value = obj.GetROProperty(prop_name)
 ' Compare the actual value to the expected value
 If actual_value = expected_value Then
 Reporter.ReportEvent micPass, "VerifyProperty Succeeded", "The " &
prop_name & " expected value: " & expected_value & " matches the actual
value"
 VerifyProperty = True
 Else
 Reporter.ReportEvent micFail, "VerifyProperty Failed", "The " &
prop_name & " expected value: " & expected_value & " does not match the
actual value: " & actual_value
 VerifyProperty = False
 End If
```

## 10. Results

- If you inserted the function in a function library, the function is now accessible to any associated test or component.

If you inserted the function directly in an action in the Editor, it can be called only from within the specific action.

To associate the function library with additional tests or application areas, see ["How to Manage Function Library Associations" on page 697](#).

- If you registered the function to a test object, the function is displayed in the list of available operations for the test object, as well as in the general **Operation** list in the Step Generator (for function libraries). For details on where you can view the list of operations available for a test object, see ["Registered User-Defined Functions" on page 688](#).

You can add additional **RegisterUserFunc** statements, to register the function to additional test object classes. For details, see ["Register the function to a test object class - optional" on page 702](#).

- If you specified that the function is the default operation for the test object, the function is used as the test object operation, by default, when you create steps with that test object. For details, see ["Registered User-Defined Functions" on page 688](#).

- If you defined a description and a **Documentation** string for the test object operation, they are displayed in the Keyword View and the Step Generator.

## Function Definition Generator Dialog Box

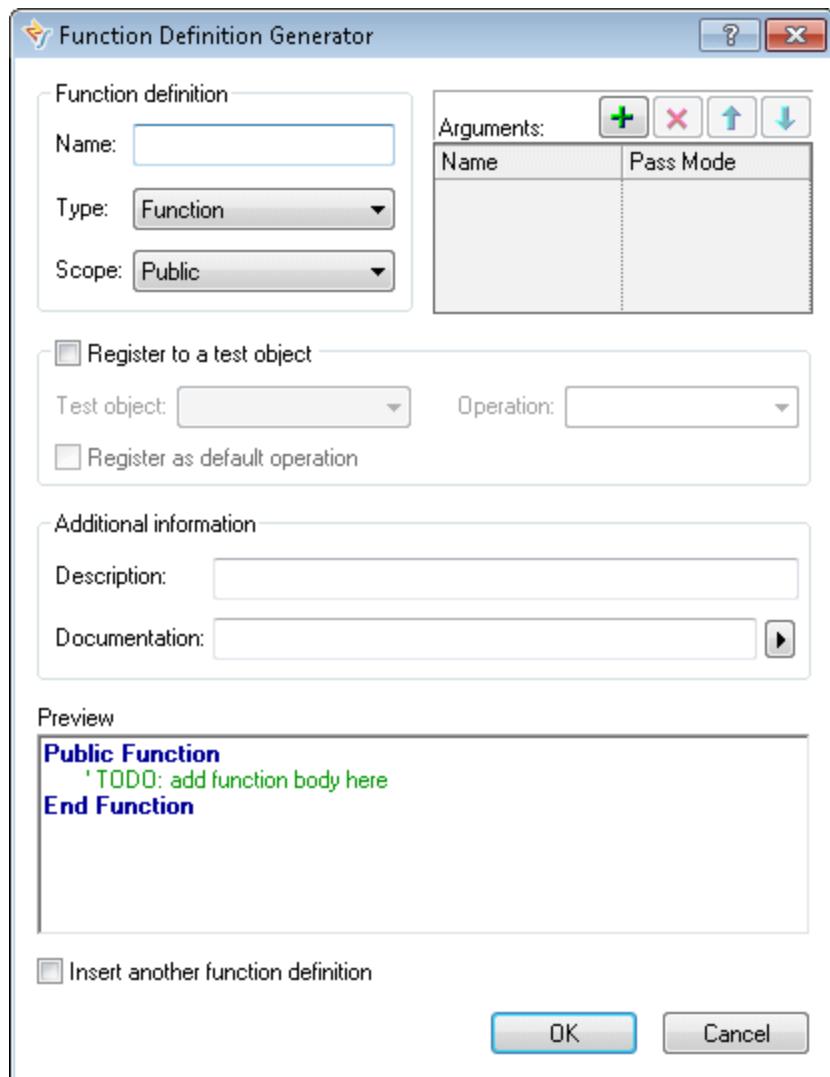
### **Relevant for: GUI tests and components**

This dialog box enables you to:

- Generate definitions for new user-defined functions.
- Add header information to the function you create.
- Register the functions to a test object class, if needed.

You fill in the required information and the Function Definition Generator creates the basic function definition for you. You complete the function by adding its content (code).

For a task that describes the order of working with this dialog box and provides examples, see ["How to Create and Register a User-Defined Function Using the Function Definition Generator" on page 703](#).

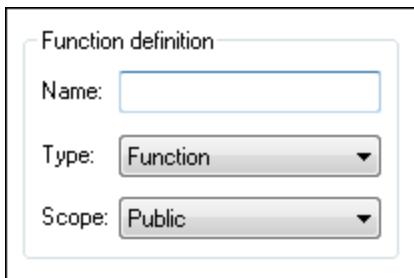


<b>To access</b>	Select Design > Function Definition Generator
<b>Important information</b>	Before you open the dialog box, make sure that the function library or action (displayed in the Editor) in which you want to insert the function definition is the active document. (You can click the function library or action document's tab to bring it into focus.) This is because the Function Definition Generator inserts the function in the currently active document when you click <b>OK</b> .
<b>Relevant tasks</b>	<ul style="list-style-type: none"> <li><a href="#">"How to Create and Register a User-Defined Function Using the Function Definition Generator" on page 703</a></li> <li><a href="#">"How to Create and Work with a User-Defined Function" on page 700</a></li> </ul>
<b>See also</b>	<ul style="list-style-type: none"> <li><a href="#">"User-Defined Functions" on page 685</a></li> <li><a href="#">"User-Defined Function Storage and Access" on page 687</a></li> <li><a href="#">"Registered User-Defined Functions" on page 688</a></li> </ul>

This dialog box contains the following key areas:

- "Function Definition Area" below
- "Registration Area" on the next page
- "Arguments Area" on page 712
- "Additional Information Area" on page 713
- "Bottom Area" on page 714

## Function Definition Area

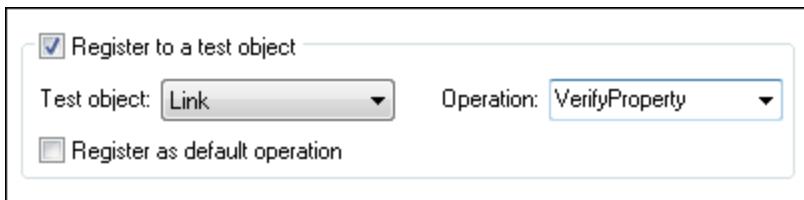


User interface elements are described below:

UI Elements	Description
<b>Name</b>	<p>A name for the new function.</p> <p>The name should clearly indicate what the operation does so that it can be easily selected from the Step Generator or in the Keyword View.</p> <p>For a list of naming conventions, see "<a href="#">Troubleshooting - Naming Conventions</a>" on page 1144.</p> <p><b>Note:</b></p> <p>Do not use any of the built-in function names (for example, <b>GetLastError</b>, <b>MsgBox</b>, or <b>Print</b>). For a list of built-in functions, see the <b>Built-in functions</b> list in the Step Generator (<b>Design &gt; Step Generator</b>).</p> <p>Try to give each function a name that is unique within the resources associated with a specific test or application area.</p> <p>For details, see "<a href="#">User-Defined Functions</a>" on page 685.</p>
<b>Type</b>	<p>The type of function.</p> <p><b>Possible values:</b></p> <ul style="list-style-type: none"> <li>• <b>Function</b></li> <li>• <b>Sub</b> (subroutine)</li> </ul>
<b>Scope</b>	<p>The scope of the function.</p> <p><b>Possible values:</b></p> <ul style="list-style-type: none"> <li>• <b>Public.</b> The function can be called by any test associated with this function library and any component whose application area is associated with this function library.</li> </ul>

UI Elements	Description
	<ul style="list-style-type: none"> <li>• <b>Private.</b> The function can be called only from elsewhere in the same function library.</li> </ul> <p><b>Default value:</b> Public</p> <p><b>Note:</b></p> <ul style="list-style-type: none"> <li>• Only public functions can be registered to a test object class.</li> <li>• If you create a user-defined function manually and do not define the scope as <b>Public</b> or <b>Private</b>, it is treated as a public function, by default.</li> </ul>

## Registration Area

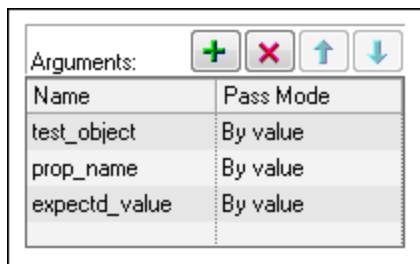


User interface elements are described below:

UI Elements	Description
<b>Register to a test object</b>	<p>Indicates whether to register this function as an operation for a UFT test object class.</p> <p>All user-defined functions are available as global operations. If you register the function to a test object class, it can be also be called by test objects of that class, and is displayed in the list of available operations for such test objects. For details, see "<a href="#">Registered User-Defined Functions</a>" on page 688.</p> <p><b>Note:</b> If you select this option, then when the Function Definition Generator creates your function definition, it also automatically adds a <b>RegisterUserFunc</b> statement with the correct argument values immediately after the definition.</p>
<b>Test object</b>	<p>The test object class (type) to which you want to register the function.</p> <p><b>Available:</b> When <b>Register to a test object</b> is selected</p>
<b>Operation</b>	<p>The operation name to use for this function.</p> <p>You can select an existing test object operation to override its functionality, or enter a name for a new operation to add to the test object class.</p> <p>Do not include spaces in the test object operation name.</p> <p><b>Available:</b> When <b>Register to a test object</b> is selected</p>
<b>Register as default operation</b>	<p>Indicates whether the function should be the default operation for the test object class.</p> <p>This instructs UFT to use the function as the test object operation, by default, when you create steps for test objects of the specified type. For details, see "<a href="#">Registered User-Defined Functions</a>" on page 688.</p>

UI Elements	Description
	<p><b>Note:</b> If you select this option, then when the Function Definition Generator creates your function definition, it specifies the value <b>True</b> as the fourth argument of the <b>RegisterUserFunc</b> statement.</p> <p><b>Available:</b> When <b>Register to a test object</b> is selected</p>

## Arguments Area



<b>Important information</b>	<p>When calling a function that is registered to a test object class, UFT passes the test object as the first argument. Therefore:</p> <ul style="list-style-type: none"> <li>If you select the <b>Register to a test object</b> check box, the Function Definition Generator automatically adds the argument, <b>test_object</b>, as the first argument in this area.</li> <li>If you clear the <b>Register to a test object</b> check box, the default <b>test_object</b> argument is automatically removed from this area (unless you renamed it).</li> </ul>
------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

User interface elements are described below:

UI Elements	Description
	<p>A toolbar that enables you to add, remove, or re-arrange arguments in the list. You can add as many arguments as you want to the list.</p> <p><b>Caution:</b></p> <ul style="list-style-type: none"> <li>If you are registering the function to a test object class, do not remove the <b>test_object</b> argument, change its location in the list, or modify its <b>Pass Mode</b>.</li> <li>If you are registering the function to override an existing test object method, then after the <b>test_object</b> argument, the function must have the same number of arguments as the method it overrides.</li> </ul>
<b>Name</b>	<p>The argument name.</p> <p>The name should clearly indicate the value that needs to be entered for the argument. For a list of naming conventions, see "<a href="#">"Troubleshooting - Naming Conventions" on page 1144</a>".</p>
<b>Pass Mode</b>	<p>Specifies whether the argument is passed to the function <b>By value</b> or <b>By reference</b> during run-time.</p>

## Additional Information Area

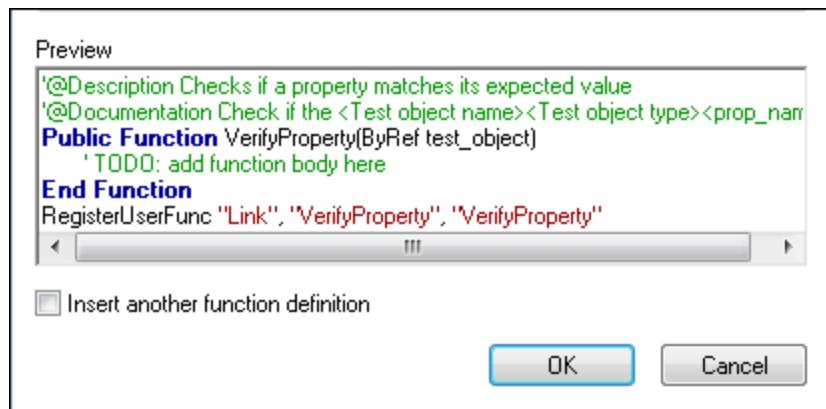
Additional information	
Description:	Checks if a property matches its expected value
Documentation:	Check if the <Test object name><Test object type><prop> ▶

The Function Definition Generator can add header information to your user-defined function definitions, based on the information you add in this area.

User interface elements are described below:

UI Elements	Description
<b>Description</b>	<p>The function's description.</p> <p>If you register the function as a test object operation, the description is displayed as a tooltip in UFT when the cursor is positioned over the operation in the Step Generator, in the Keyword View, and when using the statement completion feature.</p> <p>Keep the description text brief and clear.</p>
<b>Documentation</b>	<p>A sentence (in the imperative form) that specifies exactly what a step using your function does.</p> <p>If you register the function as a test object operation, the text that you add here is displayed in the <b>Step documentation</b> box of the Step Generator (tests only) and in the <b>Documentation</b> column in the Keyword View. Therefore, the sentence must be clear and understandable.</p>
▶	<p><b>Insert Documentation Element.</b> Displays a list that contains the function's arguments, and the items <b>test object name</b> and <b>test object type</b>, enabling you to include these items in the <b>Documentation</b> text.</p> <p>If you use the argument or test object items in the <b>Documentation</b> text, they are replaced dynamically by the relevant test object names and types or argument values when you create a step that uses the operation.</p> <p><b>Note:</b> The <b>Test object name</b> and <b>Test object type</b> items are available in the list only if you selected the <b>Register to a test object</b> check box.</p>

## Bottom Area



User interface elements are described below:

UI Elements	Description
<b>Preview</b>	<p>Displays the VBScript code that the Function Definition Generator will add to your active document, based on the information that you enter in the dialog box.</p> <p>The code is displayed in read-only format and includes:</p> <ul style="list-style-type: none"> <li>An empty function definition</li> <li>Header information for the function documentation (if defined)</li> <li>A <b>RegisterUserFunc</b> statement (if you selected <b>Register to a test object</b>)</li> </ul> <p>The function definition is displayed dynamically, as you enter the information. You can review your function and make any changes, as needed, in the various areas of the dialog box.</p>
<b>Insert another function definition</b>	<p>Specifies whether the dialog box should remain open after you click <b>OK</b>, enabling you to define an additional function.</p> <p>If you select this option, the <b>OK</b> button changes to <b>Insert</b>.</p>
<b>OK / Insert</b>	Inserts the generated VBScript code in the active document and clears the data from the dialog box.

# Troubleshooting and Limitations - Function Libraries

## Relevant for: GUI tests and components

This section describes troubleshooting and limitations for user-defined functions and function libraries.

- If you define a VBScript class, it can be called only within the UFT action or function library in which you defined it.

### Workaround:

- a. You can use an **ExecuteFile** statement to call a VBScript class defined in an external function library. However, you cannot debug a file that is called using an ExecuteFile statement, or any of the functions contained in the file. In addition, when debugging a test or component that contains an ExecuteFile statement, the execution marker may not be correctly displayed.
- b. In the function library in which you define the class, create a function that sets an object as your class, and returns the object. Then you can call this function from anywhere in your test or component, to assign an object of your class to a variable you define.

### Example:

Suppose you want to use the class `myClass`. In the same function library as you define `myClass`, define the following function:

```
public function myClassGenerator()
set myClassGenerator = new myClass
end function
```

Now, you can use the `myClass` class by calling `myClassGenerator`, like this:

```
set myClassObject = myClassGenerator()
```

`myClassGenerator()` creates a new `myClass` object and assigns it to `myClassObject`.

- You can use the **RegisterUserFunc** statement to register a user-defined function that overrides an existing test object method. You can also register a user-defined function to override a test object method that was created using a UFT Extensibility SDK. If you override this type of test object method, the user-defined function must not (recursively) call the test object method that it overrides.
- By default, steps that use user-defined functions are not included in the run results after a run session. If you want the function to appear in the run results, you must add a **ReportEvent Method** statement to the function code. For example, you may want to provide additional information or to modify the test, component, or business process test status. For details on the **Reporter.ReportEvent** statement, see the **Utility Objects** section of the *HP UFT Object Model Reference for GUI Testing*.

If a step within your user-defined function calls a standard UFT test object method, this step will appear in the run results after the run session. However, you can still add a **Reporter.ReportEvent**

statement to the function code to provide additional information and to modify the test, component, or business process test status, if required.

- **For tests:** If you use a partial run or debug option, such as **Run from step** or **Debug from step**, to begin running a test from a point after method registration was performed in a test step (and not in a function library), UFT does not recognize the method registration because it occurred prior to the beginning of the current run session.
- If you delete a function in use from an associated function library, the step using the function will display the  icon. In subsequent run sessions for the test, component, or business process test, an error will occur when the step using the non-existent function is reached.
- To use an **Option Explicit** statement in a function library associated with your test or component, you must include it in all of the function libraries associated with the test or component. If you include an **Option Explicit** statement in only some of the associated function libraries, UFT ignores all of the **Option Explicit** statements in all function libraries.

In test actions, you can use **Option Explicit** statements directly without any restrictions.

- Each function library must have unique variables in its global scope. If you have two associated function libraries that define the same variable in the global scope using a `Dim` statement or define two constants with the same name, the second definition causes a syntax error. If you need to use more than one variable with the same name in the global scope, include a `Dim` statement only in the last function library (since function libraries are loaded in the reverse order).
- Function libraries that run in the same run session must not contain different definitions for the same class. Make sure that each class is defined in only one location.
- If another user modifies a function library that is referenced by a test or component, or if you modify the function library using an external editor (not UFT), the changes take effect only after the test or component is reopened.
- Function libraries that contain the Cyrillic 'Я' character and are saved in ANSI encoding may be interpreted incorrectly in UFT. For example, the 'Я' character may be interpreted as a newline character, causing the run to fail.

**Workaround:** If this problem occurs, use a text editor to convert the function library by saving it in Unicode encoding.

# Chapter 52: Generated Programming Operations

**Relevant for: GUI tests and scripted GUI components**

This chapter includes:

• Programming Statements Overview .....	718
• Conditional Statements .....	718
• With Statements .....	720
• Message Statements .....	722
• Test Synchronization .....	723
• Synchronization Points .....	724
• Exist and Wait Statements .....	725
• Timeout Values Modification .....	725
• Step Generator .....	726
• How to Insert Steps Using the Step Generator .....	726
• How to Generate With Statements for Your Test .....	727
• Add Synchronization Point Dialog Box .....	730
• Step Generator Dialog Box .....	732
• Storage Location Options Dialog Box .....	738

# Programming Statements Overview

## Relevant for: GUI tests and scripted GUI components

When you design tests, you usually begin by adding steps that represent the operations an end-user would perform as part of the business process you want to test. Then, to increase the power and flexibility of your test, you can add steps (programming statements) that contain programming logic to the basic framework.

Programming statements can contain:

- **Test object operations.** These are methods and properties defined by UFT. They can be operations that a user can perform on an object, operations that can retrieve or set information, or operations that perform operations triggered by an event.
- **Native operations.** These are methods and properties defined within the object you are testing, and therefore are retrieved from the run-time object in the application.
- **VBScript programming commands.** These affect the way the test runs, such as [Conditional Statements](#) and [Synchronization Points](#). These are often used to control the logical flow of a test.
- **Comments.** Use comments to explain sections of your tests to improve readability and to make them easier to update. A comment is an explanatory remark in a program, and does not get processed when UFT runs.

To learn more, see:

- [Conditional Statements](#) ..... 718
- [With Statements](#) ..... 720
- [Message Statements](#) ..... 722
  - [Run session messages in the HTML report](#) ..... 722
  - [Run session messages in the Run Results Viewer](#) ..... 722
  - [Step messages in the Run Results Viewer](#) ..... 722
  - [Display messages during the run session](#) ..... 723

## Conditional Statements

### Relevant for: GUI tests and scripted GUI components

You can control the flow of your test with conditional statements. Using conditional **If...Then...Else** statements, you can incorporate decision-making into your tests.

The **If...Then...Else** statement is used to evaluate whether a condition is true or false and, depending on the result, to specify one or more statements to run. Usually the condition is an expression that uses a comparison operator to compare one value or variable with another. The following comparison

operators are available: less than <, less than or equal to <=, greater than >, greater than or equal to >=, not equal <>, and equal =.

Your **If...Then...Else** statement can be nested to as many levels as you need.

The statement has the following syntax:

**If condition Then statements [Else elsestatements] End If**

Or, you can use the block form syntax:

```
 If condition Then
 [statements]
 ElseIf condition-n Then
 [elseifstatements] . . .
 Else
 [elsestatements]
 End If
```

For more details:

- On inserting conditional statements using the Step Generator, see "["Step Generator Dialog Box" on page 732](#).
- On working with conditional steps in the Editor, see "["Comments, Control-Flow, and Other VBScript Statements " on page 661](#), and the VBScript documentation (select **Help > HP Unified Functional Testing Help > VBScript Reference**).
- On working with conditional steps in the Keyword View, see "["Conditional and Loop Statements" on page 116](#).

### Example:

The following example checks whether a valid order number is entered in the Order No. box.

To do this, UFT activates the Open Order dialog box (brings it into focus), selects the Order No. check box, and opens a box in which the user inserts a value—the relevant order number—and clicks **OK**. The first conditional statement instructs UFT to verify that the value entered by the user is greater than zero. **If** it is not greater than zero, a message box is displayed, stating that the value entered is invalid. When the user clicks **OK** to close the message box, the run session ends.

Otherwise, if the value entered is greater than zero, UFT inserts the above value in the Order No. box.

The next **If** statement instructs UFT to check whether the order number exists in the application and to send a report to the Run Results indicating that the step passed or failed. If the step failed because the order number was invalid, the Flight Reservations error message opens, and UFT clicks **OK** to close this message box before ending the run session.

**Note:** Many of the lines in the example below are comments.

For example:

```
'Set the focus on (activate) the Open Order dialog box
Window("Flight Reservation").Dialog("Open Order").Activate
'Insert a check mark in the Order No. check box
Window("Flight Reservation").Dialog("Open Order").WinCheckBox("Order No.").Set
"ON"

Insert an order number in the displayed box and save the value as "OrderNo" for
use later in the script.
If the value is less than or equal to 0, generate a message box.
(If the value is illegal and a message box is generated, end the run session
when the user clicks OK.)
OrderNo = InputBox("Enter Order Number")
If OrderNo <= 0 Then
 MsgBox "You entered an invalid order number."
 ExitAction
End If
'Insert the saved order number value in the Order No. box
Window("Flight Reservation").Dialog("Open Order").WinEdit("OrderNumberEdit").Set
OrderNo
'Click OK to close the Open Order dialog box
Window("Flight Reservation").Dialog("Open Order").WinButton("OK").Click
'Check if an error message opens and send a report to the run results
If Window("Flight Reservation").Dialog("Open Order").Dialog
("FlightReservations").Exist Then
 Reporter.ReportEvent micFail, "Check that the value of the order number is
legal", "The order number does not exist."
 Window("Flight Reservation").Dialog("Open Order").Dialog
("FlightReservations").WinButton("OK").Click
Else
 Reporter.ReportEvent micPass, "Check that the value of the order number is
legal", "The order number exists."
End If
```

## With Statements

### **Relevant for: GUI tests and scripted GUI components**

**With** statements make your script (in the Editor) more concise and easier to read by grouping consecutive statements with the same parent hierarchy.

In addition, using **With** statements might help your script run faster. When running a **With** statement, UFT identifies the object in the application before running the first statement, but does not re-identify it before running each statement.

On the other hand, this can affect the running of your test if the object referenced by the **With** statement is refreshed, redrawn, or changed in some way in the application while running the **With** statement. To instruct UFT to re-identify the object in the application before running the next statement, add a statement that calls the **RefreshObject** test object operation. For details on the

**RefreshObject** method, see the **Common Methods and Properties** section of the *HP UFT Object Model Reference for GUI Testing*.

The **With** statement has the following syntax.

```
With object
 statements
End With
```

Item	Description
<i>object</i>	An object or a function that returns an object.
<i>statements</i>	One or more statements to be performed on an object.

## Example

For example, you could replace this script:

```
Window("Flight Reservation").WinComboBox("Fly From:").Select "London"
Window("Flight Reservation").WinComboBox("Fly To:").Select "Los Angeles"
Window("Flight Reservation").WinButton("FLIGHT").Click
Window("Flight Reservation").Dialog("Flights Table").WinList("From").Select
"19097 LON "
Window("Flight Reservation").Dialog("Flights Table").WinButton("OK").Click
```

with the following:

```
With Window("Flight Reservation")
 .WinComboBox("Fly From:").Select "London"
 .WinComboBox("Fly To:").Select "Los Angeles"
 .WinButton("FLIGHT").Click
 With .Dialog("Flights Table")
 .WinList("From").Select "19097 LON "
 .WinButton("OK").Click
 End With 'Dialog("Flights Table")
End With 'Window("Flight Reservation")
```

You can instruct UFT to automatically generate **With** statements when you record a test or to generate **With** statements for any existing action. You can also remove **With** statements from an action.

Using **With** statements in your test has no effect on the run session itself, only on the way your test appears in the Editor. Generating **With** statements for your test does not affect the Keyword View in any way.

For more details, see ["How to Generate With Statements for Your Test" on page 727](#).

## Message Statements

### Relevant for: GUI tests and scripted GUI components

Add notes to the run results, or to be displayed in the Output pane while running your test.

For example, you might want to add notes to the run results about the application tested, or operating system used. Or, send a message to the run results indicating that a particular object was missing during a specific step.

### Run session messages in the HTML report

If you work with the HTML report, add a note by inserting a **Reporter.AddTestInformation** step in your test or component.

#### Example

```
Reporter.AddTestInformation "Test status", "Passed"
```

In the run results, this information is displayed in the run results summary.

### Run session messages in the Run Results Viewer

If you work with the Run Results Viewer, add a note by inserting a **Reporter.ReportNote** step in your test or component.

#### Example

```
Reporter.ReportNote "This test was run from 12.34.56.89 using a wireless connection."
```

The note is displayed in the Run Results Viewer on the **Executive Summary** page.

### Step messages in the Run Results Viewer

Send messages about specific steps to the run results by inserting a **Reporter.ReportEvent** step.

#### Example

```
Reporter.ReportEvent micFail, "Password edit box", "Password edit box does not exist"
```

In this example, `micFail` indicates the status of the report (failed). `Password edit box` is the report name, and `Password edit box does not exist` is the report message.

Use the following statuses:

<b>micPassed</b>	Causes this step to pass and sends the message to the report.
<b>micFailed</b>	Causes this step (and therefore the test) to fail, and sends the message to the report.
<b>micDone</b>	Sends a message without passing or failing the step.
<b>micWarning</b>	Sends a warning status for the step, but does not stop, pass, or fail the step.

## Display messages during the run session

Use the following methods to display messages during the run system.

<b>MessageBox VBScript function</b>	Displays a message that pauses the run session until the message box is closed.
<b>Print Utility statement</b>	Displays messages in the Output pane during a run session.

For example:

The following code iterates all the items in the **Flight Table** dialog (in the sample Flight application) and uses the **Print** Utility statement to print the content of each item to the Output pane.

```
Set FlightsList = Window("Flight Reservation").Dialog("Flights Table").
 WinList("From")
For i = 1 to FlightsList.GetItemCount
 Print FlightsList.GetItem(i - 1)
Next
```

## Test Synchronization

### Relevant for: GUI tests and scripted GUI components

When you run a test, your application may not always respond with the same speed. For example, it might take a few seconds:

- for a progress bar to reach 100%
- for a status message to appear
- for a button to become enabled
- for a window or pop-up message to open

You can handle these anticipated timing problems by synchronizing your test to ensure that UFT waits until your application is ready before performing a certain step.

There are several options that you can use to synchronize your test:

- You can insert a **synchronization point**, which instructs UFT to pause the test until an object property achieves the value you specify. When you insert a synchronization point into your action, a **WaitProperty** statement is added to the action.
- You can insert **Exist** or **Wait** statements that instruct UFT to wait until an object exists or to wait a

specified amount of time before continuing the test.

- You can modify the default amount of time that UFT waits for a Web page to load.
- When working with tests, you can increase the default timeout settings for a test to instruct UFT to allow more time for objects to appear.

To learn more, see:

- [Synchronization Points](#) ..... 724
- [Exist and Wait Statements](#) ..... 725
- [Timeout Values Modification](#) ..... 725

## Synchronization Points

### Relevant for: GUI tests and scripted GUI components

If you do not want UFT to perform a step or checkpoint until an object in your application achieves a certain status, you should insert a synchronization point to instruct UFT to pause the test until the object property achieves the value you specify (or until a specified timeout is exceeded).

For example, suppose you record a test on a flight reservation application. You insert an order, and then you want to modify the order. When you click the **Insert Order** button, a progress bar is displayed and all other buttons are disabled until the progress bar reaches 100%. Once the progress bar reaches 100%, you record a click on the **Update Order** button.

Without a synchronization point, UFT may try to click the **Update Order** button too soon during a test run (if the progress bar takes longer than the test's object synchronization timeout), and the test will fail.

You can insert a synchronization point that instructs UFT to wait until the **Update Order** button's **enabled** property equals 1. For details, see "[Add Synchronization Point Dialog Box](#)" on page 730.

**Tip:** UFT must be able to identify the specified object to perform a synchronization point. To instruct UFT to wait for an object to open or appear, use an **Exist** or **Wait** statement. For details, see "[Exist and Wait Statements](#)" on the next page.

### Example

After you insert a synchronization point for the **Flight Confirmation** button, it may look something like this:

Flight Confirmation	Sync	Wait for the Web page to synchronize before continu...
Flight Confirmation	WaitProperty	Wait until the value of the "visible" property of the "Fli...

In the Editor, this is displayed as:

```
Browser("Welcome: Mercury Tours").Page("Flight Confirmation: Mercury").Sync
Browser("Welcome: Mercury Tours").Page("Flight Confirmation:
Mercury").WebElement("Flight Confirmation#").WaitProperty "visible",true, 10000
```

## Exist and Wait Statements

### Relevant for: GUI tests and scripted GUI components

You can enter **Exist** and/or **Wait** statements to instruct UFT to wait for a window to open or an object to appear. Exist statements return a boolean value indicating whether or not an object currently exists. Wait statements instruct UFT to wait a specified amount of time before proceeding to the next step. You can combine these statements within a loop to instruct UFT to wait until the object exists before continuing with the test.

For example, the following statements instruct UFT to wait up to 20 seconds for the Flights Table dialog box to open.

```
bInDone=Window("Flight Reservation").Dialog("Flights Table").Exist
counter=1
While Not bInDone
 Wait (2)
 bInDone=Window("Flight Reservation").Dialog("Flights Table").Exist
 counter=counter+1
 If counter=10 then
 bInDone=True
 End if
Wend
```

For more details on **While**, **Exist**, and **Wait** statements, see the specific object properties in the *HP UFT Object Model Reference for GUI Testing*.

## Timeout Values Modification

### Relevant for: GUI tests and scripted GUI components

If you find that, in general, the amount of time UFT waits for objects to appear or for a browser to navigate to a specified page is insufficient, you can increase the default object synchronization timeout values for your test and the browser navigation timeout values for your test.

Alternatively, if you insert synchronization points and **Exist** and/or **Wait** statements for the specific areas in your test where you want UFT to wait a longer time for an event to occur, you may want to decrease the default timeouts for the rest of your test.

- When working with tests, to modify the maximum amount of time that UFT waits for an object to appear, change the **Object Synchronization Timeout** in the **File > Settings > Run** pane.
- To modify the amount of time that UFT waits for a Web page to load, change the **Browser Navigation Timeout** in the **File > Settings > Web** pane. For details, see the *HP Unified Functional Testing Add-ins Guide*.

## Step Generator

### **Relevant for:** GUI tests and scripted GUI components

The Step Generator enables you to add steps by selecting from a range of context-sensitive options and entering the required values, so that you do not need to memorize syntax or to be proficient in high-level VBScript. You can use the Step Generator from the Keyword View and also from the Editor.

In the "[Step Generator Dialog Box](#)" (described on page 732) you can define steps that use:

- Test object operations (tests only).
- Utility object operations.
- Calls to library functions (tests only), VBScript functions, and internal script functions.

For example, you can add a step that checks that an object exists, or that stores the returned value of a method as an output value or as part of a conditional statement. You can parameterize any of the values in your step.

When you insert a new step using the Step Generator, it is added to your test after the selected step, and the new step is selected. For details on the hierarchy of steps and objects and the positioning of new steps, see "[Keyword View User Interface](#)" on page 110.

For more details, see "[Step Generator Dialog Box](#)" on page 732.

## How to Insert Steps Using the Step Generator

### **Relevant for:** GUI tests and scripted GUI components

This task describes how to insert steps using the Step Generator.

#### 1. [Prerequisites](#)

Select where in your test the new step should be inserted.

#### 2. [Open the Step Generator dialog box](#)

Open the Step Generator from one of the following locations:

- Keyword View
- Editor
- Active Screen

For details, see "[Step Generator Dialog Box](#)" on page 732.

### 3. Define the category, type and operations for the step

First select the category for the step operation (test object, Utility object or function) and the required object or the function library source (for example, built-in or local script functions). You can then select the appropriate operation (method, property, or function) and define the arguments and return values, parameterizing them if required.

### 4. View the step documentation or syntax

In the Step Generator, view the step documentation or statement syntax and add your new step or statement to your test or function library.

### 5. Results

The Step Generator inserts a step with the correct syntax into your test. You can continue to add further steps at the same location without closing the Step Generator.

## How to Generate With Statements for Your Test

### Relevant for: GUI tests and scripted GUI components

This task describes the steps to generate and manage **With** statements in your test.

This task includes the following steps:

- "[Instruct UFT to generate With statements while recording](#)" below
- "[Generate With statements for existing actions in the Editor](#)" on the next page
- "[Remove With statements from an action in the Editor](#)" on page 729

### Instruct UFT to generate With statements while recording

1. In the **General** pane of the GUI Testing tab of the Options dialog box (**Tools > Options > GUI Testing tab > General tab**), select **Automatically generate "With" statements after recording** option.
2. In the **Generate "With" statements for \_\_ or more objects** box, enter the minimum number of consecutive, identical objects for which you want to apply the **With** statement. The default is 2.

For example, if you only want to generate a **With** statement when you have three or more consecutive statements based on the same object, enter 3.

3. Begin recording your test. While recording, statements are recorded normally. When you stop recording, the statements in all actions recorded during the current recording session are automatically converted to the **With** format.

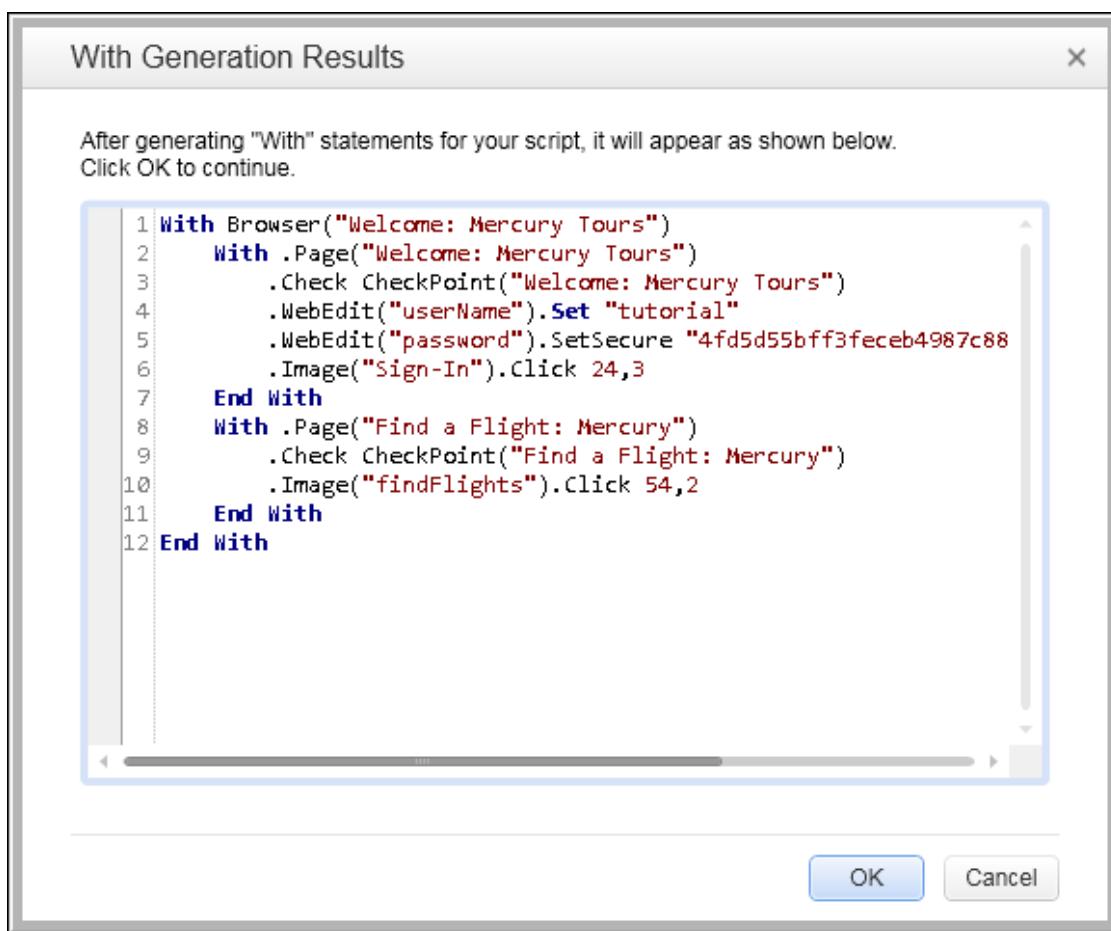
### Generate With statements for existing actions in the Editor

You can instruct UFT to generate **With** statements for any action displayed in the Editor, and to enable the statement completion functionality within existing **With** statements.

To generate With statements for existing actions:

1. In the **General** pane of the GUI Testing tab of the Options dialog box (**Tools > Options > GUI Testing tab > General tab**), select the **Automatically generate "With" statements after recording** option.
2. Confirm that the proper number is set for the **Generate "With" statements for \_\_\_ or more objects**. The default is 2.
3. Display the action for which you want to generate **With** statements.

4. From the Editor, select **Edit > Format > Apply "With" to Script**. The "With" Generation Results window opens.



Each **With** statement contains only one object.

To confirm the generated results, click **OK**. The **With** statements are applied to the action.

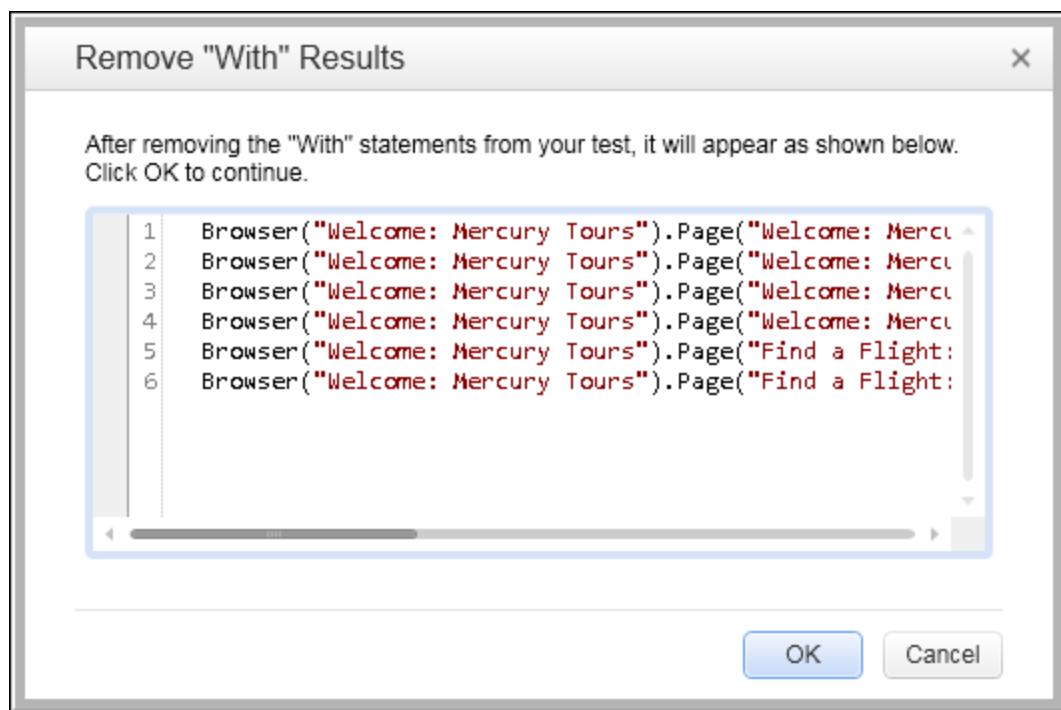
**Tip:** If you type a **With** statement (as opposed to creating it using the procedure described above), select **Edit > Format > Apply "With" to Script** to enable statement completion within the **With** statement.

### Remove With statements from an action in the Editor

You can remove all the **With** statements in an action displayed in the Editor.

**To remove With statements from an action:**

1. Display the action for which you want to remove **With** statements.
2. In the Editor, select **Edit > Format > Remove "With" Statements**. The Remove "With" Results window opens.

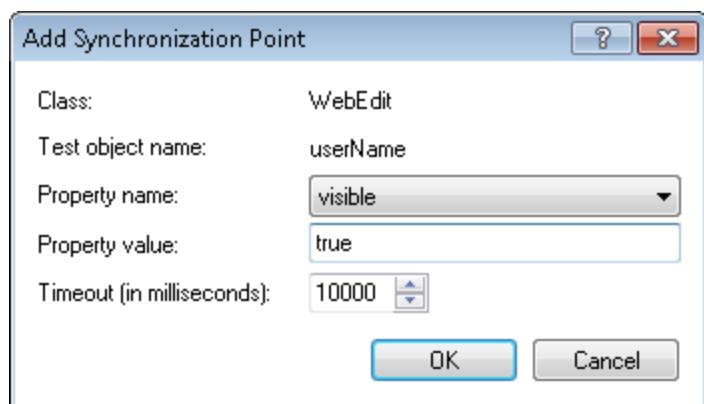


3. To confirm the results, click **OK**. The **With** statements are replaced with the standard statement format.

## Add Synchronization Point Dialog Box

**Relevant for: GUI tests and scripted GUI components**

This dialog box enables you to insert a **WaitProperty** statement to synchronize your test.



<b>To access</b>	<ol style="list-style-type: none"> <li>1. Ensure that a GUI test, action, or component is in focus in the document pane.</li> <li>2. Start a recording session.</li> <li>3. Display the screen or page in your application that contains the object for which you want to insert a synchronization point.</li> <li>4. In UFT, select <b>Design &gt; Synchronization Point</b>.</li> </ol>
<b>Important information</b>	<ul style="list-style-type: none"> <li>• When you insert a synchronization point, the pointer changes to a pointing hand.</li> <li>• For details on the <b>WaitProperty</b> method, see the <b>Common Methods and Properties</b> section in the <i>HP UFT Object Model Reference for GUI Testing</i>.</li> </ul>
<b>See also</b>	<a href="#">"Test Synchronization" on page 723</a>

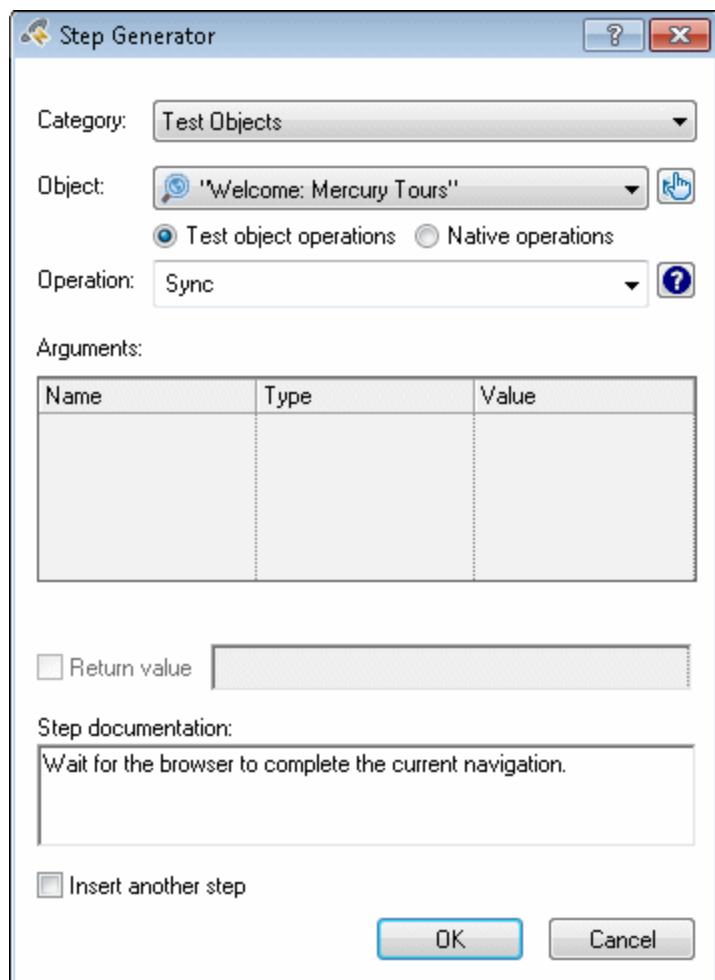
User interface elements are described below:

<b>UI Elements</b>	<b>Description</b>
<b>Class</b>	The test object class of the selected object.
<b>Test object name</b>	The name of the selected object.
<b>Property name</b>	The list of the identification properties associated with the object class. Select the property you want to use for the synchronization point.
<b>Property value</b>	<p>Enables you to specify the property value for which UFT should wait before continuing to the next step in the test.</p> <p>The property values that the object has at the time that you insert the synchronization point do not impact the specified synchronization point.</p>
<b>Timeout (in milliseconds)</b>	The time (in milliseconds) that you want UFT to wait before continuing to the next step, if the specified property value was not achieved.

## Step Generator Dialog Box

### Relevant for: GUI tests and scripted GUI components

This dialog box enables you to add steps that perform operations, using test object methods (for tests only), Utility object methods, or function calls.

**To access**

This dialog box can be accessed from the following locations:

- Keyword View
- Editor
- Function Library
- Active Screen

To open the dialog box, do the following:

1. Ensure that a GUI test or action is in focus in the document pane.
2. Select the location to insert the step, and do one of the following:

	<ul style="list-style-type: none"> <li>Right-click and select <b>Insert Step &gt; Step Generator</b>, or <b>Insert Step &gt; Step Generator</b>.</li> <li>Press F7 (not available for Active Screen).</li> <li>Select <b>Design &gt; Step Generator</b>.</li> </ul>
<b>Important information</b>	<ul style="list-style-type: none"> <li>Although the Step Generator shows information regarding the currently selected step or function, selections that you make in the Step Generator add a new step or function to your test; they do not modify the existing step or function.</li> <li>In the Step Generator, if you add an operation that returns an object, and the assignment in the test is missing a <b>Set</b> statement, the run session will fail.</li> </ul>
<b>Relevant tasks</b>	<a href="#">"How to Insert Steps Using the Step Generator" on page 726</a>

## General User Interface Elements

UI Elements	Description
<b>Category</b>	<p>The type of step to add. The following options are available:</p> <ul style="list-style-type: none"> <li><b>Test Objects.</b> Enables you to select a test object and operation for the step (for tests only). For details, see <a href="#">"Test Object Category" on page 735</a>.</li> <li><b>Utility Objects.</b> Enables you to select a Utility object and operation for the step. For details, see <a href="#">"Utility Object Category" on page 736</a>.</li> <li><b>Functions.</b> Enables you to select a function for the step from the available library functions (tests only), VBScript functions, and internal script functions. For details, see <a href="#">"Function Category" on page 737</a>.</li> </ul>
<b>Object</b>	<p>The list of available objects. The list varies according to the type of object you select in the <b>Category</b> list box.</p> <p><b>Note:</b> Available when the selected <b>Category</b> value is <b>Test Objects</b> or <b>Utility Objects</b>.</p>
<b>Library</b>	<p>The type of functions to display in the <b>Operation</b> list. Possible values:</p> <ul style="list-style-type: none"> <li><b>All.</b> Displays all functions relevant in the scope from which you opened the Step Generator.</li> <li><b>Library Functions.</b> Displays all functions defined in functions libraries associated with the action's test. (Available only when opening the Step Generator from an action.)</li> <li><b>Built-in functions.</b> Displays all of the functions that are part of the VBScript language.</li> <li><b>Local script functions.</b> Displays all of the functions defined in the action or function library from which you opened the Step Generator.</li> </ul> <p><b>Note:</b> Available when the selected <b>Category</b> value is <b>Functions</b>.</p>
<b>Operation</b>	<p><b>For objects:</b> The list of operations available for the selected object type in alphabetical order.</p> <p><b>For functions:</b> The list of functions available from the selected library type in alphabetical order.</p>
<b>Arguments</b>	The list of arguments for the operation, if applicable to the selected operation.
<b>Arguments &gt; Name</b>	The name of the argument for the selected operation.

UI Elements	Description
<b>Arguments &gt; Type</b>	The type of the argument value for the selected operation.
<b>Arguments &gt; Value</b>	<p>The value for the argument for the selected operation. Specify the argument value according to the following rules:</p> <ul style="list-style-type: none"> <li><b>Mandatory arguments.</b> If the name of the argument is followed by a red asterisk (*), you must specify a value for the argument. You cannot insert the step or view the step documentation if the values have not been defined for all mandatory arguments.</li> <li><b>Optional arguments.</b> If the name of the argument is not followed by a red asterisk (*), you can specify a value for the argument or leave the cell blank. If you do not specify a value, UFT uses the default value for the argument. (You can view the default value by moving the pointer over the cell).</li> <li><b>Required arguments.</b> If you specify a value for an optional argument, then you must also specify the values for any optional arguments that are listed before this argument. If you do not specify these values, UFT uses the default values for all required arguments. You can see the default value for each argument in a tooltip, by moving the pointer over the <b>Value</b> column.</li> <li><b>Parameterized arguments.</b> You can use a parameter for any argument value by clicking the parameterization button  .</li> <li><b>Predefined constants.</b> If an argument has a predefined list of values, UFT provides a drop-down list of possible values. If a list of values is provided, you cannot manually type a value in this box.</li> </ul>
<b>Return Value</b>	The location for storing the return values of the operation, if applicable.
<b>Step documentation (Keyword View)</b>	<p>Summary information on the current step. The following options are available:</p> <ul style="list-style-type: none"> <li>For the <b>Test Object</b> or <b>Utility Object</b> categories, the <b>Step documentation</b> box describes the operation performed by the step. When the step is inserted into your test, this description is displayed in the <b>Documentation</b> column in the Keyword View.</li> </ul> <p><b>Note:</b> If any mandatory and required argument values have not been defined for the operation, the <b>Step documentation</b> box displays a warning message.</p> <ul style="list-style-type: none"> <li>For <b>Functions</b> category, step documentation is available for user-defined functions, if you provided this information when defining them. For details, see "<a href="#">How to Create and Work with a User-Defined Function</a>" on page 700.</li> </ul>
<b>Generated step (Editor / Function Library)</b>	<p>The defined statement for the step. If all the mandatory and required argument values have not been defined for the operation, the names of the undefined arguments are highlighted in bold text. If you attempt to insert the step, an error message is displayed.</p> <p><b>Example:</b></p> <div style="border: 1px solid black; padding: 5px; width: fit-content;"> <p>Generated step:</p> <pre>Browser("Welcome: Mercury Tours").Page("Welcome: Mercury Tours").WebEdit("userName").Set Text</pre> </div>
<b>Insert another step</b>	Enables you to insert the current step and continue adding steps at the same location. The <b>OK</b> button changes to <b>Insert</b> .

## Test Object Category



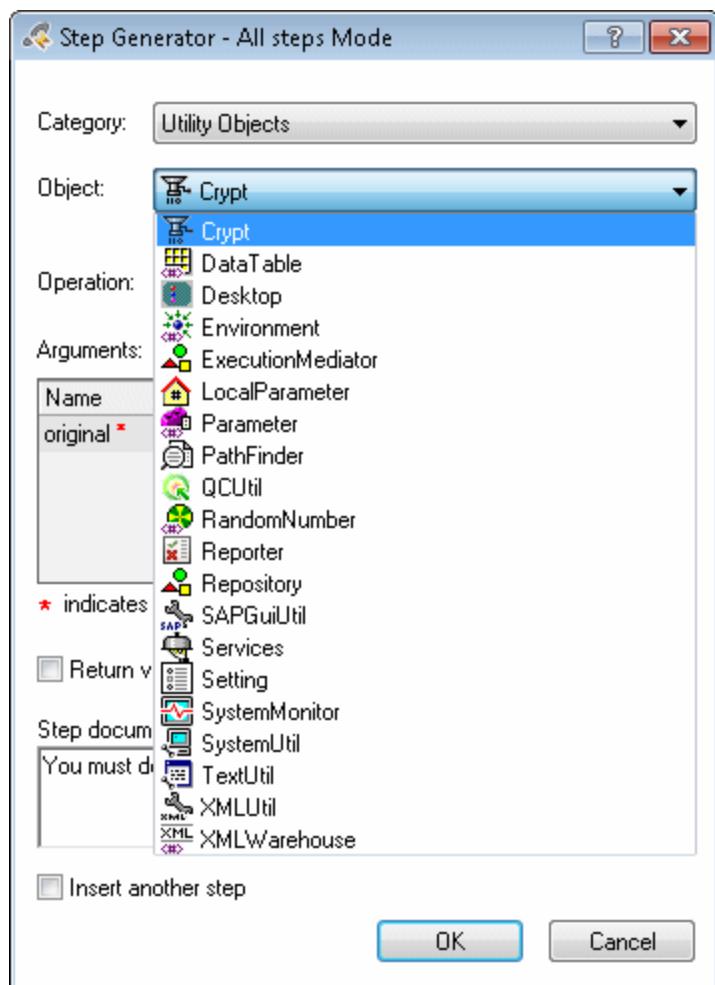
<b>Important information</b>	<ul style="list-style-type: none"> <li>You can select the object for the new step in the context of the currently selected step in your test. Alternatively, you can select any object from the object repository or from your application.</li> <li>After you select the operation for your test object, you can define the relevant argument values.</li> <li>If you click the <b>Operation Help</b> button when a native operation is selected, the <i>HP UFT Object Model Reference for GUI Testing</i> help topic opens for the selected test object. For details on specific native operations, see the documentation for the environment or application you are testing.</li> </ul>
------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

User interface elements are described below:

UI Elements	Description
<b>Object</b>	All the objects in the object repository that are at the same hierarchical level and location as the currently selected step.  <b>Note:</b> The objects are listed by name in alphabetical order.
	<b>Select Object.</b> Enables you to select an object from the object repository or from your application.
<b>Test object operations</b>	The UFT operations that can be performed on a test object.
<b>Native operations</b>	The operations of the object in your application as defined by the object creator.  <b>Note:</b> <ul style="list-style-type: none"> <li>If UFT cannot retrieve native operations for the selected object, the <b>Native operations</b> option is not available.</li> <li>If you select a native operation, the Step Generator inserts a step using <b>.Object</b> syntax. For details on using the <b>Object</b> property, see "<a href="#">Native Properties and Operations</a>" on page 663.</li> </ul>

## Utility Object Category

This option enables you to specify a utility object to insert to your test.

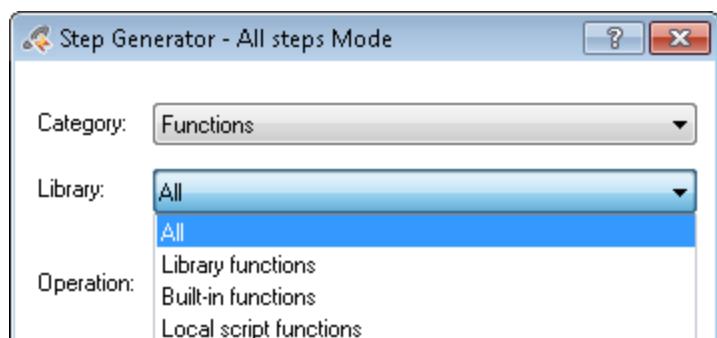


<b>Important information</b>	For details on Utility objects, see the <b>Utility Objects</b> section of the <i>HP UFT Object Model Reference for GUI Testing</i> .
------------------------------	--------------------------------------------------------------------------------------------------------------------------------------

User interface elements are described below:

UI Elements	Description
<b>Object</b>	<p>The list of Utility objects that are available.</p> <p><b>Note:</b> The image above displays the list of Utility objects that are available when you open the Step Generator from the Keyword View. When you open the Step Generator from the Editor, the list includes a number of additional Utility objects. If you have one or more add-ins loaded, the list may include additional Utility objects for those add-ins.</p>

## Function Category



**Important information** For detailed information on a selected built-in VBScript function, see Microsoft's VBScript Reference or the *HP UFT Object Model Reference for GUI Testing*.

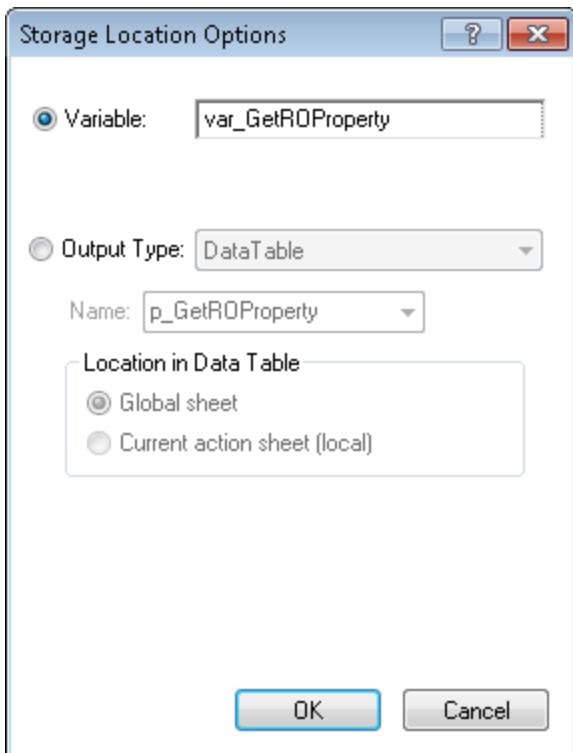
User interface elements are described below:

UI Elements	Description
<b>Library</b>	The list of function types. The following options are available: <ul style="list-style-type: none"><li><b>All.</b> Enables you to select a function from all the available functions and types.</li><li><b>Library functions.</b> Enables you to select a function from any function library associated with your test (for tests only). For details on defining and using associated function libraries, see "<a href="#">Associated Function Libraries</a>" on page 684.</li><li><b>Built-in functions.</b> Enables you to select any standard VBScript function supported by UFT. For details on working with VBScript, you can open the VBScript documentation from the UFT <b>Help</b> menu (<b>Help &gt; HP Unified Functional Testing Help &gt; VBScript Reference</b>).</li><li><b>Local script functions.</b> Enables you to select any local function defined directly in the current action or function library.</li></ul>

## Storage Location Options Dialog Box

### Relevant for: GUI tests and scripted GUI components

This dialog box enables you to specify how and where to store a return value for an operation that you have selected in the Step Generator dialog box. This dialog box also enables you to specify how and where to store the value for an output parameter for an action.



<b>To access</b>	<ul style="list-style-type: none"> <li>In the Step Generator Dialog Box, make sure the <b>Return value</b> check box is selected, click the displayed return value and then the output storage button .</li> <li>In the Action Call Properties Dialog Box, select an output parameter in the Parameter Values tab and click the output storage button  in the <b>Store in</b> column.</li> </ul>
<b>Relevant tasks</b>	<a href="#">"How to Create or Modify an Output Value Step" on page 330</a>
<b>See also</b>	<a href="#">"Default Output Definitions (tests only)" on page 328</a>

User interface elements are described below:

UI Elements	Description
<b>Variable</b>	Stores the value in a run-time variable for the duration of the run session. You can accept the default name assigned to the variable (if any) or enter a different variable name.

UI Elements	Description
<b>Output Type</b>	Stores the value in an output parameter of the specified type.

## Default Output Definitions for Action Parameter Values

When you select **Output Type** or an output action parameter value for a nested action:

- If at least one output action parameter is defined in the action calling the nested action, the default output type is **Test/action parameter** and the default output name is the first output parameter displayed in the Action Properties Dialog Box of the calling action.
- If no output action parameters are defined in the calling action, the default output type is **Data Table**, and UFT creates a new Data pane output name based on the selected value in the Global sheet of the Data pane.

When you select **Output Type** for an output action parameter value for a top-level action:

- If at least one output action parameter is defined in the test, the default output type is **Test/action parameter** and the default output name is the first output parameter displayed in the Test Properties dialog box.
- If no output action parameters are defined in the test, the default output type is **Data Table**, and UFT creates a new Data pane output name based on the selected value. The value is created in the Global sheet of the Data pane.

# Chapter 53: UFT Automation Scripts

**Relevant for: GUI tests and components**

This chapter includes:

- [UFT Automation Object Model Overview](#) ..... 741
- [When to Use UFT Automation Scripts](#) ..... 742
- [Application Object](#) ..... 743
- [UFT Automation Object Model Reference](#) ..... 744
- [Generated Automation Scripts](#) ..... 744
- [How to Create a UFT Automation Script](#) ..... 745
- [How to Run Automation Scripts on a Remote Computer](#) ..... 748
- [Troubleshooting and Limitations - Automation Scripts](#) ..... 750

# UFT Automation Object Model Overview

## Relevant for: GUI tests and components

You can use the UFT automation object model to write scripts that automate your UFT operations. The UFT automation object model provides objects, methods, and properties that enable you to control UFT from another application.

Using the objects, methods, and properties exposed by the UFT automation object model, you can write scripts that configure UFT options and run tests or components instead of performing these operations manually using the UFT interface.

Automation scripts are especially useful for performing the same tasks multiple times or on multiple tests or components, or for quickly configuring UFT according to your needs for a particular environment or application.

## What is Automation?

**Automation** is a Microsoft technology that makes it possible to access software objects inside one application from other applications. These objects can be created and manipulated using a scripting or programming language such as VBScript or VC++. Automation enables you to control the functionality of an application programmatically.

## What is the UFT Automation Object Model?

An **object model** is a structural representation of software objects (classes) that comprise the implementation of a system or application. An object model defines a set of classes and interfaces, together with their properties, methods and events, and their relationships.

The UFT **automation object model** is a set of objects, methods, and properties that enable you to control essentially all of the configuration and run functionality provided via the UFT interface. Although a one-on-one comparison cannot always be made, most dialog boxes in UFT have a corresponding automation object, most options in dialog boxes can be set and/or retrieved using the corresponding object property, and most menu commands and other operations have corresponding automation methods.

You can use the objects, methods, and properties exposed by the UFT automation object model, along with standard programming elements such as loops and conditional statements to design your script.

Automation scripts are especially useful for performing the same tasks multiple times or on multiple tests or components, or for quickly configuring UFT according to your needs for a particular environment or application.

### Example:

You can create and run an automation script from Microsoft Visual Basic that does the following:

- Loads the required add-ins for a test or component
- Starts UFT in visible mode
- Opens the test or component
- Configures settings that correspond to those in the:
  - Options dialog box
  - Test Settings or Business Component Settings dialog box
  - Record and Run Settings dialog box (tests only)
- Runs the test or component
- Saves the test or component

You can then add a simple loop to your script so that your single script can perform the operations described above for multiple tests or components.

You can also create an initialization script that opens UFT with specific configuration settings. You can then instruct all of your testers to open UFT using this automation script to ensure that all of your testers are always working with the same configuration.

## When to Use UFT Automation Scripts

### **Relevant for: GUI tests and components**

Creating a useful UFT automation script requires planning, design time, and testing. You must always weigh the initial investment with the time and human-resource savings you gain from automating potentially long or tedious tasks.

Any UFT operation that you must perform many times in a row or must perform on a regular basis is a good candidate for a UFT automation script.

The following are just a few examples of useful UFT automation scripts:

- **Initialization scripts.** You can write a script that automatically starts UFT and configures the options and the settings required for testing a specific environment.
- **Maintaining your tests and components.** You can write a script that iterates over your collection of tests and components to accomplish a certain goal. For example:
  - **Updating values.** You can write a script that opens each test or component with the proper add-ins, runs it in update run mode against an updated application, and saves it when you want to update the values in all of your tests or components to match the updated values in your application.
  - **Applying new options to existing tests and components.** When you upgrade toUFT, you may find that it offers certain options that you want to apply to your existing tests and components. You

can write a script that opens each existing test or component, sets values for the new options, then saves and closes it.

- **Modifying Actions and Action Parameters (tests only).** You can retrieve the entire contents of an action script, and add a required step, such as a call to a new action. You can also retrieve the set of action parameters for an action and add, remove, or modify the values of action parameters.
- **Calling UFT from other applications.** You can design your own applications with options or controls that run UFT automation scripts. For example, you could create a Web form or simple Windows interface from which a product manager could schedule UFT runs, even if the manager is not familiar with UFT.

## Application Object

### Relevant for: GUI tests and components

Like most automation object models, the root object of the UFT automation object model is the **Application** object.

The **Application** object represents the application level of UFT. You can use this object to return other elements of UFT such as the:

- **Test** object (which represents a test or component document)
- **Options** object (which represents the Options dialog box)
- **Addins** collection (which represents a set of add-ins from the Add-in Manager dialog box)

You can also use the **Application** object to perform operations like loading add-ins, starting UFT, opening and saving tests or components, and closing UFT.

Each object returned by the **Application** object can return other objects, perform operations related to the object and retrieve and/or set properties associated with that object.

Every automation script begins with the creation of the **UFTApplication** object. Creating this object does not start UFT. It simply provides an object from which you can access all other objects, methods and properties of the UFT automation object model.

**Note:** You can also optionally specify a remote UFT computer on which to create the object (the computer on which to run the script). For details, see **Running Automation Programs on a Remote Computer** in the **Introduction** section of the *UFT Automation Object Model Reference* in the *UFT Help*.

# UFT Automation Object Model Reference

## Relevant for: GUI tests and components

The *HP UFT Automation Object Model Reference* is a Help file that provides detailed descriptions, syntax information, and examples for the objects, methods, and properties in the UFT automation object model.

You can open the *HP UFT Automation Object Model Reference* from:

- UFT program folder (**Start > All Programs > HP Software > HP Unified Functional Testing > Documentation > Unified Functional Testing Automation Reference** or <UFT installation folder>\help\AutomationObjectModel.chm)
- UFT GUI Testing Automation and Schema References Help (**Help > HP UFT GUI Testing Automation and Schema References Help > HP UFT Automation Object Model for GUI Testing**)

# Generated Automation Scripts

## Relevant for: GUI tests and components

The Properties pane of the Test Settings dialog box, the General node of the pane of the **GUITesting** tab in the Options dialog box, and the Object Identification dialog box each contain a **Generate Script** button. Clicking this button generates an automation script file (.vbs) containing the current settings from the corresponding dialog box.

You can run the generated script as is to open UFT with the exact dialog box configuration of the UFT application that generated the script, or you can copy and paste selected lines from the generated files into your own automation script.

For example, the generated script for the Options dialog box may look something like this:

```
Dim App 'As Application
Set App = CreateObject("QuickTest.Application")
App.Launch
App.Visible = True
App.Options.DisableVORecognition = False
App.Options.AutoGenerateWith = False
App.Options.WithGenerationLevel = 2
App.Options.TimeToActivateWinAfterPoint = 500
...
...
App.Options.WindowsApps.NonUniqueListItemRecordMode = "ByName"
App.Options.WindowsApps.RecordOwnerDrawnButtonAs = "PushButtons"
App.Folders.RemoveAll
```

For more details on the **Generate Script** button and the options available in the Options, Object Identification, and Test Settings dialog boxes, see "[Configuring Object Identification](#)" on page 227, "[Global Options](#)" on page 80, and "[Document Settings](#)" on page 81.

## How to Create a UFT Automation Script

### Relevant for: GUI tests and components

This task describes when and how to create automation scripts, and includes the following steps:

- "[Prerequisites](#)" below
- "[Create the Application object](#)" below
- "[Reference the type library - optional](#)" on the next page
- "[Write your automation script](#)" on page 747
- "[Run your automation script](#)" on page 748

### 1. [Prerequisites](#)

#### • **Decide whether to use UFT Automation Scripts**

Creating a useful UFT automation script requires planning, design time, and testing. You must always weigh the initial investment with the time and human-resource savings you gain from automating potentially long or tedious tasks.

Any UFT operation that you must perform many times in a row or must perform on a regular basis is a good candidate for a UFT automation script.

#### • **Choose a language and development environment for designing and running Automation scripts**

You can write your UFT automation scripts in any language and development environment that supports automation. For example, you can use: VBScript, JavaScript, Visual Basic, Visual C++, or Visual Studio .NET.

For each language, there are a number of development environments available for designing and running your automation scripts.

### 2. [Create the Application object](#)

The procedure for creating the **Application** object differs slightly from language to language.

Below are some examples for creating the **UFTApplication** object and starting UFT in visible mode, using different programming languages. For conceptual details on the **Application** object, see "[Application Object](#)" on page 743.

## Visual Basic

The example below can be used only after setting a reference to the type library. If you are not working in a development environment that allows referencing type libraries, create the **Application** object as described for VBScript below.

```
Dim qtApp As QuickTest.Application ' Declare the object
Set qtApp = New QuickTest.Application ' Create the object
qtApp.Launch ' Start QuickTest
qtApp.Visible = True ' Make it visible
```

## VBScript

```
Dim qtApp
Set qtApp = CreateObject("QuickTest.Application")
qtApp.Launch ' Start QuickTest
qtApp.Visible = True ' Make it visible
```

## JavaScript

```
// Create the application object
var qtApp = new ActiveXObject("QuickTest.Application");
qtApp.Launch(); // Start QuickTest
qtApp.Visible = true; // Make it visible
```

## Visual C++

```
#import "QTObjectModel.dll" // Import the type library
// Declare the application pointer
QuickTest::_ApplicationPtr spApp;
// Create the application object
spApp.CreateInstance("QuickTest.Application");
spApp->Launch(); // Launch the application
spApp->Visible = VARIANT_TRUE; // Make it visible
```

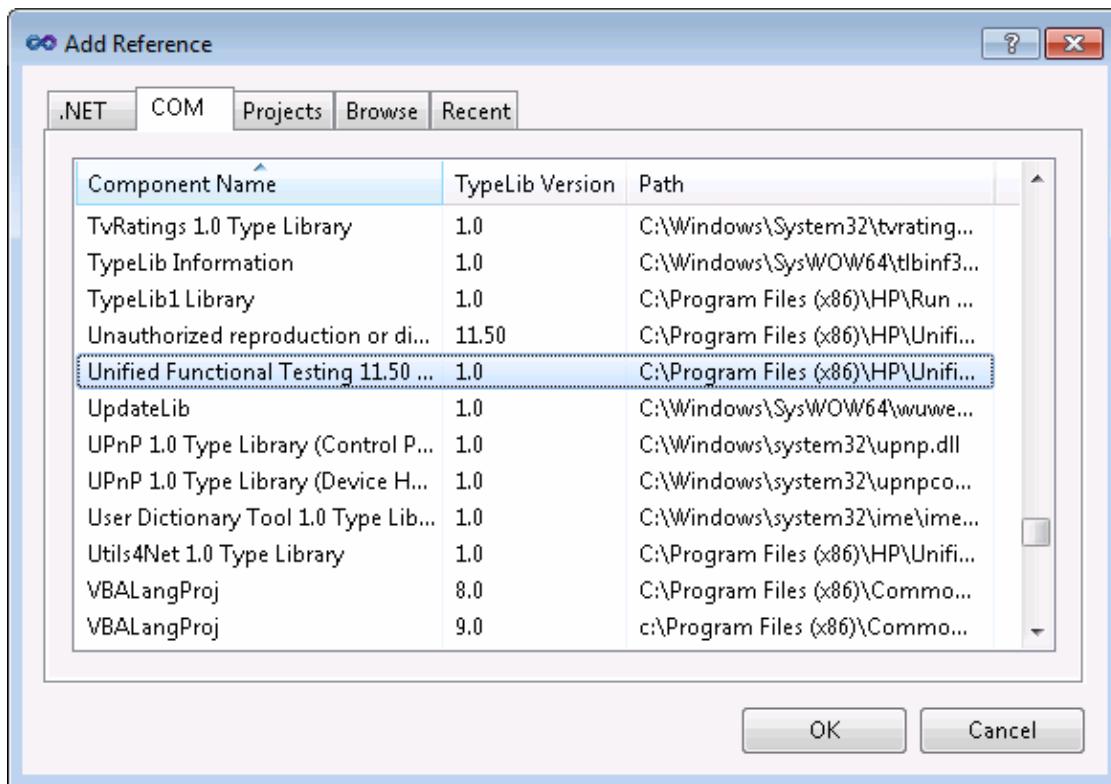
### 3. Reference the type library - optional

Some development environments support referencing a type library. A **type library** is a binary file containing the description of the objects, interfaces, and other definitions of an object model.

If you choose a development environment that supports referencing a type library, you can take advantage of features like Microsoft IntelliSense, automatic statement completion, and status bar help tips while writing your script. The UFT automation object model supplies a type library file named **QTObjectModel.dll**. This file is stored in <UFT installation folder>\bin.

If you choose an environment that supports it, be sure to reference the UFT type library before you begin writing or running your automation script. For example, if you are working in Microsoft Visual

Basic, select **Project > Add Reference** to open the **Add Reference** dialog box for your project. Then select **Unified Functional Testing<Version> Object Library** (where <Version> is the current installed version of the UFT automation type library).



#### 4. Write your automation script

The structure for your script depends on the goals of the script. You may perform a few operations before you start UFT such as retrieving the associated add-ins for a test or component, loading add-ins, and instructing UFT to open in visible mode.

After you perform these preparatory steps, if UFT is not already open on the computer, you can open UFT using the **Application.Launch** method. Most operations in your automation script are performed after the Launch method.

For details on the operations you can perform in an automation program, see *HP UFT Automation Object Model Reference* (**Help > HP UFT GUI Testing Automation and Schema References Help > HP UFT Automation Object Model Reference**).

**Tip:** You can generate automation scripts from UFT that contain the settings for the Test Settings dialog box, the GUI Testing tab in the Options dialog box, and the Object Identification dialog box as they are set on your computer. You can then run each generated script as is to instruct UFT to open on other computers with the exact dialog box configuration defined in the generated script, or you can copy and paste selected lines from the generated files into your own automation script. For details, see "Generated Automation Scripts" on page 744.

When you finish performing the necessary operations, or you want to perform operations that require closing and restarting UFT (such as changing the set of loaded add-ins), use the **Application.Quit** method.

## 5. Run your automation script

There are several applications available for running automation scripts. You can also run automation scripts from the command line using Microsoft's Windows Script Host.

For example, you could use the following command line to run your automation script:

```
WScript.exe /E:VBSCRIPT myScript.vbs
```

For details on running automation scripts on a remote computer, see "[How to Run Automation Scripts on a Remote Computer](#)" below.

# How to Run Automation Scripts on a Remote Computer

## Relevant for: GUI tests and components

By default, when you create an `Application` object in your automation script, it is created on your local computer (using your local copy of UFT). You can also run automation scripts on a remote UFT computer. To do so, you must:

### 1. Set DCOM Configuration Properties on the Remote Computer

UFT automation enables UFT to act as a COM automation server. Therefore, to run a UFT automation script on a remote computer, you must ensure that the DCOM configuration properties for that computer give you the proper permissions to launch and configure the UFT COM server.

The procedure below describes the steps you need to perform on the remote computer to enable your automation script to run on that computer. Note that the DCOM Configuration Property the appearance and names of the dialog boxes and options mentioned here may vary depending on the computer's operating system.

#### To enable automation scripts to access a Unified Functional Testing computer remotely:

- a. On the computer where you want to run the automation script, select **Start > Run**. The Run dialog box opens.
- b. Enter `dcomcnfg` and click **OK**. The Distributed COM Configuration Properties dialog box or the Component Services window opens (depending on your operating system) and displays the list of COM applications available on the computer.
- c. Select **QuickTest Professional Automation** from the DCOM Config list and open the Properties dialog box for the application. (Click the **Properties** button or right-click and select **Properties**, depending on your operating system.)

- d. In the QuickTest Professional Automation Properties dialog box, click the **Security** tab.
- e. In the launch permissions section, select the custom option and click **Edit**.
- f. Use the **Add** and **Remove** options to select the network users or groups for which you want to allow or deny permission to launch UFT via an automation script. When you are finished, click **OK** to save your settings.
- g. Repeat the previous two steps for the configuration permissions section to select the users or groups who can modify UFT configuration options via an automation script.
- h. In the QuickTest Professional Automation Properties dialog box, click the **Identity** tab and select the **interactive user** option.
- i. Click **OK** to save the QuickTest Professional Automation Properties settings.
- j. Click **OK** to close the Distributed COM Configuration Properties dialog box or the Component Services window.

## 2. Create an Application Object on the Remote Computer

After you set the necessary DCOM Configuration settings for a remote computer, you can specify that computer in your automation script.

In VBScript, you do this by specifying the computer name as the optional location argument of the `CreateObject` function. The computer name should be the same as the computer name portion of a share name. For example, to run an automation script on a computer called `MyServer`, you could write:

```
Dim qtApp
Set qtApp = CreateObject("QuickTest.Application", "MyServer")
```

For details on the syntax for specifying the remote computer in another language you are using, see the documentation included with your development environment or the general documentation for the programming language.

# Troubleshooting and Limitations - Automation Scripts

## **Relevant for: GUI tests and components**

This section describes troubleshooting and limitations for automation scripts.

**Automation:** In an automation script, if you run a GUI test that calls an API test, make sure to use the **Application.Quit** method to close UFT, before ending the script. Otherwise, UFT will behave unexpectedly.

Similarly, if you want to set the **Application.Visible** property to true after running such a GUI test, first run the **Application.Quit** method, followed by **Application.Launch** to reopen UFT.

# Chapter 54: Writing Event Handlers for API Test Steps

**Relevant for: API testing only**

This chapter includes:

• Event Based Coding - Overview .....	753
• Writing Code for API Test Events - Overview .....	755
• Writing Events for API Tests - Use-case Scenarios .....	756
• Custom Code Steps for API Tests .....	764
• How to Open a Window for Writing Custom Code .....	765
• How to Manipulate Web Service Call/HTTP Request/SOAP Request Step Input/Output Properties ..	767
• How to Stop an API Test Run from an Event .....	778
• How to Manipulate Data Programmatically .....	779
• How to Access and Set the Value of Step Input, Output, or Checkpoint Properties .....	784
• How to Report Test Run-Time Information .....	789
• How to Retrieve and Set Test or User Variables .....	792
• How to Encrypt and Decrypt Passwords .....	794
• API Event Coding - Event Structure Overview .....	795
• API Event Coding - Standard Event Structure .....	795
• API Event Coding - Web Service Event Structure .....	799
• API Test Event Coding Common Objects .....	802
• Activity Object .....	803
• Assert Object .....	804
• Checkpoint Object .....	804
• CurrentIterationNumber Object .....	805
• EncryptionMngr Object .....	806
• EnvironmentProfile Object .....	807
• InputAttachment Object .....	807
• InputEnvelope Object .....	809
• OutputAttachment Object .....	810
• OutputEnvelope Object .....	812

• Parent Object .....	813
• TestProfile Object .....	814
• UserLogger Object .....	814
• API Test Event Coding Common Methods .....	815
• Export Method .....	816
• ExportToExcelFile Method .....	817
• GetDataSource Method .....	818
• GetValue Method .....	819
• GetVariableNames Method .....	820
• GetVariableValue Method .....	821
• Import Method .....	822
• ImportFromExcelFile Method .....	823
• Info Method .....	824
• Report Method .....	825
• SelectSingleNode Method .....	825
• SetValue Method .....	826
• SetVariableValue Method .....	827
• Troubleshooting and Limitations - Using Data in API Tests and Components .....	828

# Event Based Coding - Overview

## **Relevant for: API testing only**

When testing your application's API, you can use event handlers to change or extend how you test your application's process. An **event handler** is a specific occurrence of a defined code process triggered at a specific point in the overall test flow.

When you run a test of your application's API (or run the application itself), each business process is executed as defined in your application's code. These business processes are represented in your test by the test steps you create. However, events (when running the application's code) and event handlers (when running the test) are used at specific places in the application execution or the test run. For example, if you are testing an application using a Web service, the core part of your test is the Web service call processes in which data is sent to and from the Web service. Events and event handlers can be added before, after, and at other points in the test flow, such as a special event handler that runs after the application compiles the Web service call response, sets security for the Web service response data, or adds attachments to the Web service call response.

Event handlers are designed to be run at a specific point in the application/test workflow. As a result, the objects, methods, and properties available in a given event handler are limited to the context of where the event occurs in the application or test workflow. For example, when you are working in an event handler that occurs before an application process/test step, you cannot access the process's/test step's output properties, as these properties are in a part of the application/test that has not yet run.

**Example**

You have an application based on a Web service, in which the following steps occur:

1. The application generates the Web service request.
2. The application sets the security for the Web service call request.
3. The application adds the attachments for the Web service request.
4. The application sends the request to the Web service.
5. The application receives the responses from the Web service.
6. The application reads the response and the attachments from the response.

Using event handlers, you can code an event handler in your test for any of these steps. For example, if you want to set the request properties of one step based on the response properties of a previous step, you can create code in an event handler called **BeforeExecuteStep**, and enter the property values you would like your test to use. When writing the code for this event handler, the available properties/methods for your code are limited to the objects contained in the context of the step's flow (the output/response properties available in the previous step and the input/request properties of the current step) and the event handler (which takes place immediately after the step).

You could also add another event handler that builds the attachment data into an XML document to attach to the Web service request. Likewise, you could add numerous other event handlers that supplement the core processes that your application performs.

However, if you use an event handler out of the context in which it is designed, the properties/methods of the step with which the event should run are not accessible. Thus, if you try to code an event handler for one of the response steps above after a step involved in making the Web service request (such as generating the Web service request), the properties of the response step - while accessible - are null in the response context, and your test run gets an exception when running this particular event handler code.

For details about how UFT uses event handlers, see "[Writing Code for API Test Events - Overview](#)" on the [next page](#).

# Writing Code for API Test Events - Overview

## Relevant for: API testing only

Each step has predetermined event handlers which are run at specific points in the test execution. Within these event handlers, you can add additional code (above and beyond the test step's regular execution flow) which enables you to define properties, parameters, or variables and additional processes that help to facilitate your test flow. For most test steps, there are three standard event handlers which run before the test step, after the test step, and as a checkpoint for the test step. For Web service and SOAP request steps, there are additional event handlers that mimic the process of a Web service call. For details on the event handler structure, see "[API Event Coding - Event Structure Overview](#)" on page 795.

In addition, you can use Custom Code steps, for which the entire test step flow is event handlers. For details about Custom Code steps, see "[Custom Code Steps for API Tests](#)" on page 764.

## Examples

### Case 1

Before entering customer information into a flight booking service, your application must connect to a database located locally on your computer (which mimics the application connecting to a local database). However, using the existing UI framework of the UFT API test, you cannot connect to the database. Using an event handler designed to run before the test step, you can write code that accesses your database and imports the information to your test to be entered into the flight booking service.

### Case 2

After receiving a response from your Web service (in XML format), you need to extract certain information from the response file to use as the input for another test step. You can write an event handler to run after the test step which can read and extract the information from this XML file.

Event handlers should be used to extend the behavior of existing test steps, instead of providing all the property/parameter values for test steps. It is not recommended to use custom code to set the property/parameter values of your steps and execute the steps, but instead to use the grid in the Input/Properties tab in the Properties pane to set these values.

**Note:** This functionality is in contrast to the manner GUI testing uses coding. GUI tests and components enable you to write the entire test or action flow using code. However, an API event handler or custom code activity is only a portion of the larger test flow.

It is recommended to have experience and/or knowledge in writing code before attempting to write event handlers for your tests. All API test events use the C# language and syntax, even if your application uses a different language. For details on C#, see the [Microsoft C# Reference](#).

**Note:** Any add-ins you choose in the Add-in Manager when opening UFT do not affect API tests or event/custom coding.

## Writing Events for API Tests - Use-case Scenarios

### **Relevant for: API testing only**

The following scenarios describe use-cases on how to use event coding in the context of a test of a realistic application. Each of the scenarios describes the general context in which the application is used, the workflow of the application, and a test created for the application. The steps in the test include possible input and output properties for each step, and also describe how event coding can be used to enhance various test steps.

You can view a use-case scenario for the following types of applications:

- "[Web Service application](#)" below
- "[REST Service application](#)" on page 760
- "[Standard application](#)" on page 762

### Web Service application

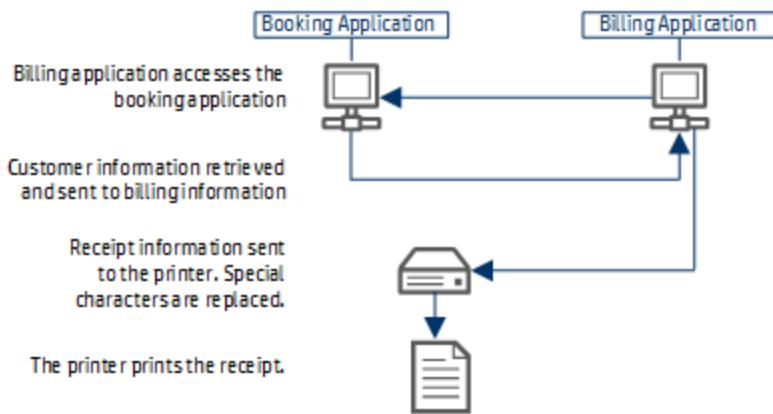
#### **Scenario**

A hotel has an application that deals with customer room charges and prints receipts for customers. The application takes the customer's room number, compiles customer charges of the customer into a bill, charges the customer's credit card, and prints a receipt. This receipt must display the customer's name, including any special characters in their name. However, the printer for the receipt can only print English characters.

The billing application is based upon a Web service which works on a cloud-based application and database.

## Application Flow

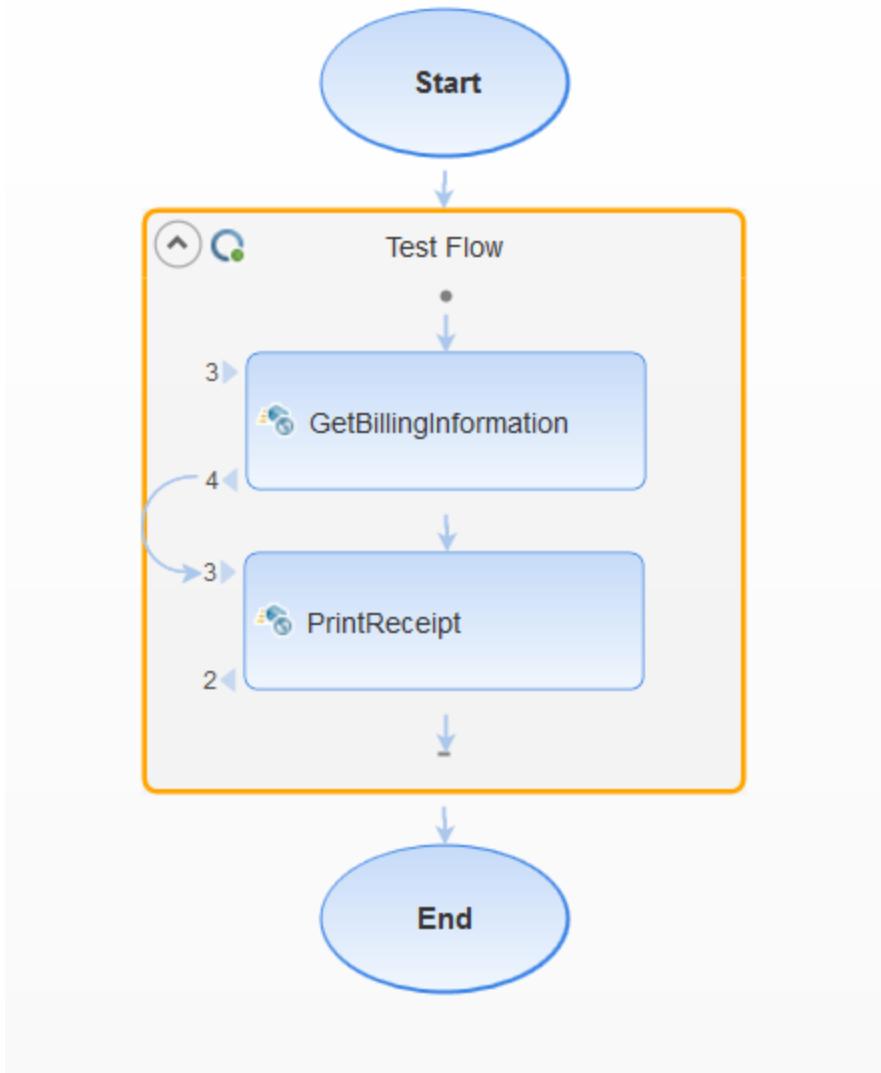
The billing application has the following workflow:



1. The billing application accesses the hotel booking application where customer information is stored. The hotel billing application retrieves the customer's name, credit card number, and total charges.
2. The billing application sends the information for the bill to the receipt printer.
3. The receipt is printed.

## Test Setup

You create the following test for the application, including a number of events:



### 1. GetBillingInformation step

This step accesses the hotel booking application in order to retrieve the customer data, including the customer name, total charges, and customer credit card number.

#### Properties:

<b>Input Properties</b>	Web service address for the hotel booking application.
<b>Output Properties</b>	<ul style="list-style-type: none"><li><i>Customer Name</i></li><li><i>Customer credit card number</i> These properties are contained in a <i>description</i> element in the Web service response.</li><li><i>Price</i></li></ul>

<b>Checkpoints</b>	Checkpoint to check whether the connection to the hotel booking application succeeded.
--------------------	----------------------------------------------------------------------------------------

## 2. PrintReceipt step

This step takes the customer information (from the description element of the GetBillingInformation step) and prints a receipt for the customer.

### Properties:

<b>Input Properties</b>	<i>Description.</i> This property is linked to the data source imported in the first step. Price. The total price to charge to the customer.
<b>Output Properties</b>	Response file containing the receipt from the
<b>Checkpoints</b>	None

### Events:

For this step, you must create an event handler for the *BeforeExecuteStepEvent* event. This event handler searches the description element for non-English characters and replaces them with the correct characters:

```

/// <summary>
 /// Handler for the StServiceCallActivity10 Activity's
BeforeExecuteStepEvent event.
 /// </summary>
 /// <param name=\"sender\">The activity object that raised the
BeforeExecuteStepEvent event.</param>
 /// <param name=\"args\">The event arguments passed to the
activity.</param>
 /// Use this.StServiceCallActivity10 to access the
StServiceCallActivity10 Activity's context, including input and output
properties.
 public void StServiceCallActivity10_OnBeforeExecuteStepEvent(object
sender, STActivityEventArgs args)
 {
 //Get the description text
 XmlNamespaceManager nsmgr = new XmlNamespaceManager
(this.StServiceCallActivity10.InputEnvelope.NameTable);
 nsmgr.AddNamespace("a",
@"http://schemas.datacontract.org/2004/07/CustomerBillingService");
 var OriginalText =
this.StServiceCallActivity10.InputEnvelope.SelectSingleNode
("//a:Description", nsmgr).InnerText;

 //In case there are non-English characters in the description,
convert them to English ones
 var ConvertedText = ConvertSpecialCharactersToEnglishCharacters
(OriginalText);

```

```
//Update the description with the converted text
this.StServiceCallActivity10.InputEnvelope.SelectSingleNode
("//a:Description", nsMgr).InnerText = ConvertedText;
}
```

## REST Service application

### Scenario

**Note:** This scenario is based on the Flights API application included with the UFT installation.

You have a flight booking application built using REST services. Your flight application retrieves the flights from the service, creates a flight order, and then reserves the customer order. The service then creates a flight order and total price, which is forwarded to the customer.

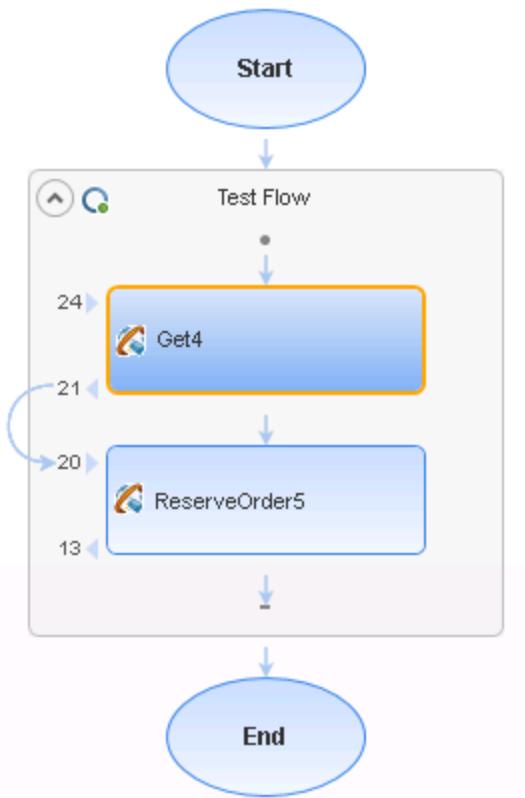
### Application Flow

The billing application has the following workflow:

1. The flight application accesses the flights database through an HTTP connection, stored externally.
2. The flight application retrieves the flights matching the specified parameters. You can search for flights using any of the following criteria:
  - Airlines
  - Arrival City
  - Arrival Time at destination
  - Departure City
  - Departure Time from departure point
3. The flight application finds a specific flight, using the output from the flight retrieval, creates a flight order, and reserves the customer's place on the flight.
4. The flight application provides the customer with a flight order number and a price.

## Test Setup

You create the following test for the application, including a number of events:



### 1. Get step

In this step, the flight application accesses the flights database, and retrieves the flights that match the customer preferences.

#### Properties:

<b>Input Properties</b>	<ul style="list-style-type: none"><li><i>Airlines</i></li><li><i>ArrivalCity</i></li><li><i>ArrivalTime</i></li><li><i>DepartureCity</i></li><li><i>DepartureTime</i></li><li><i>FlightNumber</i></li></ul>
<b>Output Properties</b>	<ul style="list-style-type: none"><li><i>Class</i></li><li><i>DepartureDate</i></li></ul>

	<ul style="list-style-type: none"> <li>• <i>FlightNumber</i></li> </ul> <p><b>Note:</b> The output properties are defined by importing a ResponseXML file, which details the required response parameters.</p>
<b>Checkpoints</b>	None

## 2. ReserveOrder step

In this step, the flight application creates a flight order based on the specified output from the Get step, and reserves the customer's place on the flight.

As part of this step, the test needs to add a checkpoint ensuring that the flight number meets standards of being greater than 999 and less than 100000.

### Properties:

<b>Input Properties</b>	<ul style="list-style-type: none"> <li>• <i>Class</i>. This property is linked to the <i>Class</i> output property of the Get step.</li> <li>• <i>CustomerName</i></li> <li>• <i>DepartureDate</i>. This property is linked to the <i>DepartureDate</i> output parameter from the Get step.</li> <li>• <i>FlightNumber</i>. This property is linked to the <i>FlightNumber</i> output property from the Get step.</li> <li>• <i>NumberofTickets</i></li> </ul>
<b>Output Properties</b>	<ul style="list-style-type: none"> <li>• <i>OrderNumber</i></li> <li>• <i>TotalPrice</i></li> </ul>
<b>Checkpoints</b>	A checkpoint ensuring that the flight number meets specified parameters. This checkpoint is added with an event handler.

### Events:

For this test step, you add two additional events:

- A CodeCheckpointEvent event. For this event, the code checks that the flight number is greater than 999 and less than 100000. In the event handler, you also add code to stop the test if the checkpoint fails.
- An AfterExecuteStepEvent event. In this event, you add code to save the response XML document as an attachment for the customer to receive.

## Standard application

### Scenario

You have an application which delivers a response received from a Web-based application. The

application creates a file in which to save the response's data, and then extracts the relevant data and adds it to the created file.

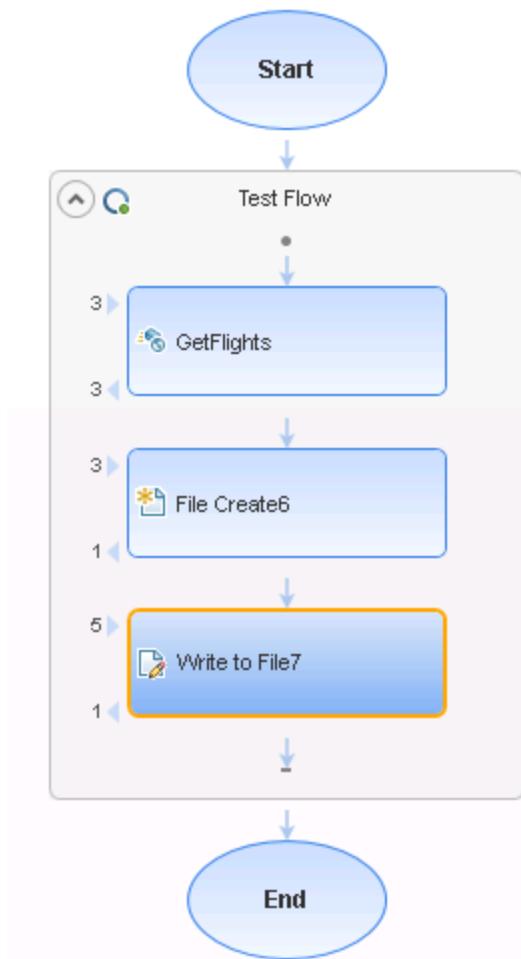
## Application Flow

The application has the following workflow:

1. The application receives the response from the Web-based application.
2. The application creates the file for the response data.
3. The application extracts the necessary content from the response, and adds this to the created file.

## Test Setup

To test the application, you create the following test:



1. A Web service step.

This step details the response expected from the Web-based application.

## 2. A File Create step.

This step creates a file in the specified directory in which the extracted data will be written.

### Properties:

<b>Input Properties</b>	<ul style="list-style-type: none"> <li>• <i>Folder Path</i> This property is linked to the <i>Class</i> output property of the Get step.</li> <li>• <i>File Name</i></li> </ul> <p>Both of these properties are entered manually in the test.</p>
<b>Output Properties</b>	None
<b>Checkpoints</b>	None

## 3. A Write To File step.

In this step, you extract the data from the Web service response, and then add the data to the file created in the CreateFile step.

### Properties:

<b>Input Properties</b>	<ul style="list-style-type: none"> <li>• <i>File Path</i>. This property is linked to the <i>Folder Path</i> property of the CreateFile step. However, since you can only link to output properties using the API testing UI, you must add an event handler to link the properties.</li> <li>• <i>Content</i>. The content comes from the response of the Web service. As this content is created dynamically during the test run, you must use an event handler to access the content.</li> </ul>
<b>Output Properties</b>	None
<b>Checkpoints</b>	None

### Events:

For this step, you need 2 events:

- A BeforeExecuteStepEvent event: This event links the *File Path* property of the current step to the *Folder Path* property from the CreateFile step. The code passes the value specified for the folder path to this test step so that the WriteToFile operation writes in the same folder and file that you created.
- An AfterExecuteStepEvent event: In this event, you access the response XML from the Web service, and use the code to extract the binary data from the Web service response. Then, you add the binary data as the content for the WriteToFile operation, in the folder and file specified in the BeforeExecuteStepEvent event.

## Custom Code Steps for API Tests

### Relevant for: API testing only

In addition to using event handlers for your API test steps, you can add a **Custom Code** step to any test. This step enables you to create a step execution flow using your own special code.

Custom Code steps are like any other API test activity. They run at the specific point in the test flow, and follow the standard event model, beginning with any code entered for the BeforeExecuteStepEvent event, followed by the ExecuteEvent event, and finishing with the AfterExecuteStepEvent event. However, unlike most API test steps, there is no predetermined step flow. For example, when you select a standard activity test step, UFT has already preset how to perform the activity, and the only modifications you can make to the test step come by writing special event handler code. For Custom Code events, the step execution is performed only with the special code you enter.

Because the step execution flow is limited only by the code you use, you can use Custom Code events in a number of ways:

- Creating unique steps not supported out of the box by UFT API tests.
- Casting variables or parameters for use in other steps

Each Custom Code step can access properties, parameters, or variables from any test step preceding the step or from a parent activity of the step. However, you should not use Custom Code steps to set step properties or parameters for other steps. Because the step runs at its place in the test flow - separate from other test steps, setting a property or parameter value for a test step in a Custom Code step does not affect the property values of the other test steps when they run in the Test Flow.

Each Custom Code step is composed of four events:

- BeforeExecuteStepEvent
- ExecuteStepEvent
- AfterExecuteStepEvent
- CodeCheckpointEvent

Only the ExecuteStepEvent event is mandatory when using a Custom Code step. When you add a Custom Code step to the canvas, UFT creates an alert in the canvas with an error in the Errors pane reminding you to create an ExecuteEvent event in the TestUserCode.cs file. Until this error is resolved, you cannot run your test.

## How to Open a Window for Writing Custom Code

### **Relevant for: API testing only**

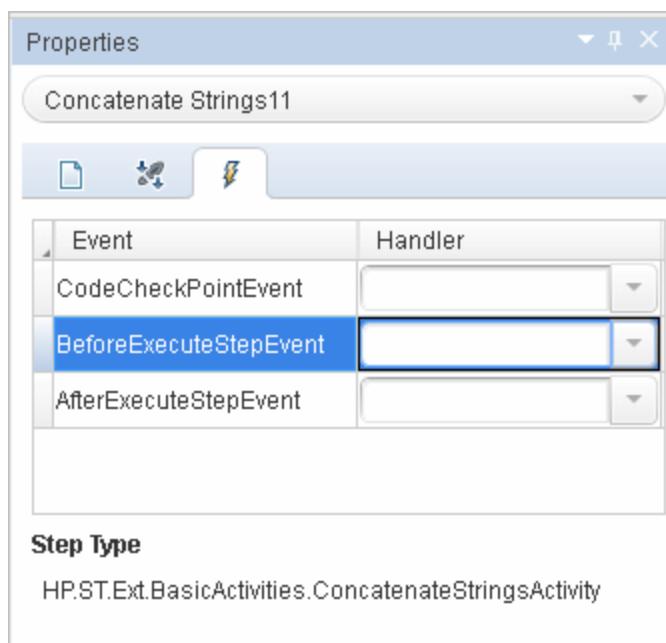
When creating or editing a test, you can use event handlers to test non-standard behavior of your application's API. For non-custom code activities, the default event handlers include events for checkpoints, before step execution, and after step execution.

This task includes the following steps:

- "Open the Events tab" on the next page
- "Select an event" on the next page
- "Edit the code" on the next page
- "Save the changes" on page 767

## 1. Open the Events tab

- a. In the canvas, select an activity.
- b. In the Properties pane, open the **Events** tab .



**Note:** You can also open the Events tab by double-clicking a **Custom Code** activity in the canvas.

## 2. Select an event

In the Handler column, double-click the row for the event to which you want to provide code.

The `TestUserCode.cs` file opens as a separate tab.

## 3. Edit the code

In the TestUserCode.cs tab, locate the **TODO** section for your event handler and add your custom code.

**Note:** Changes you make in the canvas are not reflected in the event handler code Intellisense/autocomplete options until you save the document.

