

HP Unified Functional Testing

Software Version: 12.53

User Guide



Legal Notices

Warranty

The only warranties for Hewlett Packard Enterprise Development LP 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. HPE 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 HPE 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 Enterprise Development LP

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.hpe.com>.

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

Support

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

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

HPE 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 HPE 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 HPE Passport user and sign in. Many also require a support contract. To register for an HPE Passport ID, go to: <https://softwaresupport.hpe.com> and click **Register**.

To find more information about access levels, go to:

<https://softwaresupport.hpe.com/web/softwaresupport/access-levels>.

HPE Software Solutions & Integrations and Best Practices

Visit **HPE Software Solutions Now** at <https://softwaresupport.hpe.com/group/softwaresupport/search-result-/facetsearch/document/KM01702710> to explore how the products in the HPE Software catalog work together, exchange information, and solve business needs.

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

Contents

HP Unified Functional Testing	1
Part 1: UFT Introduction	35
Chapter 1: Overview	36
UFT program use	39
Licensing	39
Required permissions for UFT	40
Required permissions for ALM	41
Required permissions for BPT	41
Demo applications	41
Accessibility	42
Unicode Compliancy	42
Known Issues- Multilingual Applications	43
Known Issues - UFT Tools and Programs	45
UFT program tools	45
Characters do not display correctly	46
UFT and DEP (Data Execution Prevention)	46
Chapter 2: UFT Licensing	47
Seat licenses	47
Concurrent licenses	47
Commuter licenses	47
Install licenses using the License Wizard	48
Install licenses from the command line	51
UFT licensing FAQs	53
Can I use my old license (from before UFT 12.50) with the new License Server?	54
How do I get a new license?	54
What if I have trouble regenerating licenses on the HP License Portal?	54
Which license should I install?	54
How do I install the Autopass License Server?	55
Can I install the License Server with silent installation?	55
If I am using concurrent licenses, how do I get UFT to work with the License Server?	55
How do I install licenses if I am deploying UFT across an enterprise network?	56
How do I manage the concurrent licenses on the License Server?	56
Can I set up my License Server to work with a redundant (backup) License Server?	56
What is a cleanup license?	56
My demo license is expiring early. What can I do?	57
Known issues with UFT licenses	57
Chapter 3: UFT Document Management	58
UFT and version control systems	59

Set up UFT to work with GIT	59
Update changes for a document	59
Commit changes for a document	60
Compare a document with the repository version	61
Revert a document	61
Resolve conflicts between document versions	61
Relative paths for test resources	62
Portable copies of tests	63
Opening tests with locked resources	64
Upgrading documents from previous versions	64
Upgrading QuickTest tests or components	64
For Service Test tests or components	66
Naming conventions	70
Known Issues - Opening documents	74
 Part 2: UFT Panes	76
Chapter 4: Active Screen Pane	77
Saving Active Screen content	77
The Active Screen and Insight	78
The Active Screen and Web-based applications	78
Stop saving Active Screen information	78
Update a single Active Screen capture	79
Chapter 5: Bookmarks Pane	80
Chapter 6: The Canvas	81
Chapter 7: Data Pane	82
Data table content	82
Data table values	82
Changing column headers	82
Importing Excel files to the Data Pane	82
Data Pane Specifications	83
Chapter 8: Debug Panes	84
Chapter 9: Document Pane	85
Chapter 10: Errors Pane	86
Code syntax errors	86
Missing resources	86
Missing references	87
Missing property values	87
Chapter 11: Output Pane	88
Chapter 12: Properties Pane	90
Chapter 13: Run Step Results Pane	91
Chapter 14: Search Results Pane	92

Chapter 15: Solution Explorer Pane	93
Chapter 16: Tasks Pane	94
Manage TODO comments	94
Chapter 17: Toolbox Pane	95
The Toolbox pane and GUI testing	95
The Toolbox pane and API testing	95
The Toolbox pane and Business Process Testing	96
Part 3: UFT Configuration	97
Chapter 18: Global Options	98
Chapter 19: Document Settings	100
Chapter 20: Set Options Programmatically	102
Part 4: GUI Test Design	104
Chapter 21: GUI Test Creation Overview	105
Keyword-driven methodology	105
Recording	106
Importing tests from Sprinter	106
Enhancing your tests	107
Checkpoints	107
Parameterization	107
Output Values	107
Programming Statements	107
Active Screen Updates	108
Create a keyword-driven GUI test	108
Analyze your application	108
Prepare the testing infrastructure	109
Add steps to the actions in your test action repository	111
Enhance your test	111
Sample test	111
Chapter 22: Record GUI Tests and Components	113
Normal Recording	113
Analog Recording	114
Record a GUI test or component	116
Prerequisites	116
Start a recording session	116
Record steps into the test	116
Use the Record toolbar to manage your recording session	117
Capture objects to an object repository	117
Switch to other recording modes	117
Known Issues When Recording	119

Chapter 23: Actions in GUI Testing	121
Structure your test with actions	122
Display / modify action data	123
Create an action template	123
Action and action call properties	124
Exit an action using programming statements	124
Known Issues with Actions	125
Copies of tests	125
Chapter 24: Keyword View	126
Standard steps in the Keyword View	127
Define or modify an item value	127
Parameterize an Item value for an argument	128
Encode a password	128
Add a standard step after a conditional or loop block	129
Comments in the Keyword View	129
Comments in actions	129
Comments in components	130
Conditional and loop statements	130
Known Issues - Keyword View	133
Chapter 25: Running API Tests with GUI Tests	134
Using API tests in a GUI test - Use-case scenario	137
Calling API tests with parameters - Use-case scenario	139
Chapter 26: Maintaining Tests or Components	147
Application errors	147
Application changes	147
Missing objects	148
Maintenance Run mode	148
When do you use Maintenance Mode?	149
Maintenance Run Mode prerequisites	150
Determine UFT wait time	150
Run the test or component in Maintenance Run Mode	150
Merge changes to your shared object repository	151
Maintenance Run Wizard Workflow	152
Update Run mode	153
Smart Identification	153
Update test object descriptions, checkpoints, or output values, or Active Screen captures	155
Run in Update Run Mode	156
Export and merge changes	156
Analyze the results	156
Known Issues in Maintenance and Update Run Modes	157
Chapter 27: Recovery Scenarios	159

When to use recovery scenarios	160
Programmatically controlling the recovery mechanism	161
Manage recovery scenarios	161
Create a new recovery scenario operation	162
Associate a recovery scenario in the Test Settings	162
Associate a recovery scenario in the Solution Explorer	162
Associate a recovery scenario with a component/ application area	162
Enable/disable specific recovery scenarios	163
Set default recovery scenario settings	163
Known Issues- Recovery Scenarios	164
Chapter 28: Using Performance Testing and Business Service Management Products with UFT GUI Tests	165
Designing tests for HP performance testing products	166
Running GUI tests from HP performance testing products	167
Running GUI tests from HP Business Process Monitor	167
Measuring transactions	168
Insert and run GUI tests in Performance Center and LoadRunner	170
Silent Test Runner	170
Part 5: Test Objects / Checkpoints / Output Values	172
Chapter 29: The Test Object Model	173
How UFT learns objects	173
How UFT applies the test object model concept	174
Test object descriptions	176
UFT test object hierarchy	178
Properties and operations for test objects and run-time objects	178
How UFT identifies objects during a run session	180
Object identification process workflow	183
Use the Object Spy	184
Prerequisites	184
Use the pointing hand to select the application object	184
Add objects to the object repository	185
Use the Remote Object Spy	185
Prerequisites	185
Open the Remote Object Spy dialog box	186
Use the mouse to select the application object	186
View object details or add it to your object repository	186
Chapter 30: Configuring Object Identification	188
Mandatory and assistive properties	189
Ordinal identifiers	190
Index identifiers	190

Location identifiers	191
Visual relation identifiers	192
How UFT interprets horizontal and vertical visual relations	193
Smart identification	195
The Smart identification process	196
How UFT uses smart identification - Use-case scenario	197
Test object mapping for unidentified or custom classes	199
Configure object identification for a test object class	199
Set properties for identifying an object	199
Set properties for Smart Identification	200
Map an unidentified or custom class to a Standard Windows class	200
Define a visual relation identifier for a test object - Use-Case scenario	201
Background	201
Open the Visual Relation Identifier dialog box	202
Highlight the objects that match the test object's description	202
Define the first related test object using horizontal visual relations	203
Define the second related object using vertical visual relations	204
Define the third related object using distance visual relations	204
Results	205
Chapter 31: Test Objects in Object Repositories	206
The Object Repository window vs. the Object Repository Manager	208
aLocal copies of objects in shared object repositories	209
Adding and deleting test objects	210
Maintaining identification properties	211
Repository parameters	213
Repository parameter value mappings	214
Managing shared object repositories using automation	215
Add a test object to an object repository	216
Add test objects to the object repository	216
Add an Insight test object to the object repository	217
Add a test object to the local object repository while adding a step	217
Define a new test object	217
Add a test object to the object repository with the Object Spy	218
Add a test object to the local object repository from the Active Screen	218
Add a test object to the local object repository by inserting a step from the Active Screen	218
Maintain test objects in object repositories	219
Specify an identification property value	219
Update identification properties	219
Restore the mandatory property set	220
Rename test objects	220
Add properties to a test object description	220

Define a new identification property	220
Remove properties from a test object description	221
Specify an ordinal identifier	221
Define related objects for a specific test object	221
Export the objects from a local object repository	222
Copy an object to the local object repository	222
Modify identification properties during a run session	223
Create and manage shared object repositories	223
Prerequisites	223
Enable editing for a shared object repository	223
Associate a shared object repository with actions or components	223
Merge object repositories into shared ones	224
Manage repository parameters	224
Import a shared object repository from XML	225
Export a shared object repository to XML	225
Locate an object in an object repository	225
Find an object	225
Highlight an object in your application	226
Locate an object from your application in the object repository	226
Known Issues- Object Repositories	227
Chapter 32: Comparing and Merging Object Repositories	228
Object conflicts	229
Different Objects with the Same Name Conflict	230
Identical Description Different Name Conflict (Test Objects Only)	230
Similar Description Conflict (Test Objects Only)	231
Compare two object repositories	231
Prerequisites	232
Select the Shared Object Repositories to compare	232
Analyze the initial comparison results	232
Analyze the detailed comparison results	232
Utilize additional tools to help you perform the comparison - optional	232
Merge two shared object repositories	232
Prerequisites	233
Select the shared object repositories to merge	233
Analyze the initial merge results	233
Analyze the detailed merge results	233
Utilize additional tools to help you perform the comparison - optional	233
Adjust object conflict resolutions	233
Save the target object repository	234
Update a shared object repository from a local object repository	234
Prerequisites	234

Select the shared object repository and the local repositories that you want to merge into it	234
Analyze the initial merge results	235
Analyze the detailed merge results	235
Adjust object conflict resolutions	235
Save the target object repository	235
Chapter 33: Extending UFT Object Identification	236
Identifying objects using Insight	236
Creating Insight test objects	237
Creating steps with Insight Test Objects	238
Running tests with Insight	238
Work with Insight test objects	239
Add an Insight object	239
Modify an Insight test object's image	240
Retrieve text from an Insight Object	241
Update Insight test object details	241
UI Automation in UFT	242
Enable UI Automation support	243
How does UFT use the UI Automation framework?	243
Control Types and UFT Test Objects	243
Supported Patterns and Test Object Methods	244
When should you use UFT UI Automation support?	246
Native UI Automation methods	251
Use UFT UI Automation support	253
Learn objects in UI Automation mode	253
Record steps in UI Automation mode	254
Identifying unsupported objects in test runtime	255
Create a programmatic description of the object	256
Set the unsupported/unidentified object as Static object type	257
Assign the unsupported/unidentified object to a supported object type	257
Use non-test object methods	257
Known Issues - UI Automation Support	258
Chapter 34: Virtual Objects	260
How virtual objects are defined and recognized	260
Define virtual objects for unsupported objects	261
Display the object to define as a virtual object	261
Use the Virtual Object wizard	261
Chapter 35: Checkpoints in GUI Testing	262
Adding existing checkpoints	262
Checkpoint types	263
Standard checkpoints	266
Accessibility checkpoints	267

Bitmap checkpoints	267
Fine-tuning the bitmap comparison	269
Database checkpoints	271
File Content checkpoints	272
Table checkpoints	273
Page checkpoints	273
Text and text area checkpoints	274
Text recognition in run-time	275
Checking Text in an image - Use-case scenario	277
XML checkpoints	279
Insert a checkpoint step	280
Important information before inserting the checkpoint	281
Define automatic page checkpoints - optional	282
Set global accessibility checkpoint preferences	283
Insert a checkpoint step while recording your test or component	283
Insert a checkpoint step while editing your test or component	283
Use programming to insert checkpoints in a test or scripted component	285
Set options for the checkpoint	285
Move checkpoint objects from the local object repository to a shared object repository - optional	287
Include and ignore areas when comparing a bitmap - Use-case scenario	288
Configure text recognition settings	291
Prerequisites	291
Analyze the characteristics of the text	291
Set the appropriate options	291
Check the text recognition settings	293
Adjust the settings as necessary	293
Known Issues- Using Checkpoints	294
Chapter 36: Developing Custom Comparers for Bitmap Checkpoints	296
Custom bitmap comparer development	297
Custom comparer for images whose location changes - Use-case scenario	297
Develop a custom comparer	299
Prerequisites	299
Develop the custom comparer COM object	299
Prepare the custom comparer installation	299
Install the custom comparer	300
Test the custom comparer	300
Implement the bitmap comparer Interfaces	300
Prerequisite - Reference the type library	301
Implement the CompareBitmaps method	301
Implement the CompareBitmaps method	301
Implement IBitmapCompareConfiguration	302

Install your custom comparer and register it to UFT	303
Prerequisites	303
Install the custom comparer COM object on the UFT computer	304
Place the custom comparer documentation in the correct location	304
Set the Custom Comparer Name - Optional	305
Results	305
Use the bitmap checkpoint custom comparer samples	306
Prerequisites	306
Generate the sample comparer	306
Install the custom comparer on a UFT computer	306
Register the custom comparer to the component category	307
Study the custom comparer functionality	308
Develop a custom comparer - Tutorial	309
Create a new ATL project—SampleCPPCustomComparer	309
Create a new class—CBitmapComparer	310
Implement the comparer interfaces for the CBitmap Comparer class	310
Move the function bodies for the comparer interface methods	311
Implement the bitmap checkpoint comparer interface methods	312
Design your custom comparer to register to the component category	314
Compile and run your DLL	316
Test your custom comparer	316
Bitmap checkpoint comparer Interfaces	317
Chapter 37: Output Values in GUI Testing	322
Output Value types	322
Storing output values	324
Create or modify an output value step	326
Chapter 38: Parameterizing Object Values	330
Data table parameters	331
Global and local (Action) data table parameters	333
Environment variable parameters	335
Automatically parameterizing steps	337
Data Driver	338
Parameterize values for operations	339
Parameterize a value for an operation	339
Use the Data Driver to parameterize a constant value	339
Enter parameters as values in the Editor	339
Parameterize a checkpoint property value	340
Parameterize a value for a property in a checkpoint	340
Use the Data Driver to parameterize a constant value	340
Enter parameters as values in the Editor	340
Test and action parameters	340
Sharing action information	343

Adding action parameters as steps in the Editor	346
Use action parameters	346
Data tables and sheets in GUI tests and components	348
Using different data tables	349
Formulas in data tables	350
Define and manage data tables	351
Add an external data table file	351
Manually enter information	352
Import information into the data table	352
Add a data table file to your ALM project	352
Define the number of iterations for an action or test	352
Change a column name	353
Use an autofill list	353
Insert formulas into data tables for use in checkpoints	354
Import data using Microsoft Query	354
Use user-defined external environment variables	355
Create an external environment variables file	355
Upload the environment resource file to ALM	355
Use environment variables in your test	355
Results	355
Create an external environment variables file	355
Create an external environment variable file in the Test Settings	356
Create an external environment variable file manually	356
Regular expressions	357
Regular expression characters and usage options	358
Chapter 39: Value Configuration and Parameterization	363
Local and component parameters	363
Configure constant and parameter values	364
Parameterize input values	365
 Part 6: API Test Design	367
Chapter 40: API Test Creation Overview	368
Automatically generating API tests	370
Create an API test	372
Prerequisite - Analyze your application	372
Prerequisite - Configure UFT according to your testing needs	372
Prerequisite - Prepare the service references (optional)	372
Build your test structure	373
Enhance your test steps	374
Results	375
Create an API test - Use-case scenario	376

Analyze the Flight API application	376
Create or import your test resources	378
Create your test steps	379
Enhance your test steps	382
Run the test	383
Automatically generate API tests	383
Generate your test from a WSDL file	383
Generate your test from a SOAPUI test file	384
Chapter 41: Standard Activities	386
Checkpoint validation	386
XPath checkpoints	387
Test with Docker activities	388
Configure ports	388
Pull an image from the Docker registry	389
Create a container	389
Run a container	390
Run an image	390
Add additional test steps	391
Stop the Docker container image	391
Set array checkpoints	391
Enable active content on your computer	391
Add a step with an array output	391
Select an array validation method	392
Validate individual array elements	392
Set XPath checkpoints	393
Add a step with XML output	393
Set the namespace setting	393
Add an XPath checkpoint	393
Use Flow Control activities	393
Add a conditional step	393
Add a loop	395
Add steps to pause and restart the test	396
Add a wait step	397
Send a multipart HTTP or REST Service request	397
Add an HTTP or REST Service step	397
Set the properties for the first part of the request	397
Set the properties for the other parts	397
Create a call to a Java class	398
Implement the UFT API Java interface	398
Compile the Java source code	398
Package your custom step - optional	399
Set up the Java environment - optional	399

Add a Call Java Class activity	399
Set the Java step property values	399
Call external tests or actions	400
Prerequisites	400
Call an API Test or Action or Service Test test	401
Call a GUI Test or QuickTest action or test	401
Add a LoadRunner script activity	402
Prepare and run a Load test	403
Prerequisite	403
Create a load-enabled API test	403
Add test steps	403
Prepare for load testing	403
Set the data retrieval properties - optional	404
Set the run configuration to Release	404
Run in Load Testing mode to validate the test	404
Incorporate the test into LoadRunner	404
Test Web sockets communication	405
Open a Web socket connection	405
Send a message to another Web socket	405
Receive a message from another Web socket	406
Close the Web socket connection	407
Known Issues- Standard Activities	408
System activities	408
Java activities	408
Network	408
Database	409
FTP	409
IBM Websphere MQ	409
JSON	409
Load Testing	409
Web Sockets	410
XPath Activity Checkpoints	410
Chapter 42: Custom Activities	411
Web Services	411
REST Services	411
Web Application Services	412
Network Capture Activities	412
.NET Assemblies	413
SAP-based Services	413
Activity sharing	414
Perform activity sharing	414
Connect to ALM	414

Set up the repository paths	414
Import a WSDL or create a REST service	415
Move the activity to a repository	415
Negative testing of Web services	415
Passing REST service properties	416
Exposing REST service properties	420
Import a WSDL-based Web service	421
Import your service	421
Validate the WSDL file - optional	422
Update your WSDL information - optional	422
Add input attachments to the test - optional	423
Validate output attachments	423
Configure SOAP Fault information - optional	424
Create a REST service model	424
Prerequisite	424
Create the service model hierarchy	424
Set the General properties	425
Set the URL property values	425
Define the method's HTTP properties	425
Define custom properties - optional	425
Enter the request body directly - optional	426
Link the body request to a data source - optional	426
Test the method	427
Save the service model to your test	428
Expose input and output properties	428
Import a REST service model	428
Import the service	429
Set authentication and response settings for a SAP HANA service	429
Use the service's methods in your test	430
Send and receive a JSON request for a REST service	430
Set the HTTP properties	430
Load the request body	430
Add a request header - optional	431
Modify the JSON body - optional	431
Import a Web Application service	432
Prerequisite	432
Import the WADL document	432
Import a Network Capture file	434
Create a capture file	434
Prerequisite - study the structure of your network capture file	434
Import the network capture file	435
Create an SAP API test step	436

Prerequisite	436
Define an SAP Connection	436
Select an iDoc from your SAP server	436
Import and create a .NET Assembly API test step	436
Prerequisite	436
Import a .NET assembly	437
Select a GAC assembly - optional	437
Add an ExecuteEvent event handler	437
Add custom input and output properties - optional	438
Known Issues- Custom Activities	439
Web services	439
REST services	439
Web Application Services	440
Network Capture Activities	440
.NET Assembly Activities	440
Chapter 43: Actions for API Tests	441
Actions and data sources	442
Use actions in an API test	443
Create a new action	443
Call an existing action or a test	443
Set action properties	443
Enable an action's data for editing when called by another test	444
Override data from an action called by another test	444
Chapter 44: Data Usage in API Tests	445
Data assignment in arrays	445
Add data sources to an API test	449
Add an Excel data source	449
Add an XML data source	450
Add a database data source	451
Add a local data table	453
Assign data to API test/component steps	454
Manually enter the data for your test step properties	454
Link your test step to a data source	454
Link your test step to another step	455
Link your test step to multiple sources	455
Link your test steps to a test or user-defined variable	456
Data drive the test step	457
Add data keywords	457
Assign data to API test steps - Tutorial	458
Prerequisite - import the Web Service methods for the sample application	458
Associate a data source with your test	459
Create your test steps	459

Manually enter the input properties for the GetFlights step	459
Link the CreateFlightOrder input properties to the data source	460
Link the Report step input properties to the CreateFlightOrder step	460
View the run results	461
Define API test properties or user/system variables	461
Define test parameters	461
Define user variables	462
Set user variable values	462
Define user variable profiles	462
Set OS variable values for the test - optional	463
Parameterize XML data	463
Data drive array checkpoints	466
Enable active content on your computer	466
Add a step with an array output	466
Data drive the array	466
Set the evaluation expression	467
Provide data for the array	467
Select an array validation method	467
Set the number of iterations - optional	467
Navigating within a data source	468
Parent/Child data source relations	469
Set the data source navigation properties	470
Add a data source to the Test Flow or test loop	470
Specify the navigation properties for the data source	470
Create a new child relation	471
Known Issues- Using data in API tests and components	472
Chapter 45: Updating Services and Assemblies	473
Update a Web Service	474
Prerequisites	475
Update the service	475
Run the Update Port Security wizard - optional	475
Run the Update Step wizard	476
Update a .NET Assembly	476
Select the assembly to update	477
Import a new .NET assembly	477
Handle warnings - optional	477
Modify custom code - optional	477
Resolve conflicts in a REST service test step	477
Modify the step as required	477
Open the wizard	477
Run the wizard	478
Update an SAP RFC or IDoc	478

Update the RFC/IDoc from its original location	478
Update the RFC/IDoc from a different location	478
Run the Update Step wizard	478
Known Issues- Updating Services	480
Chapter 46: Web Service Security	481
Security scenarios	481
Web Service scenario	482
WCF Service scenarios	485
Set security for a standard Web Service	489
Create a Web Service scenario	489
Configure the HTTP settings	489
Add message level security with a Username Token	489
Add message level security by signing with an X.509 Certificate	490
Encrypt a Web service message using a Certificate	491
Send a username token and encrypt the token with an X.509 Certificate	492
Sign and encrypt a Web service message	492
Configure the WS-Addressing (optional)	493
Customize security for WCF-type Web services	493
Create a WCF scenario	493
Configure the settings for a Web Service using WSHTTPBinding	493
Configure the settings for a Web Service using CustomBinding	494
Configure the settings for a WCF Federation Web service	494
Configure the settings for a WCF service using netTcp or namedPipe transport	495
Configure the settings for a Web service using WSE3 security configuration with a server certificate	495
Configure the settings for WCF service using mutual certificate authentication	496
Configure the settings for a WCF scenario using binding with TCP transport to require an X.509 client certificate	496
Set up Advanced Standards testing	497
Test a Web Service using MTOM	497
Change the WS-Addressing version of a service	497
Enable support for a service or activity that uses 256-bit SSL encoding	497
Known Issues- Web Service Security	498
Chapter 47: Asynchronous Service Calls	500
Test an asynchronous Web service	502
Create a test for WS-Addressing	502
Create a test for HTTP Receiver	503
Create a test for a Web service publish subscribe pattern	504
Create a test for Dual WSDL Files	504
Known Issues- Asynchronous Testing	506
Chapter 48: API Testing Extensibility	507
Custom Activity files	507

Runtime files	508
Signature files	509
Property definitions	513
Addin files	517
Resource files	519
Use the Wizard to create a custom Activity - C#	520
Run the Activity Wizard	520
Add execution code	520
Add Logger code - optional	521
Add a Report statement - optional	521
Compile the project into a DLL	521
Deploy the activity in UFT	522
Use the Wizard to create a custom Activity - Java	522
Prerequisite	522
Run the Activity Wizard	522
Edit the code	522
Add Logger code - optional	523
Add a Report statement - optional	523
Compile the Java into a class	523
Deploy the activity	523
Manually create a custom Activity in C#	524
Prerequisite - create a runtime file	524
Create a signature file	524
Create an addin file	525
Provide a graphic for your activity - optional	526
Check the implementation	526
Create a runtime file	526
Add Using statements	526
Specify the namespace and class	527
Set the internal logging	527
Initialize the properties	527
Retrieve the property values	528
Define events	529
Execute the step	529
Set the status	530
Compile the runtime file	530
Known Issues - Extensibility (API Testing)	531
 Part 7: Creating and Enhancing UFT Tests with Code	532
Chapter 49: The Editor	533
Statement completion	533

Automatic code completion	535
Searching and replacing	536
Use code snippets and templates	538
Insert code snippets into your document in the Editor	538
Modify an existing list of code templates	539
Search for references or classes	540
Search for references to the currently selected function or method	540
Search for classes derived from the currently selected class	540
Search for methods that override a virtual method	540
Search for the base class of the current class	541
Known Issues - Statement Completion	541
Chapter 50: Programming Tests	542
Programmatic descriptions	542
Static programmatic descriptions	545
Dynamic programmatic descriptions	548
Retrieving child objects with programmatic descriptions	550
Using the Index property	551
Programmatic description checks	552
Opening and closing applications	553
Comments, control-flow, and other VBScript statements	554
Retrieving and setting identification property values	555
Checkpoint and output statements	556
Native properties and operations	556
Use the Windows API in test steps	557
Basic VBScript syntax	559
General syntax rules and guidelines	560
Formatting text	562
Comments	563
Parameter indications	564
Parentheses	565
Calculations	567
Variables	568
Do...Loop statement	569
For...Each Statement	570
For...Next Statement	571
If...Then...Else Statement	571
While...Wend Statement	573
With Statement	573
Chapter 51: User-Defined Functions	575
Associated function libraries	575
User-defined functions	576
Registered user-defined functions	578

Preparing the user-defined function for registration	580
Registering user-defined functions as test object methods	580
Unregistering user-defined test object methods	581
Running an overriding user-defined test object method	582
Loading function libraries during a run session	584
Manage function library associations	585
View the list of associated function libraries	585
Associate the currently active function library	585
Associate a function library using the Test Settings dialog box	586
Associate a function library with the Solution Explorer pane	586
Associate a function library with an application area	586
Modify the priority of an associated function library	587
Remove a function library association	587
Specify default function libraries for all new tests	587
Load a function library dynamically during a run session	587
Create and work with a user-defined function	588
Prerequisites - Open the function library or test	588
Create the function	588
Register the function to a test object class - optional	589
Associate the function library with a test or application area	590
Call the function	591
Navigate to the function's definition - optional	591
Unregister the function - optional	591
Create and register a user-defined function using the Function Definition Generator	591
Open the function library/test and the Function Definition Generator	591
Specify the details for the function definition	592
Register the function to a test object class - optional	592
Add arguments to the function - optional	592
Add documentation details to the function - optional	593
Insert the function in your active document	593
Add the content (code) of the function	593
Known Issues- Function Libraries and user-defined functions	594
Chapter 52: Generated Programming Operations	597
Message statements	597
Run session messages in the HTML report	597
Run session messages in the Run Results Viewer	598
Step messages in the Run Results Viewer	598
Display messages during the run session	598
Test synchronization	599
Step Generator	601
Generate With statements	602
Instruct UFT to generate With statements while recording	602

Generate With statements for existing actions in the Editor	602
Remove With statements from an action in the Editor	604
Chapter 53: UFT Automation Scripts	605
When to use UFT automation scripts	606
Application Object	607
UFT Automation Object Model Reference	608
Generated automation scripts	608
Create a UFT automation script	609
Prerequisites	609
Create the Application object	610
Run your automation script	613
Run Automation scripts on a remote computer	613
Set DCOM Configuration Properties on the Remote Computer	613
Create an Application Object on the Remote Computer	614
Known Issues- Automation Scripts	615
Chapter 54: Event Handlers for API Test Steps	616
Writing code for API test events	618
Writing events for API tests - Use-case Scenarios	619
Custom code steps	628
Open a window for writing custom code	629
Open the Events tab	630
Select an event	630
Edit the code	631
Manipulate Web Service call/HTTP Request/SOAP Request Step input/output properties	631
Access and set property values for input properties	632
Add checkpoint property values	633
Specify a SOAP Fault and SOAP Fault values	634
Assign a specific request file to a test step	635
Assign a specific request file to a Web service step in the OnSendRequest event	635
Set asynchronous Web service call properties	636
Add an input attachment to a Web service call	637
Access an attachment from a Web service call response	639
Add a HTTP Header for Web Service Calls	640
HTTP Headers for REST Service Calls	641
Modify a SOAP Request security header in runtime	642
Stop an API test run	643
Manipulate data programmatically	644
Prerequisite	644
Retrieve a value from a data source	644
Set a property value from a data source	645
Import a data source file to your test	646