```

1 namespace Script
2 {
3 using System;
4 using System.Xml;
5 using System.Xml.Schema;
6 using HP.ST.Ext.BasicActivities;
7 using HP.ST.Fwk.RunTimeFWK;
8 using HP.ST.Fwk.RunTimeFWK.ActivityFWK;
9 using HP.ST.Fwk.RunTimeFWK.Utilities;
10 using HP.ST.Fwk.RunTimeFWK.CompositeActivities;
11 using HP.ST.Ext.CustomDataProviders.Extensions;
12 using HP.ST.Ext.CustomDataProviders.ExcelFileArguments;
13 using HP.ST.Ext.WebServicesActivities;
14
15 [Serializable()]
16 public class TestUserCode : TestEntities
17 {
18
19 /// <summary>
20 /// Handler for the StServiceCallActivity4 Activity's AfterExecuteStepEvent event.
21 /// </summary>
22 /// <param name="sender">The activity object that raised the AfterExecuteStepEvent
23 /// <param name="args">The event arguments passed to the activity.</param>
24 /// Use this.StServiceCallActivity4 to access the StServiceCallActivity4 Activity.
25 public void StServiceCallActivity4_OnAfterExecuteStepEvent(object sender, STActivi
26 {
27 // TODO: Add your code here...
28 }
29 }
30 }
```

#### 4. Save the changes

Click **File > Save All** to save the custom code and the test.

## How to Manipulate Web Service Call/HTTP Request/SOAP Request Step Input/Output Properties

### Relevant for: API testing only

Using code, you can access and set the properties of your HTTP/SOAP Request or Web Service steps.

**Note:** Using event handler code for REST service and WADL steps is done in the same way as for standard API testing activities. For details, see "[How to Access and Set the Value of Step Input, Output, or Checkpoint Properties](#)" on page 784.

This task includes the following activities:

- "Access and set property values for input properties" below
- "Add checkpoint property values" on the next page
- "Specify a SOAP Fault and SOAP Fault values" on page 770
- "Assign a specific request file to a test step" on page 771
- "Assign a specific request file to a Web service step in the OnSendRequest event" on page 771
- "Set asynchronous Web service call properties" on page 772
- "Add an input attachment to a Web service call" on page 773
- "Access an attachment from a Web service call response" on page 774
- "Add a HTTP Header for Web Service Calls" on page 775
- "HTTP Headers for REST Service Calls" on page 776
- "Modify a SOAP Request security header in runtime" on page 777

### Access and set property values for input properties

1. In the canvas, select a Web service or SOAP Request step.
2. If you are using a SOAP Request step, load the XML containing the body of your SOAP request:
  - a. In the Properties pane, open the **XML Body** tab .
  - b. In the XML Body tab, click the **Load XML** button and navigate to your request file.
3. In the Properties pane, open the **Events** tab .
4. In the Events tab, create an event handler for the **AfterGenerateRequest** event. The `TestUserCode.cs` file opens.
5. In the **TODO** section of the `TestUserCode.cs` file, add the property value, using the following syntax:

```
this.StServiceCallActivity<activity #>.InputEnvelope.SelectSingleNode(XPath
to property).InnerText = "<value>";
```

For details on the Input Envelope object, see "[InputEnvelope Object](#)" on page 809. For details on the `SelectSingleNode` method, see "[SelectSingleNode Method](#)" on page 825.

### Example

The following example assigns the value of the DepartureCity input property for a flight booking web service from an Excel data source:

```
string newDepartureCityValue = GetDataSource("SampleAppData!Input").GetValue
(this.Loop2.CurrentIterationNumber-1, "DepartureCity").ToString();
string departureCityXpath = /*#__[local-name(.)='Envelope'][1]/*[local-name(.)
='Body'][1]/*[local-name(.)='GetFlights'][1]/*[local-name(.)='DepartureCity'][1]
";
this.StServiceCallActivity4.InputEnvelope.SelectSingleNode
(departureCityXpath).InnerText = newDepartureCityValue;
```

### Add checkpoint property values

You can use code to add checkpoint values in a Web service or SOAP request step. This can be very useful if the response on which the checkpoints is based is dynamically created.

**Note:** You cannot change the value of already existing checkpoints set in the Web service call.

1. Create an event handler for the CodeCheckpointEvent, as described in ["Set the value of a checkpoint" on page 788](#).
2. Following the line containing your code for enabling the checkpoint (`args.Checkpoint.RunUICheckpoints = true`), enter the value of your checkpoint, using the following syntax:

```
args.Checkpoint.Assert.Equals(<actual value>,<expected value>);
```

In the `<actual value>` and `<expected value>` parameters, you access the checkpoint properties through the step's output envelope. To access the output parameters, use the same syntax as described in ["Access and set property values for input properties" on the previous page](#). However, you must change the `InputEnvelope` to `OutputEnvelope` to access the correct properties.

### Example

In the following example, the checkpoint value for the DepartureCity value in a flight booking Web service is set:

```
string departureCityActualValueXpath = /*#__[local-name(.)='Envelope'][1]/*[local-name(.)='Body'][1]/*[local-name(.)='GetFlightsResponse'][1]/*[local-name(.)='GetFlightsResult'][1]/*[local-name(.)='Flight'][1]/*[local-name(.)='DepartureCity'][1]";
string ActualValue = this.StServiceCallActivity4.OutputEnvelope.SelectSingleNode(departureCityActualValueXpath).InnerText;

string departureCityExpectedValueXpath = /*#__[local-name(.)='Envelope'][1]/*[local-name(.)='Body'][1]/*[local-name(.)='GetFlights'][1]/*[local-name(.)='DepartureCity'][1]";
string ExpectedValue =
this.StServiceCallActivity4.InputEnvelope.SelectSingleNode(departureCityExpectedValueXpath).InnerText;

args.Checkpoint.Assert.Equals(ActualValue, ExpectedValue);
```

## Specify a SOAP Fault and SOAP Fault values

Using code, you can set your Web service call or SOAP Request to expect a SOAP fault, as well as to specify the expected fault properties.

1. In the canvas, select a Web service or SOAP Request step.
2. In the Properties pane, open the **Events** tab .
3. In the Events tab, create an event handler for the AfterGenerateRequest event. The `TestUserCode.cs` file opens.
4. In the **TODO** section of the `TestUserCode.cs` file, specify the expected fault, using the following syntax:

```
this.<activity>.FaultExpected = true;
```

5. In the Events tab, create an event handler for the CodeCheckpointEvent event.
6. In the **TODO** section of the `TestUserCode.cs` file, in the **CodeCheckpointEvent** section, specify the expected fault information, using the following syntax:

```
string xpath = "<path to fault property>";
string actualValue = this.StServiceCallActivity<activity
#>.OutputEnvelope.SelectSingleNode(xpath).InnerText;
string expectedValue = "soap:Server";
```

```
args.Checkpoint.Assert.Equals(actualValue,expectedValue);
```

**Note:** For the specified string names in the syntax above, you can use your own names.

### Example

```
string xpath = "/*[local-name(.)='Envelope'][1]/*[local-name(.)='Body'][1]/*
[local-name(.)='Fault'][1]/*[local-name(.)='faultcode'][1]";
string actualValue = this.StServiceCallActivity4.OutputEnvelope.SelectSingleNode
(xpath).InnerText;
string expectedValue = "soap:Server";
args.Checkpoint.Assert.Equals(actualValue,expectedValue);
```

## Assign a specific request file to a test step

1. In the canvas, select a Web service or SOAP Request step.
2. In the Properties pane, open the **Events** tab .
3. In the Events tab, create an event handler for the **AfterGenerateRequest** event. The `TestUserCode.cs` file opens.
4. In the **TODO** section of the `TestUserCode.cs` file, specify the expected fault, using the following syntax:

```
this.<activity>.InputEnvelope.LoadXml(@"<path to response file>");
```

**Note:** You can also load the response from a previous step instead of from a file. In this case, you need to access the `OutputEnvelope` from a previous step in place of the `@"<path to response file>"` string. For details on accessing an output property from a step, see ["Set the value of a checkpoint" on page 788](#).

## Assign a specific request file to a Web service step in the **OnSendRequest** event

1. In the canvas, select a Web service or SOAP Request step.
2. In the Properties pane, open the **Events** tab .
3. In the Events tab, create an event handler for the **OnSendRequest** event. The `TestUserCode.cs` file opens.
4. In the **TODO** section of the `TestUserCode.cs` file, specify the expected fault, using the following syntax:

```
System.Xml.XmlDocument envelope = new XmlDocument();
```

```
envelope.LoadXml(System.Text.Encoding.UTF8.GetString(args.Message));

string xpath = "<fully qualified XPath to Property>";
envelope.SelectSingleNode(xpath).InnerText = "<value to enter>";

args.Message = System.Text.Encoding.UTF8.GetBytes(envelope.OuterXml);
```

### Example

```
// Load request envelope into XML document
System.Xml.XmlDocument envelope = new XmlDocument();
envelope.LoadXml(System.Text.Encoding.UTF8.GetString(args.Message));

// Find and change the required node
string xpath = "/*[local-name(.)='Envelope'][1]/*[local-name(.)='Body'][1]/*[local-name(.)='EchoArr'][1]/*[local-name(.)='arr'][1]/*[local-name(.)='int'][1]";
envelope.SelectSingleNode(xpath).InnerText = "10";

// Save changed envelope back
args.Message = System.Text.Encoding.UTF8.GetBytes(envelope.OuterXml);
```

## Set asynchronous Web service call properties

1. In the canvas, select a Web service or SOAP Request step.
2. In the Properties pane, open the **Events** tab .
3. In the Events tab, create an event handler for **AfterGenerateRequest** event. The `TestUserCode.cs` file opens.
4. In the **TODO** section of the `TestUserCode.cs` file, specify the asynchronous call, using the following syntax:

```
<activity name>.IsAsync = true;
```

5. Below the code for the `IsAsync`, specify the port on which to listen for the Web service or SOAP Request response, using the following syntax:

```
this.<activity>.ListenOnPort = <port number>;
```

### Example

```
StServiceCallActivity8.IsAsync = true;
this.StServiceCallActivity8.ListenOnPort = 8822;
```

## Add an input attachment to a Web service call

You can use code to send an attachment with a Web service call. This is very useful when the attachment is generated dynamically, and you cannot add this attachment using the API testing interface during test design.

**Note:** If you are loading an attachment from outside your test, skip to step 5.

1. Add a custom code step to the canvas.
2. In the Properties pane, open the **Input/Output Properties** tab .
3. In the Input/Output Properties, click **Add** and select **Add Input Parameter**. In the Add Input Parameter dialog box, give the parameter a meaningful name.
4. Link the parameter to the desired attachment. For details, see "[How to Assign Data to API Test/Component Steps](#)" on page 551.
5. In the canvas, select a Web service or SOAP Request step.
6. In the Properties pane, open the **Events** tab .
7. In the Events tab, create the **AfterGenerateRequest** event. The TestUserCode.cs file opens.
8. In the **TODO** section of the TestUserCode.cs file, specify the attachment to add, using the following syntax:

```
string attachmentsInfo =
 @"<InputAttachments>
 <Type> <attachment type> </Type>
 <Attachments>
 <Origin> <path to file> </Origin>
 <OriginType> File </OriginType>
 <ContentType> <type of content> </ContentType>
 <ContentID>Auto</ContentID>
 </Attachments>
 </InputAttachments>";

this.StServiceCallActivity<activity #>.InputAttachments = new XmlDocument();
this.StServiceCallActivity<activity #>.InputAttachments.LoadXml
(attachmentsInfo);
```

You must define the attachment properties before the code that adds the attachment, as seen in the @<InputAttachments> of the code. These properties are then displayed for the test step in the **Attachments** tab in the Properties pane.

**Note:** You can define multiple attachments between the opening <InputAttachments> tag and the closing </InputAttachments> tag, using the syntax displayed above (between the <Attachments> and </Attachments> tags).

9. Optional - to override an attachment already defined in the Attachments tab in the Properties pane, you can use the following syntax:

```
string <string name> = "<fully qualified XPath to attachment defined in the Attachments tab>";
this.StServiceCallActivity<activity #>.InputAttachments.SelectSingleNode(<string name>).InnerText = @"<path to file>";
```

**Note:** This does not update the attachment's properties, but simply overwrites the attachment file.

### Examples:

- The following example adds two text file attachments to your test:

```
string attachmentsInfo =
 @"<InputAttachments>
 <Type>DIME</Type>
 <Attachments>
 <Origin>C:\somefile1.txt</Origin>
 <OriginType>File</OriginType>
 <ContentType>text/plain</ContentType>
 <ContentID>Auto</ContentID>
 </Attachments>
 <Attachments>
 <Origin>C:\somefile2.txt</Origin>
 <OriginType>File</OriginType>
 <ContentType>text/plain</ContentType>
 <ContentID>Auto</ContentID>
 </Attachments>
 </InputAttachments>";

this.StServiceCallActivity5.InputAttachments = new XmlDocument();
this.StServiceCallActivity5.InputAttachments.LoadXml(attachmentsInfo);
```

- The following example replaces an existing attachment with a text file:

```
string firstAttachmentOriginXpath = "/*[local-name(.)='InputAttachments'][1]
/*[local-name(.)='Attachments'][1]/*[local-name(.)='Origin'][1]";
this.StServiceCallActivity5.InputAttachments.SelectSingleNode(firstAttachmentOriginXpath).InnerText = @"c:\somefile.txt";
```

## Access an attachment from a Web service call response

By default, UFT saves attachments from a Web server response in the run results folder contained within the test's folder. However, you can also access these attachments using an event handler:

1. In the canvas, select the Web service or SOAP Request step for which you want to save the Web service call.
2. In the Properties pane, open the **Events** tab .
3. In the Events tab, create an event handler for the **AfterExecuteStepEvent** event. The `TestUserCode.cs` file opens.
4. In the **TODO** section of the `TestUserCode.cs` file, access the attachment information, using the following syntax:

```
string <string name> = System.IO.Path.Combine
(this.StServiceCallActivity<activity
#>.Context.ReportDirectory, "Attachments");
string[] <name> = System.IO.Directory.GetFiles(<string name>);
```

This event handler returns an array which contains the full paths to the attachments returned with the Web service response.

#### Example

```
string responseAttachmentsFolder = System.IO.Path.Combine
(this.StServiceCallActivity4.Context.ReportDirectory, "Attachments");
string[] responseAttachments = System.IO.Directory.GetFiles
(responseAttachmentsFolder);
```

### Add a HTTP Header for Web Service Calls

When you are editing your Web service call or SOAP Request steps, you can use an event to add an HTTP header. This is useful if the source for this information is created dynamically during a test run.

1. In the canvas, select a Web service or SOAP Request step.
2. In the Properties pane, open the **Events** tab .
3. In the Events tab, create an event handler for the **BeforeApplyProtocolSettings** event. The `TestUserCode.cs` file opens.
4. In the **TODO** section of the `TestUserCode.cs` file, add the header element, using the following syntax:

```
this.StServiceCallActivity4.HttpRequestHeaders.Add("<header key>", "< key
value>");
```

**Note:** You can also set the HTTP headers values by expanding the **RequestHeader** node in the Properties pane's Input/Checkpoints tab. You then link to a data source from the **Name** and **Value** rows.

If you modify the headers using code in an event handler, it will override the values in the Properties pane during the test run.

## HTTP Headers for REST Service Calls

To dynamically add an HTTP header to a REST service or to a REST service's inner HTTP Request step (when working with API tests created in UFT 11.51 or earlier or Service Test 11.51 or earlier):

1. In the canvas, select a REST method step.
2. In the Properties pane, open the **Events** tab .
3. In the Events tab, create an event handler for the **BeforeExecuteStepEvent** event. The `TestUserCode.cs` file opens.
4. In the **TODO** section of the `TestUserCode.cs` file, allocate the desired length for the array:

```
(args.Activity as HTTPActivity).RequestHeaders =
new HP.ST.Shared.Utilities.Pair<string, string>[<# of headers>];
```

```
(args.Activity as HTTPActivity).RequestHeaders = new
HP.ST.Shared.Utilities.Pair<string, string>[2];
```

5. Below the allocation code, provide the each array element with the **<HeaderName>** and **<HeaderValue>** for each array element:

```
(args.Activity as HTTPActivity).RequestHeaders[0] = new
HP.ST.Shared.Utilities.Pair<string, string>("<header name>", "<header
value>")
```

**Note:** You will need to provide a separate line for each of the headers, as specified in your allocation statement.

### Example

```
(args.Activity as HTTPActivity).RequestHeaders = new
HP.ST.Shared.Utilities.Pair<string, string>[2];
(args.Activity as HTTPActivity).RequestHeaders[0] = new
HP.ST.Shared.Utilities.Pair<string, string>"HeaderName1", "Value1");
(args.Activity as HTTPActivity).RequestHeaders[1] = new
HP.ST.Shared.Utilities.Pair<string, string>"HeaderName2", "Value2");
```

## Modify a SOAP Request security header in runtime

1. In the canvas, select a SOAP Request step (or Web Service call step using SOAP).
2. In the Properties pane, open the **Events** tab .
3. In the Events tab, create an event handler for the **AfterProcessRequestSecurity** event. The `TestUserCode.cs` file opens.
4. In the **TODO** section of the `TestUserCode.cs` file, perform a string replace to add the new user name credential:

```
string s = args.Message.InnerXml.Replace
("<wsse:Username>User</wsse:Username>", "<wsse:Username>New
User</wsse:UserName>");
args.Message.InnerXml = s;
```

5. Below the string replace code, modify the header XML:

```
 XmlDocument xmlDocument = args.Message;
 XmlNamespaceManager xmlnsManager = new XmlNamespaceManager
(xmlDocument.NameTable);
 xmlnsManager.AddNamespace("wsse", "http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd");
 xmlnsManager.AddNamespace("wsu", http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd");
 xmlDocument.SelectSingleNode("//wsse:Username", xmlnsManager).InnerText =
"New Name";
 args.Message = xmlDocument;
```

### Example

```
string s = args.Message.InnerXml.Replace
("<wsse:Username>Alex</wsse:Username>", "<wsse:Username>John
Alex</wsse:UserName>");
args.Message.InnerXml = s;

 XmlDocument xmlDocument = args.Message;
 XmlNamespaceManager xmlnsManager = new XmlNamespaceManager
(xmlDocument.NameTable);
 xmlnsManager.AddNamespace("wsse", "http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd");
 xmlnsManager.AddNamespace("wsu", http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd");
 xmlDocument.SelectSingleNode("//wsse:Username", xmlnsManager).InnerText =
"New Name";
 args.Message = xmlDocument;
```

# How to Stop an API Test Run from an Event

## Relevant for: API testing only

Using custom code, you can stop a test run. This is useful if a condition in your test fails, and you do not want to continue the test run.

To stop the test run, do the following:

1. In the canvas, select the step on which you want to stop the test run (if necessary).
  2. In the Properties pane, open the **Events** tab .
  3. In the Events tab, create an event handler. The `TestUserCode.cs` file opens. The event you select depends on the test property you want to ensure is correct and where that property occurs in the test flow.
- Tip:** To differentiate this event from other default events, enter a descriptive name like `stopTestScript` in the event name field.
4. In the **TODO** section of the `TestUserCode.cs` file, enter an `if` statement for the current activity. For details on `if` statements in C#, see [http://msdn.microsoft.com/en-us/library/5011f09h\(v=vs.90\).aspx](http://msdn.microsoft.com/en-us/library/5011f09h(v=vs.90).aspx).
  5. Below the `if` statement in the `TestUserCode.cs` file, enter the value you want to report using the following syntax:

```
this.Context.ReplayApiClient.Stop();
```

If the condition entered in your event is not met, UFT immediately stops the test run, and does not display test results. If the condition is met, the test run continues.

## Example

The following example stops a test run on a SOAP Response step. This code is set on the `OnReceiveResponse` event, given the name `stopTestScript`. It stops the test run if the SOAP response returns a fault:

```
String node_xpath = "/*local-name(.)='Envelope'][1]/*local-name(.)='Body'][1]";
if(this.StServiceCallActivity10.OutputEnvelope.SelectSingleNode(node_
xpath).FirstChild.Name=="soapenv:Fault")

 this.Context.ReplayApiClient.Stop();
```

# How to Manipulate Data Programmatically

## Relevant for: API testing only

Using code, you can retrieve or set test step property/parameter values, import or export property/parameter values, or data-drive the property/parameter values of your test steps.

This task includes the following steps:

- "Prerequisite" below
- "Retrieve a value from a data source" below
- "Set a property value from a data source" on the next page
- "Import a data source file to your test" on page 781
- "Export the property values to a file" on page 782
- "Data drive test step property/parameter values" on page 782

## Prerequisite

Add at least one data source to your test and save the test.

## Retrieve a value from a data source

In order to get a value from a data source, you must use the `GetDataSource` and `GetValue` methods:

1. Select the step for which you want to retrieve a data source value.
2. In the Properties pane, open the **Events** tab .
3. In the Events tab, create an event handler. The `TestUserCode.cs` file opens.
4. In the **TODO** section of the `TestUserCode.cs` file, call the data source value you want to retrieve using the following syntax:

```
GetDataSource("<data source name>").GetValue(<row index>, "<column name>");
```

**Note:** When entering the parameters for the `GetValue` function, your row index is based on 0.

5. (Optional) If you want to retrieve the value corresponding to the value in the current iteration, you replace the `<row index>` with the `CurrentIterationNumber` property, using the following syntax:

```
GetDataSource("<data source name>").GetValue
(<this.Loop<#>.CurrentIterationNumber, "<column name>");
```

**Notes:**

- When running a data-driven test, UFT runs one iteration for each row in the data source. Therefore, the loop number you enter corresponds to the row of the data source, unless you specify a different starting row in the Data Source navigation policies.
- The CurrentIterationNumber is one-based, meaning that the number entered for the current loop must be 1 or higher.

### Examples

- This example retrieves a value from the first row of an Excel data source, converts it to a string, and then assigns it as the property value for GetFlights test step Flight Number property:

```
this.GetFlights4.FlightNumber =
GetDataSource("GetFlights4_Input!MainDetails").GetValue(0, "Flight_
Number").ToString();
```

- This example also retrieves a value from an Excel data source, but from the current iteration's row in the Excel sheet. The event then assigns the retrieved value to the GetFlights test step's FlightNumber property.

```
this.GetFlights4.FlightNumber =
GetDataSource("GetFlights4_Input!MainDetails").
GetValue(this.Loop4.CurrentIterationNumber, "Flight_Number").ToString();
```

## Set a property value from a data source

You can use the SetValue method to insert a property value in a data source. This enables you to populate a data source with manually-entered values or values taken from another source.

1. Select the step for which you want to set a data source value.
2. In the Properties pane, open the **Events** tab .
3. In the Events tab, create an event handler. The TestUserCode.cs file opens.
4. In the **TODO** section of the TestUserCode.cs file, call the data source value you want to enter using the following syntax:

```
GetDataSource("<data source name>").SetValue(<row index>, "<column name>",
"<value to enter>");
```

**Example**

The following example writes values to the Flight\_Number and Tickets\_Ordered columns of an Excel data source attached to a test:

```
GetDataSource("CreateFlightOrder4_Input!MainDetails").SetValue(0, "Flight_Number", "Y22");
GetDataSource("CreateFlightOrder4_Input!MainDetails").SetValue(0, "Tickets_Ordered", "2");
```

**Import a data source file to your test**

You can import data to your test using the `Import` and `ImportFromExcelFile` (if you are importing an Excel file) methods. This enables you to add data in runtime and populate your property values with this data.

1. Select the step to which you want to import data values
2. In the Properties pane, open the **Events** tab .
3. In the Events tab, create an event handler. The `TestUserCode.cs` file opens.
4. In the **TODO** section of the `TestUserCode.cs` file, call the data source value you want to enter using the following syntax:

```
ExcelFileImportInputArgs <name> = new ExcelFileImportInputArgs(@"<path to data source>", "<data source name>", <boolean whether there is a header>);
GetDataSource("<data source name>").Import(<name>);
```

or

```
GetDataSource("<data source name>").ImportFromExcelFile(@"<path to data source>", "<data source name>", <boolean whether there is a header>);
```

**Example**

The following example imports an Excel Data source:

```
ExcelFileImportInputArgs a = new ExcelFileImportInputArgs(@"C:\DemoExcel.xls", "MainDetails", true);
GetDataSource("CreateFlightOrder4_Input!MainDetails").Import(a);

GetDataSource("CreateFlightOrder4_Input!MainDetails").ImportFromExcelFile
(@"C:\DemoExcel.xls", "MainDetails", true);
```

## Export the property values to a file

You can also export the data from a test step to an external file using the Export and ExportToExcelFile methods. This enables you to export values to a file that other test steps can access to provide values for their properties/parameters in runtime.

1. Select the step for which you want to export its data values
2. In the Properties pane, open the **Events** tab .
3. In the Events tab, create an event handler. The TestUserCode.cs file opens.
4. In the **TODO** section of the TestUserCode.cs file, call the data source value you want to enter using the following syntax:

```
ExcelFileExportInputArgs <name> = new ExcelFileExportInputArgs(@"<path to
file>");
GetDataSource("<data source name>").Export(<name>);
```

or

```
GetDataSource("<data source name>").ExporttoExcelFile(@"<path to file>");
```

### Example

The following example takes the values from the input properties for a CreateFlightOrder step and writes these to an Excel file:

```
ExcelFileExportInputArgs a = new ExcelFileExportInputArgs
(@"C:\ExportedExcel.xls");
GetDataSource("CreateFlightOrder4_Input!MainDetails").Export(a);

GetDataSource("CreateFlightOrder4_Input!MainDetails").ExportToExcelFile
(@"C:\ExportedExcel.xls");
```

## Data drive test step property/parameter values

You can also data drive test step property/parameter values using code. This is useful in custom scenarios where you cannot link to your data source with the user interface options or you need to link to a data source created in the test runtime.

1. Link the Test Flow/test loop with the data source. For details, see "[Add a data source to the Test Flow or test loop](#)" on page 566.
2. Set the Data Navigation policy for the data source. For details, see "[How to Set the Data Source Navigation Properties](#)" on page 566.
3. In the canvas, select the step to data drive.

4. In the Properties pane, open the **Events** tab .
5. In the Events tab, create an event handler. The `TestUserCode.cs` file opens.

**Tip:** If you are populating values for the currently selected test step, use the `BeforeExecuteEvent` step. This ensures that the properties/parameters are mapped to the appropriate data source values before the step is run.

6. Connect your property value to the data source using the following syntax:

- For test step properties are not based on an array:

```
var <variable name> = GetDataSource("<data source>").GetValue(<row index>, "<column name>").ToString();
<activity name>.<property name> = <variable name>;
```

- For test step properties based on an array:

```
var <variable name> = GetDataSource("<data source>").GetValue(<row index>, "<column name>").ToString();
<activity name>.InputEnvelope.SelectSingleNode("<fully qualified xpath to property value>").InnerText = <variable name>;
```

#### Notes:

- You can retrieve the XPath to a property name stored in an array by right-clicking the **Property name** in the **Input/Checkpoints** tab and selecting **Copy Fully Qualified XPath**.
- If you want to set the value of an output value or checkpoint stored in an array, change the `InputEnvelope` to `OutputEnvelope` in the syntax displayed above.
- If you are running multiple iterations using data-driving, you can use the `this.Loop<#>.CurrentIterationValue` property as the row index. However, since the `CurrentIterationProperty` is one-based, but the row-index is zero-based, add a `-1` to the `CurrentIterationProperty` to ensure that your last iteration does not fail.

### Example

The following example sets the **DepartureCity** and **ArrivalCity** values for a flight booking Web service from an Excel data source called "WebServiceData."

```
var GetFlights_Input_Departure_City = GetDataSource
("WebServiceData!Input").GetValue(this.Loop2.CurrentIterationNumber-1,
"DepartureCity").ToString();
StServiceCallActivity4.InputEnvelope.SelectSingleNode("//[local-name(.)"
='Envelope'][1]/*[local-name(.)='Body'][1]/*[local-name(.)='GetFlights'][1]/*
[local-name(.)='DepartureCity'][1]").InnerText = GetFlights_Input_Departure_
City;
var GetFlights_Input_Arrival_City = GetDataSource
("WebServiceData!Input").GetValue(this.Loop2.CurrentIterationNumber-1,
"ArrivalCity").ToString();
StServiceCallActivity4.InputEnvelope.SelectSingleNode("//[local-name(.)"
='Envelope'][1]/*[local-name(.)='Body'][1]/*[local-name(.)='GetFlights'][1]/*
[local-name(.)='ArrivalCity'][1]").InnerText = GetFlights_Input_Arrival_City;
```

## How to Access and Set the Value of Step Input, Output, or Checkpoint Properties

### Relevant for: API testing only

You can set the values of input, output, or checkpoint properties through event handlers and custom code.

This task includes the following activities:

- ["Access an step property" below](#)
- ["Access a step's parent activity" on page 786](#)
- ["Set the value of a step's properties" on page 786](#)
- ["Access the value of a step's property in runtime" on page 787](#)
- ["Enable or ignore selected checkpoints - optional" on page 788](#)
- ["Set the value of a checkpoint" on page 788](#)

### Access an step property

1. From the Toolbox pane, drag the activity for which you want to access properties or add a custom code activity or event handler to an existing activity. Make sure that the custom code activity is after the activity for which you want to define properties.
2. In the canvas, select the step.
3. In the Properties pane, open the **Events** tab .

4. In the Events tab, create an event handler. A `TestUserCode.cs` file opens in the document pane.
5. In the **TODO** section of the `TestUserCode.cs` file, use the `this` object to access the properties of the activity for which you want to set properties, using the following syntax:

```
this.<activity name>.Input.<property name>
```

**Note:** For more details on the `this` object, see <http://msdn.microsoft.com/en-us/library/dk1507sz.aspx>.

6. Enter the step name for which you want to set properties followed by a `.` character.
7. Enter the property name of the activity to set properties, followed by a `.` character. In this example, you set the **Prefix** and **Suffix** properties for the `ConcatenateStringsActivity` step.

```
TestUserCode.cs X APITest2
Script.TestUserCode
CodeActivity5_OnExecuteEvent(object sender, STActivityEventArgs e)
[Serializable()]
public class TestUserCode : TestEntities
{
 /// <summary>
 /// Handler for the CodeActivity5 Activity's ExecuteEvent event.
 /// </summary>
 /// <param name="sender">The activity object that raised the Event</param>
 /// <param name="args">The event arguments passed to the activity</param>
 /// Use this.CodeActivity5 to access the CodeActivity5 Activity's
 public void CodeActivity5_OnExecuteEvent(object sender, STActivityEventArgs e)
 {
 this.ConcatenateStringsActivity4.|
```

The screenshot shows the `TestUserCode.cs` file in the IDE. The code defines a class `TestUserCode` that inherits from `TestEntities`. It contains a method `CodeActivity5_OnExecuteEvent` that takes `object sender` and `STActivityEventArgs e` as parameters. A tooltip is displayed over the `this.` prefix in the code, listing properties such as `Prefix`, `Suffix`, and `Report`. The IDE interface includes tabs for `Solution`, `Errors`, and `Warnings`, along with a status bar at the bottom.

## Access a step's parent activity

The **Parent** property of a step refers to the loop, condition, or parent activity that encloses a test step.

1. To access the activity's parent, use the `this` object as described in the ["Access an step property" on page 784](#) step,
2. Instead of entering the step's properties, enter the **Parent** property for the step.
3. If you want to get the parent loop for an activity, enter the `GetParentLoop` method, using the following syntax:

```
this.<activity name>.GetParentLoop();
```

### Example

This example retrieves the parent activity and converts it into a string.

```
string ParentName = this.ConcatenateStringsActivity5.Parent.Name
```

## Set the value of a step's properties

1. In the canvas, select the step for which you want to set property values.
2. In the Properties pane, open the **Events** tab .
3. In the Events tab, create an event handler. The `TestUserCode.cs` file opens.
4. In the **TODO** section of the `TestUserCode.cs` file, enter the parameter value for the parameters/properties, using this syntax:

```
this.<activity name>.Input.<parameter name> = <parameter value>;
or
this.<activity name>.Output.<parameter name> = <parameter value>;
```

You cannot set checkpoint values using the `this.<activity name>` object. You must use the `args.Checkpoint` object instead.

**Important:** Make sure that the value is the same type as the type entered when you created the parameter, i.e. a string parameter must have a string value.

When your test runs, UFT passes the value as specified in your custom code step to the other activity.

### Example

This example sets the value of the Prefix and Suffix value for a Concatenate Strings activity:

```
CodeActivity6.Input.a = "Hello";
CodeActivity6.Input.b = "World";
this.ConcatenateStringsActivity4.Prefix = CodeActivity6.Input.a;
this.ConcatenateStringsActivity4.Suffix = CodeActivity6.Input.b;
```

### Access the value of a step's property in runtime

You can also report the runtime value of a test step's property or parameter in the Output pane during a test run. This is useful if you want to watch the value of a particular operation or object during the test run, without having to stop and debug the test.

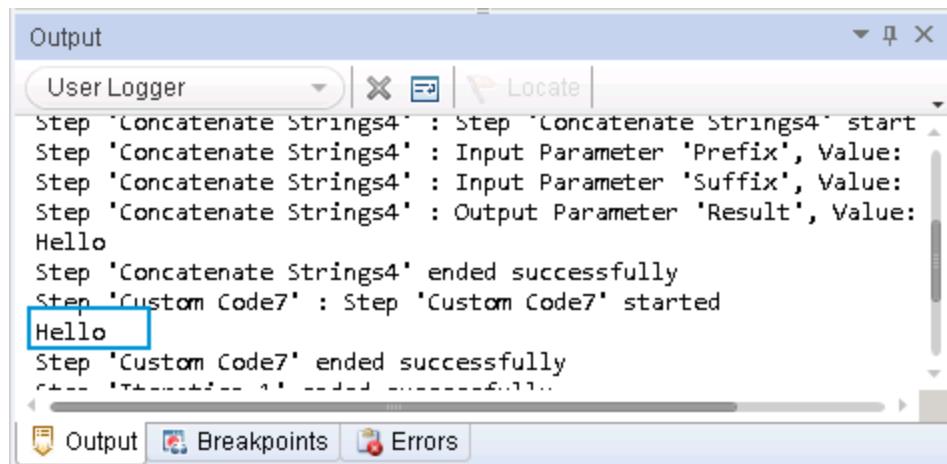
To do this, you can use a UserLogger object:

1. Add an event handler for the **OnAfterExecuteStep** event to the step for which you want to watch a property/parameter value.
2. Use the UserLogger object to report the value to the compilation log in the Output pane, using the following syntax:

```
Context.UserLogger.Info(<activity name>.<property name>);
```

**Note:** Using the Context object in an event handler or custom code enables you to get the contextual value for the property/parameter (i.e., the runtime value), instead of the entered value (what you enter in the Properties pane, for example).

When the compilation log is displayed in the Output pane, you will see a line with the property/parameter value displayed as part of the compilation log. Note that the Output pane does not list the property/parameter name with the value, so you must search within the step where you placed the code to find this value, as seen in the example below:



### Example

This example retrieves the value of the Prefix property of a **ConcatenateStrings** activity:

```
Context.UserLogger.Info(ConcatenateStringsActivity4.Prefix);
```

## Enable or ignore selected checkpoints - optional

You can instruct UFT to ignore the checkpoints for any test step. This is useful if you need to switch between enabling and ignoring checkpoints on different test runs.

You use the Checkpoint object to access the checkpoint's properties:

1. In the canvas, select the step for which you want to enable or ignore the checkpoints.
2. In the Properties pane, select the **Events** tab .
3. In the Events tab, create an event handler for the **OnCodeCheckpointEvent** event. The `TestUserCode.cs` file opens.
4. In the **TODO** section of the `TestUserCode.cs` file, enable or ignore a checkpoint for an event using the following syntax:

```
args.Checkpoint.RunUICheckpoints = true (to enable a checkpoint)
or
args.Checkpoint.RunUICheckpoints = false (to disable a checkpoint)
```

For more details on the args object, see [http://msdn.microsoft.com/en-us/library/aa884376\(v=ax.50\).aspx](http://msdn.microsoft.com/en-us/library/aa884376(v=ax.50).aspx).

## Set the value of a checkpoint

In addition to enabling or ignoring a checkpoint, you can also set the value (expected or not expected) of a checkpoint. This can be useful both to validate that your application works as expected but also does not allow unexpected behaviors.

1. Enable a checkpoint for the step, as described in "Enable or ignore selected checkpoints - optional" above.
2. Following the line containing your code for enabling the checkpoint (`args.Checkpoint.RunUICheckpoints = true`), enter the value of your checkpoint, using the following syntax:

```
args.Checkpoint.Assert.Equals("<actual value>", "<expected value>");
```

For more details on the args object, see [http://msdn.microsoft.com/en-us/library/aa884376\(v=ax.50\).aspx](http://msdn.microsoft.com/en-us/library/aa884376(v=ax.50).aspx). For details on the supported methods for the args object in UFT, see "Assert Object" on page 804.

### Examples

- The following example sets the value of a checkpoint for a **ConcatenateStrings** step:

```
args.Checkpoint.Assert.Equals(this.ConcatenateStringsActivity4.Prefix+this.ConcatenateStringsActivity4.Suffix, this.ConcatenateStringsActivity4.Result);
```

- This example checks if the text value (alphabetical order for a string) of the prefix is less than the suffix. To ensure that the checkpoint succeeds, enter a prefix value that is greater than the suffix, for example a prefix of **aa** and a suffix of **bb**.

```
args.Checkpoint.Assert.Less("Concatenate test", this.ConcatenateStringsActivity4.Prefix, this.ConcatenateStringsActivity4.Suffix, "The prefix is less than the suffix");
```

## How to Report Test Run-Time Information

### Relevant for: API testing only

Using custom events, you can report the run-time values or information of a given step, property, or parameter. You can choose either to report this to the Output pane or the run results. Viewing this information provides a simpler alternative to watching a object/property/parameter value while debugging.

To send run-time information, you can use the Report function or the UserLogger object.

This task includes the following activities:

- "[Report a custom message to the run results](#)" below
- "[Report run-time values to the Output pane](#)" on page 791

### Report a custom message to the run results

Using the Report function, you can send a custom message to the run results.

- Select the step for which you want to report information.
- In the Properties pane, open the **Events** tab .
- In the Events tab, create an event handler. The `TestUserCode.cs` file opens.

4. In the **TODO** section of the `TestUserCode.cs` file, enter the value you want to report using the following syntax:

```
this.<activity name>.Report("<report information title>", "<reported data>");
```

or

```
<activity name>.Report("<report information title>", "<reported data>");
```

**Tip:** If you use the `Context` object after the `<Activity Name>` property, you report the run-time value of the selected object, property, or parameter.

The `Report` method displays a custom message in the step's captured data. In this example, a `ConcatenateStrings` test step implemented the following code in an event handler:

```
args.Activity.Report("TestID", "APR-12-2010_CYCLE_1");
```

Name	Value
Type	HP.ST.Ext.BasicActivities.ConcatenateStringsActivity
Step ID	ConcatenateStringsActivity4
Message	Successfully concatenated strings
TestID	APR-12-2010_CYCLE_1
Prefix	'Hello '
Suffix	'World'
Result	'Hello World'
Name	'Concatenate Strings4'
Comment	"
Status	Done

### Examples

The following example reports the value used for the **Prefix** property of a **ConcatenateStrings** activity:

```
this.ConcatenateStringsActivity4.Report("Run-time Prefix Value",
this.ConcatenateStringsActivity4.Prefix)
```

### Report run-time values to the Output pane

Using a **UserLogger** object, you can view the run-time value of a particular property, parameter, or object. This value is reported in the **UserLogger** build log displayed in the Output pane during test compilation.

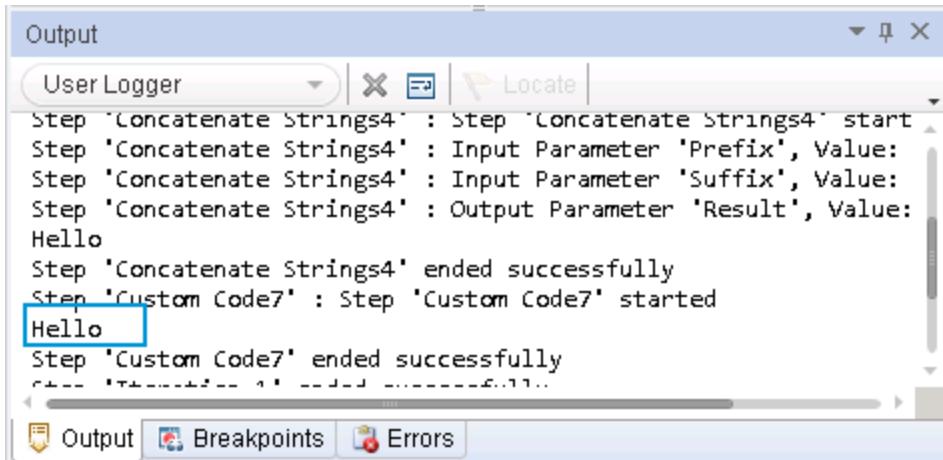
**UserLogger** statements are useful in place of debugging. By viewing the run-time value of the selected property, parameter, or object, you can see the actual value without having to use the more complex debugging features.

1. Select the step for which you want to report information.
2. In the Properties pane, open the **Events** tab .
3. In the Events tab, create an event handler. The **TestUserCode.cs** file opens.
4. In the **TODO** section of the **TestUserCode.cs** file, enter the value you want to report using the following syntax:

```
Context.UserLogger.<user logger info level>("<object/property/parameter>");
```

**IMPORTANT:** It is mandatory to use the **Context** object before the **UserLogger** object, as the **Context** object isolates the actual run-time context of the property, parameter, or object whose value you want to see.

You can then see the value you selected in the Output pane:



### Examples

The following example reports the run-time value of the `CustomerName` property (whose value comes from a data source attached to the test) in a flight booking Web service. In this example the value to be reported was set with the `DataSourceValue` variable.

```
var DataSourceValue = GetDataSource("WebServiceData!Input").GetValue(0,
"CustomerName");
Context.UserLogger.Info(DataSourceValue);
```

## How to Retrieve and Set Test or User Variables

### Relevant for: API testing only

You can use special code in test step events to retrieve or set the value of environment and user variables.

This task includes the following steps:

- "[Prerequisite - create user variables.](#)" on the next page
- "[Optional - set the test profile](#)" on the next page
- "[Retrieve a variable value](#)" on the next page
- "[Set a variable value](#)" on page 794

## Prerequisite - create user variables.

See "Define user variables" on page 558 for details on creating user variables.

**Note:** You may also want to create multiple test profiles to vary the value of the variables for different users or different test runs. For details on creating and editing user profiles, see "Define user variable profiles" on page 559

## Optional - set the test profile

If you are using multiple test profiles, you must set the test profile you want to use before running test:

1. In the canvas, select either the **Start** or **End** steps.
2. In the Properties pane, open the **Test Variables** tab .
3. In the Test Variables tab, choose the profile name from the **Active Profile** drop-down list.

If you have entered default values for any test or user variables, the values for this profile are used in the test run.

## Retrieve a variable value

You can use code to retrieve the value of a user variable during run-time. This is useful to show you the value of the variable without having to start a debugging session.

1. In the canvas, select the step during which you want to retrieve a variable value.
2. In the Properties pane, open the **Events** tab .
3. In the Events tab, create an event handler. The TestUserCode.cs file opens.
4. In the **TODO** section of the TestUserCode.cs file, call the data source value you want to retrieve using the following syntax:

```
this.<activity name>.Context.TestProfile.GetVariableValue("<variable name>"); (if you want to use a specific value from a specific profile)
```

or

```
this.<activity name>.Context.EnvironmentProfile.GetVariableValue("<variable name>");
```

### Example

```
this.concatenateStringsActivity4.Context.TestProfile.GetVariableValue("Prefix");
this.concatenateStringsActivity4.Report("Prefix Variable Value",
this.concatenateStringsActivity4.Context.TestProfile.GetVariableValue
("Prefix"));
```

## Set a variable value

1. In the canvas, select the step during which you want to retrieve a variable value.
2. In the Properties pane, open the **Events** tab .
3. In the Events tab, create an event handler. The TestUserCode.cs file opens.
4. In the **TODO** section of the TestUserCode.cs file, call the data source value you want to retrieve using the following syntax:

```
this.<activity name>.Context.TestProfile.SetVariableValue("<variable name>","<variable value>"); (if you want to enter a value for a specific test profile)
```

or

```
this.<activity name>.Context.EnvironmentProfile.SetVariableValue("<variable name>","<variable value>");
```

### Examples

- The following example retrieves the value of the **TestName** test variable:

```
string testName =
this.StServiceCallActivity4.Context.TestProfile.GetVariableValue
("TestName");
```

- The following example sets the value of the environment variable beforeExecuteStepEvent used to verify that the BeforeExecuteStepEvent event ran in the test step for the activity activity.

```
activity.Context.EnvironmentProfile.SetVariableValue
("beforeExecuteStepEvent","true");
```

# How to Encrypt and Decrypt Passwords

### Relevant for: API testing only

Password fields expect an encrypted string - if you provide an unencrypted string, the authentication will fail.

The **EncryptionMngr** method lets you encrypt and decrypt strings within your events.

This task includes the following steps:

- "[Encrypt the password](#)" on the next page
- "[Decrypt the password - optional](#)" on the next page

## 1. Encrypt the password

Use the activity's context's **EncryptionMngr** method, and select Encrypt from the autocompletion drop-down. The following example encrypts a password.

```
string plainText = "myPassword";
string encryptedText =
this.StServiceCallActivity4.Context.EncryptionMngr.Encrypt(plainText);
```

## 2. Decrypt the password - optional

Use the **Decrypt** method from the autocompletion drop-down.

The following example decrypts a password and validates it against the original string.

```
string decryptedText =
this.StServiceCallActivity4.Context.EncryptionMngr.Decrypt(encryptedText);
bool equalText = decryptedText.Equals(plainText);
```

# API Event Coding - Event Structure Overview

## Relevant for: API testing only

When adding event handlers for your API tests, you can choose from a number of different event handlers. Each of these event handlers runs at a different place in the test step's execution, and likewise can access different properties of the current activity.

For most of the API activities included in UFT, you can choose from the standard event handlers: BeforeExecuteStepEvent, AfterExecuteStepEvent, or CodeCheckpointEvent. The function of these is the same regardless of which test step they are used with or the unique properties for each test step.

For Web service and SOAP Request activities, you can access additional event handlers that enable you to add specific functionalities to the Web service call or SOAP request.

The following sections detail the possible events you can use when adding event handlers:

- [API Event Coding - Standard Event Structure](#) ..... 795
- [API Event Coding - Web Service Event Structure](#) ..... 799

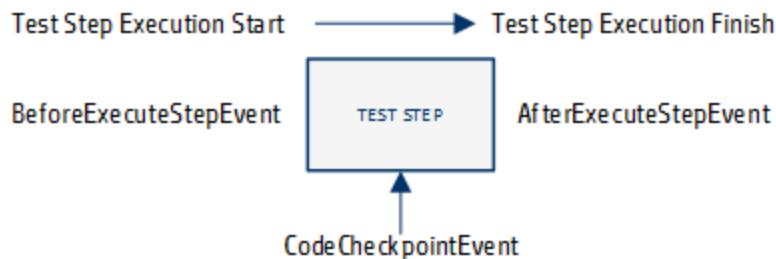
## API Event Coding - Standard Event Structure

## Relevant for: API testing only

For the majority of the activities supported for an API test in UFT, you can only use the standard event handlers:

- BeforeExecuteStepEvent
- AfterExecuteStepEvent
- CodeCheckpointEvent

The following diagram shows how each event works within the individual test step run:



Because of this design, each activity has a different purpose and has access to different properties of the individual test step:

- "BeforeExecuteStepEvent" below
- "AfterExecuteStepEvent" on the next page
- "CodeCheckpointEvent" on page 798

## BeforeExecuteStepEvent

**Purpose:** Set conditions and properties of the step required to make the step run or to handle output from a previous step required in the current step

### Accessible Properties:

- Input properties/parameters from the current activity
- User/test variables from the current test
- Output properties from a previous test step or a parent activity

### Example

```
/// <summary>
 /// Handler for the ConcatenateStringsActivity4 Activity's
 BeforeExecuteStepEvent event.
 /// </summary>
 /// <param name=\"sender\">The activity object that raised the
 BeforeExecuteStepEvent event.</param>
 /// <param name=\"args\">The event arguments passed to the
 activity.</param>
 /// Use this.ConcatenateStringsActivity4 to access the
 ConcatenateStringsActivity4 Activity's context, including input and output
 properties.
```

```
public void ConcatenateStringsActivity4_OnBeforeExecuteStepEvent(object
sender, STActivityBaseEventArgs args)
{
 ExcelFileImportInputArgs a = new ExcelFileImportInputArgs
(@"C:\Users\user\Documents\Unified Functional Testing\API Test
Resources\ConcatenateStrings.xlsx", "Sheet1", true);
 GetDataSource("ConcatenateStrings!Sheet1").ImportFromExcelFile
(@"C:\Users\user\Documents\Unified Functional Testing\API Test
Resources\ConcatenateStrings.xlsx", "Sheet1", true);
 ConcatenateStringsActivity4.Prefix = GetDataSource
("ConcatenateStrings!Sheet1").GetValue(0, "Prefix").ToString();
 ConcatenateStringsActivity4.Suffix = GetDataSource
("ConcatenateStrings!Sheet1").GetValue(0, "Suffix").ToString();
 this.Context.UserLogger.Info (ConcatenateStringsActivity4.Prefix);
 this.Context.UserLogger.Info (ConcatenateStringsActivity4.Suffix);

}
```

## AfterExecuteStepEvent

**Purpose:** Set output properties for the current step or handle the output of the step (to export, save, etc.)

### Accessible Properties:

- Output properties/parameters from the current activity
- User/test variables from the current test
- Any output from the current test step

### Example:

```
/// <summary>
 /// Handler for the FileWriteActivity7 Activity's AfterExecuteStepEvent
event.
 /// </summary>
 /// <param name=\"sender\">The activity object that raised the
AfterExecuteStepEvent event.</param>
 /// <param name=\"args\">The event arguments passed to the
activity.</param>
 /// Use this.FileWriteActivity7 to access the FileWriteActivity7 Activity's
context, including input and output properties.
 public void FileWriteActivity7_OnAfterExecuteStepEvent(object sender,
STActivityBaseEventArgs args)
 {
 ///Event code is here
 String WriteToFileContent =
```

```
this.StServiceCallActivity5.OutputEnvelope.SelectNodes("//*[local-name(.)='Envelope'][1]/*[local-name(.)='Body'][1]/*[local-name(.)='CreateFlightOrder'][1]").ToString();
 this.FileWriteActivity7.Content = WriteToFileContent;
```

## CodeCheckpointEvent

**Purpose:** To set checkpoint properties and values for the current step

### Accessible Properties:

- Input and output values for the current step
- User test/variables from the current test.
- All input and output from the current test step

### Example:

```
/// <summary>
 /// Handler for the CodeActivity15 Activity's CodeCheckPointEvent
 (General execute event for executing code checkpoints) event.
 /// </summary>
 /// <param name="sender">The activity object that raised the
 CodeCheckPointEvent event.</param>
 /// <param name="args">The event arguments passed to the
 activity.</param>
 /// Use args to access the CodeActivity15 Activity's context, including
 any input and output arguments.
 /// <example></example>
 public void CodeActivity15_OnCodeCheckPointEvent(object sender,
 CheckpointEventArgs args)
 {
 ///Event code starts here
 string attachPath = CodeActivity15.Input.attachmentPath;
 string attachmentContent = File.ReadAllText(attachPath);
 string currDate = DateTime.Now.ToString();

 currDate = currDate.Substring(0,currDate.IndexOf(":")) ;
 //args.Checkpoint.Assert.Equals(attachmentContent.Contains("Current Time = "+currDateTime+" file was attached$") ,true);
 //bool status= attachmentContent.Contains(" Current Time =
 "+currDate);
 bool status = attachmentContent.Contains(
 CodeActivity16.Output.currentDateTime);
 bool status1 = attachmentContent.Contains("file was attached$");
 if (status && status1)
 {
 args.Checkpoint.Report(attachmentContent," Current Time =
```

```
"+currDate+" file was attached$","contain", StatusEnum.Succeed);
 }
 else
 args.Checkpoint.Report(attachmentContent," Current Time =
"+currDate+" file was attached$","contain", StatusEnum.Fail);
 }
}
```

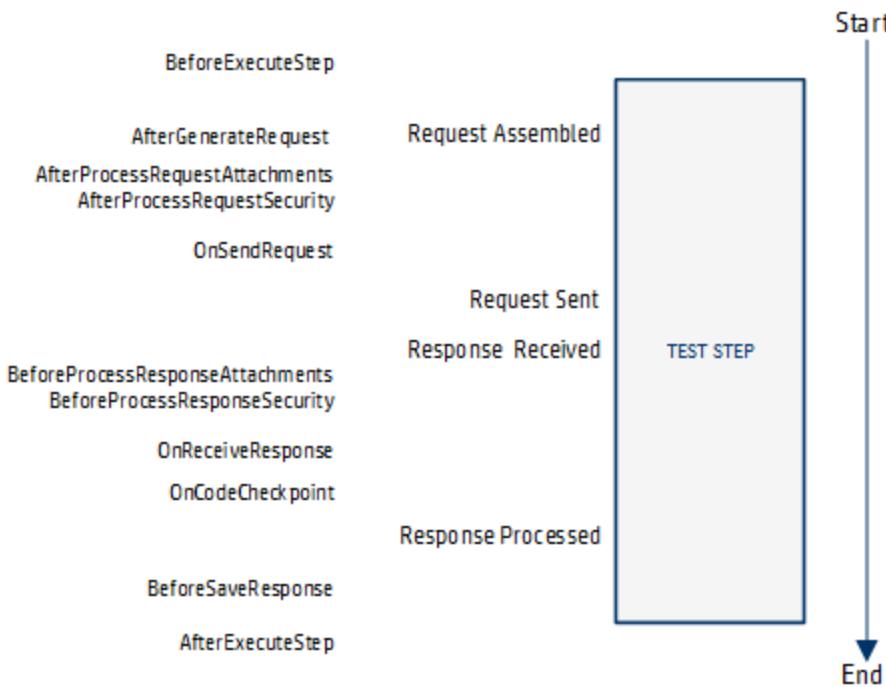
## API Event Coding - Web Service Event Structure

### **Relevant for: API testing only**

When working with a Web service call step or a SOAP Request step, there are a number of additional events available that correspond with the unique run structure of a Web service call:

- BeforeExecuteStepEvent
- AfterExecuteStepEvent
- CodeCheckpointEvent
- AfterGenerateRequest
- AfterProcessRequestSecurity
- AfterProcessRequestAttachments
- OnSendRequest
- OnReceiveResponse
- BeforeProcessResponseAttachments
- BeforeProcessResponseSecurity
- BeforeSaveResponse
- BeforeApplyProtocolSettings

The following diagram shows how each event works within the individual test step run:



However, due to the flow and timing of the various events, you should only create event handlers for specific events:

- "["BeforeExecuteStepEvent" below](#)
- "["AfterExecuteStepEvent" on the next page](#)
- "["CodeCheckpointEvent" on the next page](#)
- "["AfterGenerateRequest" on the next page](#)
- "["AfterProcessRequestSecurity \(WCF services only\)" on the next page](#)
- "["OnReceiveResponse" on page 802](#)
- "["BeforeProcessResponseSecurity \(WCF Security Scenarios only\)" on page 802](#)
- "["BeforeSaveResponse" on page 802](#)

### **BeforeExecuteStepEvent**

**Purpose:** Set conditions and properties of the step required to make the step run or to handle output from a previous step required in the current step

#### **Accessible Properties:**

- Input properties/parameters from the current activity
- User/test variables from the current test
- Output properties from a previous test step or a parent activity

## AfterExecuteStepEvent

**Purpose:** Set conditions and properties of the step required to make the step run or to handle output from a previous step required in the current step

### Accessible Properties:

- Input properties/parameters from the current activity
- User/test variables from the current test
- Output properties from a previous test step or a parent activity
- Response data from the current step
- Response attachments from the current step

## CodeCheckpointEvent

**Purpose:** Set conditions and properties of the step required to make the step run or to handle output from a previous step required in the current step

### Accessible Properties:

- Input properties/parameters from the current activity
- User/test variables from the current test
- Output properties from a previous test step or a parent activity
- SOAP Fault properties

## AfterGenerateRequest

**Purpose:** Set conditions and properties of the step required to make the step run or to handle output from a previous step required in the current step

### Accessible Properties:

- Input properties from the current step
- The input envelope from the current step
- The input attachments from the current step
- Asynchronous properties from the current step

## AfterProcessRequestSecurity (WCF services only)

**Purpose:** Update the request envelope information for Web services using a WCF security scenario with WSE defined. For details on the WCF security scenarios, see "["Security Scenarios Overview" on page 582.](#)

Use the args.Message property to access the response envelope

### Accessible Properties:

- Input envelope information for the current test.

## OnReceiveResponse

**Purpose:** Access the output envelope for the current test for Web services using a Web Service security scenario with WSE defined. For details on the WCF security scenarios, see ["Security Scenarios Overview" on page 582](#).

Use the arg.Message property to access the response envelope

### Accessible Properties:

- The response envelope information for the current step. When this runs, the Web service call step returns a byte array containing the response envelope. You must add event handler code also to use the byte array data.

Use the arg.Message property to access the response envelope

## BeforeProcessResponseSecurity (WCF Security Scenarios only)

**Purpose:** Access the output envelope for the current step for Web services using a WCF security scenario with WSE defined. For details on the WCF security scenarios, see ["Security Scenarios Overview" on page 582](#).

Use the arg.Message property to access the response.

### Accessible Properties:

- The response envelope information for the current step.

## BeforeSaveResponse

**Purpose:** Access the current step's response.

### Accessible Properties:

- The response for the current step. Use the arg.Message property to access the response.

# API Test Event Coding Common Objects

### Relevant for: API testing only

When writing custom code for events in your API test step events, you can use the following objects:

- [Activity Object](#) ..... 803
- [Assert Object](#) ..... 804
- [Checkpoint Object](#) ..... 804
- [CurrentIterationNumber Object](#) ..... 805
- [EncryptionMngr Object](#) ..... 806
- [EnvironmentProfile Object](#) ..... 807

• <a href="#">InputAttachment Object</a>	807
• <a href="#">InputEnvelope Object</a>	809
• <a href="#">OutputAttachment Object</a>	810
• <a href="#">OutputEnvelope Object</a>	812
• <a href="#">Parent Object</a>	813
• <a href="#">TestProfile Object</a>	814
• <a href="#">UserLogger Object</a>	814

## Activity Object

**Relevant for: API testing only**

### Description

Accesses the current activity's properties and arguments.

### Syntax

```
args.Activity.<supported object/method>
```

### Supported Methods

- [Comment object](#)
- [Context object](#)
- [EnableReporting object](#)
- [GetParentLoop method](#)
- [GetRootLoop method](#)
- [Name object](#)
- [Parent object](#)
- [Report method](#)
- [StepEndedAt object](#)
- [StepId object \(read-only\)](#)

## Assert Object

### **Relevant for: API testing only**

#### Description

Qualifies the preceding object.

**Note:** This object should be used only in the CodeCheckpointEvent event.

#### Syntax

```
<object>.Assert.<supported operator method("<expected value>");
```

#### Supported Methods

- Equals
- Greater
- GreaterorEqual
- Less
- LessorEqual
- NotEquals

#### Example

```
args.Checkpoint.Assert.Equals
(this.ConcatenateStringsActivity4.Prefix+this.ConcatenateStringsActivity4.Suffix
,this.ConcatenateStringsActivity4.Result);
```

## Checkpoint Object

### **Relevant for: API testing only**

#### Description

Accesses the current activity's checkpoint properties.

**Note:** This object is only accessible when using the OnCodeCheckpointEvent event.

## Syntax

```
args.Checkpoint.<supported object/method>
```

**Note:** You must use the args object before the Checkpoint object to access the selected activity's checkpoint properties.

## Supported Objects and Methods

- **Assert object**
- **RunUIcheckpoints object**
- **Report method**

## Examples

```
args.Checkpoint.RunUICheckpoints = true;
```

```
if (args.Checkpoint.Assert.Equals(this.ConcatenateStringsActivity4.Result))
 this.ConcatenateStringsActivity4.Report(ConcatenateStringsActivity4.Result,
"Passed");
else
 this.ConcatenateStringsActivity4.Report(ConcatenateStringsActivity4.Result,
"Failed");
```

## CurrentIterationNumber Object

**Relevant for: API testing only**

### Description

Gets the current iteration number.

## Syntax

```
this.<parent activity>.CurrentIterationNumber
```

## Supported Methods

None

## Example

```
var GetFlights_Input_Arrival_City = GetDataSource
("WebServiceData!Input").GetValue(this.Loop2.CurrentIterationNumber-1,
"ArrivalCity").ToString();
StServiceCallActivity4.InputEnvelope.SelectSingleNode("/*[local-name(.)
='Envelope'][1]/*[local-name(.)='Body'][1]/*[local-name(.)='GetFlights'][1]/*
[local-name(.)='ArrivalCity'][1]").InnerText = GetFlights_Input_Arrival_City;
```

## EncryptionMngr Object

### **Relevant for: API testing only**

#### Description

Accesses UFT API testing's encryption mechanism to enable you to encrypt or decrypt strings (such as passwords).

#### Syntax

```
this.<activity>.Context.EncryptionMngr.<supported method>
```

#### Supported Methods

- Decrypt
- Encrypt

## Example

```
string plainText = "myPassword";
string encryptedText =
this.StServiceCallActivity4.Context.EncryptionMngr.Encrypt(plainText);
```

## EnvironmentProfile Object

**Relevant for: API testing only**

### Description

Accesses the environmental variables for a test.

**Note:** If you are using test profiles for a test, you should use the [TestProfile](#) object instead.

### Syntax

```
this.<activity>.Context.EnvironmentProfile.<supported method>
```

### Supported Methods

- GetType
- GetVariablesNames
- GetVariableValue
- SetVariableValue
- ToString

### Example

```
this.ConcatenateStringsActivity4.Context.EnvironmentProfile.GetVariablesNames()
().ToString();
```

## InputAttachment Object

**Relevant for: API testing only**

### Description

Accesses the properties of the test step's input attachments.

### Syntax

```
this.<activity>.Context.InputAttachments.<supported object or method>
```

## Supported Methods

- **Attributes object**
- **BaseURL object**
- **ChildNodes object**
- **CreateAttribute method**
- **CreateCDataSection method**
- **CreateComment method**
- **CreateElement method**
- **CreateEntityReference method**
- **CreateNode method**
- **CreateWhitespace method**
- **CreateXMLDeclaration method**
- **DocumentElement object**
- **DocumentType object**
- **FirstChild object**
- **GetElementsById method**
- **GetElementsbyTagName method**
- **GetEnumerator method**
- **GetNamespaceOfPrefix method**
- **HasChildNodes object**
- **ImportNode method**
- **InnerText object**
- **InnerXML object**
- **InsertAfter method**
- **InsertBefore method**
- **IsReadOnly object**
- **LastChild object**
- **Load method**
- **LoadXML method**
- **LocalName object**
- **Name object**
- **NamespaceURL object**
- **NameTable object**
- **NextSibling object**

- **NodeType object**
- **OwnerDocument object**
- **ParentNode object**
- **Prefix object**
- **PrependChild method**
- **PreserveWhitespace object**
- **PreviousSibling object**
- **ReadNode method**
- **RemoveAll method**
- **RemoveChild method**
- **ReplaceChild method**
- **SchemaInfo object**
- **Schemas object**
- **SelectNodes method**
- **SelectSingleNode method**
- **Supports method**
- **Validate method**
- **Value object**
- **WriteContentTo method**
- **WriteTo method**

## Example

```
this.StServiceCallActivity8.InputAttachments.Load(@"<my file path>");
```

## InputEnvelope Object

**Relevant for: API testing only**

### Description

Accesses the input property envelope for Web Service, HTTP Request, and SOAP Request activities.

### Syntax

```
this.<activity>.InputEnvelope.<supported object/method>
```

## Supported Objects and Methods

This object is a standard object of the XML Document class. All supported objects and methods for the XML Document class can be used with this object.

### Example

```
var GetFlights_Input_Arrival_City = GetDataSource
("WebServiceData!Input").GetValue(this.Loop2.CurrentIterationNumber-1,
"ArrivalCity").ToString();
StServiceCallActivity4.InputEnvelope.SelectSingleNode("/*[local-name(.)='Envelope'][1]/*[local-name(.)='Body'][1]/*[local-name(.)='GetFlights'][1]/*[local-name(.)='ArrivalCity'][1]").InnerText = GetFlights_Input_Arrival_City;
```

## OutputAttachment Object

### **Relevant for: API testing only**

#### Description

Accesses the properties of the test steps output attachments.

#### Syntax

```
this.<activity>.Context.OutputAttachments.<supported object or method>
```

## Supported Methods

- [Attributes object](#)
- [BaseURL object](#)
- [ChildNodes object](#)
- [CreateAttribute method](#)
- [CreateCDataSection method](#)
- [CreateComment method](#)
- [CreateElement method](#)
- [CreateEntityReference method](#)
- [CreateNode method](#)
- [CreateWhitespace method](#)
- [CreateXMLDeclaration method](#)

- **DocumentElement object**
- **DocumentType object**
- **FirstChild object**
- **GetElementsById method**
- **GetElementsbyTagName method**
- **GetEnumerator method**
- **GetNamespaceOfPrefix method**
- **HasChildNodes object**
- **ImportNode method**
- **InnerText object**
- **InnerXML object**
- **InsertAfter method**
- **InsertBefore method**
- **IsReadOnly object**
- **LastChild object**
- **Load method**
- **LoadXML method**
- **LocalName object**
- **Name object**
- **NamespaceURL object**
- **NameTable object**
- **NextSibling object**
- **NodeType object**
- **OwnerDocument object**
- **ParentNode object**
- **Prefix object**
- **PrependChild method**
- **PreserveWhitespace object**
- **PreviousSibling object**
- **ReadNode method**
- **RemoveAll method**
- **RemoveChild method**
- **ReplaceChild method**
- **SchemaInfo object**
- **Schemas object**

- **SelectNodes method**
- **SelectSingleNode method**
- **Supports method**
- **Validate method**
- **Value object**
- **WriteContentTo method**
- **WriteTo method**

## Example

```
this.StServiceCallActivity8.OutputAttachments.Load(@"<my file path>");
```

## OutputEnvelope Object

### **Relevant for: API testing only**

#### Description

Accesses the output envelope information for Web Service, HTTP Request, and SOAP Request activities.

#### Syntax

```
this.<activity>.OutputEnvelope.<supported object/method>
```

#### Supported Methods

This object is a standard object of the XML Document class. All supported objects and methods for the XML Document class can be used with this object.

## Example

```
var GetFlights_Input_Arrival_City = GetDataSource
("WebServiceData!Input").GetValue(this.Loop2.CurrentIterationNumber-1,
"ArrivalCity").ToString();
StServiceCallActivity4.OutputEnvelope.SelectSingleNode("//[local-name(.)
='Envelope'][1]//*[local-name(.)='Body'][1]//*[local-name(.)='GetFlights'][1]/*
[local-name(.)='ArrivalCity'][1]").InnerText = GetFlights_Input_Arrival_City;
```

## Parent Object

**Relevant for: API testing only**

### Description

Accesses the parent activity for a selected activity.

### Syntax

```
this.<activity>.Parent.<supported method>
this.<activity>.Context.Parent.<supported method>
```

**Note:** Using the Context object here enables you to access the run-time context of the selected activity and parent activity.

### Supported Methods

- Activities **object**
- Comment **object**
- Context **object**
- EnableReporting **object**
- GetParentLoop
- GetRootLoop
- Name **object**
- Report **method**
- StepEndedAt **object**
- StepId **object**

### Example

```
string ParentName = this.ConcatenateStringsActivity5.Parent.Name
```

## TestProfile Object

**Relevant for: API testing only**

### Description

Accesses the values of the environment and user variables for the currently active test profile as specified in the Properties pane.

### Syntax

```
this.<activity>.Context.TestProfile.<supported method>
```

### Supported Methods

- GetType
- GetVariableNames
- GetVariableValue
- SetVariableValue

### Example

```
this.ConcatenateStringsActivity4.Context.TestProfile.GetVariableValue("Prefix");
```

## UserLogger Object

**Relevant for: API testing only**

### Description

Reports the specified conditions of the preceding object to the UserLogger build log in the Output pane.

### Syntax

```
this.<activity>.Context.UserLogger.<user logger detail supported method>
```

### Supported Methods

- Info
- InfoFormat

- Debug
- DebugFormat
- Error
- ErrorFormat
- Fatal
- FatalFormat
- Warn
- WarnFormat

For details on the supported methods, see <http://www.codeproject.com/Articles/140911/log4net-Tutorial>.

## Example

```
var Variablenames =
 this.ConcatenateStringsActivity4.Context.EnvironmentProfile.GetVariablesNames();
Context.UserLogger.Info(Variablenames);
```

# API Test Event Coding Common Methods

### **Relevant for: API testing only**

When writing custom code for your API test activities, there are a number of methods that you can use:

• <a href="#">Export Method</a> .....	816
• <a href="#">ExportToExcelFile Method</a> .....	817
• <a href="#">GetDataSource Method</a> .....	818
• <a href="#">GetValue Method</a> .....	819
• <a href="#">GetVariableNames Method</a> .....	820
• <a href="#">GetVariableValue Method</a> .....	821
• <a href="#">Import Method</a> .....	822
• <a href="#">ImportFromExcelFile Method</a> .....	823
• <a href="#">Info Method</a> .....	824
• <a href="#">Report Method</a> .....	825
• <a href="#">SelectSingleNode Method</a> .....	825
• <a href="#">SetValue Method</a> .....	826
• <a href="#">SetVariableValue Method</a> .....	827

**Note:** The **Query** function implemented in versions 11.20 and earlier is not supported in version 11.50 and later. To modify runtime data through an event handler, replace all occurrences of the **Query** function with **GetDataSource**, using the arguments described in this section..

## Export Method

**Relevant for: API testing only**

### Description

Exports the specified Excel data source from your test to an external file.

### Class

DataSource

### Syntax

```
ExcelFileExportInputArgs <name> = new ExcelFileImportInputArgs(@"<path to data
source>");
GetDataSource("<data source name>").Export (name);
```

**Note:** You must cast the data source in the first line of the syntax in order to use the **Export** method. If you do not want to cast the data source, use the **ExportToExcelFile** method.

### Parameters

Parameter	Description
<i>Name</i>	Name assigned to the data source.
<i>Path to data source</i>	The Windows path to the file for the data source.

### Return Type

An Excel file with the specified data in the specified directory.

### Example

```
ExcelFileExportInputArgs a = new ExcelFileExportInputArgs
(@"C:\Users\brojerem\Documents\Unified Functional Testing\API Test
Resources\File.xls");
```

```
GetDataSource("ConcatenateStrings!Sheet1").Export(a);
```

## ExportToExcelFile Method

**Relevant for: API testing only**

### Description

Exports the specified Excel data source from your test to an external file.

### Class

**DataSource**

### Syntax

```
GetDataSource("<data source>").ExportToExcelFile(@"<path to file>");
```

### Parameters

Parameter	Description
<i>Path to data source</i>	The Windows path to the file for the data source.

### Return Type

An Excel file in the specified directory.

### Example

```
GetDataSource("ConcatenateStrings!Sheet1").ExportToExcelFile
(@"C:\Users\brojerem\Documents\Unified Functional Testing\API Test
Resources\File.xls");
```

## GetDataSource Method

**Relevant for: API testing only**

### Description

Accesses the specified data source.

**Note:** This method is typically used when setting a value for a property, parameter, or variable. It is also used in conjunction with other methods which enable you to use the data from the data source, such as **GetValue** or **SetValue**.

### Class

#### Test Entities

### Syntax

```
property value = GetDataSource("<Data source name>").
```

### Parameters

Parameter	Description
<i>Data source name</i>	The name of the data source, exactly as it appears in the Data pane.  <b>Note:</b> You can right-click the data source in the Data pane and select Copy to ensure that you use the correct name in your code.

### Return Type

No explicit return object - method only accesses the data source.

### Example

```
var DataSourceValue = GetDataSource("WebServiceData!Input").GetValue(0,
"CustomerName");
Context.UserLogger.Info(DataSourceValue);
```

## GetValue Method

**Relevant for: API testing only**

### Description

Retrieves a specified value from the selected data source.

**Note:** This method is a method of the data source. In order to use this method to get a data source value, you must retrieve a data source using the [GetDataSource](#) method

### Class

DataSource

### Syntax

```
GetDataSource(<data source name>).GetValue(<row index>, "<column name>");
```

### Parameters

Parameter	Description
<i>Row index</i>	<b>Required.</b> The row from which to take the value.  <b>Note:</b> The row index is zero-based, meaning if you want to pull from the first row of the table, you must set the row index value to 0.
<i>Column name</i>	<b>Required.</b> The name of the column from which to take the value.

### Return Type

Data value (format depends based on the data source type)

### Example

```
var TableDataSourceValue = GetDataSource("WebService Customer Table").GetValue(0, "CustomerName");
```

```
var GetFlights_Input_Departure_City = GetDataSource
```

```
("WebServiceData!Input").GetValue(this.Loop2.CurrentIterationNumber-1,
"DepartureCity").ToString();
StServiceCallActivity4.InputEnvelope.SelectSingleNode("//*[local-name(.)
='Envelope'][1]/*[local-name(.)='Body'][1]/*[local-name(.)='GetFlights'][1]/*
[local-name(.)='DepartureCity'][1]").InnerText = GetFlights_Input_Departure_
City;
```

## GetVariableNames Method

**Relevant for: API testing only**

### Description

Retrieves the names of all variable values in the current test.

### Class

TestProfile

### Syntax

```
this.<activity>.Context.TestProfile.GetVariableNames();
this.<activity>.Context.EnvironmentProfile.GetVariableNames();
```

### Parameters

None

### Return Type

A list of all environment/test variables in the current test or test profile.

### Example

```
this.ConcatenateStringsActivity4.Context.EnvironmentProfile.GetVariablesNames
().ToString();
var Variablenames =
this.ConcatenateStringsActivity4.Context.EnvironmentProfile.GetVariablesNames();
Context.UserLogger.Info(Variablenames);
```

## GetVariableValue Method

**Relevant for: API testing only**

### Description

Retrieves the value of a selected test or environment variable.

### Class

**TestProfile**

### Syntax

```
this.<activity>.Context.TestProfile.GetVariableValue("<variable name>");
this.<activity>.Context.EnvironmentProfile.GetVariableValue("<variable name>");
```

### Parameters

Parameter	Description
<i>Variable name</i>	A string for the name of the variable. You can find this value in the <b>Test Variables</b> tab  in the Properties pane.

### Return Type

No explicit return - the variable value is set for the test run.

### Examples

```
this.ConcatenateStringsActivity4.Context.TestProfile.GetVariableValue("Prefix");
this.ConcatenateStringsActivity4.Report("Prefix Variable Value",
this.ConcatenateStringsActivity4.Context.TestProfile.GetVariableValue
("Prefix"));
```

```
this.ConcatenateStringsActivity4.Context.EnvironmentProfile.GetVariableValue
("Prefix");
this.ConcatenateStringsActivity4.Report("Prefix Variable Value 2",
this.ConcatenateStringsActivity4.Context.EnvironmentProfile.GetVariableValue
("Prefix"));
```

## Import Method

**Relevant for: API testing only**

### Description

Imports the specified Excel data source into your test.

### Class

DataSource

### Syntax

```
ExcelFileImportInputArgs <name> = new ExcelFileImportInputArgs(@"<path to data
source>", "<data source name in the test>", "header row");
GetDataSource("<data source name>").Import (name);
```

**Note:** You must cast the data source in the first line of the syntax in order to use the `Import` method. If you do not want to cast the data source, use the [ImportfromExcelFile](#) method instead.

### Parameters

Parameter	Description
<i>Name</i>	Name assigned to the data source.
<i>Path to data source</i>	The Windows path to the file for the data source.
<i>Data source name in the test</i>	The name the test gives the data source after importing.
<i>Header row</i>	A boolean value if the data source has a header row. Possible values are "true" or "false".

### Return Type

An Excel data source added to your test.

### Example

```
ExcelFileImportInputArgs a = new ExcelFileImportInputArgs
(@"C:\Users\user\Documents\Unified Functional Testing\API Test
```

```
Resources\ConcatenateStrings.xlsx", "Sheet1", true);
GetDataSource("ConcatenateStrings!Sheet1").Import(a);
```

## ImportFromExcelFile Method

**Relevant for: API testing only**

### Description

Imports the selected Excel file to your test.

### Class

DataSource

### Syntax

```
GetDataSource("<data source name>").ImportFromExcelFile(@"<path to Excel file>",
"<test data source name>", header row);
```

### Parameters

Parameter	Description
<i>Name</i>	Name assigned to the data source.
<i>Path to data source</i>	The Windows path to the file for the data source.
<i>Data source name in the test</i>	The name the test gives the data source after importing.
<i>Header row</i>	A boolean value if the data source has a header row. Possible values are "true" or "false".

### Return Type

Adds an Excel data source to your test.

### Example

```
GetDataSource("ConcatenateStrings!Sheet1").ImportFromExcelFile
(@"C:\Users\user\Documents\Unified Functional Testing\API Test
Resources\ConcatenateStrings.xlsx", "Sheet1", true);
```

## Info Method

**Relevant for: API testing only**

### Description

Reports the selected information to the UserLogger build log in the Output pane.

**Note:** You can also use other common .NET logging options. For details, see <http://www.codeproject.com/Articles/140911/log4net-Tutorial>.

### Class

ILog

### Syntax

```
this.<activity>.Context.UserLogger.Info(message)
```

**Note:** This method must always be used with a **UserLogger** object.

### Parameters

Parameter	Description
<i>Message</i>	A string or value to report. You can use other code strings in this parameter to report the run-time value of any object, property, parameter, or variable in run-time.

### Return Type

A message displayed in the UserLogger build log in the output pane.

### Example

```
this.concatenateStringsActivity4.Context.EnvironmentProfile.GetVariablesNames()
().ToString();
var Variablenames =
this.concatenateStringsActivity4.Context.EnvironmentProfile.GetVariablesNames();
Context.UserLogger.Info(Variablenames);
```

## Report Method

### **Relevant for: API testing only**

#### Description

Reports a custom attribute in the Captured Data pane of the run results.

#### Class

#### Action

#### Syntax

```
this.<activity>.Report("report attribute string", "<attribute value>");
```

#### Parameters

Parameter	Description
<i>Report attribute string</i>	A string for the custom field to enter into the run results.
<i>Attribute value</i>	The value for the attribute to report.

#### Return Type

A custom field in the run results.

#### Example

```
var ActivityArguments = args.Activity.StepId.ToString();
this.ConcatenateStringsActivity4.Report("Overall report", ActivityArguments);
```

## SelectSingleNode Method

### **Relevant for: API testing only**

#### Description

Selects a single node from an array containing input/output properties.

## Class

This method is not part of a UFT API testing class, but is invoked by the **InputEnvelope** object.

## Syntax

```
<activity>.InputEnvelope.SelectSingleNode("<fully qualified Xpath>").<supported
object/method>
```

## Parameters

Parameter	Description
<i>Fully qualified XPath</i>	The XPath to the property node in the array. Right-click the property/parameter name in the <b>Input/Checkpoints</b> tab  in the Properties pane and select <b>Copy Fully Qualified XPath</b> to retrieve this information.

## Return Type

No explicit return - enables the test to access the property or parameter.

## Example

```
var GetFlights_Input_Departure_City = GetDataSource
("WebServiceData!Input").GetValue(this.Loop2.CurrentIterationNumber-1,
"DepartureCity").ToString();
StServiceCallActivity4.InputEnvelope.SelectSingleNode("//[local-name(.)
='Envelope'][1]/*[local-name(.)='Body'][1]/*[local-name(.)='GetFlights'][1]/*
[local-name(.)='DepartureCity'][1]").InnerText = GetFlights_Input_Departure_
City;
```

## SetValue Method

**Relevant for: API testing only**

### Description

Sets the value of a specified item in the data source.

**Notes:**

- This method must be used with the **GetDataSource** method.
- This method cannot be used for XML data sources.

## Class

**DataSource**

## Syntax

```
GetDataSource(<data source name>).SetValue(<row index>, "<column name>");
```

## Parameters

Parameter	Description
<i>Row index</i>	<b>Required.</b> The row from which to take the value.  <b>Note:</b> The row index is zero-based, meaning if you want to pull from the first row of the table, you must set the row index value to 0.
<i>Column name</i>	<b>Required.</b> The name of the column from which to take the value.

## Return Type

No explicit return object. Sets a value in the specified data source.

## Example

```
GetDataSource("ConcActivity Strings").SetValue(1, "Suffix", "done.");
```

## SetVariableValue Method

**Relevant for: API testing only**

## Description

Sets the value of a selected test or environment variable.

## Class

**TestProfile**

## Syntax

```
this.<activity>.Context.TestProfile.SetVariableValue("<variable name>" , "<variable value>");
this.<activity>.Context.EnvironmentProfile.SetVariableValue("<variable name>" , "<variable value>");
```

## Parameters

Parameter	Description
<i>Variable name</i>	A string for the name of the variable. You can find this value in the <b>Test Variables</b> tab  in the Properties pane.
<i>Variable value</i>	The value for the variable. The format differs on the expected use of the variable.

## Return Type

No explicit return - the variable value is set for use in the test run.

## Example