Export the property values to a file	647
Data drive test step property/parameter values	648
Access and set the value of step input, output, or checkpoint properties	650
Access an step property	650
Access a step's parent activity	651
Set the value of a step's properties	652
Access the value of a step's property in runtime	653
Enable or ignore selected checkpoints - optional	654
Set the value of a checkpoint	655
Report test run-time information	656
Report a custom message to the run results	656
Report run-time values to the Output pane	657
Retrieve and set test or user variables	659
Prerequisite - create user variables	659
Optional - set the test profile	659
Retrieve a variable value	659
Set a variable value	660
Encrypt and decrypt passwords	661
Encrypt the password	661
Decrypt the password - optional	661
API test events structure	662
Standard event structure	662
Web Service event structure	665
API Test Event Coding Common Objects	669
Activity Object	669
Assert Object	670
Checkpoint Object	671
CurrentIterationNumber Object	671
EncryptionMngr Object	672
EnvironmentProfile Object	673
InputAttachment Object	673
InputEnvelope Object	675
OutputAttachment Object	676
OutputEnvelope Object	678
Parent Object	678
TestProfile Object	679
UserLogger Object	680
API Test Event Coding Common Methods	680
Export Method	681
ExportToExcelFile Method	682
GetDataSource Method	683
GetValue Method	684

GetVariableNames Method	685
GetVariableValue Method	686
Import Method	687
ImportFromExcelFile Method	688
Info Method	689
Report Method	690
SelectSingleNode Method	690
SetValue Method	691
SetVariableValue Method	692
Part 8: Run / Debug Tests	694
Chapter 55: aRunning Tests and Components	695
Run a test or component	697
Prerequisites	697
Set the number of iterations for the test	697
Run an entire test or component	698
Run to a selected step or action (GUI testing only)	699
Run a single action, test, or a component from a selected step (GUI testing only)	699
Interrupt a run session	700
Run an API test from the command line	700
View the run results	701
Running GUI tests with a disconnected Remote Desktop Connection	702
Run a GUI test with a disconnected Remote Desktop Connection	702
Prerequisites	703
Log in to the remote computer running UFT	703
Configure the Remote Desktop Connection options in the remote computer	703
Configure the Windows Task Scheduler options for automation runs - optional	704
Start the test and disconnect from the Remote Desktop connection	704
UFT Runtime Engine	705
Test Batch Runner	707
Create and run a test batch	707
Open Test Batch Runner	707
Add batches or tests	708
Select the tests to be part of the test batch run	708
Run the test batch	708
Run the test batch via the command line	708
View the test batch run results	708
Using UFT for continuous integration	709
Optional steps	709
Log tracking	710
Manually configure log tracking settings	711

Open the relevant log configuration file and specify your preferences	711
Configure the settings in the Log Tracking pane so that UFT will use the same settings that you defined in the previous step	712
Results	712
Known Issues - Run Sessions	713
Chapter 56: Using Run Results	716
Checkpoint and Output Value results	722
Interpret run results	731
Set run result reporting options	732
View step details for each test step	732
Analyze errors in your test or component	733
Analyze checkpoint results (GUI tests and components only)	733
View the data source included with your test or component	734
View the call stack to isolate errors in the test flow	735
View the step properties capture for an API test step	735
View custom messages sent to the run results	735
Send the run results by email	736
Chapter 57: Running Tests with Virtualized Services and Networks	737
Virtualized services	738
Assigning data and performance models to a virtualized service	738
Network emulation	740
Use a virtualized service for a UFT test	741
Prerequisite - Deploy the Service Virtualization server	741
Add services from a virtualization project to your test in UFT	741
Add virtualized services from a server to the test in UFT	742
Undeploy a virtualized service	742
Update service details (optional)	743
Set the data and performance models for the virtualization project	743
Pause a deployed service for a test run	744
Put a service on standby	744
Use the virtualization project in your GUI test	744
Use the virtualization project in your API test	745
Run the test with a virtualized service	745
Run a test using an emulated network	745
Prerequisites	745
Enter your credentials for accessing the NV Test Manager	746
Start a network emulation session	746
Stop a network emulation	746
Optional - exclude specific IP addresses from a network emulation	747
Run the test using the network emulation	747
Chapter 58: Debugging Tests and Components	749
Modifying and watching the values of variables and properties of objects	750

Debug a test, component, function library, or user code file	751
Prerequisites	751
Slow your debugging session (GUI testing only)	751
Step into, out of, or over a specific step during a debug session	751
Start or pause your debugging session at a specific step or action	752
Use breakpoints in your document	752
Check the values of variables and expressions	753
Modify the values of variables or expressions during a run session	754
Manually run code commands during a debug session	754
View the current call stacks	755
View currently running threads	755
View the loaded modules associated with the run session	755
Debug a function - Exercise	755
Create a new action or function	755
Associate the function library with a test)	756
Add a call to the function in your test	756
Add breakpoints	756
Begin running the test or component	756
Check the value of the variables in the debug panes	756
Check the value of the variables at the next breakpoint	757
Modify the value of a variable using the Console pane	757
Repeat a command from the command history	757
Step Into, Out of, or Over a specific step - Exercise	758
Create the sample function library and test	758
Run the function library and use the commands	758
Debug an API user code file - Exercise	759
Create test steps	759
Set properties for the math steps	759
Create parameters for the Custom Code activity	760
Link the Custom Code activity to existing steps	760
Create events for the Custom Code activity	761
Run the test	761
Check the value of the variables at the first breakpoint	761
Add a variable to the Watch Pane	761
Check the value of the variables at the next breakpoint	762
Use breakpoints	763
Set a breakpoint	763
Enable or disable a breakpoint	763
Enable or disable all breakpoints	764
Remove a single breakpoint or all breakpoints	764
Navigate to a specific breakpoint	764
Known Issues - Debugging	765

Chapter 59: Running Tests with the Runtime Engine	767
Part 9: UFT Integration With HP ALM	768
Chapter 60: ALM Integration	769
Work with tests and components in ALM	771
Prerequisites	771
Connect to an ALM Project	771
Enable ALM to run tests or components	772
Enable full access to tests from ALM	772
Enable the Remote Agent	772
Install an external certificate for your ALM server	773
Create a template test	774
Set UFT Remote Agent Preferences	774
Disconnect from the ALM project	774
Resources and Dependencies for ALM assets	774
Relative paths for tests/resources saved in ALM	776
ALM template tests	777
Create a template GUI test	778
Data drive a test in ALM	778
Prerequisites	778
Import data into a test (API testing only)	779
Data drive the test steps (API testing only)	779
Create a data resource file in your ALM project	779
Specify a default data table resource for all new test configurations	779
Define your test configurations	780
Link your configurations to requirements to create requirements coverage - optional ..	781
Run your test configuration	781
Known Issues- Resources and Dependencies	783
Known Issues- General ALM integration	783
Known Issues- ALM Integration with GUI Testing	786
Known Issues- ALM integration with API Testing	787
Chapter 61: Running Tests from ALM	788
Running tests in Server-Side Execution	788
AUT environment parameters	789
Run a test using Server-Side Execution	790
Prerequisites	790
Create tests and save them in ALM	791
Create functional test sets in ALM	791
Set up AUT Parameters in ALM and link your test parameters to them in UFT - optional	791
Set up hosts in ALM for the UFT tests	792

Schedule the tests in ALM - optional	792
Run the tests from ALM	792
Test parameterization and test configurations	793
Test Combinations Generator	795
Use-Case Scenario: Use the Test Combinations Generator	796
Set up and run test configurations	800
Prerequisite	800
Create a test configuration	801
Enter static data configuration values	801
Map test parameters to the Excel file	801
View component parameter mapping details	802
Run the test with the selected configuration	803
Use the Test Combinations Generator to create test configurations	803
Prerequisites	803
Open the Test Combinations Generator	803
Set the value of your parameters	803
Automatically generate values for parameters	804
Add a custom data generator	804
Set values as Happy Path or Error Path	807
View the selected parameter combinations	807
Change the testing combination algorithm	808
Select the configurations to generate	808
Generate the test configurations	808
Known Issues - Running tests from ALM	808
Run results	808
ALM versioning	809
External authentication	809
Stopping a test in the middle of a run	809
Test Combinations Generator	810
Running API tests on Windows 2012 R2	810
Chapter 62: Version Control in ALM	811
Asset Comparison Tool and Asset Viewer	813
Use ALM version control	816
Check in the currently open asset	817
Check out the latest version of an asset	817
Cancel a check-out operation	817
View the version history	818
Work with the Asset Comparison Tool and Asset Viewer	818
Open the Asset Viewer	818
Open the Asset Comparison Tool	820
View a comparison of two asset versions (Asset Comparison Tool)	822
Drill down to compare or view a specific element	822

View the UFT location of an element	823
View the number of differences for a specific element	823
Known Issues - ALM Version Control	824
Chapter 63: HP Sprinter	826
Part 10: Business Process Testing in UFT	829
Chapter 64: Business Process Testing in UFT	830
Business Process Testing methodologies	831
Comparing BPT in UFT to BPT in ALM	834
Set up UFT for Business Process Testing	836
Prerequisites	836
Install and load the correct addins in UFT	837
Configure settings for your application's technology	838
Select a test creation methodology	839
Create and maintain business process tests and flows in UFT	839
Prerequisites	839
Create application areas for each area of your application	840
Create components	840
Add components to business process tests and flows	840
Add steps to your component	841
Group components and flows	841
Use parameters in your test	842
Iterate components and flows	842
Add a test configuration	842
Debug and run your test	842
View the run results	843
Business Process Testing in UFT - End-to-end scenario	843
Analyze your application	844
Prepare your test infrastructure	845
Create the business process test and add steps to the test	847
Enhance your test	851
Run your test	852
Chapter 65: Business Components and Application Areas	854
Converting manual components to UFT components	857
Create and manage GUI business components	857
Prerequisites	857
Update a component from an earlier QuickTest version	858
Create a new business component	858
Convert a manual component to an automated component	858
Convert the keyword GUI component to a scripted GUI component	859
Associate a different application area with your component	859

Delete a component	860
Manage application areas	860
Known Issues- Business Components and Application Areas	861
Chapter 66: Creating Business Process Test Steps	862
Create test steps in a business process test	863
Prerequisites	864
Create shared object repositories	864
Add test objects using Capture	864
Add object repositories to an application area	864
Manually add steps to your component	864
Add steps to your component by recording	865
Record a business process test	865
Prerequisite	865
Set recording options	866
Set default parameter behavior	866
Start the test record	866
Perform steps on your application	866
Add additional components to the test (optional)	867
Stop recording	867
Add test objects to a component with Capture	867
Prerequisites	867
Set Capture options	867
Capture the test objects in the application	868
Capture a selected area of the application	869
Open the object repository for editing	870
Export the local object repository	870
Chapter 67: Using Data in Business Process Testing	871
Linking parameters	872
Promoting parameters	876
Use data in a business process test	877
Design data	878
Create parameters and set default values	878
Use component parameters in component steps	878
Link parameters	878
Promote parameters	879
Add iterations for a component or flow	880
Set data values for the parameters for each iteration	880
Export component parameters to an Excel	880
Import parameter iteration values from an Excel	881
Using data with Business Process Testing in UFT - Known Issues	882
Chapter 68: Running Business Process Tests	883
Running iterations	884

Running iterations of component groups	885
Run conditions	888
Set run conditions	889
Prerequisites	889
Select the flow or component to set run conditions	889
Set On Failure settings	889
Set run conditions	890
Chapter 69: Business Process Testing with the BPT Packaged Apps Kit	892
Learning tests and flows	893
Detecting and resolving changes	894
Learn business process tests and flows	895
Detect and resolve changes using Change Detection Mode	901
Business Process Testing with the BPT Packaged Apps Kit - Known Issues	904
 Part 11: Appendix	905
Appendix A: UFT Terminology Quick Reference	906
Appendix B: GUI Checkpoints and Output Values Per Add-in	916
Supported Checkpoints	917
Supported Output Values	921
Appendix C: Frequently Asked Questions for GUI Testing	925
Programming in the Editor and function libraries	926
Can I store functions and subroutines in a function library?	926
How can I enter information during a run session?	926
I have a Microsoft Access database that contains data I would like to use in my test.	
How do I do this?	926
How do I customize the run results?	927
Working with dynamic content	927
How can I create and run tests or components on objects that change dynamically from viewing to viewing?	927
How can I check that an object or child object exists (or does not exist)?	927
How does UFT record on dynamically generated URLs and Web pages?	928
How does UFT handle tabs in browsers?	928
Advanced Web issues	929
How does UFT handle cookies?	929
Where can I find a Web page's cookie?	929
How does UFT handle session IDs?	929
How does UFT handle server redirections?	929
How does UFT handle meta tags?	930
Does UFT work with .asp and .jsp?	930
How does UFT support AJAX?	930
Does UFT work with COM?	930

Does UFT work with XML?	930
How can I access HTML tags directly?	931
How can I send keyboard key commands (such as shortcut commands) to objects that do not support the Type method?	931
Working with Windows applications	931
How can I record on nonstandard menus?	931
Can I copy and paste to and from the Clipboard during a run session?	932
Test and component maintenance	932
How do I maintain my test or component when my application changes?	932
Can I increase or decrease Active Screen information after I finish recording a test?	933
Testing localized applications	934
Improving GUI testing performance	934
How can I improve the working speed of UFT when working with GUI testing?	934
How can I decrease the disk space used by UFT for GUI tests and components?	936
Is there a recommended length for tests?	936
Send Us Feedback	937

Part 1: UFT Introduction

Chapter 1: 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.

Tests	A test is a series of calls to actions. Actions divide your test into logical units, such as: <ul style="list-style-type: none">• The main sections of a web site• Specific activities you perform in your applications Create tests that call multiple actions for modularity and efficiency.
Actions	Each action is a series of steps. Add steps to your actions and view them in the Keyword View or the Editor.
Test runs	In a run session , UFT performs each action in your test. View a report detailing each step, and which ones succeeded or failed.
Checkpoints	Enhance your test with checkpoints . Add a step to check the properties of specific objects in your application, such as: <ul style="list-style-type: none">• Whether a specific string is displayed• Whether a hyperlink goes to the correct URL
Function libraries	Enhance your test with function libraries . <ul style="list-style-type: none">• Create libraries of code you want to use multiple times in your tests.• Call the functions from your test as often as you need.

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 "[Running API Tests with GUI Tests](#)" on page 134.

For details about calling GUI tests from API tests, see "[Call external tests or actions](#)" on page 400.

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 769, 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.

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 829](#).

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 permissions for UFT

Required file system permissions

Read/write permissions	You must have read/write permissions to the following files and folders, as well as any sub-folders: <ul style="list-style-type: none">• The Temp folder• The folder containing UFT solutions, tests, or run results• The <Program Files>\Common Files\Mercury Interactive folder• The <Program Data>\HP folder (Windows 7 or Windows Server 2008 systems)• User Profile folders• The <Windows>\mercury.ini file• The following AppData folders: %userprofile%\AppData\Local\HP %appdata%\Hewlett-Packard\UFT %appdata%\HP\API Testing
Read permissions	You must have read permissions to the following folders: <ul style="list-style-type: none">• The Windows folder• The System folder

Required registry key permissions

Read/write permissions	All keys under: <ul style="list-style-type: none">• HKEY_CURRENT_USER\Software\Mercury Interactive or [HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Hewlett-Packard]• HKEY_CURRENT_USER\SOFTWARE\Hewlett Packard
Read and Query Value permissions	<ul style="list-style-type: none">• HKEY_LOCAL_MACHINE keys• HKEY_CLASSES_ROOT keys

Required permissions for ALM

Read/write permissions	<ul style="list-style-type: none">• ALM cache folder• <Program Data>\HP folder• UFT Add-in for ALM installation folder
Administrative permissions	For the first connection to ALM

Required permissions for BPT

Ensure that you have the required ALM permissions before working with business components and application areas.

Component steps

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).

Parameters in ALM or other testing tools

To work with parameters in ALM or in a testing tool, you must have all the parameter task permissions set in ALM.

Application areas

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.

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.

Known Issues- Multilingual Applications

Relevant for: GUI tests and components

This section includes general limitations about multilingual issues in UFT.

VMware

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.

Recording

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 tests

- 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
```

- 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).

Localized versions of UFT

The image displayed when selecting **View Sample Snapshot** in the UFT Asset Comparison tool is not localized

Additionally, some user interface items are not localized.

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>

Examples of interface items that are not localized include:

- 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

Known Issues - UFT Tools and Programs

Relevant for: GUI tests and components and API testing

UFT program tools

You cannot use some of the UFT tools when the UAC (User Account Control) option in is set to ON.

Workaround: Temporarily turn off the UAC option while using these tools, by doing the following:

For Windows Server 2008:	<ol style="list-style-type: none">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:	<ol style="list-style-type: none">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.x
and higher:**

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.

Chapter 2: UFT Licensing

UFT supports various types of licenses. You can install licenses using the License Wizard or from the command line.

Seat licenses

A machine-specific license based on a specific locking code per computer.

The key must be entered once only, and provides one installation per key.

A computer with multiple bootable partitions may generate a different locking code for each partition.

When obtaining a seat license key, you must use the locking code for the partition on which you want to use UFT.

Concurrent licenses

A license taken from a license server on a per-session basis. You must have an active network connection to install and check out commuter licenses.

Each time UFT starts, UFT tries to connect to the License Server for an available license.

Each key provides unlimited installations . The license server regulates the number of licenses in use at any given time.

Note: Install a special tool to track license usage (both for UFT and other products) across your network. This tool is available here: <https://hpln.hpe.com/contentoffering/hpe-usage-hub>

Commuter licenses

A license checked out for a period of time to use when you are not connected to the license server.

You or another user must have an active network connection to install and check out commuter licenses.

The license key is based on the machine identification, and is specific for the computer making the request.

A commuter license key needs to be entered only once, and provides one installation for a limited period of time.

After the computer license expires, UFT automatically returns to the previously used license type.

 See also:

- "Install licenses using the License Wizard" below
- "Install licenses from the command line" on page 51
- "UFT licensing FAQs" on page 53
- "Known issues with UFT licenses" on page 57

Install licenses using the License Wizard

The Functional Testing License Wizard enables you install, check out, or switch between license types, and requires administrator permissions.

Access the wizard from the **Start menu (HP Software > HP Unified Functional Testing > Tools > Functional Testing License Wizard)** or the file system (**C:\Program Files (x86)\HP\Unified Functional Testing\bin\HP.UFT.LicenseInstallationWizard**)

Exit the wizard when the installation is complete. Restart the LeanFT runtime engine to apply the new license.

 **Note:** See also: "Install licenses from the command line" on page 51

You can view current license information from the Help > About Unified Functional Testing screen by clicking the **License** button.

Additionally, UFT warns you if your license is about to expire. If you have multiple licenses, UFT displays the date of the license closest to expiration.

Install a Seat license (wizard)

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 **.dat** file.
 - Paste the license key in the edit field.
If you don't yet have a license key, expand the **How can I get a license key file** section.
3. Verify that the license key is valid, and click **Install**.

 **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 (wizard)

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 following format:
<license server address>:<port>
Default port = **5814**
The address format must identical the one used in the **Main** tab of the License Server Configuration pane.
For details, see the *AutoPass License Server User Guide*.
4. Click **Connect** to connect to the License Server.
5. (Optional) Define a redundant License Server.
If your primary License Server is unavailable, UFT will connect to the redundant License Server to obtain a license. For more details, see the *AutoPass License Server User Guide*.
Expand the **Add Redundant Server** link and enter the address for the redundant License Server.
6. From the product license drop-down list, select the appropriate license and click **Install**.

Check out and install a Commuter license

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

1. **Prerequisite:** Make sure you are connected to the network and can access the License Server.
Alternative, if you cannot access the License Server: "[Check out and install a Remote Commuter license](#)" on page 51

2. In the License Wizard start screen, select **Additional Options > Commuter License**.
3. In the Commuter License Installation screen, enter the License Server address in the following format:

<license server address>:<port>

Default port = **5814**

The address format must identical the one used in the **Main** tab of the License Server Configuration pane.

For details, see the *AutoPass License Server User Guide*.

4. Click **Connect** to connect to the License Server.
5. After the list of available licenses is displayed, ensure that **Available** is selected below the License Server address field.
6. From the list of available licenses, select the licenses you need.
7. In the **Check out licenses for (days)** field, enter the number of days for which you need the commuter license.
Maximum = 180 days
8. Click **Check Out**, and then **Next** to install the license.

Return a Commuter license

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.

1. **Prerequisite:** Make sure you are connected to the network and can access the License Server.
Alternative, if you cannot access the License Server: "[Check out and install a Remote Commuter license](#)" on the next page

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 in the following format:

<license server address>:<port>

Default port = **5814**

The address format must identical the one used in the **Main** tab of the License Server Configuration pane.

For details, see the *AutoPass License Server User Guide*.

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**, and then **Next**. The list of checked out licenses is cleared.

Check out and install a Remote Commuter license

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

1. In the License Wizard start screen, select **Additional Options > Remote Commuter license**.
2. In the Remote Commuter License Installation screen, ensure that **Generate Request File** is selected.
3. From the list of available licenses, select the license you need.
4. In the **Check out licenses for (days)** field, enter the number of days for which you need the commuter license.
Maximum = 180 days
5. Click **Generate Request File**.
Send the generated **.lcor** request file to a License Server administrator or to a user with access permissions to the License Server.
The other user must check out and send you a license key file.
6. Save the file, and then click **Choose File** to browse to the file you received.
7. Click **Install** to install the license.

Return a Remote Commuter license

1. In the License Wizard start screen, select **Additional Options > Remote Commuter license**.
2. In the Remote Commuter License Installation screen, ensure that **Generate Request File** is selected.
3. In the Generation screen, click **Generate and Save Check In Request**, and save the **.lcir** check in request file.
4. Click **Next** to uninstall the license.

The license wizard reports that the remote commuter license is uninstalled. UFT reverts to the previous license type as the active license.

Install licenses from the command line

Install and verify the statuses of seat or concurrent licenses directly from the command line.

Enter the following command, followed by a set of parameters described below.

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

Install a Seat license (command line)

Add one of the following to install a UFT seat license:

- seat "<license key string>"
- seat "<path to the license key file>"

For example:

Install a seat license key from a file saved locally:

```
"C:\Program Files (x86)\HP\UFT\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\UFT\bin\HP.UFT.LicenseInstall.exe" seat "9CDG C9MA  
H9P9 8HW3 UXB5 HWWF Y9JL KMPL B89H MZVU 6R4Q LHWE JHRP 3FQ3 CMRG HPMR MFVU A5K9  
MWEC EKW9 HKDU LWWP SRL7 QPJQ YM55 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
```

Note: If the license key contains a quotation mark character ("") in the license key string, add a backslash character (\) before the quotes.

Install a Concurrent license (command line)

Add the following to install a UFT concurrent license:

```
concurrent <license ID> <license version> <server name/address> [<redundant  
server name/address>] [/force]
```

Use the following format for the server or redundant server name/address:

<license server address>:<port>

Default port = **5814**

The address format must identical the one used in the **Main** tab of the License Server Configuration pane.

For details, see the *AutoPass License Server User Guide*.

For example:

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

/force parameter	The /force parameter saves the license installation information even if the current installation fails. In subsequent sessions, LeanFT will check the listed license server for the listed license.
Optional parameters	Optional parameters include: <ul style="list-style-type: none">• port• redundant server name/address• force

Modify server connection details

Add one of the following:

Modify the primary License Server address	config protocol.primary <http/https>
Modify the secondary License Server address	config protocol.second <http/https>

Verify available licenses

Add the following:

```
licenses <server name/address> [<redundant server name address>]
```

For example:

```
"C:\Program Files (x86)\HP\UFT\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 before UFT 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 must upgrade your licenses.

For details, see the topic on upgrading licenses in the *HP Unified Functional Testing Installation Guide*.

How do I get a 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*.

What if I have trouble regenerating licenses on the HP License Portal?

Contact your regional support for HP Licensing. You can find the appropriate person for yourself here:

<https://h30580.www3.hpe.com/poeticWeb/portalintegration/hppWelcome.htm?stepaction=contactLicenseSupport>

Which license should I install?

In UFT, you can install a number of different license types. The following table should help you to identify the one to install.

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. 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 traveling and will not have access to a license server?	Commuter
Are you already traveling 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 must 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 "[Install licenses from the command line](#)" on page 51.

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.hpe.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.

My demo license is expiring early. What can I do?

If you are having problems with the trial license period (30 days maximum), ensure the following:

- Ensure that you have full permissions to the **C:\ProgramData\Hewlett-Packard\UFT** folder and all its subfolders
- Ensure that you have not changed the system time. If you have moved the system time, the license mechanism can reduce the trial period based on the number of days that were back-dated.

Known issues with UFT licenses

Relevant for: GUI testing and API testing

Modifying the computer date	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.
NAT	The License Server does not support the use of Network Address Translation (NAT).
Demo licenses	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.
Changing types	You must have administrator permissions to change the license type from seat to concurrent or vice versa.

Chapter 3: UFT Document Management

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 (<code>TestUserCode.cs</code> 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 you are using GIT, use the GIT client version 2.5.2 or later.

If your documents and resources are saved in an ALM project or a version-controlled ALM project, see "[Version Control in ALM](#)" on page 811

Set up UFT to work with GIT

Before using GIT inside UFT you must configure GIT locally:

1. Install the GIT or SVN client on the computer running UFT.
2. Create your GIT workspace on your computer.
3. In UFT, create the test and save it in the folder configured to work with GIT or SVN.
4. When prompted, 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.

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.

After you set up the GIT workspace, the relevant icons are displayed in your testing documents in the Solution Explorer and you can use GIT commands for your documents directly from the Solution Explorer.

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 .

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 that opens, select a document with repository conflicts.
2. At the bottom of the dialog, choose one of the following options:

Use My Changes

Saves the changes you made to the document and overwrites the version saved in the repository.

Use Their Changes	Takes the version in the repository and overwrites your changes.
Merge	Merges both versions together. 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).
View in Explorer	Open the folder containing the document, where you can manually modify the conflicts.

- 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.

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 <code>C:\Current_Version</code> and <code>C:\Current_Version\Libraries</code> 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.</p> <p>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)	<p>Opens the document in read-only mode, enabling you to run it but not make any changes to it.</p> <p>You can:</p> <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	<p>Opens the document in read-write mode, enabling you to run and edit it, and save any changes you make to it.</p> <p>However, any changes to referenced and locked UFT files cannot be saved.</p> <p>Locked resource files are opened in read-only mode even if the referencing document is opened in read-write mode.</p>

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

UFT 12.53 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 UFT12.53.

- 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 UFT12.53.

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" on the next page](#).