```
var activity = ((HP.ST.Ext.WebServicesActivities.StServiceCallActivity)(
args.Activity));
activity.Report("codeCheckPointEvent" , "code CheckPoint event");
args.Checkpoint.Report("CheckPoint","CheckPoint","=" ,
HP.ST.Fwk.RunTimeFWK.CheckpointFWK.StatusEnum.Succeed);
activity.Context.EnvironmentProfile.SetVariableValue(
"codeCheckpointEvent","true");
```

## Troubleshooting and Limitations - Using Data in API Tests and Components

### Relevant for: API testing only

This section describes troubleshooting and limitations for working with data.

<b>General</b>	<ul style="list-style-type: none"> <li>If a specified data source is or becomes inaccessible, the test will not fail. The Errors pane and report, however, indicate that there was an error in retrieving the data.</li> <li>Keywords such as #SKIP#, and so forth, are not supported in the Input property grid.</li> </ul> <p><b>Workaround:</b> Link to a data source that contains the keyword.</p>
----------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	<ul style="list-style-type: none"><li>When adding a referenced Excel data source, if the file requires special credentials (for example, a location on another domain or a drive requiring authentication), you must verify that the operating system will allow access to the file.</li><li>Data sources with parent-child relationships, should not be accessed together in the same loop, unless the child data source is used for data-driving array elements. Accessing parent-child data sources in the same loop, may corrupt the test.</li><li>Linking to file names is not supported for <b>HTTP Request</b> and <b>HTTP Receiver</b> activities that use the <b>File</b> type message body.</li><li>For data sources with child relations: If you change the name the Key column in the child sheet, the Define New/Edit Data Relation dialog box does not reflect the new column name in the drop down lists.</li></ul>
<b>Data driving</b>	<ul style="list-style-type: none"><li>Data driving for JSON requests or responses, is only supported for Excel.</li><li>Data driving for an Excel data source is only effective for the first 254 properties of a step. If a step has more than 254 properties, they will not be data-driven.</li><li>Data driving for an Excel data source is only supported for property values consisting of 255 characters or less. If a property value has more than 255 characters, the data driving mechanism offers to truncate the string.</li><li>When data driving a test step that has an array with two or more nested levels, the data driving engine only copies the first element of each array to the Excel data tables.</li><li>Keyword data driving to an XML data source is not supported.</li></ul>

# Part 8: UFT Running and Debugging Operations

# Chapter 55: Running Tests and Components

**Relevant for: GUI tests and components and API testing**

This chapter includes:

• <a href="#">Running Tests and Components - Overview</a> .....	832
• <a href="#">Running GUI Tests with a Disconnected Remote Desktop Connection</a> .....	833
• <a href="#">Command Line Syntax for Running API Tests</a> .....	834
• <a href="#">How to Run a Test or Component</a> .....	835
• <a href="#">How to Run a GUI Test with a Disconnected Remote Desktop Connection</a> .....	838
• <a href="#">UFT Runtime Engine - Overview</a> .....	840
• <a href="#">Test Batch Runner Overview</a> .....	842
• <a href="#">How to Create and Run a Test Batch</a> .....	842
• <a href="#">How to Run a Test Batch Using the Windows Command Line</a> .....	844
• <a href="#">Using UFT for Continuous Integration</a> .....	845
• <a href="#">Optional Steps</a> .....	846
• <a href="#">Default Optional Steps</a> .....	846
• <a href="#">Log Tracking</a> .....	847
• <a href="#">Manually configure log tracking settings</a> .....	847
• <a href="#">How to Set Optional Steps</a> .....	849
• <a href="#">Troubleshooting and Limitations - Run Sessions</a> .....	849

# Running Tests and Components - Overview

## Relevant for: GUI tests and components

When you run a test or component, UFT performs the steps it contains. If you have defined test or component parameters, UFT prompts you to enter values for them. When the run session is complete, UFT displays a report detailing the results.

For details about running business process tests, see "[How to Create, Maintain, and Run Business Process Testing Tests and Flows in UFT](#)" on page 981.

You can run UFT tests from a number of different places:

- [ALM](#)
- [UFT Runtime Engine](#)
- [Test Batch Runner](#)
- [Silent Test Runner](#)

Running tests differs slightly when running GUI tests/components or API tests/components:

## GUI tests and components

UFT always runs a test or component from the first step, unless you specify otherwise. You can:

Method of Running	Description
<b>Run the entire test or component from the beginning</b>	UFT runs each step, starting from the first step in the first action, in sequential order.
<b>Run only a part of a test</b>	You can run sections of a test using these commands: <ul style="list-style-type: none"><li>• <b>Run from Action:</b> Runs the test from the specified action to the end of the test.</li><li>• <b>Run to Action:</b> Runs the test from the beginning of the test to a specific action</li><li>• <b>Run from Step:</b> Runs the test from a specified step to the end of the test.</li><li>• <b>Run to Step:</b> Runs the test from the beginning of the test to the specified step</li></ul>
<b>Run an iteration of a single action (tests only)</b>	Using the <b>Run Current Action</b> command, you can run a single iteration of an individual action.  <b>Note:</b> If the action contains nested actions, UFT runs the nested actions for the defined number of iterations.
<b>Debug a section of a test or component</b>	If you need to debug a section of your test, you can use the following commands to narrow down the scope of the debugging: <ul style="list-style-type: none"><li>• <b>Debug from Step:</b> Starts the run/debug session from a specified step.</li><li>• <b>Debug from Action:</b> Starts the run/debug session from the first step of the selected action.</li></ul>

	<b>Note:</b> Make sure the application is open to the relevant section when using these commands.
<b>Update your test or component to change test object descriptions</b>	You can use the <b>Update Run</b> command to run the test in update mode, which enables you to update test object descriptions, or checkpoint values and Active Screen images/values (when running a test), For additional details, see " <a href="#">"Maintain GUI Tests or Components " on page 133</a>
<b>Run a set of tests</b>	Using the Test Batch Runner, you can create a batch of tests and then run all of these tests together in a step.
<b>Skip certain steps if necessary</b>	You can designate certain steps as optional, which enables UFT to skip them if they fail.
<b>Specify the number of iterations for a test or action (tests only)</b>	If your test contains data parameters stored in the Global data sheet, by default UFT runs an iteration for each row in the Global data sheet. Likewise, if a test action references an individual action data sheet, UFT by default runs an iteration of the action for each row in the Action data sheet. In the Run Pane of the Test Settings dialog box or the Run tab of the Action Call Properties dialog box, you can specify whether to run one iteration, an iteration for each row, or a selected number of iterations.

If you run GUI tests on a remote machine, you can also run your GUI tests via the Windows Remote Desktop connection with a disconnected Remote Desktop Connection. For details, see "["Running GUI Tests with a Disconnected Remote Desktop Connection" below](#)

## API tests and components

UFT runs the test steps contained in the test flow in sequence, unless specified otherwise. You can:

<b>Debug the test</b>	If you have inserted breakpoints in your custom code or event handler code, UFT pauses the test run to enable you to debug the code.  In order to use debugging, you must run the test in Debug mode. This option is set in the API Testing <b>General</b> pane of the Options Dialog Box.
<b>Stop the test run if a checkpoint fails</b>	In the Test Settings tab for the test, you can instruct UFT to stop a test run if a test fails.
<b>Run a set of tests</b>	Using the Test Batch Runner, you can create a batch of tests and then run all of these tests together in a step.

## Running GUI Tests with a Disconnected Remote Desktop Connection

### Relevant for: GUI tests only

Using options set in the Run Sessions pane of the Options dialog box (**Tools > Options > General tab > Run Sessions** node), you can run a GUI test via the Remote Desktop Connection if the Remote Desktop session is closed.

When running a GUI test with Remote Desktop Connection, you work in the following manner:

1. On your local machine, you open the Remote Desktop connection and connect to the remote computer running UFT.
2. On the remote computer, you open UFT and run the test.
3. You close the Remote Desktop connection on the local computer. The UFT session and test on the Remote computer continue to run until completion.

By enabling the options for the Remote Desktop connection, you can free your local computer for other tasks, as well as not need to keep your local computer running and connected to the remote computer.

For task details, see "[How to Run a GUI Test with a Disconnected Remote Desktop Connection](#)" on page [838](#)

## Command Line Syntax for Running API Tests

### **Relevant for: API testing only**

You can also run API tests using the ServiceTestExecuter.exe application, located in the product's bin directory.

The following table describes the command line options for ServiceTestExecuter.exe:

Parameter name	Description
<b>-test</b>	The full path of the test (required). Specify the test directory—not the solution directory.
<b>-inParams</b>	The full path of an XML file containing the input property values (optional).
<b>-outParams</b>	The full path of an XML file containing the output property values (optional).
<b>-profile</b>	The name of an Test profile (optional). For details, see " <a href="#">How to Define API Test Properties or User/System Variables</a> " on page <a href="#">558</a> .
<b>-report</b>	The directory in which to store the report.

**Tip:** To retrieve a list of all of available parameters, run ServiceTestExecuter.exe without any parameters.

The following example runs Test1 using input properties from inParams.xml and output properties from outParams.xml:

```
%ProgramFiles%\HP\Unified Functional Testing\bin> ServiceTestExecuter.exe -test "c:\MyTests\Test1\" -inParams "c:\MyData\inParams.xml" -outParams "c:\MyData\outParams.xml" -profile Profile1
```

where the input property file, InParams.xml, has, for example, this structure:

```
<TestParameters>
 <Values>
 <Arguments>
 <a>1
 2
 </Arguments>
 </Values>
</TestParameters>
```

**Note:** To run a test from the command line, you must save and run the test at least once.

## How to Run a Test or Component

### Relevant for: GUI tests and scripted GUI components

This task describes the various ways in which you can run a test or component.

This task includes the following steps:

- "Prerequisites" below
- "Set the number of iterations for the test" on the next page
- "Run an entire test or component" on the next page
- "Run to a selected step or action (GUI testing only)" on the next page
- "Run a single action, test, or a component from a selected step (GUI testing only)" on page 837
- "Interrupt a run session" on page 838
- "View the run results" on page 838

### Prerequisites

1. Do one of the following:

- **For GUI tests and components:** Ensure that any required UFT add-ins are loaded in the Add-in Manager when you open UFT.
- **For API tests and components:** Set the run mode as Release or Debug in the API Testing General pane in the Options dialog box.

**Note:** **Release** mode runs the test more quickly, as it does not load the debugging mechanism.

2. Open the test or component you want to run.

## Set the number of iterations for the test

Do one of the following:

<b>For GUI tests</b>	In the <b>Run</b> tab of the Test Settings dialog box, specify the number of iterations: <ul style="list-style-type: none"> <li><b>Run one iteration only.</b> Runs the test only once, using only the first row in the global data table.</li> <li><b>Run on all rows.</b> Runs the test with iterations using all rows in the global data table.</li> <li><b>Run from row __ to row __.</b> Runs the test with iterations using the values in the global data table for the specified row range.</li> </ul>
<b>For API tests</b>	<ol style="list-style-type: none"> <li>1. In the canvas, select the <b>Test Flow</b> or <b>Loop</b> box</li> <li>2. In the Properties Pane, open the <b>Input/Checkpoints</b> tab .</li> <li>3. Set the number of iterations.</li> </ol>

## Run an entire test or component

1. Open the Run dialog box in one of the following ways:
  - Click the **Run** button.
  - Select **Run > Run**.

The Run dialog box opens.

2. In the Run dialog box, choose where to save the run session results, and define any input parameters you want to use.

**Note:** (For tests) When running part of a test within the scope of an action, you need to specify the action's parameters, not the test parameters, in the Input Parameters tab of the Run dialog box.

3. Click **OK**. The Run dialog box closes and the run session starts.

**Note when using GUI testing:** When running a test with external resource files (like function libraries, data tables, recovery scenarios,etc.) that are saved in ALM, the resource files are not refreshed for each test run. As a result, any changes made to these external resource files during the current session are not reflected in a test run until you close and reload test and its resource files.

## Run to a selected step or action (GUI testing only)

1. Do one of the following:

<b>For tests</b>	<ul style="list-style-type: none"> <li>• Select <b>Run &gt; Run to Step</b>.</li> <li>• Right-click a step and select <b>Run to Step</b>.</li> </ul>
------------------	------------------------------------------------------------------------------------------------------------------------------------------------------

	<ul style="list-style-type: none"> <li>Right-click an action in the canvas and select <b>Run to Action</b>.</li> </ul>
<b>For components</b>	Select <b>Run &gt; Run to Step</b> .

- In the Run dialog box, choose where to save the run session results, and define any input parameters you want to use.

**Note:** (For tests) When running part of a test within the scope of an action, you need to specify the action's parameters, not the test parameters, in the Input Parameters tab of the Run dialog box.

The run starts at the beginning of the test or component and pauses at the selected step.

### Run a single action, test, or a component from a selected step (GUI testing only)

- Make sure your application is in a state matching the step or action you want to run.
- Select the step or action where you want to start running the test or component
  - In the test flow canvas, select the action.
  - In the Keyword View, highlight a step or action row.
  - In the Editor, place your cursor in a line of VBScript.

**Note:** Make sure that the step or action you choose is not dependent on previous steps, such as a retrieved value or a parameter defined in a previous step.

- Do one of the following:

<b>For tests</b>	<ul style="list-style-type: none"> <li>Select <b>Run &gt; Run from Step</b>.</li> <li>Select <b>Run &gt; Run Current Action</b>.</li> <li>Right-click the step and select <b>Run from Step</b>.</li> <li>Right-click the action in the canvas and select <b>Run from Action</b>.</li> </ul>
<b>For components</b>	Select <b>Run &gt; Run from Step</b>

- In the Run dialog box, choose where to save the run session results, and define any input parameters you want to use.

**Note:** (For tests) When running part of a test within the scope of an action, you need to specify the action's parameters, not the test parameters, in the Input Parameters tab of the Run dialog box.

## Interrupt a run session

Do one of the following:

- Click the **Pause** button  in the toolbar or select **Run > Pause**. The run pauses. To resume running a paused run session, click the **Run** button or select **Run > Run**. (Before using the **Pause** button, select **Run test in debug mode** in the **Options > API Testing** tab > **General** node.)
- Click the **Stop** button, select **Run > Stop**, or press **F4**.
- Perform a file operation (for example, open a different test or component or create a new test or component).

## View the run results

By default, when the run session ends, the run results opens.. .

**Note:** If you cleared the **View results when run session ends** check box in the **Run Sessions** pane of the Options dialog box (**Tools > Options > General** tab > **Run Sessions** node), the run results do not open at the end of the run session.

You can also optionally automatically upload your run results to ALM if you are running a test from ALM. This option is set in ALM as a site parameter for your project. For details, see the *HP Application Lifecycle Management Administrator Guide*.

## How to Run a GUI Test with a Disconnected Remote Desktop Connection

### Relevant for: GUI tests only

This task describes how to run a GUI test after disconnecting an Remote Desktop Connection to a computer running UFT. This enables you to start a test run and disconnect your computer from the UFT computer, enabling the test to run independently while you continue work on your own computer.

**Note:** This feature is not supported in the Microsoft Windows® XP environment or the Hyper-V virtualization server.

This task includes the following steps:

- "Prerequisites " on the next page
- "Log in to the remote computer running UFT" on the next page
- "Configure the Remote Desktop Connection options in the remote computer" on the next page
- "Configure the Windows Task Scheduler options for automation runs - optional" on the next page
- "Start the test and disconnect from the Remote Desktop connection" on page 840

## 1. Prerequisites

If you want to run UFT in a minimized RDP session, and you are using an RDP 6.0 or later client, enable this by setting a registry value on the computer that is running the Remote Desktop session

- a. If necessary, create the `RemoteDesktop_SuppressWhenMinimized` registry value (DWORD type) in one of the following registry paths on the computer that is running the RDP client:
  - **For 32-bit operating systems:** <HKEY\_CURRENT\_USER or HKEY\_LOCAL\_MACHINE>\Software\Microsoft\Terminal Server Client
  - **For 64-bit operating systems:** <HKEY\_CURRENT\_USER>\Software\Wow6432Node\Microsoft\Terminal Server Client
- b. Set the data for this value to **2**.
- c. Restart your remote session in order for this setting to take effect.

## 2. Log in to the remote computer running UFT

On your local computer, open a Windows Remote Desktop Connection session and connect to the remote computer running UFT.

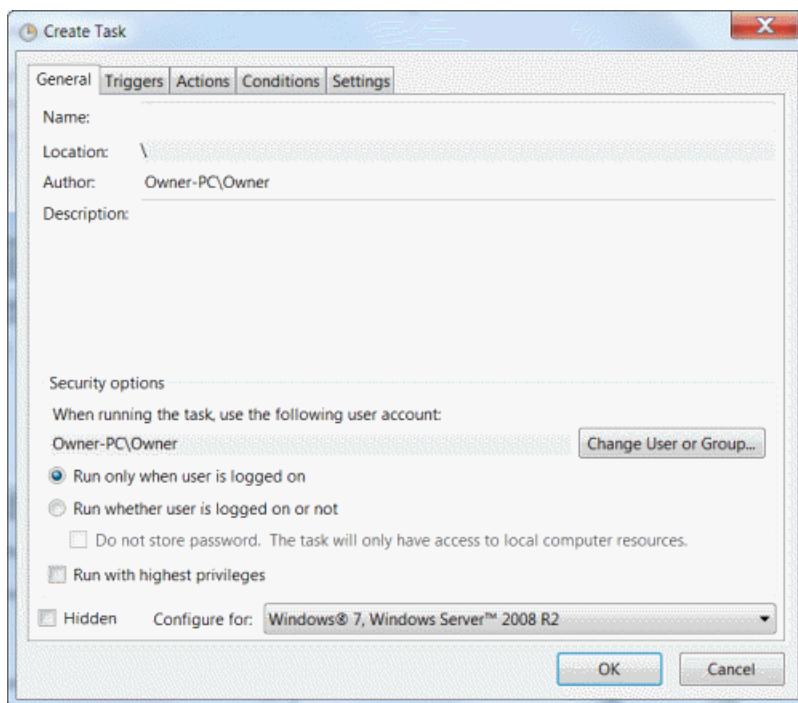
## 3. Configure the Remote Desktop Connection options in the remote computer

You must configure the Remote Connection options before beginning to run the test:

- a. In the Options dialog box, open the **Run Sessions** pane (**Tools > Options > General tab > Run Sessions** node).
- b. Enable the Remote Desktop Connection options by selecting the **Allow UFT to continue running GUI or business process tests after disconnection from an RDP computer** option.
- c. Enter the user name and password of the user connecting to the remote computer with the Remote Desktop connection.

## 4. Configure the Windows Task Scheduler options for automation runs - optional

If you are running the test via automation, in the **General** tab of the Windows Task Scheduler dialog box, ensure that the **Run only when user is logged on** option is selected:



## 5. Start the test and disconnect from the Remote Desktop connection

- On the remote computer, start the test run.
- On your local computer, close your Remote Desktop Connection session.

**IMPORTANT:** You must remain logged on to the computer running UFT.

The test run continues on the remote computer until completion.

**Note:** For a UFT server that allows multiple users to log in with the same username/password: When a user initiates a test run and disconnects from RDP during the run, the run execution goes to the background and the user has no access to the execution when they log in again, because the test is executed in another session.

## UFT Runtime Engine - Overview

### Relevant for: GUI tests and components, API testing, and business process tests and flows

The UFT Runtime Engine enables you to run UFT tests (both GUI and API) and business process tests on your computer without installing the entire UFT IDE. In addition, you can also install the Runtime Engine without the Run Results Viewer, UFT Add-in for ALM, sample applications, or Help documentation. This can potentially save you valuable disk space on your computer.

When you run tests with the Runtime Engine, you can access and run the test from a number of different places without the need to open the UFT interface and configure UFT options. When the test runs it runs in the background. At the end of the test run, you can view the test results.

Using the Runtime Engine requires very little experience in using UFT - you do not need to edit tests, change configurations or settings, or understand how to make UFT work with your application. You simply select the test, run the test, and view the run results.

The Runtime Engine can be used in a number of different scenarios:

<b>Running tests and components from ALM</b>	You can set up test runs from the Test Lab module in ALM, and run these on a computer using the Runtime Engine. Using the Runtime Engine enables you to run the test, without the need to interact with the UFT interface (such as loading UFT add-ins in the Add-in Manager dialog box).
<b>Running tests from automation:</b>	You also can run tests using automation using the Runtime Engine. The Runtime Engine installation enables you to save disk space on the computer running these tests, freeing up system resources on that computer for other tasks
<b>Running tests using the Jenkins plug-in:</b>	The Runtime Engine can be installed on a build server or computer running builds of your applications. Using the Jenkins plug-in, you run a UFT test as a post-build action of your application's build process. Having the Runtime Engine installed on this computer to run the UFT test enables you to free up system resources for the important application build tasks.
<b>Running tests using external UFT tools</b>	When you install the Runtime Engine, you also have external tools which enable you to run UFT tests locally, including the Test Batch Runner and the Silent Test Runner. These tools enable you to run a test locally against your application as it is developed, and view the results instantly after the test run. Because the Runtime Engine does not enable you to edit a test, this version of the UFT installation can be used by your application's developers and QA on an ongoing basis to provide regular testing of the application throughout the development process.

The Runtime Engine also supports all the UFT Add-ins as the the full UFT IDE, so you can run tests using any supported technology using the Runtime Engine. All objects and methods for all UFT Add-ins are supported for use with the Runtime Engine.

As part of running a test, you can set specific run-time options. These options are set in the Runtime Engine Settings dialog box, available from the Start Menu (**Start > All Programs > HP Software > HP Unified Functional Testing > Tools > Runtime Engine Settings**):

<b>Add-ins</b>	You can specify add-ins to be loaded.
<b>Run options</b>	You can specify how the Runtime Engine runs tests, including the format of the run results, whether the run results are opened automatically after a test run, and if the Runtime Engine takes screen captures or movies of the run session.
<b>Remote connection options</b>	You can specify if other HP applications are permitted to run tests on this computer using the Runtime Engine or specify how another computer can run tests via Remote Desktop Connection.
<b>Run result export options</b>	You can specify how and what the Runtime Engine should export from the run results after a run session.
<b>Text Recognition</b>	You can specify how the Runtime Engine works with text in your application when running a GUI test.

options	
<b>Web and Windows Application options</b>	You can specify how the Runtime Engine runs tests for specific scenarios against a Web application or Windows application.

## Test Batch Runner Overview

### Relevant for: GUI testing and API testing

Test Batch Runner enables you to run tests in a collective, successive test run. Tests are run individually but sequentially in a single session.

You can use Test Batch Runner only on tests stored on the file system. You cannot use Test Batch Runner for tests stored in ALM.

Using Test Batch Runner, you create a list of tests and save the list as a batch .mtb file, so that you can run the same batch of tests again at another time. You can choose to include or exclude a test in your batch list from running during a particular batch run without affecting the other tests in the batch.

You can add tests individually to Test Batch Runner by navigating to the folder in which the test is located. Test batches can also be added to another test batch by adding an .mtb test batch file to a new test batch. When Test Batch Runner opens, it checks to make sure that all tests within a test batch exist.

When running the test batch, the Output pane allows you view the results of the test run in run time. During the test batch run, the Output pane displays the test's path in the file system, the progress of the test, as well as any errors that occur during the run.

Following the test batch run, the results are saved to a run results file including whether the test passed or failed and errors in running the test. The **Report** column of the **Tests** pane displays a link to the run results file.

If you do not want to run the test batch through the Test Batch Runner interface, you can run the Test Batch Runner from the Windows Command Line. You provide the location of a .mtb file, a test folder, or a directory of tests as a command argument and Test Batch Runner runs the test batch and displays the run results.

**Tip:** Using the command line options of the Test Batch Runner, you can include UFT tests in as part of build runs in a continuous integration system.

## How to Create and Run a Test Batch

### Relevant for: GUI tests and components and API testing

This task explains how to create and run a test batch using Test Batch Runner.

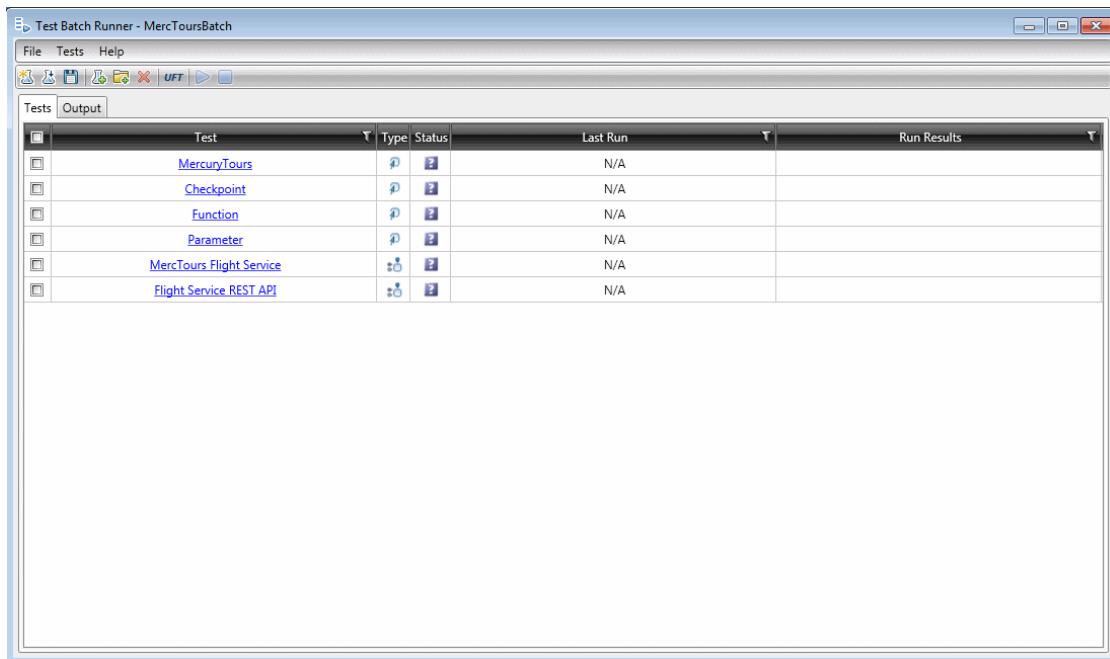
This task includes the following steps:

- "Open Test Batch Runner" below
- "Access the test batch file" below
- "Add batches or tests" on the next page
- "Select the tests to be part of the test batch run" on the next page
- "Save the test batch" on the next page
- "Run the test batch" on the next page
- "View the test batch run results" on the next page

## 1. Open Test Batch Runner

Select **Start > All Programs > HP Software > HP Unified Functional Testing > Tools > Test Batch Runner** or <UFT installation folder/bin/UFTBatchRunner.exe.

This opens a separate window for the Test Batch Runner program.



**Note:** You do not need to have UFT open to use Test Batch Runner.

## 2. Access the test batch file

- To create a new test batch file, select **File > New** or click the **New** batch file button .
- To open an existing batch file, select **File > Open** or click the **Open** batch file button . In the Open dialog box, navigate to the folder in which the batch file is found and click **Open**. The tests from the opened batch file are added to the Test Batch Runner main window.

### 3. Add batches or tests

- To add a test batch file (.mtb), select **File > Add** or click the **Add** button  . Navigate to the folder in which the batch file is saved.
- To add individual tests, select **Tests > Add** or click the **Add** button  . In the **Browse For Folder** dialog box, select the folder in which your tests are located. All the tests from the selected folder are added to the **Tests** pane in the main Test Batch Runner window.

**Note:** When adding tests through the **Tests > Add** menu command, you must select all the tests from the target folder. If you do not want to run all the tests in the target folder, select the check boxes next to the tests you want to run before you run the test batch.

### 4. Select the tests to be part of the test batch run

Select the checkboxes  for the tests that you want to include in the test batch run.

### 5. Save the test batch

Select **File > Save** or click the **Save** button  in the toolbar to save the test batch with the included test. Give your batch file a meaningful name and assign it to a place in your directory.

### 6. Run the test batch

Click the **Run** button  to run the test batch. The Output pane provides run log details of the batch run while the batch is running.

### 7. View the test batch run results

In the Tests pane, click the results link for a specific test in the **Run Results** column. This opens the results for that test.

## How to Run a Test Batch Using the Windows Command Line

### Relevant for: GUI tests and components and API testing

This task describes how to run a test using the Windows Command Line.

This task includes the following steps:

- "Open the Windows Command Line window" on the next page
- "Provide the source folder for the batch file or tests" on the next page
- "Run the test batch" on the next page
- "View the test batch run results (API testing only)" on the next page

## 1. Open the Windows Command Line window

Run cmd.exe to open the Command Line window. (For example, from the Windows Run dialog box.)

## 2. Provide the source folder for the batch file or tests

In the Command Line window, enter UFTBatchRunnerCMD.exe and the **source** switch followed by the test batch file (.mtb) or folder containing the test.

For example, your command line might contain text like this:

```
UFTBatchRunnerCMD.exe -source "C:\users\MySample.mtb"
UFTBatchRunnerCMD.exe -source "C:\users\APITest1"
```

## 3. Run the test batch

After entering the Test Batch Runner command and the location of the folder containing your tests, press ENTER. Test Batch Runner runs the test batch. For API tests, the test log is displayed in the command window.

## 4. View the test batch run results (API testing only)

In the test's **Report** folder, navigate to the run\_results.html file and double-click to open it in your browser.

# Using UFT for Continuous Integration

## Relevant for: GUI testing and API testing

As more software companies utilize continuous integration practices, you may also need to integrate functional tests into your testing process. This integration helps developers ensure that new builds did not introduce regressions.

The **HP Application Automation Tools** plugin for the Jenkins continuous integration server and Bamboo continuous integration server provides a mechanism for executing UFT tests as part of a build script. This open source plugin allows you to trigger an HP test as a build step and present the results in the Jenkins or the Bamboo user interface.

For more details, see

**Jenkins:** <https://wiki.jenkins-ci.org/display/JENKINS/HP+Application+Automation+Tools>

**Bamboo:**

<https://marketplace.atlassian.com/plugins/com.atlassian.plugins.bamboo.hpe.application.automation.bamboo>

# Optional Steps

## **Relevant for: GUI tests and scripted GUI components**

An **optional** step is a step that is not necessarily required to successfully complete a run session. For example, suppose that when creating a test or component, you add login steps because the application you are testing prompts you to enter a user name and password in a login window. Suppose, too, that this particular application remembers user login details, so that you do not need to log in every time you open the application. During a run session, the application does not prompt you to enter your user name and password because it retained the information that was previously entered. In this case, the steps that you added for entering the login information are not required and should, therefore, be marked optional.

During a run session, if the object of an optional step does not exist in the application, UFT bypasses this step and continues to run the test or component. When the run session ends, a message is displayed for the step indicating that the step was not performed, but the step does not cause the run to fail.

However, if, during a run session, UFT cannot find the object from the optional step in the object repository (for example, if the object name was modified in the test or component but not in the object repository, or if the object was removed from the object repository), an error message is displayed listing the required object, and the run fails.

During a recording session, UFT automatically marks steps that open certain dialog boxes as optional. (For a list of these dialog boxes, see "[Default Optional Steps](#)" below.)

You can also manually designate steps as optional. For example, you can add conditional statements or use recovery scenarios to automatically click a button, press ENTER, or enter login information in a step. For details, see "[Conditional Statements](#)" on page 718 and "[Recovery Scenarios for GUI Testing](#)" on page 144.

## Default Optional Steps

## **Relevant for: GUI tests and scripted GUI components**

By default, UFT considers steps that open the following dialog boxes or message boxes as optional steps:

Dialog Box / Message Box Title Bar
AutoComplete
File Download
Internet Explorer
Enter Network Password
Error

Dialog Box / Message Box Title Bar
Security Alert
Security Information
Security Warning
Username and Password Required

For an overview and task details, see "[Optional Steps](#)" on the previous page and "[How to Set Optional Steps](#)" on page 849.

## Log Tracking

### Relevant for: GUI tests and components

If the Windows-based application you are testing uses a supported Java or .NET log framework that includes a UDP appender, you can enable UFT to receive log messages from that framework and send them to the run results.

Analyzing the log messages that were generated as a result of a particular step can help you pinpoint the causes of unexpected behavior in your application, such as application crashes during automated runs.

Do this by configuring your application's log configuration file, either in the **Log Tracking** pane of the Settings dialog box, or manually.

For a list of supported log frameworks, see the *HP Unified Functional Testing Product Availability Matrix*.

### Manually configure log tracking settings

#### 1. Verify the following prerequisites:

- Your Windows-based application must use a Java or .NET log framework that includes a UDP appender.
- Logging must be enabled on your application. Verify that you know how to enable and disable logging.
- The log configuration file must be writable.

#### 2. Open the relevant log configuration file and specify your preferences

- a. Add an **appender-ref ref** attribute with the value of `QtpUdpAppender` to the **root** element.
- b. Specify the minimum log message level that you want to include in the run results.

**Note:** Log tracking messages are available only when viewing the run results in the Run Results Viewer.

**Example:**

```
...
<root>
 <level value="DEBUG" />
...
 <appender-ref ref="QtpUdpAppender" />
</root>
```

- c. Add an **appender** element and its attributes, as shown in the following example.

```
<appender name="QtpUdpAppender"
 type="log4net.Appender.UdpAppender">
 <remoteAddress value="1.1.1.1" />
 <remotePort value="18081" />
 <encoding value="utf-16" />
 <layout type="log4net.Layout.XmlLayoutSchemaLog4j">
 <prefix value="" />
 </layout>
</appender>
```

**Note:** To enable UFT to receive log messages, the Add log messages to run results area of the Log Tracking pane of the Settings dialog box must also be configured, as described below.

3. Configure the settings in the Log Tracking pane so that UFT will use the same settings that you defined in the previous step

This step enables UFT to receive log messages during a run session.

In the Log Tracking pane of the Settings dialog box:

- Select the Add log messages to run results check box.

**Note:** Log tracking messages are available only when viewing the run results in the Run Results Viewer.

- Specify the settings in the upper half of the pane:
  - Log messages source
  - Port
  - Minimum level to add node to results tree
- Clear the Auto-configure log mechanism check box. This prevents UFT from modifying the configuration file.

4. Results

During a run session, UFT receives any log messages that match or exceed the minimum log message level that you specified in the Log Tracking pane and displays them in the run results.

In the run results tree, a node is also inserted for the first message that matches or exceeds the Minimum level to add node to results tree. This node is inserted directly after the step that triggered (or preceded) the relevant log message (according to its timestamp).

## How to Set Optional Steps

### Relevant for: GUI tests and scripted GUI components

This task describes how to set an optional step.

Do one of the following:

- In the Keyword View, right-click the step and select **Optional Step**.
- In the Editor, add `OptionalStep` to the beginning of the VBScript statement. For example:

```
OptionalStep.Browser("Browser").Dialog("AutoComplete").WinButton("Yes").Click
```

For details on working in the Editor, see "[Programming in GUI Testing Documents in the Editor](#)" on page [647](#).

## Troubleshooting and Limitations - Run Sessions

### Relevant for: GUI and API tests

This section describes troubleshooting and limitations related to running tests.

#### UFT run sessions

- When running UFT on a remote machine using a Remote Desktop Connection session (RDC) or using Citrix, if the computer on which the application is being tested is logged off or locked, the following problems may occur:
  - The test or component run session may fail.
  - Steps that contain keyboard or focus operations may fail.
  - The still image capture and/or the Screen Recorder (in the run results) may display a black screen.
  - Steps for which the device level replay is configured to use the mouse (instead of browser events) to run mouse operations may fail. (You set the device level replay using a **Setting.WebPackage ("ReplayType")** statement or by setting the **Replay type** option in the Advanced Web Options dialog box.)

**Workaround:** If you are using Citrix or a Remote Desktop Connection session to run a test or component, make sure that the computer on which the application is being tested is not logged off or locked.

- When running UFT tests or components on a local machine, if the computer on which the application is being tested is locked, your test run may fail.

**Workaround:** Install UFT on a virtual machine (without a screensaver or lock password), and start or schedule your run session on the virtual machine. Then you can lock your local computer without locking the virtual machine.

- It is not recommended to use Test Batch Runner with the UAC (User Account Control) feature set to ON.

## Running GUI tests with a disconnected RDP connection

If you are running GUI tests from the ALM Test Lab, you must select the **Allow connections from computers running any version of Remote Desktop** option in the Windows Remote Settings (**Control Panel > System > Remote Settings**)

## Learning objects, running steps, and recording steps (GUI testing only)

- UFT cannot record or run steps if it has limited access to the processes of the application you are testing.

### Workarounds:

- Make sure that the application you are testing is started by the same Windows user as UFT.
- Make sure that neither you nor the tested application actively prevent UFT from accessing the application's processes.
- When the title of a window changes during recording, UFT may fail to recognize objects within that window while running the test or component.

**Workaround:** Remove the text property from the window's test object description in the Object Repository window.

- UFT cannot record steps in an action opened as read-only.

**Workaround:** Make sure that you have read-write access to the action you are recording in.

- If steps in your test or component seem to take a long time, try the following:
  - a. Open the run results and expand all nodes in the results.
  - b. If you see many Smart Identification icons , try to improve your object descriptions so that UFT does not have to use smart identification as much.
  - c. To improve your object descriptions you can:
    - In the run results, in the step that used smart identification, see which properties UFT used to uniquely identify the object, and add those properties to the object description in the object repository.

- Use the **Update test object descriptions** option of **Update Run Mode** to change the set of properties used for object identification for all objects in a specific class.

- **When running steps that contain Insight objects:**

- The computer session must be active, and the application visible (and not minimized), for the steps to run successfully. This is because Insight uses data from the screen to compare to the images stored with your test.
- If you are testing an application running on a remote computer, and you use a minimized Remote Desktop Connection window, these steps will fail.

**Workaround:** Use a different program (for example, Virtual Network Computing) instead of Remote Desktop Connection to run steps on a remote computer.

- For Insight object identification to succeed, the display size defined in the operating system and the browser zoom levels (if working on a browser) must be the same when the test runs as they were when the objects were learned or recorded.
- When using excluded areas in Insight, the included area must contain enough significant content to enable object recognition. If the remaining content is not detailed enough, UFT may locate too few or too many matching controls on the screen.

**Workaround:** Do one or both of the following:

- Use a larger area of the screen for the test object image (make sure to adjust the ClickPoint such that UFT clicks the correct location on the control).
- Define Visual Relation Identifiers to help identify the control.
- When searching for an Insight object within a parent Browser test object, UFT searches within the selected browser tab, not in the whole browser window.
- **Dual monitor support.** Insight is supported only on the primary monitor. Therefore, if you are working with dual monitors, make sure that your application is visible on the primary monitor when you use UFT for Insight test objects.
- If you selected the **Save movie to results** option in the Screen Recorder Pane of the Options dialog box, the Screen Recorder records a movie of the operations performed on your primary monitor.

Therefore, if you are working with multiple monitors, make sure that your application is fully visible on your primary monitor when recording or running a test or component.

## Running API Tests

- If you are running an API test that is stored in ALM and calls a GUI test, you must either ensure that UFT is open with a solution open, or you must close UFT and allow ALM to open it. The test will not run if UFT is open with no solution.

- When you manually add a reference to an external .dll, UFT prompts you to save it locally. To change your preference about a specific referenced file, remove the reference and add it again manually.
- Running tests on remote machines using shared folders, may require adjusting the .NET 2.0 security settings.

**Suggestion:** Open the **Control Panel** and locate the **Administrative Tools**, either by browsing or through a search. In the list of **Administrative Tools**, look for the following entry: Microsoft .NET Framework 2.0 Configuration. If it is not present, you must install the .NET Framework 2.0 SDK.

- The Validate Structure checkpoint fails if the expected value is a SOAP Fault and the Web Service call returns an **UnsupportedMediaType** status.
- When running an API test in Load Test mode, checkpoint verification in API test steps is not supported.

# Chapter 56: Using Run Results

This chapter includes:

- [UFT Run Results - Overview](#) ..... 854
- [Checkpoint and Output Value Results](#) ..... 861
- [How to Interpret Run Results](#) ..... 871

# UFT Run Results - Overview

## **Relevant for: GUI tests and components, API testing, and business process tests and flows**

After you run a test or component, UFT displays results that detail the success or failure of each step, and the reasons behind each failure (if necessary). These run results are opened automatically as a separate tab in the document pane immediately after the test or component run.

The run results are saved on a single HTML page, with links to other resources, such as the data table used with a test run, screen captures of the application being tested, and movies of the test run.

Because the run results are saved in HTML format, they can be exported and/or sent to other people without the need to have UFT installed.

To view the run results on your browser, you should use one of the following browsers:

- Internet Explorer 10 or higher (with Compatibility Mode disabled)
- The latest supported version of Chrome.

If you are viewing the run results from a test run from ALM, the run results report is supported on when running a test from ALM versions 11.52 patch 5 or higher or ALM 12.21.

The run results contain descriptions or links to a number of items:

- "[Test flow](#)" on the next page
- "[Error and warning information for test steps](#)" on page 856
- "[Run-time screen captures of your application \(GUI testing only\)](#)" on page 857
- "[Movies of the run session \(GUI testing only\)](#)" on page 858
- "[Data resources](#)" on page 859
- "[Call stack details for errors](#)" on page 858
- "[Custom report messages \(GUI testing only\)](#)" on page 859
- "[Captured data for test steps \(API testing only\)](#)" on page 859
- "[Full details of your business process test](#)" on page 860

## Test flow

In the run results, you can see a description of each step in your test (even if the test or a specific action ran over multiple iterations), with details about application objects and test objects used in the step, and the properties used to identify the test object:

The screenshot shows the UFT Report interface for a run titled "MercuryTours - Res22". The main header includes the title, execution time (2015-06-03 13:23:02), duration (00:00:17), tool name (HP Unified Functional Testing 12.5), and status indicators (0 errors, 0 warnings). Below the header, there are tabs for "Errors list" and "Test flow", with "Test flow" selected. A search bar is also present. The "Test Iteration Details" panel on the right displays information for "Iteration: Row 1", including execution time (2015-06-03 13:23:04) and duration (00:00:15). The "Test flow" panel lists steps under "Test Iteration: Row 1": "Login" (successful), "Accessibility Checkpoint: 'Welcome: Merc...' (successful), "password.SetSecure" (successful), "Sign-In.Click" (successful), and "FlightFinder" (successful). Each step is accompanied by icons representing the action type and its status.

You can also choose to filter what items are shown in the test flow.

## Error and warning information for test steps

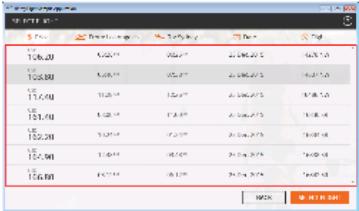
In addition to viewing only the test flow, you can view specific information about the errors (including warnings) that occur in your test flow. The run results give a description of the error, including the test object used and the expected result. For checkpoints, the checkpoint details (as selected in the Checkpoint Properties dialog box for a GUI test) are also displayed:

**Checkpoint Details**

**Standard Checkpoint**  
"CheckCost"

**Description**  
Verification type: String Content. Settings: Exact match - ON; Ignore space - ON; Match case - OFF. Rows Range : 1 - 7. Results: Checked 1 cells; Succeeded: 0; Failed: 1

**Execution Time**  
2015-11-18 13:12:32



---

**Test Object**  
 WpfWindow: "HP MyFlight Sample Application"

**Repository**  
C:\Users\ljerem\Desktop\UFT\_Tutorial\Tests\NET Tutorial\Tutorial\_Object Repositories>Select Flight.tsr

**Object Path**  
WpfWindow("HP MyFlight Sample Application").WpfTable("flightsDataGrid")

**Properties**  

devname	flightsDataGrid
---------	-----------------

---

**Captured Data**

0	1	2	3	4
USD106.20	[NONE]	[NONE]	[NONE]	[NONE]
[NONE]	[NONE]	[NONE]	[NONE]	[NONE]
[NONE]	[NONE]	[NONE]	[NONE]	[NONE]
[NONE]	[NONE]	[NONE]	[NONE]	[NONE]

## Run-time screen captures of your application (GUI testing only)

As part of the test step result details, you can see a screen capture of your application during the run session, either for every step or for steps with errors. For steps or errors on specific object in the application, UFT highlights the application object in the screen capture:

### Details

#### Step

creditCard.Select



#### Description

"Visa"

#### Execution Time

2015-04-01 11:29:22

#### Test Object

Browser.Page.WebList.Select

#### Repository

C:\Users\brojerem\Desktop\UFT\_Tutorial\_Tests\Tutorial\Tutorial\_ObjectRepositories\MercuryToursBookFlight.i

#### Object Path

Browser("Book a Flight: Mercury").Page("Book a Flight: Mercury").WebList("creditCard")

#### Operation

Select

#### Operation Data

"Visa"

#### Properties

name	creditCard
html tag	SELECT

**Note:** You select the level of screen capture detail in the Screen Capture pane of the Options dialog box (**Tools > Options > GUI Testing tab > Screen Capture node**).

## Call stack details for errors

When you encounter an error in your test, you may need to perform some investigation to find where the error occurs. To help with this, the run results display a call stack log for any errors encountered during the test run:

Cannot identify the object "home" (of class Image). Verify that this object's properties match an object currently displayed in your application.



### Description

Cannot identify the object "home" (of class Image). Verify that this object's properties match an object currently displayed in your application.

Line (12): "Browser("Flight Confirmation: Mercury").Page ("Flight Confirmation: Mercury").Image("home").Click".

### Execution Time

2015-04-01 11:30:28

---

## Test Object

Cannot identify the object "home" (of class Image). Verify that this object's properties match an object currently displayed in your application.

### Repository

C:\Users\brojerem\Desktop\UFT\_Tutorial\_Tests\Tutorial\Tutorial\_ObjectRepositories\MercuryToursFlightCo

### Object Path

Browser("Flight Confirmation: Mercury")

---

## Stacktrace

**BookFlight Line 12:** "Browser("Flight Confirmation: Mercury").Page("Flight Confirmation: Mercury").Image("home").Click"

---

## Movies of the run session (GUI testing only)

In the **Screen Capture** pane of the Options dialog box (**Tools > Options > GUI Testing tab > Screen Capture** node), you can instruct UFT to capture a movie of the run session. The run results provide a link and enable you open the file directly from the results:

Execution Time 2015-04-01 11:28:37 Duration 00:03:10 Tool Name HP Unified Functional Testing Tool Version 12.5

Show More > Video

**Note:** If you send the run results which have a movie to another user, you must send the entire folder containing the run results. If you do not, the movie link will not display the movie.

## Data resources

With the run results, you can open the data table used with the test run directly from the run results.

The run results provide a link and you simply open the file on your computer:

Execution Time 2015-11-18 13:11:10	Duration 00:00:36	Tool Name HP Unified Functional Testing 12.52	Result Name Res4	< Show Less
Operation System Windows 7 Service Pack 1 (x64)	CPU Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz	Core Number 2	Total Memory 4096MB	
Host NM13	User broje	Timezone +02:00:00	Locale English (United States)	Test Data

**Note:** If you send the run results which have a data table to another user, you must send the entire folder containing the run results. If you do not, the data table link will not display the data table.

## Custom report messages (GUI testing only)

When you run steps in a GUI test using the **Reporter** object, these messages are displayed in the run results, in the step in which they are run. For details on the **Reporter** object, see the **Utility Objects** section of the *HP UFT Object Model Reference for GUI Testing*.

## Captured data for test steps (API testing only)

For each API test step, the run results report the property values used to run each step. These values are reported in a grid which shows the input and output values.

This grid also includes any custom messages send to the run results with event handler code.

For each API test step in the test flow, the run results provide the captured values as part of the test step summary:

Step Details			
<b>Step</b>			ReserveOrder10
<b>Execution Time</b>			2015-11-18 13:17:52
<hr/>			
<b>Captured Data</b>			
Name	Step ID	Status	Message
ReserveOrder10	RESTActivityV210	Done	Successfully invoked an HTTP request
<hr/>			
More			
RequestBody	<FlightOrderDetails ... /FlightOrderDetails>		
HttpRawRequest	POST http://localhost:8000/FlightOrderDetails		
RequestHeader_Content-Type	text/xml		
RequestHeader_Host	localhost:8000		
RequestHeader_Content-Length	263		
RequestHeader_Expect	100-continue		
ResponseHeader_Content-Length	183		
ResponseHeader_Content-Type	text/xml; charset=utf-8		
ResponseHeader_Date	Wed, 18 Nov 2015 11:17:52 GMT		

You can also view run results in the Run Results Viewer. For details, see the the *HP Run Results Viewer User Guide*.

## Full details of your business process test

When you run a business process test in UFT, UFT can display all the details of the entire structure of your test, including:

- Business process flows
- Component groups
- Run conditions
- Component requests
- Steps with all types of components, including scripted GUI components, keyword GUI components, and API components

The example below shows the results from a business process test:

The screenshot shows the 'ComplicatedBPT - BPTReport' window. At the top, it displays 'Execution Time 2015-12-16 08:22:10', 'Duration 00:03:25', and 'Tool Name HP Unified Functional Testing 12.52'. Below this is a navigation bar with tabs: 'Errors list' (selected), 'Test flow', and a search bar. The main area is divided into two panes: 'Errors list' on the left and 'Flow Details' on the right.

**Errors list:**

- Flow: MixedFlow (#1)
  - Group: Group \_2Sc(2iter) (#1)
    - Group: Group \_2Sc(2iter) (#2)
  - Group: Group \_scPlusApi (#1)
    - sc2
    - API-bc2
  - Flow: MixedFlow (#2)
    - sc1
      - Condition Failure
    - API-bc1
    - ComponentRequest\_Random\_18655
  - bc1

**Flow Details:**

**Flow:** MixedFlow (#2)  
Execution Time: 2015-12-16 08:24:08  
Duration: 00:01:04

**Note:** Within each type of component, the individual steps are displayed in the same way as for a GUI test or API test, with each test step displayed separately and details in the Details pane.

## Checkpoint and Output Value Results

### Relevant for: GUI tests only

For checkpoints and output values, the run results provide additional detail relevant for the checkpoint or output value type:

- "Standard checkpoints" on the next page
- "Accessibility checkpoints" on the next page
- "Bitmap Checkpoints" on page 866
- "File Content Checkpoints" on page 867
- "Table and Database Checkpoints" on page 868
- "Text and Text Area Checkpoints" on page 869
- "XML Checkpoints" on page 870

- "Checkpoint and Output Value Results" on the previous page
- "Checkpoint and Output Value Results" on the previous page

## Standard checkpoints

For each standard checkpoint step in your test, the run results display detailed results of the selected checkpoint, including an icon indicating the checkpoint status (**Passed** or **Failed**), and the time and date of the checkpoint step.

The summary area also displays a list of details relevant to the checkpoint and the object being checked:

- The name of the checkpoint
- The properties being checked
- A screen capture of the object used in the checkpoint (if available depending on the selected options in the **Screen Capture** pane of the Options dialog box)

In the following example, the checkpoint to check the name in an input field fails because the expected name is not the entered name:

The screenshot shows the HP Unified Functional Testing interface. On the left, the 'Errors list' pane shows two failed checkpoints: 'Accessibility Checkpoint - Alt Property Check' (time +00:00:04) and 'Standard Checkpoint: "CheckName"' (time +00:00:31). On the right, the 'Test flow' pane displays the results of the 'Standard Checkpoint: "CheckName"'. The results show the checkpoint name, execution time (2015-04-05 22:27:36), and a screenshot of the application window. Below this, the 'Captured Data' table compares the expected and actual values for an input field:

Name	Expected	Actual
html tag	INPUT	INPUT
value	John	Bethami

## Accessibility checkpoints

When you include an accessibility checkpoint in your test, the run results display the results of each of the accessibility options you selected (in the **Web > Advanced** pane of the Options dialog box).

The test flow section of the run results displays a separate step for each accessibility option selected. For example, if you selected all the available options for your accessibility checkpoint, the run results, displays them like this:

The screenshot shows a list of accessibility checkpoints under a main category. The main category is 'Accessibility Checkpoint' with a red error icon and a blue expand/collapse arrow. Below it are six sub-items, each with a small icon and a checkmark or exclamation mark indicating status.

- ✓ ✖ ▼ Accessibility Checkpoint
  - Accessibility Checkpoint - ActiveX Check
  - ✖ Accessibility Checkpoint - Alt Property C...
  - ! Accessibility Checkpoint - Applet Check
  - Accessibility Checkpoint - Frame Titles C...
  - Accessibility Checkpoint - Multimedia Lin...
  - ! Accessibility Checkpoint - Server-side Im...
  - ! Accessibility Checkpoint - Tables Check

For those options that return warnings or errors, the summary area displays further details. The following example shows the information reported when the **ALT** property check of your application fails:

## Details

### Accessibility Checkpoint - Alt Property Check

"Alt property check"



#### Description

One or more objects in this page are missing a required <ALT> tag.

#### Execution Time

2015-04-05 22:30:46

### Captured Data

Object Tag	Object Name	Alt Value
IMG	Mercury Tours	Mercury Tours
IMG	html	[NONE]
IMG	boxad1	[NONE]
IMG	banner2	[NONE]
IMG	mast_flightfinder	[NONE]
IMG	spacer	[NONE]
IMG	spacer	[NONE]

Using the information in the run results, you can pinpoint parts of your Web site or Web application that do not conform to the W3C Web Content Accessibility Guidelines. The run results are based on these W3C requirements.

**Note:** Some of the W3C Web Content Accessibility Guidelines that are relevant to accessibility checkpoints are cited or summarized below. This information is not comprehensive. When checking if your Web site or Web application satisfies the W3C Web Content Accessibility Guidelines, you should see the complete document at <http://www.w3.org/TR/WAI-WEBCONTENT/>.

You can view the following specific information:

Option	Description

<b>ActiveX check</b>	Guideline 6 of the W3C Web Content Accessibility Guidelines requires you to ensure that pages are accessible even when newer technologies are not supported or are turned off. When you select the ActiveX check, UFT checks whether the selected page or frame contains any ActiveX objects. If it does not contain ActiveX objects, the checkpoint passes. If the page or frame does contain ActiveX objects, then the results display a warning and a list of the ActiveX objects so that you can check the accessibility of these pages on browsers without ActiveX support.
<b>Alt Property check</b>	Guideline 1.1 of the W3C Web Content Accessibility Guidelines requires you to provide a text equivalent for every non-text element. The <b>ALT</b> property check checks if objects that require an <b>ALT</b> property do actually have this attribute. If the selected frame or page does not contain any such objects, or if all such objects have the required attribute, the checkpoint passes. If one or more objects that require the property do not have it, the test fails and an error is displayed in the Test Flow and Errors list.
<b>Applet check</b>	The Applet check also helps you ensure that pages are accessible, even when newer technologies are not supported or are turned off (Guide 6 of the W3C Web Content Accessibility Guidelines). This option instructs UFT to find any Java applets or applications in the checked page or frame. The checkpoint passes if the page or frame does not contain any Java applets or application. Otherwise, the run results display a warning and a list of Java applets and applications.
<b>Frame Titles check</b>	Guideline 12.1 of the W3C Web Content Accessibility Guidelines requires you to title each frame to facilitate frame identification and navigation. When you select the Frame Titles check, UFT checks whether Frame and Page objects have the TITLE tag. If the selected page or frame and all frames within it have titles, the checkpoint passes. If the page, or one or more frames, do not have the tag, the test fails and the run result details display a list that shows which objects are lacking the tag.
<b>Multimedia Links check</b>	Guidelines 1.3 and 1.4 of the W3C Web Content Accessibility Guidelines require you to provide an auditory, synchronized description of the visual track of a multimedia presentation. Guideline 6 requires you to ensure that pages are accessible, even when newer technologies are not supported or are turned off. The Multimedia Links Check identifies links to multimedia objects so that you can confirm that alternate links are available when necessary. The checkpoint passes if the page or frame does not contain any multimedia links. Otherwise, the results displays a warning and a list of the multimedia links.
<b>Server-side Image check</b>	Guideline 1.2 of the W3C Web Content Accessibility Guidelines requires you to provide redundant text links for each active region of a server-side image map. Guideline 9.1 recommends that you provide client-side image maps instead of server-side image maps except where the regions cannot be defined with an available geometric shape. When you select the Server-side image check, UFT checks whether the selected page or frame contains any server-side image. If it does not, the checkpoint passes. If the page or frame does contain server-side images, then the results display a warning and a list of the server-side images so that you can confirm that each one answers the guideline requirements.
<b>Tables check</b>	Guideline 5 of the W3C Web Content Accessibility Guidelines requires you to ensure that tables have the necessary markup to be transformed by accessible browsers and other user agents. It emphasizes that you should use tables primarily to display truly tabular data and to avoid using tables to layout purposes unless the table still makes sense when linearized. The <b>TH</b> , <b>TD</b> , <b>THEAD</b> , <b>TFOOT</b> , <b>TBODY</b> , <b>COL</b> , and <b>COLGROUP</b> tags are recommended so that user agents can help users to navigate among table cells and access header and other table cell information through auditory means, speech output, or a Braille display.  The Tables check checks whether the selected page or frame contains any tables. If it does not, the checkpoint passes. If the page or frame does contain tables, the results display a warning and a visual representation of the tag structure of the table.

## Bitmap Checkpoints

The step details display the checkpoint step results, including its status (**Passed** or **Failed**), the date and time the checkpoint was run and the portion of the checkpoint timeout interval that was used (if any).

The manner in which you decide to run the bitmap checkpoint is reported differently in the run results:

<b>When Comparing Expected Bitmaps with Actual Bitmaps</b>	<p>The step details show the expected and actual bitmaps that were compared during the run session, and a <b>Difference</b> view. When you click any of the images, UFT opens the captured bitmap in a separate tab. When you click the Difference image, UFT displays an image that represents the difference between the expected and actual bitmaps. This image is a black-and-white bitmap that contains a black pixel for every pixel that is different in the two images.</p>	
<h3>Details</h3>		
<b>Bitmap CheckPoint</b> "HP MyFlight Sample Application"	<p><b>Execution Time</b>            2015-04-05 22:01:49</p>	
<hr/> <h4>Captured Bitmap</h4>		
<b>Expected</b> 	<b>Actual</b> 	<b>Difference</b> 
<b>Note:</b> Similar results would be displayed for a component:		
<b>When Locating Specified Bitmaps in Actual Bitmaps</b>	<p>The step details show the actual bitmap of the runtime object in the application and the source bitmap that UFT attempted to locate within the object. It may also show the coordinates of a possible candidate that was found, and the image similarity percentage used to find the candidate.</p>	
<b>Note:</b> By default, screen captures are available for bitmap checkpoints is available only if the bitmap checkpoint fails. You can change the conditions for when bitmaps are saved in the run results, using the <b>Save still image captures to results</b> option in the <b>Screen Capture</b> pane ( <b>Tools &gt; Options &gt; GUI Testing</b> tab > <b>Screen Capture</b> node) of the Options dialog box. For details, see the section describing the <b>Screen Capture</b> pane in the <i>HP Unified Functional Testing User Guide</i> .		

**Note:**

- When comparing bitmaps, if the checkpoint is defined to compare only specific areas of the bitmap, the run results display the actual and expected bitmaps with the selected area highlighted.
- When comparing bitmaps, if the dimensions of the actual and expected bitmaps are different, UFT fails the checkpoint without comparing the bitmaps. In this case the **View Difference** functionality is not available in the results.
- The **View Difference** functionality is not available when viewing results generated in a version of QuickTest earlier than 10.00.
- If the bitmap checkpoint is performed by a custom comparer:
  - UFT passes the bitmaps to the custom comparer for comparison even if their dimensions are different.
  - The Result Details pane also displays the name of the custom comparer (as it appears in the **Comparer** box in the Bitmap Checkpoint Properties dialog box), and any additional information provided by the custom comparer.
  - The difference bitmap is provided by the custom comparer.

## File Content Checkpoints

The step details display detailed results of the selected checkpoint, including its status (**Passed** or **Failed**), and the date and time the checkpoint was run. It also displays the number of lines that were checked, the number of changes found in the checked lines, and the total number of changed lines found in the file (including both the lines that were selected in the checkpoint and the lines that were not).

The details area also specifies whether the checkpoint includes the following options: **Match case**, **Ignore spaces**, **Verify page count**, and **Fail checkpoint for added or removed lines**.

For failed steps, the captured data displays a link enabling you to see the content comparison between the files.

In the following example, the details of the failed checkpoint indicate that the expected results and the current results do not match.

**File Content Checkpoint**

"run\_results.html"

**Description**

Checked lines: 3

Changes found (checked lines): 1

Changes found (all lines): 1

Match case: ON

Ignore spaces: OFF

Verify page count: OFF

Fail checkpoint for added or removed lines: OFF

**Execution Time**

2015-12-01 21:29:45

---

**Test Object**

 "run\_results.html"

**Repository**

Local

**Object Path**

FileContent("run\_results.html")

---

**Captured Data**

[Content Comparison](#)

---

**Context Information**

 run\_results.html

FileContent

---

## Table and Database Checkpoints

The run results displayed for a table or a database checkpoint are very similar. The run results display the checkpoint result, including an icon with the checkpoint status (**Passed** or **Failed**), the date and time the checkpoint was run.

The summary section also displays the object-relevant details for the checkpoint, including:

- Verification settings you specified for the checkpoint (in the Checkpoint Properties dialog box)
- The number of individual tables cells or database records that passed and failed the checkpoint.
- If the checkpoint fails, the summary section shows the table cells or database records checked by the checkpoint. If you click the Captured Data grid, a popup window shows another grid which displays the expected values and the actual values.

The following is an example of the results for a table checkpoint:

The screenshot shows the 'Errors list' section of the test flow. A single error is listed: 'Standard Checkpoint: "CheckCost"' with a status icon showing a red cross, indicating it failed. The timestamp '+00:00:19' is next to it. To the right, under 'Captured Data', there is a table with 8 rows and 5 columns labeled 0 through 4. The first row contains the value 'USD106.20' in the first column and '[NONE]' in the other four. All subsequent rows from 1 to 7 also contain '[NONE]' in all columns. The last row at index 8 also contains '[NONE]' in all columns.

0	1	2	3	4
USD106.20	[NONE]	[NONE]	[NONE]	[NONE]
[NONE]	[NONE]	[NONE]	[NONE]	[NONE]
[NONE]	[NONE]	[NONE]	[NONE]	[NONE]
[NONE]	[NONE]	[NONE]	[NONE]	[NONE]
[NONE]	[NONE]	[NONE]	[NONE]	[NONE]
[NONE]	[NONE]	[NONE]	[NONE]	[NONE]
[NONE]	[NONE]	[NONE]	[NONE]	[NONE]

## Text and Text Area Checkpoints

When you add a text or a text area checkpoint, the run results display relevant information for the checkpoint, including an icon indicating its status (Passed or Failed), and date and time of the checkpoint step. In addition, the summary also shows the relevant object details for the checkpoint, including:

- Expected text and the actual text in the application
- Verification settings you selected for the checkpoint
- A screen capture of the object for which you set the text or text area checkpoint

The following is an example of a text area checkpoint for a popup message that is displayed when you click the **ORDER** button in an application:

The screenshot shows the HP Unified Functional Testing interface. On the left, the 'Test flow' tab is selected, displaying a tree view of test steps. A node named 'CheckCost' is expanded, showing a 'Database Checkpoint' and several 'WpfWindow.WpfButton.Click' steps. One of these steps is highlighted with a blue border. On the right, a 'Details' panel is open for this step. It shows the following information:

- Text Checkpoint**: "CheckOrderCompletedText"
- Description**: Text Checkpoint: captured "completed" Match case: OFF Exact match: OFF Ignore spaces: ON
- Execution Time**: 2015-04-05 22:18:54

Below the description, there is a small screenshot of a Windows application window showing a message box with the text "Order completed".

## XML Checkpoints

The step details display the checkpoint step results. If the checkpoint or schema validation failed, the reasons for the failure are also shown.

If the checkpoint failed, you can view details of each check performed in the checkpoint by clicking **Content Comparison** link. A separate tab opens, displaying details of the checkpoint's failure.

The following is an example of the results for an XML checkpoint:

### Checkpoint Details

#### Xml Checkpoint

"file.xml"

#### Description

XML Checkpoint failed.

#### Execution Time

2015-12-01 17:21:16

---

#### Test Object

 "file.xml"

#### Repository

Local

#### Object Path

XMLFile("file.xml")

#### Properties

filename	C:\QTP.QA.Automation\TestOutput\HP.QTP.Tests.HTMLReport.Basic.HTMLReportGuiTestSanityFull\GUITest_Test HTML Report Pass\file.xml
----------	----------------------------------------------------------------------------------------------------------------------------------

---

#### Captured Data

[Content Comparison](#)

---

#### Context Information

 file.xml

XMLFile

## How to Interpret Run Results

### Relevant for: GUI tests and components, API testing, and business process tests and flows

This task describes navigate and interpret the information presented in the run results. This can help you quickly understand the results and errors in your test to isolate and solve problems in your application.

This task includes the following steps:

- "[Set run result reporting options](#)" on the next page
- "[View step details for each test step](#)" on the next page
- "[Analyze errors in your test or component](#)" on page 873
- "[Analyze checkpoint results \(GUI tests and components only\)](#)" on page 873

- "View the data source included with your test or component" on page 874
- "View the call stack to isolate errors in the test flow" on page 874
- "View the step properties capture for an API test step" on page 875
- "View custom messages sent to the run results" on page 875
- "Send the run results by email" on page 875

## Set run result reporting options

Before you begin a test run, you can set a number of different options to change the information included with the test results. This includes instructing UFT when to take screen captures and capture movies during a run result, as well as in what format to automatically open the run results.

You can set any of the following options in the Options dialog box (**Tools > Options**):

Option	Options Dialog pane	Description
<b>Instruct UFT to automatically open the run results</b>	Run Sessions pane ( <b>General</b> tab > <b>Run Sessions</b> node)	Select the <b>View results when run session ends</b> option.
<b>Select the format of the run results</b>	Run sessions pane ( <b>General</b> tab > <b>Run Sessions</b> node)	Select either the <b>HTML Report</b> or <b>Run Results Viewer Report</b> option.
<b>Take screen captures of steps</b>	Screen Capture pane ( <b>GUI Testing</b> tab > <b>Screen Capture</b> pane)	Select the <b>Save still image captures to results</b> option. You can specify to save screen captures for: <ul style="list-style-type: none"> <li>• <b>Always</b> (all steps)</li> <li>• <b>For errors</b></li> <li>• <b>For errors and warnings</b></li> </ul>
<b>Capture movies of the run session</b>	Screen Capture pane ( <b>GUI Testing</b> tab > <b>Screen capture</b> pane)	Select the <b>Save movie to results</b> option. You can specify to capture the movie: <ul style="list-style-type: none"> <li>• <b>Always</b> (all steps)</li> <li>• <b>For errors</b></li> <li>• <b>For errors and warnings</b></li> </ul>

## View step details for each test step

In the run results, you can see details for each test step. Do the following:

1. In the lower part of the run results, display the **Test Flow**. The run results displays a full list of all steps included in the test run.
2. From the step tree, select the step whose results you want to view. A summary of the details for that step are displayed.
3. View the details for the step, including:

- A description of the step, including the operation performed
- The test object used in the test step and the test object description
- A screen capture of the application with the test object highlighted (if you selected the option)
- Call stack information, if the test step reported an error

## Analyze errors in your test or component

In addition to reporting the results of every test step, the run results also display a special section detailing the errors and warnings that occurred during the test run. You can view these errors without the test flow to determine the root cause of the error:

1. In the lower part of the run results, display the **Errors List**.
2. Select an error from the list of errors. A summary of the error details are displayed.

**Tip:** If you are in the Test Flow view, you can also use the **Previous Error** and **Next Error** buttons  to quickly jump to the next error in the list.

3. Use the available details to isolate the cause of the error including:
  - A description of the error
  - The test object being used in the step
  - The properties of the test object
  - The call stack of the current step

## Analyze checkpoint results (GUI tests and components only)

For each checkpoint step, you can view information about the checkpoint, either if the checkpoint passed or failed. This can help you see how the application is performing.

<b>For a checkpoint that succeeded</b>	<ol style="list-style-type: none"><li>1. In the lower part of the run results, display the <b>Test Flow</b>.</li><li>2. Select the checkpoint step from the step list. A summary of the step details is displayed.</li><li>3. Use the information in the summary to view the checkpoint, including:<ul style="list-style-type: none"><li>• The properties checked in the checkpoint</li><li>• The test object used in the checkpoint</li><li>• The identification properties of the test object used in the step</li></ul></li></ol>
<b>For a checkpoint that failed</b>	<ol style="list-style-type: none"><li>1. In the lower part of the run results, display the <b>Errors</b>.</li><li>2. Select the failed checkpoint from the list. A summary of the error is displayed.</li></ol>

	<p>3. Use the information in the summary to find the source of the error, including:</p> <ul style="list-style-type: none"> <li>• The properties checked in the checkpoint</li> <li>• The expected and actual values of the checkpoint</li> <li>• The test object used in the checkpoint</li> <li>• The identification properties of the test object used in the step</li> </ul>
--	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

For full details on specific information displayed for each type of checkpoint, see "[Checkpoint and Output Value Results](#)" on page 861.

### View the data source included with your test or component

If your test uses a data source, this data source is attached to the test as an external file. This enables you to see exactly which data was used for this test run.

The location of the data differs depending on the test type and the type of data source:

Test type	Data source type	Location
GUI test or component	Data table	<p>A link named <b>Data</b> displayed in the <b>See More</b> section above the <b>Test Flow</b> and <b>Details</b> area of the run results. The data table is opened as an Excel file.</p> <p>You can also find the Excel file in the run results folder: <b>&lt;run results folder&gt;\Report\Default.xls</b></p>
API test	<ul style="list-style-type: none"> <li>• Excel</li> <li>• Local table</li> <li>• XML</li> <li>• Database</li> </ul>	<p>A link named <b>Data</b> displayed in the <b>See More</b> section above the <b>Test Flow</b> and <b>Details</b> area of the run results. This link opens a new browser page in which the specific data sources for the test are displayed. You can click on the name of a data source to view it as an external file.</p> <p><b>Note:</b> The actual run-time values used in each step (taken from the data source) are displayed in the Details section for each test step.</p>

**Note:** The run results do not sync the selected step in the Test Flow with the data source.

### View the call stack to isolate errors in the test flow

When you have an error in your test, you can use the Call Stack to determine exactly where this error occurs. This helps you isolate the specific line in the test that contains the error.

1. In the lower section of the run results, display the **Errors**.
2. From the list of errors and warnings, select an error. The summary of the error is displayed.
3. In the error details, find the section containing the **StackTrace**. This section displays the following:

- The section of the test containing the error (an action, function library, etc.)
- The specific function containing the error (if the error occurred in the context of a function call)
- The full script line in which the error occurred
- The line number of the error in the relevant document

## View the step properties capture for an API test step

When you run an API test, each test step requires specific property values to run the step. In the run results, you can view the properties and the values used in this specific test run:

1. In the lower section of the run results, display the **Test Flow**.
2. In the Test Flow, select the step you want to view. A summary of the test step is displayed.
3. In the summary, view the Captured Data section of the summary.

If the Captured Data contains links to external resources (for example a Web service Request/Response), you can click the link in the Captured Data and a floating window displays the detailed data.

## View custom messages sent to the run results

<b>For GUI tests</b>	Use the <b>Reporter</b> object to send custom messages to the run results. These messages appear in the Test Flow in the step in which you inserted the statement.
<b>For API tests</b>	When you send a custom message to the run results from an step's event handler, it is displayed as part of the step's captured data, which includes the properties and values used for a specific test run. <ol style="list-style-type: none"><li>1. In the lower section of the run results, display the <b>Test Flow</b>.</li><li>2. In the Test Flow, select the step you want to view. The details of the test step are displayed in the right pane.</li><li>3. In the details, view the Captured Data. In the captured data, you can see the custom field added in the data grid.</li></ol>

## Send the run results by email

In the tab displaying the run results, right-click the tab name and select **Send by Email**. A mail message opens in your default mail application with the run results attached.

**Note:** On computers running Windows 7, after you open the mail message window, you cannot use any user interface items in the UFT window until the mail window is closed.

# Chapter 57: Running Tests with Virtualized Services and Networks

**Relevant for: GUI tests and API tests**

This chapter includes:

- [Running Tests with Virtualized Services - Overview](#) ..... 877
- [Assigning Data and Performance Models to a Virtualized Service](#) ..... 877
- [Running a Test with a Network Emulation - Overview](#) ..... 878
- [How to Use a Virtualized Service for a UFT Test](#) ..... 879
- [How to Run a Test Using an Emulated Network](#) ..... 883

# Running Tests with Virtualized Services - Overview

## **Relevant for: GUI tests and API tests**

Application testing is usually performed on a real deployment of an application. However, sometimes the service upon which an application is based is unavailable or impractical for repeated use or testing. For example, it is impractical to test a flight booking application which requires the entry of a customer's credit card using the real credit card service run in combination with the application, as each time the test is run, the customer's credit card is charged. In such cases, you can replace your application's service with a **virtualized service** during testing.

Using HP Service Virtualization, you create a virtualized service by configuring the behavior of the virtual service to match the expected behavior of the real service. When you are finished creating the service's details in Service Virtualization, the service's details are saved as part of a **virtualization project**.

Then, in UFT, you add the virtualization project to a test. The project's settings are saved with the test for future testing sessions.

After adding a virtualization project, when designing your test, you use the virtual service address differently for GUI and API tests:

- For a GUI test, you insert the service address into your application's code in the function where the application calls the real service.
- For an API test, you insert the service's address in place of a URL or service address as a step property.

Then, before running the test containing the virtualized service, you deploy the service on the Service Virtualization Server. Then, when you run your test, the test runs using the virtual service as needed.

For details on creating a virtualization project and virtual services, see the *HP Service Virtualization User Guide*.

For task details, see "[How to Use a Virtualized Service for a UFT Test](#)" on page 879.

## Assigning Data and Performance Models to a Virtualized Service

## **Relevant for: GUI tests and API tests**

When you create and design a virtualized service, part of the configuration process is defining the expected behavior for the service: how quickly it should respond, what the requests and responses to this service should be, and so forth.

Because of this, you define performance models and data models for each virtualized service using Service Virtualization. These models are then saved with your virtualization project.

Later, when you add a virtualization project in a UFT test, the performance models and data models are included with each virtualized service. When you run a test using the virtualized service, you select one of the models to use for a test run.

## Performance Models

When you create a virtualization project and add services to the project, you can specify precisely how these services should perform when deployed and run. You can define how quickly the service responds, how often to send requests and responses to the service, the data load for the simulated server, and so forth.

After you create the necessary models in Service Virtualization and add the project to a test in UFT, you can choose which model to use for each test run.

There are a number of different types of performance models for a virtualization project:

- **User-defined:** This model reflects the customized performance settings created in Service Virtualization for a service. Each of the user-defined performance models is available for use in your UFT test.
- **Offline:** This model simulates the unavailability of a service. This model is available for all UFT tests.
- **None:** This model makes the service respond as quickly as possible. This model is available for all UFT tests.

For details on setting and defining performance models for your service, see the *HP Service Virtualization User Guide*

## Data Models

In addition to defining performance models for your virtualized service, you can also specify data models. Like performance models, the settings for each model are defined in Service Virtualization, and available for use in UFT tests.

Data models enable you to customize the requests and responses of the service to simulate real service performance. When you create a virtual service, you define the data model, either by providing the requests and responses for the service, or providing a data source that supplies the request and response values. In addition, you can set rules defining the data source use for each of the services and each of the different models.

All data models are user-defined.

For details on setting and defining data models for your service, see the *HP Service Virtualization User Guide*.

# Running a Test with a Network Emulation - Overview

### Relevant for: GUI tests and components

Network Virtualization enables you to network performance while your application is running in a test run session.

Network virtualization emulates real-world network conditions by imposing impairments and constraints on a lab-network during the software testing process. These include network latency, packet loss, and bandwidth limitations, among others.

For example, an application is run on a server located in New York.

- The server is accessed by users in London.

When users access the server, there is a delay due to network impairments and constraints that inevitably exist on an extended network, such as the one between New York and London.

- Software updates are developed for the system, and tested by the QA team based in New York.

Because QA is located so close to the sever, network impairments in the testing environment are much less than those that exist in the "live" system, and QA results may therefore not be accurate.

In your test or component, add steps to start and stop an emulation session to connect to the NV Test Manager and deploy an emulated network.

Run results include steps for the emulation session statements. Emulated network performance is displayed only in the NV Test Manager results.

For details, see "[How to Run a Test Using an Emulated Network](#)" on page 883.

Emulation session steps include the following methods:

- **NV.StartEmulation**
- **NV.StartEmulationExcludeIPs**
- **NV.StopEmulation**

For details on creating network emulation profiles, see your *Network Virtualization for Mobile* documentation.

## How to Use a Virtualized Service for a UFT Test

### **Relevant for: GUI tests and API tests**

This task describes how to deploy and use a virtualization project in your test.

This task includes the following steps:

- "[Prerequisite - Deploy the Service Virtualization server](#)" on the next page
- "[Add services from a virtualization project to your test in UFT](#)" on the next page
- "[Add virtualized services from a server to the test in UFT](#)" on the next page
- "[Undeploy a virtualized service](#)" on page 881
- "[Update service details \(optional\)](#)" on page 881
- "[Set the data and performance models for the virtualization project](#)" on page 882
- "[Pause a deployed service for a test run](#)" on page 882
- "[Put a service on standby](#)" on page 882
- "[Use the virtualization project in your GUI test](#)" on page 883
- "[Use the virtualization project in your API test](#)" on page 883

- "Run the test with a virtualized service" on page 883
- "View the virtualized service details in the run results" on page 883

## 1. Prerequisite - Deploy the Service Virtualization server

Before you use a virtualization project in a UFT session, you must start the Service Virtualization Server. For details, see the *HP Service Virtualization User Guide*.

## 2. Add services from a virtualization project to your test in UFT

- In UFT, click the **Virtualized Services Settings** button  in the toolbar.
- In the Virtualized Services Settings Dialog Box, click the **Add Services** button.
- In the Add New Services dialog box, select the **Project** radio button and do one of the following:
  - Click **Browse** and navigate to your virtualization project or virtualization project archive
  - Enter the location of the project in the edit field.

**Note:** You can open virtualization projects saved in the file system or in an ALM project.

- Click **Next**.
- In the next window, specify the **Server address** and click **Next**. If necessary, specify the credentials for accessing your Service Virtualization server.  
UFT checks that the server is accessible and deployed.
- In the next window, select the services to add to the test and click **Finish**.  
In the main UFT Service Virtualization dialog box, these services are now listed under the virtualization project name.
- In the main Service Virtualization setup dialog box, click **Save Setup** to add the virtualization project to your UFT test.

**Note:**

- You can add multiple virtualization projects to the same test.
- If you loaded a test or a service from the file system or an ALM project and you lost or changed the ALM connection information, any services and projects are reported as missing when you refresh the test. If you check the deployment of the services, they will still work as intended.

## 3. Add virtualized services from a server to the test in UFT

- In UFT, click the **Virtualized Services Settings** button  in the toolbar.
- In the Virtualized Services Settings Dialog Box, click the **Add Services** button.

- c. In the Add New Services dialog box, select the **Running Server** radio button and enter the name of your Service Virtualization server.

**Note:** You can open virtualization projects saved in the file system or in an ALM project.

- d. In the next window, provide the user name and password for the server.
- e. Click **Next**.
- f. In the next window, select the services to add to the test and click **Finish**.  
In the main UFT Service Virtualization dialog box, these services are now listed under the virtualization project name.
- g. In the main Service Virtualization Setup dialog box, click **Save Setup** to add the virtualization project to your UFT test.

**Note:** You can add multiple virtualization projects to the same test.

#### 4. Undeploy a virtualized service

By default, when you add a project or service to your test, it is automatically deployed. To stop the deployment, do the following:

- a. In the Virtualized Services Settings Dialog Box, click the **Show Runtime** button. The Service Virtualization Runtime dialog box opens
- b. In the Runtime dialog box, select the project you want to deploy.
- c. Click the **Undeploy** button. UFT pauses and checks, and removes the project on the server.  
If a project did not deploy correctly, hover over the error indication to see a description of the problem.

#### 5. Update service details (optional)

By default, the service information - including the location of the virtualization project or Service Virtualization server and the credentials required to access the server - is entered when you first add the service. If you need to update these details, do the following

- a. In the Virtualized Services Settings Dialog Box, click the parent link (in blue) for the services that you need to update. The service settings dialog box opens.
- b. In the Service Settings dialog box, update any of the following:

Service Detail	Where?	How to update
<b>Server address and credentials</b>	<b>SV Server</b> tab	In the SV Server tab, enter: <ul style="list-style-type: none"><li>o The revised Service Virtualization address</li><li>o The revised user name or password for the Service Virtualization server</li></ul>
<b>Virtualization project details</b>	<b>SV Project Location</b> tab	Enter the path to the project (on the file system or ALM) and the password (if necessary) for the project.

- 
- 
- 
- 
- 
- 6. Set the data and performance models for the virtualization project**

For each of your virtualization projects, you can configure how the service uses data when running the virtualized service. Before running a test using the virtualized service, you need to instruct UFT which of the associated data models to use:

- a. In the Virtualized Services Settings Dialog Box, select the virtualization project to use for the current test run.
- b. In the **Data Model** and **Performance Model** column for each service, from the drop-down list, select the name of the data model and performance model from the drop-down list to use for the current test run.

UFT uses the settings specified in the virtualization project for the service's performance and data usage.

For details on data and performance models in your virtualized service, see "[Assigning Data and Performance Models to a Virtualized Service](#)" on page 877.

## 7. Pause a deployed service for a test run

By default, all services are deployed when you add them to your test. If you do not want a particular service to run in a specific test run (but you do not want to undeploy the service), you can put it on standby:

- a. In the main Service Virtualization Setup dialog, click the **Show Runtime** button. The Service Virtualization Runtime dialog box opens.
- b. In the Service Virtualization Runtime dialog, select the services you want to pause.
- c. Above the service list, click the **Stop** button. UFT pauses for a moment while it stops the service on the Service Virtualization server.

**Note:** To restart the service for a test run, select the service again and click **Simulate**.

- d. Click **Close** to return to the main Service Virtualization setup dialog box.  
When UFT runs the test, it will use the run-time settings for your virtualized services.

## 8. Put a service on standby

You can put an entire virtualized service on standby, so that it is not available to be used in a test run:

- a. In the Service Virtualization setup window, select the service you want to put on standby.
- b. Above the service list, click **Stand-By**.

**Note:** To restore the service's availability, select the services again and click **Simulate**.

UFT pauses and sets the service to standby mode. The icon next to the service name also changes to a pause symbol.

## 9. Use the virtualization project in your GUI test

Make sure that your application is configured to use the virtual service address, as specified in the virtualization project. For details on defining virtual service addresses in your virtualization project, see the *HP Service Virtualization User Guide*.

## 10. Use the virtualization project in your API test

When creating your API test steps, you can use the virtualized services in place of calls or requests to real services, including:

- The URL for your Web Service steps
- The URL for your REST Service steps
- The URL for a HTTP Request or SOAP request step

## 11. Run the test with a virtualized service

After making all necessary changes for your associated virtualization project, run the test by selecting **Run > Run** or clicking the **Run** button .

**Note:** If you deployed a service with the Service Designer in Service Virtualization, you must stop the service simulation in the Service Virtualization Designer window, before running the test that uses the virtualized service. Failure to do so will result in the service being locked for all other users.

When the test runs the step using the virtualized service, it accesses the necessary service as defined in your virtualization project and runs the service.

## 12. View the virtualized service details in the run results

In the run results, navigate to the virtualization step. The service address is displayed in the step summary, as well as details about the service performance (both for the data and performance models).

# How to Run a Test Using an Emulated Network

## Relevant for: GUI tests and components

This task describes how to trigger a network emulation session from UFT and run tests on the virtualized network. This enables you to view how your network performs while your application is running.

This task includes the following steps:

1. "Prerequisites" below
2. "Enter your credentials for accessing the NV Test Manager" below
3. "Start a network emulation session" below
4. "Optional - exclude specific IP addresses from a network emulation" on the next page
5. "Run the test using the network emulation" on page 886

### 1. Prerequisites

Before running a test with a virtualized network, you must:

- Have access to the Network Virtualization Test Manager location.
- Create the necessary profiles in the NV Test Manager.
- For details on creating network profiles, see the section on managing network profiles in the *Network Virtualization for Mobile User Guide*.

### 2. Enter your credentials for accessing the NV Test Manager

In the Network Virtualization pane of the Options dialog (**Tools > Options > General tab > Network Virtualization node**), enter the following:

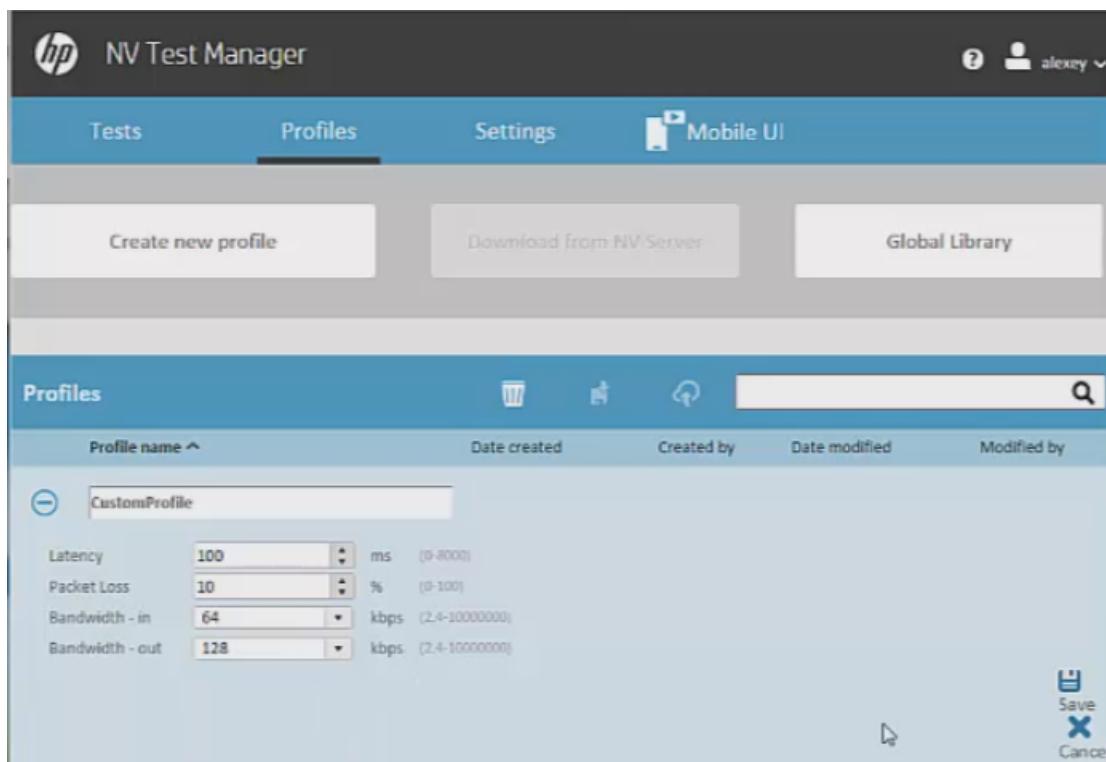
- The URL of the NV Test Manager, in the format `http://<NV Test Manager Address>:<NV Test Manager Port>`
- User name and password

### 3. Start a network emulation session

In your test or component, add a step to trigger an emulation session start:

```
NV.StartEmulation("profile name")
```

The profile name used is taken from the **Profiles** page in the NV Test Manager.



#### 4. Stop a network emulation

When you start a network emulation with the **NV.StartEmulation** or **NV.StartEmulationExcludeIPs** methods, the method returns a token with emulation ID.

This token is required to stop the emulation session.

Use the **NV.StopEmulation** method:

```
token = NV.StartEmulation("profile name")
NV.StopEmulation(token)
```

**Note:** You can name the token variable in the example above to any name.

#### 5. Optional - exclude specific IP addresses from a network emulation

In your network emulation profile, you can define network conditions for multiple networks. When you run a particular network emulation, you may want to exclude certain networks from the emulation.

You can exclude specific network locations (by IP address) in one of the following ways:

<b>In the Network</b> <b>Virtualization pane of</b>	In the Excluded IPs section add the IPs to exclude. These IP addresses are excluded from all network emulation sessions launched from UFT.
--------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------

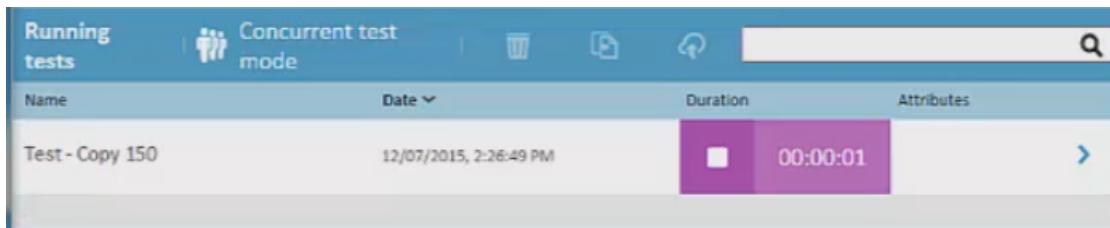
<b>the Options dialog box</b>	
<b>Using the NV.StartEmulationExcludeIPs method</b>	<p>Enter a step using the <b>.Start EmulationExcludeIPs</b> method:</p> <pre>NV.StartEmulationExcludeIPs("profile name", array of excluded IPs)</pre> <p>These IP address are excluded only from the current emulation session.</p>

## 6. Run the test using the network emulation

After you have configured the connection information for the NV Test Manager, and added the necessary statements to your test, run the test.

An icon  is displayed in the UFT status bar to indicate that the network virtualization is started.

In addition, you can see the network emulation running in the NV Test Manager.



In the run results, each emulation Start and Stop step is displayed separately. For full details on the network emulation performance, see your NV Test Manager test results.

# Chapter 58: Debugging Tests and Components

**Relevant for: GUI actions, scripted GUI components, function libraries, user code files, and business process tests**

This chapter includes:

• <a href="#">Debugging Overview</a> .....	888
• <a href="#">Considerations for Debugging Tests and Components</a> .....	889
• <a href="#">Running to a Step and Debugging from a Step</a> .....	890
• <a href="#">Modifying and Watching the Values of Variables and Properties of Objects During a Run Session</a>	892
• <a href="#">How to Debug Your Test, Component, Function Library, or User Code File</a> .....	893
• <a href="#">How to Debug a GUI Action or a Function - Exercise</a> .....	897
• <a href="#">How to Step Into, Out of, or Over a Specific Step in a GUI Test - Exercise</a> .....	900
• <a href="#">How to Debug an API User Code File - Exercise</a> .....	903
• <a href="#">Breakpoints</a> .....	906
• <a href="#">Enabling and Disabling Breakpoints</a> .....	907
• <a href="#">How to Use Breakpoints</a> .....	908
• <a href="#">Run Errors</a> .....	910
• <a href="#">Troubleshooting and Limitations - Debugging</a> .....	911

# Debugging Overview

**Relevant for: GUI actions, scripted GUI components, function libraries, user code files, and business process tests**

After you create testing documents such as a test, component, function library, event handler, or user code file, you should check that it runs smoothly, without errors in syntax or logic. If there are problems, you can stop and debug your tests.

**Note:** In order to debug a function library, you must first associate it with a test or with a component via its application area, and then debug it from that test or component.

UFT provides different options that you can use to debug your tests in order to detect and isolate defects in a document. For example:

- You can control the run session and begin debugging by using the **Pause** command, breakpoints, and various step commands that enable you to step into, over, and out of a specific step.
- When a run session is suspended, you can use the Debug panes or Quick Watch to check and modify the values of code objects and variables and to manually run script or code commands.
- If UFT displays a run error message during a run session, you can click the **Debug** button on the error message to suspend the run and debug the document.
- You can run a single step or step-by-step in a test, component, function library, or user code file by using the **Step Into**, **Step Out**, and **Step Over** commands:

<b>Step Into</b>	Runs only the current step in the active document.  <b>Note: (For GUI testing):</b> When debugging a GUI test, if the current step calls another action or a function, the called action or function is displayed in the document pane. The test or function library pauses at the first line of the called action or function.
<b>Step Out</b>	Continues the run to the end of the function, or user code file, returns to the calling test, component, or function library, and then pauses the run session at the next line (if one exists).
<b>Step Over</b>	If the current step calls a user-defined function, the called function is executed in its entirety, but the called function script is not displayed in the document pane. The run session then returns to the calling document and pauses at the next step (if one exists).  <b>Note: (For GUI testing):</b> If the current step calls another action, the called action is displayed in the document pane, and the run session pauses at the first line of the called action (like <b>Step Into</b> ).

- Run to or from a specific step or action. For details, see "[Running to a Step and Debugging from a Step](#)" on page 890.

# Considerations for Debugging Tests and Components

## Relevant for: GUI actions, scripted GUI components, and function libraries

Note the following considerations when debugging your tests or components:

### For GUI tests and components

- While a test, component, action, or function library is running in debug mode, it is read-only. You can modify the content after you stop the debug session (not when you pause it). After you implement your changes, you can continue debugging your document.

**Note:** If needed, you can enable the function library for editing after you stop the session. Right-click the function library tab in the document pane and select **Enable Editing**. (You cannot enable editing if the function library is locked by another user or checked in to an ALM project.)

- If you perform a file operation (for example, you open a different test or component, or create a new test or component), the debug session stops.
- **When debugging a test:** If a file is called using an **ExecuteFile Statement**, you cannot debug the file or any of the functions contained in the file. In addition, when debugging a test that contains an **ExecuteFile** statement, the execution marker may not be displayed correctly.

To debug a dynamically loaded function library, use a **LoadFunctionLibrary Statement** statement to load it instead of an **ExecuteFile** statement.

- When you open a test or component, UFT creates a local copy of the external resources that are saved to your ALM project. Therefore, any changes you apply to any external resource that is saved in your ALM project, such as a function library, will not be recognized in the test or component until the test or component is closed and reopened. (An external resource is any resource that can be saved separately from the test or component, such as a function library, a shared object repository, or a recovery scenario.)

In contrast with this, any changes you apply to external resources saved in the file system, such as function libraries, are implemented immediately, as these files are accessed directly and are not saved as local copies when you open your test.

### For API tests and components

- In order to use the API debugging features, you must enable the debugger in the API Testing **General** pane of the Options dialog box (**Tools > Options > APITesting tab > General node**). Select the **Run test in debugging mode** check box.
- While tests are running in debug mode, they are read-only. You can modify user code files after you stop the debug session (not when you pause it).
- When you open a test saved on an ALM project, UFT creates a local copy of the external references that are saved to your ALM project. Therefore, any changes you apply to any external reference that is saved in your ALM project, such as a WSDL file, will not be recognized in the test or component until

the test or component is closed and reopened. (An external reference is any resource that can be saved separately from the test or component, such as a WSDL file, XML file, or data table.)

## Running to a Step and Debugging from a Step

### Relevant for: GUI actions and scripted GUI components

You can use the **Run to Action** (tests only), **Run to Step**, and **Debug from Step** commands to instruct UFT to run a test, action, or component (including any associated function library), until it reaches a particular step or action, or to begin debugging from a specific step or action. You can also use the **Debug from Action** (tests only), **Run from Action** (tests only), and **Run from Step** commands to start or continue a run from a particular step or action.

The **Run to Step**, **Debug from Step**, and **Run from Step** commands are available in the **Run** menu, and when you right-click a step in your action or component.

The **Run to Action**, **Debug from Action**, and **Run from Action** commands are available when you right-click an action in the test canvas.

Each command differs slightly in how it runs:

### Run to Step

You can instruct UFT to run from the beginning of the test, action, or component—or from the current location in the test, action or component—and to stop at a particular step. This is similar to adding a temporary breakpoint to a step. For example, if you want to begin debugging your test, action, or component from a particular step, you may want to run your test, action, or component to that step, as this opens your application to the relevant location.

### Debug from Step

You can instruct UFT to begin your debug session from a particular step instead of beginning the run at the start of the test, action, or component. Before you start debugging from a specific step, make sure that the application is open to the location where you want to start debugging. You can begin debugging from a specific step in your test, action, or component when editing a test, action, or component.

For task details, see "[Start or pause your debugging session at a specific step or action in your test or component \(GUI testing only\)](#)" on page 894.

### Run from Step

You can instruct UFT to run your test, action, or component from a particular step instead of from the beginning of the test, action, or component. Before you start running from a specific step, make sure that the application is open to the location where you want the run to begin.

#### For tests

- In the Editor, the **Run from Step** option runs your test from the selected step until the end of the action (or until it reaches a breakpoint). Using **Run from Step** in this mode ignores any iterations. However, if the action contains nested actions, UFT runs the nested actions for the defined number of iterations of the

	<p>nested action.</p> <ul style="list-style-type: none"><li>In the Keyword View, if a single action is displayed, the <b>Run from Step</b> option runs your action from the selected step until the end of the action (or until it reaches a breakpoint). If the test flow is displayed, the <b>Run from Step</b> option runs the test from the selected step until the end of the test (or until it reaches a breakpoint), as long as all of the actions are internal actions that are local to your test.</li></ul> <p>Iterations are handled as follows:</p> <ul style="list-style-type: none"><li>If you select an Action node, the <b>Run from Step</b> option runs only one iteration of the test, from the selected step until the end of the test. Within the part of the test that runs, UFT runs each action for the number of iterations defined for that action.</li><li>If you select a step inside an action, the <b>Run from Step</b> option runs only one iteration of the action, from the selected step until the end of the action. Within the part of the action that runs, UFT runs any nested actions for the number of iterations defined.</li></ul> <p><b>Note:</b> If your test contains a call to an external action, the run session stops when that action is reached. Similarly, if you use the <b>Run from Step</b> option from within an external action, the run stops at the end of that action (or when a breakpoint is reached).</p>
<b>For components</b>	The <b>Run from Step</b> option runs your component from the selected step until the end of the component (or until it reaches a breakpoint).

#### Example:

After you debug a part of your action or component, you may want to skip over a set of steps that you know work correctly, and then continue the debug session from a later step. You can do this by inserting a breakpoint in the step where you want to continue debugging, and then using the **Run from Step** option to run the action or component from the step at which you stopped debugging until it reaches the breakpoint.

Alternatively, you can use the **Run from Step** option to continue the run session from a particular step to the end of the test, action, or component instead of stopping at a breakpoint.

### Run to Action (tests only)

You can instruct UFT to run from the beginning of the test until the beginning of the selected action and then pause the run session. For example, if you want to begin debugging your test from a particular action, you may want to run your test until that action, as this opens your application to the relevant location.

### Debug from Action (tests only)

You can instruct UFT to begin a debug session, and pause it, at the beginning of the selected action.

### Run from Action (tests only)

You can instruct UFT to start a run session from the beginning of the selected action.

## Modifying and Watching the Values of Variables and Properties of Objects During a Run Session

### **Relevant for: GUI actions, scripted GUI components, function libraries, and user code files**

During a run session, you can use the Watch pane, Local Variables pane, or Quick Watch to view the current value of different code expressions, variables, and object properties:

- The **Local Variables pane** displays the current values and types of all variables in the main script of the current action, or in a selected function in your test, function library, or user code files.
- The **WatchPane** displays the current values and the types of code expressions and objects that you add to the pane.
- The **Quick Watch** enables you to view the current value of a selected object in a line in your test or component, evaluate the value of an expression, or add an item to the Watch Pane.
- You can hover over objects, variables, or expressions in the Editor and see the value of these expressions.

As you continue stepping through the subsequent steps in your test, function library, or user code file, UFT automatically updates the Watch pane and Local Variables pane with the current value for any variable or expression whose value changes. In addition, UFT reevaluates the information displayed in the Watch pane and Local Variables pane as you make changes in the context of your debug session (in the Console Pane).

You can also change the value of a variable or property manually in Watch pane and Local Variables pane. For example, for test objects that support the **Object** property, you can edit the value of a runtime object property displayed in the Watch pane, thereby changing the value of the property in the application you are testing before you resume the run session.

You can add any of the following types of expressions to the Watch pane or the Quick Watch:

- The name of a GUI test object
- The name of a variable
- The name of a property
- Any other type of code expression

**Caution:** UFT runs the expressions in the Watch pane to evaluate them. Therefore, do not add a method or any expression whose evaluation could affect the state of the test or any GUI test object, as this can lead to unexpected behavior of your test, component, function library, or user code file.

Expressions added to the Watch pane are saved with the document and updated accordingly as you make changes to your document.

For task details, see "[Check the values of variables and expressions](#)" on page 895.

# How to Debug Your Test, Component, Function Library, or User Code File

**Relevant for: GUI actions, scripted GUI components, function libraries, user code files, and business process tests**

This task describes different ways you can control and debug your run sessions so you can identify and handle problems in your documents.

To practice this task, see "[How to Debug a GUI Action or a Function - Exercise](#)" on page 897 (for GUI testing) or "[How to Debug an API User Code File - Exercise](#)" on page 903 (for API testing).

**Note: (for GUI testing):** You can use the Screen Recorder to capture a movie of your application as it is being tested.

This task contains the following sections:

- "Prerequisites" below
- "[Slow your debugging session \(GUI testing only\)](#)" on the next page
- "[Step into, out of, or over a specific step during a debug session \(GUI and API tests only\)](#)" on the next page
- "[Start or pause your debugging session at a specific step or action in your test or component \(GUI testing only\)](#)" on the next page
- "[Use breakpoints in your document](#)" on the next page
- "[Check the values of variables and expressions](#)" on page 895
- "[Modify the values of variables or expressions during a run session](#)" on page 896
- "[View the current call stacks](#)" on page 896
- "[View currently running threads \(API testing only\)](#)" on page 896
- "[View the loaded modules associated with the run session \(API testing only\)](#)" on page 897

## Prerequisites

- You must have the Microsoft Script Debugger installed to run tests or components in debug mode. If it is not installed, you can use the UFT Additional Installation Requirements Utility to install it. (Select **Start > All Programs > HP Software > HP Unified Functional Testing > Tools > Additional Installation Requirements** or <UFT installation folder>\bin\UFTInstallReqs.exe.)
- **For GUI testing:** To debug components in UFT, you must enable integration between UFT and your ALM project. In UFT, select **Tools > Options > GUI Testing tab > Run Sessions** node and select the **Allow other HP products to run tests and components** check box.
- **For API testing:** To debug API tests, you must enable the debugger. Select **Tools > Options > API Testing tab > General** node and select **Run test in debugging mode**.

## Slow your debugging session (GUI testing only)

During a run session, UFT normally runs steps quickly. While you are debugging a test, component, or function library, you may want UFT to run the steps more slowly so you can pause the run when needed or perform another task.

In the **Test Runs** pane of the Options dialog box (**Tools > Options > GUI Testing tab > Test Runs node**), specify the time (in milliseconds) UFT pauses between each step by modifying the **Delay each step execution by** option.

## Step into, out of, or over a specific step during a debug session (GUI and API tests only)

- Select **Run > Step Into**, click the **Step Into** button , or press **F11**.
- Select **Run > Step Out**, click the **Step Out** button , or press **SHIFT+F11**.
- Select **Run > Step Over**, click the **Step Over** button , or press **F10**.

## Start or pause your debugging session at a specific step or action in your test or component (GUI testing only)

- Select the step in your document at which you want UFT to stop and select **Run > Run to Step** or press **CTRL+F10**.
- Select the step at which you want UFT to start the run and select **Run > Debug from Step**.

**Note:** These commands can also be used to stop at a specific action. Right-click an action in the canvas and select **Run to Action**, **Debug from Step**, or **Run from Action**.

## Use breakpoints in your document

For details, see "[How to Use Breakpoints](#) " on page 908.

### Note:

- If you run a test using the Run automation method, the test does not stop at breakpoints even if they are saved in the test.
- If you run a test with breakpoints using the Run automation method, the breakpoints remain visible but are ignored during the test run.
- If you are running a test from the ALM Test Lab module in **hidden mode** (as specified in the UFT Remote Agent), UFT will not stop the test at the breakpoints.
- If you are running a test from the ALM Test Plan module not in hidden mode, the test stops at breakpoints if you select the **Run Test Sets in debug mode** option in the UFT Remote Agent

## Check the values of variables and expressions

<b>In the Watch Pane</b>	<p>The Watch pane displays the value of selected variables and expressions that have been added to the Watch Pane.</p> <p>To add an expression do one of the following:</p> <ul style="list-style-type: none"> <li>Click the <b>Add New Watch Expression</b> button  and enter the name of the expression in the Add New Watch dialog box.</li> <li><b>For GUI actions, scripted GUI components, and function libraries only:</b> Highlight the selected expression and select <b>Run &gt; Add to Watch</b> right-click the expression and select <b>Add to Watch</b> from the context menu.</li> </ul> <p><b>Note:</b></p> <ul style="list-style-type: none"> <li>To add an <b>identification property</b> to the Watch pane, you must use an expression that calls <code>GetROProperty</code>. This enables you to watch the run-time value of the object's identification property. For example, to watch the value currently displayed in the Calculator application, you can add the expression:</li> </ul> <pre>Window("Calculator").WinEdit("Edit").GetROProperty("text")</pre> <ul style="list-style-type: none"> <li>You can add an expression to the Watch pane from the Editor or from a function library, but not from the Keyword View.</li> </ul>
<b>In the Quick Watch</b>	<ol style="list-style-type: none"> <li>In the line in your test or component containing the object, variable, or expression, do one of the following: <ul style="list-style-type: none"> <li>Right-click and select <b>Quick Watch</b></li> <li>Select <b>Run &gt; Quick Watch</b></li> </ul> </li> <li>In the Quick Watch, do one of the following: <ul style="list-style-type: none"> <li>In the Expression field, enter the name of the object, variable, or expression and click <b>Evaluate</b>. UFT displays the value in the current context of the test or component.</li> <li>Enter the name of the object, variable, or expression and click <b>Add to Watch</b>. UFT adds it to the list in the Watch Pane</li> </ul> </li> </ol>
<b>In the Local Variables pane</b>	<p>UFT displays all variables in the test or component (up to the current step) and their value in the current context of the test or component run.</p>
<b>In the Editor</b>	<p>Hover over an object, variable, or expression when the test run is paused. UFT displays a floating tooltip with the value:</p>

<b>obj</b>	<3 sub-items>	Window
<b>[Methods]</b>	<38 sub-items>	
Activate()	Activate(BUTTON)	void
CaptureBitmap()	CaptureBitmap(FullFileName, OverrideExis	void
CaptureBitmapToBase64Strin	CaptureBitmapToBase64String(lbstrBitmap	HRESULT
CaptureSnapshotToBase64St	CaptureSnapshotToBase64String(imageFo	HRESULT
Check()	Check(Verify)	Boolean
CheckProperty()	CheckProperty(PropertyName, PropertyVa	Boolean
ChildObjects()	<Object>	Object
Click()	Click(X, Y, BUTTON)	void
ClickOnText()	ClickOnText(TextToFind, Left, Top, Right, B	void
Close()	Close()	void

**Note:** To expand the ability of UFT to display information on test objects, it is recommended to register PDM from Internet Explorer (for versions of Internet Explorer 8 or higher and Visual Studio 2008). Enter the following in the command line to register the .dll: regsvr32 "%ProgramFiles%\Internet Explorer\pdm.dll"

## Modify the values of variables or expressions during a run session

Do one of the following:

- In the Console pane, enter a command to change the value of an object, variable, or expression.
- In the Watch or Local Variables pane, in the Value column for the object, variable, or expression, manually change the value.

## Manually run code commands during a debug session

In the Console pane, enter the command to run.

## View the current call stacks

To view the currently running call stacks in your run session, select **View > Debug > Call Stack**. You can double-click on a stack name in the pane to navigate directly to the line of code that begins the call stack.

**Note:** If the location of the call stack is not in a currently open file, UFT opens the relevant file.

## View currently running threads (API testing only)

Select **View > Debug > Threads**. In the list of threads that is displayed, you can double-click on the thread name to navigate directly to the beginning of the thread.

**Note:** If the file containing the beginning of the selected thread is not open, then UFT opens the

relevant file.

## View the loaded modules associated with the run session (API testing only)

Select **View > Debug > Loaded Modules**. UFT displays the list of currently loaded modules (depending where in the test you have paused).

# How to Debug a GUI Action or a Function - Exercise

### Relevant for: GUI actions, scripted GUI components, and function libraries

In this exercise, you create and debug an action or a function, to practice using some of UFT's debugging capabilities for GUI tests.

Suppose you create an action or a function that defines variables that are used in other parts of your test or function library. You can add breakpoints to the action or function to see how the value of the variables changes as the test or function library runs. To see how the test or function library handles the new value, you can also change the value of one of the variables during a breakpoint.

**Note:** For a task related to this exercise, see "[How to Debug Your Test, Component, Function Library, or User Code File](#)" on page 893.

This exercise includes the following steps:

- "[Create a new action or function](#)" below
- "[Associate the function library with a test or application area \(function libraries only\)](#)" on the next page
- "[Add a call to the function in your test or component \(function libraries only\)](#)" on the next page
- "[Add breakpoints](#)" on page 899
- "[Begin running the test or component](#)" on page 899
- "[Check the value of the variables in the debug panes](#)" on page 899
- "[Check the value of the variables at the next breakpoint](#)" on page 899
- "[Modify the value of a variable using the Console pane](#)" on page 899
- "[Repeat a command from the command history](#)" on page 900

### 1. [Create a new action or function](#)

- a. Do one of the following:
  - **For tests:** Create or open a test. In addition, as an optional step, you can create or open a function library.
  - **For components:** Create or open a function library. For details on creating a function library, see "[How to Create and Manage Function Libraries](#)" on page 693.
- b. Create a new function called **SetVariables**. For details on working with functions, see "[User-](#)

[Defined Functions and Function Libraries" on page 683.](#)

Enter the VBScript code in the Editor of your action or the function library, as follows:

**Action**

```
Dim a
a="hello"
b="me"
MsgBox a
```

**Function Library**

```
Function SetVariables

Dim a
a="hello"
b="me"
MsgBox a

EndFunction
```

For more details on the Editor, see ["Programming in GUI Testing Documents in the Editor" on page 647.](#)

**Note:** If you are working only with an action in the Editor, skip to [Add Breakpoints](#). If you are working in a function library (in a test or component), proceed to the next step.

## 2. **Associate the function library with a test or application area (function libraries only)**

- a. If you are working with a component, make sure the application area associated with your component is open.
- b. Bring the function library into focus.
- c. Right-click the function library document tab and select **Associate Library '<Function Library Name>' with '<Test/ Application Area Name>'**. UFT associates the function library with your test or application area.

**Note:** For additional details on how to associate a function library, see ["How to Manage Function Library Associations" on page 697.](#)

## 3. **Add a call to the function in your test or component (function libraries only)**

Do one of the following:

- **For tests:** Add a call to the function by typing the following in the Editor:

```
SetVariables
```

- **For components:** Add a call to the function by inserting a new operation and choosing **SetVariables** from the **Operation** list.

#### 4. **Add breakpoints**

Add breakpoints at the lines containing the text `b="me"` and `MsgBox a`. For details on adding breakpoints, see "[Breakpoints](#)" on page 906.

#### 5. **Begin running the test or component**

Run the test or component. The test or component stops at the first breakpoint, before executing that step (line of script).

#### 6. **Check the value of the variables in the debug panes**

- Select **View > Debug > Watch** to open the Watch pane.
- In the Editor of your test action or in the function library, highlight the variable **a** and select **Run > Add to Watch**.

UFT adds the variable **a** to the Watch pane. The **Value** column indicates that the value of **a** is currently "**hello**", because the breakpoint stopped after the value of variable **a** was initiated. The **Type** column indicates that **a** is a **String** variable.

- In the Editor displaying your test action or in the function library, highlight the variable **b** and select **Run > Add to Watch**.
- UFT adds the variable **b** to the Debug Watch pane. The **Value** column indicates **<Variable is undefined: 'b'>** (and the **Type** column displays **Incorrect expression**), because the test or component stopped before variable **b** was declared.
- Select **View > Debug > Local Variables** to open the Local Variables pane.

**For tests:** Only variable **a** is displayed (with the value "**hello**"), because **a** is the only variable that was initiated up to this point.

**For components:** Both **SetVariables** (with the value **Empty**) and variable **a** (with the value "**hello**") are displayed.

For both tests and components, variable **b** is not displayed because the test or component stopped before variable **b** was declared.

#### 7. **Check the value of the variables at the next breakpoint**

Click the **Run** button to continue running the test or component.

The test or component stops at the next breakpoint. Note that the values of variables **a** and **b** were both updated in the Watch and Local Variables panes.

#### 8. **Modify the value of a variable using the Console pane**

- Select **View > Debug > Console**.
- At the command prompt at the bottom of the pane, enter:  
`if b="me" then a="b is me" else a="b is you" end if`

Then press **ENTER** on the keyboard.

- c. Click the **Local Variables** pane to verify that the value of variable **a** was updated according to the command you entered and now displays the value: "b is me"
- d. Click the **Run** button to continue running the test or component.

The message box that opens displays "b is me" (which is the modified value of **a**). This indicates that you successfully modified the values of both **a** and **b** using the Debug Console pane.

- e. Click **OK** to close the message box.

## 9. **Repeat a command from the command history**

- a. Remove the first breakpoint and run the test or component again.

When the test or component stops at the breakpoint (before displaying the message box), modify the value of variable **b** in the Console pane by running the command `b="not me"`.

- b. In the Console pane, highlight the command line that reads `if b="me" then a="b is me"` `else a="b is you"`. Then right-click and select **Copy**.
- c. In the command prompt, right-click and select **Paste**.
- d. Press **ENTER** to run the command, and then click the **Run** button to complete the test or component run.

A message box saying "b is you" opens. Click **OK** to close the message box.

# How to Step Into, Out of, or Over a Specific Step in a GUI Test - Exercise

## **Relevant for: GUI actions, scripted GUI components, and function libraries**

In this exercise, you create a sample function library and run it (from a test or component) using the **Step Into**, **Step Out**, and **Step Over** commands.

**Note:** For a task related to this exercise, see "[How to Debug Your Test, Component, Function Library, or User Code File](#)" on page 893.

This exercise includes the following steps:

- "[Create the sample function library and test \(tests only\)](#)" on the next page
- "[Create the sample function library and component \(components only\)](#)" on the next page
- "[Run the function library from your test or component, and use the Step Into, Step Out, and Step Over commands](#)" on page 902

## 1. Create the sample function library and test (tests only)

**Note:** This procedure is relevant for test actions and function libraries only. If you are working with a component, see the relevant "[Create the sample function library and component \(components only\)](#)" below for components below.

- a. Select **File > New > Function Library** to open a new function library.
- b. In the function library, enter the following lines exactly:

```
public Function myfunc()
msgbox "one"
msgbox "two"
msgbox "three"
```

The End Function is automatically added by UFT.

- c. Save the function library to your ALM project or to the file system, with the name SampleFL.qfl. (For details, see "[How to Create and Manage Function Libraries](#)" on page 693.)
- d. Select **File > New > Test** and select **GUI Test** to open a new test.
- e. Click the document tab for the SampleFL.qfl function library to bring it into focus.
- f. Right-click the function library document tab and select **Associate Library 'SampleFL.qfl' with 'Test'** to associate the function library with your test.
- g. Click the document tab for the action you created to bring it into focus. Click the Editor/Keyword View toggle button to display the Editor and enter the following lines exactly:

```
myfunc
myfunc
myfunc
endOfTest="true"
```

## 2. Create the sample function library and component (components only)

**Note:** This procedure is relevant for components and function libraries only. If you are working with a test, see the relevant "[Create the sample function library and test \(tests only\)](#)" above for tests, above.

- a. Select **File > New > Function Library** to open a new function library.
- b. In the function library, enter the following lines exactly:

```
public Function myfunc()
msgbox "one"
msgbox "two"
msgbox "three"
```

- The End Function is automatically added by UFT.
- c. Save the function library to your ALM project or to the file system, with the name SampleFL.qfl. (For details, see ["How to Create and Manage Function Libraries" on page 693](#).)
  - d. Select **File > New > Application Area** to open a new application area.
  - e. Save the application area in your ALM project with the name **SampleAA**. (For details, see ["How to Create and Manage Application Areas" on page 1016](#).)
  - f. Click the document tab for the SampleFL.qfl function library to bring it into focus.
  - g. Right-click on the function library document tab and select **Associate Library 'SampleFL.qfl' with 'SampleAA'** to associate the function library with the open application area.
  - h. Select **File > New > Business Component** and select **GUI Keyword Component**. In the New Dialog Box, associate the component with the **SampleAA** application area that you created and saved in your ALM project.
  - i. In the component, insert four identical steps. For each step:
    - In the **Item** cell, select **Operation**.
    - In the **Operation** cell, select **myfunc**.
3. **Run the function library from your test or component, and use the Step Into, Step Out, and Step Over commands**
- a. Select the first step of the action or component (the first call to the myfunc function) and add a breakpoint by pressing **F9 (Insert/Remove Breakpoint)**. The breakpoint symbol is displayed in the left margin . For details, see ["Breakpoints" on page 906](#).
  - b. Run the test or component. The test or component pauses at the breakpoint.
  - c. Press **F11 (Step Into)**. The execution arrow points to the first line (`msgbox "one"`) of the function in the function library.
  - d. Press **F11 (Step Into)** again. A message box displays the text one.
  - e. Click **OK** to close the message box. The execution arrow moves to the next line in the function.
  - f. Continue pressing **F11 (Step Into)** (and pressing **OK** on the message boxes that open) until the execution arrow leaves the function and is pointing to the second step in the action or component (the second call to the myfunc function).
  - g. Press **F11 (Step Into)** to enter the function again. The execution arrow points to the first `msgbox` line within the function.
  - h. Press **SHIFT+F11 (Step Out)**. Close each of the message boxes that opens. Notice that the execution arrow continues to point to the first line in the function until you close the last of the three message boxes. After you close the third message box, the execution arrow points to the next line in the action or component (the third call to the myfunc function).
  - i. Press **F10 (Step Over)**. The three message boxes open again—this time, in the Keyword View. The execution arrow remains on the same step in the action or component until you close the last of the three message boxes. After you close the third message box, the execution arrow points to the next step in the action or component.

# How to Debug an API User Code File - Exercise

## Relevant for: User code files

In this exercise, you create and debug an API user code file to practice using some of UFT's debugging capabilities for API tests.

**Note:** For a task related to this scenario, see "[How to Debug Your Test, Component, Function Library, or User Code File](#)" on page 893.

This scenario includes the following steps:

- "Create test steps" below
- "Set properties for the math steps" below
- "Create parameters for the Custom Code activity" below
- "Link the Custom Code activity to existing steps" on the next page
- "Create events for the Custom Code activity" on the next page
- "Run the test" on page 905
- "Check the value of the variables at the first breakpoint" on page 905
- "Add a variable to the Watch Pane" on page 905
- "Check the value of the variables at the next breakpoint" on page 905

### 1. [Create test steps](#)

- a. Create an API test.
- b. From the Toolbox Pane, drag the **Add** activity, **Multiply** activity, and the **Custom Code** activity to the canvas.

### 2. [Set properties for the math steps](#)

- a. In the **Input/Checkpoints** tab  , in the Input pane, enter the values for the **Add4** operation.
  - In the **Value** column for **A**, enter 10.
  - In the **Value** column for **B**, enter 6.
- b. In the **Input/Checkpoints** tab, enter the values for the **Multiply** operation.
  - In the **Value** column for **A**, enter 2.
  - In the **Value** column for **B**, enter 4.

### 3. [Create parameters for the Custom Code activity](#)

- a. In the Add Input/Output Property/Parameter dialog box, enter the details for the input parameter.

- In the **Name** field, enter AddResult.
- In the **Type** field, select String from the drop-down list (if it is not already selected).

A new input property called **AddResult** appears in the **Input** pane inside the **Input/Checkpoints** tab.

b. Create another input parameter called **MulResult**:

- In the **Name** field, enter MulResult.
- In the **Type** field, select String from the drop-down list (if it is not already selected).

A new input property called **MulResult** appears in the **Input** pane inside the **Input/Checkpoints** tab.

c. Click the **Add** button again and select **Add Output Property**.

d. Enter the details for the output parameter.

- In the **Name** field, enter Result.
- In the **Type** field select Decimal from the drop-down list.

e. In the **Checkpoints** pane, enter the value of 128.

**4. Link the Custom Code activity to existing steps**

a. In the Select Link Source Dialog Box, select the **AddActivity** step. In the right pane, select **Result** and click **OK**.

The link source Step.OutputProperties.AddActivity<number>.Result appears in the **AddResult** row.

b. In the dialog, select the **Multiply** step. In the right pane, select **Result** and click **OK**.

The link source Step.OutputProperties.MultiplyActivity<number>.Result appears in the **MulResult**.

**5. Create events for the Custom Code activity**

a. In the Properties pane, select the **CustomCode** activity from the drop-down list or by clicking the **CustomCode** activity in the canvas.

b. In the Properties pane, select the **Events** tab.

c. In the Events Tab, create a default handler for **ExecuteEvent** and **AfterExecuteStepEvent**. Two events, CodeActivity<number>\_OnExecuteEvent and CodeActivity<number>\_OnAfterExecuteEvent are added to the TestUserCode.cs file.

For details on API testing event handlers, see "[Writing Code for API Test Events - Overview](#)" on [page 755](#).

d. In the TestUserCode.cs file, enter the following text in the CodeActivity\_OnExecuteEvent:

```
decimal AddResult = AddActivity.Result;
decimal MulResult = MultiplyActivity.Result;
CodeActivity<number>.Output.Result = AddResult*MulResult;
```

- e. Enter a breakpoint on the last line of this method.
- f. In the TestUserCode.cs file, enter the following text in the CodeActivity<number>\_OnAfterExecuteStepEvent:

```
decimal result = CodeActivity<number>.Output.Result;
```

- g. Enter a breakpoint on the last line of this method.
- h. Save the test.

## 6. Run the test

Select **Run > Run** or press F5.

## 7. Check the value of the variables at the first breakpoint

- a. When the test run stops at the first breakpoint, select **View > Debug > Local Variables** to open the Local Variables pane.
- b. In the Local Variables pane, see the current values of the **Add** and **Multiply** activity. The current values should be 16 for the **AddActivity** and 8 for the **MultiplyActivity**.

You can expand the notes on various rows to see the variable values of the different items used in your test run.

## 8. Add a variable to the Watch Pane

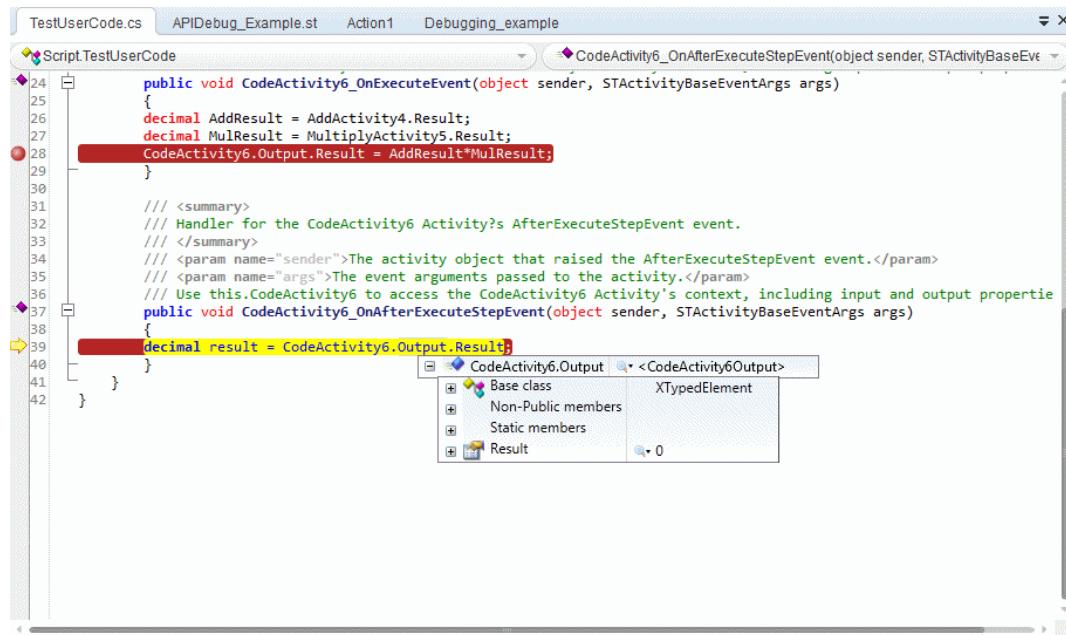
- a. In the **TestUserCode.cs** tab, highlight the text **AddResult**.
- b. Open the Watch pane by selecting **View > Debug > Watch**.
- c. In the Watch Pane, click the **Add New Watch Expression** button  and enter **Add Result**.  
A line with the expression **AddResult**, with a value of 16, and type **Decimal** appears.
- d. Click the **Add New Watch Expression** button again to add the variable **MulResult** to the Watch pane. The pane should display the expression **MulResult**, with a value of 8, and type **Decimal**.

## 9. Check the value of the variables at the next breakpoint

- a. Continue the run session by selecting **Run > Continue** or pressing F5.
- b. When the run session pauses at the next breakpoint, highlight the text **CodeActivity<number>.Output.Result** and add it to the Watch pane.  
The pane displays that the value of this variable is 128.  
Note that the **AddResult** and **MulResult** values, which you added in the previous step to the Watch pane, are undefined with a type **Incorrect Expression**. This is because these values are present and relevant to the current event.

- c. Click the Local Variables tab. Note that the line `Result` displays a value of 128, since the custom code entered earlier noted that `Result` is equal to `CodeActivity6.Output.Result`, which was equal to `AddResult*MulResult`.

In addition, if you hover over the variable names in the Editor in the paused run session, an expandable tooltip displaying the current value of the variable and its properties can be viewed.



- d. Select **Run > Continue** or click F5 to complete the run session and view the run results.

## Breakpoints

**Relevant for: GUI actions, scripted GUI components, function libraries, user code files, and business process tests**

You can use breakpoints to instruct UFT to pause a run session at a predetermined place in a document. UFT pauses the run when it reaches the breakpoint, before executing the step. You can then examine the effects of the run up to the breakpoint, make any necessary changes, and continue running the document from the breakpoint.

Breakpoints are saved with your document and are maintained even after you close and reopen UFT. UFT maintains the breakpoints inserted in a test or component per user on a given computer.

You can use breakpoints to:

- Suspend a run session and inspect the state of your application
- Mark a point from which to begin stepping through a document using Step Into, Step Out, or Step Over commands.

For task details, see "[How to Use Breakpoints](#)" on page 908.

**Note:**

- If you run a test using the Run automation method, the test does not stop at breakpoints even if they are saved in the test.
- If you run a test with breakpoints using the Run automation method, the breakpoints remain visible but are ignored during the test run.
- If you are running a test from the ALM Test Lab module in **hidden mode** (as specified in the UFT Remote Agent), UFT will not stop the test at the breakpoints.
- If you are running a test from the ALM Test Plan module not in hidden mode, the test stops at breakpoints if you select the **Run Test Sets in debug mode** option in the UFT Remote Agent

## Enabling and Disabling Breakpoints

**Relevant for: GUI actions, scripted GUI components, function libraries, user code files, and business process tests**

You can instruct UFT to ignore an existing breakpoint during a debug session by temporarily disabling the breakpoint. Then, when you run your document, UFT runs the step containing the breakpoint, instead of stopping at it. When you enable the breakpoint again, UFT pauses there during the next run. This is particularly useful if your document contains many steps or events, and you want to debug a specific part of it.

You can enable or disable breakpoints individually or all at once. For example, suppose you add breakpoints to various steps throughout your document, but for now, you want to debug only a specific part of it or a specific event. You could disable all breakpoints in your document and then enable breakpoints only for specific steps or in specific event handlers. After you finish debugging that section, you could disable the enabled breakpoints, and then enable the next set of breakpoints (in the section or event you want to debug). Because the breakpoints are disabled and not removed, you can find and enable any breakpoint, as needed.

**Enabled breakpoint.** An enabled breakpoint is indicated by a filled red circle icon (●) in the left margin adjacent to the selected step.

**Disabled breakpoint.** A disabled breakpoint is indicated by an empty circle icon (○) in the left margin adjacent to the selected step.

When UFT runs a test without stopping on breakpoints, such as when running in hidden mode, the Editor indicates that the breakpoint is not in use.

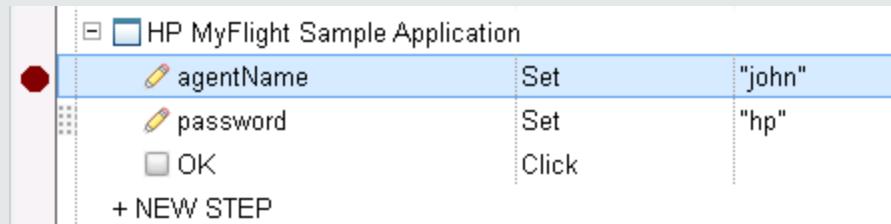
# How to Use Breakpoints

**Relevant for: GUI tests, scripted GUI components, function libraries, user code files, and business process tests**

The following steps describe how to set breakpoints, and temporarily enable or disable them. After you finish using them, you can remove them from your document.

## Example of Inserting a Breakpoint in an Action

UFT automatically places the breakpoint next to the appropriate item for the step. In the example shown below, even if you click next to the **HP MyFlight Sample Application** window item, the breakpoint is automatically inserted next to the **userName** edit item, on which the step is actually performed. When you collapse items, the breakpoint icons remain in the left margin next to the closest visible item, so you can see that the test contains breakpoints.



A screenshot of the UFT interface showing a test action. The action is titled "HP MyFlight Sample Application". It contains three steps: "agentName" (Set, "john"), "password" (Set, "hp"), and "OK" (Click). A red circular breakpoint icon is positioned in the left margin next to the "agentName" step. The "OK" step is collapsed, but the breakpoint icon remains visible in the margin.

HP MyFlight Sample Application			
Step	Action	Condition	Value
agentName	Set		"john"
password	Set		"hp"
OK	Click		
+ NEW STEP			

This task includes the following sections:

- ["Set a breakpoint" below](#)
- ["Enable or disable a breakpoint" on the next page](#)
- ["Enable or disable all breakpoints" on the next page](#)
- ["Remove a single breakpoint or all breakpoints" on the next page](#)

## Set a breakpoint

To set a breakpoint, do one of the following:

- Click in the left margin of a step in the document where you want the run to stop.
- Select a step and press F9.
- Select the line where you want the run to stop and select **Run > Insert/Remove Breakpoint**.

The breakpoint symbol  is displayed in the left margin adjacent to the selected step.

### Note:

- If you run a test using the Run automation method, the test does not stop at breakpoints even if they are saved in the test.

- If you run a test with breakpoints using the Run automation method, the breakpoints remain visible but are ignored during the test run.
- If you are running a test from the ALM Test Lab module in **hidden mode** (as specified in the UFT Remote Agent), UFT will not stop the test at the breakpoints.
- If you are running a test from the ALM Test Plan module not in hidden mode, the test stops at breakpoints if you select the **Run Test Sets in debug mode** option in the UFT Remote Agent

## Enable or disable a breakpoint

To enable/disable a specific breakpoint, do one of the following:

- Right-click the step containing the breakpoint and select **Enable/Disable Breakpoint**.
- In the Breakpoints Pane, select the breakpoint you want to enable or disable and press the **Disable/enable breakpoint** button .

## Enable or disable all breakpoints

To enable/disable all breakpoints, select **Run > Enable/Disable All Breakpoints**. If at least one breakpoint is enabled, UFT disables all breakpoints in the document. Alternatively, if all breakpoints are disabled, UFT enables them.

## Remove a single breakpoint or all breakpoints

To remove a single breakpoint, click the breakpoint icon in the left margin of the step. The breakpoint symbol is removed from the left margin of the document.

To remove all breakpoints, do one of the following:

- Select **Run > Clear All Breakpoints**.
- In the Breakpoints Pane, click the **Remove all** button  or right-click and select **Remove all**.

All breakpoint symbols are removed from the left margin of the document.

## Navigate to a specific breakpoint

1. In the Breakpoints Pane, select the specific breakpoint to which you want to navigate.
2. Do one of the following:
  - Double-click the line containing the breakpoint name.
  - Click the **Go to source** button .
  - Right-click the line containing the breakpoint and select **Go to Source**.

The cursor flashes in the main document window in the line containing the breakpoint.

# Run Errors

## **Relevant for: GUI actions, scripted GUI components, and function libraries and user code files**

Run errors are treated differently for GUI and API tests:

<b>For GUI tests and components</b>	<p>There are two types of Run Error message boxes that can be displayed during a run session:</p> <ul style="list-style-type: none"><li>• A pure syntax error. When a syntax run error message box is displayed, click <b>OK</b> in the message box and address the error.</li><li>• The other message box can be displayed in a number of situations. It offers information about the error and a number of buttons for dealing with errors encountered.</li></ul>
<b>For API tests and components</b>	Run-time errors are displayed in the UserLogger build log when running the test or in the run results.

# Troubleshooting and Limitations - Debugging

## Relevant for: GUI actions and scripted GUI components, and function libraries

This section describes troubleshooting and limitations for debugging.

- If a run-time error occurs during a run session, you can click **Skip** in the error message box that opens, to skip the problematic step and continue the run session. However, during a debugging session, if you set a breakpoint on a line immediately after a line that causes a run-time error, UFT does not stop at this breakpoint when the run session continues after the error.

**Workaround:** Set the breakpoint at least two lines after the line that causes the run-time error.

- When debugging a test that is running from ALM, use the F9 key and **Debug** menu command to toggle breakpoints. Clicking the left banner in the Editor to toggle breakpoints is not supported in this scenario.
- **GUI testing only:** UFT may stop responding during a debug run session in the following scenario: If there is a breakpoint at a function call from an associated function library, and you open the function library after the run stops at the breakpoint, and then you insert or remove a breakpoint on the first line of the called function.
- **GUI testing only:** You might not be able to run the Microsoft Visual Studio debugger if UFT is configured to record and run your test or component on any open Windows-based application. For tests, you can configure this by selecting **Run > Run Settings > Windows Applications**.

**Workaround:** Do one of the following:

- In the **MicIPC** section of the mercury.ini file (located in %SYSTEMROOT%) add these entries:

```
devenv.exe=0
msdev.exe=0
msscrdbg.exe=0
```

- Open the debugging program from a user account other than the one running UFT.
- **GUI testing only:** When Microsoft PDM 9.x or later is installed on your computer, if you add action automation objects to the **Debug > Watch** pane and then close and reopen your test without closing and opening UFT, these actions may not load successfully.

**Workaround:** Restart UFT and open your test.

- In some scenarios, when adding or viewing complex objects (objects with multiple levels) in the Watch or Local Variables panes, UFT reports an "Out of memory" exception.

**Workaround:** Register a newer version of Microsoft pdm.dll than the one provided with the UFT installation. You can find the current pdm.dll version at this registry key: **HKEY\_CLASSES\_ROOT\{78A51822-51F4-11D0-8F20-00805F2CD064}\InprocServer32**.

To register a newer version of the pdm.dll, do the following:

- a. Locate the copy of the pdm.dll that you want to use. (This file is installed and registered with Microsoft Visual Studio and Microsoft Office. It is also installed with Internet Explorer 8 and above but not registered.) You can usually find this file at **C:\Program Files (x86)\Internet Explorer or C:\Program Files\Internet Explorer**.
  - b. Move the pdm.dll file and the msdbg.dll file from this folder to a different location (registering these dll files from the original location behaves differently.)
  - c. Run the following commands:
    - regsvr32 <full path to pdm.dll>\pdm.dll
    - regsvr32 <full path to msdbg2.dll>\msdbg2.dll
- If you are running a test from the ALM Test Lab module in **hidden mode** (as specified in the UFT Remote Agent, UFT will not stop the test at the breakpoints.  
If you are running a test from the ALM Test Lab module not in hidden mode, the test stops at breakpoints if you select the **Run Test Sets in debug mode** option in the UFT Remote Agent.
  - If you run a BPT test from the ALM Test Plan module or from UFT, UFT stops the test at all breakpoints in both **Debug** and **Normal** modes.
  - When using the floating tooltip to evaluate a **With...End With** statement, UFT only recognizes the first level of the steps contained within this block.
  - When using the Quick Watch on computers running Windows 8.X or higher or Windows Server 2012, if you are watching an Insight object, the object must be visible in your application to display the tooltip. If the object is not visible, UFT automatically switches to the parent object of the Insight object and does not display the tooltip.
  - If you want to evaluate an array expression using the Quick Watch, you must highlight the array expression to display the floating tooltip. For example, if your expression is arr1(2), UFT will display the value of the third element in the tooltip unless you highlight the arr1 expression without the 2.

# Chapter 59: Running Tests with the Runtime Engine

**Relevant for: GUI tests and components, API testing, and business process tests and flows**

In addition to running tests with the full UFT IDE, you can also run tests with only the Runtime Engine installed. This option provides you the ability to run any type of UFT test without the need to install the entire UFT interface.

## Part 9: UFT Integration With HP ALM

# Chapter 60: ALM Integration

**Relevant for: GUI tests and components and API testing**

**Note:** Unless otherwise specified, references to **Application Lifecycle Management** or **ALM** apply to all currently supported versions of ALM and Quality Center. Note that some features and options may not be supported in the specific edition of ALM or Quality Center that you are using.

For details, see the *HP Unified Functional Testing Product Availability Matrix* and the *HP Application Lifecycle Management User Guide* or the *HP Quality Center User Guide*.

This chapter includes:

• <a href="#">ALM Integration Overview</a> .....	916
• <a href="#">How to Work with Tests and Components in ALM</a> .....	917
• <a href="#">Running UFT Tests from ALM</a> .....	920
• <a href="#">Running Tests in Server-Side Execution</a> .....	920
• <a href="#">AUT Environment Parameters</a> .....	921
• <a href="#">How to Run a Test Using Server-Side Execution</a> .....	922
• <a href="#">ALM Template Tests</a> .....	925
• <a href="#">How to Create a Template GUI Test</a> .....	926
• <a href="#">How to Create a GUI Test in ALM Using a Template Test</a> .....	926
• <a href="#">Data Awareness in ALM</a> .....	928
• <a href="#">How Does Data Awareness Work?</a> .....	929
• <a href="#">Advantages of Data Awareness</a> .....	930
• <a href="#">Considerations for Data Awareness</a> .....	930
• <a href="#">How to Data Drive a Test Using Data Stored in ALM</a> .....	931
• <a href="#">How to Enable the BPT Remote Agent</a> .....	935
• <a href="#">How to View or Modify Remote Agent Settings</a> .....	936
• <a href="#">How to Install a Certificate for ALM Servers in a Environment Using External Authentication</a> .....	937
• <a href="#">Troubleshooting and Limitations - General ALM Integration</a> .....	937
• <a href="#">Troubleshooting and Limitations - ALM Integration with GUI Testing</a> .....	940
• <a href="#">Troubleshooting and Limitations - ALM Integration with API Testing</a> .....	941

# ALM Integration Overview

## **Relevant for: GUI tests and components and API testing**

UFT integrates with ALM, the HP centralized quality solution. ALM helps you maintain a project of all kinds of tests (such as tests, components, business process tests, manual tests, tests created using other HP products, and so on) that cover all aspects of your application's functionality. Each test or component in your project is designed to fulfill a specified testing requirement of your application. To meet the goals of a project, you organize the tests in your project in unique groups.

ALM provides an intuitive and efficient method for scheduling and running tests or components, collecting results, analyzing the results, and managing test and component versions. It also features a system for tracking defects, enabling you to monitor defects closely from initial detection until resolution.

At its most basic level, integrating UFT with ALM enables you to store and access tests, components, application areas, and resource files in an ALM project, when UFT is connected to ALM. In addition, if you have the UFT Add-in for ALM installed on your computer, you can create and edit UFT tests and components directly from ALM. After you have created and edited tests and/or components, you can run them from ALM, and view the run results directly in ALM.

When UFT is connected to ALM, you can:

<b>Create tests, components, and resources and save them in your ALM project.</b>	You can save test and components in your project and thereby make them accessible to multiple users. Any changes made by any user are then saved and updated for all users of the test.
<b>View the contents of your tests.</b>	This can help you decide if you want to run a test as part of a test set. Note that the Test Flow in ALM and the canvas in UFT display only the actions that are run when the currently selected test runs. This means that if a nested action is commented out, for example, that action is not displayed in ALM or in the UFT canvas. You can uncomment it in the UFT Editor when needed.
<b>Run your tests and components and view the results in ALM.</b>	In much the same way as saving a test in ALM enables all users to use and see the changes, running a test in ALM enables all users of the ALM project to see the run results of a particular test. You can also use these run results to automatically or manually add defects to your ALM project.
<b>Associate a test, an API component, or a GUI component's application area with external files stored in the Test Resources module of an ALM project.</b>	When you save the resources files for a test or component in your ALM project, it enables you to save just one copy of the resource and link it to multiple tests or components.
<b>Associate external files for all tests or for a single test.</b>	For example, suppose you set the shared object repository mode as the default mode for new GUI tests. You can instruct UFT to use a specific object repository stored in the Test Resources module in ALM. Likewise, you can set the default activity repository location for your custom API test activities in the Test Resources module in ALM.
<b>Take advantage of all the</b>	For details, see " <a href="#">Resources and Dependencies Model</a> " on page 942.

<b>features provided with the Resources and Dependencies model.</b>	
<b>Use the QCUtil object to access and use the full functionality of the ALM OTA (Open Test Architecture)</b> (GUI testing only)	This enables you to automate integration operations during a run session, such as reporting a defect directly to an ALM database. For details, see the Utility Objects section of the <i>HP UFT Object Model Reference for GUI Testing</i> and the ALM Open Test Architecture documentation.
<b>Use the TDOTA object in your automation scripts to access the ALM OTA</b> (GUI testing only)	For details, see the <i>HP UFT Object Model Reference for GUI Testing</i> or the <b>HP UFT GUI Testing Automation and Schema References Help &gt; Automation Object Model Reference</b> .

For a list of required access permissions for working with ALM, see "[Required access permissions](#)" on [page 29](#).

## How to Work with Tests and Components in ALM

### **Relevant for: GUI tests and components and API testing**

The following steps describe the workflow of how to work with tests and components saved in an ALM project.

This task includes the following steps:

- "[Prerequisites for Windows 7, Windows 8, Windows Server 2008 R2, and Windows Server 2012 Users](#)" on the next page
- "[Connect to an ALM Project](#)" on the next page
- "[Enable ALM to run tests or components](#)" on page 919
- "[Enable full access to tests from ALM](#)" on page 919
- "[Create a template test \(GUI testing only\) - optional](#)" on page 919
- "[Create a test using a template test \(GUI testing only\) - optional](#)" on page 919
- "[Set UFT Remote Agent Preferences](#)" on page 919
- "[Run your test or component in the ALM project](#)" on page 919
- "[Manage versions of your project using version control - optional](#)" on page 919
- "[Disconnect from the ALM project](#)" on page 919

## Prerequisites for Windows 7, Windows 8, Windows Server 2008 R2, and Windows Server 2012 Users

The security settings in Windows 7 and Windows Server 2008 R2 may prevent you from connecting to an ALM project.

This can occur when the UAC (User Account Control) option is set to ON, and you have never connected to an ALM project.

To connect to ALM for the first time, you must disable the UAC option and restart your computer. After you successfully connect to ALM, you can turn the UAC option on again. Thereafter, you should be able to connect to ALM, as needed. For details, see ["Tools in the HP UFT program group" on page 1138](#).

### Connect to an ALM Project

1. In the toolbar, click the **ALM Connection** button . The ALM Connection dialog box opens.

2. In the ALM Connection dialog box, enter the following information:

- **Server URL:** the URL in which you access the ALM project in your browser

**Note:** Ensure that the format you are using for the URL is the same as the URL you use to access ALM via your browser. For example, if you use the IP address to access the ALM server via the browser, you should use the IP address to access ALM via UFT.

- **User Name**

- **Password**

3. Click **Connect** to connect to the ALM server. UFT pauses for a few moments to connect with the ALM server.

**Note:** After you have connected to the server, you must also connect to a specific project to access your tests and components.

4. In the lower part of the ALM Connection dialog box, select the following:

- **Domain**

- **Project**

5. Click **Connect** to access your ALM project.

6. Click **Close** to close the ALM Connection dialog box and begin working with your tests and components.

**Note:** If you are connecting to an ALM server using external authentication, you are prompted as part of the login to select your external certificate.

## Enable ALM to run tests or components

In UFT, select the **Allow other HP products to run tests and components** option in the **Test Runs** pane of the Options dialog box (**Tools > Options > GUI Testing tab > Test Runs** node).

For security reasons, remote access to your UFT application is not enabled by default. This option enables ALM (or other remote access clients) to open and run tests.

## Enable full access to tests from ALM

Install the Unified Functional Testing Add-in for ALM or BPT is installed on your ALM client computer. This enables you to view the test and view the run results.

You can install the Add-in from:

- The main UFT installation Start screen, by selecting the Unified Functional Testing Add-in for ALM
- The ALM Add-ins page by choosing **Help > Add-ins Page > HP ALM Connectivity** in ALM

**Note:** After you install the UFT Add-in for ALM as part of the standard installation, you must also install the Microsoft Visual C++ 2005 SP1 Redistributable Package on your computer. You can download this file from <http://www.microsoft.com/en-us/download/details.aspx?id=5638>.

## Create a template test (GUI testing only) - optional

Perform this step to create a template test that has pre-defined test settings. You can then use this template test when creating new tests in ALM. For details, see "How to Create a Template GUI Test" on page 926.

## Create a test using a template test (GUI testing only) - optional

Perform this step to create a test that has the pre-defined test settings defined in a template test. For details, see "How to Create a GUI Test in ALM Using a Template Test" on page 926.

## Set UFT Remote Agent Preferences

Use the Remote Agent to view or modify the settings that UFT uses when ALM runs a test or business process test on your computer.

## Run your test or component in the ALM project

See "How to Run a Test or Component" on page 835.

## Manage versions of your project using version control - optional

For details, see "Managing Versions of Assets in ALM" on page 952.

## Disconnect from the ALM project

1. In the toolbar, click the **ALM Connection** button .

2. In the ALM Connection dialog box, in the **Login to project** section, click the **Logout** button.
3. In the **Connect to server** section, click the **Disconnect** button.
4. Click **Close** to close the ALM Connection dialog box and continue working with UFT.

**Note:** When you disconnect from a project, all open documents from the project automatically close.

## Running UFT Tests from ALM

### Relevant for: GUI tests only

If UFT is connected to ALM, you can run tests that are stored in an ALM database via:

- UFT
- ALM client that is installed on your computer
- A remote ALM client

**Note:** ALM cannot run tests using UFT if the UFT computer is logged off or locked.

Before an ALM client can run GUI tests on your computer, you must give ALM permission to use UFT (described in ["How to Work with Tests and Components in ALM" on page 917](#)).

When you run a test or business process test from ALM, the UFT Remote Agent opens on the UFT computer where the test runs. The settings in the UFT Remote Agent determines how UFT behaves when a test is run by a remote application such as ALM.

When ALM runs your test, it uses the associated add-ins list to load the proper add-ins for your test on the UFT computer. You can view the associated add-ins in the **Properties** pane of the Test Settings dialog box or the Properties pane.

By default, UFT opens and runs in hidden mode when ALM activates it to run a test in a test set. This helps to improve performance. While UFT is running in hidden mode, you can open UFT to view the steps being run in the Editor by clicking the  button in the status bar. If you do not want UFT to run in hidden mode, you can change this setting in the Remote Agent.

## Running Tests in Server-Side Execution

### Relevant for: GUI tests and API testing

You can run tests in server-side execution when the tests are saved on a Lab Management-enabled ALM server.

Server-side execution enables ALM to run UFT tests on remote hosts at predefined times or on an ad-hoc basis, without anyone logged in to the host to initiate and control the test runs.

In contrast, client-side execution requires a user to be logged in to the host computer on which ALM runs the UFT test.

To set up server-side execution on ALM:

1. In the **Testing > Test Lab** module, create functional test sets. A functional test set is a group of automated tests or test configurations in an ALM project, designed to achieve a specific goal.
2. In the **Lab Resources > Testing Hosts** module, select hosts upon which the tests can run remotely.
3. If you want to use different values for specific parameters when running the tests on different environments or situations, you can define AUT parameters in the **Lab Resources > AUT Environments** module.
4. In the **Testing > Timeslots** module, schedule automatic test runs, or reserve timeslots to use for running manually.

**Tip:** You do not need to reserve a timeslot if you run the tests ad-hoc.

For additional information, see the *HP Application Lifecycle Management User Guide*.

In UFT, you can link your test parameters to ALM AUT Environment parameters. This enables the test to use AUT environment parameter values passed from ALM when your UFT test runs in server-side execution. For details, see "[AUT Environment Parameters](#)" below.

**Note:** Server-side execution is available only for ALM Edition and Performance Center Edition, version 11.50 or later. You must also have Lab Management support for the ALM project.

For task details, see "[How to Run a Test Using Server-Side Execution](#)" on the next page.

## AUT Environment Parameters

### Relevant for: GUI tests and API testing

In ALM, you can define AUT environments, which contain AUT environment parameters, to be used in tests that run using server-side execution. When ALM runs the tests, it passes the values for the AUT parameters to the test.

To create multiple sets of parameter values for your test runs, you define multiple AUT environment configurations in your ALM project. This enables you to use different values for various data in your test, depending on the application, situation, or environment on which the test runs. For example, you may want to run the same test on different Web-based databases, each requiring a different URL, user name, and password.

In UFT, you can link test parameters to AUT parameters defined in a specific AUT environment in ALM. Subsequently, when ALM runs the test in server-side execution, the test parameters will receive the AUT environment parameter values passed from ALM.

- When the test runs in server-side execution, it uses the values provided by ALM for the corresponding AUT environment parameter.

- When the test runs from UFT, the test uses the parameter's default values.

For details on how to link test parameters to AUT parameters, see "[How to Run a Test Using Server-Side Execution](#)" below

**Note:** All of the AUT parameters that you use must be from the same ALM AUT environment.

If you save the test to the file system, the links to the AUT parameters are removed, and the test parameters become ordinary parameters. The default values remain defined.

In the Properties pane, an ALM icon  next to a parameter's default value indicates that it is an AUT parameter.

Input	Default Value
Properties	
Region	127.0.0.14
Expiration	0001-01-01T00:00:00.000
ip	 127.0.0.13

## What happens if the ALM AUT parameters linked to the UFT test parameters are deleted from ALM?

- If you open a GUI test with parameters linked to deleted AUT parameters, UFT displays a message specifying the missing parameters, and the AUT environment configuration to which they belonged. The the links to the missing AUT parameters are removed, and those test parameters become ordinary parameters.
- If you open an API test with test parameters that are linked to deleted ALM AUT parameters, the links remain unchanged.
- If you run a test with test parameters linked to deleted ALM AUT parameters, the test will fail to run.

# How to Run a Test Using Server-Side Execution

## Relevant for: GUI tests and API testing

This task describes how to run a UFT test from ALM, using server-side execution.

For an overview, see "[Running Tests in Server-Side Execution](#)" on page 920.

This task includes the following steps:

- "[Prerequisites](#)" on the next page
- "[Create tests and save them in ALM](#)" on the next page
- "[Create functional test sets in ALM](#)" on the next page
- "[Set up AUT Parameters in ALM and link your test parameters to them in UFT - optional](#)" on the next page

- "Set up hosts in ALM on which the UFT tests can run " on the next page
- "Schedule the tests in ALM - optional" on the next page
- "Run the tests from ALM" on page 925

Most of the steps in this task must be performed on ALM. In UFT, you can link your test parameters to AUT parameters defined in ALM.

## 1. Prerequisites

- Connect to an ALM project with Lab Management support.
- Make sure that HP ALM Lab Service is installed on the UFT computer.

For details, see the ALM Lab Management Guide.

## 2. Create tests and save them in ALM

Create your tests in UFT or ALM, and save them in ALM.

## 3. Create functional test sets in ALM

- In ALM, in the **Testing > Test Lab** module, create a functional type test set and specify the required information.
- Select the **Execution Grid** tab and click **Select Tests**.
- In the Test Plan tree, select the tests you want to add to the test set.

## 4. Set up AUT Parameters in ALM and link your test parameters to them in UFT - optional

To use parameter values from value sets stored in ALM:

- In ALM, in the **Lab Resources > AUT Environments** module, define AUT environments with parameters and set values for the parameters. For details, see the *HP Application Lifecycle Management User Guide*.
- In UFT, open the Properties pane to the test parameters tab by doing one of the following:
  - Open an API test, select the **Start** or **End** steps in the canvas, and open the Properties pane. Then select the Test Input/Output Parameters tab .
  - Select a GUI test in the solution explorer. Open the Properties pane, and select the Parameters tab .
- Click **Add > Add Input Parameter**.
- In the Add Input Parameter dialog box, click the **Select ALM application parameters** button .
- In the Select AUT Parameter dialog box, expand the **AUT Environment** node and select a parameter. Click **OK**.

- f. In the Add Input Parameter dialog box, provide a name for the parameter and set the default value, if necessary. Click **OK**.

In the Properties pane, an ALM icon  next to a parameter's default value indicates that it is an AUT parameter.

Input	Default Value
Properties	
Region	127.0.0.14
Expiration	0001-01-01T00:00:00.000
ip	 127.0.0.13

- g. To edit the parameters, click the **Edit Parameter** button  in the Properties pane.
- h. To change the parameter to an ordinary parameter, without a linkage to an AUT environment, click the **Remove link to ALM application parameters** button  in the Edit Parameter dialog box .

You can repeat these steps to add additional test parameters and link them to ALM AUT parameters, but all of the AUT parameters that you use must be from the same ALM AUT environment.

For more details, see "[AUT Environment Parameters](#)" on page 921.

## 5. Set up hosts in ALM on which the UFT tests can run

In ALM, in the **Lab Resources > Testing Hosts** module, you can view the hosts defined in the Lab Management project for all projects on your ALM server. Optionally, you can define additional host machines for your project.

When you define a new host, define its **purpose**, for example, QuickTest or Service Test, to indicate the type of test to run on this host. For details, see the *HP Application Lifecycle Management User Guide*.

### Notes:

- Make sure that UFT is installed on the host computers you create or define in ALM.
- If UFT is not available in the list of purposes that you can select for a host, select **QuickTest Professional** for GUI tests or **Service Test** for API tests. Select both if you plan to run both types of tests.
- For each UFT host on which you want to run GUI tests, enable the **Allow other HP products to run tests and components** option in UFT.

## 6. Schedule the tests in ALM - optional

In ALM, in the **Testing > Timeslots** module, schedule times for the tests to run automatically, or

reserve timeslots to use later to run the tests manually.

If you are running the tests ad-hoc, you can skip this step.

## 7. Run the tests from ALM

In ALM's **Testing > Test Lab**, specify the hosts on which you want to run the tests, and run the tests. For details, see the *HP Application Lifecycle Management User Guide*.

You can also optionally automatically upload your run results to ALM if you are running a test from ALM. This option is set in ALM as a site parameter for your project. For details, see the *HP Application Lifecycle Management Administrator Guide*.

# ALM Template Tests

## Relevant for: GUI tests only

Template tests serve as the basis for all tests created in ALM. A **template test** is a test that contains default test settings. For example, a template test might specify the UFT add-ins, associated function libraries, and recovery scenarios that are associated with a test. You can modify these test settings in the Test Settings dialog box (**File > Settings**) in UFT.

In addition to default test settings, a template test can also contain any comments or steps you want to include with all new tests created in ALM. For example, you may want to add a comment notifying users which add-ins are associated with the template test, or you may want to add a step that opens a specific Web page or application at the beginning of every test. Any steps or comments you add to a template test are included in all new tests created in ALM that are based on that template test.

All template tests are saved in your ALM project (except for the default template test, which is located on the ALM client) and do not need to be copied to each user's local computer. This enables users to customize their local default template tests, if needed, and still have access to globally maintained template tests. When an ALM user creates a new test in ALM, the default template test for the installed UFT version is automatically associated with the test unless the user selects another template test.

A default template test is installed on each ALM client when the Unified Functional Testing Add-in for ALM is installed in the <Unified Functional Testing folder>\bin\Templates folder on your computer (for example: %ProgramFiles%\HP Software\Unified Functional Testing\bin\Templates\Template1). Because this test is installed locally, any changes you make in the template test are applied only to the tests created on your computer (using the ALM client).

When tests based on a specific template test are run from ALM, UFT automatically loads the associated add-ins and applies the required settings, as defined in the test.

For details on how to create and work with template tests, see "[How to Create a Template GUI Test](#)" on the next page and "[How to Create a GUI Test in ALM Using a Template Test](#)" on the next page.

# How to Create a Template GUI Test

## Relevant for: GUI tests only

This task describes how to create a template GUI test. For an overview, see "[ALM Template Tests](#)" on the previous page.

### How to create a template GUI test in UFT

1. Open an existing test with the required add-ins loaded. For details on loading UFT add-ins, see the section on loading UFT add-ins in the *HP Unified Functional Testing Add-ins Guide*.

**Note:** Make sure that the add-ins included in the opened test are actually installed on the UFT computer on which the test will eventually run. Otherwise, when the test is run, UFT will not be able to load the required add-ins and the test may fail.

2. Define the required settings in the Test Settings dialog box (**File > Settings**). For details, see "[Document Settings](#)" on page 81.
3. If you want to include comments or steps in all tests based on this template test, add them.
4. Select **File > Save As**. In the Open Dialog box, save the test to your ALM project using a descriptive name that clearly indicates its purpose.

#### Tip:

- In the ALM test plan tree (Test Plan module), you may want to create a special folder for your template tests. This enables other users to quickly locate the relevant template test when they create new tests in ALM.
- When you save the test in UFT, you should apply a descriptive name that clearly indicates its purpose. For example, if the template test is used to associate the ActiveX and Web Add-ins with a new test, you could call it ActiveX\_Web\_Addins\_Template.

### How to create a template GUI test in ALM

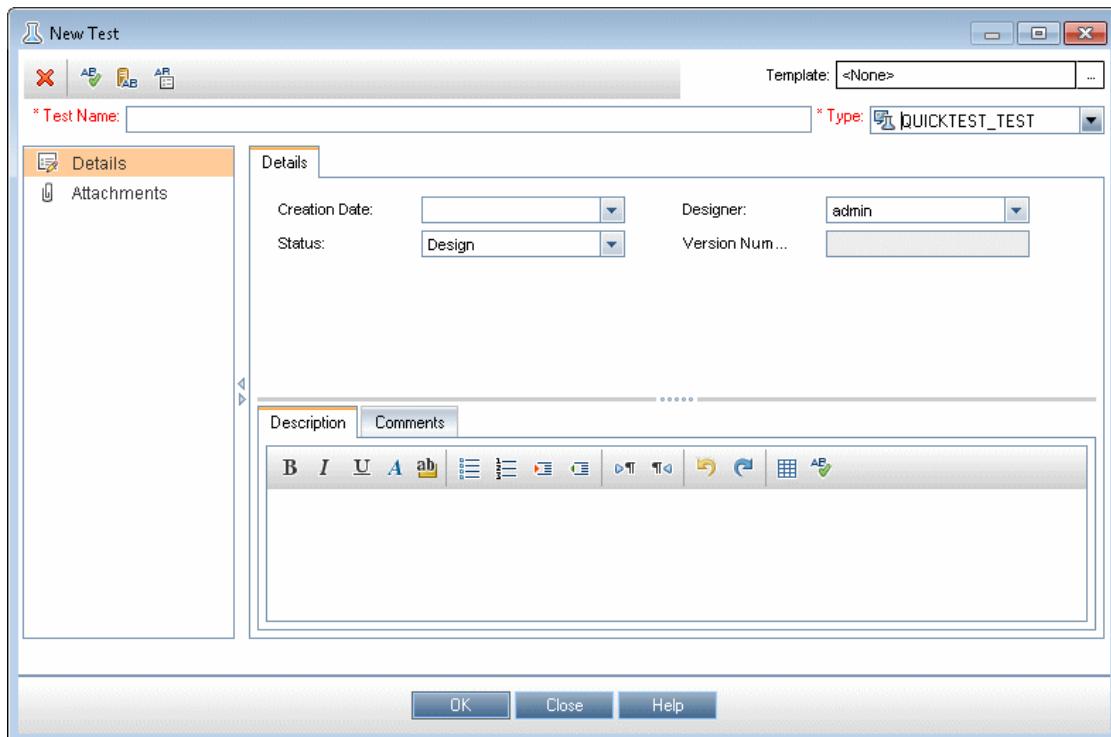
1. Open the project in ALM, click the **Test Plan** button  on the sidebar to open the Test Plan module, and browse to the test you previously created in UFT and saved in your ALM project.
2. Right-click the test and select **Mark as Template Test**. The test is converted to a template test.

# How to Create a GUI Test in ALM Using a Template Test

## Relevant for: GUI tests only

This task describes how to create a GUI test in ALM using a template test as its basis.

1. In ALM, click the **Test Plan** button  on the sidebar to open the Test Plan module.
2. In the test plan tree, select a folder.
3. Click the **New Test** button  or select **Test > New Test**. The Create New Test dialog box opens.



**Note:** The **Template** list is displayed only if the **Unified Functional Testing Add-in for ALM** is installed on your computer. If the **Template** box is not displayed, you must install the **Unified Functional Testing Add-in for ALM** from the Unified Functional Testing DVD or from the More ALM add-ins page (opened from the ALM Options window, or from **Help > Add-ins Page** in ALM).

4. From the **Test Type** list, select **QUICKTEST\_TEST**.
5. In the **Test Name** box, type a name for the test start using English (Roman) letters, numbers, and underscores (if needed). For a list of naming conventions, see "[Troubleshooting - Naming Conventions](#)" on page 1144.
6. Click the **Template** box browse button. The Select Tests dialog box opens.
7. Expand the folder containing your template test.
8. Select the template test on which to base your new test and click the **Add** button . The Select Tests dialog box closes and the template test you selected is displayed in the **Template** box (in the Create New Test dialog box).
9. In the Create New Test dialog box, click **OK**. The new test is created with the test settings defined in the template test.

10. The new test is displayed in the Test Plan tree under the subject folder you selected.

**Note:** If the Required Fields dialog box opens, set the required values and click **OK**. For details, see the ALM administrator guides.

11. Continue creating the test. For details on creating tests in ALM, see the *HP Application Lifecycle Management User Guide*.

## Data Awareness in ALM

### Relevant for: GUI tests and components and API testing

**Note:** The data awareness feature is available when integrating UFT with ALM. It requires the Unified Functional Testing Add-in for ALM or the QuickTest Professional Add-in for ALM.

In ALM, you can upload and store your Microsoft Excel (.xls) test data resource files in the Test Resources module, enabling you to use the same data for multiple tests or components, or different data for each test run.

In ALM, you run **configurations** instead of standard tests. When working with data-driven tests in ALM, each **configuration** is a test that is set to run with a selected data resource file and optional data filter settings. The filter settings enable you to filter data by specified row numbers, by parameter text values, or both.

**Note:** You define and manage configurations only in ALM.

### Example

Suppose you define three configurations in a single test to test an application that provides different solutions for Gold Card, Silver card, and Bronze Card users. You could use the same data resource for each configuration by filtering the rows or text of the data resource to match the relevant user type. Similarly, if your data is stored in various .xls or .xlsx files, you might want to run the same test using a different data source each time. You would do this by associating a different data source with each configuration.

By managing your data as a resource in ALM, you can see at a glance which data resource files are associated with a particular test or configuration, and which tests or configurations are using a specific data resource. For more details, see "[Advantages of Data Awareness](#)" on page 930.

To learn more, see:

- [How Does Data Awareness Work?](#) ..... 929
- [Advantages of Data Awareness](#) ..... 930
- [Considerations for Data Awareness](#) ..... 930

## How Does Data Awareness Work?

### Relevant for: GUI tests and API tests

When you create a test in ALM (or save a test to an ALM project from UFT for the first time), a default **configuration** with the same name as the test is created simultaneously. You can view this configuration in the **Configuration** tab of the **Test Plan** module.

In the ALM Test Resources module, you create one or more data resources and upload one Microsoft Excel (.xls or .xlsx) file for each.

After you upload a data resource file to your ALM project, you can define the test-level configuration settings for a particular test. You do this in the Parameters tab of the Test Plan module by selecting the data resource file and mapping its column names to the data table parameters in your test. You can associate one data resource with the test-level configuration. By mapping the column names to data table parameters, you enable UFT to identify and use the correct parameter value during a run session.

#### Note:

##### For GUI testing:

- Specifying a test-level data resource in the Parameters tab of the ALM Test Plan module overrides the **Data pane** settings in the Resources pane of the Test Settings dialog box.
- The data table parameters in your test must be defined in the **Global** sheet.

##### For API testing:

When running a test, API Testing's Data Navigation feature looks at the data resulting from the specific configuration. For example, if your ALM configuration filters the data to use only rows 3 through 5, and the API Test Navigation says to iterate rows 2 through 3, the actual data used will be rows 4 and 5 from the test resource.

In the configuration tab of the Test Plan module, you can create as many additional configurations as needed. By default, every configuration uses the test-level configuration settings unless you override the data resource.

If you associate a configuration with the data resource defined in the Parameters tab, you do not need to map the data table parameters to column names. You can, however, modify the filter settings by applying text filters to any parameter, and/or selecting the rows on which the configuration can run.

If you associate a different data resource with a configuration, or you select to override the test data resource and then select the same data resource file that you selected in the Parameters tab (which is the same as selecting a different data resource), you need to map the data table parameters to the column names in the .xls or .xlsx file. You can also apply filter settings, as described previously.

If the data table parameter names in the **Global** sheet and the column names in the associated data resource are identical, you do not need to perform any mappings.

Next, in the Test Lab module, you can run any or all of the configurations defined for a test (or associated with a requirement) by adding them to a test set. When you run a configuration containing steps that use data table parameters, the parameter values are retrieved from the data resource files according to the settings defined for that configuration.

For a task describing how to work with configurations, see "[How to Data Drive a Test Using Data Stored in ALM](#)" on the next page.

## Advantages of Data Awareness

### **Relevant for: GUI tests and API testing**

Using the Data Awareness feature in ALM provides a number of advantages:

<b>Test reuse</b>	You can use the same test to test various scenarios, each time with a different set of data. You can do this associated a different data resource with each configuration, by associating a single data resource with different configurations and then filtering each configuration, or by using a combination of both.
<b>Data reuse</b>	You can associate the same data resource with multiple tests or configurations.
<b>Ease of Management</b>	All of your resources are stored in one central location.
<b>Visibility</b>	You can see which tests are using a particular data table, and you can see which data table each test is using.
<b>Requirements Coverage</b>	You can cover all of your testing requirements by linking them to specific configurations. This enables you, for example, to use a single test to cover multiple requirements by associating different configurations in the same test with each requirement.
<b>Resources and Dependencies Model</b>	By storing your data in the Test Resources module, you can take advantage of additional ALM integration features, such as: <ul style="list-style-type: none"><li>• Version control and baselines</li><li>• Asset comparison Tool and Asset Viewer</li><li>• Sharing data resources across ALM projects</li></ul>

## Considerations for Data Awareness

### **Relevant for: GUI tests and API testing**

Consider the following when managing your data resources in ALM:

- Only Microsoft Excel (.xls or .xlsx) files are supported as data resources.
- If you modify a data table parameter or a column name in the associated data resource after they are mapped to each other, you must update the mappings accordingly. Additionally, in API tests, if you update an Excel data source and link properties to the new data source, the new values will not be reflected in ALM until you remap the properties using the Map Parameters dialog box.

For details, see "[How to Data Drive a Test Using Data Stored in ALM](#)" on the next page.

- **For GUI testing only:** If the data table parameter names in your test's Global sheet are identical to the column names in the associated data resource file, you do not need to perform any mapping.

# How to Data Drive a Test Using Data Stored in ALM

## Relevant for: GUI tests, API testing, and business process tests

This task provides a general overview of the steps involved in data driving a test with data stored in ALM. After you are familiar with the steps, you can perform many of them in the order you choose. Some steps may not be necessary in all cases.

This task includes the following steps:

- "Prerequisites" below
- "Import data into a test (API testing only)" below
- "Data drive the test steps (API testing only)" below
- "Export the business process test's iteration parameter data to Excel (Business Process Testing only)" below
- "Create a data resource file in your ALM project" on the next page
- "Specify a default data table resource for all new test configurations" on the next page
- "Define your test configurations" on page 933
- "Link your configurations to requirements to create requirements coverage - Optional" on page 934
- "Run your test configuration" on page 935

### 1. Prerequisites

- a. Connect to ALM.
- b. Make sure that your test is saved in your ALM project.
- c. **For GUI testing:** Make sure you have a test that uses data table parameters from the **Global** sheet.

### 2. Import data into a test (API testing only)

- a. In the Data pane, click the **New Data Source** button  and select **Excel**.
- b. In the New/Change Excel Data Source Dialog Box, select the .xls or .xlsx file containing the data and Select the **Allow other tools to override the data** option. Click **OK** to import the data source into your test.

### 3. Data drive the test steps (API testing only)

For details, see "How to Assign Data to API Test/Component Steps" on page 551.

### 4. Export the business process test's iteration parameter data to Excel (Business Process Testing only)

For details, see "Export component parameters to an Excel" on page 1043.

## 5. Create a data resource file in your ALM project

### In the Test Resources module:

- a. Expand the Resources tree and select the required node.
- b. Select **Resources > New Resource** to add a resource under that node.
- c. In the New Resource dialog box:
  - In the **Type** list, select **Data table**.
  - In the **Name** box, enter a name for the data resource, for example, the name of the Microsoft Excel (.xls or .xlsx) file you plan to use.
  - Fill in the remaining fields (optional) and click **OK** to close the dialog box.
- d. In the Resource Viewer tab, click **Upload File**. Then browse to and upload the relevant .xls or .xlsx file.

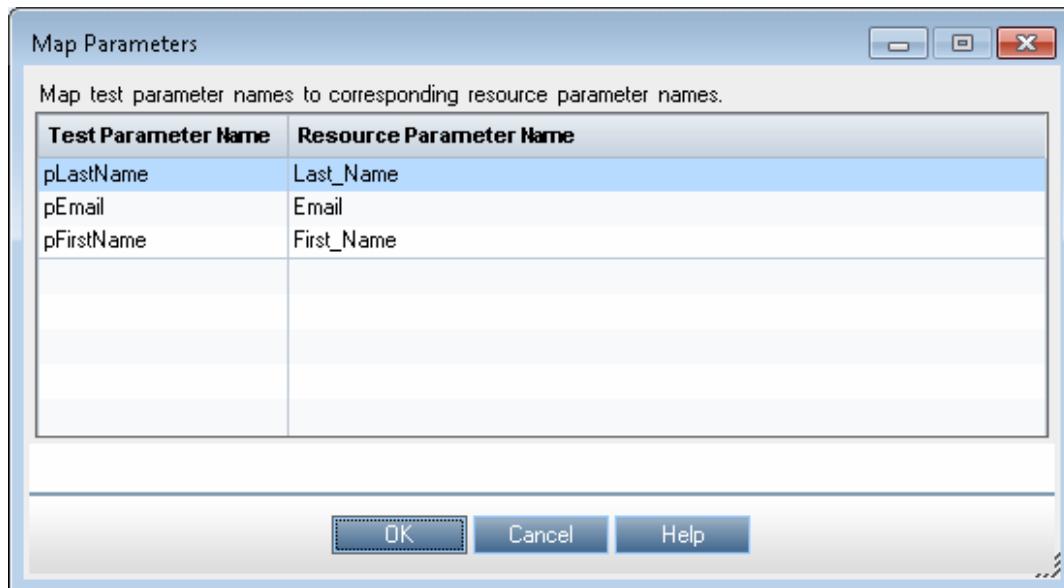
**Note:** You can convert an internal data table from an open test to an uploadable data resource file by right-clicking the Data pane, selecting **File > Export**, saving the data table to the file system as an .xls or .xlsx file, and then uploading it as described above.

## 6. Specify a default data table resource for all new test configurations

- a. In the Parameters tab of the Test Plan module, select the data table resource you want to use as the default for all test configurations.  
**For GUI testing:** If you do not specify a data table resource, the data specified in the Resources pane of the Test Settings dialog box (**File > Settings**) is used instead.

**Note:** In this tab, only the Parameter Name column is relevant for GUI tests.

- b. Click the **Map Parameters** button . In the Map Parameters dialog box, map the data table parameters (column headings) to the test parameters by entering the matching data table parameter names in the **Resource Parameter Name** column, as shown in the example below.



All new configurations use the default mappings unless you specify otherwise in the Data tab of the Configurations tab. For details, see the next step, **Define your test configurations**.

## 7. Define your test configurations

Define test configurations for various run sessions. For each configuration, you specify whether to use the default resource file that you specified in the previous step, or whether to use a different data resource file.

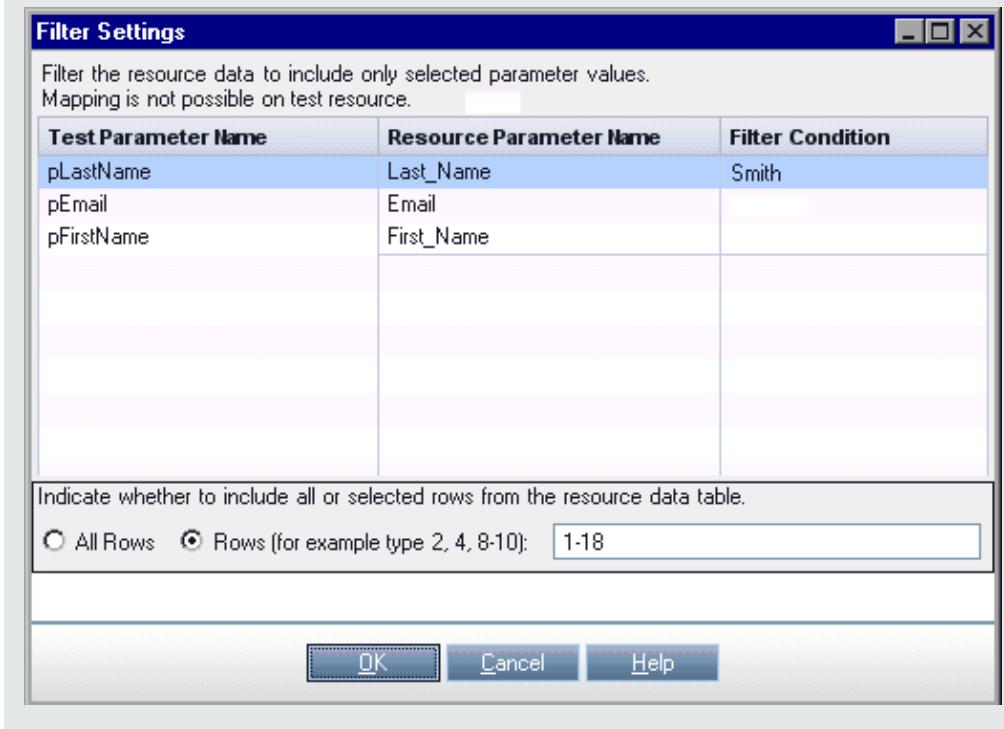
- a. In the ALM Test Plan module, browse to and select the test to which you want to associate your data table resource. For details, see the *HP Application Lifecycle Management User Guide*.
- b. Click the **Test Configurations** tab. A default configuration is displayed in the grid. This configuration was created when your test was added to the ALM project. For details on configurations, see "[Data Awareness in ALM](#)" on page 928.
- c. In the bottom pane of the Configurations tab, click the **Data** tab.
- d. In the Data tab:
  - Select the **Override test data resource** check box to select a different data resource file from the Test Resources module, or leave the check box blank to use the default resource file you selected in the Parameters tab in the previous step.
  - In the **Data Resource** box, browse to and select the relevant data resource file to associate with this configuration. (Relevant only if you selected the **Override test data resource** check box)

- Click the **Data Resource Settings** button, and in the Filter Settings dialog box:
    - Map the data table parameters from your test to the column headers in the data table file (Relevant only if you selected a different data resource file in the previous step)
    - Apply filter conditions (text strings), as needed. You can apply one filter condition to each parameter.
    - Specify the rows on which to run iterations. For example, if you run a configuration named Gold, and users of this type are listed in rows 2-114, then specify these rows only.

**Note:** If you apply filter conditions and specify rows, AND logic is used, meaning that the parameter value must equal the filter text value and the parameter value must be located in one of the specified rows.

## **Example:**

The pLastName data table parameter is mapped to Last\_Name and is filtered to include only parameter text values that equal Smith. The configuration is set to run only on rows 1-18 in the .xls or .xlsx file you uploaded as the data resource.



For details on this dialog box, see the *HP Application Lifecycle Management User Guide*.

## **8. Link your configurations to requirements to create requirements coverage - Optional**

If you want to make sure that your requirements are fully covered, link them to configurations. This enables you to select configurations to run based on requirement coverage when you plan your run session.

- a. In the Test Plan module, click the **Req Coverage** tab.
- b. Click the **Select Req** button. The Requirement Tree tab is displayed in the right pane.
- c. From the Requirement Tree tab, select a requirement to add to the Req Coverage grid. When you add the requirement, the Add Advanced Coverage dialog box opens.
- d. Select the test configurations that cover this requirement.

## 9. Run your test configuration

You run configurations from ALM.

- a. **For GUI testing:** In UFT, make sure that in the **Tools > Options > GUITesting tab > Test Runs** node, the **Allow other HP products to run tests and components** is selected.
- b. In the ALM Test Lab module, select or create a test set. For details, see the *HP Application Lifecycle Management User Guide*.
- c. In the right pane, select the **Execution Grid** tab.
- d. Click the **Select Tests** button to display the **Test Plan Tree** and **Requirements Tree** tabs in the right pane.
- e. Do one of the following to select the configurations to run:
  - From the Test Plan Tree tab, select a test to add to the **Execution Grid**. When you add the test, all of its configurations are added to the Execution Grid. (The test itself is not added to the Execution Grid because ALM runs configurations, not tests.)
  - Below the Test Plan Tree tab, expand the **Test Configurations** pane, and add the specific configurations that you want to run to the Execution Grid.
  - Below the Requirements Tree tab, expand the coverage pane, and select a test to add to the Execution Grid. When you add the test, all of its configurations are added to the Execution Grid. (The test itself is not added to the Execution Grid because ALM runs configurations, not tests.)
- f. Click the **Run** button to run the selected configurations.
- g. After the run session, click the **Launch Report** button in the Last Run Report tab to view the results.

## How to Enable the BPT Remote Agent

### Relevant for: GUI tests and components and API testing

If the **Windows Firewall** is **turned on** on your computer, and you want to enable business process tests to be run on your computer from a remote ALM client, then you must manually create a firewall exception for the remote agent.

This task describes how to create a firewall exception for the Remote Agent.

1. Make sure you open the ALM or Quality Center client at least once on your computer.
2. Run `Firewall.cpl` from the command line. The Windows Firewall dialog box opens.

**Note:** The remaining steps in this procedure may be different in different operating systems.

3. Click the **Exceptions** tab.
4. Click the **Add Program** button.
5. In the Add a Program dialog box, browse to the location where the ALM or Quality Center client is installed and select any of the following files that exist:
  - `bp_exec_agent.exe`
  - `ComWrapperRemoteAgent.exe`
  - `BptRemoteAgenApplication.exe`

**Note:** You may have to repeat this step a few times to add all relevant files.

6. Click **OK** in the Add a Program dialog box. The files you selected are added to the Programs and Services list in the Windows Firewall dialog box.
7. Click **OK** to close the Windows Firewall dialog box.

For details about Windows firewall exceptions, see your documentation or contact your system administrator.

## How to View or Modify Remote Agent Settings

### Relevant for: GUI tests and components and API testing

This task describes how to view or modify the settings in the Remote Agent Settings dialog box.

1. Select **Start > All Programs > HP Software > HP Unified Functional Testing > Tools > Remote Agent** or <UFT installation folder>\bin\UFRemoteAgent.exe.

The Remote Agent opens and the **Remote Agent** icon  is displayed in the task bar tray.

2. Right-click the **Remote Agent** icon and select **Settings**. The Remote Agent Settings Dialog Box dialog box opens.
3. View or modify the settings in the dialog box.
4. Click **OK** to save your settings and close the dialog box.
5. Right-click the **Remote Agent** icon and select **Exit** to end the Remote Agent session.

# How to Install a Certificate for ALM Servers in a Environment Using External Authentication

## **Relevant for: GUI tests and components, API testing and business process testing**

This task describes how to install a certificate to use an ALM server running in a CAC (Common Access Card) environment.

1. Request the following from your certificate authority:
  - The certificate authority certificate in PEM format. Rename to **TrustedCA.pem**.
  - The web server certificate in PEM format. The full server name should appear in the certificate. Rename to **WebServerPublicCert.pem**.
  - The server certificate private key file in PEM format. Rename to **WebServerPrivateCert.pem**.
  - The software client certificate (if a Common Access Card is not used) for one user.
2. Place the certificate files in your web server configuration directory.

If you receive certificates in different formats, you can use **openssl** to convert them. To install **openssl**, go to <http://www.openssl.org/related/binaries.html>.

- To convert from CER, use **openssl x509 -in /<webserver-directory>/conf/cert.cer -outform pem -out cert.pem**.
- To convert from PFX, do the following:
  - Export the public key by using **openssl pkcs12 -in /<webserver-directory>/conf/cert.pfx -clcerts -nokeys -out certPublic.pem**.
  - Export the private key by using **openssl pkcs12 -in /<webserver-directory>/conf/cert.pfx -nocerts -nodes -out certPrivate.pem**.

# Troubleshooting and Limitations - General ALM Integration

## **Relevant for: GUI tests and components and API testing**

- The first time you connect to your ALM server, (either within UFT or through a browser) **you must connect as an administrator**. This allows the machine to properly install the ALM client with the required connectivity. For all subsequent connections, you do not need to log in as administrator. This step is also required after installing ALM patches.

- If a test is stored in ALM, and was last modified using a version of QuickTest earlier than 9.5, it opens in read-only mode.

To edit the test, it must be upgraded to QuickTest 11.00 using the QuickTest Asset Upgrade Tool for ALM (found on the QuickTest 11.00 installation DVD).

- Testing documents that are part of the same solution cannot be stored in different ALM projects, domains, or servers.
- Before you run tests remotely from ALM, you must perform the following prerequisites:
  - a. You must enable COM+ access. For details, see the [Installation Guide](#).
  - b. Change DCOM permissions and open firewall ports. For details, see the [Installation Guide](#).
  - c. Run RmtAgentFix.exe from the <Unified Functional Testing installation>\bin folder, or use the Additional Requirements Utility, which you open by selecting **Start > Programs > HP Software > HP Unified Functional Testing > Tools > Additional Installation Requirements**.

This is due to a problem with opening DCOM permissions on Windows 7 and Windows Server 2008 R2.

- d. Disable User Account Control (UAC) in Windows and restart your computer before you first connect to ALM. For details, see "[Troubleshooting and Limitations - UFT Program Management](#)" on page 1138.
- The security settings in Windows 7, Windows Server 2008 R2, and Windows 2012 may prevent you from performing a UFT related installation, such as a patch installation, or connecting to an ALM project (either directly or from UFT). This can occur when the UAC (User Account Control) option is set to ON, and you have not yet connected to an ALM project (if relevant).

**Workaround:** Temporarily turn off the UAC option. For details, see "[Troubleshooting and Limitations - UFT Program Management](#)" on page 1138.

After disabling the UAC option as described above, perform the required installation or connect to ALM as usual. When you are finished, you can turn the User Account Control (UAC) option on again. Hereafter, you should be able to connect to ALM, as needed.

- In Windows 8, when trying connecting to ALM in UFT with the UAC turned off, you cannot connect to ALM.

**Workaround:** Run UFT as an administrator (even if you are the admin user).

- When running a test in UFT from ALM on Windows 8 or higher, you must connect to the ALM server running Internet Explorer with Administrator permissions and with the URL format  
`http://myserver:8080/qcbin/start_a.jsp?common=true`
- If you are connected to an ALM server, and you want to connect to a different server, disconnect from the first ALM server, restart UFT, and connect to the second server.
- Connecting to ALM via UFT on Windows 10 without administrative privileges is not supported.

**Workaround:** Connect to ALM with Administrator permissions immediately after performing your UFT installation.

- If UFT closes unexpectedly while an asset is open from ALM, the asset may remain locked by ALM for more than fifteen minutes. In some cases, you may be able to reopen UFT and reopen the test, but when trying to save it, you will receive an error message indicating that the test entity is already locked by you.

**Workaround:** Wait fifteen minutes or more and try again.

- Renaming a test or component from ALM may cause the test or component to behave unexpectedly.

**Workaround:** To rename a test or component, open it in UFT and rename it using the **Save As** option. If the test or component has already been renamed from ALM, use the rename option again to restore the old name, and then use the **Save As** option in UFT. Renaming a test parameter from UFT causes any run-time parameter values already set for this parameter in ALM to be lost.

- If an ALM user manually changes the status of a test instance run, ALM creates **fast run** results to record the change of the test status. The **fast run** results are not valid run results files. However, when you try to select results to open or delete in the Run Results Viewer or Run Results Deletion tool, the **fast run** results are available in the list.

- For tests or business components saved on ALM, if you perform a checkout from within ALM, it is not reflected in UFT for the current test—the test or business component still appears to be checked in.

**Workaround:** Close and reopen the test in UFT.

- When running a test from UFT with ALM 11.50 and later with an ALM project that supports versioning, the test instance run status will not be updated for tests that are checked out.

- ALM supports non-Unicode projects. If you are working with an ALM project that is not Unicode compliant:
  - You should not use Unicode values (such as the name of the test or component, the name of an application area, the default value of a test, action, or component parameter, method argument values, and so forth).

- Data that is sent to UFT from ALM (such as values for test, action, or component parameters) is not Unicode compliant.

- UFT results containing Unicode characters may appear corrupted in the ALM result grid. You can, however, open and view results containing Unicode characters in the UFTRun Results Viewer.

For additional details on UFT Unicode issues, see "["Troubleshooting and Limitations - Multilingual Applications in GUI Testing" on page 1141](#)".

- If there is a forward proxy with Basic Authentication between the server and client machines, before the first connection to an ALM platform, each ALM client must configure the proxy credentials by using the **Webgate Customization Tool**. If you do not run WebGate, you may be unable to connect, or you may need to enter your credentials multiple times.

**To run the tool:**

- Go the following location on the ALM client machine: `http:\\<ALM Platform server name>[:port number]/qcbin/Apps/`
- Click on the **Webgate Customization Tool** link and select **Run**.

- c. In **WebGate Customization**, click in the **Proxy Credentials** area. Select the **Use these credentials** check box, and type values in the **Proxy Username** and **Proxy Password** boxes.
  - d. Click **Save** and then **Close**.
- In order to run a UFT test or open a test, component, or application area stored in the **Test Plan** on an ALM server using external authentication, you must select the **Non-interactive mode** option in the Webgate Customization tool and select the external certificate in the same dialog.

**Note:** Make sure that the certificate authority that issued your client certificate is added to the Trusted Root Certification Authorities for the current user. For further details, see the documentation for the Webgate Customization tool for your ALM server.

For details about setting up certificates and the Webgate Customization tool, see the documentation for the Webgate Customization tool included with your ALM server.

- If a test running from an ALM server using external authentication fails with the message **Unable to connect to the testing tool**, make sure that your external certificate is selected in the Webgate Customization tool and that the certificate authority that issued the certificate is present in the Trusted Root Certification Authorities for the current ALM user.
- When running a test set from the Test Lab module, if you stop a test set run in the middle of the run, and then immediately start another run, UFT behaves unexpectedly.

**Workaround:** Wait after stopping the test set run.

## Troubleshooting and Limitations - ALM Integration with GUI Testing

### Relevant for: GUI tests and components

- If the parameter names in your **Global** data table are identical to the parameter names in an external data table, running a test configuration with data table parameter mapping to other data table parameters causes unexpected results. For example, if your **Global** data table parameters are Login and Password, and your external data table parameters are Login, Password, Login\_localized, and Password\_localized, a test configuration run with mapping from Login to Login\_localized and Password to Password\_localized will cause unexpected results.

**Workaround:** Split the external data table into multiple tables.

- To run a test or component from ALM, you must first invoke UFT at least once. Otherwise, ALM may be unable to open UFT.

# Troubleshooting and Limitations - ALM Integration with API Testing

## **Relevant for: API testing**

- While connected to an ALM project, if you change the available license for an API test, ALM continues to use the original license.

**Workaround:** Log out from your ALM project and restart Internet Explorer.

- When running API tests from ALM, using test parameters linked (in ALM) to data tables stored in ALM is not supported.

**Workaround:** Instead of associating test steps with test parameters, link the steps to an Excel data source added to the test.

When you add the data source to the test, select **Allow other tools to override the data**. Then, in the ALM Test Plan, instruct ALM to overwrite this Excel with data tables stored in ALM. For details on instructing ALM to overwrite data sources, see the ALM documentation.

- You can create and manage API components in much the same way as you would API tests, as described in "[API Testing Design with the UFT IDE](#)" on page 388. For details about exceptions that apply when working with API components, see "[Troubleshooting and Limitations - Business Process Testing in UFT](#)" on page 1001.

# Chapter 61: Resources and Dependencies Model

**Relevant for: GUI tests and components and API testing**

**Note:** The references to ALM in this chapter apply to Quality Center and ALM. Note that some features and options may not be supported in the Quality Center or ALM edition you are using. For information on Quality Center or ALM editions, see the *HP Quality Center User Guide* or the *HP Application Lifecycle Management User Guide*.

This chapter includes:

- [Resources and Dependencies Model Overview](#) ..... 943
- [Resources and Dependencies Model Terminology](#) ..... 943
- [Asset Dependencies - Advantages](#) ..... 945
- [Relative Paths for Tests and Resources Saved in ALM](#) ..... 947
- [Updating Relative Paths for Tests and Resources Saved in ALM](#) ..... 948
- [Troubleshooting and Limitations - Resources and Dependencies](#) ..... 950

# Resources and Dependencies Model Overview

## Relevant for: GUI tests and components and API testing

UFT enables you to use the Resources and Dependencies model to fully integrate your tests and components into ALM projects.

**Note:** Before you read this section, you may want to familiarize yourself with the "[Resources and Dependencies Model Terminology](#)" (described on page [943](#)).

- **Replaces the use of attachments with linked assets.** For example, **GUI tests, actions, and application areas** can be linked with function libraries and shared object repositories, respectively or **API tests** can be linked with data tables, user code files, or activities. You store your tests or components in the Test Plan or Business Components module, respectively, and you store your resource files (including application areas) in the Test Resources module. When you associate a resource file to a test or a GUI component's application area, these assets become linked. Linking assets improves runtime performance by decreasing download time. (Using attachments instead of resources increases download time from Quality Center and ALM.) Linking assets also helps to ensure that the relationships between dependent assets are maintained.

**Note:** To ensure that your resource files are recognized as dependencies, they must be saved in the **Test Resources** module in ALM, and they must be associated using the full ALM path.

- **Supports versioning for tests or components and resource files.** You can create versions of these assets in UFT or in ALM, and you can manage asset versions in ALM. For details, see "[Version Control in ALM](#)" on page [951](#).
- **Supports baselines for tests or components and resource files.** You can view baseline history in UFT, and you can view and manage baselines in ALM.
- **Enables you to view and compare your assets in both ALM and UFT.** You can use the Asset Comparison Tool to compare versions of individual assets and the Asset Viewer for viewing an earlier version of a asset. Both of these viewers are available in ALM and in UFT.
- **Enables you to import and share assets across ALM projects.** For details, see the *HP Application Lifecycle Management User Guide*.

For details about the advantages of working with this model, see "[Asset Dependencies - Advantages](#)" on page [945](#).

## Resources and Dependencies Model Terminology

## Relevant for: GUI tests and components and API testing

Term	Description
Asset	Any testing document or resource file, including:

Term	Description
	<p><b>For GUI testing:</b></p> <ul style="list-style-type: none"> <li>• tests</li> <li>• actions (GUI testing only)</li> <li>• components</li> <li>• application areas (GUI testing only)</li> <li>• function libraries (GUI testing only)</li> <li>• shared object repositories (GUI testing only)</li> <li>• recovery scenarios (GUI testing only)</li> <li>• data table files</li> <li>• environment variable files</li> </ul> <p><b>For API testing:</b></p> <ul style="list-style-type: none"> <li>• tests</li> <li>• components</li> <li>• data table files</li> <li>• XML files</li> <li>• testing activities</li> <li>• user code files</li> </ul> <p><b>Note:</b> In ALM, UFT assets are referred to by the more general term <b>entities</b>.</p>
<b>Resource</b>	<p>Any asset stored in the ALM Test Resources module that is used by a test or component. For example, a <b>GUI test or component</b> may contain calls to functions in associated function libraries, and it may reference test objects stored in shared object repositories that are associated with the test or component's application area. An <b>API test or component</b> may contain references to a data table file or use a custom testing activity.</p> <p>GUI testing resources include:</p> <ul style="list-style-type: none"> <li>• application areas</li> <li>• function libraries</li> <li>• shared object repositories</li> <li>• recovery scenarios</li> <li>• data table files</li> <li>• environment variable files</li> </ul> <p>API testing resources include:</p> <ul style="list-style-type: none"> <li>• data table files</li> <li>• XML files</li> <li>• testing activities</li> <li>• Web references (done via Service Test Management)</li> <li>• user code files</li> </ul> <p><b>Note:</b> In some cases, a resource can be used by another resource. For example, a GUI test's recovery scenario can use functions in a function library.</p>

Term	Description
<b>Dependency</b>	<p>The linked relationship between a resource or external GUI action and a particular test or application area. Associated resource files and GUI actions are linked to each test or GUI component's application area that uses these resources or calls these GUI actions.</p> <p>In some cases, a resource can also be linked to another resource. For example, a GUI test's recovery scenario can call functions in a function library.</p> <p>Assets are considered dependencies if they are associated using absolute paths, and they are stored in the following modules:</p> <ul style="list-style-type: none"> <li>• <b>Test Plan module:</b> tests</li> <li>• <b>Business Components module:</b> components</li> <li>• <b>Test Resources module:</b> <ul style="list-style-type: none"> <li>• GUI testing resources like application areas, function libraries, shared object repositories, recovery scenarios, data table files, environment variable files</li> <li>• API testing resources like data table files, .xml files, testing activities, and user code files</li> </ul> </li> </ul> <p><b>Note:</b> Tests or components stored in the <b>Unattached or Obsolete</b> folders in the Test Plan or Business Components module, respectively, are not considered dependencies because they are not associated with any test or component.</p>
<b>Configuration</b>	<p>A test that is set to run from ALM with an optional data resource file and optional data filter settings. In ALM, you can define various configurations for the same test or business process test.</p> <p>For tests, see "<a href="#">Data Awareness in ALM</a>" on page 928</p> <p>For components, see the <i>HP Business Process Testing User Guide</i>.</p>

## Asset Dependencies - Advantages

### Relevant for: GUI tests and components and API testing

When you associate a **dependent** resource file with a test, component, or a GUI component's application area, the assets become integrally linked, and these links can be viewed in ALM (in the Dependencies tab of various modules) and—for external GUI actions—in UFT (in the Action Properties dialog box).

There are a number of advantages to creating asset dependencies:

### Assets stay linked

When you move a test, or rename a test, action, component, application area, folder, or resource, dependent assets are automatically updated to reflect the change. This helps ensure that there are no missing resources during a run session.

### Resource files are all stored in one ALM module

Resource files are stored in the Test Resources module, enabling you to manage your resources in one central location, and to view at a glance which tests and application areas are using each resource file.

For details on the Test Resources module, see the *HP Application Lifecycle Management User Guide*.

## Using resources stored in the Test Resources module improves runtime performance

Tests or components open and run faster when the associated resource files are stored in the Test Resources module (instead of being stored as attachments to tests in the Test Plan module).

## Version control can also be applied to resource files

When version control is enabled for a project, all of the assets can be checked into the version control database. For details, see "[Version Control in ALM](#)" on page 951.

## You can create, view, compare, and run baselines

In ALM, you can create baselines that capture the developmental stage for each asset, view and compare these read-only baseline "snapshots", and run baselines from a project. In UFT, you can view and compare baselines.

## You can share assets with other projects and synchronize them

You can copy assets from other projects. This enables you to reuse your existing assets instead of creating new assets whenever you create a new project. For example, you can create a "template" set of assets to use as a basis for new projects.

You can synchronize these assets in both projects when changes are made, or you can customize your assets to suit the unique needs of each development project. For details, see the sections on importing and synchronizing libraries in the *HP Application Lifecycle Management User Guide*.

## Deleting assets is easier

When you delete an asset (a reusable GUI action or component or associated resource file), a warning message informs you if the asset is used by other tests (or more than once in the current test) or is associated with an application area. This message contains a **Details** section that lists the tests or application areas that are associated with this asset or contain calls to this action so you can modify the tests or application areas, as needed. This helps you manage your business process tests and GUI action calls so that tests do not inadvertently fail.

## You can verify which tests or components are associated with specific resources and vice versa

In the ALM Test Plan module and Business Components module, you can highlight a test or component and, in the Dependencies tab, see which assets are using the test or component, and see which assets the test or component is using. Similarly, in the ALM Test Resources module, you can highlight a resource file and see the assets with which it is associated.

### You can verify which tests contain calls to an action (GUI testing only)

You can view a list of the tests that contain calls to a particular action by focusing on the action and opening the Used By tab in the Action Properties dialog box. (Right-click an action in the canvas and select **Action Properties**.)

## Relative Paths for Tests and Resources Saved in ALM

When you save a test and its resources in ALM, ALM must create a path to the necessary test resources to ensure that they are loaded when you run the test. In UFT, the path to your test's resources is defined either as a relative or an absolute path. A relative path provides the location of the resources relative to the test with which it is associated. An absolute path lists the full path in the ALM project, regardless of its association with your test or other tests.

You can associate resources either from the Test Plan module in ALM or the Test Resources module in ALM. UFT deals with each module differently:

- If you associate resources stored in the **Test Plan** module, UFT prompts you to associate the resource using a relative path. You can choose to change the path to a relative path or maintain the full absolute path.
- If you associate a resource stored in the **Test Resources** module, the absolute path is used.

As a result, you have the choice to maintain relative paths or use absolute paths depending on your needs.

### Relative vs Absolute Paths - Advantages and Disadvantages

Depending on whether you choose to use a relative path or an absolute path for your test's associated resources, there are a number of advantages and disadvantages:

Type of Path	Advantages	Disadvantages
<b>Absolute</b>	<ul style="list-style-type: none"><li>• Run-time performance times for loading and accessing associated resources are faster.</li><li>• You can use the <a href="#">Resources and Dependencies model</a> for your associated test resources.</li></ul>	You cannot use localized resources with the same test, as you need to specify a full path to each resource to use.
<b>Relative</b>	<p>You can change the content of a particular resource without having to associate an addition resource with the test.</p> <p>For example, you can change the objects in a function library to be the objects from a particular</p>	<ul style="list-style-type: none"><li>• Run-time performance times for loading and accessing associated resources are slower.</li><li>• Dependency information for these assets is not displayed in:</li></ul>

	<p>language of your application without having to add an additional object repository to the test.</p> <ul style="list-style-type: none"> <li>The Using and Used By grids in the Dependencies tab in ALM</li> <li>The message box that opens when you delete an asset that is associated with other assets</li> <li><b>For GUI tests only:</b> The Used By tab of the Action Properties dialog box in UFT</li> <li>ALM does not verify that these assets are included during the baseline verification process.</li> <li>When opening or running tests or components from a baseline, any associated external GUI action or resource file that is associated via a relative path but is <b>not</b> included in the baseline is considered a missing resource. This may cause a test or component run to fail. (Note that the baseline version of an associated asset is used if the asset associated via a relative path <b>is</b> included in the baseline.)</li> <li>When using the Asset Comparison Tool to view a test or component, you cannot drill down to view assets that are associated via a relative path.</li> </ul>
--	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## Updating Relative Paths for Tests and Resources Saved in ALM

### **Relevant for: GUI tests and components and API testing**

If your test's associated resources are saved with a relative path, you have the option to update these relative paths to an absolute path. This conversion improves the test run performance by reducing the amount of time necessary for UFT to find the associated resources. It is recommended that you perform this conversion unless you have a specific reason for maintaining a relative path.

### **In order to convert your test's associated resources from a relative path to an absolute (full path), you can do the following:**

1. If your test resources are associated with a relative path, UFT displays a warning message in the Errors pane.
2. If you double click on this warning message, UFT opens another dialog which enables you to convert the relative paths to the full absolute ALM path for resources associated with the selected test.
3. In the dialog, select the tests and resources for which you want to convert the path, and UFT performs the conversion automatically.

If you do not want to convert the associated test resource's relative paths, you can do one of the following:

- In the warning message, select the option for UFT to stop showing the warning message.
- Clear the **Provide warning to replace relative paths for resources associated with test saved in ALM** option in the **Folders** pane of the Options dialog box (**Tools > Options > GUI Testing tab > Folders** node).

**Note:** Converting the test's association to the ALM resources associated with a relative path is

irreversible. Therefore, it is strongly recommended to back up your tests before performing the conversion.

If you want to later re-enable UFT to convert the associated resources relative paths, you can enable the option in the **Folders** pane of the Options dialog box or enable the warning per test in the **Properties** pane of the Test Settings dialog box (**File > Settings > Properties** node).

For an overview of relative paths for tests and resources saved in ALM, see "[Relative Paths for Tests and Resources Saved in ALM](#)" on page 947.

# Troubleshooting and Limitations - Resources and Dependencies

## **Relevant for: GUI tests and components and API testing**

This section describes troubleshooting and limitations for resources and dependencies.

- When you save a resource to ALM (either from UFT or using the **Upload** option from the ALM Test Resources module), and the resource file has a comma in the file name, the resource appears to be saved successfully, but the file is not actually uploaded to the ALM server.
- **For GUI testing:** If you insert a call to an external action that is associated with a data table, and that data table was previously renamed or moved in the Test Resources module of Quality Center or ALM, UFT tries to locate the data table in its original location.

**Workaround:** Save the test, close it, and reopen it.

- If you are working with the Resources and Dependencies model, and the test containing the action you are renaming is stored the Test Plan module in ALM, the internal (default) action name is always displayed in the Used By Tab in the Action Properties Dialog Box). This is true even if you rename the action.

# Chapter 62: Version Control in ALM

**Relevant for: GUI tests and components, API testing, and business process tests and flows**

**Note:** The references to ALM in this chapter apply to Quality Center and ALM. Note that some features and options may not be supported in the Quality Center or ALM edition you are using. For information on Quality Center or ALM editions, see the *HP Quality Center User Guide* or the *HP Application Lifecycle Management User Guide*.

This chapter includes:

- [Managing Versions of Assets in ALM](#) ..... 952
- [How ALM Manages Assets](#) ..... 953
- [Version History Versus Baseline History](#) ..... 955
- [Asset Comparison Tool and Asset Viewer](#) ..... 955
- [How to Use ALM Version Control](#) ..... 958
- [How to Work with the Asset Comparison Tool and Asset Viewer](#) ..... 960
- [Troubleshooting and Limitations - ALM Version Control](#) ..... 964
- [Troubleshooting and Limitations - Asset Comparison Tool](#) ..... 965

# Managing Versions of Assets in ALM

## Relevant for: GUI tests and components, API testing, and business process tests and flows

When UFT is connected to an ALM project with version control support, you can update and revise your UFT assets while maintaining earlier versions of each asset. This helps you keep track of the changes made to each asset and see what was modified from one version to another. Assets can include tests, components, function libraries, application areas, shared object repositories, recovery scenarios, and external data tables, XML files, user code files, or test activities.

You can check in the asset at any time. For example, you may want to check the asset in every day or when you complete a task. While the asset is checked out to you, other users can view the last checked in version of that asset in read-only mode, but they cannot modify the asset or view your changes until you check in the asset.

If the asset is...	You can...
checked in	<ul style="list-style-type: none"><li>Open the asset in read-only mode using the <b>Open</b> option. You cannot modify the asset.</li><li>Open the asset and check it out by selecting <b>ALM &gt; Check Out</b>.</li></ul> <p><b>Note:</b> If you check out a test, and that test is associated with a resource that is currently checked in to the ALM project, the resource is read-only, even though the test is editable. To modify the resource, you must first check out the resource in the ALM Test Resources module.</p>
checked out to your ALM user name	Open the asset in read-write mode, using the <b>Open</b> option and modify the asset as needed. <p><b>Tip:</b> If a test, component, function library, or application area is checked out to your ALM user name, an unlocked icon  in the Solution Explorer indicates this status.</p>
checked out to another ALM user	Open the asset in read-only mode using the <b>Open</b> option. UFT displays a message indicating that the asset is checked out to another ALM user. You can view the last checked in version of the asset now, and you can check out the asset later after the other user checks in the asset. <p><b>Tip:</b> If a test, component, function library, or application area is checked out to another ALM user name, the document tab indicates this status by displaying a locked icon  and <b>(Read-Only)</b> adjacent to the document's name.</p>

In UFT, you can check out only the latest version of an asset, although you can view and compare earlier versions. This is because assets that are stored in ALM are often linked to or **dependent on** one another.

For example, if you try to run an earlier version of a test or component with a later version of a shared object repository or a data table, your test or component might fail because the objects in the object repository or data table would not necessarily match the objects or steps in the test or component. For more details, see "[Troubleshooting and Limitations - ALM Version Control](#)" on page 964.

# How ALM Manages Assets

## **Relevant for: GUI tests and components, API testing, and business process tests and flows**

You manage asset versions by checking assets in and out of the version control database.

You can do the following using the version control database:

### **Add assets**

When you create a new asset or use **Save As** to save an existing asset in an ALM project with version control support, UFT automatically saves the asset in the project, checks the asset into the version control database with version number 1, and then checks it out so that you can continue working. This is an administrative version of the asset, similar to a placeholder. The version number indicates that the asset exists in the database. When you later check in the asset, the version number remains version number 1—the first version that you are checking in. Subsequent checkins increase the version number by 1.

Saving your changes to an existing asset does not check them in. Even if you save and close the asset, the asset remains checked out until you choose to check it in. For details, see "[Check assets in](#)" on the next page below.

You add an asset to the version control database by saving it in an ALM project with version control support. When you save an asset for the first time, UFT automatically checks the asset into the ALM version control database, assigns it version number 1, and automatically checks the asset out for you so that you can continue working on it. When you check the asset in, the asset retains version number 1, since this is the first version that can contain content. Then, each time the asset is checked out and in again, the version number increases by 1.

### **Check assets out**

When you open an asset that is currently checked in to the version control database, it is opened in read-only mode. You can review the checked-in asset. You can also run the asset and view the results.

To modify the asset, you must check it out. When you check out an asset, ALM copies the asset to your unique check-out folder (automatically created the first time you check out an asset), and locks the asset in the project database. This prevents other users of the ALM project from overwriting any changes you make to the asset. However, other users can still run the version that was last checked in to the database.

You can save and close the asset, but it remains locked until you return the asset to the ALM database. To release the asset, either check the asset in, or undo the check out operation. For more details on checking assets in, see "[Check assets in](#)" on the next page below. For details on undoing the check-out, see "[Cancel a check-out operation](#)" on page 959.

In UFT, the check out option accesses the latest version of the asset. In ALM, you can also check out earlier versions of any asset except for application areas.

## Check assets in

While an asset is checked out, ALM users can run the previously checked-in version of your asset. For example, suppose you check out version 3 of an asset and make a number of changes to it and save the asset. Until you check the asset back into the version control database as version 4, ALM users can continue to run version 3.

When you have finished making changes to an asset and you are ready for ALM users to use your new version, you check it in to the version control database.

**Note:** If you do not want to check your changes into the ALM database, you can undo the check-out operation.

When you check an asset back into the version control database, ALM deletes the asset copy from your checkout folder and unlocks the asset in the database so that the asset version is available to other users of the ALM project.

## View version control information when opening a test

When you open a test from an ALM project with version control support, you can view version control information for the test by using the **Details** view in the Open dialog box.

The **Checked Out To** column specifies the user name of the ALM user to whom the test is checked out, if it is checked out. If the test is currently checked in to the version control database, there is no indication in the dialog box.

## View and compare asset versions

You can view and compare the versions of an asset using the Asset Comparison Tool. For details, see "[How to Work with the Asset Comparison Tool and Asset Viewer](#)" on page 960.

If your project administrator creates project baseline versions when a milestone is reached during product development, you can view and compare the asset versions stored in these baselines. For details, see "[View baseline history](#)" below below.

**Note:** With the exception of the **Baseline History** option, the **ALM Version Control** options in the **ALM** menu are available only when you are connected to an ALM project with version control support, and an asset stored in ALM is open in the UFT window.

## View baseline history

In ALM, a project administrator can create baselines that provide "snapshots" of an entire project (or part of a project) at different stages of development. A **baseline** represents a version of a project at a specific point in a project's life cycle. For example, baselines are often created for each milestone or when specific phases in a project are completed.

Baselines can be created for ALM projects that are enabled for version control, and for projects for which version control is not enabled.

The project administrator creates the baseline in the Libraries tab of the Management module in ALM. Creating a baseline is a two-fold process. The administrator first creates a library, which specifies the root folders from which to import the data. The administrator makes sure to include all of the associated resource files, such as shared object repositories and function libraries. The administrator then creates the actual baseline, which comprises the latest versions of every asset included in the library. If the project is version control-enabled, then these are the latest checked in versions of every asset.

During the creation process, ALM verifies that all of these assets (such as associated resource files) are included in the baseline. If any assets are not included, ALM informs the administrator so that the library and baseline can be modified accordingly. For more details, see the ALM user guide.

In ALM, these baselines can be viewed and compared in their entirety.

In UFT, you can view and compare the assets saved in these baselines. This enables you to review the content of an asset at a specific phase in the project time line.

You can also run a test or component from a baseline.

## Version History Versus Baseline History

### **Relevant for: GUI tests and components and API testing**

This section focuses on the differences between version history and baseline history and describes when to use each.

- You use version control to check in and check out assets as needed. For example, you may want to check in an asset on a daily basis or only when significant results are achieved. This enables you to monitor the asset's development.

If you want to view the content of an asset on a particular date or after a particular user checked in the asset, use the **Version History** option to view or compare the asset.

- The ALM project administrator creates baselines that represent "snapshots" of a project's assets at various milestones in a project's life cycle. Each baseline links to the assets specified by the administrator when the baseline was created. The asset version represented in the baseline is always the version that was checked in when the baseline was created.

If you want to view an asset as it was saved for a particular milestone, use the **Baseline History** option.

## Asset Comparison Tool and Asset Viewer

### **Relevant for: GUI tests and components and API testing**

An **asset** is a UFT testing document (such as a test, component, or application area) or any resource file that is used by a UFT testing document (such as a function library, a shared object repository, a data

table, a recovery scenario, or an environment variable XML file). The Asset Comparison Tool and Asset Viewer enable you to view and compare versions of a particular asset.

Using these tools, you can:

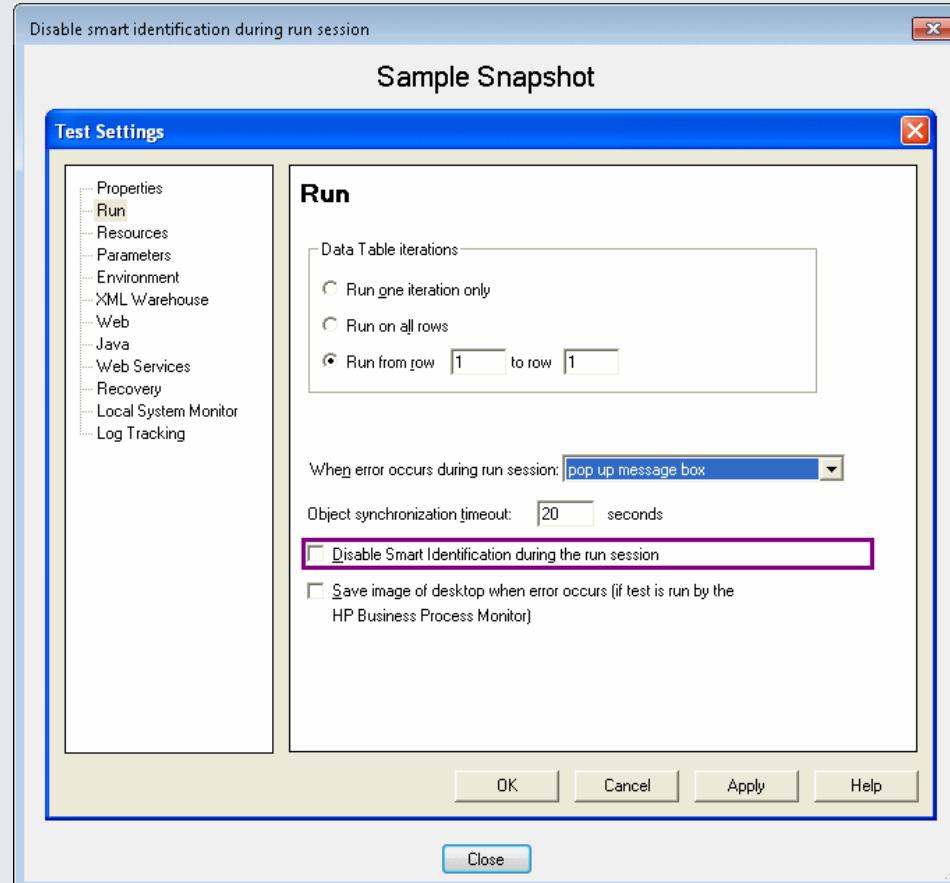
<b>View any version of an asset using the Asset Viewer</b>	After you select a specific version in the Version History dialog box, you can see specific details about that version of the asset, even if it is different from the working copy of the asset. Using this feature enables you to see where specific settings and details of the asset have been changed.
<b>Compare two versions of an asset using the Asset Comparison Tool</b>	After you have selected the specific versions to view in the Version History dialog box, you can view differences between the two versions.
<b>Drill down to view or compare versions</b>	<ul style="list-style-type: none"> <li>View or compare versions of an <b>integral element</b>. An integral element is a resource file that is a part of the test or component (not saved as an external resource), such as a local object repository (for GUI testing) or a data table (for API testing). When you check in a test or component, these elements are checked in, too, because they are part of the test or component. Therefore, when you drill down in the asset, you can view or compare the version that existed when the test or component was checked in, in addition to the currently saved version.</li> <li>View or compare versions of <b>associated external assets</b>. An associated asset is any external (not integral) resource file used by an asset (such as a function library, a shared object repository, a data table, a recovery scenario, or an environment variable XML file).</li> </ul> <p><b>Note:</b></p> <ul style="list-style-type: none"> <li>When you drill down while viewing or comparing an asset, the currently checked in content of the associated assets is displayed, even if you are viewing or comparing an earlier version of the main asset. To view or compare an earlier version of the drilled-down associated asset, open the resource file itself and use the Asset Viewer or Asset Comparison Tool.</li> <li>To view or compare by drilling down, the resource file must be associated via an absolute or ALM path. For details, see "<a href="#">"Updating Relative Paths for Tests and Resources Saved in ALM" on page 948</a>.</li> </ul>
<b>View a screen capture depicting the UFT location of an element (GUI testing)</b>	The screen capture displays an example of the relevant dialog box. The option (or area) for the node you right-clicked is highlighted in the screen capture.

### Examples

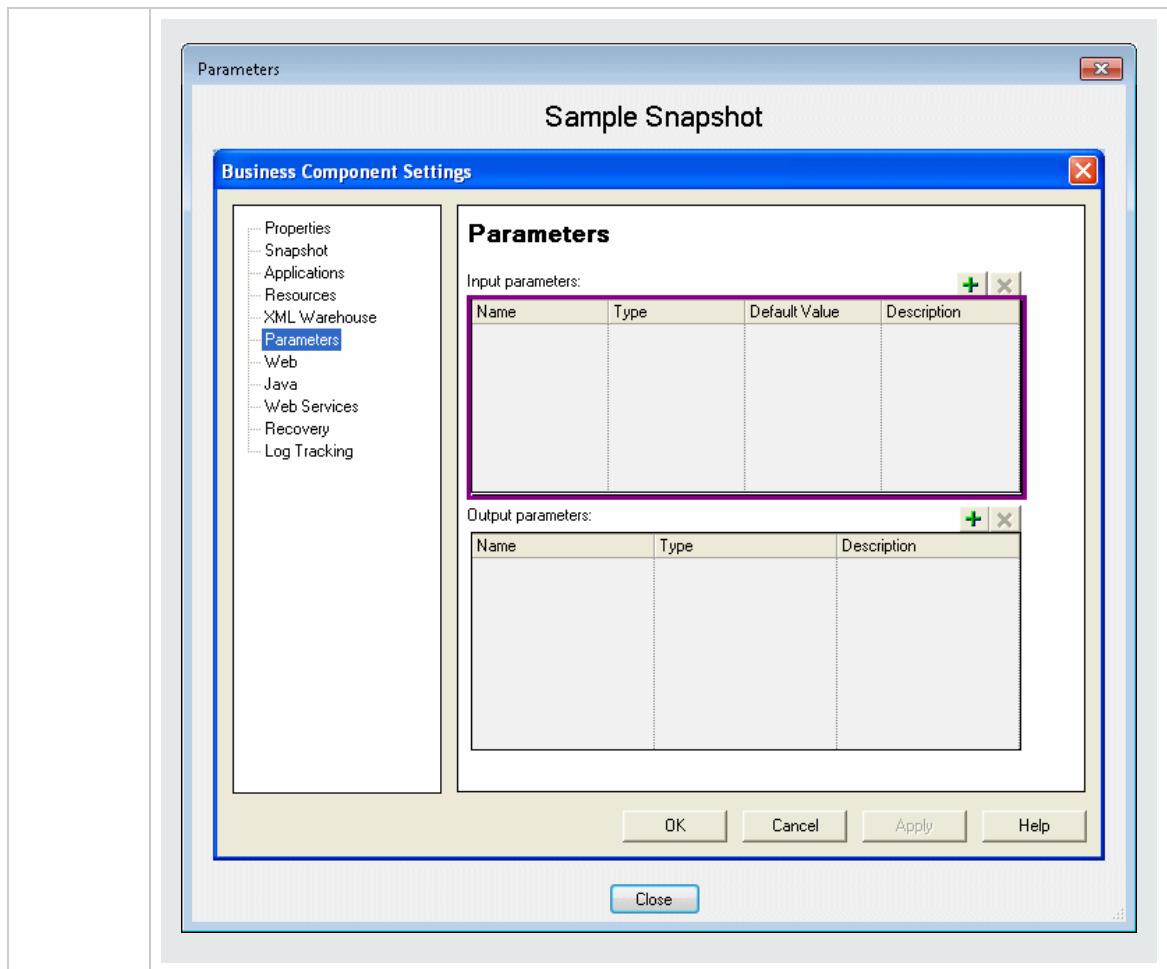
**For tests:** Suppose you are viewing a comparison of a test and you notice that the `Disable Smart Identification` during the `run session` node is highlighted, indicating that it was modified. If you are not sure where this option is located in UFT, you can right-click the node in the

only)

comparison tree and select **View Sample Snapshot**. UFT then opens a dialog box showing you that this area is located in the Run pane of the Test Settings dialog box. The title bar of the dialog box lists the selected element, and a highlighted rectangle outlines the option.



**For components:** Suppose you are viewing a comparison of a component and you notice that the Input parameters node is highlighted, indicating that it was modified. If you are not sure of where this option is located in UFT, you can right-click the node in the comparison tree and select **View Sample Snapshot**. UFT then opens a dialog box showing you that this area is located in the Parameters pane of the Business Component Settings dialog box. The title bar of the dialog box lists the selected element, and a highlighted rectangle outlines the option.



For details, see "[How to Work with the Asset Comparison Tool and Asset Viewer](#)" on page 960.

To see a demonstration of this functionality, go to the [HP Live Network \(HPLN\) page](#) and select the **UFT-ALM Integration** movie from the list.

## How to Use ALM Version Control

### **Relevant for: GUI tests and components, API testing, and business process tests and flows**

This task describes how to check in the currently open asset, check out the latest version of an asset in order to edit it, and cancel a check-out operation if needed.

After you edit an asset, you can save changes and close it without checking in the modified asset. However, your changes are not available to other ALM users until you check it in. If you do not want to check your changes in, you can undo the check-out by canceling it.

For more details on checking assets in, see "[Check assets in](#)" on page 954.

This task includes the following steps:

- "[Check in the currently open asset](#)" below
- "[Check out the latest version of an asset](#)" below
- "[Cancel a check-out operation](#)" below
- "[View the version history](#)" on the next page

## Check in the currently open asset

1. Confirm that the currently open asset is checked out to you.

**Note:** If the open asset is currently checked in, the **Check In** option is disabled. If you open an asset that is checked out to another user, all **ALM Version Control** options, except the **Version History** option, are disabled.

2. Select **ALM > Check In**, and check in the asset using the Check In dialog box.

## Check out the latest version of an asset

1. Make sure the asset you want to check out is currently checked in. If you open an asset that is checked out to you, the **Check Out** option is disabled. If you open an asset that is checked out to another user, all ALM version control options, except the **Version History** option, are disabled.
2. Open the resource, as follows:

If the asset is a:	Do this:
<b>Test, Component or Function Library</b>	In the main UFT window, select <b>File &gt; Open</b> and select <b>Test, Business Component, Function Library</b> , or <b>Application Area</b> or click the <b>Open</b> down arrow and select the asset type from the list.
<b>Shared Object Repository (GUI testing only)</b>	In the Object Repository Manager, select <b>File &gt; Open</b> or click the <b>Open</b> button.
<b>Recovery Scenario (GUI testing only)</b>	In the Recovery Scenario Manager, click the <b>Open</b> button.

3. Browse to and open the asset.

The document opens in the document pane as read-only with a lock icon in the document's tab.

4. Select **ALM > Check Out** to check out the document and edit it.

## Cancel a check-out operation

1. Open the asset if it is not already open.
2. Select **ALM > Undo Check out**.
3. Click **Yes** to confirm the cancellation of your check out operation.

The check out operation is cancelled. The checked out asset closes, and the previously checked in version reopens in read-only mode.

## View the version history

1. In the document pane or in the Solution Explorer, select the test or resource file for which you want to view the history.
2. Select **ALM > Version History**.

The Version History dialog box opens, displaying the information on the versions of the selected asset.

# How to Work with the Asset Comparison Tool and Asset Viewer

## Relevant for: GUI tests and components and API testing

The following steps describe the tasks most often performed using the Asset Comparison Tool and the Asset Viewer.

## Open the Asset Viewer

You can open the Asset Viewer from any of the following:

<b>The main UFT window</b>	<ol style="list-style-type: none"><li>1. Open the test, component, or function library for which you want to view an earlier version.</li><li>2. Select <b>ALM &gt; Version History</b>. The Version History dialog box opens.</li><li>3. Select a version and click <b>View</b>. The Asset Viewer opens.</li></ol>
<b>The Object Repository Manager (GUI Testing only)</b>	<ol style="list-style-type: none"><li>1. Open the Object Repository Manager (<b>Resources &gt; Object Repository Manager</b>).</li><li>2. Browse to and open the shared object repository for which you want to view an earlier version. For details, see "<a href="#">How to Create and Manage Shared Object Repositories</a>" on page 206.</li><li>3. Select <b>ALM &gt; Version History</b>. The Version History dialog box opens.</li><li>4. Select a version and click <b>View</b>. The Asset Viewer opens.</li></ol>
<b>The Recovery Scenario Manager (GUI Testing only)</b>	<ol style="list-style-type: none"><li>1. Open the Recovery Scenario Manager (<b>Resources &gt; Recovery Scenario Manager</b>).</li><li>2. Open the recovery scenario file for which you want to view an earlier version.</li><li>3. Click the <b>Version Control</b> down arrow and select <b>Version History</b>.</li><li>4. Select a version and click <b>View</b>. The Asset Viewer opens.</li></ol>
<b>From within ALM</b>	Connect to the project containing the asset you want to view, and do one of the following: <ul style="list-style-type: none"><li>• <b>To view the current version of an asset:</b> In the Test Resources module, select the resource and click the <b>Resource Viewer</b> tab.</li></ul>

	<ul style="list-style-type: none"> <li>● <b>To view the current or earlier version of an asset:</b> <ol style="list-style-type: none"> <li>a. Do one of the following:           <ul style="list-style-type: none"> <li>○ In the sidebar, click the <b>Test Plan</b> button (for tests) or the <b>Business Components</b> button (for components) to open the Test Plan or Business Components module.</li> <li>○ Click the <b>Test Resources</b> button to open the <b>Test Resources</b> module. This module contains the resource files associated with your test or component, such as function libraries, shared object repositories, data tables, recovery scenarios, and environment variable XML files.</li> </ul> </li> <li>b. In the tree, select the file for which you want to view an earlier version.</li> <li>c. Click the <b>History</b> tab, and then click the <b>Versions</b> or <b>Baselines</b> tab.</li> <li>d. In the grid, select a version, and then click the <b>View</b> button. (You cannot view a version that is currently checked out.) A window opens with buttons in the sidebar enabling you to access version-specific information for the selected asset. (These buttons are identical to the tabs displayed in the right pane of the main window for the latest version of the selected asset.) For details, see the ALM user guide.</li> </ol> </li> </ul>
<b>The Command Line Interpreter (cmd.exe) (tests only)</b>	<ol style="list-style-type: none"> <li>1. Open the Command Line Interpreter.</li> <li>2. Enter the command using the following syntax:</li> </ol> <pre style="background-color: #f0f0f0; padding: 10px;">"&lt;UFT installation folder&gt;\bin\UFTDiffApplication.exe" P1: "&lt;file path 1&gt;"</pre> <p>where <b>P1</b> = the file system path to the asset.</p> <p><b>Example:</b></p> <pre style="background-color: #f0f0f0; padding: 10px;">"%ProgramFiles%\HP\Unified Functional Testing\bin\UFTDiffApplication.exe" P1: "%ProgramFiles%\HP\Unified Functional Testing\Tests\Test1"</pre>

## Open the Asset Comparison Tool

You can open the Asset Comparison Tool from any of the following:

<b>The main UFT window</b>	<ol style="list-style-type: none"> <li>1. Open the test, component, or function library (GUI testing only) whose versions you want to compare.</li> <li>2. Select <b>ALM &gt; Version History</b> or <b>Baseline History</b>. The Version History or Baseline History dialog box opens.</li> <li>3. Select two versions (using the <b>CTRL</b> key) and click <b>Compare</b>. The Asset Comparison Tool opens.</li> </ol>
<b>The Object Repository Manager (GUI testing only)</b>	<ol style="list-style-type: none"> <li>1. Open the Object Repository Manager (<b>Resources &gt; Object Repository Manager</b>).</li> <li>2. Browse to and open the shared object repository whose versions you want to compare. For details, see <a href="#">"How to Create and Manage Shared Object Repositories" on page 206</a>.</li> <li>3. Select <b>ALM &gt; Version History</b> or <b>Baseline History</b>. The Version History or Baseline History dialog box opens.</li> <li>4. Select two versions (using the <b>CTRL</b> key) and click <b>Compare</b>. The Asset Comparison Tool opens.</li> </ol>

<b>Recovery Scenario Manager (GUI testing only)</b>	<ol style="list-style-type: none"> <li>1. Open the Recovery Scenario Manager (<b>Resources &gt; Recovery Scenario Manager</b>).</li> <li>2. Open the recovery scenario file whose versions you want to compare.</li> <li>3. Click the <b>Version Control</b> down arrow and select <b>Version History</b> or <b>Baseline History</b>.</li> <li>4. Select two versions (using the <b>CTRL</b> key) and click <b>Compare</b>. The Asset Comparison Tool opens.</li> </ol>
<b>From within ALM</b>	<ol style="list-style-type: none"> <li>1. In ALM, connect to the project containing the asset you want to compare.</li> <li>2. Do one of the following: <ul style="list-style-type: none"> <li>• In the sidebar, click the <b>Test Plan</b> button (for tests) or the <b>Business Components</b> button (for components) to open the Test Plan or Business Components module.</li> <li>• Click the <b>Test Resources</b> button in the sidebar to open the <b>Test Resources</b> module. This module contains the resource files associated with your test or component, such as function libraries, shared object repositories, data tables, recovery scenarios, or environment variable XML files.</li> </ul> </li> <li>3. In the tree, select the file whose versions you want to compare.</li> <li>4. Click the <b>History</b> tab, and then click the <b>Versions</b> or <b>Baselines</b> tab.</li> <li>5. In the grid, select two versions to compare (using the <b>CTRL</b> key), and then click the <b>Compare</b> button.</li> <li>6. In the sidebar of the window that opens, click the <b>QTP Comparison/ST Comparison or Automation</b> button. The Asset Comparison Tool opens.</li> </ol> <p><b>Tip:</b> You can also compare baselines from the <b>Management</b> module. Click the <b>Management</b> button in the side bar to open the <b>Management</b> module. Select a baseline in the tree and click the  <b>Compare To</b> button. For details, see the ALM user guide. For more details on baselines, see "Managing Versions of Assets in ALM" on page 952.</p>
<b>The Command Line Interpreter (cmd.exe) (tests only)</b>	<ol style="list-style-type: none"> <li>1. Open the Command Line Interpreter.</li> <li>2. Enter the command using the following syntax:</li> </ol> <pre>"&lt;UFT installation folder&gt;\bin\UFTDiffApplication.exe" P1: "&lt;file path 1&gt;" P2: "&lt;file path 2&gt;"</pre> <p>where <b>P1</b> = the file system path to the first asset, and <b>P2</b> = the file system path to the second asset.</p> <p><b>Note:</b> Make sure you insert a blank space after each argument. The options are not case-sensitive and can be entered in any order.</p> <p><b>Example</b></p> <pre>""%ProgramFiles%\HP\Unified Functional Testing\bin\UFTDiffApplication.exe" P1: "%ProgramFiles%\HP\Unified Functional Testing\Tests\Test1" P2: "%ProgramFiles%\HP\Unified Functional Testing\Tests\Test2"</pre>

## View a comparison of two asset versions (Asset Comparison Tool)

1. With a test or component selected, select **ALM > Version History**. The Version History dialog box opens.
2. In the Version History dialog box, select two versions to compare by pressing the version row and the **CTRL** key.
3. Click **Compare**. The Asset Comparison Tool window opens with a detailed list of the similarities and differences between the selected versions.

## Drill down to compare or view a specific element (GUI testing only)

**Note:** You can drill down any asset that has a blue drilldown arrow ↴ adjacent to it.

- Click the blue drilldown arrow ↴ adjacent to any asset that can be compared. (The pointer changes into a pointing hand in the proximity of the drilldown arrow.)
- Double-click the asset.
- Right-click the asset and select **View Drilldown**.
- Select an asset and press **ENTER** on your keyboard.

## View the UFT location of an element (GUI testing only)

Right-click the relevant node and select **View Sample Snapshot**. The screen capture displays an example of the relevant dialog box. The option (or area) for the node you right-clicked is highlighted in the screen capture. For details, see "[Asset Comparison Tool and Asset Viewer](#)" on page 955.

## Modify text and background colors

Modify the text and background colors for the filter types (modified, added, deleted, and so on) in the Asset Comparison Tool window using the Color Settings dialog box.

When you modify the background color of a filter type, the color of the filter type in the legend at the top of the window changes accordingly. These changes remain in effect unless you change them again or restore the default settings.

## View the number of differences for a specific element

In the Asset Comparison Tool, collapse the node representing an element.

If the sub-elements of that element are different between versions, a legend is displayed adjacent to the node. The legend indicates the number of differences that exist under the collapsed element.

In the following example, three sub-elements were modified, one was deleted, and seven were added:



# Troubleshooting and Limitations - ALM Version Control

## **Relevant for: GUI tests and components, API testing, and business process tests and flows**

This section describes troubleshooting and limitations for ALM version control.

- If you need to check out an earlier version of an asset, for example, to roll back to an earlier version, contact your ALM project administrator. Your administrator needs to ensure that the correct versions of all relevant assets become the latest versions.
- When working with a version-control-enabled ALM project, it takes a long time to save a test for the first time (up to twice as long as saving the same test in a project without version control support enabled). This delay does not occur on subsequent saves of the test.

# Troubleshooting and Limitations - Asset Comparison Tool

## **Relevant for: GUI tests and components and API testing**

This section describes troubleshooting and limitations for the Asset Comparison Tool and the Asset Viewer.

### **For GUI tests and components:**

- Sometimes, checkpoint and output value comparison information disappears from the bottom panes of the Asset Comparison Tool after refreshing it. This can occur if you:
  - a. Open the Asset Comparison Tool to view a comparison.
  - b. Switch to the Object Repository Manager.
  - c. Modify any of the current comparison's checkpoints or output values.
  - d. Save your changes.
  - e. Switch back to the Asset Comparison Tool.
  - f. Refresh the Asset Comparison Tool.
  - g. Select the checkpoint or output value that you modified.

**Workaround:** If this occurs, close the Asset Comparison Tool, and then open it again after you save your changes.

- When comparing two baselines, if the only change in a resource is its association to a test or component, the Asset Comparison Tool does not indicate any change in the resource even though ALM may indicate that the resource is **Modified**.
- When working with a localized installation of UFT, selecting the **View Sample Snapshot** option in the UFT Asset Comparison Tool opens a window containing a sample image of the selected element in UFT. The image displays the English user interface.

### **For API tests and components**

- The Asset Comparison tool is not available for API components.
- The Asset Comparison tool does not show differences for the following features:
  - Changes of properties for asynchronous Web service calls
  - Changes in multipart properties for HTTP Request steps
  - Changes in XML fault checkpoints

- Changes in attachments for Web service calls and SOAP Request steps
- Changes in security settings for Web services and SOAP Request steps

# Chapter 63: HP Sprinter

## Relevant for: GUI tests only

Although UFT automated tests can answer many of your testing needs, you may also need to perform some of your tests manually. You can run your manual tests using HP Sprinter, HP's solution for manual testing. Sprinter provides advanced functionality and tools to make manual testing more efficient and effective.

Manual testing often requires that you leave your testing application to accomplish tasks related to your test. For example, you may need to use graphic software to take a screen capture of your application, you may want to record a movie of the application during the test, and you need to switch to your defect tracking software to report defects.

Sprinter addresses these needs in the manual testing process, and enables you to accomplish these tasks without disrupting your test flow. Sprinter includes many tools to help you detect and submit defects. Sprinter can also perform many of the repetitive and tedious tasks of manual testing for you. These features ensure that you can perform all the tasks necessary for your manual test with minimum interruptions to your testing work.

Sprinter is fully integrated with ALM 11.00 and later, enabling you to get the maximum benefit from both solutions.

**Note:** Unified Functional Testing (UFT) and Sprinter share a variety of system resources. Consider the following when deciding whether to install Sprinter on your UFT computer:

- Sprinter and UFT can be installed on the same computer.
- Sprinter and UFT cannot be run simultaneously on the same computer.
- Any changes to the installation of one of these products will affect the other. If you uninstall, modify, or upgrade one product, the other may fail. You need to repair the installation of the affected product. For details, see the *HP Unified Functional Testing Installation Guide* and the *HP Sprinter Readme*.

For details on the supported compatible versions of UFT and Sprinter, see the *HP Unified Functional Testing Product Availability Matrix*.

With Sprinter you can:

<b>Run ALM manual tests and Business Process tests with new step display</b>	<ul style="list-style-type: none"><li>• <b>Clear steps display.</b> Steps are presented in a clear, organized, and user-friendly design, making it easier to view step information and update step status.</li><li>• <b>Move easily between tests in your run.</b> You can easily move between the tests in your run without interrupting your test flow. Sprinter updates all your displayed step and run information to match your current test.</li><li>• <b>Edit actual parameter values during your test run.</b> You can easily edit the actual values of parameters in your test, during your test run.</li><li>• <b>Multiple views.</b> Change the way you view your steps depending on your testing needs. View in normal</li></ul>
------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	<p>mode when more details are needed, or view in Subtitles mode if you need to see more of your application.</p> <ul style="list-style-type: none"> <li>• <b>Actual value including screen captures.</b> Attach a plain or annotated screen capture of your application to the step's actual value.</li> </ul>
<b>Create formal tests from exploratory tests with no pre-defined steps.</b>	If you run a test without pre-defined steps, Sprinter can keep a record of all the user actions you took during your test. You can then export this list to an Excel spreadsheet, modify the text as needed and import the spreadsheet to a test in ALM. This converts an exploratory test to a formal test, with pre-defined steps.
<b>Submit defects to ALM</b>	Submit an ALM defect directly from within Sprinter. You can optionally let Sprinter create a defect scenario by automatically generating a text description of all the user actions or steps in your test. You can also attach a screen capture or a movie of your application to the defect.
<b>Create and annotate screen captures of your application.</b>	Sprinter provides tools that enable you to take and annotate a screen capture of your application at any point in the testing process. Tools are included that make measuring and comparing user interface elements easier. Report defects in the display by attaching the annotated screen capture to an ALM defect, saving it as a file, or attaching it to an email. You can also include annotated screen captures in the Actual Result of a step.
<b>Let Sprinter perform some manual testing tasks for you.</b>	You can create and run <b>macros</b> to automate a set of actions in your application. Sprinter can also <b>inject data</b> automatically into fields in your application.
<b>Replicate your actions on another computer.</b>	Replicate your user actions on multiple computers with different configurations (operating system, browser). Sprinter detects differences in the displays of these computers and enables you to report defects on these differences.
<b>View test results.</b>	Sprinter includes a powerful Storyboard that displays each action you performed in your test. For each action you can see a screen capture of the action, any defects that you reported, defect reminders, and comments. If you ran the test with multiple configurations you can view the differences between the displays of different computers.
<b>Use Sprinter tests to create UFT tests and business components</b>	Using Sprinter data files, you can automatically convert your manual tests into GUI tests or business components by importing these files into UFT. UFT then converts the data into a test or business component.

For more details, contact your HP representative.

# Part 10: Business Process Testing in UFT

# Chapter 64: Business Process Testing in UFT

**Relevant for: Business process tests and flows**

This chapter includes:

• Business Process Testing Overview .....	971
• Business Process Test and Flow Overview .....	971
• Business Process Testing in UFT - Overview .....	972
• Comparing BPT in UFT to BPT in ALM .....	973
• Business Process Testing Methodologies .....	974
• How to Set Up UFT for Business Process Testing .....	978
• How to Create, Maintain, and Run Business Process Testing Tests and Flows in UFT .....	981
• Business Process Testing in UFT - End-to-End Scenario .....	984
• Business Process Testing in UFT User Interface .....	993
• Business Process Testing in UFT - BPT View .....	996
• Troubleshooting and Limitations - Business Process Testing in UFT .....	1001

# Business Process Testing Overview

## **Relevant for: business process testing**

Business Process Testing provides you with a customizable, component-based testing framework that supports:

### **Component reuse and modularization**

Component reuse and modularization keep costs low by speeding up test creation, maintenance, and execution.

### **Creation of tests for both simple and complex applications**

An application under test can be a simple, HTML-based web application or a complex business process involving packaged applications and back-end services and databases.

### **Collaboration between various personas**

The testing framework is flexible enough to meet the needs of various personas, such as manual testers, automation engineers, and subject matters experts.

Business Process Testing helps you document your components and tests, including screenshots illustrating how they should be used, and so on. This makes it possible for people with different roles and skill sets to share others assets.

### **Management of parts of a test**

Managing parts of a test includes component documentation, test run results, version control, reporting, and history. Additionally, using ALM, you can generate documents containing information about the tests, flow, and components in a given project.

In Business Process Testing, you arrange components, including keyword GUI components, scripted GUI components, or manual components in a business process test or business process flow to test different scenarios. Business Process Testing enables you to arrange components to suit your needs.

**Note:** Business Process Testing is available with ALM Edition and Quality Center Enterprise Edition.

For more information about the HP Business Process Testing editions and their functionality, see the *HP Application Lifecycle Management User Guide*. To find out what edition of HP Business Process Testing you are using, ask your ALM site administrator.

# Business Process Test and Flow Overview

## **Relevant for: business process testing**

When working with Business Process Testing, you can use both **business process tests** and **business process flows** to organize your testing.

Each business process test or flow consists of business components, which are added and ordered (or grouped) together. When UFT runs a business process test, it runs each of the components and their steps in sequence.

You can also include a business process flow inside a business process test.

While tests and flows are fundamentally the same as they both contain a specified order of business components, there are differences:

- A business process test is a scenario comprising a sequence of business components or flows, designed to test a specific scenario in an application.
- A flow is a type of test that comprises a logical set of business components, in a fixed sequence, that performs a specific task. Flows share the same functionality as business process tests (for example, iterations, parameters, and results). When designing flows, they can be considered as "compound components."

In addition, flows cannot contain other flows.

You can use a flow in multiple business process tests. When you modify a flow or any of its components, all business process tests containing that flow reflect that modification.

## Business Process Testing in UFT - Overview

### Relevant for: business process tests and flows

In UFT, you can create and edit business process (BPT) tests and flows with the native UFT [Toolbox](#), [Data](#), and [Properties](#) panes.

To work with BPT from within UFT, you must first connect to an ALM project with BPT support.

In UFT, you can create and edit keyword GUI components, scripted GUI components, and API components. For details about keyword and scripted GUI components, see "[Business Components](#)" on [page 1002](#). API components are much like API tests, described in "[API Testing Design with the UFT IDE](#)" on [page 388](#). Some exceptions and limitations for working with API components are described in "[Working with API components](#)" on [page 1001](#).

In addition, business components use application areas to store settings and resources that may be required by multiple components, such as shared object repositories and function libraries. Components are automatically linked to all of the resources and settings defined in the associated application area. For more details about application areas, see "[Application Areas](#)" on [page 1014](#).

For a task on how to create and edit your business process tests in UFT, see "[How to Create, Maintain, and Run Business Process Testing Tests and Flows in UFT](#)" on [page 981](#).

Using the BPT View (an alternate version of the Start Page for Business Process Testing users), you can create, set up, and maintain all your business process tests and flows.

Business process tests and flows can also include manual testing components. For details about manual testing, as well as details about using BPT in ALM, see the *HP Business Process Testing User Guide* in the ALM documentation set.

## Comparing BPT in UFT to BPT in ALM

### **Relevant for: business process tests and flows**

Business process tests and flows are displayed in the UFT document pane, in a display similar to the ALM Test Script tab. The following table describes where to find the BPT functionality you are familiar with from working in ALM, when you are working in UFT.

For details about using BPT in ALM, see the *HP Business Process Testing User Guide*.

Function	ALM	UFT
Create or open test or flow	Test Plan > New or Details button	Open/New <Document>/<Resource> Dialog Box
Check in and check out tests and flows	Version Control user interface	ALM menu command
View test properties	Test Plan > Test Details dialog box > Details tab	General Properties Tab in the Properties Pane
View, add, or modify test parameters	Test Plan > Parameters tab	Test Parameters Tab in the Properties Pane
Add component or flow to test or flow	Test Plan > Test Script tab > Select Components and Flows pane	Toolbox Pane
Reorder, group, or open components or flows in tests	Test Plan > Test Script tab	BPT Test tab in the document pane
View run conditions or failure settings for components or flows		
Edit run conditions for components in flows	Test Plan > Test Script tab > Run Conditions button	Properties tab in the Properties pane
Edit failure settings for components in flows	Test Plan > Test Script tab > On Failure link in grid	Properties tab in the Properties pane
Add or modify component parameters	Business Components > Parameters tab	Parameters tab of the Properties pane
Link parameters	Test Plan > I/O Parameters link in grid > Link I/O column > Select Output Parameter dialog box	Select Link Source dialog box
Promote parameters	Test Plan > Test Script tab > Select Components and Flows pane > QuickAdd button > Add While Setting Promote Options	Parameters tab of the Properties pane Data Pane BPT Testing tab of the Options dialog box
View and modify BPT comments for a specific component instance	Test Plan > Test Script tab > Comments tab	Component tab of the Properties pane

Function	ALM	UFT
Add and modify iterations and iteration data for components, flows, and groups	Test Plan > Test Script tab > Iterations link in grid	Data Pane
Run or debug test	Test Plan > Test Script tab > Run or Debug Test button	Run dialog box

## Business Process Testing Methodologies

### **Relevant for: business process tests and flows**

BPT is flexible and does not require any one particular model for incorporating business processes into your testing environment. The actual workflow in an organization may differ for different projects, or at different stages of the application development life cycle.

The chapters in this guide are structured according to the bottom-up methodology.

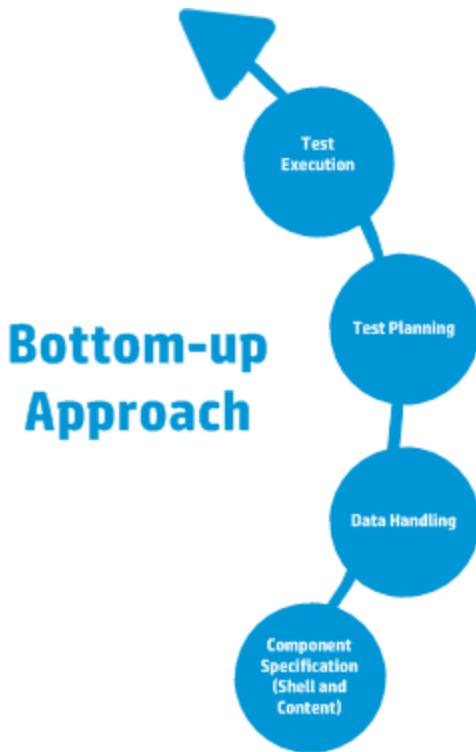
### **Bottom-up Methodology**

Defining low-level components first and then designing business process tests based on the defined components is called a bottom-up methodology. This methodology is particularly useful:

- For regression testing
- When the business processes in the organization are clearly defined
- When users are new to BPT

Using the approach enables you to create resources - components, application areas, and object repositories that can be reused across tests. For example, you can use the same component in a number of tests. Likewise, you can design many different components that contain the same application area (which is based upon a specific area of your application).

The bottom-up methodology is based on the following design phases:



Phase	Description
<b>Component Specification</b>	<ul style="list-style-type: none"> <li>Develop a component tree with components.</li> <li>Create the component shell by adding basic details</li> <li>Create component content by adding manual and/or automated implementations. Component content can contain:           <ul style="list-style-type: none"> <li>Manual implementation for manual components</li> <li>Automation, for automated components</li> <li>Both manual implementation and automation</li> </ul> </li> </ul>
<b>Data Handling</b>	Design the data that each business process test, flow, or component uses when run
<b>Test Planning</b>	Build test plans and design business process tests and flows
<b>Test Execution</b>	Create a subset of the business process tests in your project and run them

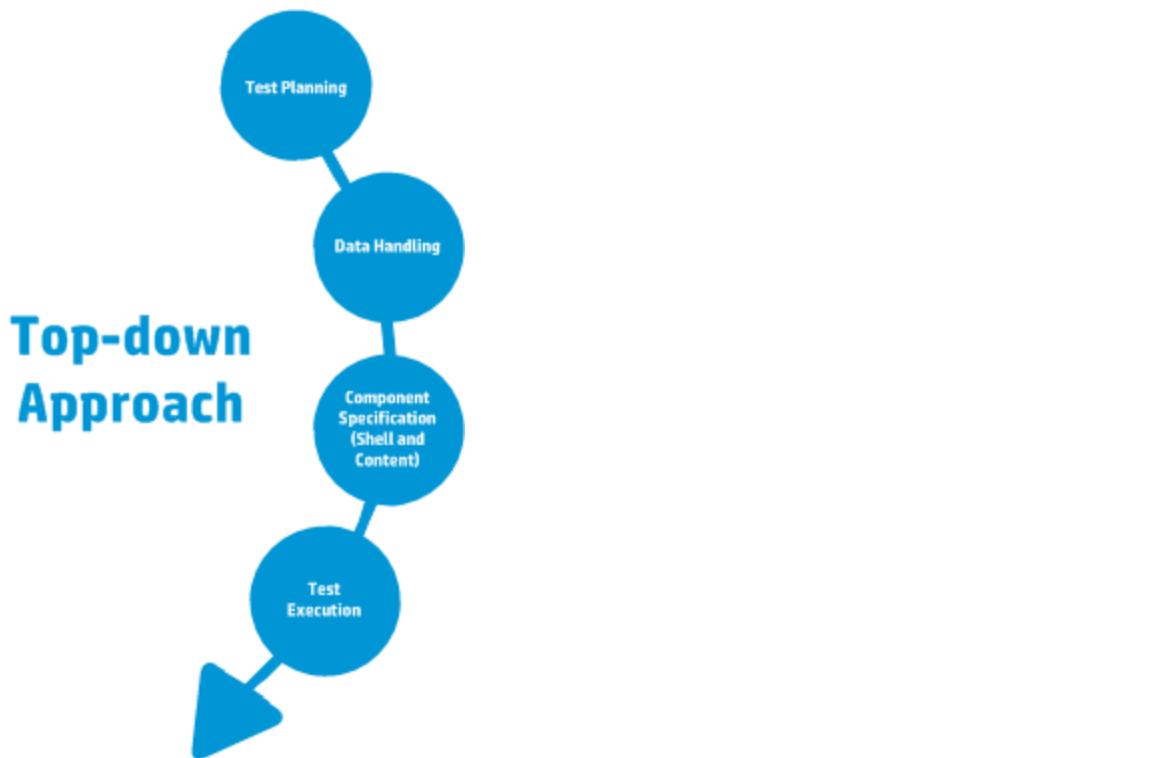
## Top-down Methodology

The top-down methodology advocates the creation of business process testing entities according to the following hierarchy:

- Business process tests, which contain flows and/or business components
- Flows, which contain business components

- Business components, which contain manual and/or automated steps

The top-down methodology is based on the following design phases:



Design Phase	Description
<b>High-level design</b>	<p>Includes the high-level design, creation of a structure for business process tests, and determining the test configurations for testing different use-cases that will be needed.</p> <p>When designing at the high-level, facilitate automation:</p> <ul style="list-style-type: none"><li>• By designing with modularity in mind. Design tests to use smaller, reusable components that automated tests can call multiple times</li><li>• By designing tests with reusable components, which makes maintaining tests easier</li></ul>
<b>Mid-level design</b>	<p>Includes the:</p> <ul style="list-style-type: none"><li>• Creation of flows (sets of business components in a logical order that can be run). Flows are considered "compound components."</li><li>• Creation of business components (reusable units that perform specific tasks in a business process). Only the shell of the component is created during this phase.</li><li>• Specification of criteria for more granular test coverage (requirements) as necessary.</li><li>• Linking to other testing documents.</li><li>• Adding business components to business process tests and flows.</li></ul>
<b>Low-level implementation</b>	<p>Includes the low-level implementation of business component content by:</p> <ul style="list-style-type: none"><li>• Creating component steps (the content of the business component), including automated steps when necessary</li></ul>

- Grouping components
- Setting up iterations (for business process tests, flows, groups, and components)
- Parameterizing

## Agile Methodology

This approach is based on using BPT to provide testing in sprints, as developers code features for the application under test. Components and tests are created and updated in parallel with development.

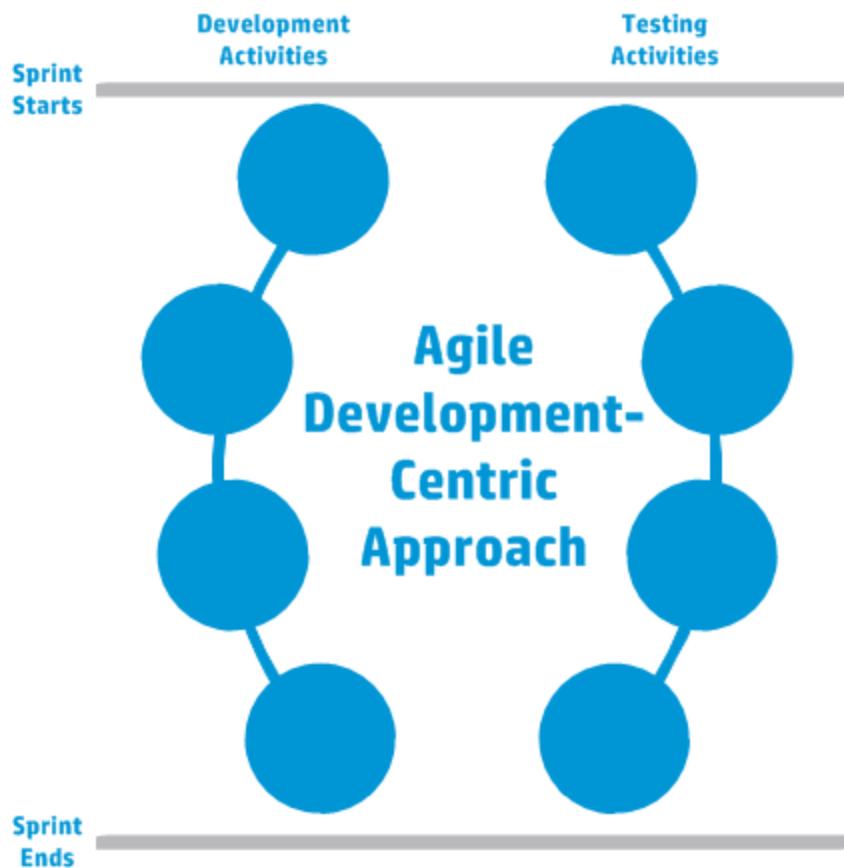
### Example

If the application under test is implemented in Java, components might be grouped by the classes that represent certain groups of UI elements, such as toolbar buttons. Each time a button is added to the toolbar, the component representing that class is updated.

This approach encourages:

- **Automation.** Because sprints are short, it is important to automate as much as possible.
- **Component reuse.** Component reuse can be designed in the same way that the developers implement modularly for reuse.

The following presents the Agile Development-centric approach.



## How to Set Up UFT for Business Process Testing

### **Relevant for: business process tests and flows**

Before you create business process tests and steps, you must set up UFT. This enables you to create your tests without having to stop frequently to troubleshoot UFT program issues.

This task includes the following steps:

- "Prerequisites" below
- "Install and load the correct addins in UFT" on the next page
- "Configure settings for UFT to work with your application's technology" on page 980
- "Select a test creation methodology" on page 980

### **Prerequisites**

Before using UFT for business process testing, you must do the following:

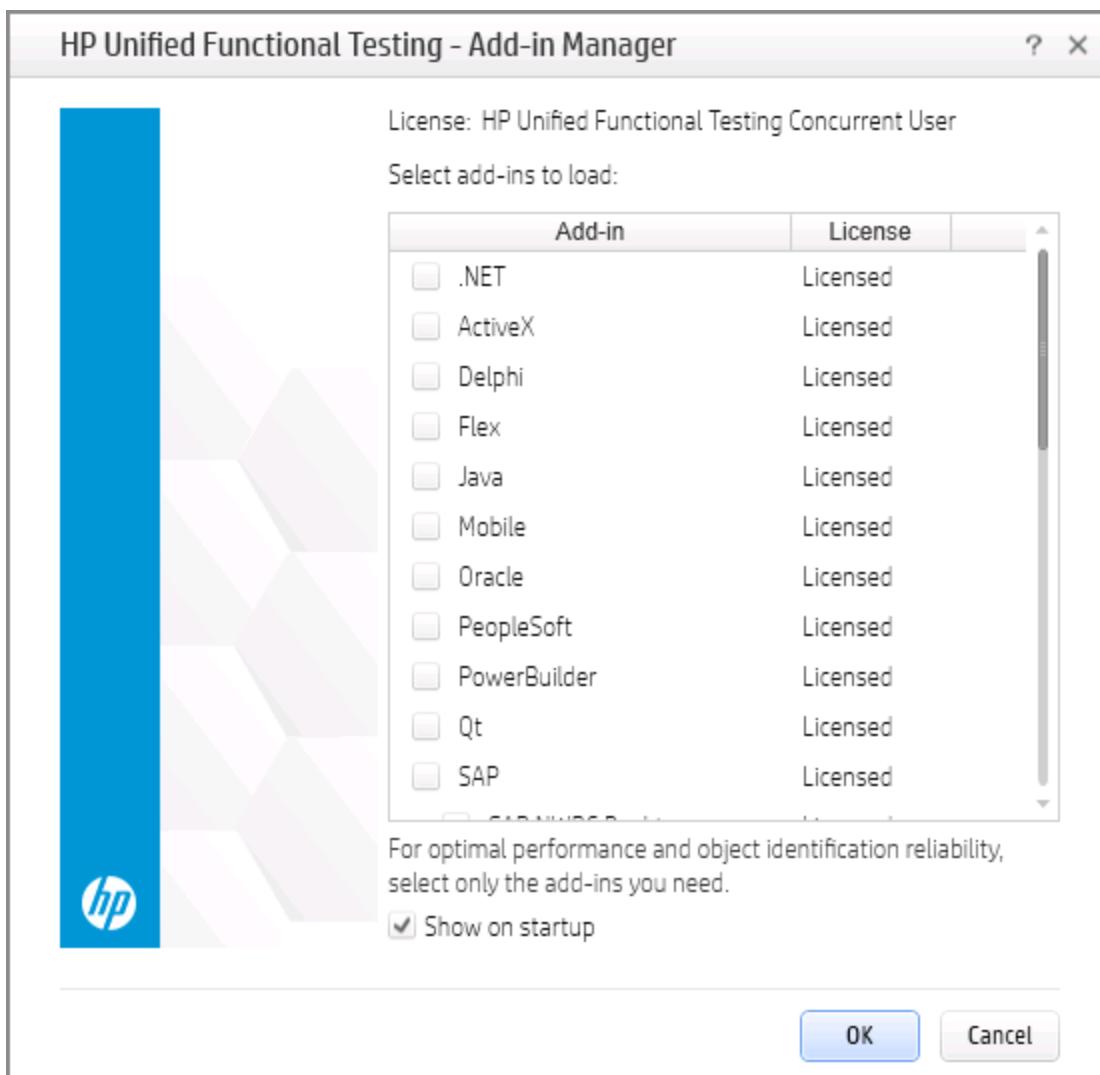
- Have access to an ALM server with a Business Process Testing license
- Connect to ALM

- Install a Unified Functional Testing license on the UFT machine

### Install and load the correct addins in UFT

In order to create a business process test of your application, UFT must be able to recognize the technology in your application. Therefore, you must load the proper technology support.

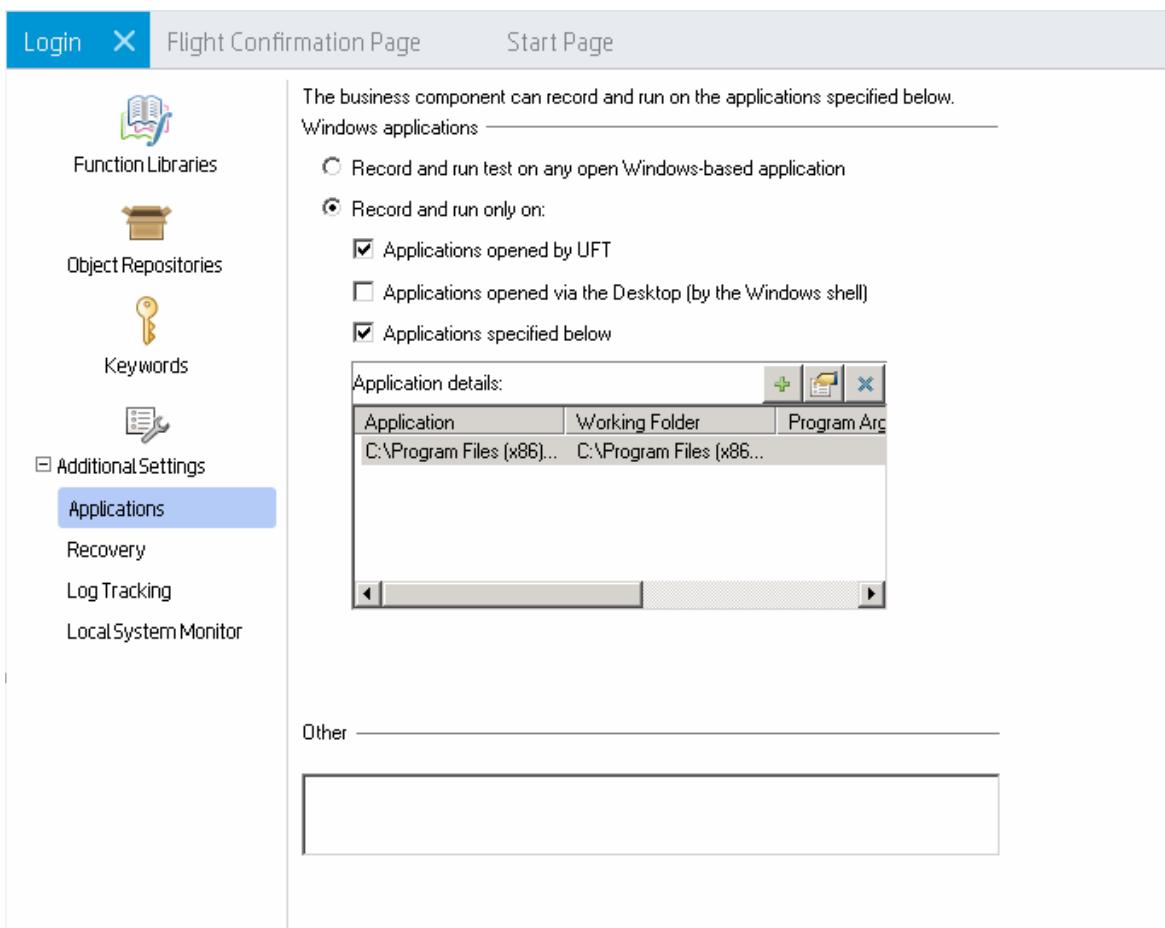
1. During the UFT installation, in the installation wizard, install the correct add-ins in the **Custom Setup** screen. For details on the installation process, see the *HP Unified Functional Testing Installation Guide*.
2. When starting UFT, load the correct add-ins in the Add-in Manager:



After you load the add-in, UFT can communicate and recognize objects in your application.

## Configure settings for UFT to work with your application's technology

For many technologies, you must configure UFT to recognize and communicate with the application. This is done in the application area's Additional Settings tab:



**Note:** You must add your application area to the current solution to edit these settings. To add the application area, select **File > Add > Existing Application Area** and navigate to the relevant application area in your ALM project.

## Select a test creation methodology

When creating a business process test, you have the option of creating your test in different ways:

<b>Component-centered (or bottom-up)</b>	<p>Using this methodology, you first create individual components for each area or page of your application, including test steps inside each of the components. Then, in the business process test, you insert the components in the desired order, grouping the components as needed and running individual components for multiple iterations.</p> <p>This approach enables you to reuse components in multiple tests. This approach, however, does not approach the test creation as an end-to-end scenario when planning the test. Instead, creating end-to-end tests of the</p>
------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	business processes of your application may require additional planning.
<b>Test flow-centered (or top-down)</b>	Using this methodology, you create a high-level test flow, and then separate the test into components as needed. You can divide the test steps into multiple components as you create the test. This approach enables you to see the overall user or business process flow when designing the test. As a result, you can ensure that you create an end-to-end test flow of your application. This approach, however, may make the reuse of components more difficult.

## How to Create, Maintain, and Run Business Process Testing Tests and Flows in UFT

### **Relevant for: business process tests and flows**

This task describes the high-level steps on how to create, maintain, and run business process tests and flows in UFT.

This task includes the following steps:

- "[Prerequisites](#)" below
- "[Create application areas for each area of your application](#)" below
- "[Create components](#)" on the next page
- "[Add components to business process tests and flows](#)" on the next page
- "[Add steps to your component](#)" on page 983
- "[Group components and flows](#)" on page 983
- "[Use parameters in your test](#)" on page 983
- "[Iterate components and flows](#)" on page 983
- "[Add a test configuration to your test](#)" on page 983
- "[Debug and run your test](#)" on page 984
- "[View the run results](#)" on page 984

### **[Prerequisites](#)**

- When opening UFT, select the same add-ins in the Add-in Manager Dialog Box as the test objects in your component object repositories (both the component's local object repository or the application area's object repository).
- Connect to an ALM server and project. (All business process tests, flows, and business components are saved in their respective locations in your ALM project.)

### **[Create application areas for each area of your application](#)**

Before creating tests and their components, you should create application areas for each area of your application. The application area contains the object repositories with the test objects (representing

real objects in your application), function libraries containing functions to use in a component, and specific settings to use for the component.

To create an application area, do the following:

1. Do one of the following:

- In the toolbar, click the **New** button down arrow  and select **New Application Area**.
- In the BPT View, click the **Add a New Application Area** button.

2. In the New Application Area dialog box, navigate to the directory in your ALM project in which you want to save the application area and provide a name for the application area.
3. Click **Create**. UFT adds the application area to your ALM project and opens the application area in the document pane.

## Create components

Components make up the building blocks of a business process test. Therefore, before creating the business process tests, you must create the individual components.

**Note:** If you are recording the steps in your test, you do not need to create components before beginning to record. For details on how to add components to the test by recording, see "Add components to business process tests and flows" below.

1. In the toolbar, click the **New** button down arrow  and select **New Business Component**.
2. In the New Business Component dialog box, select the type of component: **Keyword GUI** or **Scripted GUI**.
3. In the New Business Component dialog box, provide a **Name** and **Location** for the new component.
4. In the **Application Area** field, click the **Browse** button and navigate to an application area to use with your component.
5. Click **Create**. UFT opens the new component in the document pane.

## Add components to business process tests and flows

Your business process test consists of components, groups of components, or business process flows. In order to run a business process test, you must build the test with its components.

<b>From the Toolbox pane</b>	<ol style="list-style-type: none"><li>1. Open the Toolbox pane.</li><li>2. Under the <b>Components</b> tree, expand the nodes and navigate to the component or flow you want to add.</li><li>3. Double-click or drag and drop the component to the test grid or canvas.</li><li>4. Order the items in your test or flow as needed by dragging and dropping the individual components or using the arrow buttons  in the BPT toolbar.</li></ol>
------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<b>While recording</b>	<ol style="list-style-type: none"><li>1. In the toolbar, click the <b>Record</b> button .</li><li>2. In the New Business Component dialog box, specify a <b>Name</b>, <b>Location</b>, and default <b>Application Area</b> for the test.</li><li>3. Record user actions on your application.</li><li>4. As you record, in the Record toolbar, click the <b>New Business Component</b> button .</li><li>5. In the New Business Component dialog box, specify the name of the new component. Components added during recording are saved in the same location specified in the beginning of the recording session.</li><li>6. When you are finished recording, click the <b>Stop</b> button .</li></ol> <p>UFT adds all the recording components to the test with the recording steps (from the steps you performed on your application).</p>
------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## Add steps to your component

For details, see "[How to Create Test Steps in a Business Process Test](#)" on page 1023.

## Group components and flows

In the document pane (grid view or canvas view), select the components or flows you want to group, and click **Group** .

To ungroup, select the group and click **Ungroup** .

**Note:** When iterating groups, all items to be included in the group must have the same number of iterations, or an error message is displayed when you group the items.

## Use parameters in your test

For details, see "[How to Use Data in a Business Process Test](#)" on page 1039

Default values for component or flow parameters are used during the run session, if no other value is supplied.

## Iterate components and flows

By default, each component or flow you add to your test has a single iteration. If you need to run specific components multiple times, you can add iterations for these components and specify different values for the component parameters in each iteration.

For details on defining iterations and parameter values for the iterations, see "[Add iterations for a component or flow](#)" on page 1042 and "[Set data values for the parameters for each iteration](#)" on page 1042.

## Add a test configuration to your test

You can add test configurations to your test to enable you to run the test with varying sets of data. For details, see "[How to Set Up and Run Test Configurations](#)" on page 1061.

## Debug and run your test

<p><b>To debug a test or component</b></p>	<p>Insert breakpoints in specific components or flows in your test, and then run your test. The run session stops at each breakpoint.</p> <p><b>Note:</b> If you run a BPT test from the ALM Test Plan module or from UFT, UFT stops the test at all breakpoints in both <b>Debug</b> and <b>Normal</b> modes. However, before running the BPT test, you <b>must open</b> the business components with the breakpoints and add them to the solution in UFT.</p>
<p><b>To run your test:</b></p>	<p>In the toolbar, click the <b>Run</b> button .</p> <p><b>Note:</b></p> <ul style="list-style-type: none"> <li>Before running a test from ALM, you must enable the <b>Allow other HP products to run tests and components</b> option in the Test Runs pane (<b>Tools &gt; Options &gt; GUI Testing tab &gt; Test Runs</b> node)</li> <li>For improved performance when running business process tests or flows, UFT creates and runs a hosting test, named <b>Test Runtime</b>. The <b>Test Runtime</b> test is recreated each time the test or flow runs, and is not saved with the run.</li> </ul>

## View the run results

After the business process test run is complete, by default, the run results open. You can choose to display them in the Run Results Viewer or as an HTML-based report:

1. In the Run Sessions pane of the Options dialog box (**Tools > Options > General tab > Run Sessions** node), select the report format you want: **HTML Report** or **Run Results Viewer Report**.
2. Run your test.

The results are displayed in the document pane (for a HTML report) or in the Run Results Viewer.

**Note:** You can view the report in HTML format only when your business process test is stored in an ALM server running version 12.50.

# Business Process Testing in UFT - End-to-End Scenario

### Relevant for: business process tests and flows

When you plan to use Business Process Testing in UFT to test your application, there are a number of stages, from planning to the test run. This use-case scenario will show the end-to-end process of using Business Process Testing in UFT to test a real application.

For the purposes of this use-case-scenario, the application being tested is the UFT sample flight reservation application (**Start > All Programs > HP Software > HP Unified Functional Testing > Sample Applications > Flight GUI**).

Implementing a complete test of the application requires a number of steps:

- "Analyze your application" below
- "Prepare your test infrastructure." below
- "Create the business process test and add steps to the test." on page 988
- "Enhance your test." on page 991
- "Run your test." on page 992

## 1. Analyze your application

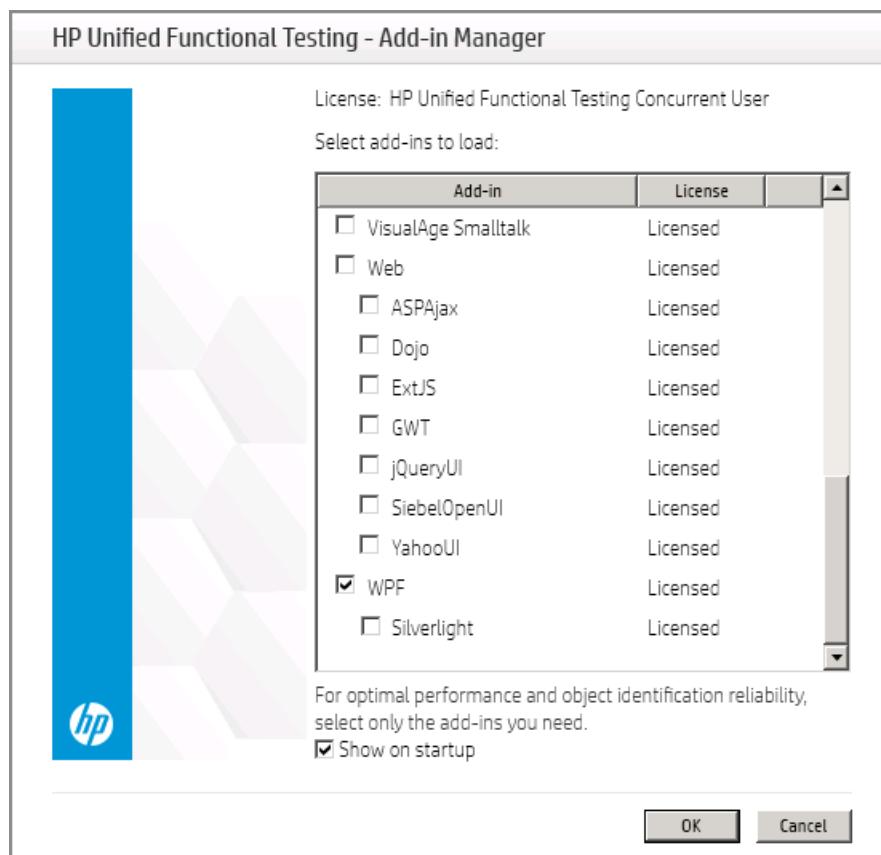
Before you can begin creating actual components and tests of the flight reservation application, you need to analyze the application and see what is needed to create a comprehensive test:

<b>Determine the technologies used in your applications.</b>	<p>This is important as you must load the appropriate add-ins in UFT to test your application's technologies. Depending on what technologies your application uses, UFT loads the necessary tools that enable UFT to communicate with and recognize objects in your application.</p> <p>For the flight reservation application, the application is based on Windows Presentation Foundation (WPF) technologies, so you must load the WPF Add-in in UFT.</p>
<b>Determine the functionality that you want to test.</b>	<p>Before you begin, you must consider how a user uses the application. Once you do this, you can determine how what resources you need and what components you need to create for the different part of your applications.</p> <p>For the flight reservation application, you have a multiple areas of the application, with a number of user actions:</p> <ol style="list-style-type: none"> <li>a. <b>Login window:</b> The user logs into the flight reservation application</li> <li>b. <b>Flight search window:</b> The user searches for available flights or booked flights, including:           <ul style="list-style-type: none"> <li>◦ Find a flight based on departure and arrival cities and departure date</li> <li>◦ Find a flight based on the seating class of the passenger</li> <li>◦ Search for booked flights by customer name, flight number, or flight date</li> </ul> </li> <li>c. <b>Flight selection window:</b> Select a flight from the available flights</li> <li>d. <b>Flight confirmation and booking window:</b> Confirm the selected flight and entering the customer details, including:           <ul style="list-style-type: none"> <li>◦ Entering the customer's name</li> <li>◦ Entering the exact number of tickets needed</li> <li>◦ Booking the flight</li> <li>◦ Waiting for confirmation of the order from the application</li> <li>◦ Restarting the reservation process</li> </ul> </li> </ol>