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. <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

UI Elements	Description
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	<p>When upgrading a Service Test test from Service Test version 11.00 (via 11.10), UFT does not retain the Security settings.</p> <p>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.</p>
-----------------	--

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.
---	---

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
Action (for GUI tests)	<ul style="list-style-type: none">Must be unique in the test.Cannot be named Global, 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.Cannot begin or end with a space.Cannot contain the following characters: \ / : * ? " < > % ' ! { }
Action parameter (for tests)	<ul style="list-style-type: none">Case-sensitive.Must begin with a letterCannot contain spaces.Cannot contain the following characters: ! @ # \$ % ^ & * () + = [] \ { } ; : " , . / < >
Action (for API tests)	<ul style="list-style-type: none">Cannot begin or end with a spaceCannot exceed 1,023 charactersCannot contain the following characters: \ / : * ? " < > % ' ! { }
Application area (for components)	<ul style="list-style-type: none">Cannot exceed 220 characters (including the path).Cannot contain, begin, or end with spaces.Cannot contain the following characters: \ / : " ? < > * ! { } ' % ; ,Cannot contain multibyte punctuation symbols and other multibyte special characters, such as multibyte question marks, multibyte spaces, and multibyte brackets.
Component parameter	<ul style="list-style-type: none">Cannot contain brackets in the name (e.g. {Param1})

Item	Naming Convention
Checkpoint	<ul style="list-style-type: none"> Cannot begin or end with a space. Cannot contain " (double quotation mark). Cannot contain the following character combinations: <ul style="list-style-type: none"> := @@
Component (for components)	<ul style="list-style-type: none"> Cannot exceed 220 characters (including the path). Cannot contain, begin, or end with spaces. Cannot contain the following characters: \ / : ? < > * ! { } % ; Cannot contain multibyte punctuation symbols and other multibyte special characters, such as multibyte question marks, multibyte spaces, and multibyte brackets.
Data Table file (for tests and scripted components)	ALM: cannot contain the following characters: ! % * { } \ ' : " / < > ? ; ,
Data Table > Parameter name (column header) (for tests and scripted components)	<ul style="list-style-type: none"> Must be unique in the sheet. Must begin with a letter or underscore. Can only contain the following: <ul style="list-style-type: none"> Letters (excluding special characters, machine-dependent characters, or platform-dependent characters) Numbers Periods Underscores
Environment variable (parameter)	<ul style="list-style-type: none"> Must begin with a letter. Can only contain the following: <ul style="list-style-type: none"> Letters Numbers Underscores
Environment variables file	File system: Cannot contain the following characters: ! % * { } \ ' : " / < > ? ; ALM: Cannot contain the following characters: ! % * { } \ ' : " / < > ? ; ,

Item	Naming Convention
Function library file	<p>File system: Cannot contain the following characters: ! % * { } \ ' : " / < > ? ;</p> <p>ALM:</p> <ul style="list-style-type: none"> Cannot contain the following characters: ! % * { } \ ' : " / < > ? ;, Cannot contain non-English characters
Function / Function argument	<ul style="list-style-type: none"> Cannot contain non-English letters or characters. Function names cannot be the same as a VBScript registered keyword. Must begin with a letter. Cannot contain spaces or any of the following characters: ! @ # \$ % ^ & * () + = [] \ { } ; ' : " , / < > ? ,
Object repository file	<p>File system: Cannot contain the following characters: ! % * { } \ ' : " / < > ? ;</p> <p>ALM: Cannot contain the following characters: ! % * { } \ ' : " / < > ? ;,</p>
Output value	<ul style="list-style-type: none"> Cannot begin or end with a space. Cannot contain " (double quotation mark). Cannot contain the following character combinations: <ul style="list-style-type: none"> := @@
ALM file or folder name	<ul style="list-style-type: none"> Cannot exceed 90 characters (including the path). Cannot begin or end with a space. Cannot contain the following characters: \ : * ? " < > % ' ; Cannot contain multibyte punctuation symbols and other multibyte special characters, such as multibyte question marks, multibyte spaces, and multibyte brackets.
Recovery Scenario	<p>File system: Cannot contain the following characters: ! % * { } \ ' : " / < > ? ;</p> <p>ALM: Cannot contain the following characters: ! % * { } \ ' : " / < > ? ;,</p>

Item	Naming Convention
Test name (for tests)	<ul style="list-style-type: none"> Cannot exceed 220 characters (including the path). 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. Cannot begin or end with a space. Cannot contain the following characters: \ / : * ? " < > % ' ; Cannot contain multibyte punctuation symbols and other multibyte special characters, such as multibyte question marks, multibyte spaces, and multibyte brackets.
Test resources (when saving a test with resources)	The path name (resource name and file path together) cannot exceed 256 characters.
Test object class (extensibility only)	<ul style="list-style-type: none"> Cannot contain non-English letters or characters. Cannot contain spaces or any of the following characters: ! @ # \$ % ^ & * () + = [] \ { } ; ' : " " , / < > ?
Test object name	<ul style="list-style-type: none"> must be unique within the same class and hierarchy in the object repository Cannot begin or end with a space. Cannot contain " (double quotation mark). Cannot contain the following character combinations: <ul style="list-style-type: none"> := @@
Test object identification properties	<ul style="list-style-type: none"> Cannot contain non-English letters or characters. Cannot contain spaces or any of the following characters: ! @ # \$ % ^ & * () + = [] \ { } ; ' : " " , / < > ?
Test object method / Test object method argument	<ul style="list-style-type: none"> Cannot contain non-English letters or characters. Cannot contain spaces or any of the following characters: ! @ # \$ % ^ & * () + = [] \ { } ; ' : " " , / < > ?

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 Scripts](#)" on page 605.

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.

Renaming UFT documents in ALM

- 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.

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">1. 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.2. 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

1. 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.
2. In the Keyword View, click a step that you want to change. The window or page is displayed in the Active Screen pane.
3. Select **Tools > Change Active Screen**. The UFT window is hidden and the mouse pointer becomes a pointing hand.
4. Click the window or page displayed in your application.
5. 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.

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 contains 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	9.99999999999999E307
Smallest positive number	1E-307
Largest negative number	-1E-307
Smallest negative number	-9.99999999999999E307
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)

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.

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 81
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. For details: <ul style="list-style-type: none">• ""The Editor" on page 533• "Programming Tests" on page 542• "Open a window for writing custom code" on page 629
BPT in UFT	Displays and enables you to edit business process tests and flows, including grouping components and flows, and modifying run order.
Application area	Displays and enables you to edit the application area settings and resource associations. For details: " Manage application areas " on page 860

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 [386](#).

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.

UI Element	Description
 <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. 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.	

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	<p>Do one of the following:</p> <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.
------------------	--

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.



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

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.



Example:

- 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 a function into an action, component, or function library, a comment and a call to that function are added.

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.

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 98.

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.

Node	Options
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 Monitor	Options for activating and setting preferences for tracking system counters during a run session.

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 *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
```

Part 4: GUI Test Design

Chapter 21: GUI Test Creation Overview

Relevant for: GUI tests and components

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	<p>Keyword-driven testing enables you to design tests at a business level rather than at the object level.</p> <p>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.</p>
Easier to read	<p>Technical operations, such as a synchronization statement that waits for client-server communications to finish, are incorporated into higher level keywords.</p> <p>This makes tests easier to read and easier for less technical application testers to maintain when the application changes.</p>
Separated resources and tests	<p>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.</p>

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 UFTGUI 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 in GUI Testing](#)" on [page 262](#)



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 330](#).

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 in GUI Testing](#)" on [page 322](#).

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 597.

You can also manually enter standard VBScript statements, as well as statements using UFT test objects and operations. For details, see "[Programming Tests](#)" on page 542.

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 "[Update test object descriptions, checkpoints, or output values, or Active Screen captures](#)" on page 155.

Create a keyword-driven GUI test

Relevant for: GUI tests only

This task describes how to create a test using the keyword-driven methodology.

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 "[Run / Debug Tests](#)" on page 694 "Run / Debug Tests" on page 694 and "[Maintaining Tests or Components](#)" on page 147.

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	<p>Determine the functionality that you want to test.</p> <p>To do this, consider the various activities that customers perform in your application to accomplish specific tasks.</p> <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	<p>Decide how to divide these processes into smaller units that will be represented by your test's actions.</p> <p>Each action should emulate an activity that a customer might perform when using your application.</p> <p>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.</p>

Prepare the testing infrastructure

Prepare the infrastructure needed for your test.

Build the set of resources for your test

This includes:

Shared object repositories	<p>Contain test objects, which are representations of the objects in your application.</p> <p>For details, see "Test Objects in Object Repositories" on page 206.</p>
Function libraries	<p>Contain functions that enhance UFT functionality.</p> <p>For details, see "User-Defined Functions" on page 575</p>
Recovery scenarios	<p>Instruct UFT to recover from unexpected events and errors that occur in your testing environment during a run session.</p> <p>For details, see "Recovery Scenarios" on page 159.</p>
Additional optional files	<p>Includes data table files and environment variable files.</p> <p>For details, see "Parameterizing Object Values" on page 330.</p>

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 98](#)
- ["Document Settings" on page 100](#)
- ["Recovery Scenarios" on page 159](#)
- ["UFT Automation Scripts" on page 605](#)

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 121](#).

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 206](#)
- ["User-Defined Functions" on page 575](#)
- ["Recovery Scenarios" on page 159](#)

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 126.

Enhance your test

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

For details, see:

- "[Checkpoints in GUI Testing](#)" on page 262
- "[Output Values in GUI Testing](#)" on page 322
- "[Parameterizing Object Values](#)" on page 330
- "[User-Defined Functions](#)" on page 575
- "[Generated Programming Operations](#)" on page 597

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	"4ff405198a68b24867227da98b21e547cf0c2f47d31"	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
	The browser invokes the Welcome: Mercury Tours Web site.
	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: Record GUI Tests and Components

Relevant for: GUI tests and components

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" on the next page](#)
- ["Insight recording" on page 115](#)
- ["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

- Activate
- Click
- DblClick
- Drag
- Drop
- Maximize
- Minimize
- Restore
- Type

- 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 236.

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:

- "Record a GUI test or component" below
- "Known Issues When Recording" on page 119

Record a GUI test or component

Relevant for: GUI tests only

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

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**).

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.

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.

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.

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 in the Capture Toolbar, 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.

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)

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 <code>InsightObject</code> '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.

Limited access to application objects

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.

Changes in the application during a recording session

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.

Chapter 23: 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:

["Naming conventions" on page 70](#)

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.

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.

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 Call Properties dialog box	<ul style="list-style-type: none">• Run Tab (Action Call Properties Dialog Box)• Parameter Values Tab (Action Call 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.

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 24: Keyword View

Relevant for: GUI tests and components

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

- 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.
- When a **Value** cell is selected, press **CTRL+F11** to open the Value Configuration Options Dialog Box.

Working in the Keyword View does not require any programming knowledge. The programming required to actually perform each test step is done automatically behind the scenes by UFT.

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

Add steps, comments, and programming to your test as follows:

A standard step	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 the next page .
A checkpoint step	A step to test the state of an object in your application at a specific point in the application/test flow. For details, see " Standard steps in the Keyword View " on the next page .
An output value step	A step to produce a value from an object which can be used later in the test. For details, see " Output Values in GUI Testing " on page 322.
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 129.

Steps using programming logic	<p>You can use programming logic to perform a number of tasks:</p> <ul style="list-style-type: none">• send information to the run results• put a comment line in your test• synchronize your test with your application• measure a transaction in your test• insert conditional or loop statements <p>For details, see "Generated Programming Operations" on page 597, "Conditional and loop statements" on page 130.</p>
--------------------------------------	--

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.

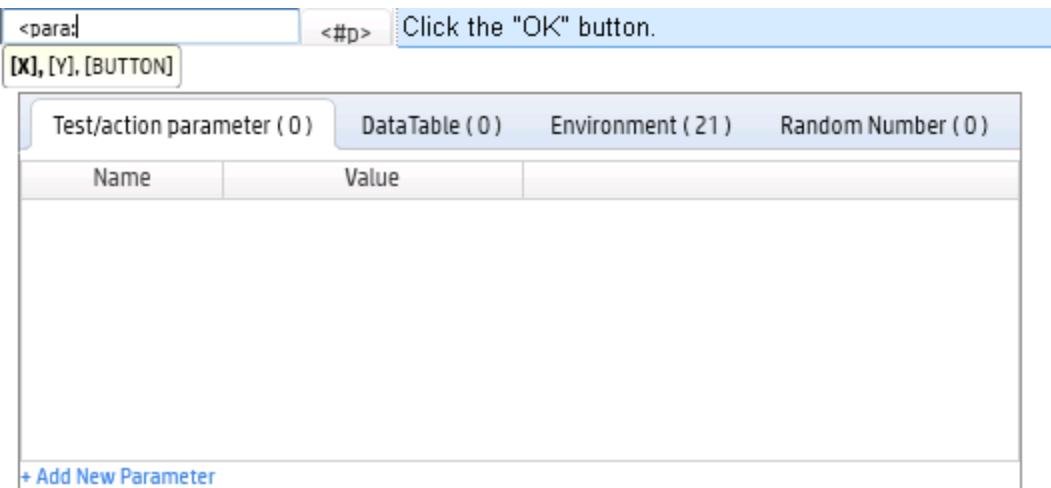
In tests, after first defining a regular statement (such as **x=10**), it can be edited only in the Editor.

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

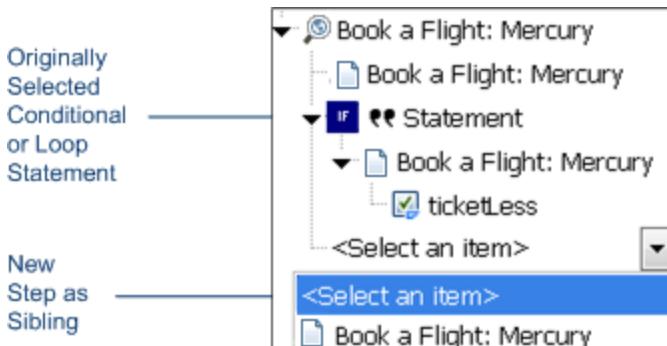
You can encode passwords for use in method arguments and data table cells. For details on how to encode passwords, see:

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 127.

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 Elself...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.

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.

Find a Flight: Mercury - Find a Flight: Mercury	GetROProperty	"item count"	Retrieve the current value of the "item count" property for the "toDay" li-
toDay		For i = 0 to ItemsCount - 1	Repeat the following step(s) one or more times according to the defined
Statement	GetItem	"i"	Retrieve the value of the item with index "i" in the "toDay" list.
toDay	Select	"ItemName"	Select the item "ItemName" from the "toDay" list.

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

Known Issues - Keyword View

Relevant for: **GUI actions and components**

Object properties	<p>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.</p> <p>This may affect UFT's response time when you open and select from the various drop-down lists in the step.</p> <p>If this occurs, use the Editor when working with the Object property.</p>
Calls to other actions	<p>If you insert a call to another action, you cannot expand the action node to view the steps in the called action.</p> <p>Instead, double-click the called action node to open the action steps in another tab.</p>

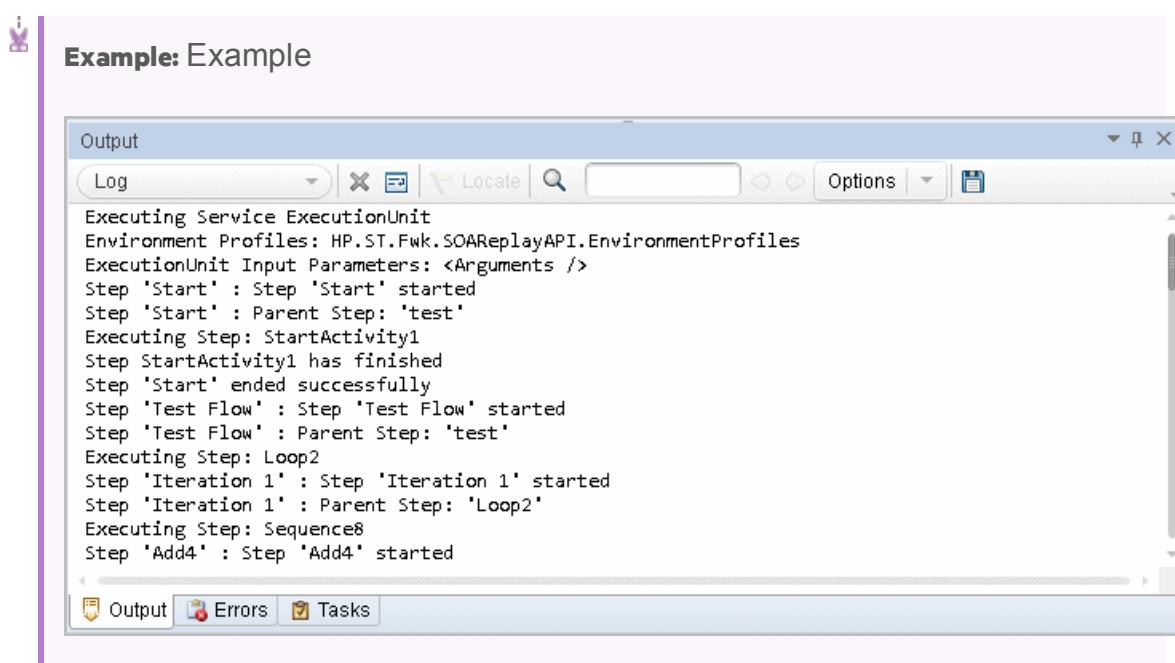
Chapter 25: Running API Tests with GUI Tests

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.



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 137](#).

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.

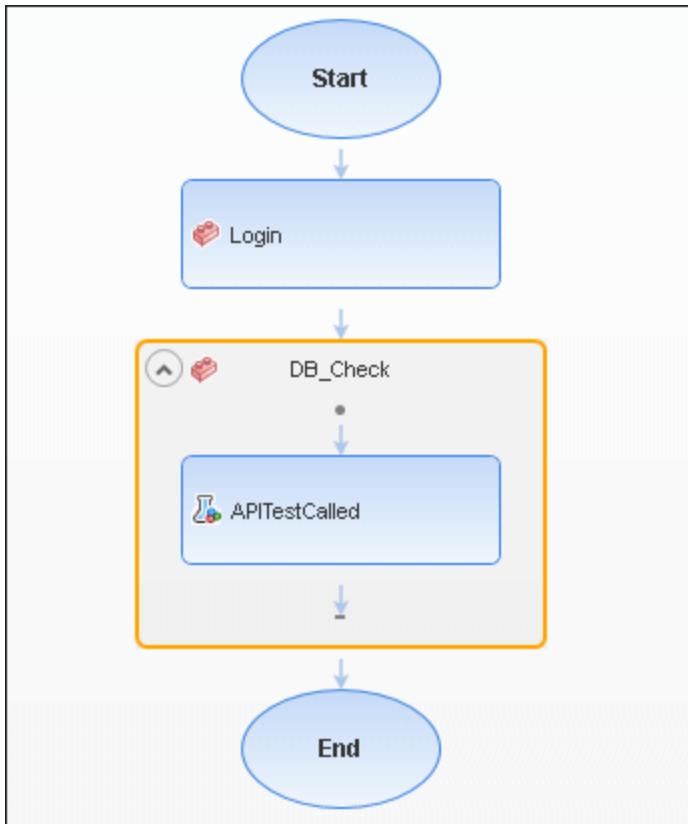
UFT inserts a step that calls the API test or action.

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

For 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.

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:

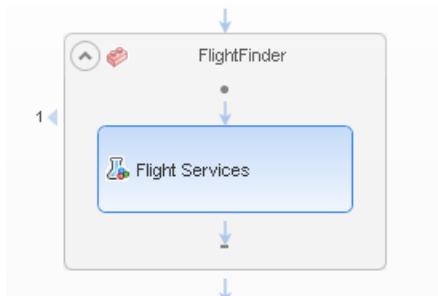
Finding user login credentials in the database	Login operation
Searching for a list of all available flights and displaying the flight list	FindFlights operation
Creating a flight order	CreateFlight operation
Confirming a flight booking	ConfirmBookFlight operation
Updating a flight order	UpdateFlightOrder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

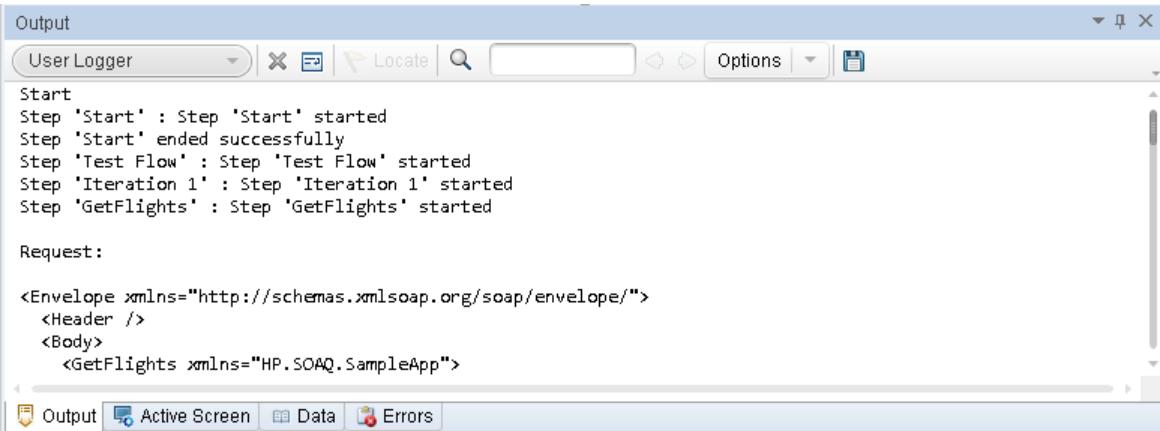
Running the test

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
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.SOAP.SampleApp">
```

At the bottom of the pane, there are four tabs: 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.

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.

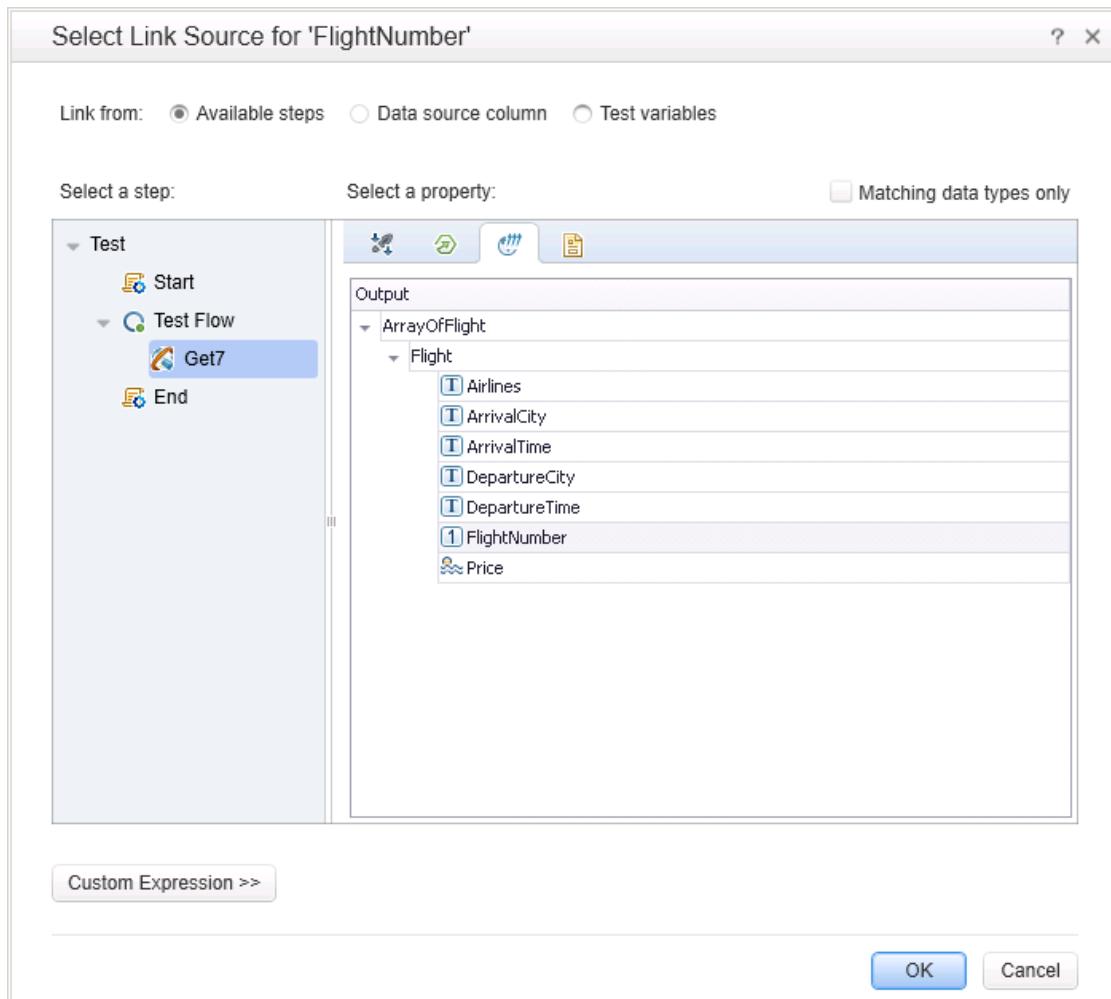
In the API test, click on the **End** step.

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.

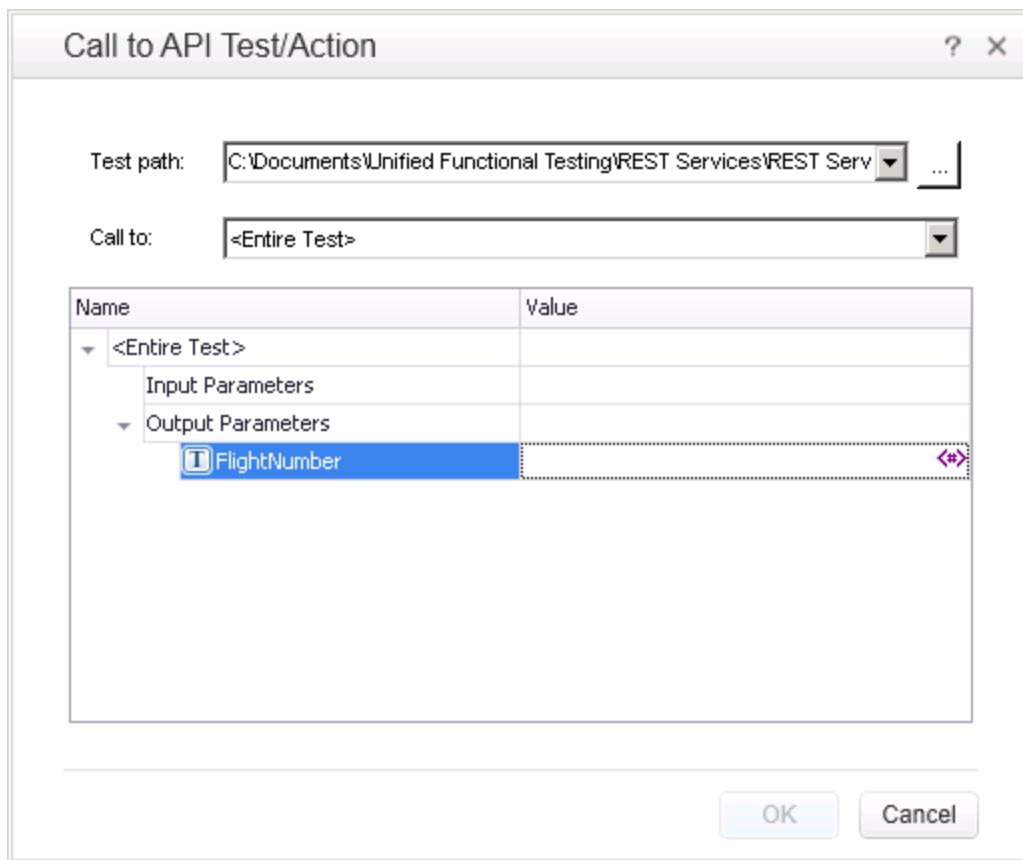
Click the **Link to data source** button  in the Test Input/Output Parameters tab, and then select the **Get** steps from the **Available steps** option.



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.

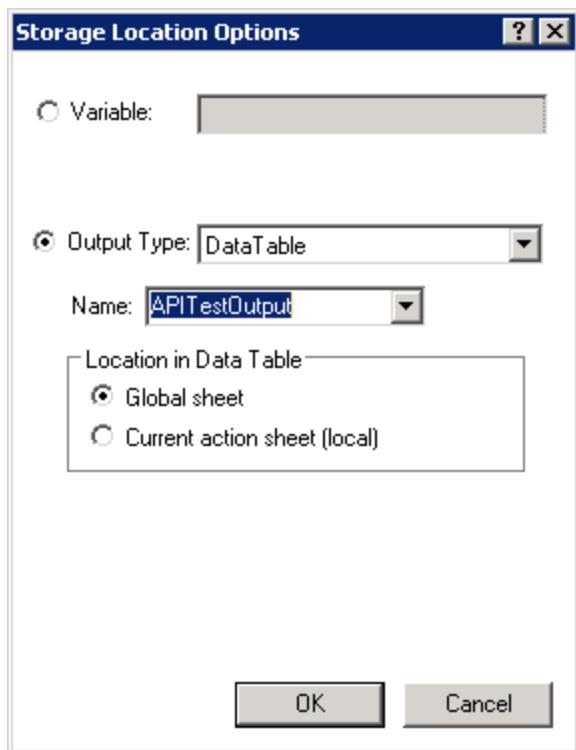
Then you can see the output parameter in the **Call to API Test/Action** dialog box after you navigate to the API test.



4. From this dialog box, select where in the GUI test to store the output parameter data.

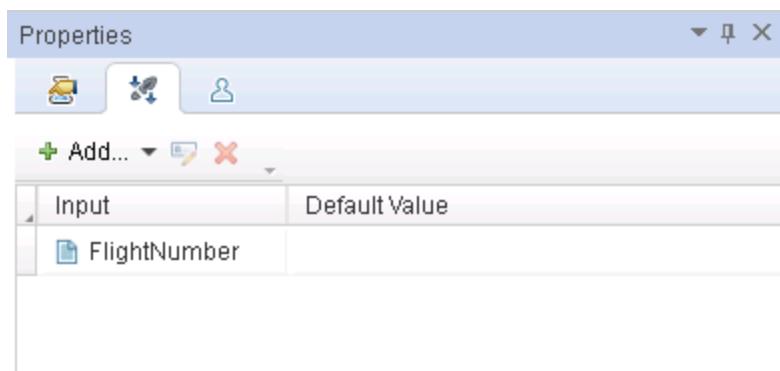
In the **Value** column for the **FlightNumber** parameter, click the **Configure** icon and open the Storage Location Options dialog box.

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.