<b>Decide how to divide the testable functionality into smaller units</b>	<p>By dividing the test into smaller chunks, you can make it more manageable and easier to follow.</p> <p>Based on this list of application windows and tasks, you can create four different components for the different areas of the application.</p>

## 2. Prepare your test infrastructure.

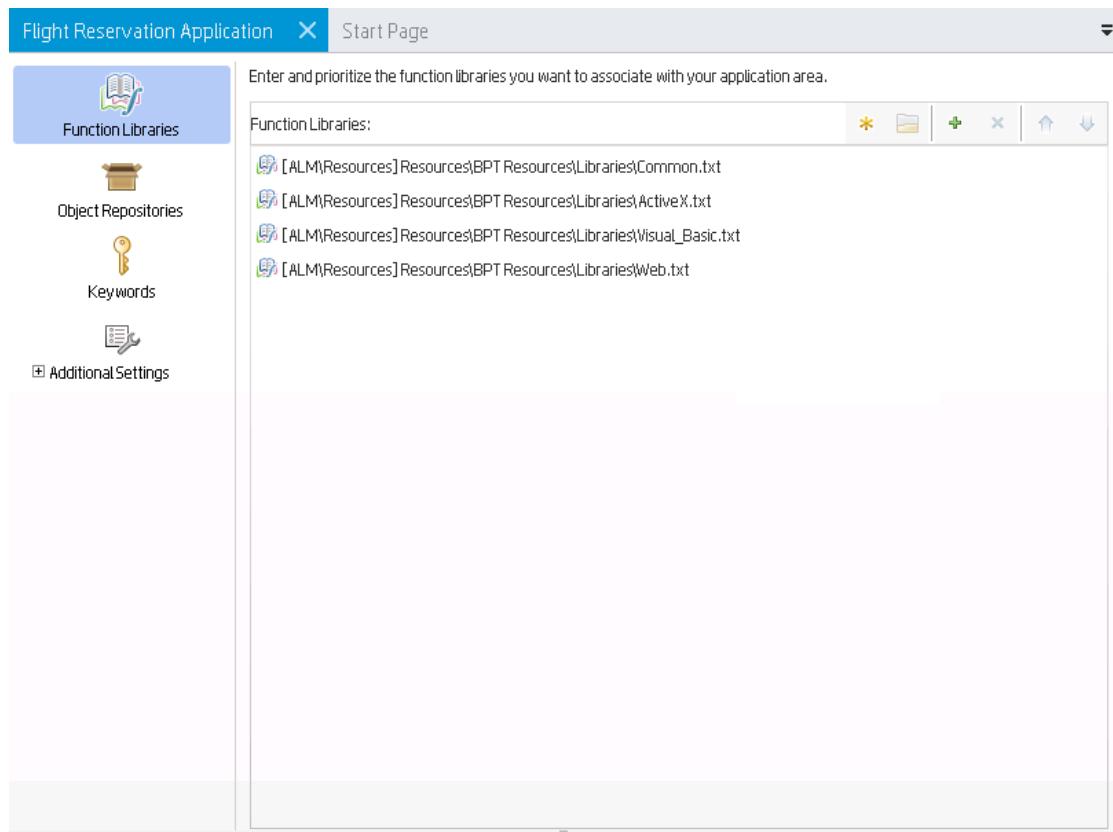
Before you begin creating the tests, you need to prepare UFT and the required resources.

**First**, you need to load the WPF add-in in the Add-in Manager. This loads the necessary support for UFT to work with WPF-based applications.

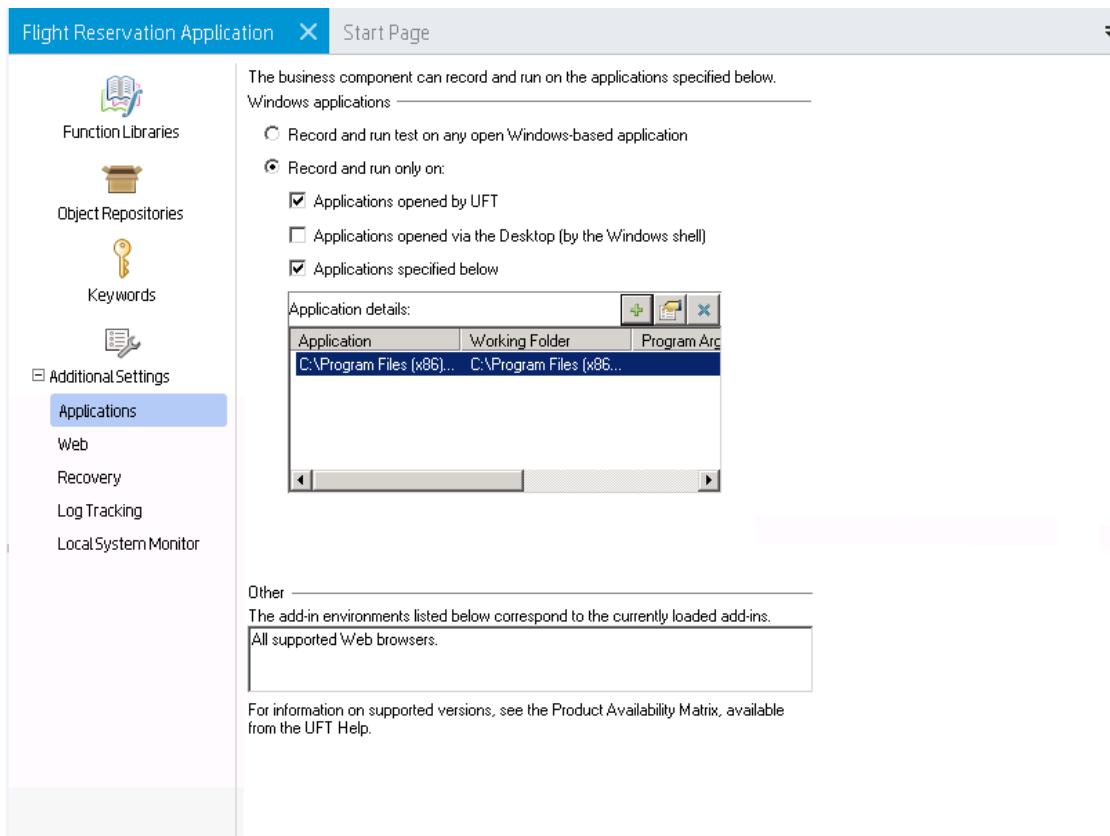


**Second**, after UFT starts, you must then connect to ALM to enable yourself to use Business Process Testing with UFT. Once you are successfully connected with ALM, UFT displays an icon  in the lower right corner of the main window.

**Third**, you must [create or have a default application area](#). This application area contains the object repositories, custom function libraries, and application settings:



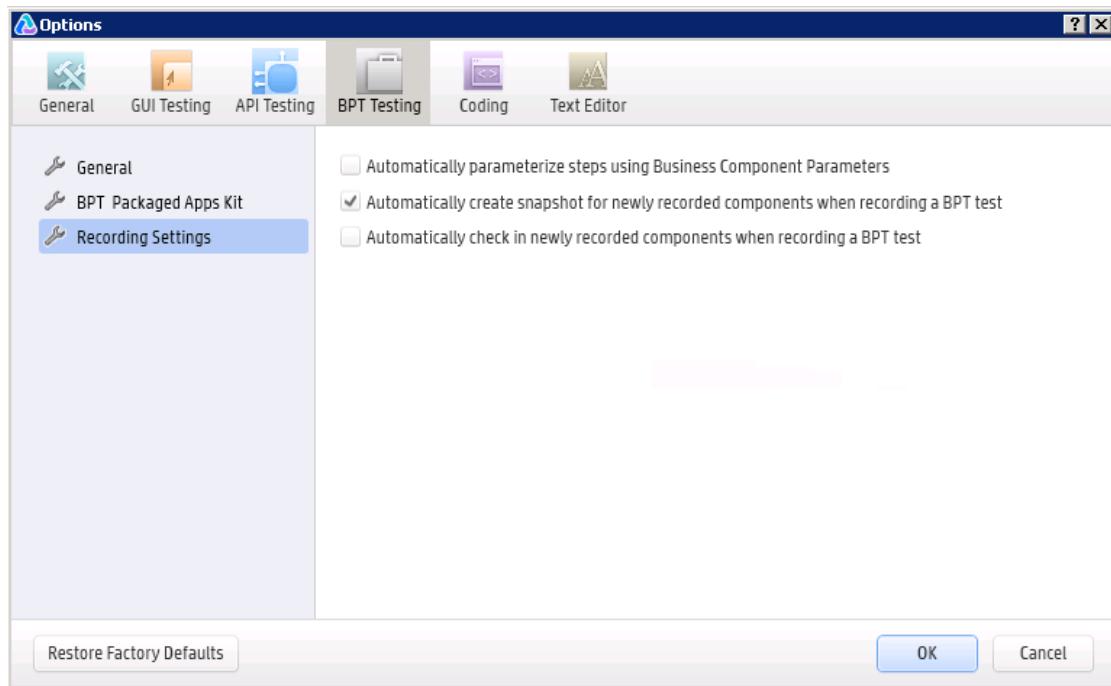
**Lastly**, in the application area, settings you must configure UFT to work with the flight reservation application:



3. **Create the business process test and add steps to the test.**

Once you have created a default application area and configured it work with the flight reservation application, you can begin creating a business process test. For this use-case scenario, you will record the steps and create the business process test.

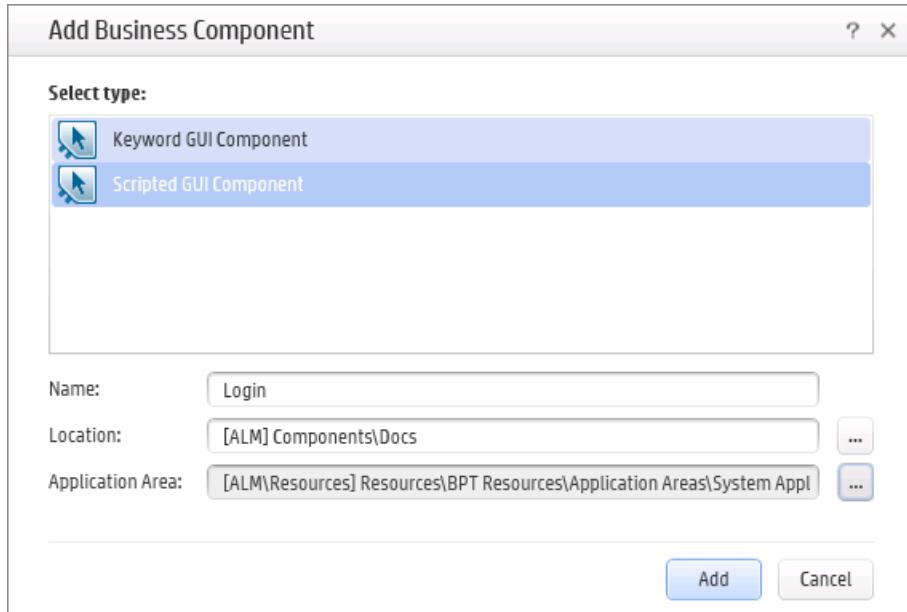
Before you begin recording, set the options for how UFT records a business process test in the **Recording Settings** pane of the Options dialog box (**Tools > Options > BPT Testing tab > Recording Settings node**):



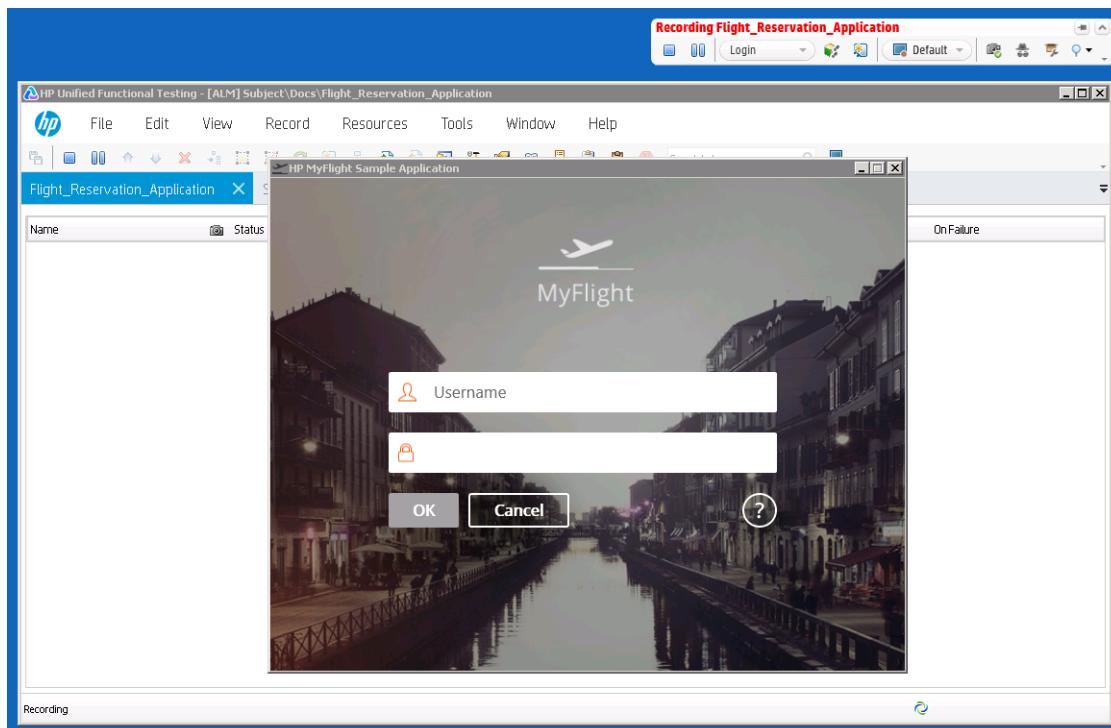
For this scenario, you instruct UFT to **Automatically parameterize steps using Business Component Parameters**. This enables UFT to create a parameter for each object in your application. After recording your test, you can later parameterize steps in the test using these parameters.

However, before you can begin adding the test, you create a new business process test.

After the test is created, then press the **Record** button  in the toolbar to start creating the test. UFT prompts you to create a business component. When specifying the settings for the component, select the application area created previously:



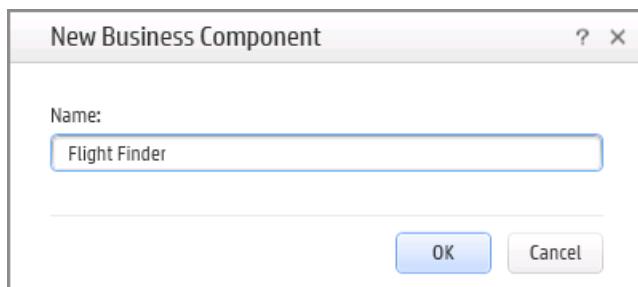
After you add this business component, the UFT window goes into the background and the Record Toolbar is minimized. Open the flight reservation application:



Perform actions in your application and UFT records the actions. As you perform these actions, the Record Toolbar is updated with the number of steps recorded:



In addition, as you record, you have the option of creating new components to divide the test into logical parts. Click the **Add New Business Component** button in the Record Toolbar to create a new business component:



Subsequent steps are recorded in the new business component.

Continue performing steps in the flight reservation application, and UFT continues recording these steps into the test. In addition, you should logically create additional business components for the Select Flight page (the table where you select available flights) and the Flight Confirmation page (where you enter passenger details).

After recording all the user actions, press the **Stop** button in the Record Toolbar. The Record Toolbar closes, and UFT creates the components and orders them in the test:

Name	Status	I/O Parameters	Iterations	Run Condition	On Failure
Login [1]	Under Development	8In	1 Iterations	Continue	
Flight Finder [1]	Under Development	9In	1 Iterations	Continue	
Select Flight [1]	Under Development		1 Iterations	Continue	
Flight Confirmation [1]	Under Development	1In	1 Iterations	Continue	

You do not need to save anything - it is automatically saved in ALM and the components are saved in the specified place in ALM.

#### 4. Enhance your test.

Now that your basic test has been created, you can modify the test in a number of ways:

<b>Group components</b>	Select components and group them together to enable certain functionalities (like common parameterization, running, etc.) for all components in the
-------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------

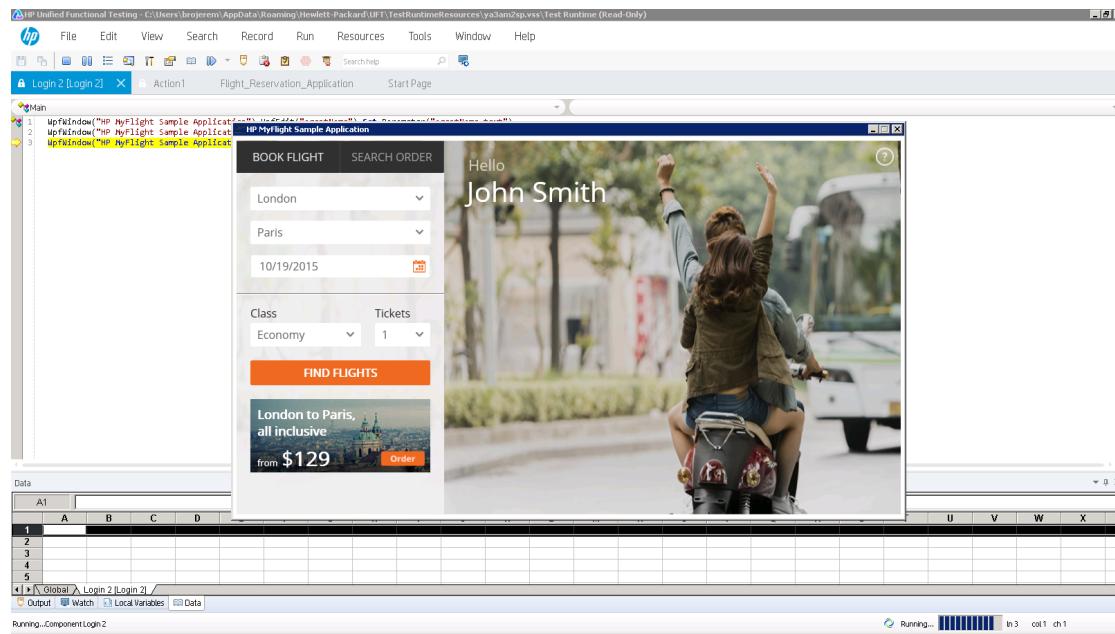
	<p><b>group:</b></p>																											
<b>Change parameter values across iterations</b>	<p>If you select any component in the test, and then view the Parameters tab in the Properties pane, you see that UFT has automatically created component parameters for the objects used in the component steps.</p> <p>In the data pane, add additional iterations and change the values of the parameters in other iterations:</p> <table border="1"> <thead> <tr> <th></th> <th>fromCity_item</th> <th>toCity_item</th> <th>WpfImage_X</th> <th>WpfImage_Y</th> <th>WpfImage_X_1</th> <th>WpfImage_Y_1</th> <th>Su_Date</th> <th>Class_Item</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>Denver</td> <td>Los Angeles</td> <td>0</td> <td>9</td> <td>9</td> <td>9</td> <td>22-Oct-2015</td> <td>Business</td> </tr> <tr> <td></td> <td>London</td> <td>Frankfurt</td> <td></td> <td></td> <td></td> <td></td> <td>23-Oct-2015</td> <td>Economy</td> </tr> </tbody> </table>		fromCity_item	toCity_item	WpfImage_X	WpfImage_Y	WpfImage_X_1	WpfImage_Y_1	Su_Date	Class_Item	1	Denver	Los Angeles	0	9	9	9	22-Oct-2015	Business		London	Frankfurt					23-Oct-2015	Economy
	fromCity_item	toCity_item	WpfImage_X	WpfImage_Y	WpfImage_X_1	WpfImage_Y_1	Su_Date	Class_Item																				
1	Denver	Los Angeles	0	9	9	9	22-Oct-2015	Business																				
	London	Frankfurt					23-Oct-2015	Economy																				
<b>Set run conditions for components in the test</b>	<p>If you select a component, in the Properties tab of the Properties pane, you can specify to stop (<b>Exit</b>) or continue (<b>Continue</b>) the test if a component fails on a particular test run.</p>																											
<b>Create a test configuration to change the parameterization of your components across test runs</b>	<p>In the Test Configurations tab of the Properties pane, you can create a test configuration to vary the data your test uses when running. (This data is created by uploading an Excel spreadsheet.)</p> <p>Then, when UFT runs the test, it uses the data provided by the test configuration instead of the set values provided in the individual components.</p>																											

## 5. Run your test.

In the toolbar, click the **Run** button. UFT automatically runs the test, performing each step in the application:

## User Guide

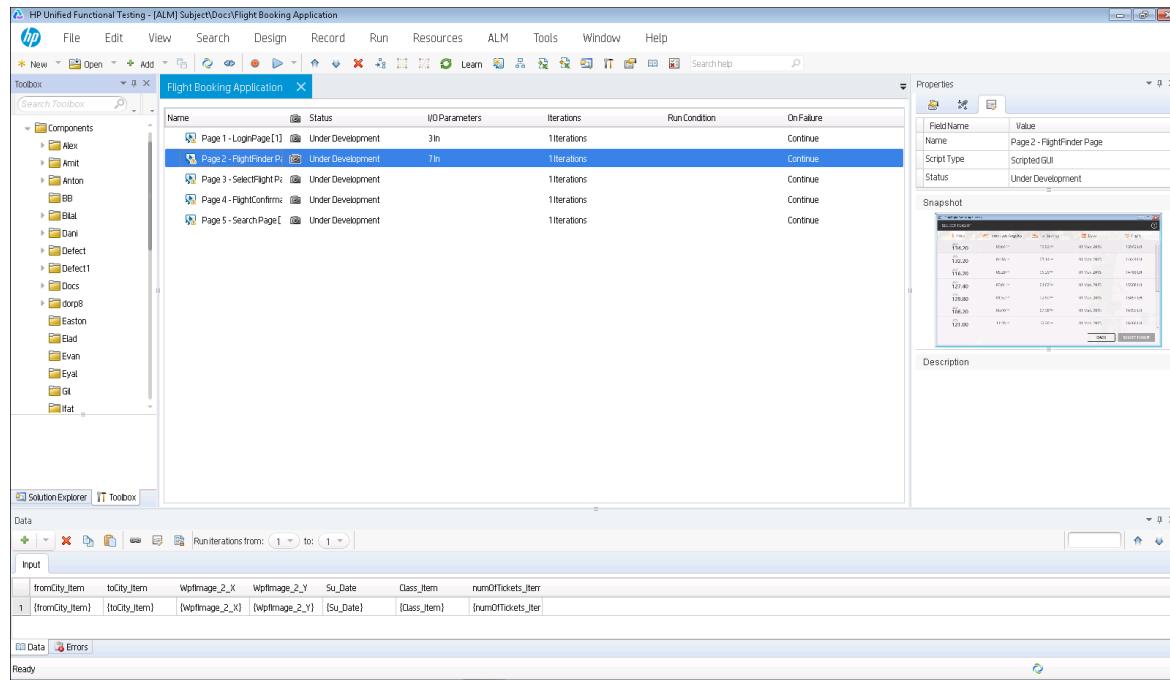
### Chapter 64: Business Process Testing in UFT

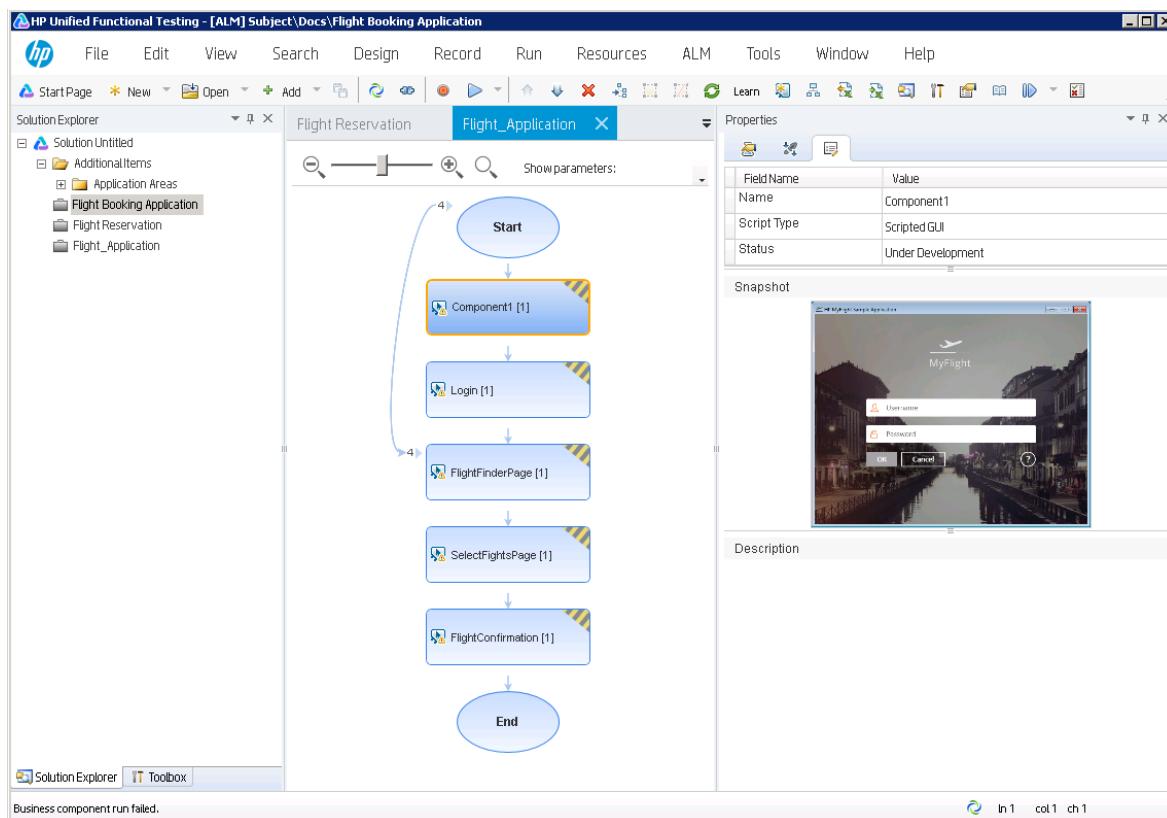


## Business Process Testing in UFT User Interface

### Relevant for: business process tests and flows

Business process tests and flows are displayed in the document pane in a grid. Selecting a specific component, flow, or group in the document pane causes information to change in the Properties and Data panes.





<b>To access</b>	"Prerequisites" on page 981. The business process test or flow opens as a tab in the document pane. To switch between the canvas view and the grid view, in the toolbar, click the <b>Toggle between the Grid View and Canvas View</b> button
<b>Important information</b>	<ul style="list-style-type: none"> <li>"Business Process Testing Overview" on page 971</li> <li>"Business Process Test and Flow Overview" on page 971</li> <li>"Business Process Testing in UFT - Overview" on page 972</li> </ul>
<b>Relevant tasks</b>	"How to Create, Maintain, and Run Business Process Testing Tests and Flows in UFT" on page 981

User interface elements are described below:

- "User Interface Elements (Grid Columns)" on the next page
- "Toolbar Controls" on the next page
- "Context menu items" on page 996

## User Interface Elements (Grid Columns)

UI Element	Description
<b>Name</b>	The name of the component or flow.
	<b>Snapshot.</b> Indicates whether a snapshot exists for this component.
<b>Status</b>	The ALM workflow status for the components and flows in the test.
<b>I/O Parameters</b>	The number of parameters defined for the selected component or flow.
<b>Iterations</b>	The number of iterations defined in the pane for the selected component or flow.
<b>Run Conditions</b>	<p>The attributes that must match for the component to run.</p> <p><b>Note:</b> You can modify these attributes only when editing a flow. This value is blank for components contained in a test.</p>
<b>On Failure</b>	<p>The action for UFT to take if a specific business component in the flow fails.</p> <p>Valid values include:</p> <ul style="list-style-type: none"> <li>• <b>Exit.</b> The business process test run ends if the selected business component fails.</li> <li>• <b>Continue.</b> (Default) The business process test will run the next business component or flow if the selected component fails.</li> </ul>

## Toolbar Controls

The following toolbar buttons are included in the BPT toolbar above the document pane.

Button	Description
	<b>Move up/down.</b> Enables you to change the order of entities in the business process test or flow by moving a selected component, group, or flow up or down.
	<b>Delete from Test.</b> Removes the selected business component, group, or flow from the business process test or flow.  <b>Caution:</b> If you are deleting the last component in a group, the entire group is deleted.
	<b>Go to Component/Flow.</b> Opens the selected component or flow in the document pane.  The component or flow is also added to the solution and listed in the Solution Explorer.
	<b>Group.</b> Creates a group that includes the selected business components and/or flows.  The components and flows must be contiguous. A component or flow can belong to one group only.  A group node is created above the grouped items and is identified by the group icon. By default, the group is named Group, followed by a unique number.

Button	Description
	<p><b>Tip:</b></p> <ul style="list-style-type: none"> <li>You can add other business components or flows to an existing group by dragging and dropping a component or flow from the Toolbox pane to the relevant position in the group.</li> <li>You can change the order of the members of the group by dragging and dropping.</li> </ul>
	<p><b>Ungroup.</b> Ungroups components and/or flows.</p> <p>To completely remove a group, including its members, select the group and click <b>Delete from Test</b> .</p> <p>To remove a business component or flow from a group, select the component or flow, drag it up or down out of the group, and drop it at the required location.</p>
	<p><b>Refresh.</b> Refreshes a currently open read-only test with any updated information.</p>

## Context menu items

The following table describes the context menu items available from the document pane, when editing a business process test or flow.

Button	Description
	<p><b>Move up/down.</b> Enables you to change the order of entities in the business process test or flow by moving a selected component, group, or flow up or down.</p>
	<p><b>Delete from Test.</b> Removes the selected business component, group, or flow from the business process test or flow.</p>
	<p><b>Go to Component/Flow.</b> Opens the selected component or flow in the document pane.</p> <p>The component or flow is also added to the solution and listed in the Solution Explorer.</p>
	<p><b>Group.</b> Creates a group that includes the selected business components and/or flows.</p>
	<p><b>Ungroup.</b> Ungroups components and/or flows.</p>

# Business Process Testing in UFT - BPT View

## Relevant for: business process tests and flows

**What is the BPT view?** The BPT View is a single page (similar to the UFT Start Page) that enables you to perform necessary BPT tasks with a single click. Each of the basic BPT test creation functionalities can be accessed via the BPT View.

**When you should use the BPT view?** You should use the BPT view to streamline your workflow when working with BPT in UFT. This view enables you to access all functionalities without the need to open additional panes, dialog boxes, or menus.

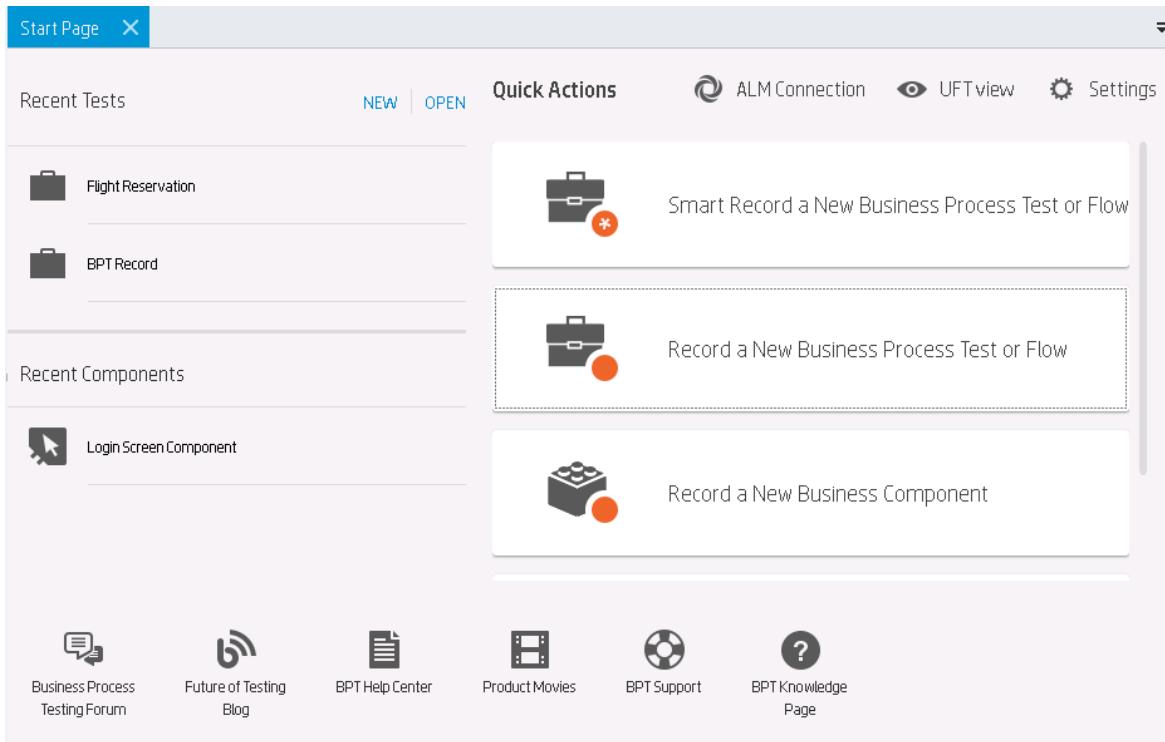
### What do you want to know?

- "How do I access the BPT view?" below
- "How to do I record a business process test?" on the next page
- "How to do I smart record a business process test?" on page 999
- "How to do I record a business component?" on page 999
- "How to do I create and set up an application area?" on page 999
- "How to do I configure my UFT settings for business process testing?" on page 999

### How do I access the BPT view?

When UFT opens, by default the Start Page is displayed. To change the Start Page to the BPT View, select **View > Switch to BPT View**. Each subsequent time you open UFT, the BPT View is displayed.

### What is available in the BPT view?



From the BPT View, you have a number of options:

<b>Recent Tests/ Recent Components</b>	A list of the most recently opened tests and components. Click on a test or component name to open it.
----------------------------------------	--------------------------------------------------------------------------------------------------------

<b>New/Open</b>	Enables you to create a business process test/flow or open a business process test/flow if it is not displayed in the list of recent tests or components.
<b>ALM Connection</b>	Opens the ALM Connection dialog box, which enables you to enter your ALM credentials and connect to your ALM server and project.  You must connect to ALM before creating or opening any business process test or flow.
<b>UFT View</b>	Opens the regular UFT Start Page.
<b>Settings</b>	Opens the Options dialog box, where you can set testing options for components that perform GUI testing or API testing, as well as general Business Process Testing options.  For details on the different Options dialog panes, see " <a href="#">Global Options</a> " on page 80.
<b>Smart Record a New Business Process Test or Flow</b>	Enables you to learn areas of your SAP application and create tests/components based on the different areas of your SAP application.  This option is displayed only when you install and load the Add-in for SAP Solutions.
<b>Record a New Business Process Test or Flow</b>	Enables you to create and record steps in a new business process test or flow.
<b>Record a New Business Component</b>	Enables you to create and record steps in a new business component.
<b>Add a New Application Area</b>	Enables you to create and set up a new application area for use with your business process tests and business components.
<b>Social Icons and Buttons</b>	Enables you to connect with other HP Unified Functional Testing and Business Process Testing users, through: <ul style="list-style-type: none"><li>• Business Process Testing Forum</li><li>• Future of Testing Blog</li><li>• BPT Support</li><li>• Knowledge Base</li></ul> In addition, you can also open the UFT Help right to the Business Process Testing section of the Help.

## How to do I record a business process test?

From the BPT View, you click the **Record a New Business Process Test or Flow** button, and UFT guides you to start recording the steps for your test.

After you click the **Record** button, UFT can prompt you twice:

- First with the dialog to create a new business process test/flow **and** add that test to the solution. If you do not want to add the created test to the solution, press **Cancel**.
- If you choose not add the created test to a solution, with a dialog to create the business process test or flow. Enter the test details in the dialog box and the test is automatically created and saved.

After you select how to include the test in UFT, you can begin recording steps. For details, see "[How to Record a Business Process Test](#)" on page 1025.

## How to do I smart record a business process test?

Using the built-in functionality of the BPT Packaged Apps Kit, UFT can easily learn parts of your SAP applications and convert the learned information into individual components.

To start learning your SAP application, click the **Smart Record a New Business Process Test or Flow** button, and UFT guides you to start recording the steps for your test.

After you click the Record button, UFT can prompt you twice:

- First with the dialog to create a new business process test/flow **and** add that test to the solution. If you do not want to add the created test to the solution, press **Cancel**.
- If you choose not add the created test to a solution, with a dialog to create the business process test or flow. Enter the test details in the dialog box and the test is automatically created and saved.

After you select how to include the test in UFT, you can begin learning your application. For details, see "[How to Learn Business Process Tests and Flows](#)" on page 1073.

## How to do I record a business component?

From the BPT View, you click the Record a New Business Component button. UFT guides you to start recording steps for your component.

For details, see "[How to Record a GUI Test or Component](#)" on page 99

## How to do I create and set up an application area?

As part of the creating components, you must associate an application area with the component. However, you must create the application area before creating the component.

From the BPT View, you can create an application area by clicking the **Add a New Application Area** button. UFT guides you to create and set up the application area.

For conceptual details on what an application area is, see "[Application Areas - Overview](#)" on page 1015. For details on how to create and set up an application area, see "[How to Create and Manage Application Areas](#)" on page 1016.

## How to do I configure my UFT settings for business process testing?

When you use Business Process Testing in UFT, you can set a variety of options:

- **Options relevant for components testing the GUI of your application:** This includes how UFT recognizes and runs tests of your application's user interface. For details on the GUI testing options, see the options in the GUI Testing tab of the Options dialog box.
- **Options relevant for components testing the API layer of your application.** For details, see the options in the API Testing tab of the Options dialog box.
- **Options relevant for using Business Process Testing in UFT.** This includes how UFT records tests, uses parameters, and how to use the BPT Packaged Apps Kit. For details, see the options in the BPT Testing tab of the Options dialog box.

**You may also want to learn:**

- "[Business Process Testing Overview](#)" on page 971
- "[Business Process Testing in UFT User Interface](#)" on page 993
- "[BPT Packaged Apps Kit - Overview](#)" on page 1068
- "[How to Create, Maintain, and Run Business Process Testing Tests and Flows in UFT](#)" on page 981
- "[How to Create Test Steps in a Business Process Test](#)" on page 1023
- "[How to Set Up UFT for Business Process Testing](#)" on page 978

# Troubleshooting and Limitations - Business Process Testing in UFT

## **Relevant for: business process tests and flows**

This section includes:

- "[General functionality](#)" below
- "[Working with API components](#)" below

### **General functionality**

To create BPT tests and flows in UFT, you must work with ALM 11.52 or later.

### **Working with API components**

You can create and manage API components in much the same way as you would API tests, as described in "[API Testing Design with the UFT IDE](#)" on page 388. The following exceptions apply when working with API components:

- You cannot use [Load Test activities](#) in a component. If you save a load-enabled test as a business component, it will no longer be load-enabled.
- You cannot use [Automated Testing Tool](#) activities in a component.
- Encoded password type properties are not supported. If the component has a property of type password (or encrypted in ALM 11.00 and later), the value will be treated as an ordinary string, without encoding or decoding.
- Multiple User Variable profiles are not supported. Remove all but one of the profiles.

**Note:** Other general differences between test and components also apply to API components, such as "[Troubleshooting - Naming Conventions](#)" on page 1144

# Chapter 65: Business Components

**Relevant for: GUI components only**

**Note:** Unless otherwise specified, references to **Application Lifecycle Management** or **ALM** apply to all currently supported versions of ALM and Quality Center. Note that some features and options may not be supported in the specific edition of ALM or Quality Center that you are using.

For details, see the *HP Unified Functional Testing Product Availability Matrix* and the *HP Application Lifecycle Management User Guide* or the *HP Quality Center User Guide*.

This chapter includes:

• <a href="#">Business Components - Overview</a> .....	1003
• <a href="#">Business Component Categories</a> .....	1004
• <a href="#">Keyword GUI Components Overview</a> .....	1005
• <a href="#">Converting Manual Components</a> .....	1006
• <a href="#">Tips and Guidelines for Selecting Application Areas for Your Component</a> .....	1007
• <a href="#">Scripted GUI Components Overview</a> .....	1007
• <a href="#">How to Create and Manage GUI Business Components</a> .....	1008
• <a href="#">How to Convert a Keyword GUI Component to a Scripted GUI Component</a> .....	1011
• <a href="#">GUI Component Statuses</a> .....	1012
• <a href="#">Troubleshooting and Limitations - Business Components</a> .....	1013

# Business Components - Overview

## Relevant for: business process tests and flows

You can create, define, modify, and manage business components in UFT or ALM. Business components provide the basis for BPT and are incorporated into business process tests and flows.

A business component is a reusable unit that:

- Performs a specific task in a business process
- Describes the condition or state of the application before and after that task

You can use a component in multiple business process tests and flows. When you modify a component or its content, all business process tests or flows containing that component reflect that modification. You can also use run conditions to enable components to run selectively, based on earlier stages within the test or flow.

## Business Component Examples

Business Component Name	Task	Application State Before	Application State After
Login	Banker logs into banking application	<none>	The application is launched and the main home page is displayed.
SearchLoan	Banker searches for an existing loan	The banker is logged in and the main home page is displayed.	The application displays the main loan details page, or a page indicating that the loan was not found.

Business components are comprised of:

- **A Shell** (general information such as component name, status).
- **Content** (steps or scripts). Low-level, detailed information, such as the component's manual steps and/or automation. The contents provide detailed instructions for performing business process tasks in the application. Component content can be manual, automated, or both—depending on whether you create a manual implementation and/or automation for the component.
  - For details on automated components, see "[Keyword GUI Components Overview](#)" on page 1005 and "[Scripted GUI Components Overview](#)" on page 1007.

## Example of Content

Step	Description	Expected Result
1	Open the application.	The application is launched and the login page is displayed.
2	Enter a user name.	The cursor advances to the password field.
3	Enter a password.	The password is displayed as asterisks.
4	Click <b>Submit</b> on the Web page.	The application's main page is displayed.

API components are much like API tests, described in "["API Testing Design with the UFT IDE" on page 388](#). Some exceptions and limitations for working with API components are described in "["Working with API components" on page 1001](#).

- For details about manual testing in BPT, see the *HP Business Process Testing User Guide* in the ALM documentation set.

To learn more, see:

- [Business Component Categories](#) ..... 1004

## Business Component Categories

### Relevant for: business process tests and flows

Because BPT is a component-based testing framework, components are largely responsible for driving the actual application testing. This component framework encourages component design and reuse, so the method you use to categorize your components has a large impact on the ability of your framework to manage your testing abilities successfully.

You can use one or more of the following methods to categorize components .

- "[Logical Components](#)" below
- "[Application Object Components](#)" on the next page
- "[Generic Components](#)" on the next page

### Logical Components

A logical component represents the use of a part of the screen with one or more controls, or a set of API calls which combine to perform some application logic. This category is based on a specific context in the application under test.

#### Examples

- A **Login** component represents the login process, based on a login window that allows you to enter a user name and password, and then click a **Login** button.
- A **Search** component represents search for an entity in the application under test. You can enter

a string for which to search, indicate capitalization and/or whole word options, and click a **Search** button.

## Application Object Components

An application object component might represent an object on the screen or a call to a single API.

This category is usually independent of the context within the application under test, and can be used in many situations. You decide the level of granularity that most encourages reuse.

### Examples

- A **Button** component represents the button object.
- A **Grid** component represents a grid object in a pane or window.
- A **Pane** component represents a pane in a window or screen.
- An **Interrogate** component represents the interrogation of the application under test's backend database.

## Generic Components

A generic component performs actions outside of the context of the application under test. It can be reused in tests of different applications.

### Example

- A **Launch** component represents the launching of a browser.

Flows can be thought of as complex components or small business component tests. Flows comprise a set of components in a fixed sequence to perform a specific task. A flow can be part of a test just like any other component, but when the flow runs, BPT executes the components that the flow contains.

# Keyword GUI Components Overview

### Relevant for: Keyword GUI components only

You can use the Keyword View to create, view, and modify a keyword component in UFT.

In the Keyword View, keyword components are divided into steps in a modular, keyword-driven, table format. Each step is a row that comprises individual parts that you can easily modify. You create and modify steps selecting items and operations and entering additional information, as required.

Each step in a keyword component is automatically documented as you complete it. This enables you to view a description of the step in understandable sentences. In addition, if you added a function library to the application area associated with the keyword component, when you define a step by selecting a

user-defined operation (function), the documentation that you added in the function library is displayed for the step. For details, see "[How to Create and Work with a User-Defined Function](#)" on page 700.

**Note:** ALM users can access components from the ALM Business Components module. For details, see the *HP Business Process Testing User Guide*.

## Keyword GUI Component Resources

- Each keyword component is based on a specific application area, which is stored in the ALM project in which you intend to save the component.
- Each application area specifies the settings and resources for the keyword component, including the location of shared object repositories, function libraries, recovery scenarios, and other information.
- There may be one or more application areas from which to choose. You select the application area that is best suited for your keyword component. For details, see "[Application Areas](#)" on page 1014.

To learn more, see:

- "[Tips and Guidelines for Selecting Application Areas for Your Component](#)" on the next page

## Converting Manual Components

### Relevant for: Scripted and Keyword GUI components only

In UFT, you can open a manual component stored in your ALM project and convert it to an automated scripted or keyword (keyword-driven) component. This conversion process is irreversible, although you can still view and run the manual steps in ALM, if needed).

When you open a manual component, UFT asks you whether you want to convert into a keyword component. If you convert a manual component to an automated component, each manual step from the manual component is converted into a `ManualStep` operation in the Keyword View:

Item	Operation	Value
Operation	ManualStep	"Step 1","Enter the user name and press tab key","cursor moves to user name field"
Operation	ManualStep	"Step 2","Enter password.","Password should be masked with asterisks"
Operation	ManualStep	"Step 3","Click the Enter button.","Login dialog box closes and application continues"

The name, description, and expected result of each manual step are added as argument values for each `ManualStep` operation. Any defined input and output parameters are converted into local parameters.

After you convert a manual component to an automated keyword component, you can still view its manual steps in ALM, and you can run it as a manual component using the ALM Manual Runner. In ALM, you can also modify or add additional steps to an automated component in the Automation tab. These steps are then updated automatically in the Component Steps tab, and then in UFT.

**Note:** You can also convert a manual component to an automated component from within ALM. For details, see the *HP Business Process Testing User Guide*.

You can manually modify step names, step descriptions, and expected results by changing the corresponding argument values in the relevant ManualStep row of the Keyword View, for example:

Item	Operation	Value
Operation	ManualStep	"Step 1","Enter the user name and press tab key","cursor moves to the user name field"
Operation	ManualStep	"Step 2","Enter password.","Password should be masked with asterisks"
Operation	ManualStep	"Step 3","Click the Enter button.","Login dialog box closes and application continues"

All modifications you make in UFT to the components ManualStep operations and regular keyword-driven steps are reflected in the Component Steps tab and Automation tab of the component in ALM and vice versa (after you save the changes). This means that you can update components in either ALM or UFT and still continue to run them manually using the ALM Manual Runner or ALM Sprinter when needed.

**Note:**

- You can also use comments to add information about a step or to separate sections of your component. Each manual step and comment appears as a separate row in the Keyword View.

## Tips and Guidelines for Selecting Application Areas for Your Component

**Relevant for: Keyword GUI components only**

- When you create a GUI component in UFT, you must select the application area to which you want to associate the component. There may be one or more application areas available from which to choose. You should select the one that is best suited for the component.
- If changes are made to your application or to the resource files and settings associated with the application area, the application area may become unsuitable, and you may need to associate a different application area with a specific component. For example, the object repository could have been modified or removed from the application area.
- As your application develops, it may include additional or different objects that are not contained in the currently associated object repository. This could cause the component or business process test to run incorrectly or to fail.
- If another application area contains the required resource files and settings, you should associate that application area with the component.
- Each time you open a component, UFT verifies that the resources specified for the component are available. If a component or application area has resources that cannot be found, such as a missing shared object repository, UFT indicates this in the Errors pane.

## Scripted GUI Components Overview

**Relevant for: Scripted GUI components only**

Scripted components are maintainable, reusable scripts that perform a specific task.

Using scripted components enables you to utilize the full power of both the Keyword View and the Editor, as well as other UFT tools and options to create, view, modify, and debug scripted components. For example, you can:

- Use the Step Generator to guide you through the process of adding methods and functions to your scripted component.
- Use the Editor to enhance the scripted component flow by manually entering standard VBScript statements and other programming statements using test objects and methods.
- Incorporate user-defined functions in your scripted component steps, parameterize selected items, and add checkpoints and output values.

## How to Create and Manage GUI Business Components

### **Relevant for: GUI components only**

This task describes the different operations you can perform to manage business components, and contains general considerations and guidelines.

This task includes:

- ["Prerequisites" below](#)
- ["Guidelines for users of earlier QuickTest versions" on the next page](#)
- ["Create a new business component" on the next page](#)
- ["Open and convert a manual component to an automated keyword component \(keyword GUI components only\)" on page 1010](#)
- ["Save a business component" on page 1010](#)
- ["Associate a different application area with your component" on page 1011](#)
- ["Delete a component" on page 1011](#)

### **Prerequisites**

- You must connect UFT to an ALM project, which is where components and application area resources and settings are stored. Connecting your ALM project enables UFT to create or open the component. This also enables the component to access all of the resources defined in the application area on which the component is based.
- Make sure you have the required ALM permissions before working with components and application areas. For details on setting user group permissions in the Business Components module, see the *HP Business Process Testing User Guide* and the *HP Application Lifecycle Management Administrator Guide*.
- Components that are currently open in ALM or another UFT session are locked and can be opened only in read-only format. To work with these components, they must be closed everywhere else. Additionally, if the resources used by a component are locked, these resource files open in read-only mode.

## Guidelines for users of earlier QuickTest versions

- To modify a component last modified using QuickTest 9.5 or earlier, you must upgrade the component to the QuickTest 11.00 format using the QuickTest Asset Upgrade Tool for ALM (found on your QuickTest 11.00 installation DVD).

After upgrading the version of the component, open the upgraded component in UFT. Before it is upgraded, though, you can view it in read-only format, and you can run it. (To work with a component last modified using QuickTest version 8.x, it must first be upgraded to QuickTest version 9.x.) All components last modified in versions of QuickTest later than 9.5 can be opened in UFT without conversion.

- After you save a converted component, it cannot be used with earlier versions of QuickTest.

## Create a new business component

1. Select one of the following:

- **File > New > Business Component**
- **File > Add > New Business Component**
- **File > Add > Business Component from Sprinter Automated Test Data File**

In the New Business Component Dialog Box, select a folder in the Business Components module in ALM in which to store your component and give your component a name.

2. In the **Application Area** field, click the **Browse** button to select a suitable application area from within the **ALM Test Resources** module. After choosing your application area, click **OK**.
3. If you are creating a business component from a Sprinter automated test data file, specify the location (on the file system) of the test data file.
4. A new business component opens in the Keyword View (for keyword components) or in the Editor (for scripted components).

Although the component does not yet contain content, it does contain all of the required settings and resources that were defined in the application area on which it is based. You can view these settings in read-only format by choosing **File > Settings**. If you later need to change these settings, you can do so in the associated application area.

5. Optional - You can now add steps and comments to your business components.

For details relevant for **keyword components**, see "[Standard Steps in the Keyword View](#)" on page [113](#) and "[Comments in the Keyword View](#)" on page [115](#).

For details, relevant for **scripted components**, see "[How to Enhance Your Actions, Scripted Components, and Function Libraries Using the Windows API](#)" on page [665](#) and "[Programming in GUI Testing Documents in the Editor - Overview](#)" on page [649](#).

## Open and convert a manual component to an automated keyword component (keyword GUI components only)

**Caution:** Converting manual components to automated keyword components cannot be undone.

1. Do one of the following:
  - Select **File > Open > Business Component**. In the Open Dialog box, select a manual component. Manual components are represented by a component icon with an **M** in the left corner of the icon.
  - Add a manual component to a business process test. In the test grid or canvas, select **Automate Component > Scripted/Keyword GUI**.
2. UFT asks whether you want to convert the manual component to a keyword component.
3. Click **Yes** to continue with the conversion (cannot be undone).
4. In the New dialog box, select an application area for your component and click **OK**. As UFT downloads, opens, and converts the component, the operations it performs are displayed in the status bar.

Each manual step from the manual component is converted into a **ManualStep** operation in the Keyword view. You can now work with the component like any other component.

**Note:** If the application area you select does not yet contain all of the required resources and settings, you can still add steps using the **ManualStep** function or the **Comment** option. This enables you to type in manual steps as you would in ALM or in another application, such as Microsoft Excel or Microsoft Word.

## Save a business component

- To save a modified component, select **File > Save**. The component is saved with the changes.
  - To save a new component or to save an existing component with a new name, select **File > Save As**.
- The business component is saved to the component tree in the ALM project (Business Components module).

**Note: (for scripted GUI components):** When working with scripted components, the data sheet name in the Data pane is identical to the scripted component name. Therefore, if you save a scripted component with a new name using **File > Save As**, the data sheet is automatically renamed. If you have a step that references the data sheet by name, the step will fail during the run session because it references the former data sheet name. If you save a scripted component with a new name, you must find any references to the former data sheet name in the Editor and replace them with the new data sheet name.

## Associate a different application area with your component

Do one of the following:

- Select **File > Change Application Area**.
- Right-click on a component node and select **Change Application Area**.

In the Change Application Area Dialog Box, you select a different application area and click **OK** to change the application area associated with a component.

## Delete a component

You delete a component in ALM, regardless of whether it was created in UFT or in ALM. For details, see the *HP Business Process Testing User Guide*.

# How to Convert a Keyword GUI Component to a Scripted GUI Component

### Relevant for: GUI components only

This task describes how to convert a single keyword component to a scripted component.

**Caution:** This replaces the existing keyword component with a scripted component, and cannot be undone.

This task includes the following steps:

- "Prerequisites" below
- "Convert the keyword GUI component to a scripted GUI component" below
- "Results" on the next page

### 1. Prerequisites

- You must connect UFT to an ALM project.
- The keyword component must be editable (meaning it must not read-only mode or locked by another user). For example, if you are working with version control, you must check out the component.

### 2. Convert the keyword GUI component to a scripted GUI component

- a. Open the keyword component you want to convert.
- b. Select **File > Convert to Scripted Component**. When prompted, click **OK** to proceed with the conversion.

### 3. Results

After the conversion is complete, the scripted component opens automatically.

## GUI Component Statuses

### Relevant for: GUI components only

You can assign statuses to business components in UFT or in ALM.

In general, the component with the most severe status determines the status of the entire business process test. For example, a component with an **Error** status causes every business process test of which it is a part to have an **Error** status.

Status	Description
<b>Error</b>	The component contains errors that need to be fixed.
<b>Maintenance</b>	The component is currently being developed and tested and is not yet to run, or it was previously implemented and is not being modified to adapt it for changes that have been made in the application.
<b>Ready</b>	The component is fully implemented and ready to be run.
<b>Under Development</b>	The component is currently under development. This status is automatically assigned to: <ul style="list-style-type: none"><li>• New component created in the Business Components module of ALM with BPT support.</li><li>• Component requests dragged into the component tree in ALM with BPT support.</li></ul>
<b>Not Implemented</b>	The component has been requested in the Test Plan module of ALM. The status changes automatically from <b>Not Implemented</b> to <b>Under Development</b> when the request is moved from the Component Requests folder in the component tree in the Business Components module.

# Troubleshooting and Limitations - Business Components

## **Relevant for: GUI components only**

This section describes troubleshooting and limitations for business components.

- For troubleshooting and limitations for learning objects, and recording and running steps, see "["Learning objects, running steps, and recording steps \(GUI testing only\)" on page 850](#).
- **Start Menu / Quick Launch Panel.** On Windows 7, Windows Server 2008 R2, Window 8 ,or Windows Server 2012, if you begin a recording session after installing UFT (or upgrading from QuickTest) without restarting your computer, UFT cannot record operations on the Windows **Start** menu or **Quick Launch Panel**.

**Workaround:** Restart your computer and start a new recording session.

- **Start menu.** UFT does not record launching Windows Help from the **Start Menu**.
- **Start Menu.** UFT does not record the selection of **Start** menu items that are customized as menus, such as My Computer, Control Panel, or Recent Documents.

**Workaround:** Customize the **Start** menu items as links so that UFT can record operations on them or record the activation of these items another way (and not through the **Start** menu).

- Renaming a UFT component from ALM may cause the test or component to behave unexpectedly.

**Workaround:** To rename a test or component, open it in UFT and rename it using the **Save As** option. If the test or component has already been renamed from ALM, use the rename option again to restore the old name, and then use the **Save As** option in UFT. Renaming a UFT test parameter from UFT will cause any run-time parameter values already set for this parameter in ALM to be lost.

- **Multilingual Support.** Names and paths of components, application areas, and resources (for example, function libraries, object repositories, and recovery scenarios) are not Unicode compliant and therefore should be specified either in English or in the language of the operating system.
- Business component scripts that were created in QTP 11.00, and contain the **ActionName** environment variable, return different values for the **ActionName** variable value when run in UFT.

For example, an **ActionName** variable that returned the BC1 value as the component name in QTP 11 returns the BC1[BC1] value in UFT.

**Workaround:** Update your scripts as needed to use the new environment variable value.

# Chapter 66: Application Areas

**Relevant for: GUI components only**

**Note:** Unless otherwise specified, references to **Application Lifecycle Management** or **ALM** apply to all currently supported versions of ALM and Quality Center. Note that some features and options may not be supported in the specific edition of ALM or Quality Center that you are using.

For details, see the *HP Unified Functional Testing Product Availability Matrix* and the *HP Application Lifecycle Management User Guide* or the *HP Quality Center User Guide*.

This chapter includes:

- [Application Areas - Overview](#) ..... 1015
- [How to Create and Manage Application Areas](#) ..... 1016
- [Application Area User Interface](#) ..... 1018
- [Troubleshooting and Limitations - Application Areas](#) ..... 1020

# Application Areas - Overview

## **Relevant for: GUI components only**

When you create a set of components to test a particular area of your application, you generally need to work with many of the same test objects, keywords, testing preferences, and other testing resources, such as function libraries and recovery scenarios. You define these files and settings in an application area, which provides a single point of maintenance for all elements associated with the testing of a specific part of your application.

An **application area** is a collection of settings and resources that are required to create the content of a business component. Resources may include shared object repositories containing the test objects in the application being tested, function libraries containing user-defined operations that can be performed on that application, and so forth. Components are automatically linked to all of the resources and settings defined in the associated application area.

For example, to be able to select test objects when implementing steps for a component, the test objects must be stored in a shared object repository associated with the component's application area. Associating function libraries with your application area enables you to access the functions defined in these function libraries as keywords.

You can create as many application areas as needed. For example, you may decide to create an application area for each Web page, module, window, or dialog box in your application. Alternatively, for a small application, one application area may be all that is needed.

Each component can have only one associated application area.

## **Application Area Settings and Associated Resources**

Application area settings and any changes you make to these settings are automatically applied to any business component with which the application area is associated.

The resources associated with your application areas are the resources that are available to the components associated with your application area.

If resources are referenced in component steps, and you later modify the resources associated with its application area, your component may not run correctly. For example, if a component uses test objects from the **MyRepository.tsr** shared object repository, and you remove this object repository from the application area, the component will not be able to access the required test objects because the object repository is no longer included in the application area.

# How to Create and Manage Application Areas

## Relevant for: GUI components only

This task includes the following steps:

- "Prerequisites" below
- "Make sure all required resources are stored in ALM" below
- "Create the application area" on the next page
- "Configure application area settings" on the next page

### 1. Prerequisites

- a. Make sure you have permissions in ALM for modifying components, adding steps, modifying steps, and deleting steps. (If you do not have all of these permissions, you can open application areas only in read-only format.) For details on setting permissions for the Business Components module, see the *HP Business Process Testing User Guide*.
- b. Evaluate the purpose of your application area and determine the resources required for the business components that will use it. For example:
  - What test objects will they need?
  - What user-defined functions can you add to ensure that all required operations are available?
  - Which keyword operations do you want to make available for each test object type?

### 2. Make sure all required resources are stored in ALM

To create an application area, you must associate the required function library, shared object repository, and recovery scenario resources with it.

Some of the required resources may already exist.

If necessary, create new resources and store them in the **Test Resources** module in ALM.

- a. Verify the existence of or create required function library resources. For details, see "[How to Create and Manage Function Libraries](#)" on page 693.

**Note:** BPT provides a set of predefined function libraries that you can use. Some of these are associated with all new application areas by default. These files are located in the **Test Resources** module of your ALM project under **Resources/BPT Resources**.

- b. Verify the existence of or create required shared object repository resources. For details, see "[How to Create and Manage Shared Object Repositories](#)" on page 206.
- c. Verify the existence of or create required recovery scenario resources. For details, see "[How to Create and Manage Recovery Scenarios](#)" on page 147.

### 3. Create the application area

- a. Connect to the ALM project in which you want to save the application area.
- b. Do one of the following:
  - Select **File > Add > New Application Area**
  - In the BPT View, click the **Add a New Application Area** button.

The New Application Area dialog box opens, enabling you to specify the location for the new application area within the **ALM Test Resources** module.

After you click **Save**, the new application area is added to the open solution and then opened in the document pane. The application area displays several panes in which you specify the settings and resource files that you want business components associated with the application area to use.

**Note:** If you do not want to add the application area to the open solution, select **File > New > Application Area**. If you do this, the open solution is closed, any open documents included in the solution are closed as well, and the new application area is opened in a new untitled solution.

### 4. Configure application area settings

- a. In the Properties pane, provide a full description of the application area.
- b. In the Associated Add-ins section of the General Properties tab (in the Properties pane), specify associations to the required UFT add-ins.

When a business process test runs, the add-ins associated with the application area for the first component in the test are loaded in UFT. Therefore, if this application area is likely to be associated with components that may be first in a business process test, ensure that you include all add-ins that any of the components in the test may need. However, to ensure expected functionality and optimum performance, do not include add-ins that are not necessary.
- c. If necessary, associate function libraries or shared object repositories. Click the **<application area name>** node in the Solution Explorer and select **Associate Function Library** or **Associate Object Repository**, as needed. These object repositories and function libraries are displayed in the Object Repositories pane and Function Libraries pane of the application area, respectively.
- d. If necessary, associate recovery scenarios and define their settings as described in the Recovery pane (located under the Additional Settings node).

**Note:** In general, you should create all necessary resources for your application area before you begin creating the application area itself. If you realize the need for another resource while you are in the process of creating the application area, it is possible to create new, empty resource files directly from the application area and associate them with your application area (and then create the content for those files at a later time).

- e. In the Keywords Pane, specify which keywords will be available for use when creating component steps.

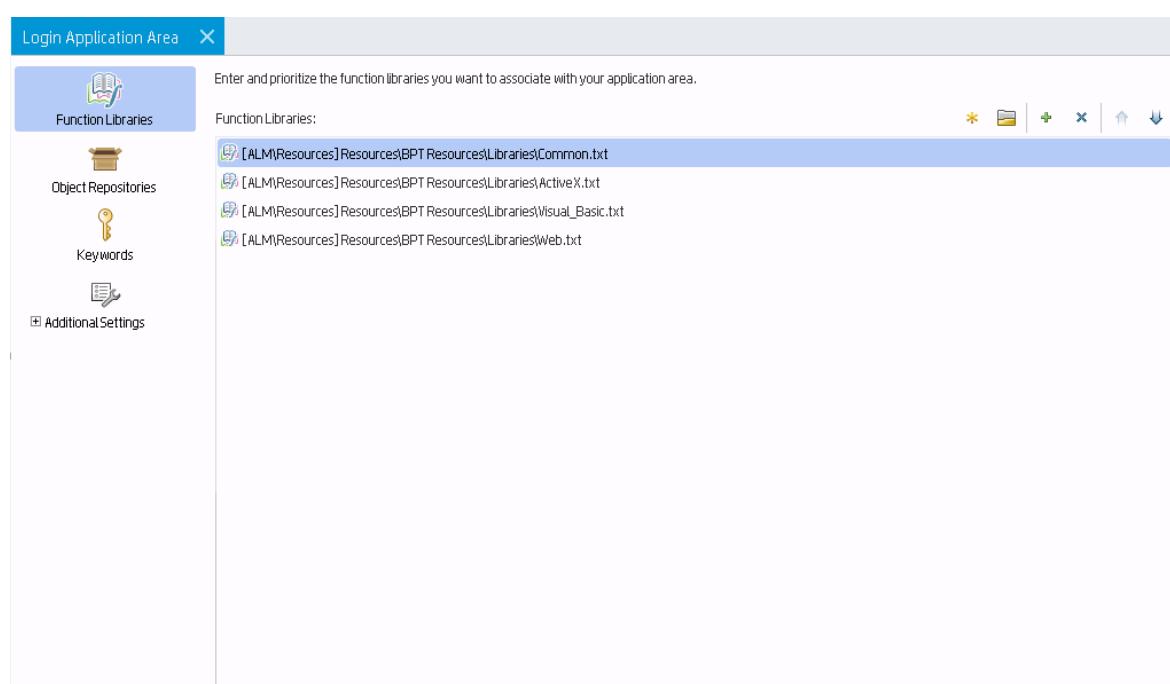
- f. In the Applications pane, specify the Windows-based applications on which components associated with the application area can record and run.
- g. In the Additional Settings, pane, if relevant, define additional setting for your application area in the Additional Settings pane, such as add-in specific settings, log tracking, and local system monitors.
- h. Save your changes .

## Application Area User Interface

### **Relevant for: GUI components only**

In the application area, you define and view application area settings and associate required resource files.

**Note:** Some basic application area settings are defined in the Properties tab in the Properties pane.



<b>To access</b>	<p><b>In UFT:</b></p> <ol style="list-style-type: none"> <li>1. Connect to your ALM project.</li> <li>2. Create a new application area or open an existing one (<b>File &gt; New/Open/Add &gt; Application Area</b>, or double-click the application area in the Solution Explorer).</li> </ol> <p><b>Note:</b> You can also access this dialog box from ALM.</p>
<b>Relevant</b>	<a href="#">"How to Create and Manage Application Areas" on page 1016</a>

tasks	
See also	<ul style="list-style-type: none"><li>• <a href="#">"Application Areas - Overview" on page 1015</a></li></ul>

User interface elements are described below:

UI Elements	Description
	<b>Function Libraries.</b> Displays the Function Libraries pane, which enables you to associate function libraries with your application area and to prioritize them. It also enables you to create and modify associated function libraries.
	<b>Object Repositories.</b> Displays the Object Repositories pane, which enables you to associate shared object repositories with your application area and to prioritize them. It also enables you to create and modify associated object repositories.
	<b>Keywords.</b> Displays the Keywords pane, which enables you to determine which built-in and user-defined keywords (operations) are available when creating components.
	<b>Additional Settings.</b> Displays the Additional Settings pane, which enables you to define settings that affect all of the components associated with this application area.  You can specify the Windows-based applications with which a component can work, recovery scenarios that a component can use, and different tracking and monitoring options.  <b>Note:</b> The Business Components Settings dialog box is very similar to this pane, but displays most of the settings in read-only mode.

# Troubleshooting and Limitations - Application Areas

## **Relevant for: GUI components only**

This section describes troubleshooting and limitations for application areas.

- Renaming a UFT test or component from ALM may cause the test or component to behave unexpectedly.

**Workaround:** To rename a test or component, open it in UFT and rename it using the **Save As** option.

If the test or component has already been renamed from ALM, use the rename option again to restore the old name, and then use the **Save As** option in UFT.

- Renaming a UFT test parameter from UFT will cause any run-time parameter values already set for this parameter in ALM to be lost. For a list of naming conventions, see "["Troubleshooting - Naming Conventions" on page 1144](#)".

# Chapter 67: Creating Business Process Test Steps

**Relevant for: business process tests**

This chapter includes:

- [Creating Steps in a Business Process Test - Overview](#) ..... 1022
- [How to Create Test Steps in a Business Process Test](#) ..... 1023
- [How to Record a Business Process Test](#) ..... 1025
- [How to Add Test Objects to a Component with Capture](#) ..... 1027

# Creating Steps in a Business Process Test - Overview

## Relevant for: business process tests or flows

After you create the structure of your business process test or flow by adding components and/or business process flows to a test, you must then provide the test steps inside each component. These test steps perform user actions on your application and enable you to use a business process test or flow to actually test your application.

In order to create test steps, you must do a number of things:

<b>Create object repositories</b>	<p>In order to create test steps, you must have test objects available in your component. These test objects are contained in object repositories that are then associated with the component (if you record the test steps) or the component's application area. Once the object repository is associated with the component, you can add the test objects to the component to create test steps of your application.</p> <p>You can add test objects to an object repository in the following ways:</p> <ul style="list-style-type: none"><li>• <b>To the component's local object repository:</b> Using the Capture toolbar, you can scan all the objects in your application or in a specified area of your application. After capturing the objects, UFT saves these objects to the component's local object repository and the test objects are available for use in the associated component.</li></ul> <p>For details on how to capture test objects from your application, see "<a href="#">How to Add Test Objects to a Component with Capture</a>" on page <a href="#">1027</a>.</p> <ul style="list-style-type: none"><li>• <b>To a shared object repository:</b> Using the Object Repository Manager, you can learn the objects in your application. These objects are then saved in a shared object repository, which is associated with your component's application area. The shared object repositories can be associated with multiple application areas and multiple components.</li></ul> <p>For more detail on creating shared object repositories, see "<a href="#">Test Objects in Object Repositories</a>" on page <a href="#">178</a>.</p>
<b>Add object repositories to your application areas</b>	<p><b>Note:</b> If you are creating your test by recording user actions, you do not need to add the object repositories to an application area.</p> <p>In order for the individual components to access the test objects, the object repositories containing the objects must be included in an application area which is then associated with a specific component or components. Once you associate the object repository with a application area and the application area with a component, you can see the test objects in the Toolbox pane (when the component is selected) and use the components in test steps.</p> <p>For details on how to associate an object repository with your application area, see "<a href="#">Add object repositories to an application area</a>" on page <a href="#">1024</a>.</p>
<b>Add test steps to the component</b>	<p>After you have the test objects available to use in your component's steps, you can add the test steps to the component:</p> <ul style="list-style-type: none"><li>• <b>By recording:</b> You can use UFT's recording functionality to record user actions on the application. UFT then turns these actions into test steps, including the test object on which the action was performed, the action (method), and any parameters of the action (such as the location on the screen where the action was performed).</li><li>• <b>Manually:</b> You can manually add test objects to the component by dragging them from the Toolbox pane,</li></ul>

	selecting them from the list of available objects in the Keyword View, or adding statements in the Editor.
<b>Enhance your component's test steps</b>	<p>In addition to adding simple test steps that you perform on your application's objects, you can add additional types of steps that extend the test:</p> <ul style="list-style-type: none"> <li>• <b>Checkpoints:</b> Checkpoint steps check the state of your application or its objects in the course of the test run. For additional details on checkpoints, see "<a href="#">Checkpoints in GUI Testing</a>" on page 289.</li> <li>• <b>Functions:</b> Function steps enable you to run a custom process within the test progress. For additional details on functions, see "<a href="#">User-Defined Functions and Function Libraries</a>" on page 683.</li> <li>• <b>Output values:</b> Output values enable you to output a property from a test object in order to use this value in other steps. For additional details on output values, see "<a href="#">Output Values in GUI Testing</a>" on page 324.</li> </ul>

For task details on how to add steps to your business process test, see "[How to Create Test Steps in a Business Process Test](#)" below.

## How to Create Test Steps in a Business Process Test

### Relevant for: business process tests or business process flows

This task describes create steps in the components included in your business process test or business process flow.

**Note:** This task is part of a higher-level task. For details, see "[How to Create, Maintain, and Run Business Process Testing Tests and Flows in UFT](#)" on page 981.

This task includes the following steps:

- "[Prerequisites](#) " below
- "[Create shared object repositories containing test objects](#)" below
- "[Add test objects to the component's local object repository by Capture](#)" on the next page
- "[Add object repositories to an application area](#)" on the next page
- "[Manually add steps to your component](#)" on the next page
- "[Add steps to your component by recording](#)" on the next page

### Prerequisites

- **If you are recording the steps in your components:** Create a business process test or business process flow
- **If you are manually creating the steps in your component:** Create or open a business process test or business process flows and add components to the test or flow.

### [Create shared object repositories containing test objects](#)

In order to create steps of objects in your application, you must create test objects in UFT to use in the test steps and store these objects in an object repository.

For details on creating shared object repositories containing these test objects, see "[Test Objects in Object Repositories](#)" on page 178.

## Add test objects to the component's local object repository by Capture

For details, see "[How to Add Test Objects to a Component with Capture](#)" on page 1027.

## Add object repositories to an application area

In order to use the test objects you create in the actual components, they must be added to the application area for the component. This enables the component to access any of the test objects contained in the object repository.

1. In the application area, select the **Object Repositories** tab.
2. In the object repositories list, click the **Add** button  A new row is added to the list of object repositories.
3. In the new row, click the **Browse** button.
4. In the Open Shared Object Repository dialog box, navigate to where you saved the object repository in your ALM project.
5. Click **Open**. The object repository is now added to the list of shared object repositories for the application area and the component.

You can also view the test object included in the object repository in the Toolbox pane when the component is selected in the document pane.

## Manually add steps to your component

After you create components, you must add steps to them to run the actual test of your application.

To create steps, do the following:

**Note:** Before editing a specific component included in your business process test or flow, you must add it to the solution. To add the component to the solution, in the test grid view, right-click the necessary component and select **Go to component/flow** or click the **Go to component/flow** button  on the toolbar.

1. Open the component tab in the document pane.
2. In the document pane, click in the Item column and select the relevant object or select **Object from repository**.
3. Select the necessary method (action) to perform on the object.
4. Add the necessary parameters to use when performing the method (action) on the object.

## Add steps to your component by recording

For details, see "[How to Record a Business Process Test](#)" on the next page

# How to Record a Business Process Test

## Relevant for: business process tests or flows

This task describes how to record a business process test. Recording enables you to create component steps or even a full business process test in your application without the need to manually create separate components and their associated application areas before starting to create steps. When recording, you perform user actions, create additional components as necessary, and your full business process test is created.

**Note:** This task is part of a higher-level task. For details, see "[How to Create Test Steps in a Business Process Test](#)" on page 1023.

This task includes the following steps:

1. ["Prerequisite" below](#)
2. ["Set recording options for the test" below](#)
3. ["Start the test record" on the next page](#)
4. ["Perform steps on your application" on the next page](#)
5. ["Add additional components to the test \(optional\)" on the next page](#)
6. ["Stop recording" on the next page](#)

### 1. Prerequisite

Create a business process test or business process flow or open an empty business process test/flow.

**Note:** You cannot record steps into an existing test.

### 2. Set recording options for the test

- a. In the **Recording Settings** pane of the Options dialog box (**Tools > Options > BPT Testing tab > Recording Options** node), select or clear the following settings as needed:

<b>Automatically parameterize steps using Business Component Parameters</b>	Instruct UFT to automatically parameterize the test steps by creating and using parameters for the steps recorded with the component  <b>Note:</b> This option adds a parameter to every object for which it is possible to parameterize. This may result in some unnecessary parameters for each component.
<b>Automatically check in newly recorded components when creating a BPT test</b>	If you are working with a version-controlled ALM project, this instructs UFT to prompt you to check in all new components after recording.

<b>Automatically create snapshot for newly recorded components when recording a BPT test</b>	Instructs UFT to take a snapshot of the application window when recording a BPT test.
----------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------

### 3. Start the test record

a. Do one of the following:

- In the UFT toolbar, press the **Record** button .
- In the BPT View, click the **Record a New Business Process Test or Flow** button.

b. In the New Business Component dialog box, provide the **Name**, **Component**, and default **Application Area** for your component.

c. Click **Create**. UFT is minimized and the Record toolbar is displayed.

### 4. Perform steps on your application

Perform user actions on your application. As you perform the steps, UFT lists the number of user actions performed in the Record Toolbar next to the name of the test being recorded.

### 5. Add additional components to the test (optional)

While recording, you can create additional components and record test steps into these other components. This enables you to create modular components for each area of your application instead of having all the application steps in one single component:

- a. While recording, in the Record toolbar, click the **New Business Component** button .
- b. In the New Business Component dialog box, provide a name for the component. (The component is saved in the same location as you entered when beginning the recording session.)

Subsequent steps are recorded in the created component.

### 6. Stop recording

a. After you have finished recording all your steps, in the Record Toolbar, press the **Stop** button .

UFT loads and creates the components, displaying them in the test canvas or grid. Each component also contains a snapshot of the application during the recording session.

If you selected the **option to automatically parameterize objects** in the **Recording Settings** pane of the Options dialog box, the parameters are displayed in the Data pane and in the Parameters tab of the Properties pane.

b. If necessary, when prompted, in the Check In dialog box, check in the new components to your version controlled ALM project.

# How to Add Test Objects to a Component with Capture

## Relevant for: business process tests or flows

This task describes how to add test objects to a component by scanning the application and capturing the application's objects into an object repository. This enables you to add the test objects in one step without having to first create and populate an object repository and then associate that object repository to the component's application area.

**Note:** This task is part of a higher-level task. For details, "How to Create Test Steps in a Business Process Test" on page 1023.

This task includes the following steps:

1. ["Prerequisites" below](#)
2. ["Set Capture options" below](#)
3. ["Capture the test objects in an application window or page" on the next page](#)
4. ["Capture the test objects in a selected area of your application" on the next page](#)
5. ["Open the object repository for editing" on page 1029](#)
6. ["Export the local object repository to a shared object repository \(optional\)" on page 1029](#)

## 1. Prerequisites

- Ensure that you have a business component open.
- Ensure that the component to which you want to add the test objects is added to the solution.  
To add the component to the open solution, do one of the following:
  - In the BPT test or flow grid view, right-click the selected component and select **Go to Component/Flow** or click the **Go to Component/Flow** button  in the toolbar.
  - Select **File > Add > Existing Business Component** and navigate to the selected component in your ALM project.

The component is opened in the document pane and a node for the component is added to the solution in the Solution Explorer pane.

## 2. Set Capture options

In the BPT General options pane of the Options dialog (**Tools > Options > BPT Testing tab > General node**), select the **Automatically open Object Repository after Capture** option if you want to edit the test object information after the object capture.

When the object repository opens after the application capture, you can modify the object names and properties as needed to suit your needs.

### 3. Capture the test objects in an application window or page

- a. In the Solution Explorer pane, select the component in which you want to include the captured objects.
- b. In the toolbar, click the **Capture** button . The Capture toolbar opens.
- c. (Optional) In the Capture toolbar, press the **Define Object Filter** button .
- d. (Optional) In the Define Object Filter dialog box, select the level of detail to capture:

<b>Selected object only</b>	Captures the selected object's properties and values, without its descendant objects.
<b>Default object types</b>	Captures the selected objects' properties and values, with the properties and values of its descendant objects according to the object types specified by the default filter. You can see which objects are in the default filter by selecting <b>Selected object types</b> , clicking the <b>Select</b> button, and then clicking the <b>Default</b> button.
<b>All object types</b>	Captures all of the application page or window's objects, including the object properties and values, together with the properties and values of all of its descendant objects.
<b>Selected object types</b>	Adds to the object repository the previously selected object's properties and values, as well as the properties and values of its descendant objects according to the object types and classes you specify in the object filter. You specify the objects and classes in the filter by clicking the <b>Select</b> button and selecting the required items.

- e. Select the top level object for which you want to capture its test objects (i.e. a Browser window, or an application screen).
- f. In the Capture toolbar, click the **Capture** button.

The application flickers and the Adding Objects message box is displayed as UFT captures representations of the objects on the page to the local object repository.

- g. Click the Close button.

If you selected the option to automatically open the Object Repository after an application capture, the local object repository opens, displaying the captured objects. These test objects are stored in the component's local object repository and are available to use in the component test steps.

### 4. Capture the test objects in a selected area of your application

- a. In the Solution Explorer pane, select the component in which you want to include the captured objects.
- b. In the toolbar, click the **Capture** button . The Capture toolbar opens.
- c. (Optional) In the Capture toolbar, press the **Define Object Filter** button .
- d. (Optional) In the Define Object Filter dialog box, select the level of detail to capture:

<b>Selected object only</b>	Captures the selected object's properties and values, without its descendant objects.
<b>Default object types</b>	Captures the selected objects' properties and values, with the properties and values of its descendant objects according to the object types specified by the default filter. You can see which objects are in the default filter by selecting <b>Selected object types</b> , clicking the <b>Select</b> button, and then clicking the <b>Default</b> button.
<b>All object types</b>	Captures all of the application page or window's objects, including the object properties and values, together with the properties and values of all of its descendant objects.
<b>Selected object types</b>	Adds to the object repository the previously selected object's properties and values, as well as the properties and values of its descendant objects according to the object types and classes you specify in the object filter. You specify the objects and classes in the filter by clicking the <b>Select</b> button and selecting the required items.

- e. Select the top level object for which you want to capture its test objects (i.e. a Browser window, or an application screen).
- f. In the Capture toolbar, press the **Capture Area** button. The mouse pointer turns into a crosshairs.
- g. In your application, draw a box around the area to capture.

The application flickers and the Adding Objects message box is displayed as UFT captures representations of the objects on the page to the local object repository.

- h. Click the Close button.

If you selected the option to automatically open the Object Repository after an application capture, the local object repository opens, displaying the captured objects. These test objects are stored in the component's local object repository and are available to use in the component test steps.

## 5. **Open the object repository for editing**

If you did not select the option to automatically open the object repository after the application capture, you can still edit the object repository after capturing the objects.

- a. In the Solution Explorer, expand the node of your component.
- b. Under the component node, double-click the **Local** node. The Object Repository window opens.
- c. In the Object Repository, edit the names and properties of the objects as needed. For full details on how to edit object repositories, see, "[Test Objects in Object Repositories](#)" on page [178](#).

## 6. **Export the local object repository to a shared object repository (optional)**

You may sometimes want to export the captured objects in the local object repository to a shared object repository which can be used by other components.

- a. In the Object Repository window, select **File > Export Local Objects**.
- b. In the Save dialog, select the location in which to save the exported object repository. (This object repository is automatically saved as a shared object repository with a .tsr extension.)
- c. Click **Create**. The local object repository is now saved as a shared object repository which can be added to other components and application areas.

# Chapter 68: Using Data in Business Process Testing

**Relevant for: business process testing**

This chapter includes:

- Using Data in Business Process Tests - Overview ..... 1032
- Business Process Testing Parameter Categories ..... 1033
- Linking Parameters in Business Process Tests ..... 1034
  - Linking Parameters When Running Iterations in a Business Process Test ..... 1035
  - Linking Parameters in a Business Process Test - Examples ..... 1036
- Promoting Parameters in a Business Process Test ..... 1038
- How to Use Data in a Business Process Test ..... 1039
- Troubleshooting and Limitations - Using Data with Business Process Testing in UFT ..... 1044

# Using Data in Business Process Tests - Overview

## Relevant for: business process testing

You can affect the behavior and results of a business process test by using parameters to define the values that components and flows receive and return. This process is known as **parameterization**.

Parameterization works at a number of different levels in a business process test:

- You create test parameters which can pass a parameter value to any component included in the test
- You create component parameters which can pass a parameter value to any test step contained in the component
- You can pass component parameters between components (by passing them first to a test parameter)
- You can export all the component parameter values to an Excel spreadsheet, set the values in the spreadsheet, and then import the spreadsheet for each test run

Parameterization enables you to perform operations on the application you are testing with multiple sets of data. Each time you run a business process test, you can supply different values for the parameters in the test (or its components and flows).

**Note:** When creating or modifying parameter for an individual component, make sure that you create or edit the parameters inside the individual component or inside a function in an associated function library.

Business Process Testing provides several types of parameters, such as:

<b>Test parameters</b>	These parameters are created at the test level. The parameter and value is available to all flows and components contained in the test (provided they are linked).  Test parameters are exclusive for the test only, and cannot be used for flows or components not contained in the test.
<b>Flow parameters</b>	These parameters are created at the business process flow level. Like a test parameter, these parameters are available to all components contained in the flow (provided they are linked).  Flow parameters are exclusive for the test only, and cannot be used for components not contained in the flow.
<b>Component parameters</b>	These parameters are created in the individual business component. The parameter and value is available to all test steps contained in the component.  Component parameters can be used in any test that contains the component (provided that you promote the parameter to the test level in a specific test).

## Example

To test the business process of a banker logging into an online banking application, you might structure a business process test from components that:

- Log into the application (Login)
- Select a customer loan (SelectLoan)
- View transactions for the loan (ViewLoan)
- Log out (Logout)

You can set up the steps in each of these business components to receive data from the business process test that runs the components (for example, the loans that a customer has). You can also parameterize any element of data, which may have different values each time the business component is run. For example, the banker may choose a different customer and customer loan to view each time he or she logs in.

Here are parameters that you might create for this scenario, listed by category:

Category	Parameters
<b>Input parameters for a component</b>	<ul style="list-style-type: none"> <li>• LoginName, entered as input by the banker when logging in</li> <li>• AccountNo, entered by the banker, perhaps from a written inquiry.</li> </ul> <p>These parameters are created as input parameters for an individual component, and then you can use them for any step inside the component.</p>
<b>Output parameters for a component</b>	<ul style="list-style-type: none"> <li>• SessionNo, a number for the login session, outputted by the business component when the banker successfully logs in</li> <li>• SelectedAccountNo, outputted by the business component after the banker selects a loan from a list</li> </ul>
<b>Test Parameters</b>	CustomerLoans, a comma-delimited list of all loans for a particular customer, accessed from the test level.

For task details, see "[How to Use Data in a Business Process Test](#)" on page 1039.

## Business Process Testing Parameter Categories

### Relevant for: business process testing

Within a BPT test, you can have different types of parameters:

Parameter Categories	Parameter and Description
<b>Input / Output</b>	<p><b>Input parameters</b> enable you to define data used by a component or flow that is provided from an external source. When creating components, tests, and flows, you define how the values are supplied for input parameters.</p> <p>An input parameter can receive:</p> <ul style="list-style-type: none"> <li>• A predefined default value, if no other value is supplied by the test or flow.</li> <li>• An output parameter value returned by a component or flow earlier in the flow or test.</li> <li>• A parameter value that is supplied at the test or flow level, when the test or flow is run.</li> </ul>

	<p><b>Output parameters</b> allow data values retrieved from a component step or flow (the source) to be passed as input parameters to a subsequent component or flow (the target) in the test run.</p> <p><b>Note:</b> You cannot set a default value for an output parameter.</p> <p>You can link input and output parameters to make data available between components or flows within the same business process test.</p>
<b>Component, Flow, and Test</b>	<p><b>Component parameters</b> are parameters defined with a component, and available for use for all steps included in the component. You can have both component input parameters and component output parameters.</p> <p>These parameters are available to:</p> <ul style="list-style-type: none"> <li>• All subsequent steps in the same component</li> <li>• Subsequent components in a flow or test, provided that: <ul style="list-style-type: none"> <li>• The component parameter is defined as an output parameter in the current test or flow, and as an input parameter in the subsequent component in the current test or flow.</li> <li>• The output parameter in the current test or flow is linked to the input parameter of the subsequent component in the current test or flow.</li> </ul> </li> </ul> <p><b>Note:</b> When creating or modifying parameter for an individual component, make sure that you create or edit the parameters inside the individual component or inside a function in an associated function library.</p> <p><b>Flow parameters</b> are parameters defined within a flow. These parameters are available to all components included in a flow. Like component parameters, flow parameters can be input or output parameters.</p> <p><b>Test parameters</b> are parameters identified within a business process test. These parameters are available to all components and flows in the test. Test parameters can only be input parameters.</p>
<b>Local parameters</b>	<p><b>Local parameter</b> values are defined within an individual business component and you can access them only in the component in which they were created. Local parameters are intended for use in a single step or between component steps, for example, as an output value for one step and input parameter for a later step.</p> <p>This type of parameter is used when working with keyword GUI components.</p>

## Linking Parameters in Business Process Tests

### Relevant for: business process testing

Parameter linkage enables you to pass data between business components and flows. Thus, if you have a value that you need for another component in one business component, you can pass the value between the components by linking their parameters. This tests the ability of your application to pass a value from one API to another in the course of the application's work.

You can also pass a test or flow parameter to an individual component parameter.

Once a parameter is linked, it is available to any step contained in the component with the parameter.

### Example

A business component,(named CreateLoan) has an output parameter that contains a generated loan ID. A subsequent business component, (named SearchLoan), can verify the loan if it has access to the CreateLoan loan ID value. This access is provided by linking the CreateLoan component output parameter to SearchLoan component input parameter.

The component or flow in which the output parameter is defined is the source. The component or flow that links to that output parameter is the target. In the example above, CreateLoan is the source component and SearchLoan is the target component.

For task details, see "[Link parameters](#)" on page 1041.

To learn more, see:

- [Linking Parameters When Running Iterations in a Business Process Test](#) ..... 1035
- [Linking Parameters in a Business Process Test - Examples](#) ..... 1036

## Linking Parameters When Running Iterations in a Business Process Test

### **Relevant for: business process testing**

As part of data driving a business process test, you can set components (or groups of components) to run multiple times in different iterations. For details on iterations, see "[Running Iterations in a Business Process Test](#)" on page 1046.

Iterations of a source component can result in multiple parameter output values. In these cases, the value provided by each iteration is passed as input to the corresponding iteration of the target.

Linking can occur successfully only when UFT can determine the target iteration for each source iteration. One of the following conditions must exist:

- **Condition 1.** The source has one iteration and the target has one or more iterations (a "1-to-n"relationship). For an example, see "[Linking Parameters when using Iterations \("1-to-n" Relationship\)](#)" on page 1037.
- **Condition 2.** The source and the target have the same number of iterations (an "n-to-n" relationship). For an example, see "[Linking Parameters when using Iterations \("n-to-n" Relationship\)](#)" on page 1038.

**Note:** When a source or target is a member of a group, the number of iterations is that of the group.

If the component iterations are not represented by a "1-to-n" or "n-to-n" relationship, a warning message is displayed.

## Considerations

- When you use the output parameter of a previous component as the value for an input parameter of a subsequent component, the link between the output and input parameter applies to all component iterations of the input parameter.
- When iterations of a source component in a business process test result in multiple output parameter values, the value that is provided by a given iteration run is passed as input to the corresponding iteration of the target component.
- Moving a business component, group, or flow can cause a parameter reference conflict, such as when a target component is moved to a position preceding the source component. If the resulting warning message is ignored, the conflicting link to the source parameter is deleted. This causes the iteration to fail, and you must redo the link between the parameters.
- Iteration errors cause a business process test or flow that contains the relevant components to fail. These errors are reported as part of the test results.

## Linking Parameters in a Business Process Test - Examples

### **Relevant for: business process testing**

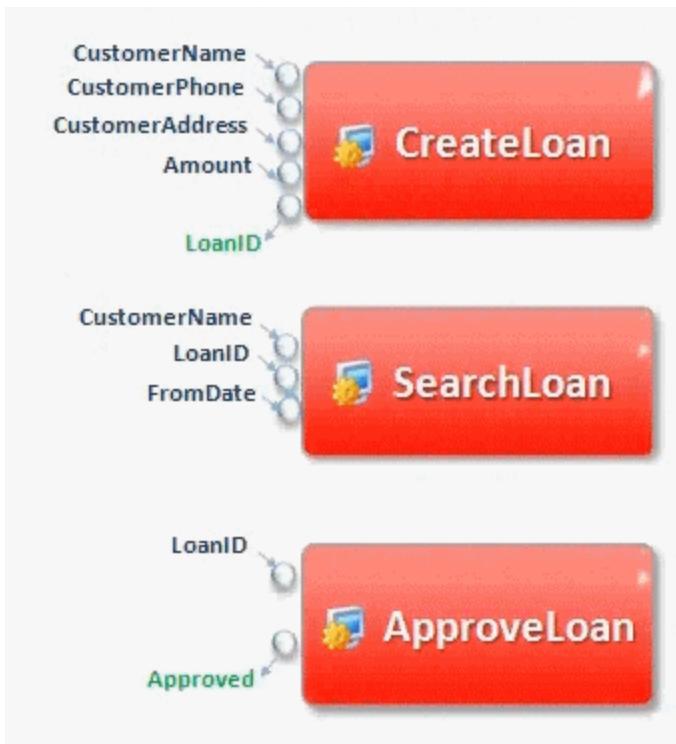
When linking component parameters as part of a business process test, there are a number of possible scenarios:

This topic includes the following examples:

- "[Linking Input and Output Parameters](#)" on the next page
- "[Linking Parameters when using Iterations \("1-to-n" Relationship\)](#)" on the next page
- "[Linking Parameters when using Iterations \("n-to-n" Relationship\)](#)" on page 1038

For these examples, you create components corresponding to the different processes of your application for processing a customer loan request:

- A component that tests the application's ability to receive the request, and generate a unique loan ID for the request (called CreateLoan). This
- A component that searches the existing loans to verify if the loan already exists (called SearchLoan).
- A component that tests the loan request approval process (called ApproveLoan).



Each of the components has a number of input parameters (in black), and output parameters (in green).

### Linking Input and Output Parameters

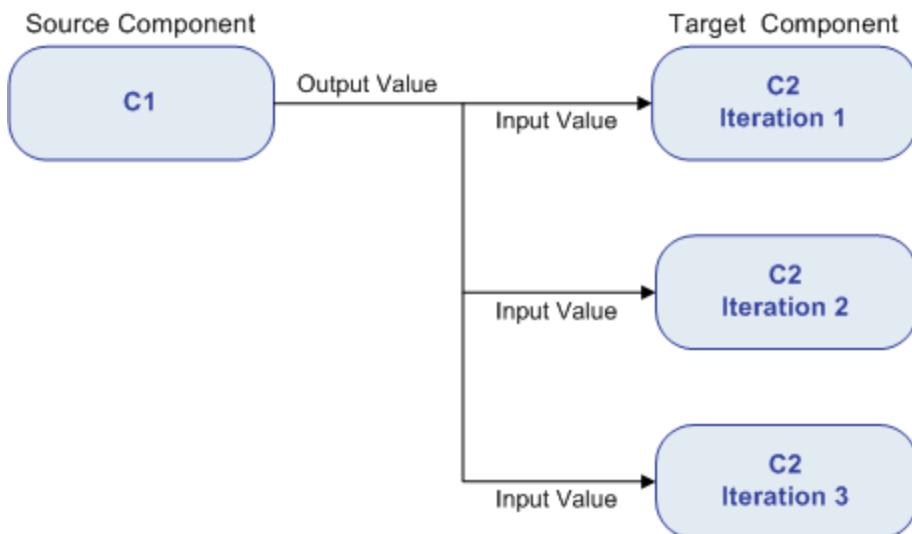
When you create a business process test, you arrange the components in order to test the entire loan approval process workflow from receiving the request through approving the request.

In this example, the value of the LoanID output parameter is passed from the CreateLoan component to the SearchLoan component as the value of the LoanID input parameter for the component. The value is also passed to the ApproveLoan component as the value for the LoanID input parameter.

### Linking Parameters when using Iterations ("1-to-n" Relationship)

In this instance (using the example above), the CreateLoan component (containing the LoanID output parameter) has one iteration, and the SearchLoan and ApproveLoan components have one or more iterations. This is called a "**1-to-n**" relationship.

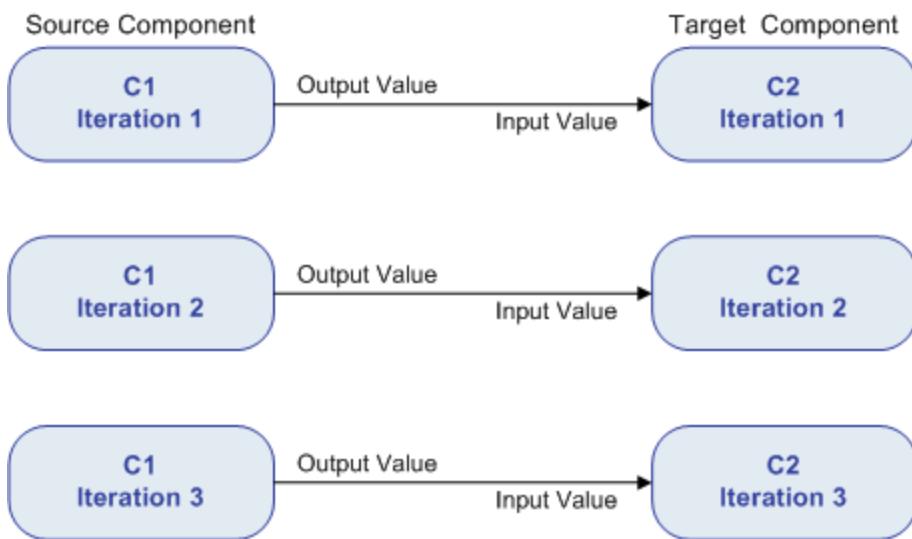
In this scenario, the value of the LoanID output parameter is used for each iteration of the SearchLoan or ApproveLoan component:



### Linking Parameters when using Iterations ("n-to-n" Relationship)

In this scenario, the CreateLoan component (containing the LoanID output parameter) has the same number of iterations as the SearchLoan and ApproveLoan components. This is called an "**n-to-n**" relationship.

For this scenario, the different values for the LoanID output parameter in each of the iterations are used in the respective iterations of the SearchLoan or ApproveLoan components:



## Promoting Parameters in a Business Process Test

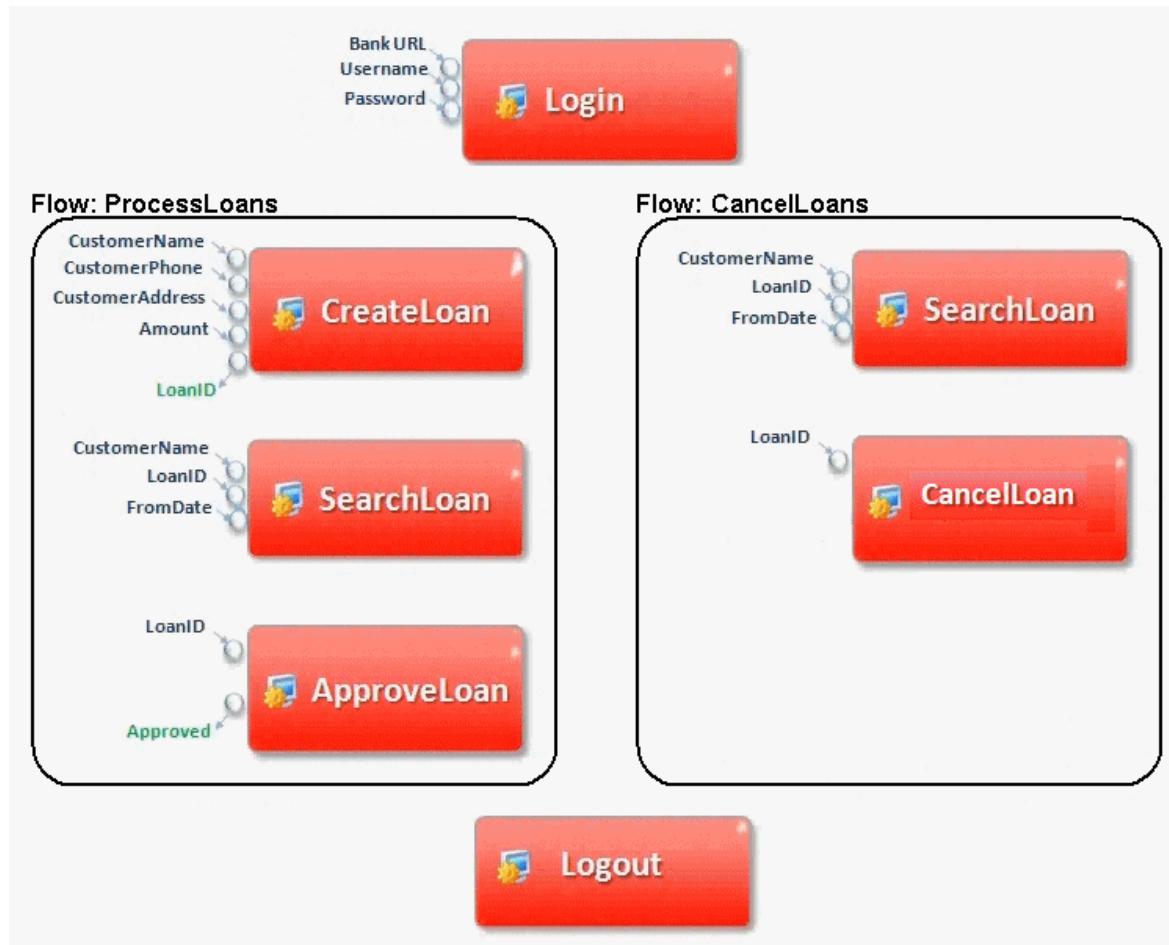
### Relevant for: business process testing

When editing a business process test, you can promote component parameters to the flow or test level at the same time as you add a component to a flow or test. Similarly, you can promote flow parameters

to the test level at the same time as you add a flow to a test. This enables you to use a parameter for all components or flows included in a given flow or test.

### Example

You create a business process test with components named CreateLoan, VerifyLoan, and ApproveLoan. Each of these three components contains a parameter called LoanID:



Using parameter promotion, you can promote the LoanID parameter to the ProcessLoans flow level, thereby making it available as a parameter for the CreateLoan, SearchLoan, and ApproveLoan components. Likewise, if you need to make the LoanID parameter available to the CancelLoans flow, you can promote it to the test level.

For task information, see "[Promote parameters](#)" on page 1041.

## How to Use Data in a Business Process Test

**Relevant for: business process testing**

This task provides general information for how to work with parameters, and iterations in business process tests.

This task is part of a higher-level task. For details, see "[How to Create, Maintain, and Run Business Process Testing Tests and Flows in UFT](#)" on page 981.

This task includes the following steps:

- "Design data" below
- "Set default parameter behavior" below
- "Create parameters and set default values" below
- "Use component parameters in component steps" on the next page
- "Link parameters" on the next page
- "Promote parameters" on the next page
- "Add iterations for a component or flow" on page 1042
- "Set data values for the parameters for each iteration" on page 1042
- "Export component parameters to an Excel" on page 1043
- "Import parameter iteration values from an Excel" on page 1043

## Design data

Consider the following before using parameters:

- Determine which parameters are dependent on other parameters so that you can link them.
- Determine which parameters should be available at the component, flow, and test levels.
- You can also use different use-case scenarios by creating iterations for a given component, flow, or test and providing values for the parameter in each iteration. Design how many times each component, flow, and business process test should run, and with what values.

## Set default parameter behavior

In the General pane of the Options dialog box (Tools > Options > BPT Testing tab > General node), select one of the following options in the **Auto-parameterization level** section:

- **Parameterize user input only:** UFT will parameterize only those objects where a user performs an action (such as text edits, etc.)
- **Parameterize all steps:** UFT will parameterize all objects in a given window of an application

## Create parameters and set default values

1. **Prerequisite:** Ensure that you add the component or flow to your open solution and check it out of the version control database (if your components/flows are saved in a version-controlled ALM project.)
2. In the Solution Explorer, select the test, flow, or component to which you want to add a parameter.

3. In the Properties pane, open the **Test Parameters or Parameters** tab 
4. In the Test Parameters/Parameters, tab, click the **Add** button and specify the type of parameter to add (either input or output).
5. In the Add Input Parameter/Add Output Parameter dialog box, enter the parameter details. You can enter a default value or leave the field blank.

If you enter a default value, you can use the default value in the event a value is not supplied for the test run, or you can use the default value as an example for the type of value that can be provided (for example, a phone number example could be ####-##-####).

**Note:** After you add a component parameter, this parameter is available to use in any step contained in a component. When creating or modifying parameter for an individual component, make sure that you create or edit the parameters inside the individual component or inside a function in an associated function library.

## Use component parameters in component steps

After you create the component parameter, it is available for every step contained in the component.

For details on linking the parameters, see "[How to Parameterize Values for Operations or Local Objects](#)" on page [340](#).

## Link parameters

1. Prerequisite: Ensure that you add the component or flow to your open solution and check it out of the version control database (if your components/flows are saved in a version-controlled ALM project.)
2. In the document pane, select a business process test or flow.
3. In the business process test or flow tab, select a flow or component.
4. In the Properties pane, open the **Test Parameters or Parameters** tab.
5. Select the parameter you want to link.
6. In the Value cell of the parameter, click the **Link** button. The Select Link Source dialog box opens.
7. In the left pane of the Select Link Source dialog box, select the test or flow from which you want to use a parameter.
8. In the right pane of the Select link Source dialog box, select the parameter to which to link.

The parameter name is displayed with a link icon  in the value column indicating the link.

**Tip:** You can also automatically link component parameters if they share the same name as a test or flow parameter. In the BPT Testing tab of the Options dialog box (**Tools > Options > BPT Testing** tab), select the **Always link to existing test parameters** option.

## Promote parameters

Do one of the following:

<b>In the Options dialog box</b>	<ol style="list-style-type: none"> <li>1. Open the BPT Testing tab (<b>Tools &gt; Options &gt; BPT Testing</b> tab).</li> <li>2. In the <b>BPT Testing</b> tab, select the <b>Promote component parameters</b> option. Any parameters in components or flows added after this option is selected are promoted to test or flow parameters.</li> </ol> <p><b>Tip:</b> If you do not want to automatically promote parameters for a selected component or flow, clear this option before adding the flow or component.</p> <ol style="list-style-type: none"> <li>3. If you do not automatically promote parameters using the options in the BPT Testing pane of the Options dialog box, you can click the <b>Promote Parameters</b> button  in the Parameters tab of the Properties pane.</li> </ol> <p>The selected parameter is promoted to the next level and the parameter value is linked to the value of the newly created parameter.</p>
<b>In the Properties pane</b>	<p>In the Parameters tab, click the <b>Promote Parameters</b> button .</p> <p>The selected parameter is promoted to the next level and the parameter value is linked to the value of the newly created parameter.</p>
<b>In the Toolbox pane</b>	<p>In the title node, click the down arrow and then click the <b>Promote parameters to test level</b> button .</p>

## Add iterations for a component or flow

Before you can set parameter values for each iteration of a component or flow, you must add iterations to the test for each component you want to run in iterations:

1. In the document pane, select the component or component group for which to add iterations.
2. If necessary, open the Data pane.
3. In the Data pane, click the **Add** button  and select one of the following to add an iteration:

<b>Add New Iteration:</b>	Adds a new iteration without any values for the component/group parameters.
<b>Copy Iteration.</b>	Copies the values for the component/group parameters from the previously entered iteration.
<b>Create Iteration with Default Values</b>	Adds a new iteration with the default values entered for the parameter in the Test Parameters/Parameters tab in the Properties pane.

The new iteration and parameter values (if selected) are added as an additional row in the Data pane.

## Set data values for the parameters for each iteration

1. In the document pane, select a flow, component, or group.
2. If necessary, open the Data pane.
3. In the Data pane, click in the **Value** cell for the parameter name. All parameters for the selected flow, component, or group are displayed in separate columns with the parameter name as column

headers.

**Note:** Make sure that you have also selected the correct row for the parameter - each separate row is its own iteration.

4. Enter the parameter value for the selected parameter.

When the test runs, the entered parameter value is used for each iteration.

## Export component parameters to an Excel

When your business process test, flow, or components contain component parameters, you can export these parameters to an Excel file. This enables many users to set values for these parameters in and run their tests with the modified data.

1. In the document pane OR the Solution Explorer, select the business process test or flow for which you want to export the component iteration values.
2. In the toolbar, select the **Export test iteration values into Excel Document** button .
3. In the Save dialog box, specify a location in which to save the modified Excel.
4. Click Save. UFT confirms the successful export of the Excel file.
5. Edit the parameter values as needed. Each component containing parameters has its own Excel sheet.

**Note:** If a component does not contain parameters, the Excel does not contain a sheet for that component.

## Import parameter iteration values from an Excel

If you [previously exported your parameter iteration values to Excel](#), you can modify the parameter values in the Excel file and then import that Excel file back into the business process test. This enables you to use different data in different test runs to check how your application performs with different data.

1. In the document pane OR the Solution Explorer, select the business process test or flow for which you want to import the component/flow parameter iteration values.
2. In the toolbar, select the **Import Excel values to parameter iteration values** button .
3. In the Open dialog, specify a location from which to import the modified Excel.
4. Click **Open**. UFT confirms the successful import of the Excel file. The component parameter iteration values are now displayed in the Data pane for the component parameters and iterations.

# Troubleshooting and Limitations - Using Data with Business Process Testing in UFT

## **Relevant for: business process testing**

- When referencing data table sheets from a business process test, you should use the data table sheet names, not the SheetIDs.
- Sharing values between components in a BPT test cannot be done using user-defined environment variables. Instead, use user-defined run-time settings that you create using the **Setting.Add** method.
- When using multiline values for component parameters:
  - If you create the parameter in ALM, you can view and edit the parameter in UFT.
  - You cannot create a multiline parameter in UFT.

# Chapter 69: Running Business Process Tests

**Relevant for: business process testing**

This chapter includes:

• <a href="#">Running Iterations in a Business Process Test</a> .....	1046
• <a href="#">Running Iterations of Component Groups in a Business Process Test</a> .....	1047
• <a href="#">Considerations for BPT Test Iterations and Parameters</a> .....	1050
• <a href="#">Run Conditions Overview</a> .....	1051
• <a href="#">Running Business Process Tests with Test Configurations</a> .....	1052
• <a href="#">Data For Test Configurations</a> .....	1053
• <a href="#">Test Cases Generator Overview</a> .....	1054
• <a href="#">Use-Case Scenario: Test Cases Generator</a> .....	1055
• <a href="#">How to Set Run Conditions for a Business Process Test</a> .....	1059
• <a href="#">How to Set Up and Run Test Configurations</a> .....	1061
• <a href="#">How to Use the Test Cases Generator to Create Test Configurations</a> .....	1064

# Running Iterations in a Business Process Test

## **Relevant for: business process testing**

When deciding how to run your business process test, you can vary the number of times a selected flow, component group, or individual component runs in the course of the test run. This enables you to run flows or components with different sets of data.

You can configure how many times, and with which data:

- A business component runs in a test
- A flow runs in a test
- A component group runs in a test

### **Examples**

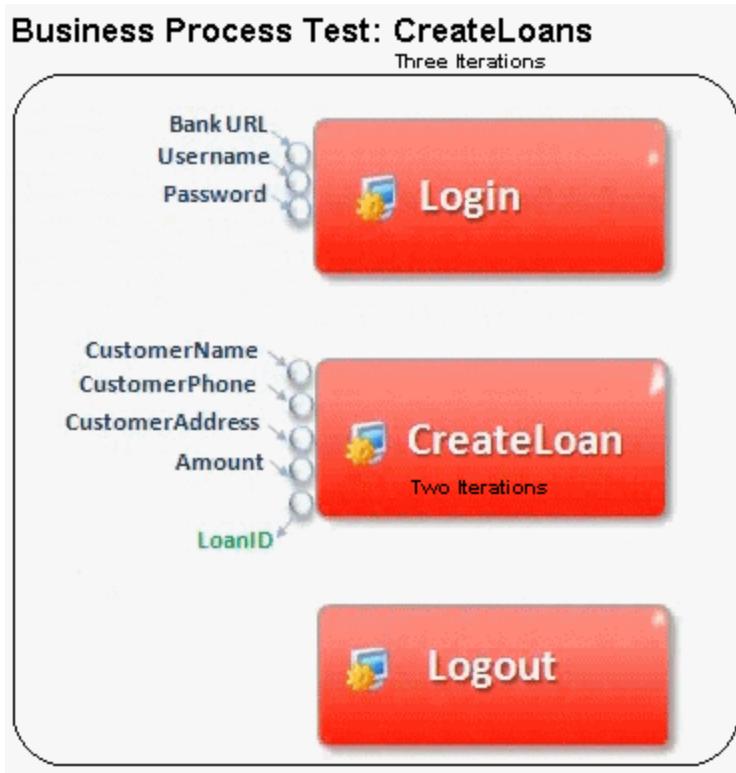
- You can create iterations for a flow that searches for different loans in a test for a banking application by supplying different loan IDs for a loan ID parameter.
- You can create iterations for a test that prepares loans with different interest rates to see which rate is most affordable for the customer.

You define the number of iterations to add in the Data pane. Then, for each iteration, you supply data for the parameter values. You can keep the default value, use the same value as a previous iteration, or vary the values between iterations.

In addition, you can export the component parameter iteration structure to an external Excel and then edit this Excel. After you provide values for the parameters in each iteration, you import the Excel back into the test and UFT automatically uses the parameter values provided in the Excel file. Therefore, you do not need to manually modify the parameter values in the Data pane before each test run.

### **Example**

You create a business process test with components named Login, CreateLoan, and Logout.



In this scenario:

- The entire business process test is iterated three times.
- You can use different values for the parameters BankURL, Username, and Password in each test iteration.
- Within each of the three test iterations, the CreateLoan component is iterated twice. This means that the CreateLoan component iterates a total of six times.
- Different values for the CustomerName, CustomerPhone, CustomerAddress, and the Amount input parameters are used for each iteration of the CreateLoan component. Six different input parameters can be supplied in total.
- The CreateLoan component provides an output value for the LoanID parameter for each iteration (six output values provided in total).

For task details, see "[Set data values for the parameters for each iteration](#)" on page 1042.

## Running Iterations of Component Groups in a Business Process Test

### **Relevant for: business process testing**

Sometimes, it may be helpful to run a group of components together for multiple iterations.

Component groups contained in your test flow are identified by a group node listed above its member components. This group node contains the group icon  and displays the number of iterations for the components included in the group.

You cannot run any of the components included in the group for a different number of iterations than the number of iterations defined for the group.

### Example

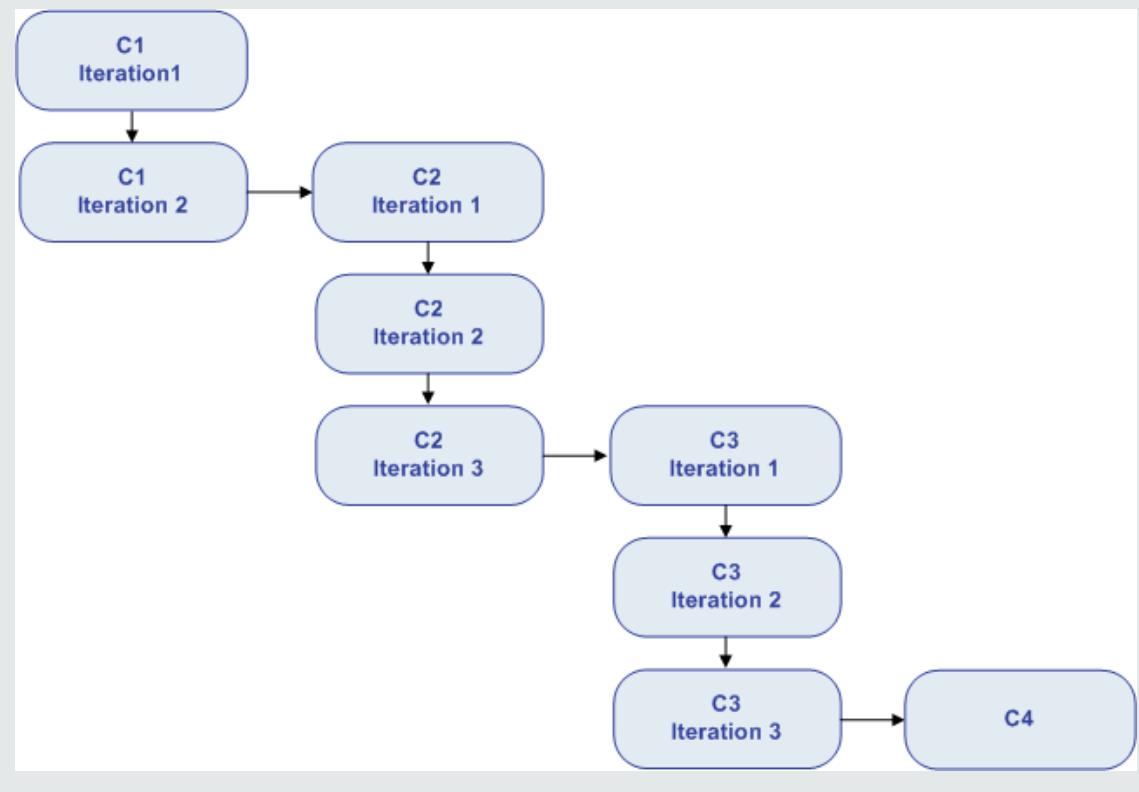
You create a business process test with the following business components: C1, C2, C3, and C4. You set the iterations for the components as follows:

- Component C1 - two iterations
- Component C2 - three iterations
- Component C3 - three iterations
- Component C4 - one iteration

These components run differently, depending on whether you group the components or not:

### Without Grouping

If you do not group any of the components, the business process test runs each component in sequence: C1 for its iterations, C2 for each of iterations, and so forth:



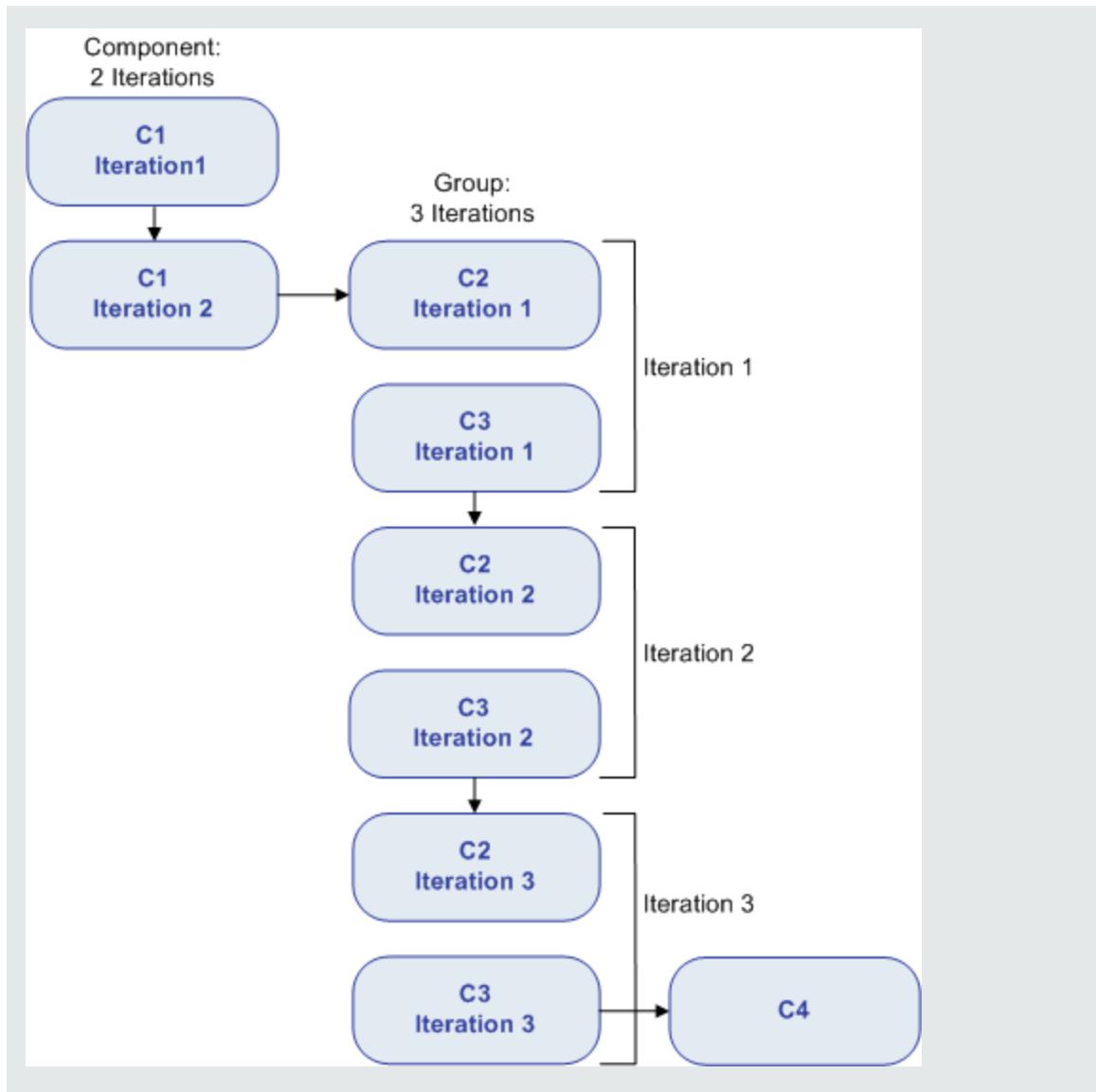
## With Grouping

Instead of running all the iterations of component C1, then all the iterations of component C2, and so forth, grouping the components together changes the manner in which the business process test is run.

In this scenario, you group components C2 and C3 together as a group, and set the group to run for three iterations. Thus, the business process test runs in the following order:

- The first iteration of component C1
- The second iteration of component C1
- The first iteration of component C2
- The first iteration of component C3
- The second iteration of component C2
- The second iteration of component C3
- The third iteration of component C2
- The third iteration of component C3
- The single iteration of component C4

This process is illustrated graphically in the example below:



## Considerations for BPT Test Iterations and Parameters

When working with iterations and parameters for business process test, consider the following:

- "Match an application's post-condition with the next iteration's pre-condition" below
- "Link input parameter values to output parameters" on the next page
- "Moving a group, or a member within a group, can cause a parameter reference to be removed" on the next page

### Match an application's post-condition with the next iteration's pre-condition

For a business component to run iterations successfully, it is essential that the post-condition (the state of the application after the last step in the component runs) match the pre-condition (the state of the

application before the first step in the component runs).

For group iterations to be successful, the state of the application at the end of the last item in the group must match the state of the application before the first item in the group.

For example, if the first component in the group assumes that the Login dialog box in an application is open, then at the point where the last component of the group ends, the Login dialog box must be in an open state before the next iteration begins.

**Note:** Components or flows in a group with input parameters must have the same number of iterations.

### Link input parameter values to output parameters

Iterations in a business process test or flow can result in multiple output parameter values. By linking parameters, each iteration can pass its output value as input to the corresponding target component or flow.

When you use the output of a previous component as the value for an input component parameter, the option applies to all component iterations for that input parameter.

To link input parameters values to other test or component parameters, use the Select Link Source dialog box.

### Moving a group, or a member within a group, can cause a parameter reference to be removed

When a group is moved to a position preceding the component that provides an input component parameter needed by a parameter in the group, the parameter linkage is deleted, and the value for the source parameter will be empty.

You can then either supply a value for the parameter or reinstate the link using the Select Link Source dialog box.

## Run Conditions Overview

### Relevant for: business process testing

You can use run conditions to insert condition statements into your components or business process flows. A **run condition** checks the current value of a component parameter before running the component in a flow. Based on the parameter value and the run condition definition, UFT determines whether to:

- Run the component
- Skip to the next component
- End the component run and set the component status to fail
- Go to a selected flow, component, or group of components

When you run business process tests containing flows with run conditions, the test run results display the results of run conditions in the test, and lists the components that did not run because a run condition was not met. If a run condition is not met, the test results also provide details about the condition that was not met to help you understand why the component run failed or did not run.

**Note:** If you set run conditions, and later add or remove a component or change the component order within a flow, the parameters may no longer be relevant and the run condition may not work. For example, if Component B uses an output parameter value from Component A, and you change the order of the components so that Component B precedes Component A, then Component B cannot receive the output parameter value from Component A and the invalid run condition is ignored.

## Running Business Process Tests with Test Configurations

### Relevant for: business process tests

When you run a business process test in UFT, you can define a specific test configuration to use for that particular test run. A **test configuration** is specific setup of test data that represents a specific use-case of a test. For example, you can have multiple test configurations which each points to a different data resource. This enables you to run the test with a variety of different data sources without manually editing the test steps.

Test configurations essentially unbind the data from the test, making the test generic and facilitating test reuse.

Using test configurations, you can:

<b>Share common data sources across different tests</b>	When you create a test configuration, you associate a specific data resource with that test configuration. However, since the test configuration - but not the data resource - is saved with the test, you can associate a data resource with many tests. When UFT runs the business process test using a test configuration, it calls the referenced data source.
<b>Test various use-cases, each time with a different set of data</b>	As part of creating a test configuration, you both associate a specific data source with the configuration and specify which parts of the data resource should be used in this test configuration. When you change the data resource with each test configuration, you are able to see how your application performs with different sets of data. Furthermore, if you do not change the data resource with each test configuration, but reference different parts of the data resource, you can also see how the application performs with differing data.

### Example

You have a flight booking Web application. In addition to booking flights through the Web site of the application, outside websites can book flights using the application. You can run test configurations to simulate data from each source, including one for the original Web application, as well as data from each outside Web site.

As part of creating a test configuration, you instruct UFT to use specific data for the test run. This data can be associated in a static or dynamic manner. For details, see "[Data For Test Configurations](#)" below.

For task details on how to set up and use test configurations, see "[How to Set Up and Run Test Configurations](#)" on page 1061.

## Data For Test Configurations

### **Relevant for: business process tests**

When running test configurations, there are different ways in which to associate data with the test configuration:

- "[Static configuration](#)" below
- "[Dynamic configuration](#)" on the next page

### **Static configuration**

Static data is used by entering the data directly into the test configuration itself. This is done in ALM in a grid in the **Test Configurations** tab (in the Test Plan module) for the selected test.

In UFT, this data is displayed as read only in the **Test Configurations** tab in the Properties pane:

The screenshot shows the UFT Properties pane with the following interface elements:

- Properties** tab selected.
- Test Configurations:** A dropdown menu set to "Flight\_Application".
- Delete** and **New** buttons.
- Selected Iterations:** Set to "All".
- Static Test Data:** A table with columns: toCity, fromCity, NumberOfTickets, and Class. The table has one row with all cells colored blue.

Static data association is recommended when you only need a small amount of data for the test run.

## Dynamic configuration

Dynamic data associations for a test is done using an external Microsoft Excel file. After you have an Excel file with your test data prepared, you create a test configuration, and UFT prompts you for the associated data resource. To associate a resource, you can do one of the following:

- **Add the Excel file from the file system.** UFT then automatically adds this data resource to the Test Resources module in ALM. This option is useful if you have the Excel file saved locally on your computer.
- **Upload it to the Test Resources module in ALM.** Then, when you add the test configuration, you can select the uploaded data resource from ALM.

After you create the test configuration with the selected data , you can instruct UFT to use different test configurations with different data sources for each test run.

This approach is recommended if you have large amounts of data that is maintained in an external file.

## Test Cases Generator Overview

### Relevant for: business process tests

The Test Cases Generator helps you prepare test configurations by using the parameters in your test and their possible values to create multiple possible data combinations. UFT can then generate them into a test configuration that you can use when running a business process test.

**Note:** By default, you must provide this data yourself. Depending on the number of parameters in your test and their possible values, the possible combinations can grow exponentially and require a lot of time to prepare. Use the Test Cases Generator to prepare this data for you.

The Test Cases Generator creates configurations based on the following algorithms:

<b>Linear</b>	Includes all possible combinations of parameters and values. May result in a very large amount of data.
<b>Pairwise</b>	Includes all combinations of valid values for pairs of input parameters. Assumes that many errors in your application are not caused by single parameters, rather interactions between pairs of parameters.
<b>Triplewise:</b>	Includes all combinations of valid values for triples of input parameters.

Furthermore, you can specify values be added to special paths:

<b>Happy Path:</b>	No error or exception is expected to be raised by using this particular value
<b>Error Path:</b>	An error or exception is expected to be raised by using this particular value

Once you have created a test with parameters, selected a combination algorithm, and added values to the Happy Path or Error Path sets, you can generate the test configurations. UFT generates the test configurations and adds them to your test.

Learn more!

["How to Use the Test Cases Generator to Create Test Configurations" on page 1064](#)

["Use-Case Scenario: Test Cases Generator" below](#)

## Use-Case Scenario: Test Cases Generator

### **Relevant for: business process tests**

Suppose you have a desktop application used for flight reservations. You want to create a business process test of this application.

You must test how the application performs with different sets of data. However, you have up to five fields that users can change, and the possible number of combinations is huge, and manually creating these data combinations is going to take too much time.

Use the Test Cases Generator to help you create this large set of data and make test coverage more manageable.

You've created the following business process test, with multiple components.

Component	Area of Application
<b>Login</b>	Login window
<b>FlightFinder Page</b>	Window to specify flight details (departure, arrival, etc.)
<b>Select Flights Page</b>	Window to select an available flight
<b>Flight Confirmation Page</b>	Window to book the flight with customer details.

For this scenario, we will focus on the **FlightFinderPage** component.

In the FlightFinderPage component, you have four different parameters, representing the fields a user can select.

The screenshot shows the 'BOOK FLIGHT' tab of the 'MyFlight Sample Application'. It includes dropdowns for 'fromCity' (London) and 'toCity' (Paris), a date field '11/11/2015', and dropdowns for 'Class' (Economy) and 'Tickets' (1). Below these is a large orange 'FIND FLIGHTS' button. To the right is a 'Properties' panel titled 'Input' which lists the four fields with their corresponding placeholder values: %[fromCity], %[toCity], %[Class], and %[NumberOfTickets]. Three green arrows point from the 'Properties' panel back to the respective fields on the application screen.

For each of the fields, there are different numbers of possible values:

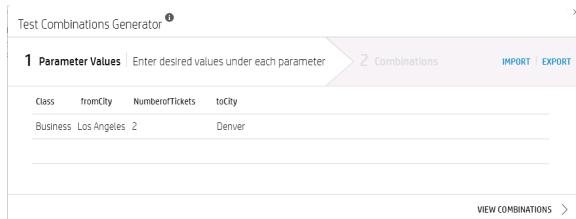
- For the **Departure** and **Arrival** fields, there are 10 possible values.
- For the **Class** field, there are 3 possible values.
- For the **Tickets** field, there are 99 possible values.

If you were to manually created every possible combination, you would have to create  $10 \times 10 \times 3 \times 99$  combinations - a huge total of 29700 combinations.

Use the Test Cases Generator to generate the combinations automatically.

## 1. Provide the possible parameter values.

After [opening the Test Cases Generator](#) in the Test Configurations pane, UFT displays the parameters in the Test Cases Generator main window.



You enter the possible values for each of the parameters.



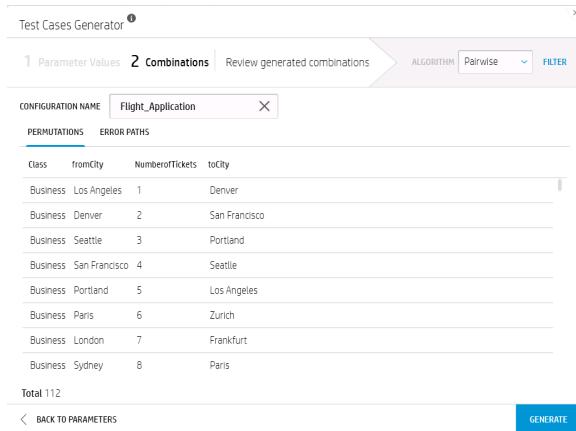
You could specify some of the values as **Happy Path** values (meaning that they are not expected to cause an exception or error in the application) or **Error Path** (meaning that the values are expected to cause an exception or error).

For this scenario, we are not going to specify any values in this manner.

**Note:** Although technically you can enter any number up to 99 in the Tickets field, realistically most users will not use all these numbers. So for testing purposes you can limit it to tickets from 1 until 10.

## 2. View the value combinations.

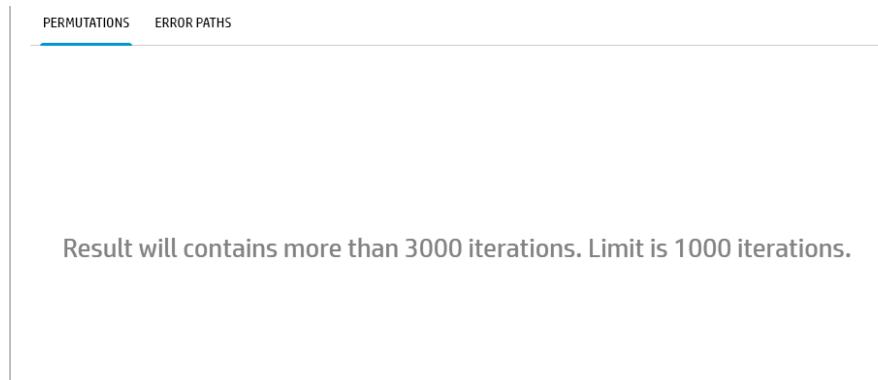
Now that you have all the possible values entered, UFT can generate possible combinations for these parameter values. Once you click the **View Combinations**, UFT displays possible combinations.



## 3. Change the combination algorithm.

By default, when you displayed the possible combinations, UFT selected the **Pairwise** algorithm, resulting in 112 combinations.

You can switch to either the **Linear** or **Triplewise** to display other combinations. However, given the entered parameter values, this results in more combinations than UFT can generate (maximum is 1000).

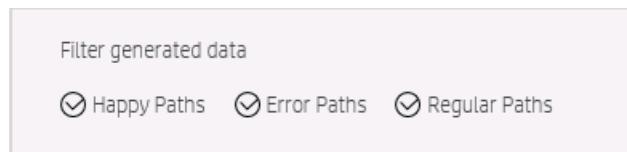


In this case, because the total number of combinations when using Triplewise or Linear algorithms is huge, the **Pairwise** algorithm is the one to select. You can select others, depending on your testing needs, and UFT will only generate 1000 combinations.

#### 4. Generate the configurations.

Now that you have entered the parameters and selected the algorithm to use, you can generate the different combinations.

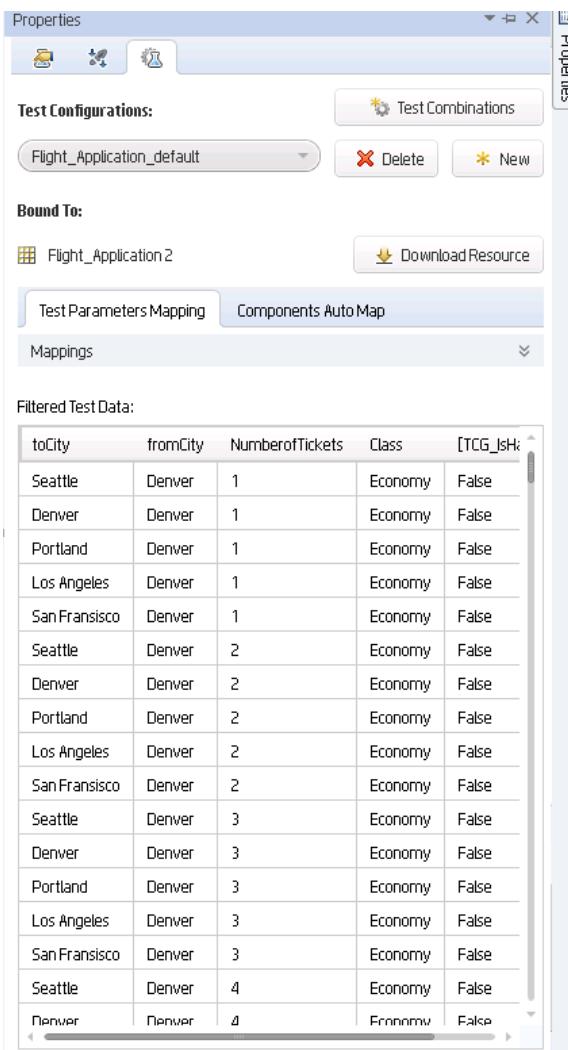
First, instruct UFT which configurations to generate. If you click **Filter**, you can choose from **Regular Path** (the standard combinations), **Happy Path**, or **Error Path** combinations:



For this scenario, you only need the **Regular Path** configurations.

Then, in the bottom part of the Test Cases Generator, click **Generate**. UFT pauses for a bit, generates the combinations and adds them as a new test configuration in the Properties pane.

This test configuration can now be used in any test run.



Learn more!

["How to Use the Test Cases Generator to Create Test Configurations" on page 1064](#)

## How to Set Run Conditions for a Business Process Test

### Relevant for: business process testing

The following steps describe how to set run conditions for the flows and components in a business process test or flow.

**Note:** This task is part of a higher-level task. For details, see ["How to Create, Maintain, and Run Business Process Testing Tests and Flows in UFT" on page 981](#).

This task includes the following steps:

- "Prerequisites" below
- "Select the flow or component to set run conditions" below
- "Set On Failure settings for a flow or component" below
- "Set run conditions for a business process flow or component" below
- "Test the run conditions" on the next page

## 1. Prerequisites

Ensure that the component for which you are setting run conditions has parameters or that the flow containing the component has a parameter.

## 2. Select the flow or component to set run conditions

- a. Do one of the following:
  - If you are setting run conditions for a flow, in the document pane, select the test containing the flow.
  - If you are setting run conditions for a component, in the document pane, select the business process test or business process flow containing the component.
- b. In the document pane, select the flow or component to set run conditions.

## 3. Set On Failure settings for a flow or component

If you are editing a test, you can set **On Failure** options for all the flows or components included in the test:

- a. In the Solution Explorer or document pane, select the business process test you are editing.
- b. In the document pane, select the flow or component for which to set On Failure settings.
- c. In the Properties pane, open the **Properties** tab .
- d. In the Properties tab, set the options for the selected flow or component:
  - **Continue.** Continues the test regardless whether the test passes or fails.
  - **Exits.** Immediately stops the test and exists the test run.

**Note:** For versions ALM 12.20 and higher, if you defined a default **On failure** condition in the component in ALM, UFT displays the default value.

## 4. Set run conditions for a business process flow or component

When you are editing a business process flow or component, you can set additional run options for the components included in the flow:

- a. In the Solution Explorer or the document pane, select the business process test or flow on which you are working.
- b. In the document pane, select the component to which you want to add run conditions.

- c. In the **Properties** tab  in the Properties pane, select the **Use run condition** option.
- d. In the middle section of the Properties tab, set the condition options:
  - **Run if:** Indicates what type of parameter (flow or component parameter) and the name of parameter for which a condition must be met
  - **Is:** the operator for the parameter value
  - **Else:** Instructs UFT what to do with the test run if the condition is not met, including:

<b>Skip to next component and continue</b>	The selected component does not run, and the test run continues with the next component in the flow. The component is not displayed in the run results.
<b>End component run and fail.</b>	The component for which the run condition is set does not run, but instead sets the status of the component run as Failed. The flow either continues to the next component or ends, depending on the failure condition set for the component.
<b>Go To</b>	<p>UFT continues to the specified place in the test. You select the location in the <b>Go To:</b> dropdown list.</p> <p>The selected flow, component, or group of components must exist in the test after the current component. For example, you cannot go to a component that has already run.</p> <p><b>Note:</b> The GOTO condition is supported only for ALM versions 12.50 patch 1 or higher.</p>

## 5. Test the run conditions

Run the test to ensure that the run conditions were performed correctly.

**Note:** If a run condition is not valid, the run condition link is displayed in red. This can happen, for example, if a reference parameter was deleted, a parameter value was encrypted, and so on. Delete the run condition and define a new one.

# How to Set Up and Run Test Configurations

## Relevant for: business process tests

This task describes how to set up and run test configurations for business process tests in UFT.

**Note:** This task is part of a higher-level task. For details, see "[How to Create, Maintain, and Run Business Process Testing Tests and Flows in UFT](#)" on page 981.

This task includes the following steps:

1. "[Prerequisite](#) " on the next page
2. "[Create a test configuration](#)" on the next page
3. "[Enter static data configuration values](#)" on the next page
4. "[Map test parameters to the Excel file](#)" on the next page

5. "View component parameter mapping details " on the next page
6. "Run the test with the selected configuration." on page 1064

## 1. Prerequisite

Create an Excel file containing your test data.

Your Excel file should contain the following, as needed

- **If you have test parameters:** The first sheet must contain the test parameters.
- A separate sheet for each component with parameters.
- In the sheets for the individual components, the columns containing the parameter names must be in the format <COMPONENT NAME>. <PARAMETER NAME>.

**Tip:** If you have iterations defined for the components included in your business process test, you can export the iteration and parameter names/values for these components into an Excel file. For details, see "[Export component parameters to an Excel](#)" on page 1043.

## 2. Create a test configuration

By default, UFT automatically creates a static test configuration with each business process test, using the name of the test. You can create other test configurations as needed:

- a. In the Properties pane, open the **Test Configurations** tab .
- b. Do one of the following:

<b>From a data set</b>	<ol style="list-style-type: none"><li>i. In the Test Configurations tab, click the <b>New</b> button.</li><li>ii. In the Open dialog box, navigate to the location (in ALM or the file system) where the Excel file is saved and select the file.</li><li>iii. Click <b>OK</b> to associate the Excel file with the test configuration. UFT displays the test configuration tabs.</li></ol>
<b>From the Test Cases Generator</b>	<ol style="list-style-type: none"><li>i. In the Test Configurations tab, click the <b>Test Combinations</b> button.</li><li>ii. Generate your test configurations as described in "<a href="#">How to Use the Test Cases Generator to Create Test Configurations</a>" on page 1064</li></ol>

## 3. Enter static data configuration values

In UFT, static data configuration values are read-only. To add data to the static configuration, you must edit the test configuration in ALM.

For details, see the task on how to associate static data in the *HP Application Lifecycle Management User Guide*.

## 4. Map test parameters to the Excel file

After you associate the Excel file with a specific test configuration, you must instruct UFT where to provide the values for your test parameters:

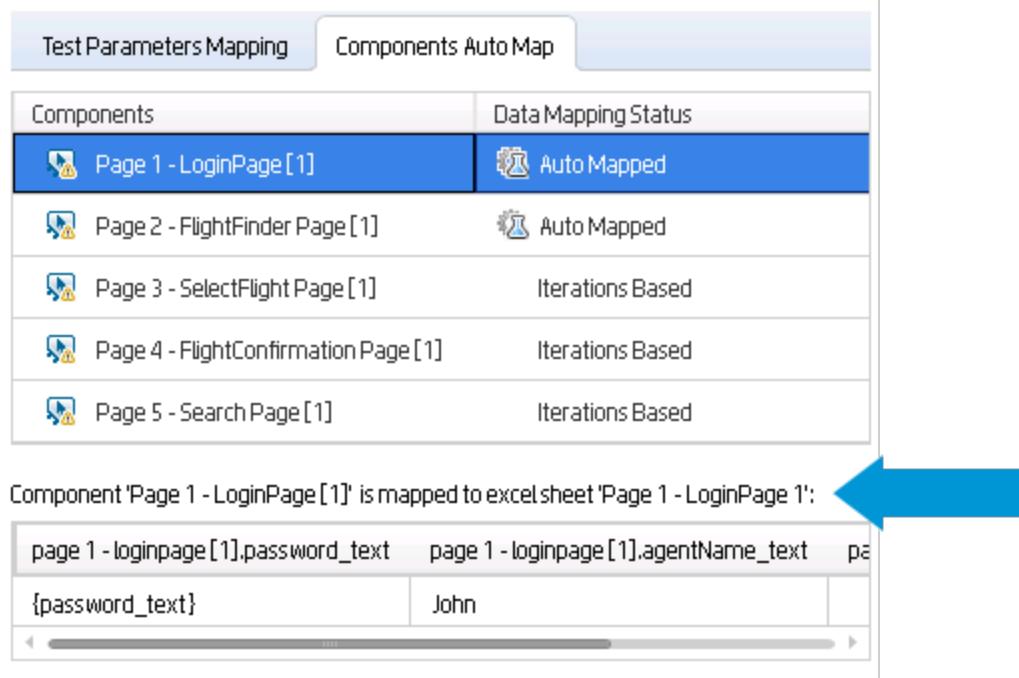
- a. In the Test Configurations tab, select the **Test Parameters Mapping** tab. A list of all test parameters is displayed.
- b. In the **Resource Column**, from the drop-down list, select the name of the column to which you want to map the parameter.
- c. (Optional) In the **Filter** column, select the value to use.

**Tip:** If you want UFT to automatically map all test parameters to the relevant column in the Excel file, click the Automap button. UFT automatically finds the correct column and displays an icon  indicating that the parameter is automatically mapped.

## 5. View component parameter mapping details

In addition to being able to map test parameters to specific columns in the Excel file, you can also see how the individual components are mapped to parts of the Excel file:

- a. In the Test Configurations tab, select the **Components Automap** tab. A list of all components in the test is displayed.
- b. In the components list, select the component you want to view.
- c. Below the components list, UFT displays which sheet in the Excel file the component and its parameters are mapped to, including the values for the parameters:



Components	Data Mapping Status
 Page 1 - LoginPage [1]	 Auto Mapped
 Page 2 - FlightFinder Page [1]	 Auto Mapped
 Page 3 - SelectFlight Page [1]	Iterations Based
 Page 4 - FlightConfirmation Page [1]	Iterations Based
 Page 5 - Search Page [1]	Iterations Based

Component 'Page 1 - LoginPage [1]' is mapped to excel sheet 'Page 1 - LoginPage [1]':

page 1 - loginpage [1].password_text	page 1 - loginpage [1].agentName_text	pa
{password_text}	John	

If there are errors, UFT displays a warning icon in the Components Automap tab and in the Data Mapping Status column. You can hover over the error name to see a description of the error.

Errors must be corrected in the Excel file.

## 6. Run the test with the selected configuration.

After you have created the necessary test configurations, you can run the test with any of the selected configurations.

- a. In the Run dialog, expand the **Options** node.
- b. From the Options area, select the **Test Configurations** tab.
- c. In the Test Configurations tab, select the Test Configuration you want to use.

When the test runs, UFT takes the data from the file associated with the selected test configuration.

# How to Use the Test Cases Generator to Create Test Configurations

## Relevant for: business process tests and flows

This task describes how to use the Test Cases Generator to generate test configurations, enabling you to create a larger variety of test configuration data sets. These test configurations can be used to test a wider variety of scenarios for your application.

**Note:** This task is part of a higher-level task. For details, see "[How to Set Up and Run Test Configurations](#)" on page 1061.

**Tip:** For a use-case scenario related to this task, see "[Use-Case Scenario: Test Cases Generator](#)" on page 1055.

This task includes the following steps:

1. ["Prerequisites" below](#)
2. ["Open the Test Cases Generator" below](#)
3. ["Set the value of your parameters" on the next page](#)
4. ["Set values as Happy Path or Error Path" on the next page](#)
5. ["View the selected parameter combinations" on the next page](#)
6. ["Change the testing combination algorithm" on the next page](#)
7. ["Select the configurations to generate" on the next page](#)
8. ["Generate the test configurations." on page 1066](#)

### 1. Prerequisites

Before using the Test Cases Generator, you must have test parameters.

### 2. Open the Test Cases Generator

In the Properties pane, select the **Test Configurations** tab , and then click the **Test Combinations** button.

The Test Cases Generator is opened, with a column for each of your test's parameters:

### 3. Set the value of your parameters

Enter the possible values for your parameters as follows:

<b>Manually</b>	Use the grid to enter the possible values for your parameters. Click on a cell and enter the values for the parameters as needed.
<b>Import</b>	Click the <b>Import</b> button to automatically import values for the parameters. You can import an Excel file saved on the file system or in your ALM project.

### 4. Set values as Happy Path or Error Path

Set component values, and click one of the following:

<b>Happy path</b>	The value is expected not to cause an exception or error.
<b>Error path</b>	The value is expected to cause an exception or error. This is useful for negative testing of your application.

UFT changes the selected parameter to green (for Happy Path values) or red (for Error Path values):

**Note:** If you need to clear a value from the Happy Path or Error Path, select the value and click the **Reset** button.

### 5. View the selected parameter combinations

After entering the possible values for all parameters, click **View Combinations**.

UFT displays all the possible combinations for your parameter values as follows:

<b>Regular/Happy path values</b>	All possible combinations of the values are listed in the Permutations tab.
<b>Error path values</b>	All possible combinations are listed in the Error Paths tab, using the error path values.

### 6. Change the testing combination algorithm

UFT can generate different configurations depending on the selected combination algorithm (**Linear**, **Pairwise**, or **Triplewise**):

In the Permutations tab, from the **Algorithm** drop-down list, select the appropriate algorithm for your testing needs. UFT updates the list of permutations accordingly.

**Note:** Depending on the number of parameters, possible values, and combination algorithm, you may generate a large number of possible configurations. UFT can only display (and generate) up to 1000 possible configurations.

### 7. Select the configurations to generate

If you define values as **Happy Path** or **Error Path**, you also have the option to generate all the different parts of the configuration.

- a. In the Permutations tab, click the **Filter** button.
- b. In the filter, select the combinations to generate.
- c. If necessary, enter a name for the generated configuration.

#### 8. **Generate the test configurations.**

In the bottom of the Test Cases Generator, click **Generate**.

UFT pauses for a few seconds or more (depending on the number of combinations), and creates the test configuration. The new configuration is then added in the Test Configurations tab of the Properties pane.

This configuration is saved with the test and is available to use in test runs.

# Chapter 70: Business Process Testing with the BPT Packaged Apps Kit

**Relevant for: business process tests and flows**

This chapter includes:

• <a href="#">BPT Packaged Apps Kit - Overview</a> .....	1068
• <a href="#">Learning Tests and Flows</a> .....	1069
• <a href="#">How UFT Reuses Learned Components</a> .....	1071
• <a href="#">Detecting and Resolving Changes</a> .....	1072
• <a href="#">How to Learn Business Process Tests and Flows</a> .....	1073
• <a href="#">How to Detect and Resolve Changes Using Change Detection Mode</a> .....	1079
• <a href="#">Business Process Testing with the BPT Packaged Apps Kit - Troubleshooting and Limitations</a> .....	1083

# BPT Packaged Apps Kit - Overview

## Relevant for: business process tests and flows

Normally, when you create business process tests of your application, you can create tests in different ways:

- **Create individual components of each area of the application.** These components each contain their own application area that contains an object repository of the objects in the area of the application.
- **Record a business process test.** You perform user actions in your application, and UFT changes these actions into steps inside the test's components. While recording, you can also add additional components, enabling you to create individual components for divisions of your application as needed.

However, when you are creating business process tests of your SAP applications, you can use the BPT Packaged Apps Kit to enable test creation. The BPT Packaged Apps Kit is a built-in functionality of UFT available when working with the Add-in for SAP Solutions in UFT.

When you use the BPT Packaged Apps Kit, you can:

- **Automatically learn the actions you perform on your SAP application**, and then generate a flow containing components based on the screens and transactions within your applications. As part of the learning process, you can reuse learned components instead of creating new components each time you learn an area of your application.
- **Run tests in Change Detection Mode**, which enables you to determine how your application has changed since the test or flow was created. After UFT determines the changes, you can automatically update your tests, flows, and components as needed.

The BPT Packaged Apps Kit works with any supported version of ALM. Change Detection Mode is available for ALM versions 12.50, ALM 12.21, ALM 12.01 patch 2 or higher, and ALM 11.52 patch 7 or higher.

For task details, see "[How to Learn Business Process Tests and Flows](#)" on page 1073 and "[How to Detect and Resolve Changes Using Change Detection Mode](#)" on page 1079.

### Note: (for testing SAP Fiori applications)

- SAP Fiori support is technology preview level.
- When using the BPT Packaged Apps Kit to test Fiori applications, UFT can learn the application but cannot run the test in Change Detection Mode.
- When learning a Fiori application or running a learned test of a Fiori application, you should open only one browser and an additional tab in the browser (in addition to the application).

# Learning Tests and Flows

## Relevant for: business process tests and flows

If you are testing an SAP application, you can use UFT to "learn" components automatically from the application.

First, you must create or open a business process test or flow, and start the Learn process. While UFT is learning the application, you perform the operations in your application that you want UFT to learn.

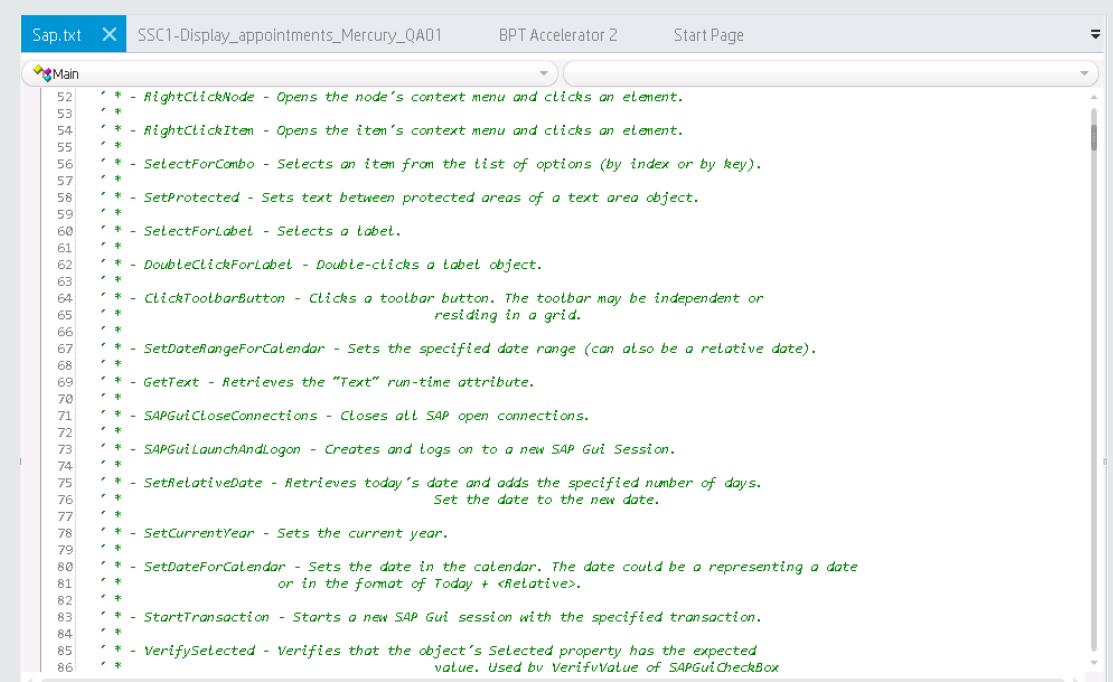
After you finish performing all of the steps in your application, UFT automatically breaks down the learned areas of the application into a series of business components. Each of these components represents a different screen or tab (or "transaction") in your application. Parameters are automatically added for the steps in the components, based on the values you used when learning the components.

**Note:** During the Learn process, resources are created in **BPT Resources** folder in the **Resources** module. For your business process tests or flows to work properly, this folder and its subfolders should not be renamed, moved, or deleted.

## Considerations

- You must have the Web and Add-in for SAP Solutions installed and loaded to learn business process tests and flows. You can also load the SAPUI5 add-in to learn business process tests and flows.
- It is recommended to log in to your SAP application before starting to learn components.

**Tip:** If you need to test the login area of your SAP application, you can create an additional component that automatically opens and logs into the application, using the **SAPGUILaunchandLogon** function, specifying the required server information:



The screenshot shows a software interface titled "BPT Accelerator 2". At the top, there's a tab labeled "Sap.txt" and a window title "SSC1-Display\_appointments\_Mercury\_QA01". Below the tabs, there's a toolbar with icons for Main, SAP Gui, and Help. The main area contains a code editor with the following content:

```

Main
52 /* - RightClickNode - Opens the node's context menu and clicks an element.
53 */
54 /* - RightClickItem - Opens the item's context menu and clicks an element.
55 */
56 /* - SelectForCombo - Selects an item from the list of options (by index or by key).
57 */
58 /* - SetProtected - Sets text between protected areas of a text area object.
59 */
60 /* - SelectForLabel - Selects a label.
61 */
62 /* - DoubleClickForLabel - Double-clicks a label object.
63 */
64 /* - ClickToolbarButton - Clicks a toolbar button. The toolbar may be independent or
65 residing in a grid.
66 */
67 /* - SetDateRangeForCalendar - Sets the specified date range (can also be a relative date).
68 */
69 /* - GetText - Retrieves the "Text" run-time attribute.
70 */
71 /* - SAPGuiCloseConnections - Closes all SAP open connections.
72 */
73 /* - SAPGuiLaunchAndLogon - Creates and logs on to a new SAP Gui Session.
74 */
75 /* - SetRelativeDate - Retrieves today's date and adds the specified number of days.
76 Set the date to the new date.
77 */
78 /* - SetCurrentYear - Sets the current year.
79 */
80 /* - SetDateForCalendar - Sets the date in the calendar. The date could be a representing a date
81 or in the format of Today + <relative>.
82 */
83 /* - StartTransaction - Starts a new SAP Gui session with the specified transaction.
84 */
85 /* - VerifySelected - Verifies that the object's Selected property has the expected
86 value. Used by VerifyValue of SAPGuiCheckBox

```

This function is available with the **SAP.txt** function library attached to any application area when you create an application area with the Add-in for SAP Solutions loaded.

- To learn a test or flow, you must be part of an ALM user group with the following task permissions: **Modify Folder (Test Plan), Modify Test, Add Component Folder, Add Component, Add Step, Add Parameter, Add Resource, Modify Component, Modify Step, Modify Parameter, Modify Resource, and Delete Resource**.
- To enable more efficient component reuse when learning, it is recommended to perform the same operation in the same manner each time you learn. For example, click the **Enter** button or pressing the **ENTER** key on the keyboard results in different learned steps. Therefore, if you are not consistent when performing such operations, two otherwise identical components might be considered only similar rather than identical.
- Using the keyboard input to navigate between areas of your application instead of click buttons outside your screen or tab results in fewer learned components in your test or flow. For example, clicking the **Enter** button during the Learn Flow process results in the creation of a new component for the Enter button, whereas pressing **ENTER** results in an additional step being added to the existing component.

## How UFT Reuses Learned Components

### Relevant for: business process tests and flows

When UFT learns an application, it analyzes each of the components in the test or flow to see if there are already components similar to, or identical to, the learned components. If such a component exists, you can reuse the existing component instead of creating a new component.

Existing components in the project are compared to the learned component using the following criteria:

- Both components represent the same screen/area of the application.
- Both components represent the same screen/area with exactly the same objects.
- Both components contain the same steps.

This is true also for checkpoint and output value steps, which must refer to the same object properties.

**Note:** If two or more steps refer to the same method, they are considered identical only if they refer to the same objects, and contain the same number of arguments.

- Both components contain the same steps in the same order.

Only components in the project that were created through the Learn process are analyzed for similarity.

You can also instruct UFT whether to automatically reuse similar or identical components (without prompting you) or enable you to choose the components to reuse. This option is available in the **BPT Packaged Apps Kit** pane of the Options dialog box (**Tools > Options > BPT Testing tab > BPT Packaged Apps Kit** node).

# Detecting and Resolving Changes

## Relevant for: business process tests and flows

When using business process tests in UFT with the BPT Packaged Apps Kit, you can run business process tests and flows in Change Detection mode. When you run a test in Change Detection Mode, UFT checks for changes made in the application since the test or flow was last updated. After the test or flow run is finished, UFT displays the changes in a Change Detection report. This report provides an indication of the modifications you should make to your test or flow to make sure that the test or flow is up-to-date.

**Note:** Running tests in Change Detection mode is supported only for SAP GUI applications.

For each change listed in the report, UFT offers possible resolutions. This enables you automatically update your test or flow without learning or manually updating the component's steps. You can also choose to resolve a change manually if you want.

For example, suppose you are testing a screen for inputting contact information for new customers. The screen contains the fields Name, Address, and Phone Number. You create a test that verifies that information entered in these fields is correctly added to your customer database. Suppose you now add an E-mail address field to the screen. If you run your test in regular mode, the test may pass and you may not notice that there is an additional field that should be tested. However, if you run the test in Change Detection mode, UFT notices that the field was added to the screen and suggests adding a step to the component corresponding to the new field. You can then run an updated version of the test that includes verification of the additional field.

Similarly, if a field was removed from the screen, UFT notices that the field was removed, even if no step in the component corresponds to the field. The Change Detection report suggests updating the component to the changed screen.

## Considerations

- To use Change Detection mode, you must use ALM 12.21, ALM 12.01 patch 2 or higher, or ALM 11.52 patch 7 or higher.
- To detect changes in Change Detection mode, a user must belong to a user group that has permissions for the **Run** task, and permissions to modify tests and business components. For details on configuring user group permissions in ALM, see the *HP Application Lifecycle Management Administrator Guide*.
- Test and flows that are run in Change Detection mode run all the iterations of the selected range of iterations of any component, flow, or test, even if multiple iterations are defined.
- Only business components created using the Learn process can run in Change Detection mode.

# How to Learn Business Process Tests and Flows

## Relevant for: business process tests and flows

This task describes how to learn areas of your SAP application in order to create business process tests and flows. This enables you to create components based on screens or areas of your SAP application quickly.

This task includes the following steps:

1. ["Prerequisites" below](#)
2. ["Set component reuse options" below](#)
3. ["Set parameter options" on the next page](#)
4. ["Perform user actions your SAP application" on the next page](#)
5. ["Optional - add checkpoints and output values while learning" on the next page](#)
6. ["Select components" on page 1075](#)
7. ["Reuse an existing component" on page 1075](#)
8. ["Remove a learned component" on page 1076](#)
9. ["Resume learning components" on page 1076](#)
10. ["Edit table parameters" on page 1077](#)

### 1. Prerequisites

Before learning business process tests and flows, you must do the following:

- Install and load the Web and Add-in for SAP Solutions or the SAPUI5 add-in.
- Log in to your SAP application.

**Note:** If you are learning an SAP Fiori application, ensure that the SAPUI5 add-in is loaded and open the browser and an additional tab in the browser before learning.

- Create or open a business process test or flow.

### 2. Set component reuse options

In the BPT Packaged Apps Kit pane of the Options dialog box (**Tools > Options > BPT Testing tab > BPT Packaged Apps Kit node**), select the appropriate option to instruct UFT on how to reuse similar or identical components:

<b>Manually select components for reuse:</b>	This option enables you to select the components in the Learn Summary dialog box after finishing the learning session.
<b>Automatically reuse identical</b>	This automatically uses existing components that are identical without prompting you after learning the application.

<b>components:</b>	
<b>Ignore comments when comparing scripts</b>	This instructs UFT to ignore comment steps when comparing learned components to existing ones.
<b>Component Type:</b>	Select from Keyword GUI or Scripted GUI components.

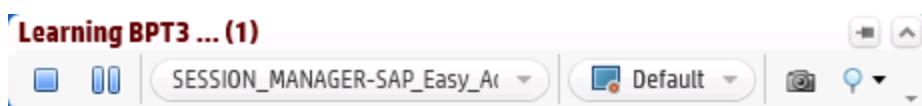
### 3. Set parameter options

In the BPT Packaged Apps Kit pane of the Options dialog (**Tools > Options > BPT Testing tab > BPT Packaged Apps Kit node**), select how you want UFT to reuse parameters:

- **Automatically parameterize steps using Business Component parameters:** Enables you to use the Business component parameters as the step value parameters
- **Use the default values from the learned flow:** Uses the values you entered when learning the flow
- **Use the values from the reused components:** Use the values from the components you select for reuse.

### 4. Perform user actions your SAP application

- a. Do one of the following:
  - In UFT, with a business process test or flow selected, in the toolbar, press the **Learn** button . UFT is minimized and the Learn toolbar is displayed.
  - In the BPT View, click the **Smart Record a New Business Process Test or Flow** button.
- b. Perform user actions in your SAP application. As you perform actions, the Learn toolbar provides a count of the number of steps performed in the application:



- c. When you are finished performing all the necessary actions, press the **Stop** button . UFT displays a progress indicator, and adds the components to your test and to your ALM project.

### 5. Optional - add checkpoints and output values while learning

Using the Learn Toolbar, you can also add checkpoints and output values while learning your SAP application. This eliminates the need for you to add these steps after learning components.

- a. While performing user actions in your SAP application, in the Learn toolbar, click the **Insert Checkpoint or Output Value**  button and select the type of checkpoint to insert.
- b. If necessary, in the Object Selection dialog box, select the object on which you want to insert the checkpoint or output value.

- c. In the Checkpoint Properties dialog that opens, select the test object properties to check and click **OK**. The Learn toolbar counter changes to note that you added a checkpoint or output value step. In addition, this checkpoint step will be part of the learned component that is created after you stop learning the application.

## 6. Select components

After you finish learning the screens/areas in your application, UFT displays a summary of the learning session in the Learn Summary dialog box.

If you want to keep the learned components without reusing similar or identical components, click **Create**. UFT creates the new components and saves them in your ALM project. In addition, the learned components are added to the open business process test.

## 7. Reuse an existing component

If UFT detects that a learned component is similar or identical to an existing component, UFT displays the number of similar components next to the component name in the component tree.

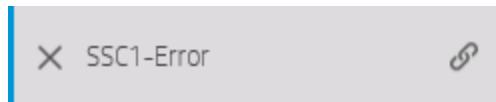
- a. In the learned component list, select the component for comparison. In the right pane, UFT displays the list of possible components to reuse.
- b. In the right pane, display the **Available Components for Reuse** section.

The screenshot shows the 'Learn Summary' dialog box. At the top, it displays '5 COMPONENTS LEARNED'. Below this, a tree view shows several learned components, some with a '50%' similarity indicator. To the right of the tree view, there are two tabs: 'STEPS' and 'AVAILABLE COMPONENTS FOR REUSE (2)'. The 'AVAILABLE COMPONENTS FOR REUSE (2)' tab is selected, showing a list of components that can be reused. At the bottom of the dialog, there are buttons for 'AUTOMATICALLY REUSE COMPONENTS', 'CREATE', and 'CANCEL'.

**Tip:** If you want to view details about a similar or identical component, in the Available components list, click the arrow to the left of the component name. UFT then displays further details about the component, including areas of similarity and the steps used in the component to be reused:

The screenshot shows a component details page. At the top, there's a circular progress bar with '50%' and the text 'SSC1-Display\_appointments\_Mercury\_QA01\_1'. To the right is a 'Reuse' button. Below the title, there are three tabs: 'PROPERTIES' (which is underlined), 'STEPS', and 'USED BY'. Under 'PROPERTIES', it says 'SIMILARITY 50%'. To the right of this are four status items: 'Represent Similar Screen' (green checkmark), 'Similar Objects' (green checkmark), 'Similar Steps' (red X), and 'Similar Step Order' (red X). Below this section, there are three more properties listed: 'Designer' (aaa), 'Date' (30.03.2015), and 'Path' (Components\FlowAutoBC\SAP\Subject\Docs\BPT Accelerator 3).

- c. From the list of available components for reuse, select a component and click the **Reuse** button. UFT replaces the learned component with the existing component, and updates it with an icon in the learned components list:



- d. Repeat this process for each component that you wish to reuse.
- e. When you are finished selecting the components that should be reused, click **Create**. UFT creates the new components and saves them in your ALM project.

## 8. Remove a learned component

- a. In the component list, select the component to remove.
- b. To the left of the component, click the **X** button.
- c. After you have removed all the unnecessary components, click **Create**. UFT creates the new components and saves them in the specified location in the ALM project.

## 9. Resume learning components

After you create an initial test with learned components, you can resume learning components in the same test.

- a. In the document pane or the Solution Explorer, select the business process test or flow to which you want to add learned components.
- b. In the toolbar, click the **Learn** button.
- c. When prompted, decide how to insert newly learned components:

<b>Yes</b>	If you select Yes, the existing components in the test are removed and newly learned components are inserted instead.
<b>No</b>	If you select No, UFT inserts the newly learned components after the already existing components in the test.

- d. Continue learning the screens/areas in your application as described in "[Perform user actions your SAP application](#)" on page 1074.

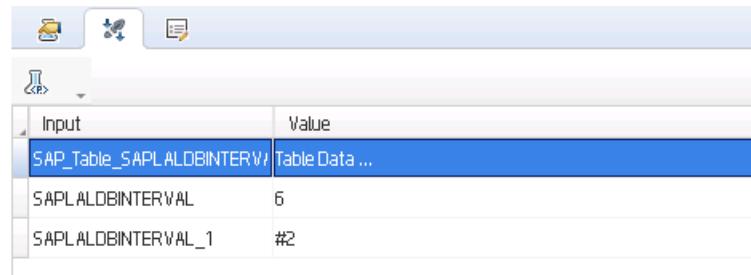
## 10. Edit table parameters

When learning your application, you have the additional option to learn and create special table parameters which represent a table object in the application. When UFT creates the components after learning, the parameter is created and you can edit the values of the parameter (like other test and component parameters) using a special dialog.

To create and edit table parameters, do the following:

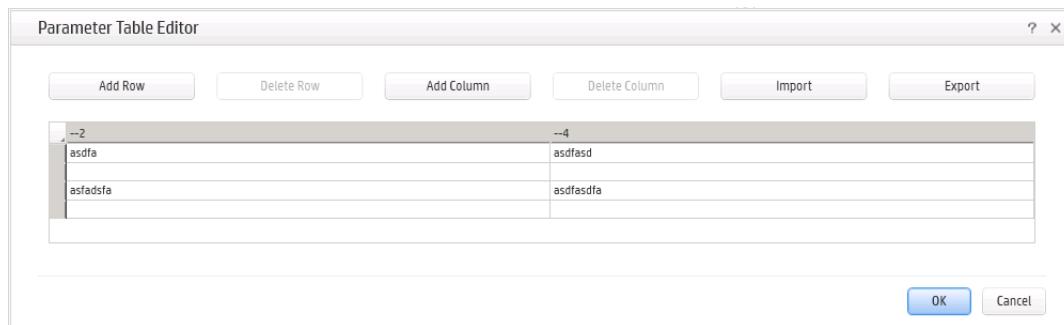
- a. Create or open a GUI test.
- b. In the SAP General pane of the Options dialog box (**Tools > Options > GUI Testing tab > SAP > General** node), select the **Auto-parameterize table and grid controls** option.
- c. Close the GUI test.
- d. Create or open a business process test.
- e. Perform user actions on the table objects in your application. While you perform these actions, UFT learns the application and creates components accordingly.
- f. After performing all necessary user actions, stop the learning session and approve/reject the learned components in the Learn Summary report.
- g. In the test grid, select the component containing the table parameter. The Properties pane tabs are changed to represent the component properties.

- h. In the Properties pane, select the **Parameters** tab .
- i. In the Parameters tab, select the parameter representing your table object. The Properties pane identifies these by labeling them in the **Value** column as **Table Data**:



Input	Value
SAP_Table_SAPALDBINTERVAL	Table Data ...
SAPALDBINTERVAL	6
SAPALDBINTERVAL_1	#2

- j. In the row containing the table parameter, click the **Table Parameter** button . The Table Parameter Editor dialog box opens:



- k. In the Parameter Table editor, edit the parameter values as needed.  
l. When you have finished editing the table parameter, click **OK** to save your changes.

# How to Detect and Resolve Changes Using Change Detection Mode

## **Relevant for: business process tests and flows**

This task describes how to run a business process test or flow of your SAP application in Change Detection Mode. This is useful to check if your SAP application has changed and enable UFT to help you automatically update the test or flow's components with the new steps.

This task includes the following steps:

1. ["Prerequisites" below](#)
2. ["Start the test run in Change Detection Mode" below](#)
3. ["Update the changed components and steps" on the next page](#)
4. ["Save changed components to your ALM project" on page 1082](#)
5. ["Optional - view run results for the test run" on page 1082](#)

### **1. Prerequisites**

Before running a test in Change Detection Mode:

- Save your test on an ALM server running ALM version 12.50, ALM 12.21, ALM 12.01 patch 2 or higher, or ALM 11.52 patch 7 or higher.
- In UFT, install and load the Add-in for SAP Solutions.
- If you want to view the Change Detection Report directly from ALM, you must have UFT installed on the same computer.

### **2. Start the test run in Change Detection Mode**

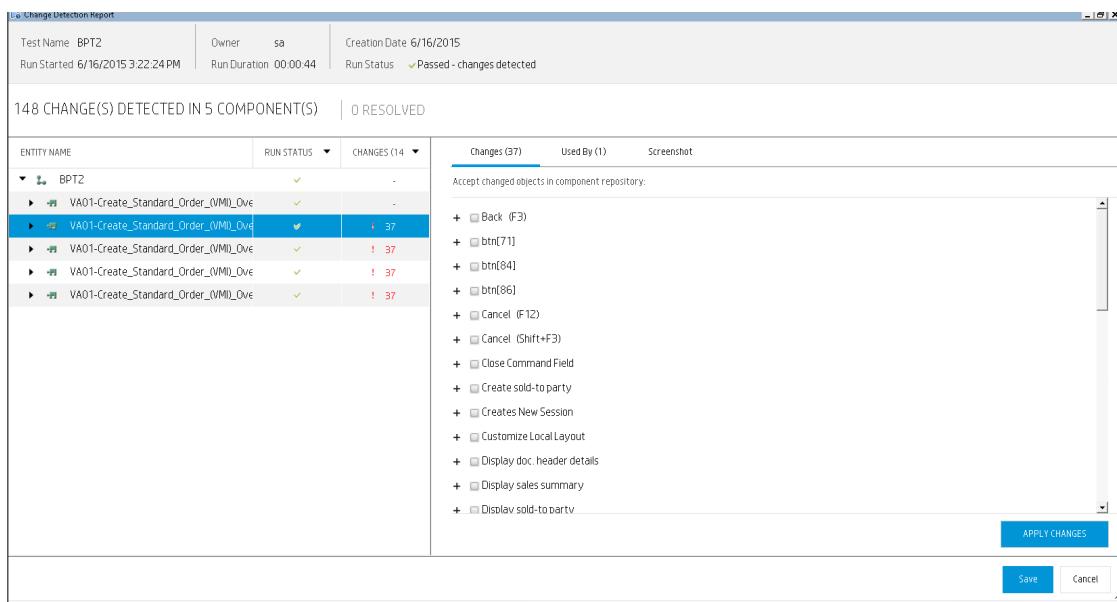
With a business process test open and selected, in the toolbar, click the **Run** button down arrow



and select **Change Detection Run Mode**.

**Note:** UFT does not support running a learned test of an SAP Fiori application.

UFT runs the test in the same manner as a regular test run. UFT is minimized and the test or flow steps are performed on your application. At the end of the test run, the Change Detection Report opens in a separate window:



### 3. Update the changed components and steps

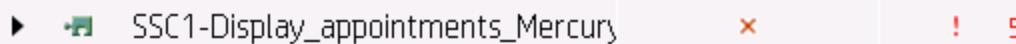
In the Change Detection Report, UFT displays a number of things:

- Run results for the test or flow, for each component, and for each step in the test.
- Changes in the application for each component
- Screenshots of the different versions of the application

Using the report, you can update your components and steps automatically:

- a. In the component tree, select the component for which you want to resolve changes.

You can see which components in which you need to resolve changes with a ! icon in the **Changes** column of the components tree:



**Tip:** If you want to see only the components needing changes, in the **Changes** column, click the down arrow and select the **Open Changes** radio button.

- b. In the right pane, view the details about the needed changes:

Changes (37)	Used By (1)	Screenshot	
Select steps to update in component script:			
<ul style="list-style-type: none"> <li>+ <input type="checkbox"/> Propose items (Ctrl+F11)</li> <li>+ <input type="checkbox"/> Reject document (Ctrl+F10)</li> <li>+ <input type="checkbox"/> Save (Ctrl+S)</li> <li>+ <input type="checkbox"/> Net value</li> <li>+ <input type="checkbox"/> Net value_Z</li> <li>+ <input type="checkbox"/> PO date</li> <li>+ <input type="checkbox"/> Purch.order no.</li> <li>+ <input type="checkbox"/> Ship-to party</li> <li>+ <input type="checkbox"/> Sold-to party</li> <li>+ <input type="checkbox"/> Standard Order (VMI)</li> <li>+ <input type="checkbox"/> mbar</li> <li>+ <input type="checkbox"/> OKCode</li> <li>- <input type="checkbox"/> /</li> </ul>	<ul style="list-style-type: none"> <li><input type="checkbox"/> Add step to component: 'Propose items (Ctrl+F11).Click'</li> <li><input type="checkbox"/> Add step to component: 'Reject document (Ctrl+F10).Click'</li> <li><input type="checkbox"/> Add step to component: 'Save (Ctrl+S).Click'</li> <li><input checked="" type="checkbox"/> Add step to component: 'Net value Set'</li> <li><input checked="" type="checkbox"/> Add step to component: 'Net value_Z Set'</li> <li><input type="checkbox"/> Add step to component: 'PO date Set'</li> <li><input type="checkbox"/> Add step to component: 'Purch.order no. Set'</li> <li><input type="checkbox"/> Add step to component: 'Ship-to party Set'</li> <li><input checked="" type="checkbox"/> Add step to component: 'Sold-to party Set'</li> <li><input type="checkbox"/> Add step to component: 'Standard Order (VMI) Set'</li> <li><input type="checkbox"/> Add step to component: 'mbar Select'</li> <li><input type="checkbox"/> Add step to component: 'OKCode Set'</li> <li><input checked="" type="checkbox"/> This object is not used in any step</li> </ul>		
<input checked="" type="checkbox"/> Changes to object repository were applied			<b>UPDATE STEPS</b>

- c. If you want to accept the proposed changes, in the lower right corner of the pane, click the **Apply Changes** button. UFT applies the changes as suggested.

In addition, the selected component's report row is updated to show that you have resolved the changes:



If steps in your component require updates due to the application change, UFT also updates the Change Detection Report:

CHANGES	USED BY	SCREENSHOT
SELECT STEPS TO UPDATE IN COMPONENT SCRIPT:		
<ul style="list-style-type: none"> <li>+ Appointment Calendar:</li> <li>- Display appointments:_2</li> </ul>	<ul style="list-style-type: none"> <li><input type="checkbox"/> Appointment Calendar::Activate</li> </ul>	

- d. In the right pane, select the checkboxes for the steps that require an update.

- e. In the lower right corner of the pane, click **Update Steps**. UFT automatically updates the steps in your components in the background.

**Note:** If you want to apply the changes in the components for the current test only, you should clear the **Update changes will affect only current test** checkbox. If you do not clear this option, the changes to the components are applied to all tests containing these components.

#### 4. **Save changed components to your ALM project**

After you have updated all the necessary components, in the lower right corner of the Change Detection report, click **Save**. UFT saves the updated components in your ALM project.

**Note:** This process may take some time, depending on the number of updates. Ensure that you do not close UFT or the Change Detection Report while UFT is saving the changes.

#### 5. **Optional - view run results for the test run**

In addition to reporting changes in the application during a test run, the Change Detection Report gives a basic report on the success or failure of the test. You can view:

- Overall run status
- Individual component run status
- Run status for individual steps in the components

If you see that a business component is reporting a **Failed** Status, you can double-click the component name and open the Run Results Viewer to open a defect for this component.

**Note:** If you have set the default report format to **HTML Report** in the **Run Sessions** pane (**Tools > Options > General tab > Run Sessions** node), the double-click function to open a defect is not supported. In order to automatically open a defect from the Change Detection Report, you must set the default report format to **Run Results Viewer Report**.

Note that the Change Detection Report does not provide data on the reasons behind the success or failure of a component. To see this information, you must run the test using the regular **Run** option and view the run results after the test.

# Business Process Testing with the BPT Packaged Apps Kit - Troubleshooting and Limitations

This topic lists troubleshooting and limitations for business process testing with the BPT Packaged Apps Kit:

- You must have the Add-in for SAP Solutions installed and loaded for the current business process test to learn flows and components or run business process tests in Change Detection Mode.
- Learning flows and components is supported only for SAP GUI for Windows and SAP Fiori applications only.
- Running tests and components in Change Detection mode is supported for SAP GUI for Windows applications only.
- If you have an open GUI test, the Record and Run Settings for the test are applicable when you learn a business process test or flow.

**Workaround:** Before starting the Learn process, change the Record and Run Settings to the default.

## Part 11: Appendix

# Appendix A: UFT Terminology Quick Reference

## Relevant for: GUI tests and components and API testing

This chapter lists the relevant testing areas for common UFT elements, as well as references for more details.

Testing type	Includes these testing documents:
<b>GUI tests</b>	<ul style="list-style-type: none"><li>• GUI tests</li><li>• actions</li><li>• function libraries</li></ul>
<b>GUI components</b>	<ul style="list-style-type: none"><li>• keyword GUI components</li><li>• scripted GUI components</li><li>• application areas</li><li>• function libraries</li></ul>
<b>API testing</b>	<ul style="list-style-type: none"><li>• API tests</li><li>• API components</li><li>• user code files</li></ul>

Term (A-Z)	Relevant for:	For details, see:
<b>.NET assembly</b>	API testing	" <a href="#">.NET Assembly Activities</a> " on page 505
<b>accessibility checkpoints</b>	<ul style="list-style-type: none"><li>• GUI tests</li><li>• scripted GUI components</li></ul>	" <a href="#">Accessibility Checkpoints Overview</a> " on page 293
<b>actions</b>	<ul style="list-style-type: none"><li>• GUI tests</li><li>• API testing</li></ul>	<ul style="list-style-type: none"><li>• <b>For GUI testing:</b> "<a href="#">Actions in GUI Testing</a>" on page 104</li><li>• <b>For API testing:</b> "<a href="#">Actions in API Testing - Overview</a>" on page 532</li></ul>
<b>active screen</b>	<ul style="list-style-type: none"><li>• GUI tests</li><li>• scripted GUI components</li></ul>	" <a href="#">Active Screen Pane</a> " on page 59
<b>activity</b>	API testing	" <a href="#">Activity Overview</a> " on page 408
<b>activity repository</b>	API testing	" <a href="#">How to Perform Activity Sharing</a> " on page 506
<b>add-ins</b>	<ul style="list-style-type: none"><li>• GUI tests</li><li>• GUI components</li></ul>	<i>HP Unified Functional Testing Add-ins Guide</i>
<b>ALM project</b>	<ul style="list-style-type: none"><li>• GUI tests</li><li>• GUI components</li><li>• API testing</li></ul>	" <a href="#">ALM Integration Overview</a> " on page 916
<b>application area</b>	GUI components	<ul style="list-style-type: none"><li>• "<a href="#">Application Areas - Overview</a>" on page 1015</li></ul>

Term (A-Z)	Relevant for:	For details, see:
		<ul style="list-style-type: none"> <li>• <a href="#">"Application Area User Interface" on page 1018</a></li> </ul>
<b>array properties</b>	API testing	
<b>automation</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> </ul>	<a href="#">"UFT Automation Object Model Overview" on page 741</a>
<b>bitmap checkpoints</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• scripted GUI components</li> </ul>	<a href="#">"Bitmap Checkpoints Overview" on page 294</a>
<b>bookmarks</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• scripted GUI components</li> <li>• function libraries</li> <li>• API testing</li> </ul>	<a href="#">"Bookmarks Pane" on page 61</a>
<b>breakpoints</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• scripted GUI components</li> <li>• function libraries</li> <li>• API testing</li> </ul>	<ul style="list-style-type: none"> <li>• <a href="#">"Breakpoints " on page 906</a></li> </ul>
<b>business process test</b>	<ul style="list-style-type: none"> <li>• GUI components</li> <li>• API testing</li> </ul>	<a href="#">"Business Process Testing in UFT - Overview" on page 972</a>
<b>canvas</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• API testing</li> </ul>	<a href="#">"The Canvas" on page 62</a>
<b>checkpoint</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> <li>• API testing</li> </ul>	<ul style="list-style-type: none"> <li>• <b>For GUI testing:</b> <a href="#">"Checkpoints Overview" on page 290</a></li> <li>• <b>For API testing:</b> <a href="#">"Checkpoint Validation for Test Steps" on page 408</a></li> </ul>
<b>checkpoint validation</b>	API testing	<a href="#">"Checkpoint Validation for Test Steps" on page 408</a>
<b>classes</b>	API testing	<a href="#">"How to Search for References or Classes in Documents in the Editor (API Testing Only)" on page 638</a>
<b>code completion</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• scripted GUI components</li> <li>• function libraries</li> <li>• API testing</li> </ul>	<a href="#">"Automatic Code Completion for GUI Testing" on page 632</a>
<b>code object</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> <li>• API testing</li> </ul>	
<b>code snippets</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> </ul>	<ul style="list-style-type: none"> <li>• <a href="#">"Automatic Code Completion for GUI Testing" on page 632</a></li> </ul>

Term (A-Z)	Relevant for:	For details, see:
	<ul style="list-style-type: none"> <li>• scripted GUI components</li> </ul>	<ul style="list-style-type: none"> <li>• <a href="#">"How to Use Code Snippets and Templates" on page 635</a></li> </ul>
<b>code templates</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• scripted GUI components</li> <li>• function libraries</li> <li>• API testing</li> </ul>	<ul style="list-style-type: none"> <li>• <a href="#">"How to Use Code Snippets and Templates" on page 635</a></li> </ul>
<b>compile</b>	API testing	<a href="#">"Output Pane" on page 69</a>
<b>conditional statement</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• scripted GUI components</li> </ul>	<a href="#">"Comments, Control-Flow, and Other VBScript Statements " on page 661</a>
<b>conditional steps</b>	API testing'	<a href="#">"Flow Control Activities" on page 437</a>
<b>data driving</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• API testing</li> </ul>	<ul style="list-style-type: none"> <li>• <b>For GUI testing:</b> <a href="#">"Data Driver" on page 364</a></li> <li>• <b>For API testing:</b> <a href="#">"How to Assign Data to API Test/Component Steps" on page 551</a></li> </ul>
<b>data table</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• scripted GUI components</li> <li>• API testing</li> </ul>	<ul style="list-style-type: none"> <li>• <b>For GUI testing:</b> <a href="#">"Data Pane" on page 63</a></li> <li>• <b>For API testing:</b> <a href="#">"Using Data in Your API Test or Component Steps" on page 537</a></li> </ul>
<b>data table parameters</b>	GUI tests	<a href="#">"Data Table Parameters" on page 349</a>
<b>data table property</b>	API testing	<ul style="list-style-type: none"> <li>• <a href="#">"Linking Property Values to a Data Source" on page 539</a></li> <li>• <a href="#">"How to Assign Data to API Test/Component Steps" on page 551</a></li> </ul>
<b>database checkpoints</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• scripted GUI components</li> </ul>	<a href="#">"Database Checkpoints Overview" on page 298</a>
<b>database query</b>	API testing	N/A
<b>Editor</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• scripted GUI components</li> <li>• function libraries</li> <li>• API testing</li> </ul>	<ul style="list-style-type: none"> <li>• <a href="#">"Editing Text and Code Documents" on page 627</a></li> <li>• <a href="#">"Editor User Interface" on page 640</a></li> </ul>
<b>environment variables</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> </ul>	<a href="#">"Environment Variables" on page 329</a>
<b>event handler</b>	API testing	<a href="#">"Writing Code for API Test Events - Overview" on page 755</a>
<b>file content checkpoints</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• scripted GUI components</li> </ul>	<a href="#">"File Content Checkpoints Overview" on page 299</a>

Term (A-Z)	Relevant for:	For details, see:
<b>function library</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> </ul>	<a href="#">"Function Library Overview" on page 684</a>
<b>global data sheet</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• scripted GUI components</li> </ul>	<a href="#">"Data Tables and Sheets in GUI Tests and Components" on page 342</a>
<b>IBM Websphere MQ</b>	API testing	<a href="#">"IBM WebSphere MQ Activities" on page 479</a>
<b>input/output properties</b>	API testing	<a href="#">"Properties Pane" on page 71</a>
<b>Insight objects</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> </ul>	<a href="#">"Identifying Objects Using Insight" on page 266</a>
<b>JMS transport</b>	API testing	<a href="#">"JMS Activities" on page 472</a>
<b>JSON</b>	API testing	<a href="#">"How to Send and Receive a JSON Request for a REST Service" on page 521</a>
<b>Keyword View</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> </ul>	<ul style="list-style-type: none"> <li>• <a href="#">Keyword View Overview</a></li> <li>• <a href="#">"Keyword View User Interface" on page 110</a></li> </ul>
<b>load testing</b>	API testing	<a href="#">"Load Testing Activities" on page 478</a>
<b>log tracking</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> </ul>	<ul style="list-style-type: none"> <li>• <a href="#">"Log Tracking" on page 847</a></li> </ul>
<b>loop statement</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• scripted GUI components</li> </ul>	<a href="#">"Comments, Control-Flow, and Other VBScript Statements " on page 661</a>
<b>maintenance run mode</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> </ul>	<a href="#">"Maintenance Run Mode" on page 135</a>
<b>missing resources</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> <li>• API testing</li> </ul>	<a href="#">"Errors Pane" on page 67</a>
<b>multipart HTTP requests</b>	API testing	<a href="#">"How to Send a Multipart HTTP or REST Service Request" on page 418</a>
<b>negative testing</b>	API testing	<a href="#">"Negative Testing of Web Services" on page 507</a>
<b>object repository</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> </ul>	<a href="#">"Object Repository Window - Overview" on page 186</a>
<b>object spy</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> </ul>	
<b>optional steps</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• scripted GUI</li> </ul>	<a href="#">"Optional Steps" on page 846</a>

Term (A-Z)	Relevant for:	For details, see:
	components	
<b>output value</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• scripted GUI components</li> </ul>	<a href="#">"Output Values Overview" on page 325</a>
<b>parameter (both input and output)</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> </ul>	<ul style="list-style-type: none"> <li>• <a href="#">"Parameterizing Values Overview " on page 338</a></li> <li>• <a href="#">"Output Values Overview" on page 325</a></li> </ul>
<b>property</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> <li>• API testing</li> </ul>	<a href="#">"Properties Pane" on page 71</a>
<b>recording</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> </ul>	<a href="#">"Recording Overview" on page 97</a>
<b>recovery scenario</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> </ul>	<a href="#">"Recovery Scenarios Overview" on page 145</a>
<b>references</b>	API testing	
<b>regular expression</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• scripted GUI components</li> </ul>	<a href="#">"Regular Expressions Overview" on page 372</a>
<b>repository parameters</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> </ul>	<a href="#">"Shared Object Repositories Overview" on page 182</a>
<b>resources</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> </ul>	
<b>REST Service</b>	API testing	<ul style="list-style-type: none"> <li>• <a href="#">"REST Service Activities" on page 503</a></li> <li>• <a href="#">"How to Create a REST Service" on page 515</a></li> </ul>
<b>run results</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> <li>• API testing</li> </ul>	<a href="#">"Using Run Results" on page 853</a>
<b>run session</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> <li>• API testing</li> </ul>	<a href="#">"Running Tests and Components - Overview " on page 832</a>
<b>run-time object</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> </ul>	<a href="#">"How UFT Applies the Test Object Model Concept" on page 166</a>
<b>SOAP</b>	API testing	<a href="#">"Web Service Scenario Overview" on page 583</a>
<b>solution</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> <li>• API testing</li> </ul>	<a href="#">"Solution Explorer Pane" on page 74</a>

Term (A-Z)	Relevant for:	For details, see:
<b>statement completion</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• scripted GUI components</li> <li>• function libraries</li> <li>• API testing</li> </ul>	<a href="#">"Statement Completion in the Editor" on page 627</a>
<b>step generator</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• scripted GUI components</li> </ul>	
<b>syntax errors</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> <li>• API testing</li> </ul>	<a href="#">"Errors Pane" on page 67</a>
<b>system monitor</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> </ul>	
<b>Test Batch Runner</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• API testing</li> </ul>	<a href="#">"Test Batch Runner Overview" on page 842</a>
<b>test loops</b>	API testing	<a href="#">"Flow Control Activities" on page 437</a>
<b>test object</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> </ul>	<a href="#">"Test Object Model - Overview" on page 164</a>
<b>test variables</b>	API testing	<a href="#">"How to Define API Test Properties or User/System Variables" on page 558</a>
<b>text checkpoint</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> </ul>	<a href="#">"Checking Text Overview " on page 301</a>
<b>text area checkpoints</b>	<ul style="list-style-type: none"> <li>• scripted GUI components</li> </ul>	
<b>text recognition</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> </ul>	<ul style="list-style-type: none"> <li>• <a href="#">"Text Recognition Overview" on page 301</a></li> </ul>
<b>TODO comments</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• scripted GUI components</li> <li>• function libraries</li> <li>• API testing</li> </ul>	<a href="#">"Tasks Pane" on page 75</a>
<b>Toolbox</b>	<ul style="list-style-type: none"> <li>• GUI actions</li> <li>• GUI components</li> <li>• function libraries</li> <li>• API testing</li> <li>Business process tests and flows</li> </ul>	<a href="#">"Toolbox Pane" on page 77</a>
<b>Update Run Mode</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> </ul>	<a href="#">"Update Run Mode " on page 138</a>

Term (A-Z)	Relevant for:	For details, see:
	<ul style="list-style-type: none"> <li>• GUI components</li> </ul>	
<b>Update Service/Assembly</b>	API testing	<a href="#">"Updating Services" on page 571</a>
<b>user code files</b>	API testing	<a href="#">"Writing Code for API Test Events - Overview" on page 755</a>
<b>Userlogger</b>	API testing	<a href="#">"UserLogger Object" on page 814</a>
<b>virtual object</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• scripted GUI components</li> </ul>	<a href="#">"Virtual Objects - Overview" on page 287</a>
<b>virtualized services</b>	API testing	<a href="#">"Running Tests with Virtualized Services - Overview" on page 877</a>
<b>Web services</b>	API testing	<a href="#">"Web Service Activities " on page 503</a>
<b>WSDL</b>	API testing	<ul style="list-style-type: none"> <li>• <a href="#">"Web Service Activities " on page 503</a></li> <li>• <a href="#">"How to Import a WSDL-Based Web Service" on page 512</a></li> </ul>
<b>XML checkpoints</b>	<ul style="list-style-type: none"> <li>• GUI tests</li> <li>• GUI components</li> </ul>	<a href="#">"XML Checkpoints Overview" on page 307</a>
<b>XPath Checkpoints</b>	API testing	<a href="#">"XPath Checkpoints" on page 409</a>

# Appendix B: Developing Custom Comparers for Bitmap Checkpoints (GUI Testing)

## **Relevant for: GUI tests and components**

This appendix is intended for COM programmers who want to customize the algorithm used to compare bitmaps in bitmap checkpoints.

This chapter includes:

- [About Developing Custom Comparers for Bitmap Checkpoints .....](#) 1093
- [Custom Comparer for Images Whose Location Changes in the Application - Use-Case Scenario ..](#) 1093
- [Custom Bitmap Comparer Development .....](#) 1094
- [How to Develop a Custom Comparer .....](#) 1095
- [How to Implement the Bitmap Comparer Interfaces .....](#) 1097
- [How to Install Your Custom Comparer and Register it to UFT .....](#) 1100
- [How to Use the Bitmap Checkpoint Custom Comparer Samples .....](#) 1102
- [How to Develop a Custom Comparer - Tutorial .....](#) 1104
- [The Bitmap Checkpoint Comparer Interfaces .....](#) 1113
  - [The IVerifyBitmap Interface .....](#) 1113
  - [The IBitmapCompareConfiguration Interface .....](#) 1114

# About Developing Custom Comparers for Bitmap Checkpoints

## **Relevant for: GUI tests and components**

A custom comparer is a COM object that you develop to run the bitmap comparison in a bitmap checkpoint according to a specific algorithm. This enables you to create bitmap checkpoints that perform the comparison according to your needs.

By default, a bitmap checkpoint in UFT compares the actual and expected bitmaps pixel by pixel and fails if there are any differences. UFT provides various bitmap checkpoint configuration options that enable you to refine the bitmap comparison and make it more flexible. For example, you can define tolerance levels, or you can instruct UFT not to compare complete images, but rather compare selected areas within them, or to locate a specific image within an object in your application. For more details, see "[Bitmap Checkpoints Overview](#)" on page 294.

If you need to further customize the way bitmaps are compared in checkpoints, you can develop custom comparers and install and register them on the UFT computer. A UFT user can then choose to use a custom comparer to perform the comparison in a bitmap checkpoint (on a per checkpoint basis).

The COM object that you develop must implement interfaces that UFT provides in a type library, and register to the component category that UFT defines for bitmap comparers. The type library (`BitmapComparer.tlb`) and the category ID (defined in `ComponentCategory.h`) are available in `<UFT installation folder>\dat\BitmapCPCustomization`.

When a UFT user creates or edits a bitmap checkpoint, UFT displays any registered custom comparers in the advanced settings in the Bitmap Checkpoint Properties dialog box (in addition to the UFT default comparer). The user can then select a comparer according to the testing requirements of the specific application or bitmap being tested. For more details about using custom comparers in UFT, see "[Custom Comparers](#)" on page 297.

You can find an example of a situation where developing a custom comparer enhanced the use of bitmap checkpoints, in "[Custom Comparer for Images Whose Location Changes in the Application - Use-Case Scenario](#)" below.

## Custom Comparer for Images Whose Location Changes in the Application - Use-Case Scenario

## **Relevant for: GUI tests and components**

Ben is a quality assurance engineer who is experienced in using UFT, and often uses bitmap checkpoints to test the appearance of different icons or pictures in the user interface he is testing. He does not have a programming background.

Joanne is a software engineer who is experienced in image processing and is familiar with COM programming.

When Ben began testing the user interface of a furniture purchasing application, he created a bitmap checkpoint to test that the pictures of the items on sale were displayed properly. In the checkpoint, he captured an image of the pane in the application that contained the pictures he wanted to test. Ben found that the bitmap checkpoint often failed, even though the graphic images displayed in the application during the run seemed identical to the ones he had captured when creating the checkpoint.

Ben reviewed the actual, expected, and difference bitmaps displayed in the run results. He also took a closer look at the application's user interface. The application contained three panes. The left pane displayed general information, the middle pane displayed the pictures of the items on sale, and the right pane displayed the corresponding list of items and relevant details. Ben found that depending on the information displayed in the left pane, the images in the middle pane sometimes shifted slightly one way or the other within the pane. While the images themselves were still identical, their changed location was causing the bitmap checkpoint to fail.

Ben did not want to use pixel tolerance to address this issue because he wanted the checkpoint to fail when the pixels within the images themselves were not identical.

When Ben mentioned his problem to a co-worker, she suggested that developing a custom comparer for his bitmap checkpoints could solve the problem, and referred him to Joanne. Joanne developed a custom comparer that would accept as input the number of pixels that the images should be allowed to shift without failing the checkpoint. The bitmap comparison that Joanne designed would pass the checkpoint only if the images were identical and they had all shifted by the same number of pixels. This way, Ben knew that his checkpoint would still catch incorrect images and cases where the application's interface looked bad because the images were no longer aligned.

Ben installed and registered the custom comparer on his UFT computer, and then selected the new custom comparer for his bitmap checkpoint. After some experimenting, he found the optimal number of pixels to enter in the configuration string, so that significant changes in the application's interface were detected, but insignificant shifting of the images did not cause the checkpoint to fail.

After Ben successfully used this custom comparer for a while, his company decided to install and register it on all of the UFT computers. The custom comparer would now be available to everyone in the quality assurance team to use for similar situations.

## Custom Bitmap Comparer Development

### **Relevant for: GUI tests and components**

To develop a custom comparer, you create a COM object that implements the UFT [bitmap checkpoint comparer interfaces](#) (described on page 1113) to perform the following:

- Accept input from UFT and perform the bitmap comparison.
- Provide comparison results to UFT.
- (Optionally) Provide information that UFT can display in the advanced settings in the Bitmap Checkpoint Properties dialog box when a user creates or edits a bitmap checkpoint.

Custom comparers run within the UFT context. You must therefore exercise care when developing your custom comparer, as its behavior and performance will affect the behavior and performance of UFT.

For UFT to recognize the custom comparer, it must be registered to the component category that UFT defines for bitmap comparers. Depending on how you implement your custom comparer, you can design the comparer to register itself when it is installed, or you can provide an additional program that needs to be run at the time of installation. For details, see "[How to Install Your Custom Comparer and Register it to UFT](#)" on page 1100.

Perform the tutorial in "[How to Develop a Custom Comparer - Tutorial](#)" on page 1104 to learn how to create and use a custom comparer. You can then create your own custom comparers in much the same way. For task details, see "[How to Develop a Custom Comparer](#)" below.

In addition to the tutorial, UFT provides source files that implement a sample custom comparer in different languages. The source files are provided in C++ and in Visual Basic. Both projects generate a similar custom comparer.

You can study the samples to help you learn about developing custom comparers for UFT bitmap checkpoints, or use them as a reference or template when you develop your own custom comparers. For details, see "[How to Use the Bitmap Checkpoint Custom Comparer Samples](#)" on page 1102.

## How to Develop a Custom Comparer

### **Relevant for: GUI tests and components**

This task describes the process for developing a custom bitmap comparer.

**Tip:** To practice performing this task, see "[How to Develop a Custom Comparer - Tutorial](#)" on page 1104.

This task includes the following steps:

- "[Prerequisites](#)" on page 1102
- "[Develop the custom comparer COM object](#)" below
- "[Prepare the custom comparer installation - optional](#)" on the next page
- "[Install the custom comparer](#)" on the next page
- "[Test the custom comparer](#)" on the next page

### **1. Prerequisites**

- Knowledge of image processing
- Experience in developing COM objects

### **2. Develop the custom comparer COM object**

- a. Create the custom comparer COM object. You can use any language and development

environment that supports creating COM objects.

**Note:** Depending on the language that you use for development, you might be able to specify the custom comparer name when you create the COM object. Otherwise the name can be specified on the UFT computer after registering the object. For details, see "[Set the Custom Comparer Name - Optional](#)" on page 1101.

- b. Program your COM object to implement the custom comparer interfaces. For details, see "[How to Implement the Bitmap Comparer Interfaces](#)" on the next page.

### 3. **Prepare the custom comparer installation - optional**

Your custom comparer might need to be installed on more than one computer. You can create a program that automatically performs the steps necessary to install and register the comparer and its documentation on those computers.

For details on the steps that such a program needs to perform, see "[How to Install Your Custom Comparer and Register it to UFT](#)" on page 1100.

For example, when you design your custom comparer installation, you must ensure that when it is installed on the UFT computer, it is also registered to the component category for UFT bitmap comparers. This can be achieved in different ways, such as:

- If you develop your custom comparer in C++ using Microsoft Visual Studio, you can modify the **DllRegisterServer** and **DllUnregisterServer** methods to handle this registration. These methods are called when you run a .dll using the regsvr32.exe program. You can see an example of this type of implementation in "[How to Develop a Custom Comparer - Tutorial](#)" on page 1104.
- If you develop your custom comparer in an environment that does not enable you to modify the registration methods, you can add an additional program that handles this registration and instruct users who install the custom comparer to run this program as well. You can see an example of this type of implementation in the Visual Basic sample custom comparer that UFT provides. For more details, see "[How to Use the Bitmap Checkpoint Custom Comparer Samples](#)" on page 1102.

### 4. **Install the custom comparer**

On the computer where you want to use the custom comparer, do one of the following:

- Run the installation program that automatically installs and registers the comparer.
- Manually install and register the custom comparer. For details, see "[How to Install Your Custom Comparer and Register it to UFT](#)" on page 1100.

### 5. **Test the custom comparer**

Create bitmap checkpoint steps in a test or component in UFT. In the Bitmap Checkpoint Properties dialog box, click **Advanced settings**, select your custom comparer, and use it to perform bitmap checkpoints and check that they perform correctly.

**Tip:** By default, UFT displays expected, actual, and difference bitmaps in the run results only for checkpoints that fail. When you test your custom comparer on UFT, you might want to see the expected, actual, and difference bitmaps in the run results even for bitmap checkpoints that pass. To configure this, select **Tools > Options > GUI Testing tab > Screen Capture** node in UFT and set the **Save still image captures to results** option to **Always**.

## How to Implement the Bitmap Comparer Interfaces

### Relevant for: GUI tests and components

This task describes how to implement the bitmap comparer interfaces so that your custom comparer COM object performs the following:

- Accepts bitmaps and compares them
- Provides comparison results to UFT
- Provides information for the advanced settings in the Bitmap Checkpoint Properties dialog box

**Note:** This task is part of a higher-level task. For details, see "[How to Develop a Custom Comparer](#)" on page 1095.

This task includes the following steps:

- "[Prerequisite - Reference the type library](#)" below
- "[Implement the CompareBitmaps method to accept input and compare bitmaps](#)" on the next page
- "[Implement the CompareBitmaps method to return the comparison results to UFT](#)" on the next page
- "[Implement IBitmapCompareConfiguration to provide information for the Bitmap Checkpoint Properties dialog box](#)" on page 1099

### Prerequisite - Reference the type library

In the COM object that you develop, reference the type library that UFT provides (located in <UFT installation folder>\dat\BitmapCPCustomization\BitmapComparer.tlb)

## Implement the CompareBitmaps method to accept input and compare bitmaps

UFT calls the ["The CompareBitmaps Method" on page 1113](#) method in the **IVerifyBitmap** interface (described on page 1113) to pass the expected and actual bitmaps to the custom comparer for comparison.

### Method syntax:

```
HRESULT CompareBitmaps ([in] IPictureDisp* pExpected,
 [in] IPictureDisp* pActual,
 [in] BSTR bstrConfiguration,
 [out] BSTR* pbstrLog,
 [out] IPictureDisp** ppDiff,
 [out, retval] VARIANT_BOOL* pbMatch);
```

Implement the ["The CompareBitmaps Method" on page 1113](#) method to perform the following:

- Accept and compare two bitmaps according to a predefined algorithm that you define based on the testing requirements.
- Accept a text string that can contain configuration information provided by the UFT user (in the Bitmap Checkpoint Properties dialog box, Advanced settings), and use it in the comparison. For example, the string could contain tolerance specifications, acceptable deviations in size or location of the image, or any other information that you want to affect the comparison.

The string can have any format you choose (XML, comma separated, INI file style, and so on). Make sure that the documentation you provide for the custom comparer describes the format. The configuration input that the UFT user enters in the advanced settings in the advanced settings in the Bitmap Checkpoint Properties dialog box must conform to this format.

## Implement the CompareBitmaps method to return the comparison results to UFT

UFT displays the results of bitmap checkpoints in the run results.

When you implement the ["The CompareBitmaps Method" on page 1113](#) method in the **IVerifyBitmap** interface (described on page 1113) to compare the bitmaps, you must also return the following information:

- Whether the bitmaps match and the checkpoint should pass.
- A text string that UFT displays in the run results.

The purpose of this string is to provide information about the comparison to the UFT user, but while you develop and test your comparer, you can use this string for debugging purposes as well.

- A bitmap that visually represents the difference between the actual and expected bitmaps.

The purpose of this bitmap is to help the UFT user understand why the checkpoint failed. The custom comparer can create this bitmap using any visualization approach you choose. For example, the default UFT comparer creates a black-and-white bitmap containing a black pixel for every pixel that is different in the two images.

## Implement IBitmapCompareConfiguration to provide information for the Bitmap Checkpoint Properties dialog box

When a UFT user selects a custom comparer in the advanced settings of the Bitmap Checkpoint Properties dialog box, UFT displays an **input** text box, and, optionally, a link to documentation provided for the custom comparer. For details, see "[Custom Comparers](#)" on page 297.

To support these options, you can implement the **IBitmapCompareConfiguration** interface (described on page 1114) to provide the necessary information for the dialog box.

- Implement the "[The GetDefaultConfigurationString Method](#)" on page 1115 method to return the default configuration string for your custom comparer.

### Method syntax:

```
HRESULT GetDefaultConfigurationString ([out, retval] BSTR*
pbstrConfiguration);
```

UFT displays this string in the **Input** box in the advanced settings in the Bitmap Checkpoint Properties dialog box.

The format of this string must be the same as the format of the configuration string that the comparer expects as input.

- Implement the **GetHelpFilename** method to return a path to the documentation about your custom comparer. A UFT user can then access the documentation from the advanced settings in the Bitmap Checkpoint Properties dialog box.

### Method syntax:

```
HRESULT GetHelpFilename ([out, retval] BSTR* pbstrFilename);
```

The documentation can be in any format that you choose. UFT opens the documentation using the program associated with the provided file type on the user's computer. Therefore, you should provide the documentation in a format for which you expect the UFT user to have the necessary program.

The documentation should provide the UFT user with the following information:

- The type of comparison the custom comparer performs (to enable the user to determine when to use it to run a bitmap checkpoint).
- The required format for the configuration string and the possible values it can contain.
- An explanation of the comparison result information that is displayed in the run results (text string and difference bitmap).

# How to Install Your Custom Comparer and Register it to UFT

## Relevant for: GUI tests and components

The custom comparer must be installed and registered on any computer that runs a test or component with a bitmap checkpoint using the custom comparer.

This task describes how to install the custom comparer on a UFT computer and register it to UFT.

- When you develop a custom comparer, you can create a program that automatically performs the steps in this task. If you choose not to create an installation program, review these steps and make sure to provide your users with all of the required files and information.
- When you install a custom comparer that you did not develop, you may need additional information from the developer to perform the steps in this task. Alternatively, you may receive an installation program from the developer that performs this task automatically.

**Note:** This task is part of a higher-level task. For details, see "[How to Develop a Custom Comparer](#)" on page 1095.

This task includes the following steps:

- "Prerequisites" below
- "Install the custom comparer COM object on the UFT computer" below
- "Register the custom comparer to the component category for UFT bitmap comparers" on the next page
- "Place the custom comparer documentation in the correct location" on the next page
- "Set the Custom Comparer Name - Optional" on the next page
- "Results" on page 1102

## Prerequisites

1. More than one custom comparer can be installed and registered on the same UFT computer. However, before installing and registering a new version of a specific custom comparer, unregister the existing comparer.
2. A custom comparer .dll is created using a specific development environment version. Make sure that the computer on which this .dll runs has the corresponding runtime environment installed.

## Install the custom comparer COM object on the UFT computer

For example, you can do this by double-clicking the .dll or running the .dll using the regsvr32.exe program.

## Register the custom comparer to the component category for UFT bitmap comparers

Register the component category ID for UFT bitmap comparers, **CATID\_QTPBitmapComparers**, as a registry key under the COM object's **HKEY\_CLASSES\_ROOT\CLSID\<Object's CLSID>\Implemented Categories** key.

**Note:** When UFT is installed, it adds this component category ID as a registry key under the **HKEY\_CLASSES\_ROOT\Component Categories** key. The component category ID is defined in <UFT installation folder>\dat\BitmapCPCustomization\ComponentCategory.h.

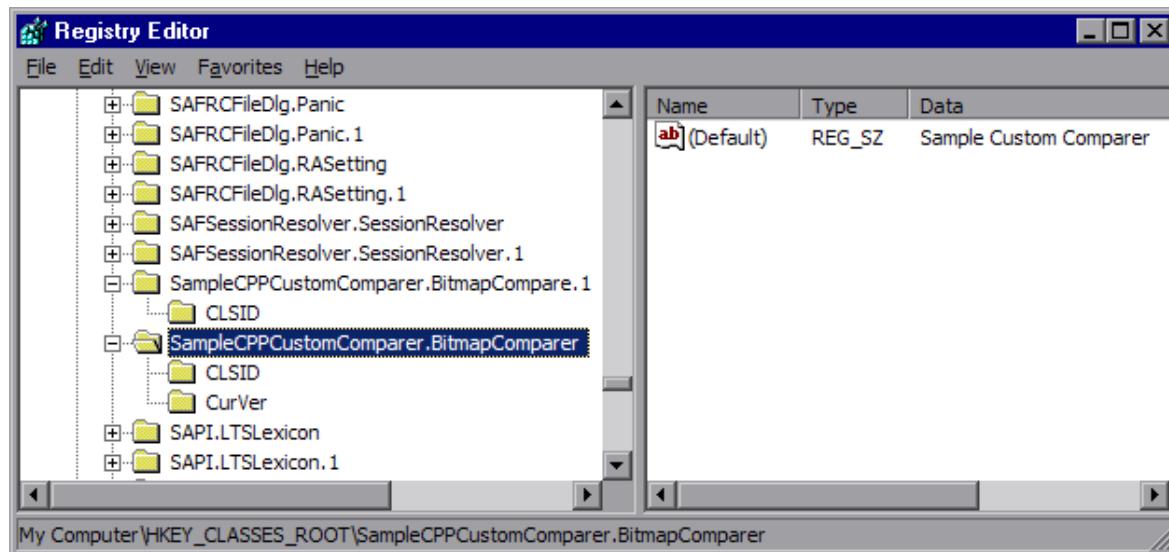
## Place the custom comparer documentation in the correct location

The Bitmap Checkpoint Properties dialog box in UFT can display documentation about the custom comparer, if such documentation is provided.

Place the custom comparer documentation in the location specified in the custom comparer object's **GetHelpFilename** method.

## Set the Custom Comparer Name - Optional

UFT displays the name of the custom comparer in the advanced settings in the Bitmap Checkpoint Properties dialog box and in the run results. The name that UFT uses is the value (in the registry) of the default property of the custom comparer ProgID key under the **HKEY\_CLASSES\_ROOT** key. For example, in the image below, the name of the custom comparer is **Sample Custom Comparer**.



- In some environments you set the name while developing the object. For example if the custom comparer is developed in C++ using Microsoft Visual Studio, this name can be specified during development in the **Type** box in the ATL Simple Object Wizard.

- In other environments, you can set or customize the name as part of the installation or registration process on each computer. For example, if the custom comparer is developed in Visual Basic, this registry value is automatically set to the COM object's ProgID. If you want to modify the custom comparer name, you can edit it manually in the registry after the comparer is installed, or design the program that performs the installation and registration to edit this value as well.

## Results

In the advanced settings in the Bitmap Checkpoint Properties dialog box, UFT displays the comparer you installed and registered, along with all of the available custom comparers, and the UFT default comparer. You can then select the appropriate comparer to use for each bitmap checkpoint.

# How to Use the Bitmap Checkpoint Custom Comparer Samples

### Relevant for: GUI tests and components

This task describes how to generate the sample custom comparer, and then register it and work with it.

This task includes the following steps:

- "Prerequisites" below
- "Generate the sample comparer" below
- "Install the custom comparer on a UFT computer" on the next page
- "Register the custom comparer to the component category for UFT bitmap comparers" on the next page
- "Study the custom comparer functionality" on the next page

#### 1. Prerequisites

Decide whether you want to use the sample C++ project or the Visual Basic one.

The samples are located under <UFT installation folder>\samples\BitmapCPSample.

#### 2. Generate the sample comparer

- a. To open the sample project, do one of the following:
  - To open the C++ project, use Microsoft Visual Studio 2005 or later.
  - To open the Visual Basic project, use Microsoft Visual Studio 6.0.
- b. Compile the custom comparer, to build the .dll.

**Note:** If you are using Microsoft Visual Studio 2010 to compile the C++ project, make the following modification before compiling: Open the stdafx.h file in <UFT installation folder>\samples\BitmapCPSample\CPPCustomComparer, locate the line #define \_WIN32\_WINNT 0x0400 and change it to #define \_WIN32\_WINNT

0x0501.

### 3. Install the custom comparer on a UFT computer

Run the custom comparer using the regsvr32.exe program to install it on the computer.

### 4. Register the custom comparer to the component category for UFT bitmap comparers

- If you are using the C++ sample project:

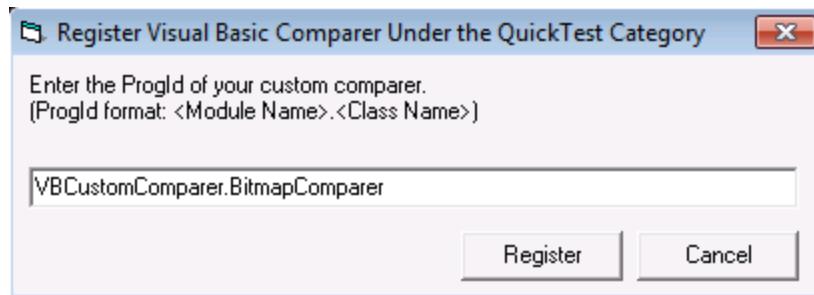
The C++ sample sources implement registering the custom comparer to UFT in the **DllRegisterServer** and **DllUnregisterServer** methods. Therefore, if you used the C++ project to create the .dll, running the .dll (in the previous step) will also register the custom comparer.

**Note:** The name displayed for the custom comparer in UFT will be **Custom QTP Bitmap Comparer**.

- If you are using the Visual Basic sample project:

The Visual Basic sample project does not implement this registration. Therefore, the Visual Basic sample also includes an additional tool that you must run after installing the custom comparer, to register the custom comparer to the component category for UFT bitmap comparers. For more details, see "[How to Install Your Custom Comparer and Register it to UFT](#)" on page 1100.

You can run the Visual Basic Comparer Registration Tool from <UFT installation folder>\samples\BitmapCPSample\VBCustomComparer\RegisterCategory.exe.



In the dialog box that opens, enter the ProgID of the custom comparer and click **Register**.

**Note:** The name displayed for the custom comparer in UFT will be its ProgID—**VBCustomComparer.BitmapComparer**. For details on modifying this name, see "[Set the Custom Comparer Name - Optional](#)" on page 1101.

### 5. Study the custom comparer functionality

Create bitmap checkpoint steps in a test or component in UFT. In the advanced settings in the Bitmap Checkpoint Properties dialog box, you can select the sample custom comparer and use it to

perform bitmap checkpoints.

- a. Note the customized information that is displayed in the dialog box when you select the custom comparer.
  - The default configuration string that the sample comparer returns (and UFT displays in the Bitmap Checkpoint Properties dialog box) is MaxSurfAreaDiff=140000.
  - The documentation provided with this sample comparer (and opened from the Bitmap Checkpoint Properties dialog box) is the SampleComparerDetails.txt text file located in <UFT installation folder>\samples\BitmapCPSample\CPPCustomComparer.
- b. Run bitmap checkpoints to test the behavior of the sample comparer. For example, you can run checkpoints on the Windows Calculator application, alternately setting the Calculator view to **Standard** or **Scientific**, to obtain different size bitmaps for the same object.

The sample custom comparer does not compare the content of the actual and expected bitmaps. It compares the total number of pixels they contain. For configuration input, this comparer expects a string that defines the MaxSurfAreaDiff parameter. The comparer fails the checkpoint if the difference in total number of pixels is greater than the number defined for MaxSurfAreaDiff.

- c. View the results of your checkpoint in the run results.

This sample bitmap custom comparer returns the actual bitmap as the difference bitmap. In addition, it provides a text string that specifies the difference in total number of pixels. UFT displays this string in the run results.

## How to Develop a Custom Comparer - Tutorial

### Relevant for: GUI tests and components

This tutorial walks you step-by-step through the process of creating a custom comparer in C++ using Microsoft Visual Studio. The custom comparer you create is similar to the sample custom comparer provided with UFT. You can create your own custom comparers in a similar way. For details about the sample custom comparer, see "[How to Use the Bitmap Checkpoint Custom Comparer Samples](#)" on page [1102](#).

**Note:** For a task related to this tutorial, see "[How to Develop a Custom Comparer](#)" on page [1095](#).

By following the instructions in this section, you create a COM object that:

- Implements the **CompareBitmaps** method to receive two bitmaps to compare and a configuration string, compare the (size of) the two bitmaps, and return the necessary results.
- Implements the **GetDefaultConfigurationString** method and the **GetHelpFilename** method, to return the information that UFT displays in the advanced settings in the Bitmap Checkpoint Properties dialog box.
- Registers to the component category for UFT bitmap comparers.

When the design of your custom comparer is complete, you can install and register it and use it in UFT to run a bitmap checkpoint.

**Note:** Depending on the version of Microsoft Visual Studio that you use to perform the tutorial, the command names may be different.

This tutorial includes the following steps:

- "Create a new ATL project—[SampleCPPCustomComparer](#)" below
- "Create a new class—[CBitmapComparer](#)" below
- "Define that the [CBitmapComparer class implements the bitmap checkpoint comparer interfaces](#)" below
- "Move the function bodies for the bitmap checkpoint comparer interface methods from [BitmapComparer.h](#) to [BitmapComparer.cpp](#)" on the next page
- "Implement the bitmap checkpoint comparer interface methods to customize the bitmap checkpoint as required" on page 1108
- "Design your custom comparer to register to the component category for UFT bitmap comparers" on page 1110
- "Compile your DLL and run it using the [regsvr32.exe](#) program" on page 1111
- "Test your custom comparer by using it for bitmap checkpoints in UFT" on page 1111

### 1. [Create a new ATL project—SampleCPPCustomComparer](#)

- a. In Microsoft Visual Studio, select **New > Project**. The New Project dialog box opens.
- b. Select the **ATL Project** template, enter [SampleCPPCustomComparer](#) in the **Name** box for the project, and click **OK**. The New ATL Project wizard opens.
- c. In **Application Settings**, make sure that the **Attributed** option is not selected, and click **Finish**.

### 2. [Create a new class—CBitmapComparer](#)

- a. In the class view, select the [SampleCPPCustomComparer](#) project, right-click, and select **Add > Class**. The Add Class dialog box opens.
- b. Select **ATL Simple Object** and click **Add**. The ATL Simple Object Wizard opens.
- c. In the **Short name** box, enter [BitmapComparer](#). The wizard uses this name to create the names of the class, the interface, and the files that it creates.
- d. In the **Type** box, enter [Sample Custom Comparer](#). This is the custom comparer name that UFT will display in the advanced settings in the Bitmap Checkpoint Properties dialog box and in the run results. For details, see "[Set the Custom Comparer Name - Optional](#)" on page 1101.
- e. Click **Finish**. The wizard creates the necessary files for the class that you added, including .cpp and .h files with implementation of [CBitmapComparer](#) class.

### 3. [Define that the CBitmapComparer class implements the bitmap checkpoint](#)

## comparer interfaces

- a. In the class view, select **CBitmapComparer**, right-click, and select **Add > Implement Interface**. The Implement Interface wizard opens.
- b. In the **Implement interface from** option, select **File**. Browse to or enter the location of the UFT bitmap checkpoint comparer type library. The type library is located in: <UFT installation folder>\dat\BitmapCPCustomization\BitmapComparer.tlb.

The wizard displays the interfaces available in the selected type library, **IBitmapCompareConfiguration** and **IVerifyBitmap**.

- c. Add both interfaces to the list of interfaces to implement, and click **Finish**.

In the **BitmapComparer.h** file, the wizard adds the declarations, classes, and method stubs that are necessary to implement the interfaces. In subsequent steps you will need to add implementation to these method stubs.

**Note:** In Microsoft Visual Studio 2005, the wizard generates the signature for the **CompareBitmaps** method in the **IVerifyBitmap** interface incorrectly. To enable your project to compile correctly, manually change the type of the last argument (**pbMatch**) from **BOOL\*** to **VARIANT\_BOOL\***.

### 4. Move the function bodies for the bitmap checkpoint comparer interface methods from **BitmapComparer.h** to **BitmapComparer.cpp**

- a. Open the **BitmapComparer.h** and **BitmapComparer.cpp** files.
- b. In **BitmapComparer.h**, create declarations for the bitmap checkpoint comparer interface methods (based on the function bodies that the wizard created): **CompareBitmaps**, **GetDefaultConfigurationString**, and **GetHelpFilename**.
- c. Move the function bodies that the wizard created for the bitmap checkpoint comparer interface methods from the **BitmapComparer.h** file to the **BitmapComparer.cpp** file.

At the end of this step, **BitmapComparer.cpp** and **BitmapComparer.h** should contain the following code:

```
// BitmapComparer.cpp : Implementation of CBitmapComparer
#include "stdafx.h"
#include "BitmapComparer.h"
// CBitmapComparer
// IBitmapCompareConfiguration Methods
STDMETHODIMP CBitmapComparer::GetDefaultConfigurationString
(BSTR * pbstrConfiguration)
{
 return E_NOTIMPL;
}
STDMETHODIMP CBitmapComparer::GetHelpFilename(BSTR * pbstrFilename)
```

```
{
 return E_NOTIMPL;
}
// IVerifyBitmap Methods
STDMETHODIMP CBitmapComparer::CompareBitmaps
 (IPictureDisp * pExpected, IPictureDisp *
pActual,
 BSTR bstrConfiguration, BSTR * pbstrLog,
 IPictureDisp ** ppDiff, VARIANT_BOOL * pbMatch)
{
 return E_NOTIMPL;
}

// BitmapComparer.h : Declaration of the CBitmapComparer
#pragma once
#include "resource.h" // main symbols
#include "SampleCPPCustomComparer.h"
// CBitmapComparer
class ATL_NO_VTABLE CBitmapComparer :
public CComObjectRootEx<CComSingleThreadModel>,
public CComCoClass<CBitmapComparer, &CLSID_BitmapComparer>,
public IDispatchImpl<IBitmapComparer, &IID_IBitmapComparer,
 &LIBID_SampleCustomComparerLib, /*wMajor = */ 1,
/*wMinor =*/ 0>,
 public IDispatchImpl<IBitmapCompareConfiguration,
 &__uuidof(IBitmapCompareConfiguration),
 &LIBID_BitmapComparerLib, /* wMajor = */ 1, /*wMinor
= */ 0>,
 public IDispatchImpl<IVerifyBitmap, &__uuidof(IVerifyBitmap),
 &LIBID_BitmapComparerLib, /* wMajor = */ 1, /*wMinor
= */ 0>
{
public:
 CBitmapComparer()
 {
 }
 DECLARE_REGISTRY_RESOURCEID(IDR_BITMAPCOMPARER)
 BEGIN_COM_MAP(CBitmapComparer)
 COM_INTERFACE_ENTRY(IBitmapComparer)
 COM_INTERFACE_ENTRY2(IDispatch, IBitmapCompareConfiguration)
 COM_INTERFACE_ENTRY(IBitmapCompareConfiguration)
 COM_INTERFACE_ENTRY(IVerifyBitmap)
 END_COM_MAP()
 DECLARE_PROTECT_FINAL_CONSTRUCT()
 HRESULT FinalConstruct()
 {
 return S_OK;
 }
```

```

 }
 void FinalRelease() {}
 // IBitmapCompareConfiguration Methods
public:
 STDMETHOD(GetDefaultConfigurationString)(BSTR *
pbstrConfiguration);
 STDMETHOD(GetHelpFilename)(BSTR * pbstrFilename);
 // IVerifyBitmap Methods
public:
 STDMETHOD(CompareBitmaps)(IPictureDisp * pExpected,
 IPictureDisp * pActual, BSTR bstrConfiguration,
BSTR * pbstrLog,
 IPictureDisp ** ppDiff, VARIANT_BOOL * pbMatch);
};

OBJECT_ENTRY_AUTO(__uuidof(BitmapComparer), CBitmapComparer)

```

## 5. Implement the bitmap checkpoint comparer interface methods to customize the bitmap checkpoint as required

In this tutorial, you implement a custom comparer similar to the sample custom comparer provided with UFT. For details about the sample custom comparer, see ["How to Use the Bitmap Checkpoint Custom Comparer Samples" on page 1102](#).

When you create your own custom comparers, this is the step during which you design the custom comparer logic. You define the configuration input that it can receive, the algorithm that it uses to compare the bitmaps, and the output that it provides.

In the `BitmapComparer.cpp` file, add `#include <atlstr.h>`, and implement the bitmap checkpoint comparer interface methods as follows:

- The **GetDefaultConfigurationString** method:

```

STDMETHODIMP CBitmapComparer::GetDefaultConfigurationString
(BSTR * pbstrConfiguration)
{
 CComBSTR bsConfig("MaxSurfAreaDiff=140000");
 *pbstrConfiguration = bsConfig.Detach();
 return S_OK;
}

```

- The **GetHelpFilename** method:

```

STDMETHODIMP CBitmapComparer::GetHelpFilename(BSTR * pbstrFilename)
{
 CComBSTR bsFilename ("..\samples\\BitmapCPSample\\")

```

```

 CPPCustomComparer\\SampleComparerDetails.txt");
 *pbstrFilename = bsFilename.Detach();
 return S_OK;
}

```

**Note:** When the GetHelpFilename method returns a relative path, UFT searches for this path relative to <UFT installation folder>\bin. The implementation above instructs UFT to use the documentation file provided with the CPP sample custom comparer.

- The **CompareBitmaps** method:

```

STDMETHODIMP CBitmapComparer::CompareBitmaps
 (IPictureDisp * pExpected,
 IPictureDisp * pActual,
 BSTR bstrConfiguration, BSTR *
 pbstrLog,
 IPictureDisp ** ppDiff, VARIANT_BOOL
 * pbMatch)
{
 HRESULT hr = S_OK;
 if (!pExpected || !pActual)
 return S_FALSE;
 CComQIPtr<IPicture> picExp(pExpected);
 CComQIPtr<IPicture> picAct(pActual);
 // Try to get HBITMAP from IPicture
 HBITMAP HbmpExp, HbmpAct;
 hr = picExp->get_Handle((OLE_HANDLE*)&HbmpExp);
 if (FAILED(hr))
 return hr;
 hr = picAct->get_Handle((OLE_HANDLE*)&HbmpAct);
 if (FAILED(hr))
 return hr;
 BITMAP ExpBmp = {0};
 if(!GetObject(HbmpExp, sizeof(ExpBmp), &ExpBmp))
 return E_FAIL;
 BITMAP ActBmp = {0};
 if(!GetObject(HbmpAct, sizeof(ActBmp), &ActBmp))
 return E_FAIL;
 CString s, tol;
 tol = bstrConfiguration;
 int EPos = tol.ReverseFind('=');
 tol = tol.Right(tol.GetLength() - EPos - 1);
 int maxSurfaceAreaDiff = _ttoi(tol);
 // Set output parameters
 CComPtr<IPictureDisp> Diff(pActual);

```

```

 *ppDiff = Diff;
 int DiffPixelsNumber = abs (ExpBmp.bmHeight * ExpBmp.bmWidth -
ActBmp.bmHeight * ActBmp.bmWidth);
 *pbMatch = DiffPixelsNumber <= maxSurfaceAreaDiff;
 s.Format(_T("The number of different pixels is: %d."),
DiffPixelsNumber);
 CComBSTR bs (s);
 *pbstrLog = bs.Detach();
 return hr;
 }
}

```

## 6. Design your custom comparer to register to the component category for UFT bitmap comparers

For UFT to recognize the COM object that you create as a custom comparer, you must register it to the component category for UFT bitmap comparers. The component category ID is defined in <UFT installation folder>\dat\BitmapCPCustomization\ComponentCategory.h.

You can implement this registration in the **DllRegisterServer** and **DllUnregisterServer** methods in the SampleCPPCustomComparer.cpp file that the wizard created as part of your project. These methods are called when you run a DLL using the regsvr32.exe program.

- Add the <UFT installation folder>\dat\BitmapCPCustomization folder to your project's include path.
- Open the SampleCPPCustomComparer.cpp file and add the following line: #include "ComponentCategory.h"
- In the SampleCPPCustomComparer.cpp file, modify the **DllRegisterServer** and **DllUnregisterServer** methods created by the wizard, to contain the following code:

```

STDAPI DllRegisterServer(void)
{
 // registers object, typelib and all interfaces in typelib
 HRESULT hr = _AtModule.DllRegisterServer();
 CComPtr<ICatRegister> spReg;
 hr = spReg.CoCreateInstance
 (CLSID_StdComponentCategoriesMgr, 0, CLSCTX_INPROC);
 if (FAILED(hr))
 return hr;
 // register comparer to the UFT bitmap comparers category
 CATID catid = CATID_QTPBitmapComparers;
 hr = spReg->RegisterClassImplCategories(CLSID_BitmapComparer, 1,
&catid);
 return hr;
}

```

```
STDAPI DllUnregisterServer(void)
{
 HRESULT hr = _AtlModule.DllUnregisterServer();
 CComPtr spReg;
 hr = spReg.CoCreateInstance
 (CLSID_StdComponentCategoriesMgr, 0, CLSCTX_INPROC);
 if (FAILED(hr))
 return hr;
 // unregister comparer from the UFT bitmap comparers category
 CATID catid = CATID_QTPBitmapComparers;
 hr = spReg->UnRegisterClassImplCategories(CLSID_BitmapComparer,
1, &catid);
 return hr;
}
```

Note the second section in these methods, that handles registration to the component category for UFT bitmap comparers—**CATID\_QTPBitmapComparers**.

## 7. Compile your DLL and run it using the regsvr32.exe program

Your custom comparer can now be used in UFT for bitmap checkpoints.

## 8. Test your custom comparer by using it for bitmap checkpoints in UFT

For details on how to work with bitmap checkpoints, see "[Bitmap Checkpoints Overview](#)" on page [294](#).

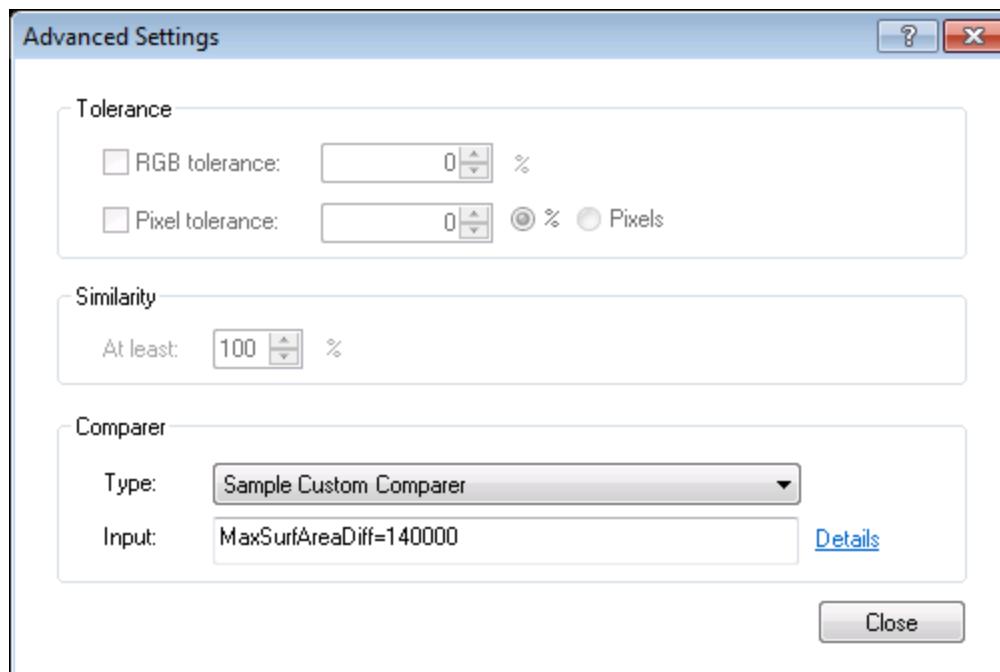
- a. Open UFT and create a bitmap checkpoint on the Windows Calculator application (Standard view).

The advanced settings in the Bitmap Checkpoint Properties dialog box include the **Comparer** option, in which you can select the default UFT comparer or your sample custom comparer.

- b. Change the Calculator view to **Scientific**. The size of the calculator object is now larger. Run the checkpoint using the default UFT comparer. The checkpoint fails.

c. Edit the checkpoint in the Bitmap Checkpoint Properties dialog box:

- Make sure the **Compare selection with runtime bitmap** checkpoint mode is selected.
- Click **Advanced settings** to open the Advanced Settings dialog box, and select **Sample Custom Comparer** in the Comparer **Type** box.



In the **Input** box, you can see the default configuration string returned by the **GetDefaultConfigurationString** method: MaxSurfAreaDiff=140000

In the **Input** box, you can see the default configuration string returned by the **GetDefaultConfigurationString** method: MaxSurfAreaDiff=140000

If you click **Details**, the text file containing documentation for the sample custom comparer opens.

The comparer you designed in this exercise checks how different the expected and actual bitmaps are in size, and fails the checkpoint if the difference is greater than the number of pixels defined in the configuration string.

If you run the checkpoint using default MaxSurfAreaDiff value, the checkpoint passes, because the difference in the total size of the calculator object when it is set to different views is less than 140000 pixels (the difference is approximately 80000 pixels). If you set MaxSurfAreaDiff to 70000, the checkpoint fails.

View the run results to see the text string and difference bitmap that your custom comparer provides to UFT after the comparison.

# The Bitmap Checkpoint Comparer Interfaces

## Relevant for: GUI tests and components

Your custom comparer must implement the interfaces described in this section. UFT calls these interfaces' methods when creating or running a bitmap checkpoint that uses your custom comparer.

To learn more, see:

- [The IVerifyBitmap Interface](#) ..... 1113
- [The IBitmapCompareConfiguration Interface](#) ..... 1114

## The IVerifyBitmap Interface

### Relevant for: GUI tests and components

This interface contains the `CompareBitmaps` method that you need to implement to perform the bitmap comparison for the checkpoint.

### The `CompareBitmaps` Method

The `CompareBitmaps` method receives the actual and expected bitmaps that need to be compared for the bitmap checkpoint, and a string that can contain configuration input for the custom comparer.

The method must compare the bitmaps according to the comparison algorithm for which this custom comparer is designed, and return the results to UFT.

The results include:

- An indication whether the bitmaps match and the checkpoint should pass.
- A text string that contains information about the results of the bitmap comparison.
- A bitmap that reflects the differences between the actual and expected bitmaps.

UFT displays the results that this method returns in the run results. For details, see the section on ["Using Run Results" on page 853](#).

### Method syntax:

```
HRESULT CompareBitmaps ([in] IPictureDisp* pExpected,
 [in] IPictureDisp* pActual,
 [in] BSTR bstrConfiguration,
 [out] BSTR* pbstrLog,
 [out] IPictureDisp** ppDiff,
 [out, retval] VARIANT_BOOL* pbMatch);
```

**Method Parameters:**

- *pExpected*. A picture object (input).  
The expected bitmap stored in the checkpoint.
- *pActual*. A picture object (input).  
The actual bitmap captured from the application being tested.
- *bstrConfiguration*. A text string (input).  
A string that contains configuration input for the custom comparer. This is the string displayed in the **Input** box in the advanced settings in the Bitmap Checkpoint Properties dialog box.  
The string can be the default configuration string that the custom comparer provides to UFT in the **GetDefaultConfigurationString** method described below, or an input string entered by the UFT user.  
The **bstrConfiguration** string can have any format you choose (XML, comma separated, ini file style, and so on). Make sure that the default configuration string returned by the **GetDefaultConfigurationString** method matches the format expected in the **CompareBitmaps** method. Additionally, make sure that the documentation you provide for your custom comparer explains the format that the UFT user must use when editing this string in the **Input** box.
- *pbstrLog*. A text string (output).  
A string that contains information about the results of the bitmap comparison. UFT displays this string in the run results.
- *ppDiff*. A picture object (output).  
A bitmap (created by the custom comparer) that reflects the difference between the actual and expected bitmaps. UFT displays this bitmap in the run results along with the actual and expected bitmaps.
- *pbMatch*. A boolean value (output).  
A value that indicates whether the bitmaps match and the checkpoint should pass.

**Possible values:**

- VARIANT\_TRUE. Actual and expected bitmaps match, checkpoint passes.
- VARIANT\_FALSE. Actual and expected bitmaps do not match, checkpoint fails.

**Return Value**

The HRESULT that this method returns indicates whether the comparison ran successfully (and not whether the bitmaps match).

## The IBitmapCompareConfiguration Interface

**Relevant for: GUI tests and components**

This interface contains the methods that you need to implement to support the custom comparer options that UFT displays in the advanced settings in the Bitmap Checkpoint Properties dialog box. For details, see "[Checkpoint Properties Dialog Box](#)" on page 320.

## The **GetDefaultConfigurationString** Method

The **GetDefaultConfigurationString** method must return the default configuration string for your custom comparer. For details on configuration strings, see "[Implement the CompareBitmaps method to accept input and compare bitmaps](#)" on page 1098.

UFT displays this string in the **Input** box in the advanced settings in the Bitmap Checkpoint Properties dialog box when a user creating a new bitmap checkpoint selects your custom comparer.

If the UFT user does not modify the input string in the dialog box, the string provided by **GetDefaultConfigurationString** is passed to the custom comparer's **CompareBitmaps** method. You must therefore make sure that the default configuration string matches the format that your custom comparer expects to receive in the **CompareBitmaps** method.

### Method syntax:

```
HRESULT GetDefaultConfigurationString ([out, retval] BSTR* pbstrConfiguration);
```

## The **GetHelpFilename** Method

The **GetHelpFilename** method must return a path to the documentation that contains information about your custom comparer for UFT users.

UFT displays the documentation when a user selects your custom comparer in the advanced settings in the Bitmap Checkpoint Properties dialog box and clicks **Details**. Make sure that when your custom comparer is installed, the documentation that you provide is installed in the location specified by the **GetHelpFilename** method.

The path can be one of the following:

- A full path to a file.
- A relative path to a file (UFT searches for this path relative to <UFT installation folder>\bin).
- A URL.

If you do not provide documentation for your custom comparer, this method should return the **HRESULT E\_NOTIMPL**. For details on the type of information you should provide, see "[Implement IBitmapCompareConfiguration to provide information for the Bitmap Checkpoint Properties dialog box](#)" on page 1099.

### Method syntax:

```
HRESULT GetHelpFilename ([out, retval] BSTR* pbstrFilename);
```

# Appendix C: GUI Checkpoints and Output Values Per Add-in

## **Relevant for: GUI tests and components**

The tables in this chapter show the categories of checkpoints and output values that are supported by UFT for each add-in.

For details about using checkpoints and output values in a specific add-in, see the relevant add-in section.

This chapter includes:

- [Supported Checkpoints](#) ..... 1117
- [Supported Output Values](#) ..... 1120

# Supported Checkpoints

## Relevant for: GUI tests and components

The following table shows the categories of checkpoints that are supported by UFT for each add-in.

### Table Legend

- S: Supported
- NS: Not Supported
- NA: Not Applicable

**Note:** Only standard and bitmap checkpoints are supported for keyword components.

For additional information, see "[Footnotes](#)" on the next page.

	Accessibility	Bitmap	Database	File Content	Image	Page	Standard	Table	Text	Text Area	XML (Application)	XML (Resource)
.NET Web Forms <sup>3</sup>	S	S	NA	NA	NA	NA	S	S	S <sup>6</sup>	S <sup>6</sup>	S	S
.NET Windows Forms	NA	S	NA	NA	NA	NA	S	S	S <sup>6</sup>	S <sup>6</sup>	NA	NA
ActiveX	NS	S	NA	NA	NS	NA	S	S	S	S	NA	NA
Delphi	NS	S	NA	NA	NS	NA	S	S	S	S	NA	NA
Flex	NA	S	NA	NA	NA	NA	S	S	S	S	NA	NA
Java	NA	S	NA	NA	NA	NA	S	S	S	S <sup>4</sup>	NA	NA
Mobile	NA	S	NA	NA	NA	NA	S	NA	S	NS	NA	NA
Oracle	NA	S	NA	NA	NA	NA	S	S	NS	NS	NA	NA
PeopleSoft	S	S	NA	NA	S	S	S	S	S <sup>1</sup>	NS	S	S
PowerBuilder <sup>2</sup>	NS	S	NA	NA	NS	NA	S	S	S	S	NA	NA
Qt	NS	S	NA	NA	NS	NA	S	S	S	S	NA	NA
SAP Web-based	S	S	NA	NA	S	S	S	S	S	NS	S	S
SAP	S <sup>5</sup>	S	NA	NA	S <sup>5</sup>	S <sup>5</sup>	S	S	S <sup>5</sup>	NS	S <sup>5</sup>	NA

	<b>Accessibil ity</b>	<b>Bitm ap</b>	<b>Databa se</b>	<b>File Conte nt</b>	<b>Ima ge</b>	<b>Pa ge</b>	<b>Standa rd</b>	<b>Tab le</b>	<b>Te xt</b>	<b>Te xt Are a</b>	<b>XML (Applicati on)</b>	<b>XML (Resour ce)</b>
<b>Windows- based</b>												
<b>Siebel</b>	S	S	NA	NA	S	S	S	S	S	NS	S	S
<b>Silverlight</b>	NA	S	NA	NA	NA	NA	S	S	S	S	NA	NA
<b>Standard Windows</b>	NS	S	NA	NA	NS	NA	S	S	S	S	NA	NA
<b>Stingray</b>	NA	S	NA	NA	NA	NA	S	S	S	S	NA	NA
<b>Terminal Emulator</b>	NA	S	NA	NA	NA	NA	S	NA	NA	NA	NA	NA
<b>VisualAge for Smalltalk</b>	NA	S	NA	NA	NA	NA	S	S	S	S	NA	NA
<b>Visual Basic</b>	NS	S	NA	NA	NS	NA	S	S	S	S	NA	NA
<b>Web</b>	S	S	NA	NA	S	S	S	S	S <sup>1</sup>	S	S <sup>7</sup>	NA
<b>Windows Runtime</b>	NA	S	NA	NA	NA	NA	S	S	S	S	NA	NA
<b>WPF</b>	NA	S	NA	NA	NA	NA	S	S	S	S	NA	NA

**Footnotes**

1 Text checkpoints are supported only for Page, Frame, and ViewLink objects.

2 When you insert a checkpoint on a PowerBuilder DataWindow control, UFT treats it as a table and opens the Table Checkpoint Properties dialog box.

3 For NET Web Forms, text checkpoints for WbfTreeView, WbfToolbar, and WbfTabStrip objects are not supported.

4 The text area checkpoint mechanism for Java Applet objects is disabled by default. You can enable it in the Advanced Java Options dialog box.

5 This is supported only when UFT records HTML elements using the Web infrastructure, but not when it records using the SAPGui Scripting Interface (as selected in the SAP pane of the Options dialog box).

6 This is supported only when UFT is configured to use the OCR (optical character recognition) mechanism.

7 XML checkpoints are not supported on Internet Explorer 9 or later running in standard mode, on Google Chrome, on Mozilla Firefox, or on Apple Safari because the WebXML test object is not supported for these browsers.

8 - Checkpoints on Mobile object are supported only when recording your test.

**Note:** For additional information about using text recognition in checkpoints, see "[Guidelines for Text Recognition](#)" on page [302](#).

# Supported Output Values

## Relevant for: GUI tests and components

The following table shows the categories of output values that are supported by UFT for each add-in.

### Table Legend

- **S:** Supported
- **NS:** Not Supported
- **NA:** Not Applicable

**Note:** Only standard and bitmap output values are supported for keyword components.

For additional information, see "[Footnotes](#)" on the next page.

	Accessibility	Bitmap	Database	File Content	Image	Page	Standard	Table	Text	Text Area	XML (Application)	XML (Resource)
<b>.NET Web Forms</b>	NA	NA	NA	NA	NA	S	S	S	S <sup>5</sup>	S <sup>5</sup>	NA	NA
<b>.NET Windows Forms</b>	NA	NA	NA	NA	NA	NA	S	S	S <sup>5</sup>	S <sup>5</sup>	NA	NA
<b>ActiveX</b>	NS	NA	NA	NA	NA	NA	S	S	S	S	NA	NA
<b>Delphi</b>	NS	NA	NA	NA	NA	NA	S	S	S	S	NA	NA
<b>Java</b>	NA	NA	NA	NA	NA	NA	S	NA	S	S <sup>3</sup>	NA	NA
<b>Mobile</b>	NA	NA	NA	NA	NA	NA	S	NA	S	NS	NA	NA
<b>Oracle</b>	NA	NA	NA	NA	NA	NA	S	S	NA	NA	NA	NA
<b>PeopleSoft</b>	NA	NA	NA	NA	NA	S	S	S	S <sup>1</sup>	NS	S	S
<b>PowerBuilder<sup>2</sup></b>	NA	NA	NA	NA	NA	NA	S	NA	S	S	NA	NA
<b>Qt</b>	NA	NA	NA	NA	NA	NA	S	S	S	S	NA	NA
<b>SAP Web-based</b>	NA	NA	NA	NA	NA	S	S	S	S	NS	S	S
<b>SAP Windows-based</b>	NA	NA	NA	NA	NA	S <sup>4</sup>	S	S	S <sup>4</sup>	NS	S <sup>4</sup>	S

	<b>Accessibil ity</b>	<b>Bitm ap</b>	<b>Databa se</b>	<b>File Conte nt</b>	<b>Ima ge</b>	<b>Pa ge</b>	<b>Standa rd</b>	<b>Tab le</b>	<b>Te xt</b>	<b>Te xt Are a</b>	<b>XML (Applicati on)</b>	<b>XML (Resour ce)</b>
<b>Siebel</b>	NA	NA	NA	NA	NA	S	S	S	S	NS	S	S
<b>Silverlight</b>	NA	NA	NA	NA	NA	NA	S	S	S	S	NA	NA
<b>Standard Windows</b>	NA	NA	NA	NA	NA	NA	S	S	S	S	NA	NA
<b>Stingray</b>	NA	NA	NA	NA	NA	NA	S	S	S	S	NA	NA
<b>Terminal Emulator</b>	NA	NA	NA	NA	NA	NA	S <sup>8</sup>	NA	S <sup>7</sup>	NA	NA	NA
<b>VisualAge for Smalltalk</b>	NA	NA	NA	NA	NA	NA	NA	S	S	S	NA	NA
<b>Visual Basic</b>	NA	NA	NA	NA	NA	NA	S	NA	S	S	NA	NA
<b>Web</b>	NA	NA	NA	NA	NA	S	S	S	S <sup>1</sup>	NS	S <sup>6</sup>	NA
<b>Windows Runtime</b>	NA	NA	NA	NA	NA	NA	S	S	S	S	NA	NA
<b>WPF</b>	NA	NA	NA	NA	NA	NA	S	S	S	S	NA	NA

**Footnotes**

1 Text output values are supported only for Page, Frame, and ViewLink objects.

2 When you insert an output value step on a PowerBuilder DataWindow control, UFT treats it as a table and opens the Table Output Value Properties dialog box.

3 The text area output mechanism for Java Applet objects is disabled by default. You can enable it in the Advanced Java Options dialog box.

4 This is supported only when UFT records HTML elements using the Web infrastructure, but not when it records using the SAPGui Scripting Interface (as selected in the SAP pane of the Options dialog box).

5 This is supported only when UFT is configured to use the OCR (optical character recognition) mechanism.

6 XML output values are not supported on Internet Explorer 9 or later running in standard mode, on Google Chrome, or on Mozilla Firefox, because the WebXML test object is not supported for these browsers.

7 You can create text output values (tests only) only for TeScreen and TeTextScreen objects.

8 In the terminal emulator window you can add text checkpoints or output values (tests only) and standard checkpoints and output values for the status bar and the dialog boxes that open from the menu options. UFT recognizes these as standard Windows objects.

# Appendix D: Frequently Asked Questions for GUI Testing

## **Relevant for: GUI tests and components**

This chapter answers some of the questions that are asked most frequently by advanced users of UFT. The questions and answers are divided into the following sections:

This chapter includes:

• <a href="#">Creating Tests or Components</a> .....	1123
• <a href="#">Programming in the Editor and Working with Function Libraries</a> .....	1123
• <a href="#">Working with Dynamic Content</a> .....	1125
• <a href="#">Advanced Web Issues</a> .....	1127
• <a href="#">Standard Windows Environment</a> .....	1130
• <a href="#">Test and Component Maintenance</a> .....	1131
• <a href="#">Testing Localized Applications</a> .....	1132
• <a href="#">Improving GUI Testing Performance</a> .....	1133

# Creating Tests or Components

## Relevant for: GUI tests and components

<b>How can I record on objects or environments not supported by UFT?</b>	<ul style="list-style-type: none"><li>Install and load any of the add-ins that are available for UFT</li><li>Map objects of an unidentified or custom class to standard Windows classes. For details on object mapping, see "<a href="#">Test Object Mapping for Unidentified or Custom Classes</a>" on page 241.</li><li>Use Insight recording mode.</li><li>Use UI Automation recording mode (for Windows-based technologies that have implemented the UI Automation interfaces).</li><li>Use add-in extensibility to extend UFT built-in support for various objects.</li><li>Define virtual objects for objects that behave like test objects, and then record in the normal recording mode. For details on defining virtual objects, see "<a href="#">Virtual Objects</a>" on page 286.</li><li>You can record your clicks and keyboard input based on coordinates in the low-level or analog recording modes. For details on low-level and analog recording, see "<a href="#">Recording Overview</a>" on page 97.</li></ul>
<b>How can I launch an application from a test or component?</b>	<ul style="list-style-type: none"><li>Configure the Record and Run Settings for your technology to automatically open an application</li><li>In a GUI test or scripted GUI component, add a <b>SystemUtil</b> step, such as:<pre>SystemUtil.Run "D:\My Music\Breathe.mp3", "", "D:\My Music\Details", "open"</pre></li><li>In a keyword GUI component, select <b>Operation</b> from the <b>Item</b> column, select <b>OpenApp</b> from the <b>Operation</b> column, and then enter the full path in the <b>Value</b> column, for example:<pre>%ProgramFiles%\HP\Unified Functional Testing\samples\flight\app\flight4a.exe</pre></li></ul>
<b>How does UFT capture user processes in Web pages?</b>	UFT hooks into your selected browser. As the user navigates the Web-based application, UFT records the user operations. (For details on modifying which user operations are recorded, see the section on configuring Web event recording in the <i>HP Unified Functional Testing Add-ins Guide</i> .) UFT can then run the test or component by running the steps as they originally occurred.

# Programming in the Editor and Working with Function Libraries

## Relevant for: GUI tests and components

This section includes answers to the following questions:

- ["Can I store functions and subroutines in a function library?" on the next page](#)
- ["How can I enter information during a run session?" on the next page](#)

- "For tests: I have a Microsoft Access database that contains data I would like to use in my test. How do I do this?" below
- "How do I customize the Run Results?" on the next page

## Can I store functions and subroutines in a function library?

You can create one or more VBScript function libraries containing your functions. You can then call the functions from any test, or use them in any component, by associating them with the test or with the component's application area. You can use the UFT function library editor to create and debug your function libraries.

You can also register your functions as methods for UFT test objects. Your registered methods can override the functionality of an existing test object method for the duration of a run session, or you can register a new method for a test object class.

For more details, see "[User-Defined Functions and Function Libraries](#)" on page 683, and, for components, "[Application Areas](#)" on page 1014.

**For tests:** You can define functions within an individual action, making them available for use inside the action. If you save the action as a reusable action, its functions are available in the actions that call it as well. However, you can help improve UFT performance by storing your functions in function libraries instead of in reusable actions.

## How can I enter information during a run session?

The VBScript **InputBox** function enables you to display a dialog box that prompts the user for input, and then continues running the test or component. You can use the value that was entered by the user later in the run session. For details on the **InputBox** function, see the *VBScript Reference*.

**For components:** You can insert the VBScript **InputBox** function into an associated function library to create a user-defined VBScript **InputBox** function.

**For tests:** The following example shows the **InputBox** function used to prompt the user for a password:

```
Browser("Mercury Tours").Page("Mercury Tours").WebEdit("username").Set
"administrator"
Passwd = InputBox ("Enter password", "User Input")
Browser("Mercury Tours").Page("Mercury Tours").WebEdit("password").Set Passwd
```

## For tests: I have a Microsoft Access database that contains data I would like to use in my test. How do I do this?

The Editor enables you to access databases using ADO and ODBC. Below is a sample test that searches for books written by an author in the "Authors" table of the database.

```
Dim MyDB
Dim MyEng
```

```
Set MyEng = CreateObject("DAO.DBEngine.35")
Dim Td
Dim rs
' Specify the database to use.
Set MyDB = MyEng.OpenDatabase("BIBLIO.MDB")
' Read and use the name of the first 10 authors.
Set Td = MyDB.TableDefs("Authors")
Set rs = Td.OpenRecordset
rs.MoveFirst
For i = 1 To 10
 Browser("Book Club").Page("Search Books").WebEdit("Author Name").Set rs
 ("Author")
 Browser("Book Club").Page("Search Books").WebButton("Search").Click
Next
```

## How do I customize the Run Results?

You can add information to the run results by using the **ReportEvent** method. This adds a step to the run results containing a free-text message and a step status that can potentially affect the status of the run session. The step can also include an image, such as a logo, if you specify an image file path.

For example:

```
Reporter.ReportEvent 1, "Custom Step", "The user-defined step failed"
```

For more details, see the **Reporter** object in the **Utility Objects** section of the *HP UFT Object Model Reference for GUI Testing*.

# Working with Dynamic Content

## Relevant for: GUI tests and components

This section includes answers to the following questions:

- "[How can I create and run tests or components on objects that change dynamically from viewing to viewing?](#)" below
- "[How can I check that an object or child object exists \(or does not exist\)?](#)" on the next page
- "[How does UFT record on dynamically generated URLs and Web pages?](#)" on the next page
- "[How does UFT handle tabs in browsers?](#)" on page 1127

## How can I create and run tests or components on objects that change dynamically from viewing to viewing?

Sometimes the content of objects in an application changes due to dynamic content. You can create dynamic descriptions of these objects so that UFT will recognize them when it runs the test or

component using regular expressions, the **Description** object, repository parameters, or **SetTOProperty** steps.

### How can I check that an object or child object exists (or does not exist)?

Some objects are created in an application only after you perform an operation. For example, a link in one window sometimes creates another window. The newly created window may be an independent object, or a child of the original window.

Before you perform operations on an object created during a run session, you may want to verify that the object already exists.

If you have a test object that corresponds to the object in the application, you can use the **Exist** property to check whether the object exists in the application. The **Exist** property looks for an object in the application that matches the test object's description. For example:

```
If Window("Main").ActiveX("Slider").Exist Then
 . . .
```

Alternatively, you can use the **ChildObjects** method to retrieve all child objects (or the subset of child objects that match a certain description) on the Desktop or within any other parent object.

#### Example

```
Set oDesc = Description.Create
oDesc("Class Name").Value = "Window"
Set coll = Desktop.ChildObjects(oDesc)
For i = 0 to coll.count -1
 msgbox coll(i).GetROProperty("text")
Next
```

#### Note:

- After you use the **ChildObjects** method to retrieve an object, UFT accesses the object directly in the application and does not save a description for the object. Therefore, you must use the object immediately after retrieving it, before anything in the application changes.
- The **Exist** property searches for objects based on their description, and is therefore not relevant for objects retrieved by the **ChildObjects** method. (The **Exist** property always returns true when called for such objects).

For more details on the **Exist** property and **ChildObjects** method, see the **Common Methods and Properties** section of the *HP UFT Object Model Reference for GUI Testing*.

### How does UFT record on dynamically generated URLs and Web pages?

UFT actually clicks links as they are displayed on the page. Therefore, UFT records how to find a particular object, such as a link on the page, rather than the object itself. For example, if the link to a

dynamically generated URL is an image, then UFT records the "IMG" HTML tag, and the name of the image. This enables UFT to find this image in the future and click on it.

## How does UFT handle tabs in browsers?

UFT provides several methods that you can use with the **Browser** test object to manage tabs in your Web browser.

**OpenNewTab** opens a new tab in the current Web browser.

**IsSiblingTab** indicates whether a specified tab is a sibling of the current tab object in the same browser window.

**Close** closes the current tab if more than one tab exists, and closes the browser window if the browser contains only one tab.

**CloseAllTabs** closes all tabs in a browser and closes the browser window.

For more details on these Browser-related methods, see the **Web** section of the *HP UFT Object Model Reference for GUI Testing*.

# Advanced Web Issues

## Relevant for: GUI tests and components

This section includes answers to the following questions:

- "[How does UFT handle cookies?](#)" below
- "[Where can I find a Web page's cookie?](#)" on the next page
- "[How does UFT handle session IDs?](#)" on the next page
- "[How does UFT handle server redirections?](#)" on the next page
- "[How does UFT handle meta tags?](#)" on the next page
- "[Does UFT work with .asp and .jsp?](#)" on the next page
- "[How does UFT support AJAX?](#)" on the next page
- "[Does UFT work with COM?](#)" on the next page
- "[Does UFT work with XML?](#)" on page 1129
- "[How can I access HTML tags directly?](#)" on page 1129
- "[Where can I find information on the Internet Explorer Document Object Model?](#)" on page 1129
- "[How can I send keyboard key commands \(such as shortcut commands\) to objects that do not support the Type method?](#)" on page 1129

## How does UFT handle cookies?

Server-side connections, such as CGI scripts, can use cookies both to store and retrieve information on the client side of the connection.

UFT stores cookies in the memory for each user, and the browser handles them as it normally would.

### Where can I find a Web page's cookie?

The cookie used by the Internet Explorer browser can be accessed through the browser's Document Object Model (DOM) using the **.Object** property (for components, you do this in a user-defined function). In the following example the cookie collection is returned from the browser:

```
Browser("Flight reservations").Page("Flight reservations").Object.Cookie
```

### How does UFT handle session IDs?

The server, not the browser, handles session IDs, usually by a cookie or by embedding the session ID in all links. This does not affect UFT.

### How does UFT handle server redirections?

When the server redirects the client, the client generally does not notice the redirection, and misdirections generally do not occur. In most cases, the client is redirected to another script on the server. This additional script produces the HTML code for the subsequent page to be viewed. This has no effect on UFT or the browser.

### How does UFT handle meta tags?

Meta tags do not affect how the page is displayed. Generally, they contain information only about who created the page, how often it is updated, what the page is about, and which keywords represent the page's content. Therefore, UFT has no problem handling meta tags.

### Does UFT work with .asp and .jsp?

Dynamically created Web pages utilizing Active Server Page technology have an **.asp** extension.

Dynamically created Web pages utilizing Java Server Page technology have a **.jsp** extension. These technologies are completely server-side and have no bearing on UFT.

### How does UFT support AJAX?

You can use UFT Web Add-in Extensibility to add your own support for custom Web controls. The Web Add-in Extensibility SDK installs a sample toolkit support set that provides partial support for some ASP .NET AJAX controls. You can use this sample to learn how to create your own support for your AJAX controls. For more details, see the *HP UFT Web Add-in Extensibility Developer Guide*.

### Does UFT work with COM?

UFT complies with the COM standard.

UFT supports COM objects embedded in Web pages (which are currently accessible only using Microsoft Internet Explorer), and you can drive COM objects in VBScript.

## Does UFT work with XML?

XML is eXtensible Markup Language, a pared-down version of SGML for Web documents, that enables Web designers to create their own customized tags. UFT supports XML and recognizes XML tags as objects.

**For tests and scripted components:** You can also create XML checkpoints to check the content of XML documents in Web pages, frames or files. UFT also supports XML output and schema validation.

For more details, see "[XML Checkpoints Overview](#)" on page 307, and the **XMLUtil** object in the **Utility Objects** section of the *HP UFT Object Model Reference for GUI Testing*.

## How can I access HTML tags directly?

UFT provides direct access to the Internet Explorer's Document Object Model (DOM) through which you can access the HTML tags directly. Access to the DOM is performed using the .Object notation.

The function below demonstrates how to iterate over all the tags in an Internet Explorer page. The function then outputs the inner-text of the tags (the text contained between the tags) to the run results using the **Reporter** object.

```
' Use the on error option because not all the elements have inner-text.
On Error Resume Next
Set Doc = Browser("CNN Interactive").Page("CNN Interactive").Object
' Loop through all the objects in the page.
For Each Element In Doc.all
 TagName = Element.tagName ' Get the tag name.
 InnerText = Element.innerText ' Get the inner text.
 ' Write the information to the run results.
 Reporter.ReportEvent 0, TagName, InnerText
Next
```

## Where can I find information on the Internet Explorer Document Object Model?

For details on the Internet Explorer DOM, browse to the following Web sites:

Document object: <http://msdn.microsoft.com/en-us/library/ms531073.aspx>

Other DHTML objects: <http://msdn.microsoft.com/en-us/library/ms533054.aspx>

General DHTML reference: <http://msdn.microsoft.com/en-us/library/ms533050.aspx>

## How can I send keyboard key commands (such as shortcut commands) to objects that do not support the Type method?

For objects that do not support the **Type** method, use the Windows Scripting **SendKeys** method. For more details, see the Microsoft VBScript Language Reference (choose **Help > HP Unified Functional Testing Help > VBScript Reference > Windows Script Host**).

# Standard Windows Environment

## Relevant for: GUI tests and components

This section includes the answers to the following questions:

- "[How can I record on nonstandard menus?](#)" below
- "[For tests: How can I terminate an application that is not responding?](#)" below
- "[For tests: Can I copy and paste to and from the Clipboard during a run session?](#)" below

### **How can I record on nonstandard menus?**

You can modify how UFT behaves when it records menus. The options that control this behavior are located in the Windows Applications > Advanced Options pane. (**Tools > Options > GUI Testing tab > Windows Applications node > Advanced node**).

For more details, see the *HP Unified Functional Testing Add-ins Guide*.

### **For tests: How can I terminate an application that is not responding?**

You can terminate any standard application while running a test in UFT by adding one of the following steps to the test:

- SystemUtil.CloseProcessByName "<app.exe>"
- SystemUtil.CloseProcessByWndTitle "<Some Title>"

### **For tests: Can I copy and paste to and from the Clipboard during a run session?**

You can use the Clipboard object to copy, cut, and paste text during a UFT run session.

The Clipboard object supports the same methods as the Clipboard object available in Visual Basic, such as:

- Clear
- GetData
- GetText
- SetData
- SetText

For details on these methods, see <http://msdn.microsoft.com/en-us/library/ms172962.aspx>.

Below is an example of Clipboard object usage:

```
Set MyClipboard = CreateObject("Mercury.Clipboard")
MyClipboard.Clear
MyClipboard.SetText "TEST"
```

```
MsgBox MyClipboard.GetText
```

## Test and Component Maintenance

### Relevant for: GUI tests and components

This section includes answers to the following questions:

- "[How do I maintain my test or component when my application changes?](#)" below
- "[For tests and scripted components: Can I increase or decrease Active Screen information after I finish recording a test?](#)" on the next page
- "[For tests: How can I remove run results files from old tests?](#)" on the next page

### **How do I maintain my test or component when my application changes?**

The way to maintain a test or component when your application changes depends on how much your application changes. This is one of the main reasons you should create a small group of tests or components, rather than one large test or component for your entire application.

**For tests:** You can also use UFT actions to design more modular and efficient tests. Divide your test into several actions, based on functionality. When your application changes, you can modify a specific action, without changing the rest of the test. Whenever possible, insert calls to reusable actions rather than creating identical pieces of script in several tests. This way, changes to your original reusable action are automatically applied to all tests calling that action. For more details, see "[Actions in GUI Testing](#)" on [page 103](#).

If you have many tests, components, and actions that contain the same test objects, it is recommended to work with shared object repositories so that you can update object information in a centralized location.

You can use the **Update Run Mode** option to update changed information for checkpoints or the Active Screen, or to change the set of identification properties used to identify the objects in your application.

If there is a discrepancy between the identification property values saved in the object repository and the object property values in the application, you can use the **Maintenance Run Mode** to help correct this. When you run a test or component in Maintenance Run Mode, UFT runs your test or component, and then guides you through the process of updating your steps and object repository each time it encounters a step it cannot perform due to an object repository discrepancy. For more details, see "[Maintenance Run Mode](#)" on [page 135](#).

## For tests and scripted components: Can I increase or decrease Active Screen information after I finish recording a test?

If you find that the information saved in the Active Screen after recording is not sufficient, or if you no longer need Active Screen information, and you want to decrease the size of your test or component, there are several methods of changing the amount of Active Screen information saved.

- To decrease the disk space used by your test or component, you can delete Active Screen information by selecting **Save As**, and clearing the **Save Active Screen files** check box.
- If you chose not to save all information in the Active Screen when testing a Windows application, you can use one of several methods to increase the information stored in the Active Screen.

Confirm that the Active Screen capture preference in the **Active Screen** pane of the Options dialog box (**Tools > Options > GUI Testing tab > Active Screen node**) is set to capture the amount of information you need and then:

- Perform an **Update Run Mode** operation to save the required amount of information in the Active Screen for all existing steps.
- Re-record the steps containing the objects you want to add to the Active Screen.

To re-record the step, select the step after which you want to record your step, position your application to match the selected location in your test or component, and then begin recording. Alternatively, place a breakpoint in your test at the step before which you want to add a step and run your test or component to the breakpoint. This brings your application to the point from which to record the step. For details on setting breakpoints, see "[Breakpoints " on page 906](#).

## For tests: How can I remove run results files from old tests?

You can use the Run Results Deletion Tool to view a list of all of the run results in a specific location in your file system or in your ALM project. You can then delete any run results that you no longer require.

The Run Results Deletion Tool enables you to sort the run results by name, date, size, and so forth, so that you can more easily identify the results you want to delete.

To open this utility, choose **Start > All Programs > HP Software > HP Unified Functional Testing > Tools > Run Results Deletion Tool**.

# Testing Localized Applications

**Relevant for: GUI tests only**

## I am testing localized versions of a single application, each with localized user interface strings. How do I create efficient tests in UFT?

You can parameterize these user interface strings using parameters from the global Environment variable list. This is a list of variables and corresponding values that can be accessed from any test. For

details, see "Parameterizing Object Values" on page 336.

## I am testing localized versions of a single application. How can I efficiently input different data in my tests, depending on the language of the application?

If you are running a single iteration of your test, or if you want values to remain constant for all iterations of an action or test, use environment variables, and then change the active environment variable file for each test run.

If you are running multiple iterations of your test or action, and you want the input data to change in each iteration, you can create an external data table for each localized version of your application. When you change the localized version of the application you are testing, you simply switch the data table file for your test in the Resources pane of the Test Settings dialog box.

# Improving GUI Testing Performance

## Relevant for: GUI tests and components

This section includes the answers to the following questions:

- ["How can I improve the working speed of UFT when working with GUI testing?" below](#)
- ["How can I decrease the disk space used by UFT for GUI tests and components? " on page 1135](#)
- ["For tests: Is there a recommended length for tests?" on page 1136](#)

## How can I improve the working speed of UFT when working with GUI testing?

You can improve the working speed of UFT by doing any of the following:

- In the Add-in Manager, load only the add-ins you need for a specific UFT session when UFT starts. This will improve performance while learning objects and during run sessions. For details on loading add-ins, see the *HP Unified Functional Testing Add-ins Guide*.
- Run your tests or components in "fast mode." From the **Test Runs** pane in the Options dialog box (**Tools > Options > GUI Testing tab > Test Runs** node), select the **Fast** option. This instructs UFT to run your test or component without displaying the execution arrow for each step, enabling the test or component to run faster.
- Decide if and when you want to capture and save images and/or movies of the application for the run results. You can reduce disk space and improve test run time by saving screen captures and movie segments only in certain situations, such as when errors occur, or by not saving them at all. To do this, use the **Save still image captures to results** and **Save movie to results** options in the **Screen Capture** pane in the Options dialog box (**Tools > Options > GUI Testing tab > Screen Capture** node).
- If you are using Insight test objects, adjust the number and size of snapshots saved with the test objects.

In the object repository, you can delete all of the snapshots stored with the Insight test objects after you finalize the test object images and verify that they enable correct object identification in all

relevant scenarios. (In the Object Repository window or the Object Repository Manager, **Tools > Delete Insight Snapshots.**)

- Save the run results report to a temporary folder to overwrite the results from the previous run session every time you run a test or component..
- **For components:** Try to use the same application area for all components in a business process test, as this improves performance.
- **For tests:** Minimize the number of actions in a test. Ideally, a test should not contain more than a few dozen actions.
- **For tests:** Store your functions in function libraries instead of as reusable actions.
- **For tests:** Remove unwanted or obsolete run results from your system, according to specific criteria that you define. This enables you to free up valuable disk space.
- **For tests and scripted components:** If you are not using the Active Screen while editing your test, hide the Active Screen while editing your test to improve editing response time by right-clicking the Active Screen pane and selecting **Hide**.
- **For tests and scripted components:** Decide if and how much information you want to capture and save in the Active Screen. The more information you capture, the easier it is to add steps to your test or component using the many Active Screen options, but more captured information also leads to slower recording and editing times. You can choose from the following Active Screen options to improve performance:
  - If you are testing Windows applications, you can choose to save all Active Screen information in every step, save information only in certain steps, or to disable Active Screen captures entirely. You set this preference in the **Active Screen** pane of the Options dialog box.
  - If you are testing Web applications, you can disable screen capture of all steps in the Active Screen. From the **Active Screen** pane of the Options dialog box, click **Custom Level** to open the Custom Active Screen Capture Settings dialog box.

Select the **Disable Active Screen Capture** option. This will improve recording time.

- When you save a new test or component, or when you save a test or component with a new name using **Save As**, you can choose not to save the captured Active Screen files with the test or component by clearing the **Save Active Screen files** option in the Save or Save As dialog box. This is especially useful when you have finished designing your test or component and you plan to use your it only for run sessions. Tests and components without Active Screen files open more quickly and use significantly less disk space.

**Tip:** (For tests and scripted components) If you need to recover Active Screen files after you save a test or component without Active Screen files, re-record the necessary steps or use the **Update Run Mode** option to recapture screens for all steps.

- Decrease the timeout settings for your application. These settings depend on the application, objects in the application being tested, and the operation being run on the object. You can find these settings in the following locations:

- In the **Run** pane of the Settings dialog box (**File > Settings > Run**), decrease the Object Synchronization timeout.
- in the Web pane of the Settings dialog box (**File > Settings > Web**), decrease the Browser Navigation timeout.
- In the Run pane of the Settings dialog box (**File > Settings > Run**), disable Smart Identification by selecting the **Disable Smart Identification during the run session** option.  
If you need smart identification to assist in identification during a run session, minimize the amount of time that the smart identification mechanism is used by fine-tuning your object's properties.  
Smart identification slows the run session because it is used only after the object synchronization timeout is reached.

**Tip:** If steps in your test or component seem to take a long time, try the following:

- a. Open the run results and expand all nodes in the results.
- b. If you see many Smart Identification icons , try to improve your object descriptions so that UFT does not have to use smart identification as much.
- c. To improve your object descriptions you can:
  - In the run results, in the step that used smart identification, see which properties UFT used to uniquely identify the object, and add those properties to the object description in the object repository.
  - Use the **Update test object descriptions** option of **Update Run Mode** to change the set of properties used for object identification for all objects in a specific class.

- Save tests on the file system instead of network drives.
- If you are using the HP Functional Testing Concurrent License Server, create the LSFORCEHOST variable to force UFT to search for an available license on a specific machine and omit the broadcasting performed each time you open UFT. For details on the LSFORCEHOST variable, see the *HP Functional Testing Concurrent License Server Installation Guide*.

## How can I decrease the disk space used by UFT for GUI tests and components?

You can decrease the disk space used by UFT by doing any of the following:

- Decide if and when you want to capture and save images and/or movies of the application for the run results. You can reduce disk space and improve test run time by saving screen captures and movie segments only in certain situations, such as when errors occur, or by not saving them at all. To do this, use the **Save still image captures to results** and **Save movie to results** options in the **Screen Capture** pane in the Options dialog box (**Tools > Options > GUI Testing tab > Screen Capture** node).
- If you are using Insight test objects, adjust the number and size of snapshots saved with the test objects. These settings can also affect UFT performance when recording and running Insight steps.

In the object repository, you can delete all of the snapshots stored with the Insight test objects after you finalize the test object images and verify that they enable correct object identification in all relevant scenarios.

- **For tests and scripted components:** Decide if and how much information you want to capture and save in the Active Screen. The more information you capture, the easier it is to add steps to your test using the many Active Screen options, but more captured information also leads to slower recording and editing times. You can choose from the following Active Screen options to improve performance:
  - If you are testing Windows applications, you can choose to Save all Active Screen information in every step, save information only in certain steps, or to disable Active Screen captures entirely. You set this preference in the **Active Screen** pane of the Options dialog box.
  - If you are testing Web applications, you can disable screen capture of all steps in the Active Screen. From the Active Screen pane, click **Custom Level** to open the Custom Active Screen Capture Settings dialog box. Select the **Disable Active Screen Capture** option. This will improve recording time.
  - When you save a new test, or when you save a test with a new name using Save As, you can choose not to save the captured Active Screen files with the test by clearing the **Save Active Screen files** option in the Save or Save As dialog box. This is especially useful when you have finished designing your test and you plan to use your test only for test runs. Tests without Active Screen files use significantly less disk space.

**Tip:** (For tests and scripted components) If you need to recover Active Screen files after you save a test without Active Screen files, re-record the necessary steps or use the **Update Run Mode** option to recapture screens for all steps in your test.

## For tests: Is there a recommended length for tests?

Although there is no formal limit regarding test length, it is recommended that you divide your tests into actions and that you use reusable actions in tests, whenever possible. An action should contain no more than a few hundreds steps and, ideally, no more than a few dozen. For details, see "[Actions in GUI Testing](#)" on page 103.

**Note:** See also "[Improving the working speed \(performance\) of UFT](#)" on page 1139

# Appendix E: UFT Troubleshooting and Limitations

**Relevant for: GUI tests and components and API testing**

This chapter includes:

- Troubleshooting and Limitations - UFT Program Management ..... 1138
- Troubleshooting and Limitations - Multilingual Applications in GUI Testing ..... 1141
- Troubleshooting and Limitations - Operating Systems in GUI Testing ..... 1143
- Troubleshooting - Naming Conventions ..... 1144

# Troubleshooting and Limitations - UFT Program Management

## **Relevant for: GUI tests and components and API testing**

This section describes troubleshooting and limitations for UFT tools and program management, and covers the following topics:

- "Tools in the HP UFT program group" below
- "Characters do not display correctly" on the next page
- "Improving the working speed (performance) of UFT" on the next page
- "Improving the working speed (performance) of UFT" on the next page

## **Tools in the HP UFT program group**

You cannot use some of the tools from the **HP UFT > Tools** program group when the UAC (User Account Control) option is set to ON. (Relevant for Windows 7, Windows 8, Windows Server 2008 R2, and Windows Server 2012)

**Workaround:** Temporarily turn off the UAC option while using these tools, by doing the following:

### **For Windows Server 2008:**

1. Log in as an administrator.
2. From the Control Panel, select **User Accounts > Change Security Settings**, and clear the **Use User Account Control (UAC) to help protect your computer** check box.
3. Restart the computer to enable this setting to take effect.
4. After working with the desired tool, return to the **Change Security Settings** screen, and select the **Use User Account Control (UAC) to help protect your computer** check box to turn on UAC again. Restart your computer if necessary.

### **For Microsoft Windows 7 and Windows Server 2008 R2:**

1. Log in as an administrator.
2. From the Control Panel, select **User Accounts > User Accounts > Change User Account Settings**.
3. In the User Account Control Settings window, move the slider to **Never notify**.
4. Restart the computer to enable this setting to take effect.
5. After working with the desired tool, return to the User Account Control Settings window, and restore the slider to its previous position to turn the UAC option on again.

**For Microsoft Windows 8:**

1. Log in as an administrator.
2. From the Control Panel, select **User Accounts and Family Safety > User Accounts > Change User Account Control Settings**.
3. In the User Account Control Settings window, move the slider to **Never notify**.
4. In the Control Panel, select **System and Security > Administrative Tools > Local Security Policy**.
5. In the Local Security Policy window, in the left pane, select **Local Policies**.
6. In the Local Policies tree, select **Security Options**.
7. In the right pane, select the **User Account Control: Run all administrators in Admin Approval mode** option.
8. Select **Action > Properties** from the menu bar.
9. In the dialog that opens, select **Disabled**.
10. Restart the computer for your changes to take effect.
11. After working with the desired tool, return to the User Account Control Settings window, and restore the slider to its previous position to turn the UAC option on again.
12. Restart the computer for your changes to take effect.

**Characters do not display correctly**

UFT does not fully support UTF-16 surrogate pairs and combining characters. The Run Results Viewer and some user interface elements in UFT do not display these characters correctly.

**UFT and DEP (Data Execution Prevention)**

On Windows 7 64-bit and Windows Server 2008 R2 operating systems, UFT behaves unexpectedly when the DEP (Data Execution Prevention) flag is set to **Always On**.

**Workaround:** Switch off the DEP, or modify its settings to be ON only for important operating system processes.

**Improving the working speed (performance) of UFT**

- If you receive a "running low on UFT resources" error message in the Windows system tray, see "[Windows Notification Area Messages](#)" on the next page.
- See also "[Improving GUI Testing Performance](#)" on page 1133.

## Windows Notification Area Messages

### **Relevant for: GUI tests and components and API testing**

UFT sometimes generates system messages in the Windows notification area.

- **UFT resources are running low.** Close any UFT documents that you do not need or remove assets from the solution. If this problem continues, restart UFT.

This message displays when UFT consumes too much memory.

To reduce UFT memory usage and improve performance, you can try one or more of the following:

- Include fewer items in your solution.
- Open fewer UFT documents simultaneously.
- Create shorter tests. You can do this by dividing your test into smaller tests, with each test covering fewer scenarios or use cases.
- Verify that the UFT installation meets all of the minimum system requirements as specified in the *HP Unified Functional Testing Readme*.

In addition, to alleviate the memory shortage, close other unneeded applications that are running on your computer.

If the problem continues, restart UFT.

# Troubleshooting and Limitations - Multilingual Applications in GUI Testing

## Relevant for: GUI tests and components

This section includes general limitations about multilingual issues in UFT:

- UFT may behave unexpectedly when using multi-byte string inputs if it is installed on a VMware operating system.

**Workaround:** Set the hardware acceleration of the display driver to **None**. If this does not work, uninstall the VMware display adapter.

- There may be cases during a run session when UFT loses the focus of the application you are testing when working in a Korean, Chinese, or Japanese operating system. This may result in a loss of data during the run session.

**Workaround:** Run an **Activate** method on the application's window to ensure focus on the window before performing the next step. For example:

```
Window("Notepad").Activate
```

- When using Chinese IME to record multiline objects:

- Some mouse operations on the IME are recorded and are not executed during the run session.

**Workaround:** Avoid performing mouse operations on the IME window while recording.

- Selecting a character from the IME Candidate window is not supported.
- Running the **Type** method on computers with Japanese, Korean, or Chinese operating systems may not work as expected.

**Workaround:** Add an English input locale to the computer (using the **Regional Options** or **Regional and Language Options** in your Control Panel).

Additional product-area specific limitations are described in the chapters that describe the relevant functionality.

## Working with a localized version of UFT:

- Some user interface items are not localized, for example:
  - Tooltips in the Properties Pane
  - Tooltips and context menu in the editor of the File Content Checkpoint dialog box
  - Compilation information in the Output pane for an API Test

- Context menu for the search box in the toolbar of the File Content Checkpoint Properties dialog box
- Error messages when importing a WSDL Web service
- The default value string for a Date parameter in the Add Input Parameter dialog box for an API test
- Warning or error messages in the Test Batch Runner, for example, when saving files using the **Save As** command
- Strings displayed when adding a date-type input parameter in the Properties pane of an API test
- Some of the UFT user interface related to XML operations
- XML Validation results in the Run Results Viewer

**Workaround:** Install the relevant user interface language .NET Framework Language Pack. You can download it from the Microsoft download center: <http://www.microsoft.com/en-us/download/default.aspx>

- The image displayed when selecting **View Sample Snapshot** in the UFT Asset Comparison tool is not localized

# Troubleshooting and Limitations - Operating Systems in GUI Testing

## **Relevant for: GUI tests and components**

Some operating systems have specific characteristics that affect the normal behavior of UFT functionality. These issues are described in the chapters that describe that functionality. The table below groups these issues by operating system so that you can find all issues that are specific to the operating system you use.

Operating System	Affected Functionality
<b>Windows 7</b>	<a href="#">Recording tests and components</a>
	<a href="#">Using UFT tools from the Windows Start menu</a>
	<a href="#">Opening and saving tests</a>
	<a href="#">DEP (Data Execution Prevention) - 64-bit only</a>
<b>Windows Server 2008 R2</b>	<a href="#">Recording tests and components</a>
	<a href="#">Exporting data tables</a>
	<a href="#">Using UFT tools from the Windows Start menu</a>
	<a href="#">DEP (Data Execution Prevention) - see <i>HP Unified Functional Testing Installation Guide</i></a>

# Troubleshooting - Naming Conventions

## Relevant for: GUI tests and components and API testing

The following table lists the restrictions to consider when naming items in UFT:

Item	Naming Convention
<b>Action (for GUI tests)</b>	<ul style="list-style-type: none"><li>Must be unique in the test.</li><li>Cannot be named <b>Global</b>, since Global is the name reserved for the Global sheet in the Data pane. If you create an action called Global, you will not be able to select the local or global data sheet when parameterizing a identification property.</li><li>Cannot begin or end with a space.</li><li>Cannot contain the following characters: \ / : * ? " &lt; &gt;   % ' ! { }</li></ul>
<b>Action parameter (for tests)</b>	<ul style="list-style-type: none"><li>Case-sensitive.</li><li>Must begin with a letter</li><li>Cannot contain spaces.</li><li>Cannot contain the following characters: ! @ # \$ % ^ &amp; * ( ) + = [ ] \ { } ; ' : " , . / &lt; &gt;</li></ul>
<b>Action (for API tests)</b>	<ul style="list-style-type: none"><li>Cannot begin or end with a space</li><li>Cannot exceed 1,023 characters</li><li>Cannot contain the following characters: \ / : * ? " &lt; &gt;   % ' ! { }</li></ul>
<b>Application area (for components)</b>	<ul style="list-style-type: none"><li>Cannot exceed 220 characters (including the path).</li><li>Cannot contain, begin, or end with spaces.</li><li>Cannot contain the following characters: \ / : " ? &lt; &gt;   * ! { } ' % ;</li><li>Cannot contain multibyte punctuation symbols and other multibyte special characters, such as multibyte question marks, multibyte spaces, and multibyte brackets.</li></ul>
<b>Component parameter</b>	<ul style="list-style-type: none"><li>Cannot contain brackets in the name (e.g. <b>{Param1}</b>)</li></ul>
<b>Checkpoint</b>	<ul style="list-style-type: none"><li>Cannot begin or end with a space.</li><li>Cannot contain " (double quotation mark).</li><li>Cannot contain the following character combinations:<ul style="list-style-type: none"><li>:=</li><li>@@</li></ul></li></ul>
<b>Component (for components)</b>	<ul style="list-style-type: none"><li>Cannot exceed 220 characters (including the path).</li><li>Cannot contain, begin, or end with spaces.</li><li>Cannot contain the following characters: \ / : " ? &lt; &gt;   * ! { } ' % ;</li><li>Cannot contain multibyte punctuation symbols and other multibyte special characters, such as multibyte question marks, multibyte spaces, and multibyte brackets.</li></ul>

Item	Naming Convention
<b>Data Table file (for tests and scripted components)</b>	<b>ALM:</b> cannot contain the following characters: ! % * {} \   ' : " / < > ? ; ,
<b>Data Table &gt; Parameter name (column header)  (for tests and scripted components)</b>	<ul style="list-style-type: none"> <li>Must be unique in the sheet.</li> <li>Must begin with a letter or underscore.</li> <li>Can only contain the following:             <ul style="list-style-type: none"> <li>Letters (excluding special characters, machine-dependent characters, or platform-dependent characters)</li> <li>Numbers</li> <li>Periods</li> <li>Underscores</li> </ul> </li> </ul>
<b>Environment variable (parameter)</b>	<ul style="list-style-type: none"> <li>Must begin with a letter.</li> <li>Can only contain the following:             <ul style="list-style-type: none"> <li>Letters</li> <li>Numbers</li> <li>Underscores</li> </ul> </li> </ul>
<b>Environment variables file</b>	<b>File system:</b> Cannot contain the following characters: ! % * {} \   ' : " / < > ? ; <b>ALM:</b> Cannot contain the following characters: ! % * {} \   ' : " / < > ? ; ,
<b>Function library file</b>	<b>File system:</b> Cannot contain the following characters: ! % * {} \   ' : " / < > ? ; <b>ALM:</b> <ul style="list-style-type: none"> <li>Cannot contain the following characters: ! % * {} \   ' : " / &lt; &gt; ? ; ,</li> <li>Cannot contain non-English characters</li> </ul>
<b>Function / Function argument</b>	<ul style="list-style-type: none"> <li>Cannot contain non-English letters or characters.</li> <li>Function names cannot be the same as a VBScript registered keyword.</li> <li>Must begin with a letter.</li> <li>Cannot contain spaces or any of the following characters: ! @ # \$ % ^ &amp; * ( ) + = [ ] \ { }   ; ' : " , / &lt; &gt; ?</li> </ul>
<b>Object repository file</b>	<b>File system:</b> Cannot contain the following characters: ! % * {} \   ' : " / < > ? ; <b>ALM:</b> Cannot contain the following characters: ! % * {} \   ' : " / < > ? ; ,
<b>Output value</b>	<ul style="list-style-type: none"> <li>Cannot begin or end with a space.</li> <li>Cannot contain " (double quotation mark).</li> <li>Cannot contain the following character combinations:             <ul style="list-style-type: none"> <li>:=</li> </ul> </li> </ul>

Item	Naming Convention
	<ul style="list-style-type: none"> <li>• @@</li> </ul>
<b>ALM file or folder name</b>	<ul style="list-style-type: none"> <li>• Cannot exceed 90 characters (including the path).</li> <li>• Cannot begin or end with a space.</li> <li>• Cannot contain the following characters: \ : * ? " &lt; &gt;   % ' ;</li> <li>• Cannot contain multibyte punctuation symbols and other multibyte special characters, such as multibyte question marks, multibyte spaces, and multibyte brackets.</li> </ul>
<b>Recovery Scenario</b>	<p><b>File system:</b> Cannot contain the following characters: ! % * {} \   ' : / &lt; &gt; ? ;</p> <p><b>ALM:</b> Cannot contain the following characters: ! % * {} \   ' : / &lt; &gt; ? ;,</p>
<b>Test name (for tests)</b>	<ul style="list-style-type: none"> <li>• Cannot exceed 220 characters (including the path).</li> </ul> <p><b>Note:</b> When you create a test and store it in the file system, UFT creates a folder with the name you specify, and also a file with the same name inside that folder. The name of this file, including its path, cannot exceed 220 characters.</p> <ul style="list-style-type: none"> <li>• Cannot begin or end with a space.</li> <li>• Cannot contain the following characters: \ / : * ? " &lt; &gt;   % ' ;</li> <li>• Cannot contain multibyte punctuation symbols and other multibyte special characters, such as multibyte question marks, multibyte spaces, and multibyte brackets.</li> </ul>
<b>Test resources (when saving a test with resources)</b>	The path name (resource name and file path together) cannot exceed 256 characters.
<b>Test object class (extensibility only)</b>	<ul style="list-style-type: none"> <li>• Cannot contain non-English letters or characters.</li> <li>• Cannot contain spaces or any of the following characters: ! @ # \$ % ^ &amp; * ( ) + = - [ ] \ { }   ; ' : "" , / &lt; &gt; ?</li> </ul>
<b>Test object name</b>	<ul style="list-style-type: none"> <li>• must be unique within the same class and hierarchy in the object repository</li> <li>• Cannot begin or end with a space.</li> <li>• Cannot contain " (double quotation mark).</li> <li>• Cannot contain the following character combinations: <ul style="list-style-type: none"> <li>• :=</li> <li>• @@</li> </ul> </li> </ul>
<b>Test object identification properties</b>	<ul style="list-style-type: none"> <li>• Cannot contain non-English letters or characters.</li> <li>• Cannot contain spaces or any of the following characters: ! @ # \$ % ^ &amp; * ( ) + = [ ] \ { }   ; ' : "" , / &lt; &gt; ?</li> </ul>
<b>Test object method / Test object method argument</b>	<ul style="list-style-type: none"> <li>• Cannot contain non-English letters or characters.</li> <li>• Cannot contain spaces or any of the following characters: ! @ # \$ % ^ &amp; * ( ) + = [ ] \ { }   ; ' : "" , / &lt; &gt; ?</li> </ul>

# Send Us Feedback



Can we make this User Guide better?

Tell us how: [docteam@hpe.com](mailto:docteam@hpe.com)