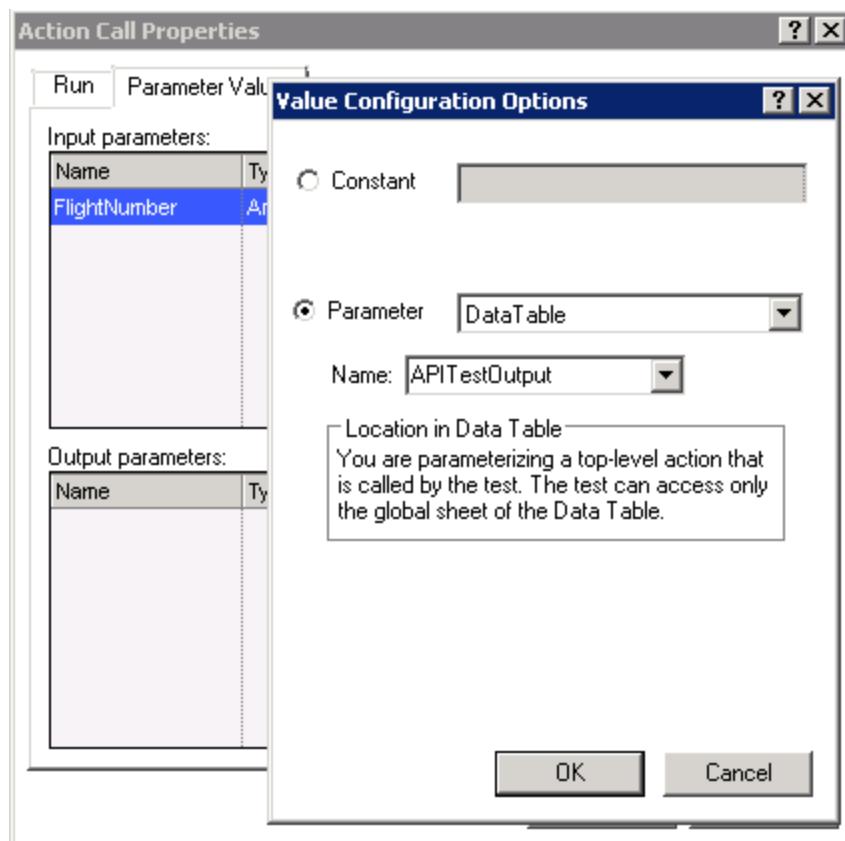
In the **Parameters** tab of the Properties pane, add an input parameter for the action called **FlightNumber** of type **Number**.



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	RunAPITest	"REST Services",DataTable("APITestOutput", dtGlobalSheet)	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.
ordersDataGridView	SelectCell		Select the cell in row "1", column "1" in th
SELECT ORDER	Click		Click the "SELECT ORDER" button.

Your test steps now look like this:

Item	Operation	Value	Documentation
Search	RunAPITest	"REST Services",DataTable("APITestOutput", dtGlobalSheet)	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.
ordersDataGridView	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
("ordersDataGridView").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	
	APITestOutput	B
1	1023	
2		
3		
4		
5		
6		
7		
8		
9		

Chapter 26: Maintaining Tests or Components

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. As a result, UFT enables you to maintain and update these test and components.

You can encounter many types of errors.

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.



Example: 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 153.

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

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 do you use Maintenance Mode?

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	<p>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.</p> <p>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.</p> <p>You can also choose to add a comment to your test or component before the failed step.</p>
Object is missing from the object repository	<p>The Maintenance Run Wizard helps you add the missing object to the repository.</p> <p>You can also choose to add a comment to your test or component before the failed step.</p>
Object exists but can only be identified through Smart Identification	<p>Identifying objects using Smart Identification may cause tests or components to run slower.</p> <p>The Maintenance Run Wizard helps you modify the description of the object, so that Smart Identification is not needed.</p> <p>For more details, see "Smart identification" on page 195.</p>

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 211.

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 .

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

Merge changes to your shared object repository

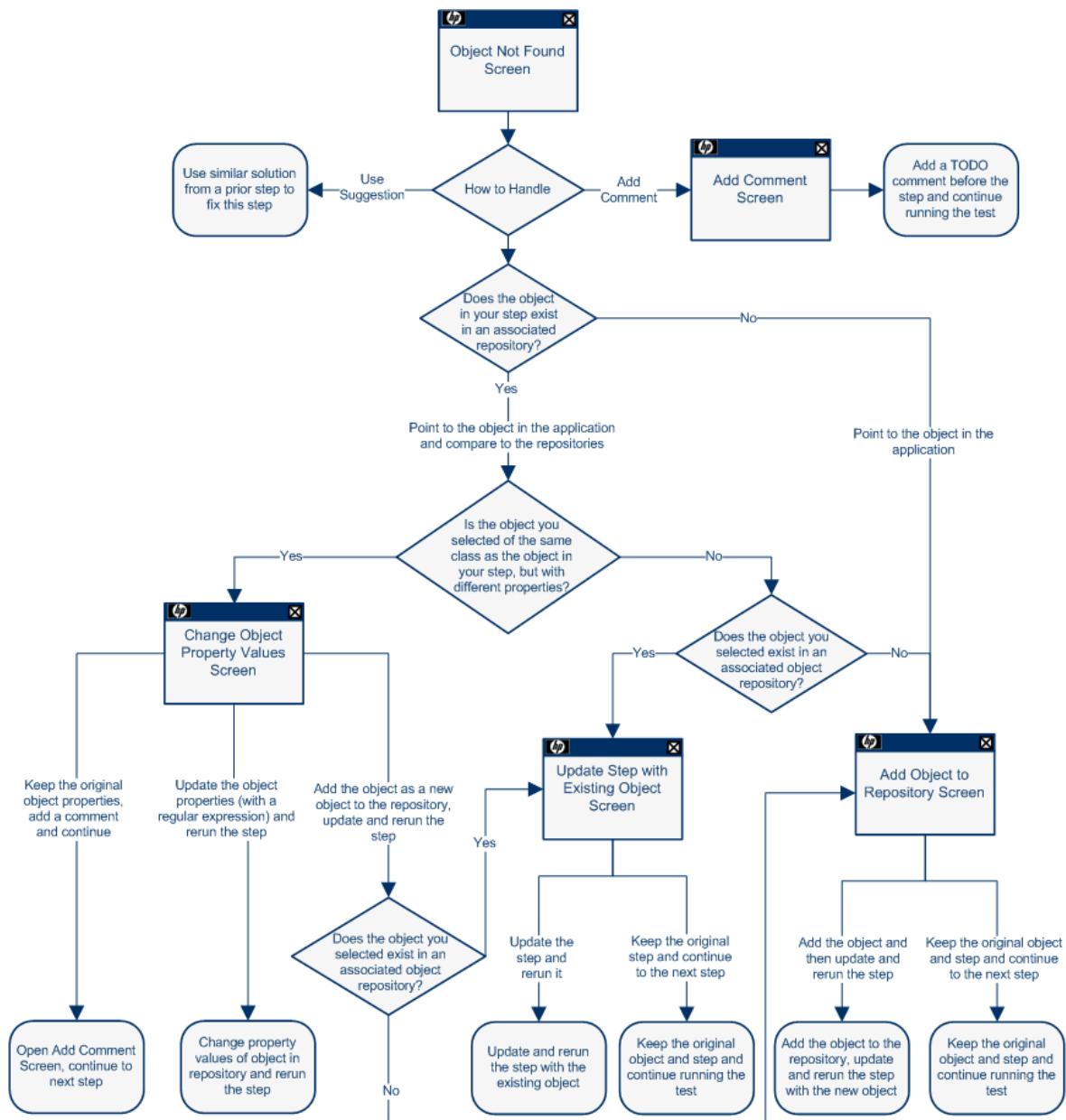
After using the Maintenance Run Wizard, you may want to merge the objects from the local repository back to a shared object repository.

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

For details, see "[Update a shared object repository from a local object repository](#)" on page 234.

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, you can use Update run mode to change the set of properties:

Objects identified using Smart Identification	<p>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.</p> <p>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.</p> <p>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.</p> <p>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).</p> <p>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.</p>
--	--

Parameters or Regular Expressions	<p>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.</p> <p>This occurs even if the values were parameterized or defined as regular expressions.</p> <p>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:</p> <ul style="list-style-type: none"> • If the previous value and the current value match UFT keeps the property's previous parameterized or regular expression value. 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. • If the previous value and the current value do not match but the object is found using Smart Identification, UFT updates the property value to the new, constant property value. 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>In this case, any parameterization or use of regular expressions is removed from the test object description.</p>
Property Case-Sensitivity	<p>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.</p> <p>In those cases, when you use Update Run to update test object descriptions, UFT also updates any case-sensitivity settings that may have changed.</p>

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.

Run in Update Run Mode

1. Specify the settings for the update run process.
2. In the Run dialog box, 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.

When a test runs in Update Run Mode, it does not update parameterized values, such as Data pane data and environment variables or property values of existing object descriptions in the object repository. To fix the object property values to match your application, use **Maintenance Run Mode**.

Export and merge changes

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 "[Update a shared object repository from a local object repository](#)" on [page 234](#).

Analyze the 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.

When the update run ends, the run results show:

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

Known Issues in Maintenance and Update Run Modes

Relevant for: GUI tests and components

Maintenance Mode

Using programmatic descriptions	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.
Objects without mandatory or assistive properties	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.
Updating a step by pointing to a different object class	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.
Updating XPath or CSS properties	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.

Objects used in function library	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.
---	---

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 27: Recovery Scenarios

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 Recovery pane of the Additional Settings tab in the application area.

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.



Example:

- 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 *UFT Object Model Reference for GUI Testing*.

Manage recovery scenarios

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.

Create a new recovery scenario operation

1. In the Recovery Scenario Manager Dialog Box, click the **New Scenario** button  . The Recovery Scenario Wizard opens.
2. Follow the on-screen instructions.

Associate a recovery scenario in the Test Settings

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 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 a component/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.

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

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.

Known Issues- Recovery Scenarios

Relevant for: GUI tests and components

- 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 28: Using Performance Testing and Business Service Management Products with UFT GUI Tests

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.

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 *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 168.

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.
- 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.

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.



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 170](#)".

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 *UFT Object Model Reference for GUI Testing*.
- Enter Start Transaction and End Transaction steps using UFT.



Example: Part of a sample test with a transaction is shown below, as it is displayed in the Keyword View:

Start transaction	<table border="1"> <tbody> <tr> <td>Services</td><td>StartTransaction</td><td>"ReserveSeat"</td><td>Start the "ReserveSeat" transaction.</td></tr> <tr> <td>Find a Flight: Mercury</td><td>Select</td><td>"London"</td><td>Select the "London" item in the "fromPort" list.</td></tr> <tr> <td>fromPort</td><td>Select</td><td>"Frankfurt"</td><td>Select the "Frankfurt" item in the "toPort" list.</td></tr> <tr> <td>toPort</td><td>Select</td><td>"12"</td><td>Select the "12" item in the "toDay" list.</td></tr> <tr> <td>toDay</td><td>Select</td><td>"Business"</td><td>Select radio button "Business" in the "servClass" radio button group.</td></tr> <tr> <td>servClass</td><td>Select</td><td>"Blue Skies Airlines"</td><td>Select the "Blue Skies Airlines" item in the "airline" list.</td></tr> <tr> <td>airline</td><td>Click</td><td>65,12</td><td>Click the "findFlights" image.</td></tr> <tr> <td>findFlights</td><td></td><td></td><td></td></tr> <tr> <td>Select a Flight: Mercury...</td><td>Select</td><td>"Blue Skies Airlines"</td><td>Select radio button "Blue Skies Airlines" in the "outFlight" radio button group.</td></tr> <tr> <td>outFlight</td><td>Select</td><td>"Blue Skies Airlines"</td><td>Select radio button "Blue Skies Airlines" in the "inFlight" radio button group.</td></tr> <tr> <td>inFlight</td><td>Click</td><td>46,8</td><td>Click the "reserveFlights" image.</td></tr> <tr> <td>reserveFlights</td><td></td><td></td><td></td></tr> <tr> <td>Services</td><td>EndTransaction</td><td>"ReserveSeat"</td><td>End the "ReserveSeat" transaction.</td></tr> <tr> <td>Book a Flight: Mercury_2</td><td></td><td></td><td></td></tr> </tbody> </table>	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 a Flight: Mercury...	Select	"Blue Skies Airlines"	Select radio button "Blue Skies Airlines" in the "outFlight" 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				Services	EndTransaction	"ReserveSeat"	End the "ReserveSeat" transaction.	Book a Flight: Mercury_2			
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 a Flight: Mercury...	Select	"Blue Skies Airlines"	Select radio button "Blue Skies Airlines" in the "outFlight" 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																																																									
Services	EndTransaction	"ReserveSeat"	End the "ReserveSeat" transaction.																																																						
Book a Flight: Mercury_2																																																									
End transaction																																																									

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"
```

Insert and run GUI tests in Performance Center and LoadRunner

Relevant for: GUI tests only

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.

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.

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.

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.

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.

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.

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.

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.

Part 5: Test Objects / Checkpoints / Output Values

Chapter 29: The Test Object Model

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.

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 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 206](#).

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 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.

In the example above, the action is displayed in the Editor like this:

```
Browser("Search Results: Search").Page("Search Results: Search").WebButton  
("Search").Click
```

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.

Test object descriptions

Relevant for: GUI tests and components

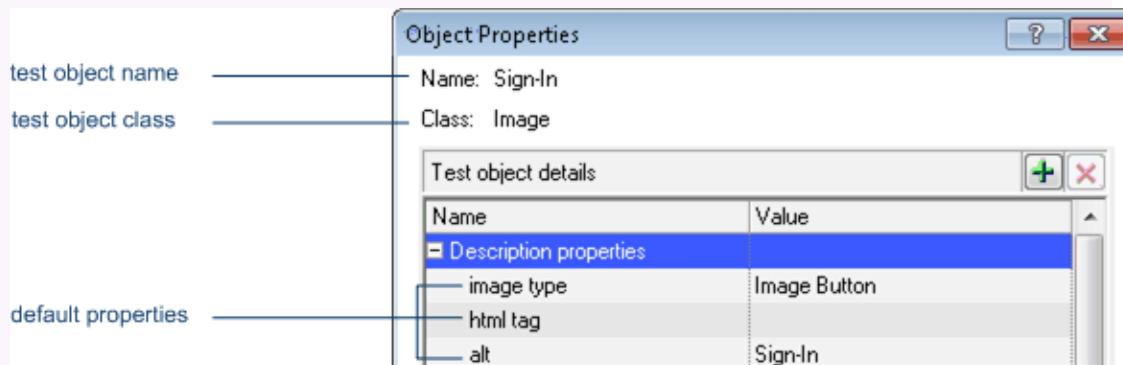
For each test object class, UFT learns a set of identification properties when it learns an object, and selects a sub-set of these properties to serve as a unique object description. UFT then uses this description to identify the object when it runs a test or component.

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 188](#).



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.

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>Identification properties 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>Native properties 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>

UFT Test Objects	Run-time Objects From Your Application
<p>A test object operation is a method or property that UFT can perform on an object from a particular test object class.</p> <p>Example: UFT can perform the Click method on a WebButton test object.</p>	<p>Native operations 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.

You can view and modify properties and operations for test objects and run-time objects in a number of ways:

Test objects	<ul style="list-style-type: none"> Retrieve or modify identification property values manually while designing your test or component, or use SetTOProperty statements during a run session. For details, see "Test Objects in Object Repositories" on page 206, "Retrieving and setting identification property values" on page 555, and the SetTOProperty method in the Common Methods and Properties section of the <i>UFT Object Model Reference for GUI Testing</i>. Use regular expressions in function libraries to identify property values based on conditions or patterns you define. For details, see "Regular expressions" on page 357. 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 330. 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 211. View the current identification property values of any visible object using the Properties tab of the Object Spy. For details, see "Use the Object Spy" on page 184. 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 "Use the Object Spy" on page 184.
---------------------	---

Run-time objects	<ul style="list-style-type: none">• View the syntax of the native operations of any visible object using the Operations tab of the Object Spy. For details, see "Use the Object Spy" on page 184.• Retrieve native property values from the run-time object during the run session by adding GetROProperty statements. For details, see "Retrieving and setting identification property values" on page 555.• If the available test object operations and identification properties do not provide the functionality you need, 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 556 and the Object property in the Common Methods and Properties section of the <i>UFT Object Model Reference for GUI Testing</i>.• 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.
-------------------------	---

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 page 174), 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

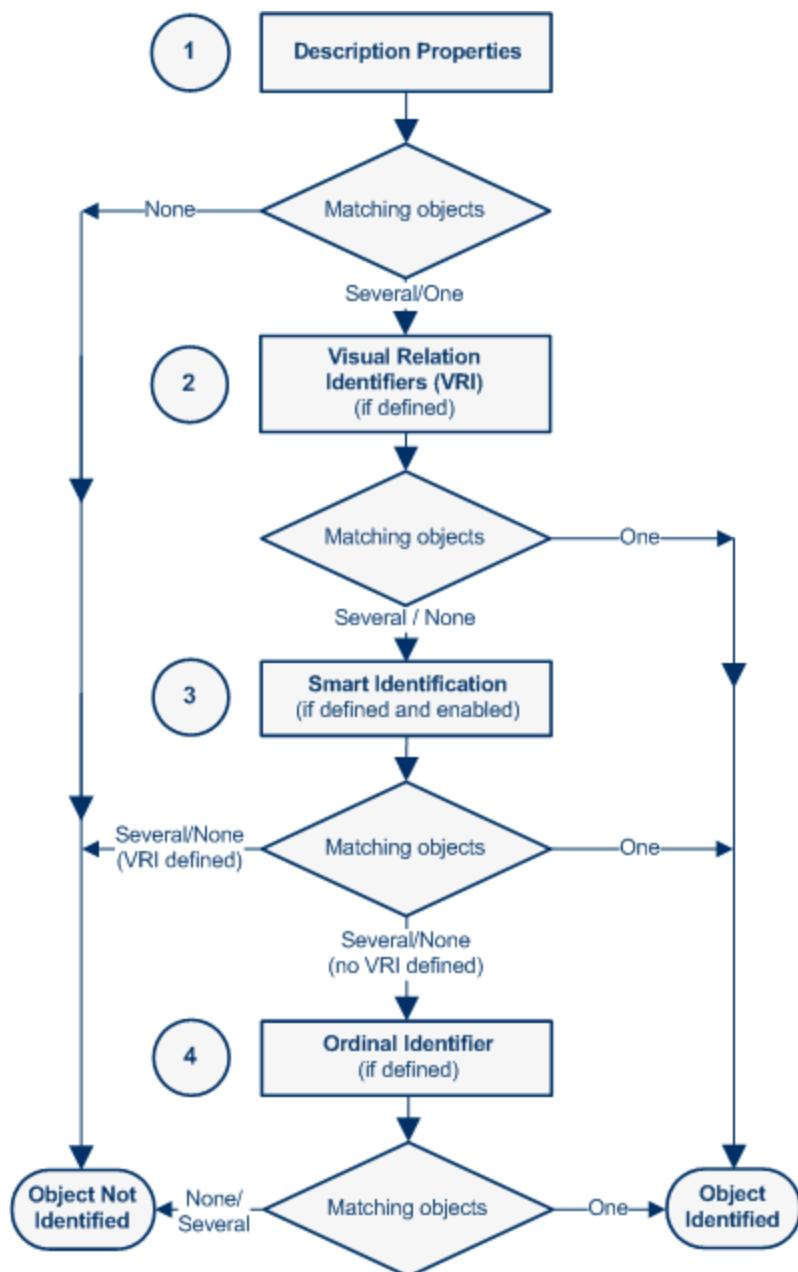
Identification. For details on Smart Identification, see ["Configuring Object Identification" on page 188](#).

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.

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).

Use the Object Spy

Relevant for: **GUI tests and components**

Note: If you are working with Web applications running in Safari on a remote Mac computer, see "[Use the Remote Object Spy](#)" on the next page.

Prerequisites

Open your application to the page containing the object on which you want to spy.

Use the pointing hand to select the application object

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 .

Note: The Object Spy does not support Insight test objects.

Add objects to the object repository

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.

Use the Remote Object Spy

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.

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).

Prerequisites

1. **In UFT:** Make sure that UFT is connected to a Remote Mac computer, using the Remote Connection button  in UFT's toolbar.
2. **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.

Open the Remote Object Spy dialog box

1. In UFT, make sure that a GUI test or action is in focus in the document pane or selected in the Solution Explorer.
2. In the toolbar, click the down arrow near the **Object Spy** button , and select the **Remote Object Spy**  button.

Use the mouse to select the application object

- 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.

View object details or add it to your object repository

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.

Chapter 30: Configuring Object Identification

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.

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 190. 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 192.

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.

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.

However, during a 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 192](#).

Index identifiers

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")
```

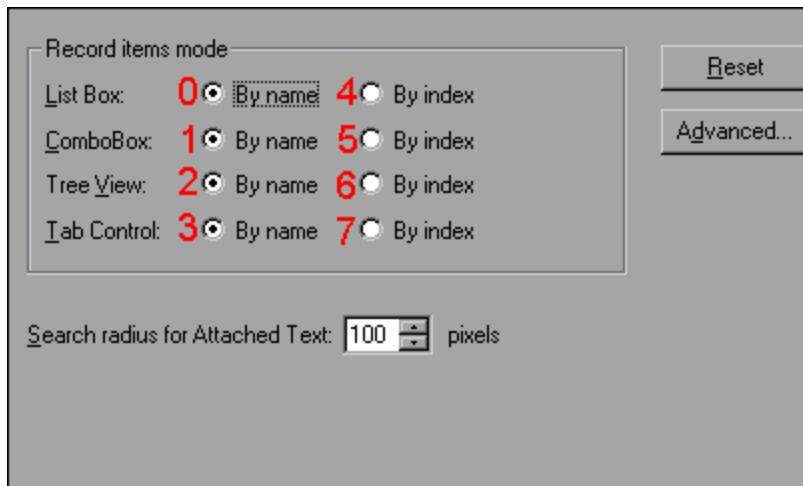
In contrast, the following statement refers to the second item in the list, as that is the first object of any type (WebElement) with the name **UserName**:

```
WebElement("Name:=UserName", "Index:=0")
```

Location identifiers

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.

Creation time identifiers

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.



Example: 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 am browser, **CreationTime = 1** to the 9:03 am browser, and **CreationTime = 2** to the 9:06 am 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 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.

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.

For example, 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.

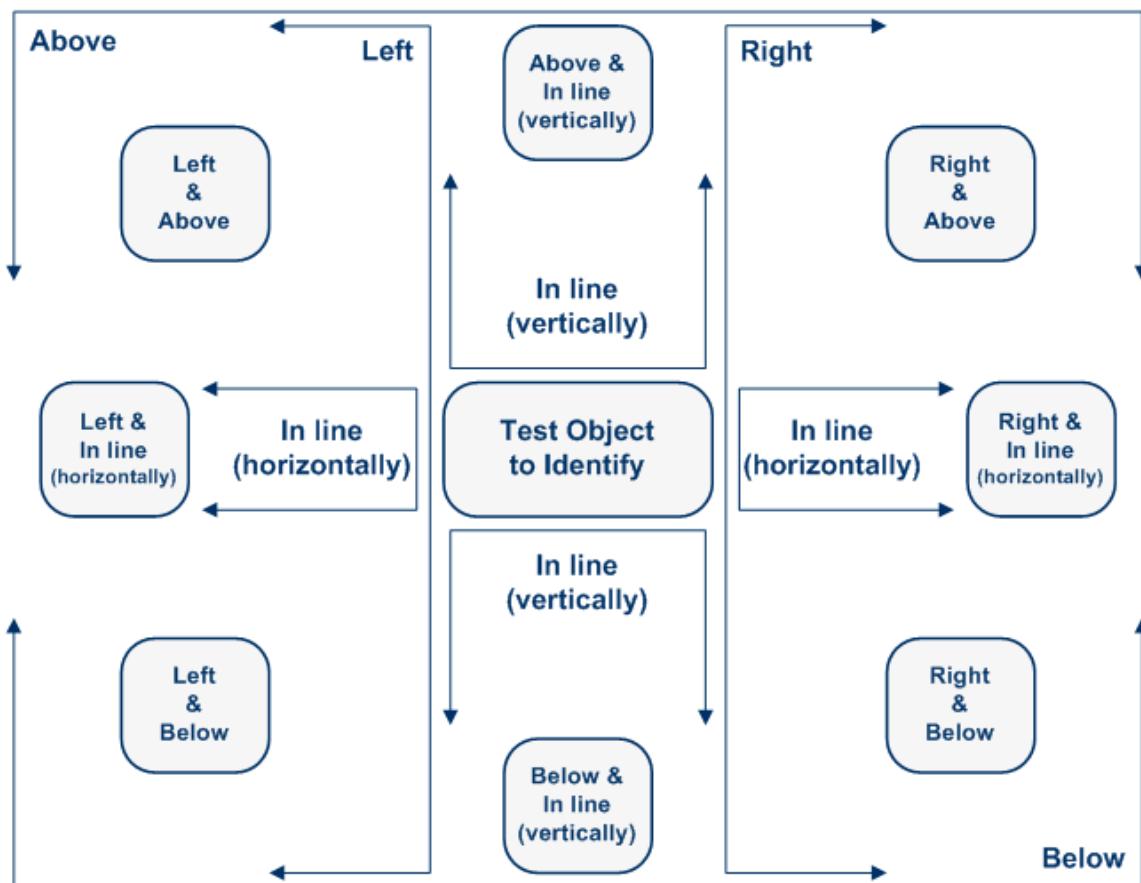
One way to remember which twin is which is by identifying *twin A* in this manner: Always look for the blue school bag and the twin's desk partner to locate *twin A*.

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. 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).

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.

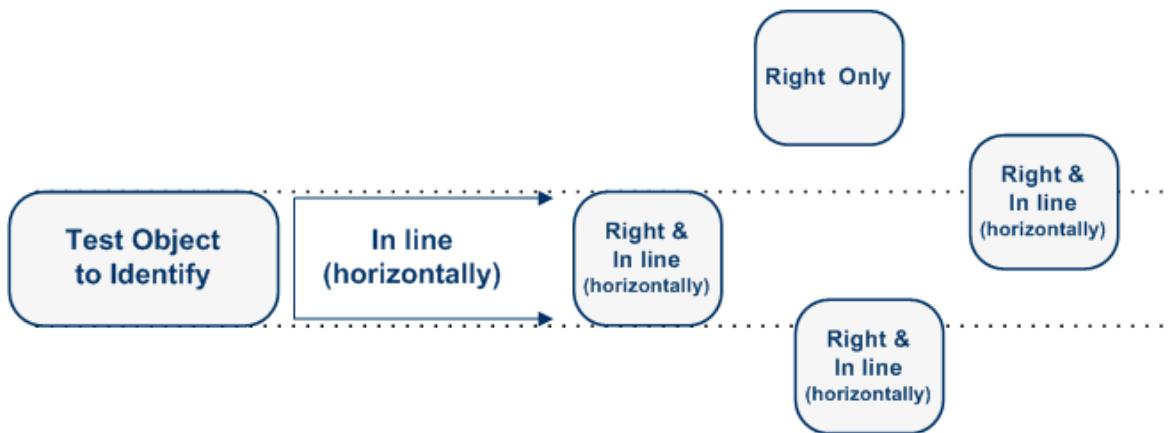


How in-line related objects are identified

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.



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.

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.

However, if you do not learn Smart Identification properties, you cannot enable the Smart Identification mechanism for an object later.

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.

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.

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:

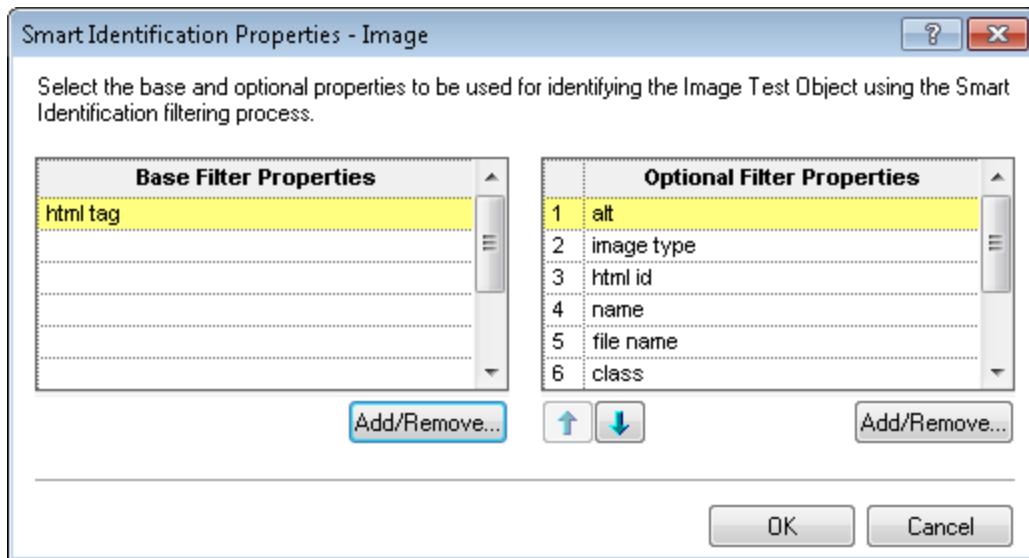
Name	Value
- Description properties	
image type	Image Button
html tag	INPUT
alt	Sign-In

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:

1. 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
alt	Login
name	login
file name	login.gif
class	<null>
visible	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.

Configure object identification for a test object class

Relevant for: GUI tests and components

Set properties for identifying an object

1. Select **Tools > Object Identification**.
2. In the Object Identification Dialog Box, set the properties that are learned for the test object description:
 - a. Select the environment.
 - b. Select the test object class.
 - c. Set mandatory and assistive properties. Click Add/Remove under the mandatory or assistive property lists and select the necessary properties from the Add/Remove Properties Dialog Box.
 - d. Select an ordinal identifier.

Set properties for Smart Identification

1. In the Object Identification Dialog Box, select the environment and test object class for which you want to configure Smart Identification.
2. Select the **Enable Smart ID** check box. The **Configure** button is enabled.
3. Click the **Configure** button to open the Smart Identification Properties Dialog Box.
4. In the Smart Identification Properties dialog box, set the base and optional properties.
5. Set the order in which the optional properties are used.
6. 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.

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.

1. In the Object Identification dialog box (**Tools > Object Identification**), in the **Environment** drop-down list, select **Standard Windows** and then click the **User-defined** button.
2. In the Object Mapping Dialog Box, 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 "[Configure object identification for a test object class](#)" on the previous page.



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.

Define a visual relation identifier for a test object - Use-Case scenario

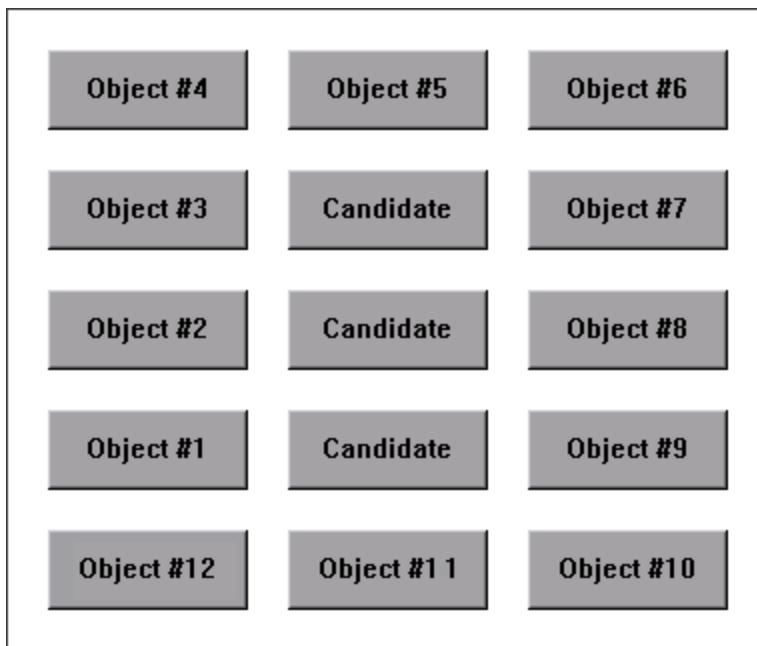
Relevant for: GUI tests and components

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 "["Maintain test objects in object repositories" on page 219](#).

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

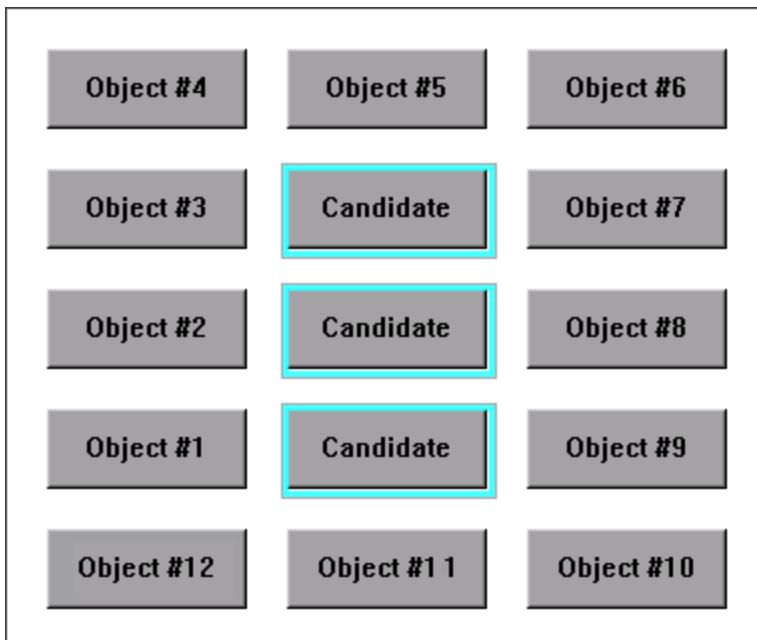
instruct UFT to identify the instance of the `Candidate` object that is located between the `Object #1` and `Object #9` object pair during every run session, even if the sorting order of the object pairs changes between run sessions.

Open the Visual Relation Identifier dialog box

1. In UFT, open the relevant object repository and select the `Candidate` test object to identify.
2. 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.
3. In the **Visual Relation Identifier Settings** row of the Object Repository window or Object Properties dialog box, click in the **Value** cell.

Highlight the objects 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.



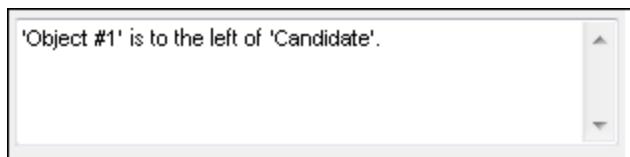
Click the **Preview** button again to restore the UFT window.

Define the first related test object using horizontal visual relations

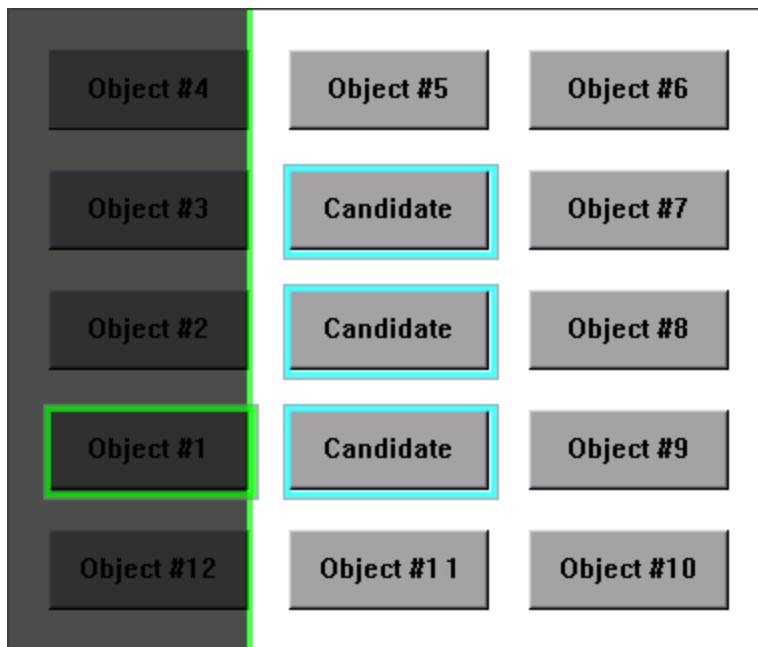
1. In the **Related Objects** area, click the **Add** button. The Select Test Object Dialog Box opens, enabling you to either select a test object from the object repository, or add an object from the application.

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.

2. 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.

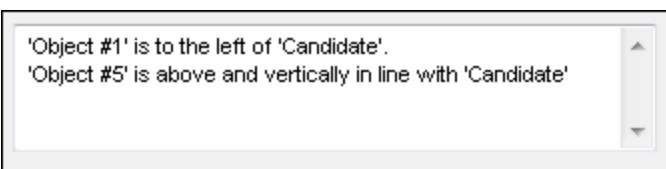


3. To verify that the visual relation is defined correctly, click the **Preview** button again. The main UFT window is hidden, and the visual relation identifier displays the objects that match the test object's description, including the currently defined visual relation. It also highlights the selected 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.

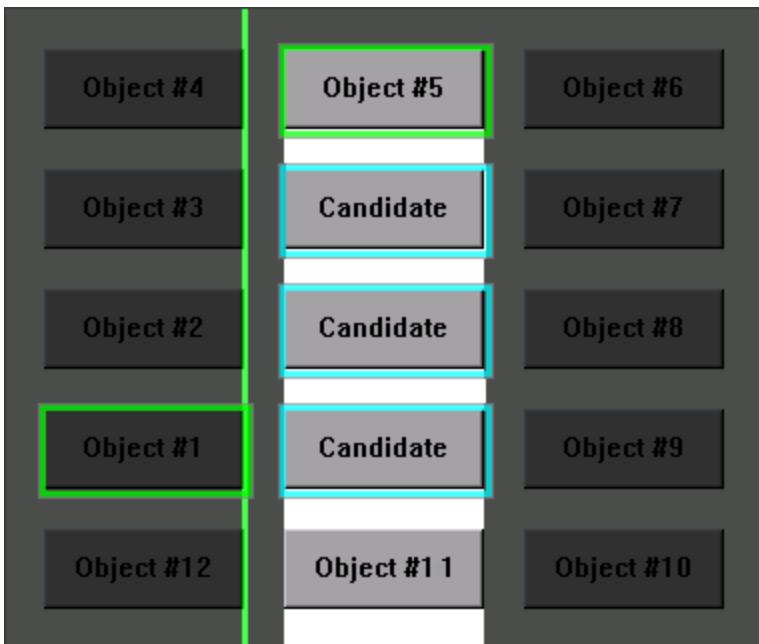


Define the second related object using vertical visual relations

1. 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.
2. 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.



3. 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.

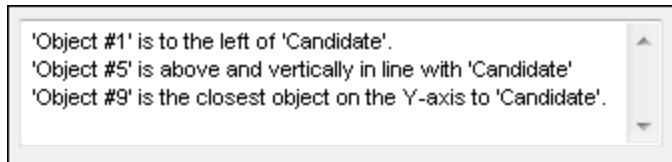


Define the third related object using distance visual relations

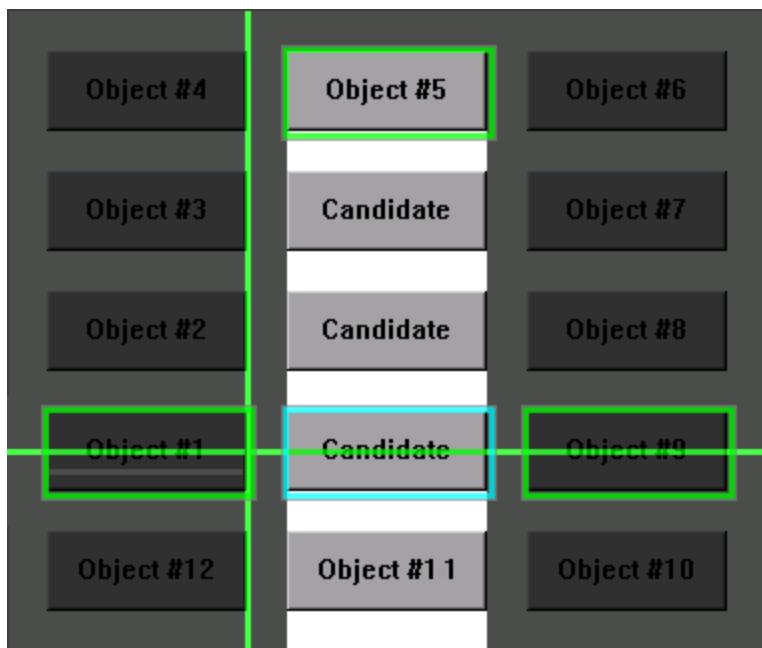
1. In the **Related Objects** area, click the **Add** button. The Select Test Object Dialog Box opens, enabling you to select another test object.

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.

2. 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.



3. 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.



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.

Chapter 31: Test Objects in Object Repositories

Relevant for: GUI tests 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. The local object repository is automatically created when you create a new action or component.

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.

To choose where to save objects, you need to understand the differences between local and shared object repositories:

Use this object repository type...	In these situations...
local object repository	<ul style="list-style-type: none"> You are creating single-action tests. You are creating simple tests or components, especially under the following conditions: <ul style="list-style-type: none"> You have only one, or very few, tests or components that correspond to a given application, interface, or set of objects. You do not expect to frequently modify object properties. You are new to using UFT. You can record and run tests or components without creating, choosing, or 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.
shared object repository	<ul style="list-style-type: none"> You are creating tests or components using keyword-driven methodologies (not by recording). You have several tests or components that test elements of the same application, interface, or set of objects. You often work with multi-action tests and regularly use the Insert Copy of Action and Insert Call to Action options. 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. 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. <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>

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).

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.

The Object Repository window vs. the 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 (the local object repository)	Object Repository Manager (the shared object repository)
Adding and deleting test objects	✓	✓
Highlighting a test object in your application	✓	✓
Locating a test object in the object repository	✓	✓
Specifying or modifying object property values	✓	✓
Updating object property values directly from objects in your application	✓	✓

Functionality	Object Repository window (the local object repository)	Object Repository Manager (the shared object repository)
Restoring default mandatory object property values	✓	✓
Renaming test objects	✓	✓
Adding properties to an object's description	✓	✓
Defining new identification properties	✓	✓
Removing properties from a test object description	✓	✓
Exporting local objects to a shared object repository	✓	✗
Adding a test object	✓	✓
Using repository parameters	✗	✓
Merging multiple object repositories	✗	✓
Exporting the repository an XML file	✗	✓

aLocal copies of objects in shared object repositories

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.

If you copy an object to the local object repository, its parent objects are also copied to the local object repository. However, you cannot copy an object to the local object repository if an object or its parent objects use unmapped repository parameters.

If an object or its parent objects are parameterized using 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.

Adding and deleting test objects

Relevant for: GUI tests and components

When you create an object repository, you can add test objects to it in various ways:

- Use the **Navigate and Learn** option to add objects to a shared object repository
- Record steps to add objects 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.)
- 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.

For task details on how to add and delete objects, see "[Add a test object to an object repository](#)" on page 216.

Maintaining identification properties

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.

There are a number of things you can do to maintain your object's properties:

Specify or modify property values	You can: <ul style="list-style-type: none">specify or modify values for properties in the test object description by using a constant value (either a simple value or a constant value that includes regular expressions) or a parameterchange the set of properties used to identify that object.automatically update the description of one or more test objects in your object repository based on the actual updated object properties in your application
--	--

Update identification properties from an object in Your application	<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. 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>
Restore default mandatory properties for a test object	<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.</p> <p>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>
Rename test objects	<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>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 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 Automatically update test and component steps when you rename test objects check box in the General pane of the Options dialog box (Tools > Options > GUI Testing tab > General node).</p>

Add properties to a test object description	<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>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.</p>
Define new identification properties	You can add any valid identification property to a test object description, even if it does not appear in the Add Properties Dialog Box.
Add ordinal identifiers	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. 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.

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.

If 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.



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.

For task details, see "[Manage repository parameters](#)" on page 224.

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.

After defining the repository parameter, mapping a repository parameter specifies that the property values are taken from:

Tests	<ul style="list-style-type: none">• Data table• random number• environment• test or action parameter
Components	<ul style="list-style-type: none">• local parameter• component parameter

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.

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.

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.

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.

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.

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.

Add test objects to the object repository

1. Perform one of the following:

To add to the local object repository	In the Object Repository window, Select Object > Add Objects to Local or click the Add Objects to Local toolbar button 
To add to a shared object repository	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. In your application, 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 selected objects is the bottom of the hierarchy	The object is added directly to the object repository.
If the selected object is a parent (container) object	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

For details, see "[Add an Insight object](#)" on page [239](#).

Add a test object to the local object repository while adding a step

1. In the Item cell of the Keyword View, from the drop-down list, select **Object from repository**.
2. In the Select Test Object Dialog box, select the object to add.
3. Click OK to create a step using the selected object.

The selected step is added to the test and the object is added to the action or component's local object repository.

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.
3. In the **Environment** drop-down list, select the UFT add-in environment for your object.
4. In the **Class** drop-down list, select the object type.
5. In the **Name** edit box, give the object a name.
6. In the **Test Object details** area, provide the necessary identification properties for the object.
7. Click **Add** to insert the new object in the object repository.
8. Click **Close** to return to the main object repository window.

Add a test object to the object repository with the Object Spy

1. Click the **Object Spy** button  from UFT or the Object Repository Manager.
2. In the Object Spy window, click the pointing hand. UFT is minimized.
3. In your application, click on the object you want to add. The properties of the object are displayed in the main part of the Object Spy window.
4. In the Object Spy, select the appropriate object from the hierarchy to add.
5. 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 from the Active Screen

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

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).

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

1. In the Object Repository window or the Object Repository Manager, select the test object whose property value you want to specify.
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 .

Update identification properties

1. In the object repository tree, select the test object whose description you want to update.
2. Select **Object > Update from Application** or click the **Update from Application** button . UFT is hidden, and the pointer changes into a pointing hand.
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. 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.
4. In the New Property dialog box, provide details for your property and click **OK**.

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.
4. In the Ordinal Identifier dialog box, provide the ordinal details and click **OK**.

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.

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.

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**.

In addition, 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.

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

Add a **SetTOProperty** statement in a user-defined function, or in your action with the following syntax:

```
Object(description).SetTOPropertyProperty, Value
```

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.

Prerequisites

If your shared object repository is stored in ALM, connect to ALM either from UFT or from the Object Repository Manager by clicking the **ALM Connection** button .

Enable editing for a shared object repository

Select **File > Enable Editing** or click the **Enable Editing** button . The shared object repository becomes editable.

Associate a shared object repository with actions or components

Do one of the following:

To associate it with an action	<ol style="list-style-type: none">1. In the Solution Explorer, right-click the action name node and select Associate Repository with Action.2. In the Open Shared Object Repository dialog box, select your object repository and click Open.
To associate it with a component	<ol style="list-style-type: none">1. In the application area, open the Object Repositories tab.2. In the Object Repositories tab, at the top of the tab, click the New button . A new row is added to the list of object repositories.3. At the right side of the object repositories list, in the new row, click the Browse button.4. In the Open Shared Object Repository dialog box, select your object repository and click Open.

Merge object repositories into shared ones

In the Object Repository Manager, select **Tools > Update from Local Repository** and select the object repositories to merge.

Add test objects using Navigate and Learn

1. Select **Object > Navigate and Learn**. 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. 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. The Object Repository Manager is redisplayed, showing the objects you just added to the shared object repository.

Manage repository parameters

1. In the Object Repository Manager, select **Tools > Manage Repository Parameters**.
2. In the Manage Repository Parameters dialog box, add and edit repository parameters as needed.

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.

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

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.

Locate an object in an object repository

Relevant for: GUI tests and components

Find an object

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 (Object Repository) 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. In the test object tree, select the object you want to view.



Note: This option is not supported for Insight test objects.

3. 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.
4. 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.

Known Issues- Object Repositories

Relevant for: GUI tests and components

- 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.

- 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 "[aLocal copies of objects in shared object repositories](#)" on page 209.
- 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.

Chapter 32: Comparing and Merging Object Repositories

Relevant for: GUI tests and components

When working with object repositories, you may need to compare object repositories to see differences or merge multiple object repositories to simplify your testing assets.

To assist with this UFT provides tools to help you compare and merge object repositories:

Object Repository Comparison Tool	<p>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.</p> <p>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.</p> <p>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.</p> <p>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.</p>
--	---

Object Repository Merge Tool	<p>UFT enables you to merge two shared object repositories into a single shared object repository using the Object Repository Merge Tool.</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>For task details, see "Merge two shared object repositories" on page 232.</p>
-------------------------------------	---

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.

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.

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.

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.

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.

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.

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.

Prerequisites

- Make sure that a GUI test or component is currently open.
- Make sure that the Object Repository Manager window is open.
- Make sure the color settings are configured to match your needs.

Select the Shared Object Repositories to compare

1. In the Object Repository Manager window, select **Tools > Object Repository Comparison Tool**.
2. In the New Comparison dialog box, specify the two object repository files you want to compare.

Analyze the initial comparison results

After the comparison is complete, view the results summary in the Comparison Statistics Dialog Box.

Analyze the detailed comparison results

Review and analyze the comparisons between the repositories in the Object Repository Comparison Tool Main Window.

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 (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 (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 (Object Repository Comparison Tool).

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.

Prerequisites

- Make sure that a GUI test or component is in focus.
- Make sure that the Object Repository Manager window is open.
- Make sure the resolution and color settings are configured to match your needs.

Select the shared object repositories to merge

1. 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.
2. Specify the two object repository files you want to merge.

Analyze the initial merge results

After the merge is complete, you can view the results summary in the Merge Statistics Dialog Box (Object Repository Merge Tool).

Analyze the detailed merge results

Review and analyze the merge between the repositories in the Object Repository Merge Tool Main Window.

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 (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 (Object Repository Merge Tool).

Adjust object conflict resolutions

If one or more of the merge resolutions does not match your needs, follow the steps below to adjust them:

1. 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.

2. 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.
3. 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.

Save the target object repository

When the object conflicts are resolved satisfactorily, save the new merged shared object repository.

Update a shared object repository from a local object repository

Relevant for: GUI tests and components

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.
- Make sure the resolution and color settings are configured to match your needs.

Select the shared object repository and the local repositories that you want to merge into it

1. 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**.
2. Select **Tools > Update from Local Repository** to open the Update from Local Repository Dialog Box.
3. 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**.

Analyze the initial merge results

View the initial merge results in the Merge Statistics Dialog Box (Object Repository Merge Tool).

Analyze the detailed merge results

Review and analyze the detailed merge results in the Object Repository Merge Tool - Multiple Merge Window.

Utilize additional tools to help you perform the comparison - optional

- Filter the objects and show only the objects that you want to view by using the Filter Dialog Box (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 (Object Repository Merge Tool).

Adjust object conflict resolutions

If one or more of the merge resolutions does not match your needs, follow the steps below to adjust them:

1. 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.

2. 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.
3. 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.

Save the target object repository

When the object conflicts are resolved satisfactorily, save the new merged shared object repository.

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.

Chapter 33: Extending UFT Object Identification

Relevant for: GUI tests and components

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 242.

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:

Description property	UFT stores an image of the object for the description property. This image is used later to identify the object.
Ignore areas	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.
Object configuration	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.
Visual relation identifiers	After UFT creates a description for an Insight test object, add visual relation identifiers to improve identification of the object.
Similarity identification property	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.

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 *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.

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.

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**).

Automatic

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.

Manual	<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> 
---------------	---

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.



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 *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 *UFT Object Model Reference for GUI Testing*.

- **Modify the ordinal identifier** created for the test object. For details, see "[Ordinal identifiers](#)" on page 190.
- **Define visual relation identifiers** for the test object. For details, see "[Visual relation identifiers](#)" on page 192.

For more details , see "[Maintain test objects in object repositories](#)" on page 219.



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:

Element tree	The hierarchy of elements in the application, which displays a logical division and hierarchy of all user interface elements in the application
Control Type property	The appearance and functionality of the object.
Control Patterns	<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:
50000	<i>UIAButton</i>
50001	<i>UIACalendar</i>
50002	<i>UIACheckBox</i>
50003	<i>UIACComboBox</i>

50004	<i>UIAEdit</i>
50005	<i>UIAHyperLink</i>
50008	<i>UIAList</i>
50013	<i>UIARadioButton</i>
50015	<i>UIASlider</i>
50018	<i>UIATab</i>
50023	<i>UIATree</i>
50028	<i>UIATable</i>
50031	<i>UIASplitButton</i>
50032	<i>UIAWindow</i>
50036	<i>UIATable</i> 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 *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:
ExpandCollapse	<ul style="list-style-type: none"><i>.Expand</i><i>.Collapse</i>
Grid	<i>.GetItem</i>
Invoke	<i>.Click</i>
RangeValue	<ul style="list-style-type: none"><i>.Decrement</i><i>.Increment</i><i>.SetValue</i>
Scroll	<ul style="list-style-type: none"><i>.Scroll</i><i>.ScrollDown</i><i>.ScrollLeft</i><i>.ScrollRight</i><i>.ScrollDown</i><i>.SetScrollPercent</i>
ScrollItem	<i>.ScrollIntoView</i>
Selection	<ul style="list-style-type: none"><i>.Select</i><i>.AddToSelection</i><i>.RemoveFromSelection</i><i>.GetSelection</i>
SelectionItem	<ul style="list-style-type: none"><i>.Select</i><i>.AddToSelection</i><i>.RemoveFromSelection</i>
Table	<ul style="list-style-type: none"><i>.GetColumnHeaders</i><i>.GetRowHeaders</i>

TableItem	<ul style="list-style-type: none"><i>.GetColumnHeaderItems</i><i>.GetRowHeaderItems</i>
Text	<i>.GetText</i>
Transform	<ul style="list-style-type: none"><i>.Move</i><i>.Resize</i><i>.Rotate</i>
Toggle	<i>.Set</i>
Value	<i>.SetValue</i>
Window	<ul style="list-style-type: none"><i>.Maximize</i><i>.Minimize</i><i>.Restore</i><i>.Close</i>

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.

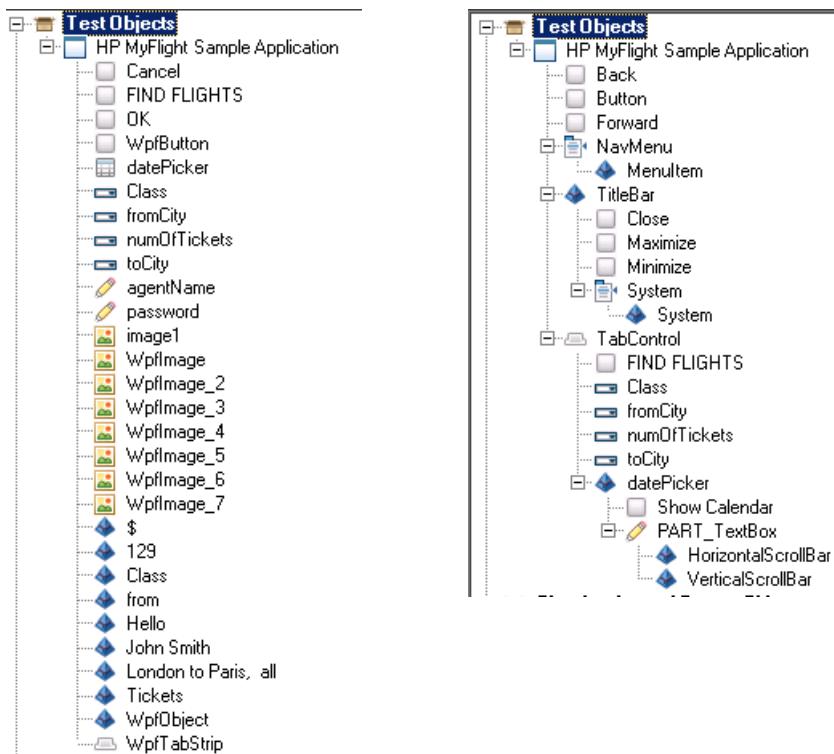
This can mean:

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

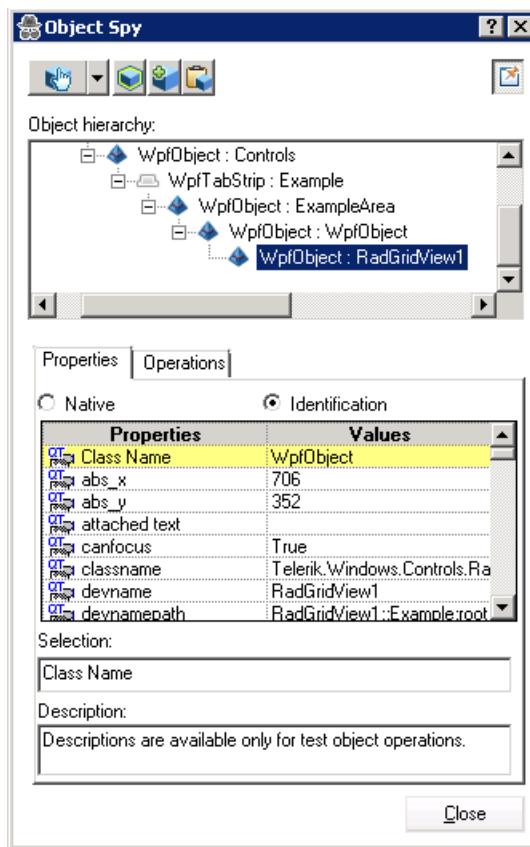
Drag a column header and drop it here to group by that column

	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	

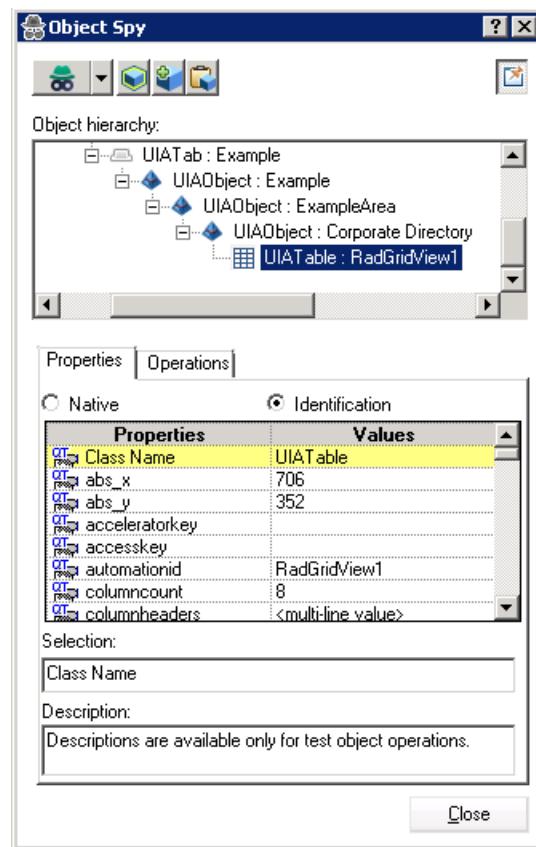
When UFT uses the Spy to view the main area of this window, the results are very different.

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.

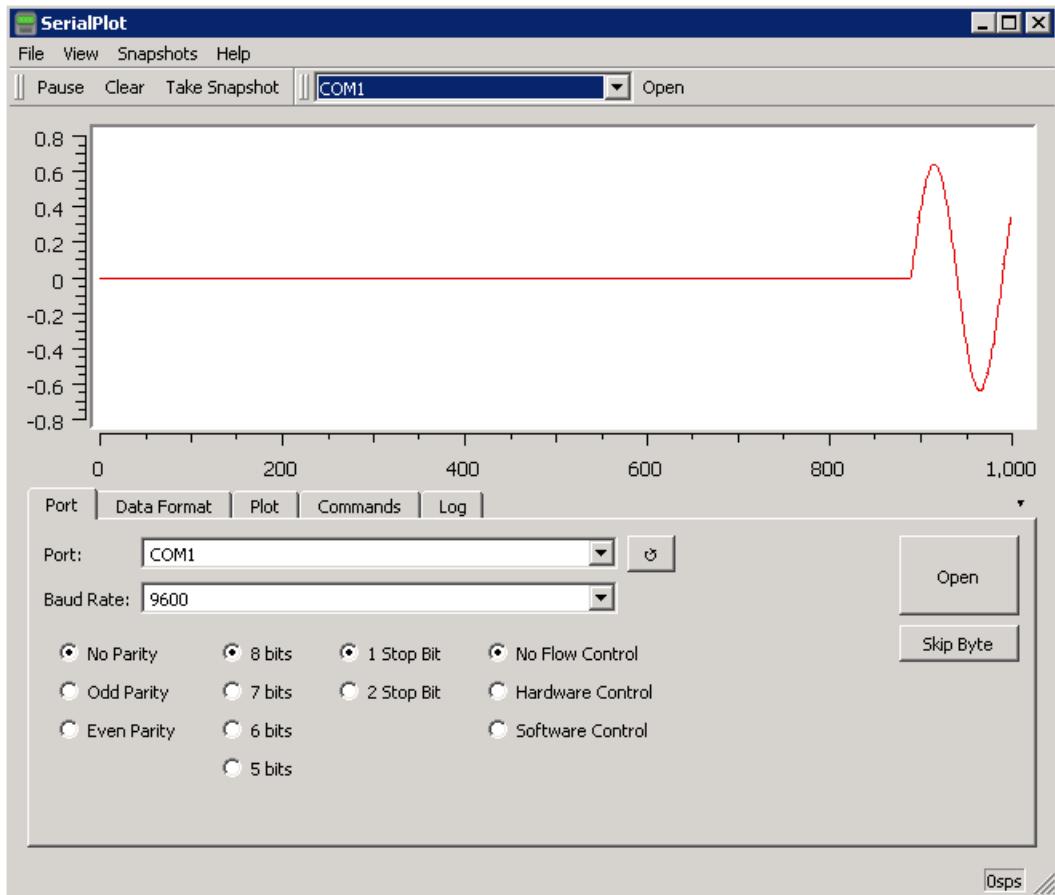
- When UFT does not support your technology or your version of a technology**

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.

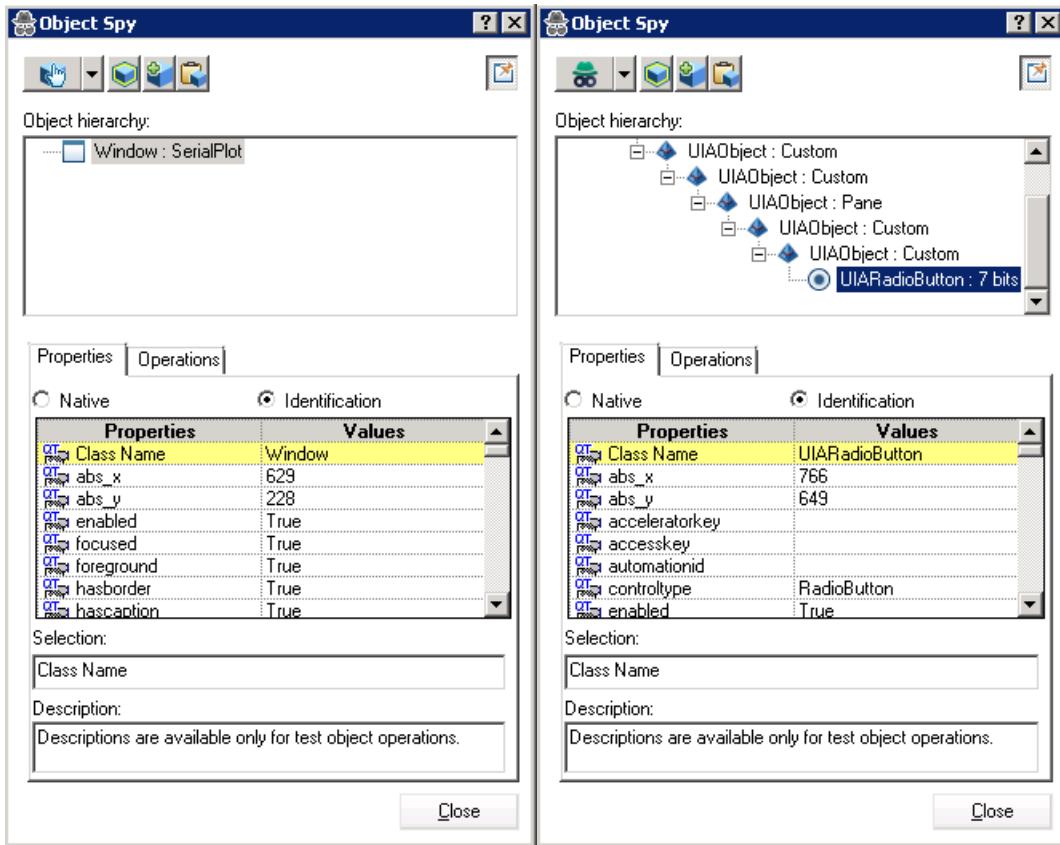
Use UI Automation when it will enable you to identify objects in your application when UFT otherwise could not.

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 for a number of native methods accessible with the **.Object** method. These methods are available for all UI Automation objects.

Each object is assumed to be a collection, even if it is a regular object. This enables UFT and the **.Object** method to work with both a single object and a collection.

Note: When using the **.Object** method, UFT returns the value "as-is", which can be complex and include several different flags. See the MSDN reference for the value of these numerical properties. For example, for the State property, see [here](#).

Method	Description
Compare	<p>Checks if an element is equal to the selected object.</p> <p>Syntax</p> <pre>.Object.CompareElement</pre> <p>Parameters</p> <p>Element: The element to which to compare the selected element.</p>
ElementFromPoint	<p>Finds an object at the selected coordinates.</p> <p>Syntax</p> <pre>.Object.ElementFromPointx,y</pre> <p>Parameters</p> <ul style="list-style-type: none"> • X: The x-coordinate at which to find the object. • Y: The y-coordinate at which to find the object.
GetChildren	<p>Returns a list of all children of the selected object.</p> <p>Syntax</p> <pre>.Object.GetChildren</pre>
GetFirstChild	<p>Returns the first child object of the selected object. If there are no child objects, UFT returns a NULL value.</p> <p>Syntax</p> <pre>.Object.GetFirstChild</pre>
GetLastChild	<p>Returns the last child object of the selected object. If there are not child objects, UFT returns a NULL value.</p> <p>Syntax</p> <pre>.Object.GetChildren</pre>
GetNextSibling	<p>Returns the element which is next in the overall hierarchy to the selected object.</p> <p>Syntax</p> <pre>.Object.GetNextSibling</pre>

Method	Description
GetParent	Returns the parent element for a selected object. Syntax .Object.GetParent
GetPreviousSibling	Returns the element which is prior in the overall hierarchy to the selected object. Syntax .Object.GetPreviousSibling

 See also:

- ["Native properties and operations" on page 556](#)
- The `.Object` property in the *UFT Object Model Reference for GUI Testing*

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.

Learn objects in UI Automation mode

1. Activate the UI Automation mode in UFT before learning the objects.
 - Select the **Use UI Automation by default** open in the **Windows Applications** > **Advanced** pane of the Options dialog (**Tools** > **Options** > **GUI Testing** tab > **Windows Applications** > **Advanced** node).

- In the Object Repository Manager window, select **UI Automation** from the Learn mode dropdown in the toolbar.
2. Add objects to your object repository using one of the following:

In the Object Spy	Add Object to Repository button 
In the Object Repository window or Object Repository Manager	<p>Do one of the following:</p> <p>Add UI Automation Objects to Local button </p> <p>The Add UI Automation Objects to Local button is not available unless:</p> <ul style="list-style-type: none"> • You select the Use UI Automation by Default option in the Windows Applications > Advanced pane of the Options dialog box; and • You have activated the UI Automation mode by clicking the UI Automation button  in the Object Repository window toolbar. <p>Add test objects using the Navigate and Learn toolbar</p>
In the Keyword View	<ol style="list-style-type: none"> a. In the Item cell, from the drop-down list, select Object from repository. b. In the Select Test Object dialog, from the pointing hand button, click the drop-down arrow and select UI Automation. c. Click the pointing hand button. UFT is minimized. d. Select the object from your application. The object is added (with its parent objects if necessary) to the Select Test Object dialog box. e. Click OK. The object is now added to the local object repository.

Record 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:

- Recording may add additional unnecessary steps to your test. Remove the unneeded steps manually after finishing your recording session.
- 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

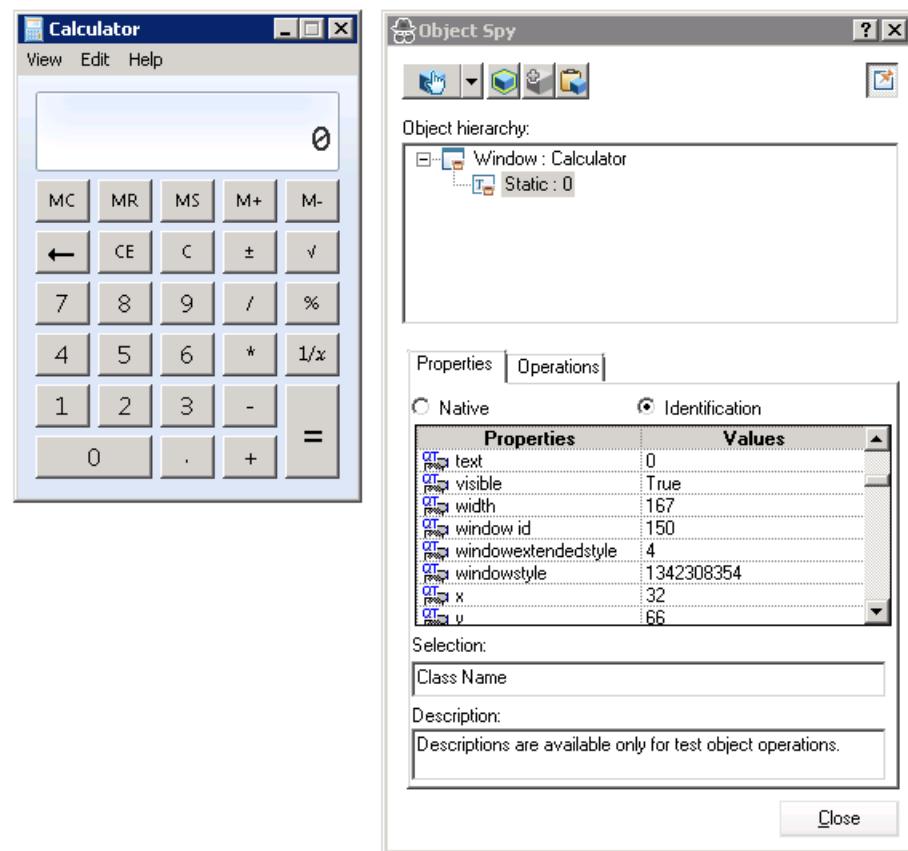
UFT may fail to find a test object during a test run because the expected properties are incorrect, and may fail to identify and learn an object because it is not a supported object for a specific technology.

In such cases, use a Static test object or static identification properties to identify the object.

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.



Example: 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.



Example: 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.



Example: 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.



Example: 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.



Example: 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.

Known Issues - UI Automation Support

Unsupported features	The following features are not supported together with UI Automation: <ul style="list-style-type: none">• Active Screen• F1 support for objects and methods• Table checkpoints• Using UI Automation with the UFT Add-in for ALM• Recording on Drag and Drop methods
UI Automation on Windows 10	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.)
Recording on environments with slow GUI refresh	If you are using UI Automation to record in an environment with a slow GUI refresh such as a slow RDP connection, recording can experience unexpected and unstable behavior.
UI Automation and Java applications	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. This is due to changes in the Java framework between Java 6 and Java 8.
Regular expressions as property values	The controltype identification property cannot have a value that is a regular expression.

UIA container objects with large numbers of elements	<p>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.</p> <p>Workaround: Do one or more of the following:</p> <ul style="list-style-type: none">• In your application, implement the ItemContainer pattern with the FindItemByPropertyO method.• Manually add additional identification properties for the items in the object, such as index. These properties are not added by default.• Use .Object methods to access items within the control.
UIA with other "spy" tools	If you start other "spy" tools, such as the Microsoft Inspect tool, UFT experiences unexpected behavior.

Chapter 34: Virtual Objects

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.

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.

Define virtual objects for unsupported objects

Relevant for: **GUI tests and scripted GUI components**

Display the object 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.

Note: You can define virtual objects only for objects on which UFT records **Click** or **DblClick** methods. Otherwise, the virtual object is ignored.

Use the Virtual Object wizard

Open the Virtual Object wizard (**Tools > Virtual Object > New Virtual Object**).

Chapter 35: Checkpoints in GUI Testing

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.

Adding existing checkpoints

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.

For details of how to insert existing checkpoints, see Add Existing Checkpoint Dialog Box.

Checkpoint types

Relevant for: GUI tests and components

You can insert the following checkpoint types to check objects in an application:

Checkpoint Type	Description
Standard Checkpoint	<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 "Supported Checkpoints" on page 917).</p> <p>For details on standard checkpoints, see "Standard checkpoints" on page 266.</p>
Image Checkpoint (tests and scripted components only)	<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 "Supported Checkpoints" on page 917).</p> <p>For details on image checkpoints, see "Standard checkpoints" on page 266.</p>
Accessibility Checkpoint (tests and scripted components only)	<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 "Supported Checkpoints" on page 917).</p> <p>For details on accessibility checkpoints, see "Accessibility checkpoints" on page 267.</p>

Checkpoint Type	Description
Bitmap Checkpoint	<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> <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 "Supported Checkpoints" on page 917.</p> <p>For details on bitmap checkpoints, see "Bitmap checkpoints" on page 267.</p>
Database Checkpoint (tests and scripted components only)	<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 "Supported Checkpoints" on page 917).</p> <p>For details on database checkpoints, see "Database checkpoints" on page 271.</p>
File Content Checkpoint (tests only)	<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 on specific pages in that .pdf.</p> <p>File content checkpoints are supported for all add-in environments (see "Supported Checkpoints" on page 917).</p> <p>For details on file content checkpoints, see "File Content checkpoints" on page 272.</p>

Checkpoint Type	Description
Page Checkpoint (tests and scripted components only)	<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 "Supported Checkpoints" on page 917).</p> <p>For details on page checkpoints, see "Page checkpoints" on page 273.</p>
Table Checkpoint (tests and scripted components only)	<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 "Table checkpoints" on page 273.</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 "Supported Checkpoints" on page 917.</p>
Text Checkpoint (tests and scripted components only)	<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 "Supported Checkpoints" on page 917).</p> <p>For details on text checkpoints, see "Text and text area checkpoints" on page 274.</p>

Checkpoint Type	Description
Text Area Checkpoint (tests and scripted components only)	<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 <Num>, where <Num> is replaced by the four digit code entered in a form elsewhere in the application. You can 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 "Supported Checkpoints" on page 917). Text area checkpoints are also supported for some other add-in environments, such as Java.</p> <p>For details on text area checkpoints, see "Text and text area checkpoints" on page 274.</p>
XML Checkpoint (tests and scripted components only)	<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 "XML checkpoints" on page 279</p> <p>The XML Checkpoint (Web Page/Frame) option is supported for the Web add-in environment. The XML Checkpoint option is supported for all add-in environments (see "Supported Checkpoints" on page 917).</p>

Standard checkpoints

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.

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

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.

You can instruct UFT to create automatic accessibility checkpoints for every page in all tests. 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:

- ["Insert a checkpoint step" on page 280](#)
- Checkpoint Properties Dialog Box

Bitmap checkpoints

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.

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.

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:

RGB Tolerance	<p>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.</p> <p>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.)</p> <p>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.</p> <p>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.</p>
----------------------	---

Pixel Tolerance	<p>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.</p> <p>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.</p> <p>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.</p>
Image Similarity	<p>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.</p> <p>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.</p> <p>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.</p>

Custom Comparers	<p>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.</p> <p>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.</p> <p>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 - Use-case scenario" on page 297". For details on developing or installing custom comparers, see ""Developing Custom Comparers for Bitmap Checkpoints" on page 296".</p> <p>If you select a custom comparer, some of the options in the Bitmap Checkpoint Properties Advanced Settings dialog box are different.</p>
-------------------------	---

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.

Database checkpoints

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.

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.

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. 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. If the expected data and the current results do not match, the database checkpoint fails.

File Content checkpoints

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 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

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.

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

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.
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.

Text and text area checkpoints

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).



Note: 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.

- **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 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.

Text recognition in run-time

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.

In addition, 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 **.GetVisibleText**, the **.GetTextLocation** test object methods, the **TextUtil.GetText** or **TextUtil.GetTextLocation** reserved object methods, or the **.GetText** (for Terminal Emulator objects).



Note: Text recognition is not supported for objects in the Active Screen.

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.

You should also note the following considerations for performing effective text recognition:

Fonts in your text	<ul style="list-style-type: none">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. 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.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.
Colors and color contrast	<ul style="list-style-type: none">The color schema of the background should be permanent and without gradient.High contrast between the background and text is best for text recognition (for example, black text on a white background).
Text within images	<ul style="list-style-type: none">If your text is found within an image, it is recommended to the 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).

Dimension for text recognition	<ul style="list-style-type: none">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.
---------------------------------------	--

Checking Text in an image - Use-case scenario

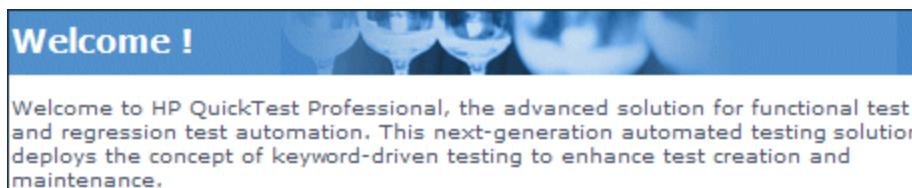
Relevant for: GUI tests and scripted GUI components

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.

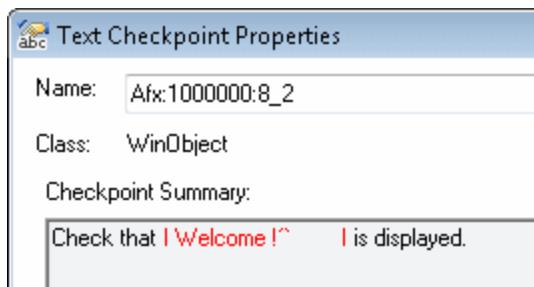
For one of his projects, Ben also needed to verify the text in the graphics, so he decided to use text checkpoints.

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.

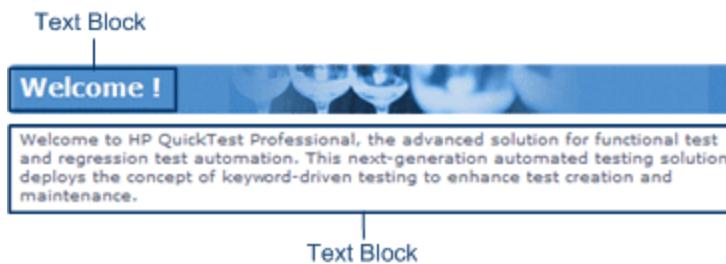
Ben then inserted a text checkpoint on the entire area shown above and received the following results in the Text Checkpoint Properties dialog box:



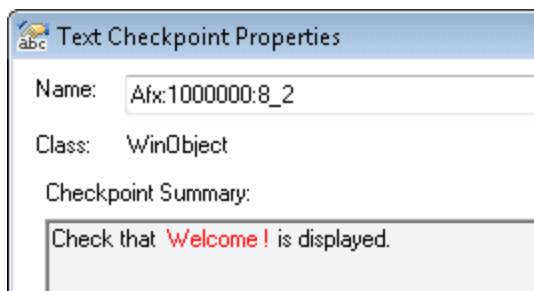
Ben noticed that there were extra characters in the **Checkpoint Summary** area of the text checkpoint, but he did not know why.

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.

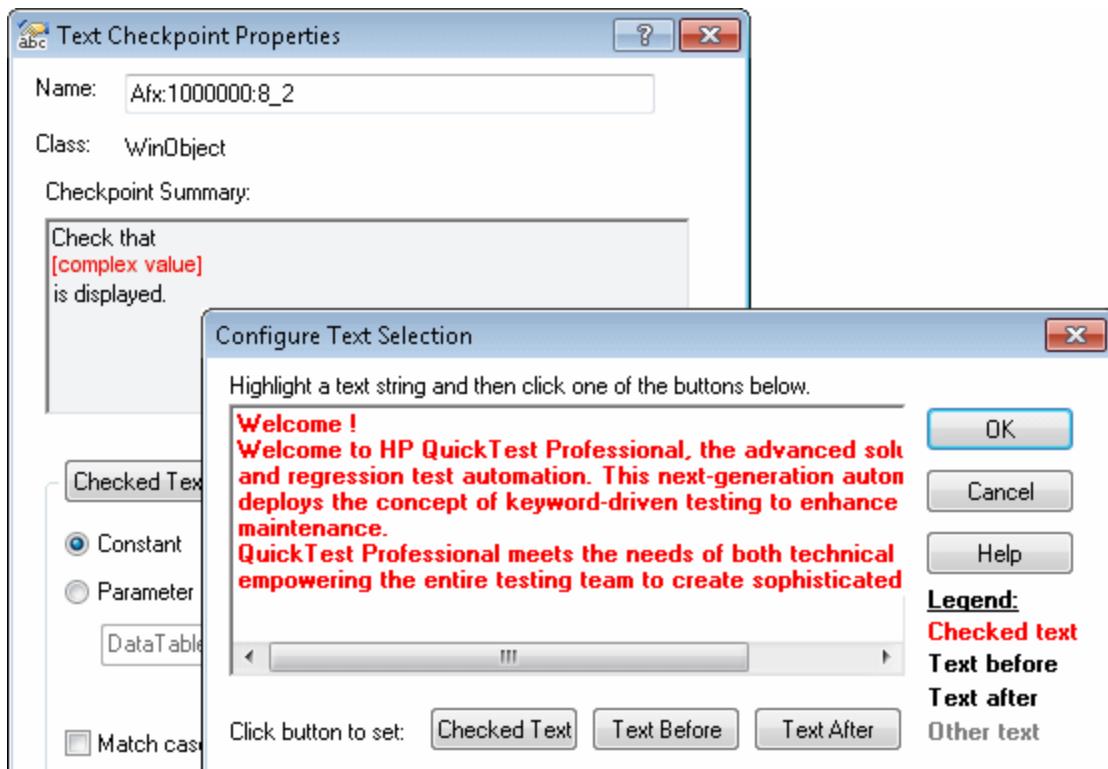
George also pointed out that the area Ben defined for the text checkpoint consisted of multiple text blocks because the text was not uniform in font size, color, or background. The title area consisted of white characters on a blue-gray background, while the remaining text was smaller and consisted of 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

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.

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. During a run session, UFT compares the expected results of the checkpoint to the current results.

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.

In addition, 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.

For details on XML objects and methods, see the **Supplemental Objects** section of the *UFT Object Model Reference for GUI Testing*.

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/>

Insert a checkpoint step

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 916

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.

Important information before inserting the checkpoint

Object visibility in application	<ul style="list-style-type: none">During an editing session, make sure the object is visible in your application before inserting a standard checkpoint.For bitmap checkpoints: 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 MakeVisible statement (for relevant environments) prior to your bitmap checkpoint step. For details on the MakeVisible method, see the specific object methods and properties in the <i>UFT Object Model Reference for GUI Testing</i>.For file content checkpoints: The source file must be located on the file system.
Availability	<ul style="list-style-type: none">Recording sessionsEditing sessionsActive Screen (not supported for file content checkpoints or XML checkpoints)

Important Information	<ul style="list-style-type: none">Checkpoints can be viewed in the following modes:<ul style="list-style-type: none">Simple Mode: Displays only the basic properties and expected values of the checkpoint.Advanced Mode: Displays all supported properties and expected values of the checkpoint.In ALM, you cannot create, edit, or rename checkpoints for keyword GUI components.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.For bitmap checkpoints: 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.For text or text area checkpoints:<ul style="list-style-type: none">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 (Tools > Options > GUI Testing tab > Text Recognition node).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.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.
 - 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 *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.



Example: 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 single cell to or remove it from the check	Double-click the cell
Add an entire row to or remove it from the check	Double-click the row header
Add an entire column to or remove it from the check	Double-click the column header.
Add all cells to or remove all cells from the check	Double-click the column header.
Add a range of cells to the check	Select the cells to add to the check and click the Add to Check button
Remove a range of cells from the check	Select the cells to remove from the check and click the Remove from Check button



Example: 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 single cell to or remove it from the check	Double-click the cell
Add an entire row to or remove it from the check	Double-click the row header
Add an entire column to or remove it from the check	Double-click the column header.
Add all cells to or remove all cells from the check	Double-click the column header.



Add **a range of cells** to the check

Select the cells to add to the check and click the **Add to Check** button

Remove **a range of cells** from the check

Select the cells to remove from the check and click the **Remove from Check** 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**.



Example: 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.

Include and ignore areas when comparing a bitmap - Use-case scenario

Relevant for: GUI tests and components

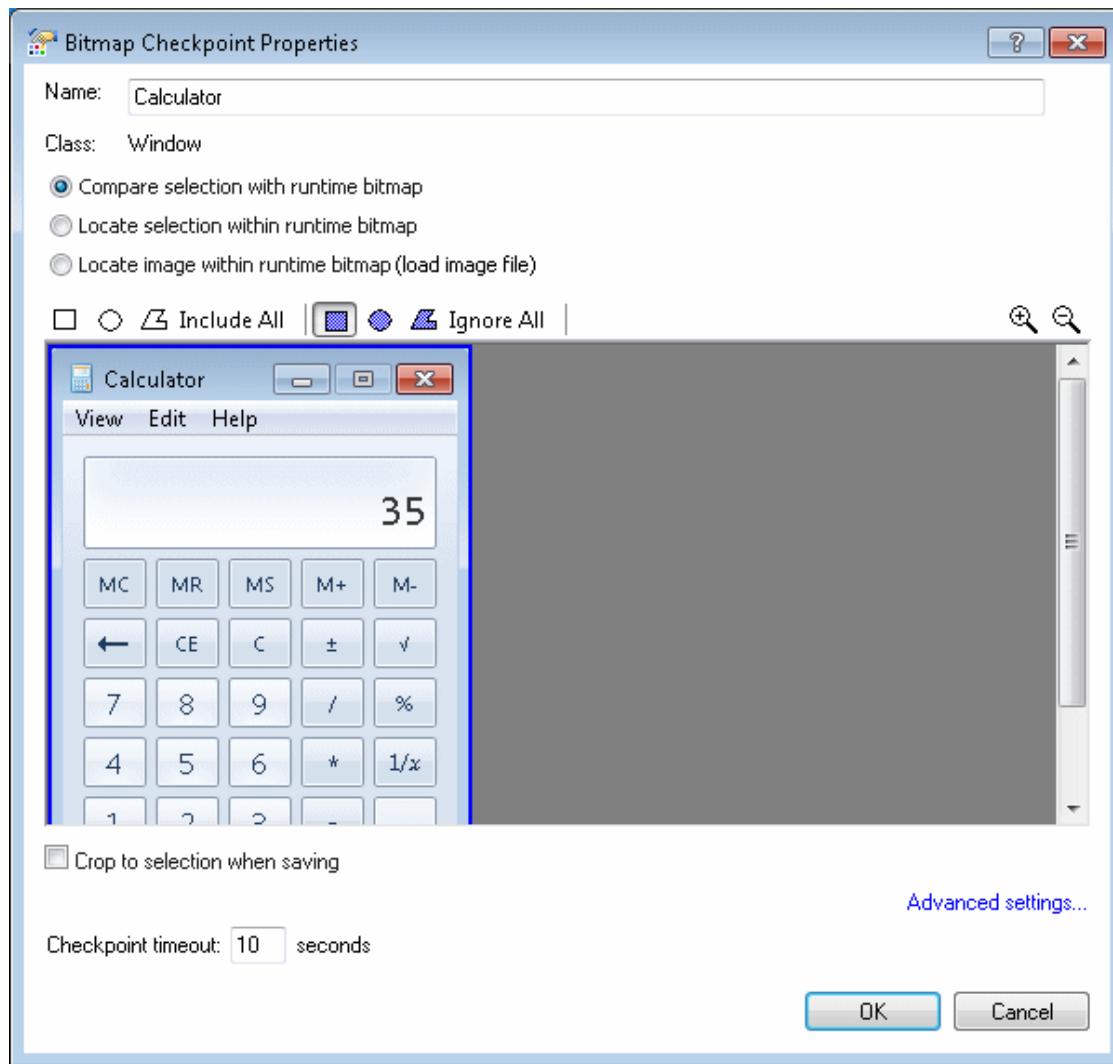
Suppose you want to compare an expected bitmap with an actual bitmap of the object in your application, but there are areas of this runtime bitmap that might be affected by dynamic values and parameters. If you include these areas in the checkpoint, running the steps with different values may cause the 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 "[Insert a checkpoint step](#)" on [page 280](#).

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.



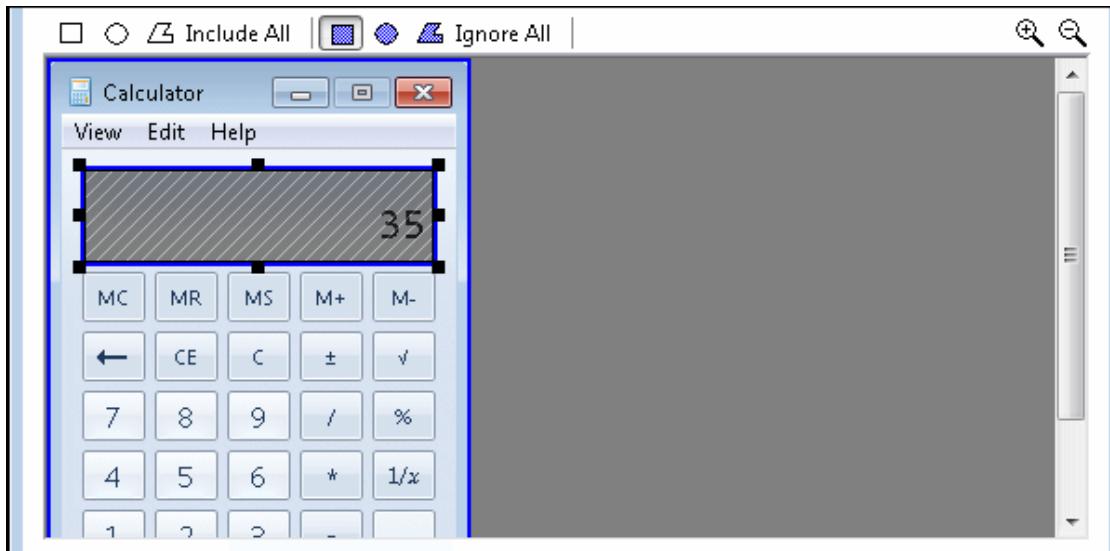
Note: (For tests and scripted components) For the purpose of this use-case scenario, you insert the bitmap checkpoint while recording your steps. However, the options are relevant also when inserting a bitmap checkpoint from the Active Screen.

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.

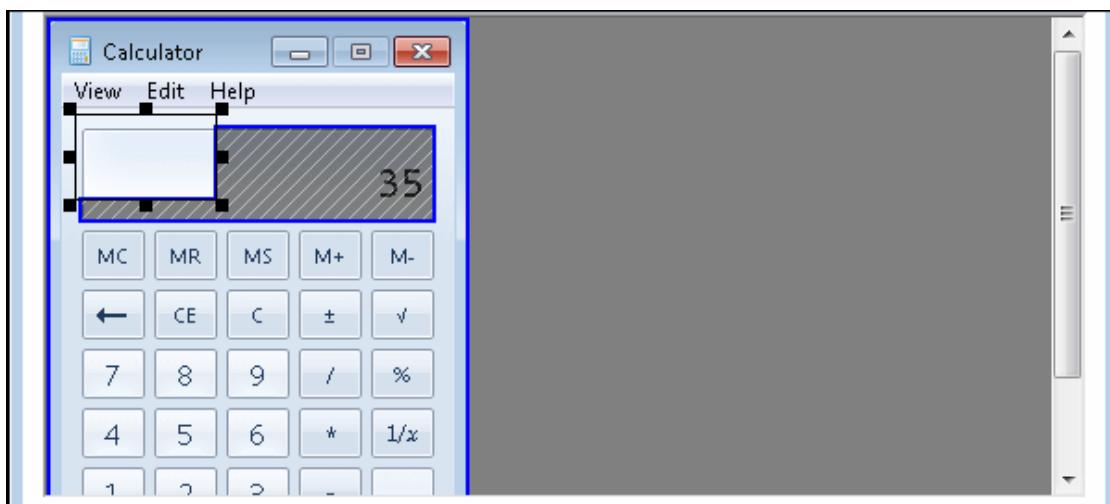
To do this, click the **Mark Rectangle to Ignore** button  in the toolbar, and then select the entire number line. Modify the size and position of your selection as needed. When you are finished, double-click to finalize the selection.

The following example shows the bitmap display area when the number line is selected to be ignored:

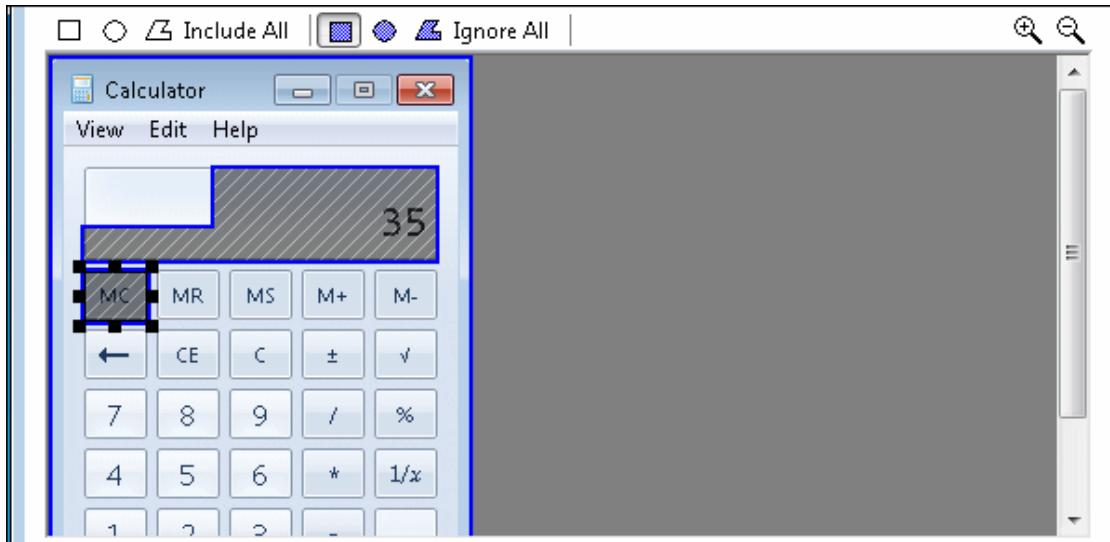


7. (Optional) If you need to modify your ignored selection after you finalize it, you can do so by clicking the **Mark Rectangle to Include** button  in the toolbar, and then selecting the areas that you want to clear. Any areas that you clear from the ignored selection are included in the comparison.

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.

Configure text recognition settings

Relevant for: GUI tests and components

Prerequisites

In your application, display the text you want to capture.

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.

Set the appropriate options

In the **Text Recognition** pane of the Options dialog box (**Tools > Options > GUI Testing** tab > **Text Recognition** node), set the following options:

OCR engine type	Select either the Abby OCR or Tesseract OCR text recognition option. 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.
Text Recognition mode	<ul style="list-style-type: none"> Single text block mode: 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: Multiple text block mode: 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:
Available languages and supported languages	<p>(For the Abby OCR engine only) From the list of selected languages, select the supported languages for text recognition.</p> <p>You can select multiple non-hieroglyphic languages (which include Chinese, Japanese, or Korean), or one of the hieroglyphic languages.</p>
Symbols for text recognition	(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.
Current language pack	<p>(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.</p> <p>You can download additional language packs from the Tesseract OCR engine download site: https://sourceforge.net/projects/tesseract-ocr-alt/files/?source=navbar. After downloading, add the files from the language packs in the <UFT installation directory>/dat/tessdata folder.</p>
Text recognition mode	Select whether you want UFT to perform with greater text recognition accuracy or better test run performance. Clear the Fast mode checkbox to run with greater accuracy.

Use configuration from a file	Instructs UFT load text recognition configuration from an externally created file. For details on creating a file, see http://www.sk-spell.sk.cx/tesseract-ocr-parameters-in-302-version .
Preprocess the image before using text recognition	Instructs UFT to process the background image before performing text recognition. This enables UFT to identify the image elements before using text recognition.

Check the text recognition settings

1. Create or open a test or component.
2. 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`

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.

Known Issues- Using Checkpoints

Relevant for: GUI tests and components

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	<ul style="list-style-type: none">The format of captured values varies depending on the specific system settings. For example, date and time values may be set to different formats. <p>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.</p> <ul style="list-style-type: none">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	<ul style="list-style-type: none">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.

XML checkpoints	<ul style="list-style-type: none">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 36: Developing Custom Comparers for Bitmap Checkpoints

Relevant for: GUI tests and components

This chapter is intended for COM programmers who want to customize the algorithm used to compare bitmaps in bitmap checkpoints.

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.

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 "[Fine-tuning the bitmap comparison](#)" on page 269.

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 - Use-case scenario](#)" on the next page.

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](#) 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 "[Install your custom comparer and register it to UFT](#) on page 303".

Perform the tutorial in "[Develop a custom comparer - Tutorial](#)" on page 309 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 "[Develop a custom comparer](#)" on page 299.

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 "[Use the bitmap checkpoint custom comparer samples](#)" on page 306.

Custom comparer for images whose location changes - 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.

Develop a custom comparer

Relevant for: GUI tests and components



Tip: To practice performing this task, see "["Develop a custom comparer - Tutorial" on page 309.](#)

Prerequisites

- Knowledge of image processing
- Experience in developing COM objects

Develop the custom comparer COM object

1. 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 305.](#)

2. Program your COM object to implement the custom comparer interfaces. For details, see "["Implement the bitmap comparer Interfaces" on the next page.](#)

Prepare the custom comparer installation

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 "["Install your custom comparer and register it to UFT" on page 303.](#)

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 .d11 using the

regsvr32.exe program. You can see an example of this type of implementation in "[Develop a custom comparer - Tutorial](#)" on page 309.

- 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 "[Use the bitmap checkpoint custom comparer samples](#)" on page 306.

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 "[Install your custom comparer and register it to UFT](#)" on page 303.

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**.

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 "Develop a custom comparer" on page 299.

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

UFT calls the **CompareBitmaps** method in the **IVerifyBitmap** interface 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** 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

UFT displays the results of bitmap checkpoints in the run results.

When you implement the **The CompareBitmaps Method** method in the **IVerifyBitmap** interface 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

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 ["Fine-tuning the bitmap comparison" on page 269](#).

To support these options, you can implement the **IBitmapCompareConfiguration** interface to provide the necessary information for the dialog box.

- Implement the **GetDefaultConfigurationString** 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).

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 "[Develop a custom comparer](#)" on page 299.

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**.

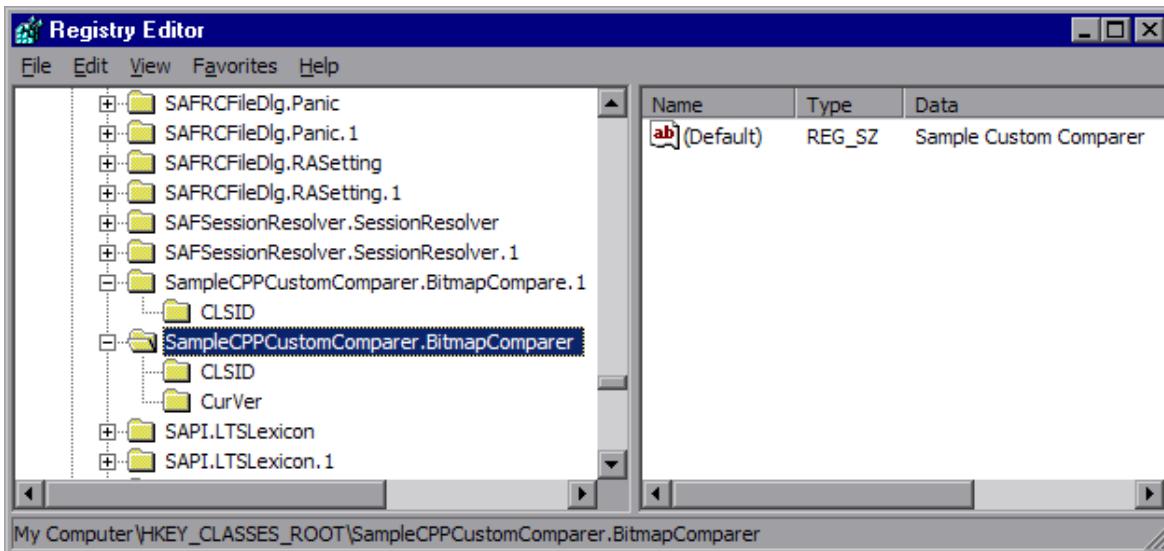
Place the custom comparer documentation in the correct location

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.

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.

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

- **If you are using the C++ sample project:**

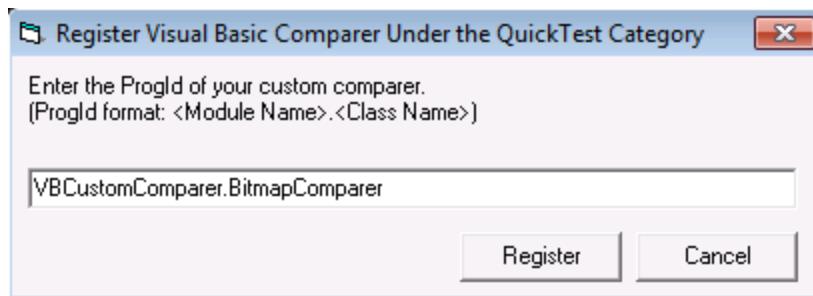
The C++ sample sources implement registering the custom comparer to UFT in the **DllRegisterServer** and **DllUnregisterServer** methods. Therefore, if you used the C++ project to create the .dll, running the .dll (in the previous step) will also register the custom comparer.

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 "[Install your custom comparer and register it to UFT](#)" on page [303](#).

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 [305](#).

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.

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 ["Use the bitmap checkpoint custom comparer samples" on page 306](#).

Note: For a task related to this tutorial, see ["Develop a custom comparer" on page 299](#).

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.

Create a new ATL project—SampleCPPCustomComparer

1. In Microsoft Visual Studio, select **New > Project**. The New Project dialog box opens.
2. Select the **ATL Project** template, enter **SampleCPPCustomComparer** in the **Name** box for the project, and click **OK**. The New ATL Project wizard opens.
3. In **Application Settings**, make sure that the **Attributed** option is not selected, and click **Finish**.

Create a new class—CBitmapComparer

1. In the class view, select the **SampleCPPCustomComparer** project, right-click, and select **Add > Class**. The Add Class dialog box opens.
2. Select **ATL Simple Object** and click **Add**. The ATL Simple Object Wizard opens.
3. 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.
4. 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 305.
5. Click **Finish**. The wizard creates the necessary files for the class that you added, including .cpp and .h files with implementation of **CBitmapComparer** class.

Implement the comparer interfaces for the CBitmapComparer class

1. In the class view, select **CBitmapComparer**, right-click, and select **Add > Implement Interface**. The Implement Interface wizard opens.
2. 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**.
3. 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***.

Move the function bodies for the comparer interface methods

1. Open the **BitmapComparer.h** and **BitmapComparer.cpp** files.
2. 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**.
3. 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)
```

Implement the bitmap checkpoint comparer interface methods

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 ["Use the bitmap checkpoint custom comparer samples" on page 306](#).

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 = _toi(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;
}
```

Design your custom comparer to register to the component category

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.

1. Add the **<UFT installation folder>\dat\BitmapCPCustomization** folder to your project's include path.
2. Open the **SampleCPPCustomComparer.cpp** file and add the following line: **#include "ComponentCategory.h"**
3. 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 = _AtlModule.DllRegisterServer();
    CComPtr 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;
}
```

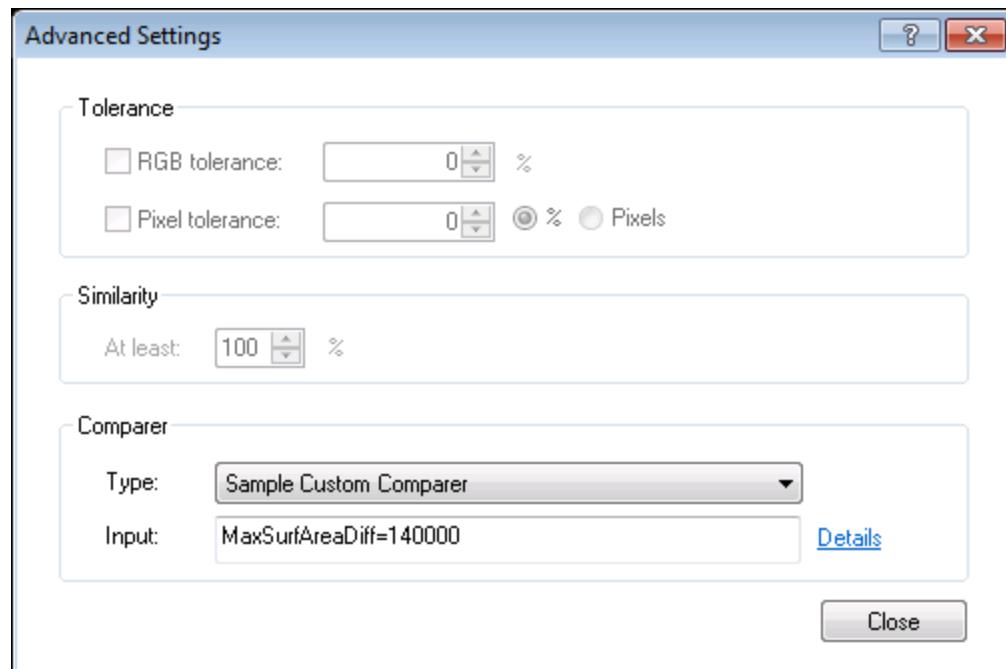
Note the second section in these methods, that handles registration to the component category for UFT bitmap comparers—**CATID_QTPBitmapComparers**.

Compile and run your DLL

Use **regsvr32.exe** to register your DLL after it is compiled. Your custom comparer can now be used in UFT for bitmap checkpoints.

Test your custom comparer

1. 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.
2. 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.
3. 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.

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.

The comparer includes the following interfaces:

IVerifyBitmap Interface

This interface contains the `CompareBitmaps` method that you need to implement to perform the bitmap comparison for the checkpoint.

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 716.

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).

IBitmapCompareConfiguration Interface

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.

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](#)" on page 301.

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" on page 302](#).

Method syntax:

```
HRESULT GetHelpFilename ([out, retval] BSTR*  
pbstrFilename);
```

Chapter 37: Output Values in GUI Testing

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 921.

Output Value types

Relevant for: GUI tests and components

You can create the following categories of output values:

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.
-------------------------------	--

File Content Output Values	<p>You can use file content output values to output the contents from any of the following file types:</p> <ul style="list-style-type: none">• HTML files• Word (.doc) files• Text files• PDF files• RTF files <p>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.</p>
Table Output Values	<p>Table output values are a subset of standard output values. 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.</p>
Text and Text Area Output Values	<p>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.</p> <p>You can use text area output values to output text strings displayed within a defined area of a screen in a Windows-based application.</p> <p>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 <code>Error</code>. If it exists, you output the text in this window (assuming that the window size is the same for all possible error messages).</p>
Database Output Values	<p>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.</p>

XML Output Values	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.
Existing Output Values	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.

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	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	<p>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.</p> <p> 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.</p> <p>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.</p>
------------------------------	---

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 916

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](#)" on the next page
- "[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 page 328
- "[Insert an existing output value step while editing \(tests and scripted components only\)](#)" on page 329
- "[Set the options for the output value object](#)" on page 329

Important information before inserting the output value step

Object visibility in application	<ul style="list-style-type: none"> During an editing session, make sure the object is visible in your application before inserting an output value step on it. For file content output values: The source file must be located on the file system.
Availability	<ul style="list-style-type: none"> Recording sessions Editing sessions Active Screen
Active Screen	Make sure that the Active Screen contains sufficient data for the object for which you want to define an output value.
Important Information	<ul style="list-style-type: none"> Output values can be viewed in the following modes: <ul style="list-style-type: none"> Simple Mode: Displays only the basic properties and expected values of the output value. Advanced Mode: Displays all supported properties and expected values of the output value. For text/text area checkpoints: <ul style="list-style-type: none"> 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 Text Recognition pane of the Options dialog box (Tools > Options > GUI Testing tab > Text Recognition node). 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. 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 "Text recognition in run-time" on page 275.

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.



Example: 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 types" on page 322](#), 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.

Chapter 38: Parameterizing Object Values

Relevant for: GUI tests and scripted GUI components

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.

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 206.

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 340.
- **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 the next page.
- **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 335.
- **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 337.
- 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 338.

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.

**Example:**

- 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.

- 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.

Data								
D5								
	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								

Global \ Login \ FlightFinder \ SelectFlight \ BookFlight

Output Errors Tasks Active Screen Data

In this example, UFT submits a separate query for each itinerary when you run the test.

The screenshot shows the UFT interface with two dropdown menus and a data table:

- Departing From:** A dropdown menu listing cities: Acapulco, Frankfurt, London, New York, Paris, Portland, San Francisco, Seattle, Sydney, Zurich. "New York" is selected.
- Arriving In:** A dropdown menu listing cities: Acapulco, Frankfurt, London, New York, Paris, Portland, San Francisco, Seattle, Sydney, Zurich. "San Francisco" is selected.
- Data Table:** A grid with columns: departure, arrival, fromMonth, toMonth. The data is as follows:

	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								

Below the table, the navigation bar shows: Global > Login > FlightFinder > SelectFlight > BookFlight. The tabs at the bottom are: Output, Errors, Tasks, Active Screen, Data.

Global and local (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**.

Global data table parameters	<p>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.</p> <p>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.</p> <p>You can use the parameters defined in the Global data sheet in any action.</p> <p>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 in GUI Testing" on page 322.</p>
Local data table parameters	<p>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.</p> <p>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.</p>

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.

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.



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:

Built-in environment variables	<p>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.</p> <p>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.</p> <p>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.</p> <p>You can also select built-in environment variables when parameterizing values.</p> <p>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 <i>HP Unified Functional Testing Add-ins Guide</i>.</p>
User-Defined Internal Environment Variables	<p>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.</p> <p>You can create or modify internal, user-defined environment variables for your test or component in the:</p> <ul style="list-style-type: none">• Environment pane of the Test or Component Settings dialog box.• Parameter Options dialog box. <p>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.</p>

User-Defined External Environment Variables	<p>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.</p> <p>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.</p> <p>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.</p> <p>For details on creating and using user-defined external environment variable files see, ""Use user-defined external environment variables" on page 355.</p>
--	--

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.

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

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.

Parameterize values for operations

Relevant for: GUI tests and scripted GUI components

Parameterize a value for an operation

1. Do one of the following:
 - For *test step operations*, in the Keyword View click in the **Value** column of the required step.
 - For *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. Open the Value Configuration Options dialog by: .
 - For a value for an operation, in the Keyword View, click the parameterization icon  for the value that you want to parameterize.
 - For 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 .
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 a constant value

Use the Data Driver Dialog Box (**Tools > Data Driver**).

Enter parameters as values in the Editor

You can enter input and output parameters as values in the Editor using the **Parameter** utility object.

Parameterize a checkpoint property value

Relevant for: GUI tests and scripted GUI components

Parameterize a value for a property in a checkpoint

1. Open the dialog box for the checkpoint properties:
 - 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 a constant value

Select **Tools > Data Driver**.

Enter parameters as values in the Editor

You can enter input and output parameters as values in the Editor using the **Parameter** utility object.

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:

The test	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. Test parameters are created in the Parameters tab of the Properties pane.
-----------------	--

The action Action parameters enable you specify parameters that are available for all steps contained in the action.

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:

- A test parameter
- The parameters of a parent action (if the action is nested)
- The output of a previous action call (for sibling actions)

You create parameters in the **Parameters** tab of the Properties pane or the **Parameters** tab of the Action Properties dialog box. You set the actual value of the action parameter in the **Parameter Values** tab of the Action Call Properties dialog box.

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.

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 task details on using test and action parameters, see ["Use action parameters" on page 346](#).

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.

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 Variable	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.
--	---

Sharing Values Using the Dictionary Object	<p>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.</p> <p>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".</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"><p style="margin: 0;">HKEY_CURRENT_USER\Software\Mercury Interactive\QuickTest Professional\MicTest\ReservedObjects\GlobalDictionary</p></div> <p>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.</p> <p>For more information on the Dictionary object, see the VBScript Reference documentation (Help > HP Unified Functional Testing Help > VBScript Reference > Script Runtime).</p>
Sharing Values Using Output Values	<p> Example:</p> <p>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:</p> <pre>GlobalDictionary.RemoveAll</pre> <p>Then you can retrieve the date from the PurchaseTickets action as follows:</p> <pre>Dim CompareDate CompareDate=GlobalDictionary("DateCheck")</pre> <p>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.</p>

Adding action parameters as 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:

Example:

- Parameter("ParameterName")
- Parameter("ActionName", "ParameterName")
- 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):

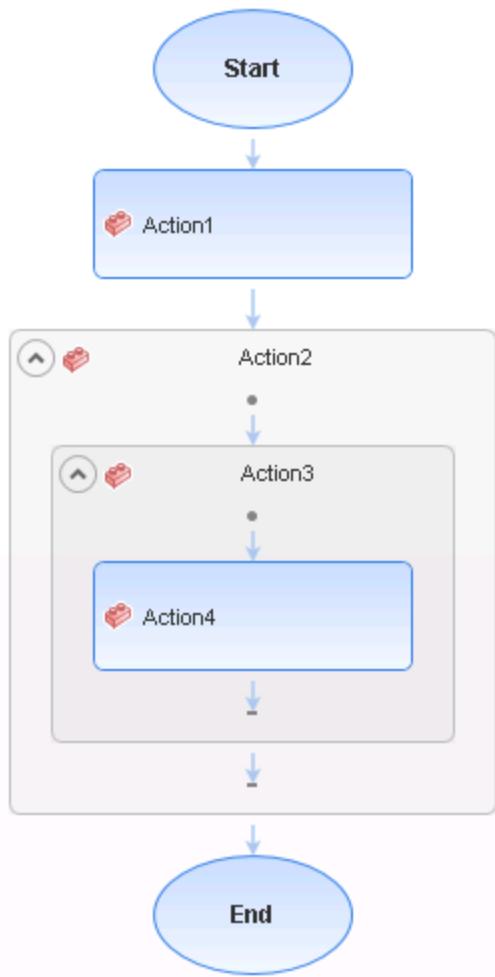
```
Browser("Mercury").Page("List Of Sales").WebEdit("Sold to").Set Parameter  
("SoldToCode")  
Browser("Mercury").Page("List Of Sales").WebEdit("Materials").Set Parameter  
("MaterialCode")  
Browser("Mercury").Page("List Of Sales").WebButton("Enter").Click  
NumTableRows = Browser("Mercury").Page("List Of Sales").WebTable  
("Orders").RowCount  
Parameter("TotalValue") = Browser("Mercury").Page("List Of Sales").WebTable  
("Orders").GetCellData(NumTableRows, "Total")
```

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 340.



Note: You can also modify many of the action properties from the Properties pane.

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 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
---------------	--------------------	-------------

Global sheet	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>If you save your test in ALM or use test configurations, 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>
Action sheet	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>
Component sheet	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).

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	<ul style="list-style-type: none"> 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 <i>UFT Object Model Reference for GUI Testing</i> .
When to save a run-time data table in ALM	<p>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.</p> <p>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.</p>

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".

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
1	TRUE
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.

Define and manage data tables

Relevant for: GUI tests and scripted GUI components

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 information

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. 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.
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:

For actions	<ol style="list-style-type: none">1. In the canvas, right-click an action and select Action Call Properties. The Action Call Properties dialog box opens.2. In the Run tab of the Action Call Properties dialog box, select the appropriate option:<ul style="list-style-type: none">• Run one iteration only• Run on all rows - UFT runs an iteration for each row in the action's data sheet• Run from row <X> to row <X> - UFT runs an iteration for each of the defined rows
For tests	<ol style="list-style-type: none">1. Select File > Settings to open the Settings dialog box.2. In the Run pane, select the appropriate option:<ul style="list-style-type: none">• Run one iteration only• Run on all rows - UFT runs an iteration for each row in the action's data sheet• Run from row <X> to row <X> - UFT runs an iteration for each of the defined rows

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.

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.

Insert formulas into data tables for use in checkpoints

Relevant for: GUI tests and scripted GUI components

1. Select the object or text for which you want to create a checkpoint and open the Insert Checkpoint dialog box as described in "[Insert a checkpoint step](#)" on [page 280](#).
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.

Import data 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.

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.

Use user-defined external environment variables

Relevant for: GUI tests only

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 the next page.

2. Upload the environment resource file to ALM

- 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.
- 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.

Create an external environment variables file

Relevant for: GUI tests only

Note: This task is part of a higher-level task. For details, see "[Use user-](#)

"defined external environment variables" above.

Create an external environment variable file in the Test Settings

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:**<UFT installation folder>\help\QTEnvironment.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.

```
<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>
```

```
</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.

Regular expressions

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**.



Example:

- 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:

- Defining the property values of an object in dialog boxes or in programmatic descriptions used within a function library
- Defining expected values for checkpoints in tests
- Defining pop-up window conditions in a recovery scenario
- 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](#)" below.

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:

Backslash Character (\)	<p>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.</p> <p>If the backslash character is not used for either of these purposes, it is ignored.</p> <p>For example:</p> <ul style="list-style-type: none"> • w matches the character w • \w is a special character that matches any word character including underscore • \\ matches the literal character \ • \c matches the literal character c • one\two matches the string onetwo <p>For example, if you were looking for a Web site calle 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 newtours\.demoaut\com</p>
Matching Any Single Character (.)	<p>A period (.) instructs UFT to search for any single character (except for \n).</p> <p>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.</p> <p>To match any single character including \n, enter (.\n)</p>
Matching Any Single Character in a List ([xy])	<p>Square brackets instruct UFT to search for any single character within a list of characters.</p> <p>For example, to search for the date 1967, 1968, or 1969, enter 196 [789]</p>

Matching Any Single Character Not in a List (^xy)	<p>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.</p> <p>For example [^ab] matches any character except a or b</p> <p>The caret has this special meaning only when it is displayed first within the brackets.</p>
Matching Any Single Character within a Range ([x-y])	<p>To match a single character within a range, you can use square brackets ([]) with the hyphen (-) character.</p> <p>For instance, to match any year in the 1960s, enter 196[0-9]</p> <p>A hyphen does not signify a range if it is displayed as the first or last character within brackets, or after a caret (^).</p> <p>For example, [-a-z] matches a hyphen or any lowercase letter.</p> <p>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.</p>
Matching Zero or More Specific Characters (*)	<p>An asterisk (*) instructs UFT to match zero or more occurrences of the preceding character.</p> <p>For example, ca*r matches car, caaaaaar, and cr.</p>
Matching One or More Specific Characters (+)	<p>A plus sign (+) instructs UFT to match one or more occurrences of the preceding character.</p> <p>For example ca+r matches car and caaaaaar, but not cr</p>
Grouping Regular Expressions (())	<p>Parentheses (()) instruct UFT to treat the contained sequence as a unit, just as in mathematics and programming languages.</p> <p>Using groups is especially useful for delimiting the arguments to an alternation operator () or a repetition operator: (*, +, ?, { })</p>
Matching One of Several Regular Expressions ()	<p>Matching One of Several Regular Expressions ()</p> <p>A vertical line () instructs UFT to match one of a choice of expressions.</p> <p>For example, foobar causes UFT to match either foo or bar. By contrast, fo(b)ar causes UFT to match either foor or fobar</p>

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. <ul style="list-style-type: none"> • \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 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 (<u>_</u>). 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 (<u>_</u>). 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	<p>You can combine regular expression operators in a single expression to achieve the exact search criteria you need.</p> <p>For example, you can combine the ‘!’ and ‘**’ characters to find zero or more occurrences of any character (except \n).</p> <p>For example, start.* matches start, started, starting, starter</p> <p>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]*</p> <p>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.</p> <p>This regular expression matches any number between 0 and 1200: ([0-9]?[0-9]?[0-9]1[01][0-9][0-9])1200</p>

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**).

Chapter 39: Value Configuration and Parameterization

Relevant for: GUI tests and 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.

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 330](#).

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
Local parameter	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

	<p>Options Dialog Box and the Parameters Tab of the Properties Pane.</p> <p>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.</p> <p>You cannot delete local parameters, but you can cancel the input or output to them.</p>
Component parameter	<p>Variable values defined within a component for use in the same component or later components in the business process test.</p> <p>Component input parameter values can be received and used as the values for specific, parameterized steps in the component.</p> <p>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.</p> <p>You define component input and output parameters in the Parameters pane of the Component Settings dialog box, or in the ALM Business Components module.</p> <p>Additionally, if you are working with a business process test or flow, you can define parameters in the Properties pane.</p>

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 "[Parameterize input values](#) on the next page".

For instructions on how to parameterize output values, see "[Create or modify an output value step](#) on page 326".

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:

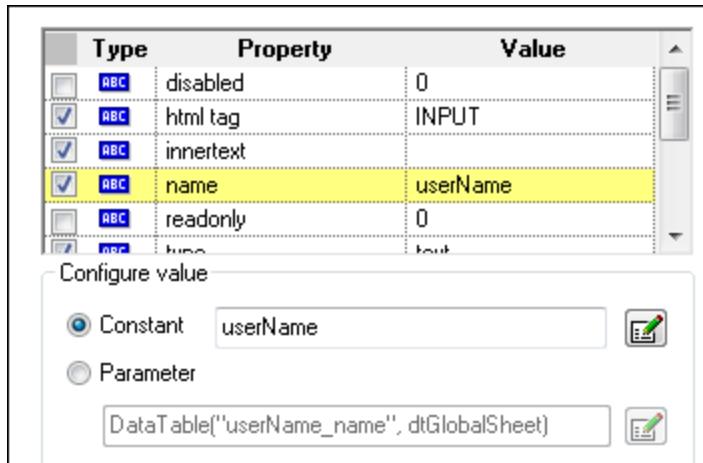
Do one of the following:

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:

Arguments:			
Name	Type	Value	
x	Integer	0	

2. Above the **Configure value** area of a dialog box, select a property or argument:



Then, in the **Configure value** area, select the **Constant** or **Parameter** radio button and click the **Constant Value Options** OR **Parameter Options** button

Parameterize input values

Relevant for: Keyword GUI components

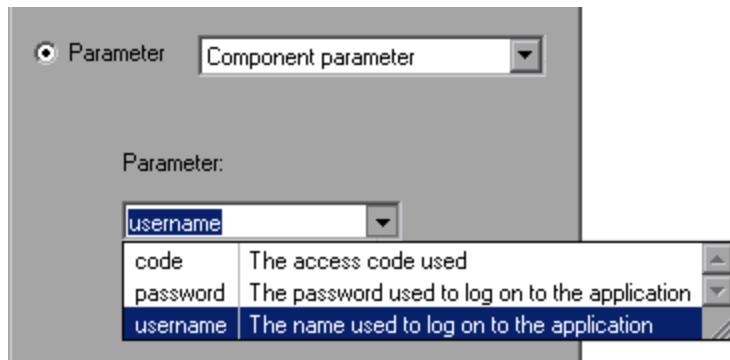
This task describes how to parameterize input values for a step using a local parameter or a component parameter.

1. Do one of the following:

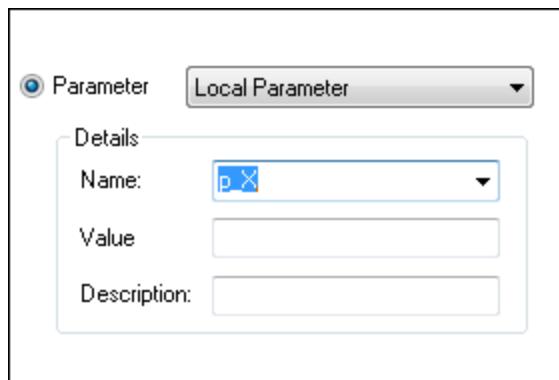
- In the Keyword View, click the parameterization button for a selected value.
- In the Checkpoint Properties Dialog Box, click the **Parameter Options** button
- In the Parameterization / Properties Dialog Box (Checkpoints), select the **Parameter** radio button. If a parameter is already defined, you must then click the **Parameter Options** button to open this dialog box.

2. 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:



- **Local parameter.** The details for the local parameter type are displayed.



3. 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.

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 Test Design

Chapter 40: API Test Creation 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 UFTAPI 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.

UFT provides a number of standard activities that test common application processes, including:

Standard Activities, such as	<ul style="list-style-type: none">• Flow Control activities, such as Wait, Break, and Conditional steps. These activities enable you to customize your test to match any special application workflows.• String Manipulation activities, such as Concatenate Strings and Replace String.• File system activities, which enable you to test your application's interaction with the file system.• Database activities, which enable you to test your application's ability to connect and communicate with a database.• FTP activities, which enable you to test your application's ability to perform FTP-related procedures.• Network activities, such as HTTP Request and SOAP Request, which enable you to test your application's connection with a network or web-based server.• JSON and XML activities, which enable you to test your application's ability to convert XML and JSON data to text and vice versa.• Math and Date/Time activities, which enable you to perform math operations or perform tasks using the system date and time.• Other Miscellaneous activities, including Custom Code activities, Run and End program activities, and a Report activity.
-------------------------------------	--

Technology-specific activities, such as:	<ul style="list-style-type: none"> A Call Java Class activity, which enables you to test Java-based processes that run in your application JMS (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. IBM WebSphere MQ 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. SAP activities, which enable you to test your application's ability to communicate with an SAP server using IDOCS and RFCs. Load Testing activities, which enable you to add steps for your test to be run as a load test in HP LoadRunner. HP Automated Testing Tools 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.
Custom activities, based on your services, such as:	<ul style="list-style-type: none"> Web Service activities imported from a WSDL file. For details, see "Import a WSDL-based Web service" on page 421. Web Application activities imported from a WADL file. For details, see "Import a Web Application service" on page 432, REST Service activities created in UFT using the Add REST Service dialog box or by importing a Swagger REST API. For details, see "Create a REST service model" on page 424. Network Capture activities imported from a Network Capture file

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 "[Event Handlers for API Test Steps](#)" on page 616.
- Create custom test activities with the UFT Activity Wizard (**Start > All Programs > HP Software > HP Unified 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 507.

APItesting integrates with the following products:

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.
Service Virtualization	In order to mimic services that your application may use, UFT integrates with Service Virtualization. After creating your service models in Service Virtualization, you then run them as a service for your test.
Other tools	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 " Call external tests or actions " on page 400.

Automatically generating API tests

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">• Positive Testing: 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).• Standard Compliance: Checks the service compliance with industry standards such as WS-I and SOAP.• Security Testing. Tests the security of the service. You can select one of the following types of security testing:<ul style="list-style-type: none">• SQL Injection Vulnerability. This checks if the service is vulnerable to SQL injections by injecting SQL statements and errors into relevant parameters.• Cross-site Scripting (XSS). Attempts to hack the service by injecting code with relevant parameters that will disrupt its functionality.• Boundary Testing. 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">• Extreme Values: Provides invalid data types to the services and verifies they are not accepted.• Null Values. Provides NULL parameters to the components to verify they are not accepted.
SOAPUI to Test Converter	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 "[Automatically generate API tests](#)" on page 383.

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.

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 [98](#).
- Setting up **test-specific preferences**, including global properties, test input and output properties and parameters, or user profiles and variables.
For details, see "[Define API test properties or user/system variables](#)" on page [461](#).
- 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 "[Import a WSDL-based Web service](#)" on page [421](#) or "[Import a Web Application service](#)" on page [432](#).

- **Creating REST service resources and methods** if your application is based upon REST services.
For details, see "[Create a REST service model](#)" on page 424.
- **Importing .NET assemblies** referenced by your test.
For details, see "[Import and create a .NET Assembly API test step](#)" on page 436.
- 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:

Create additional test flow steps- optional	Expand the Toolbox pane nodes and drag Flow Control activities onto the canvas: <ul style="list-style-type: none">• 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.
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.

Provide the step properties	<p>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 386.</p> <p>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.</p>
------------------------------------	---

Enhance your test steps

To enhance your test steps, do any of the following:

Define data sources for the test	For details, see " Assign data to API test/component steps " on page 454.
Create a Custom Code activity	<ol style="list-style-type: none">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.

Add an event handle - option al	<p>For any activities, you can define default event handlers for checkpoints, and before and after step executions.</p> <ol style="list-style-type: none"> 1. Select a step in the canvas and open the Events tab  in the Properties pane. 2. In the row containing the event execution point you want (before, after,etc.), select Create a default handler. 3. Edit the code in the <code>TestUserCode.cs</code> 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. 4. To access the properties of an activity, cast it before the activity name. For example: <pre>ConcatenateStringsActivity cat = args.Activity as ConcatenateStringsActivity; args.Checkpoint.Assert.Equals(cat.Prefix+cat.Suffix, cat.Result);</pre> <ol style="list-style-type: none"> 5. Click File > Save All to save the <code>TestUserCode.cs</code> file and the test. For details and examples, see "Event Handlers for API Test Steps" on page 616.
--	---

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.	<p>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.</p> <p>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.</p>
Run your test to debug it.	<p>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.</p> <p>For general details on debugging, see "Debugging Tests and Components" on page 749.</p>

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.
--------------------------------	---

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.

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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• 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.

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 "[Import a WSDL-based Web service](#)" on page 421.

You can also create the following REST Service resources and methods:

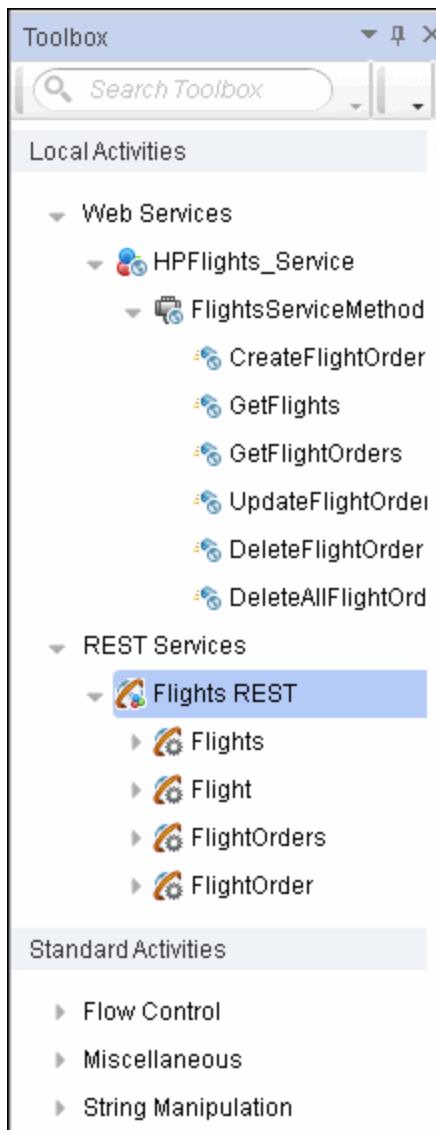
- **Flights Get**
- **Flight Get**
- **FlightOrders Get**
- **FlightOrders ReserveOrder**
- **FlightOrder Get**
- **FlightOrder Update**
- **FlightOrder Delete**
- **FlightOrder DeleteAll**

For details on how to create REST Service methods, see "[Create a REST service model](#)" on page 424.

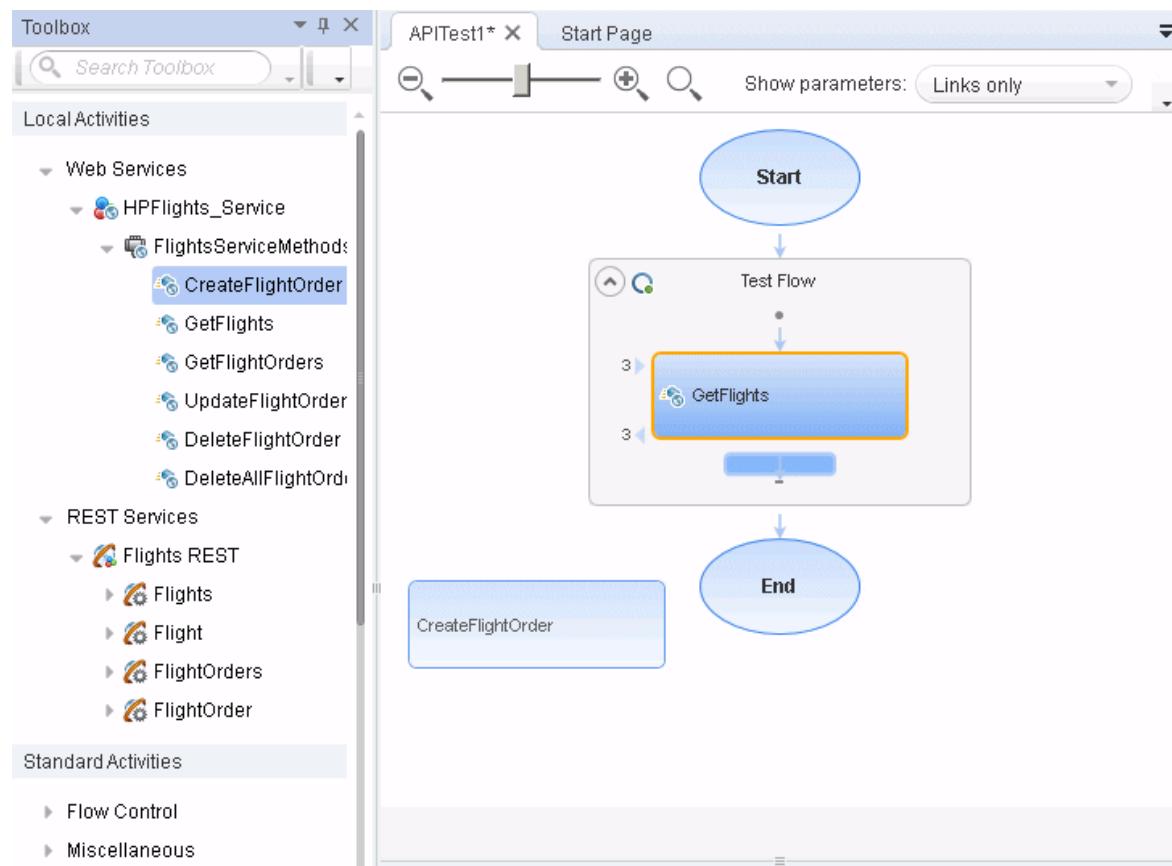
Create your test steps

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.

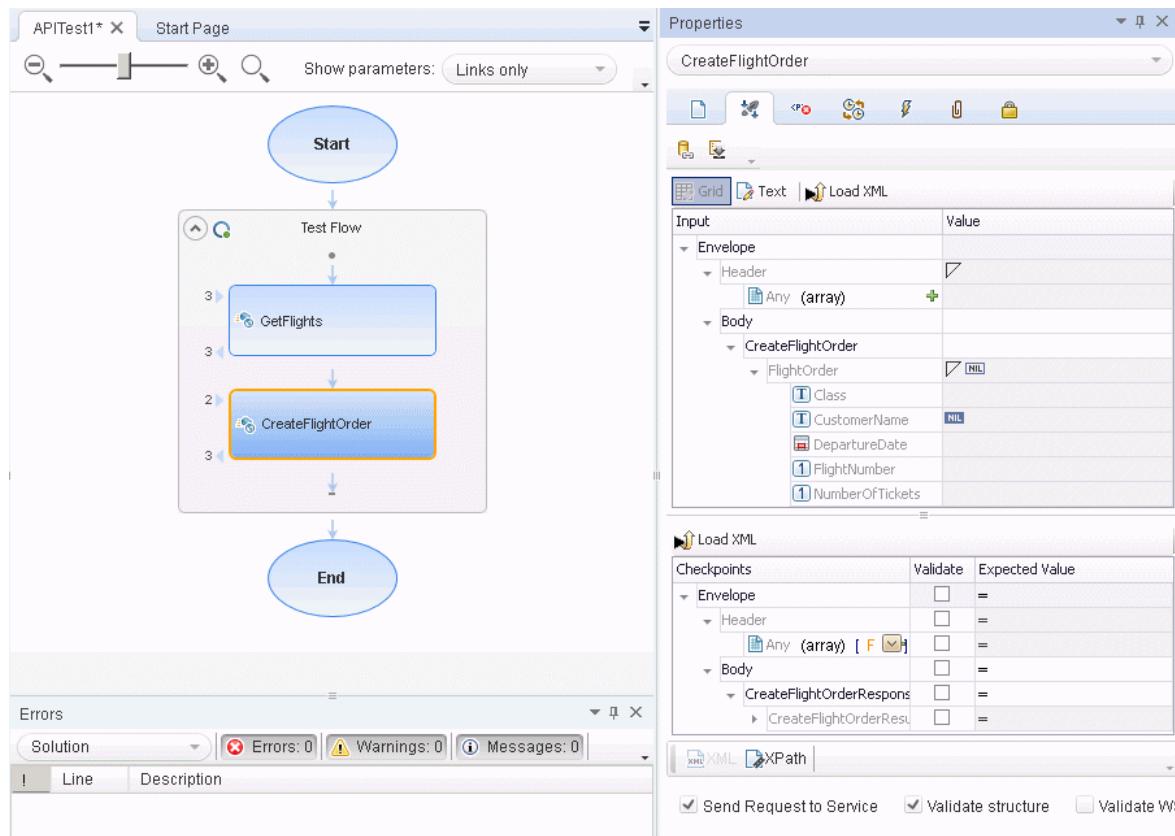
The activities that are available for your test steps are stored in the Toolbox pane. This includes custom methods that were imported or created for your test:



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 "[Create an API test](#)" on page 372.

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.

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.

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 "[Create an API test](#)" on page 372.

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.
If needed, enter the authentication and proxy settings if you are importing the WSDL from a URL location.
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:

Positive Testing	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).
Standard Compliance	Checks the service compliance with industry standards such as WS-I and SOAP.
Security Testing - SQL Injection Vulnerability	Checks if the service is vulnerable to SQL injections by injecting SQL statements and errors into relevant parameters.
Security Testing - Cross-site Scripts (XSS)	Attempts to hack the service by injecting code with relevant parameters that will disrupt its functionality.
Boundary Testing - Extreme Values	Provides invalid data types to the services and verifies they are not accepted.
Boundary Testing - Null Values	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.

1. 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**).
2. In the SOAPUI to API Test Converter window, navigate to the directory in which your SOAPUI test was saved.
3. 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
/source	The absolute path to the soapUI file with an .xml extension, to be converted.
/destination	The absolute path of the folder to where the created API tests will be written.
/logs (optional)	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.
-? or /?	Show the parameters and their usage. For example: soapUI2APITestCMD.exe -?

Chapter 41: Standard Activities

Relevant for: API testing only

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 the following categories:

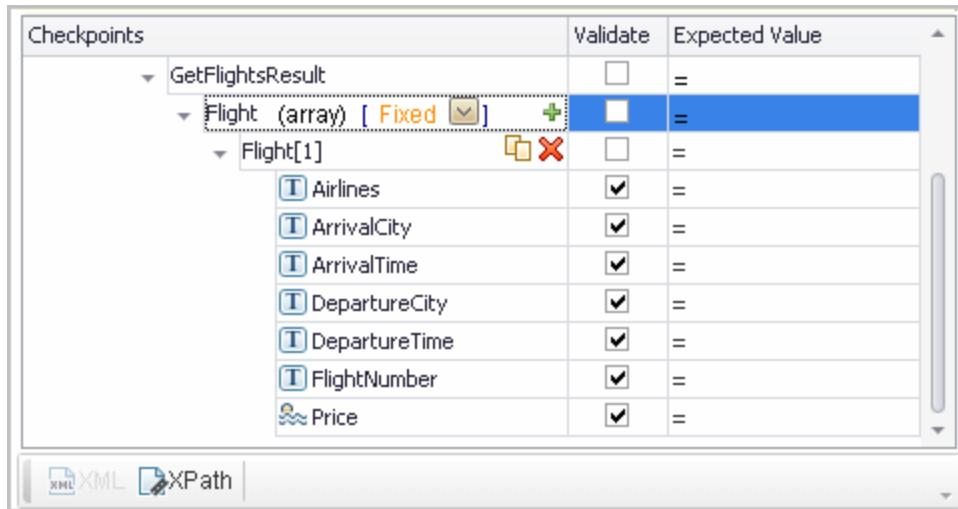
Standard activities	This category includes the built-in activities, such as String Manipulation , Database , Network , File System , as well as new custom activities you create using the extensibility API provided with the product.
Local activities	This category includes the activities that are stored as part of the test or business component. These can be imported services such as Web or REST service operations, and .NET assemblies.
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.

-  See also:
- ["Custom Activities" on page 411](#)
 - ["API Testing Extensibility" on page 507](#)
 - ["API Test Creation Overview" on page 368](#)

Checkpoint validation

Relevant for: API testing only

When creating a test, it is helpful to confirm that the application performed its activities as expected. An application's response can contain several properties. Use **checkpoints** to define the expected values of the properties.



Enter checkpoint 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.

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.

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. Specify a fully qualified XPath expression or instruct UFT to ignore the namespaces and prefixes during the test run.

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, 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.

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 "[Set XPath checkpoints](#)" on page 393.

Test with Docker activities

Relevant for: API testing

Use UFT's native API testing capabilities to test your applications stored in remote Docker containers.

Docker activities allow UFT to manage Docker, download images from the Docker registry, and run containers based on those images. Docker activities are available in the Toolbox pane under the **Docker** node.

UFT uses applications packed with Docker as follows, accessing both the image and the container:

1. UFT first sends a request to Docker to download (pull) the image from the Docker registry.
If you have a special configuration for how you want a Docker container to run, create a container and load a JSON file with the configuration.
2. Docker starts the container based on the downloaded image.
3. While the application is running, UFT performs additional test steps on the application.
4. When the test run is complete, UFT sends a request to Docker with a request to stop the container.

For general information about Docker, see the Docker documentation.

Configure ports

Before you test, configure ports in the **Run Image** or **Create Container** activities to map the container port to the Docker host port.

This enables applications to access the mapped port inside the container.

Update the properties as follows:

Run Image activity	Configure the port in the Port Bindings property in the Properties pane. For details, see " Run an image " on the next page.
Create Container activity	Specify the port bindings inside the JSON body of the request and then load it into the activity. For details, see " Create a container " below.



Tip: To make the port accessible from outside a container, use the '**EXPOSE**' parameter in the Docker file of the image.

Pull an image from the Docker registry

1. In the Toolbox pane, from the **Docker** node, add a **Pull Image** activity to the canvas.
2. In the **Input/Checkpoints** tab of the Properties pane, enter the access details for the Docker container:
 - **URL** of the Docker server in which the Docker image is stored
 - **Image** to pull from the Docker hub on the Docker server
 - The **tag** used to identify the image - optional
3. If necessary, add checkpoints to validate the activity:
 - **Digest:** specify the digest identifier of the image to ensure that you imported the correct image
 - **Status:** the status of the image import operations

Create a container

If you have special custom parameters or need advanced configuration for your container, create your own custom container.

This enables you to configure the container creation using any of the parameters provided by Docker.

1. Ensure that the Docker image already exists on the Docker server.
2. In the Toolbox pane, from the **Docker** node, add a **Create Container** activity to the canvas.
3. In the **Input/Checkpoints** tab of the Properties pane, provide the container details:

- **Server URL** of the Docker server
 - **Container name** for the created container. By default this argument should remain empty, as Docker generates the value automatically
4. In the **Advanced Properties** tab, click the **Load JSON** button.
The **.json** file body is loaded with a REST request with parameters, ready to be used when the container is created.

Run a container

1. In the Toolbox pane, from the Docker node, add a **Start Container** activity to the canvas.
2. In the **Input/Checkpoints** tab  of the Properties pane, provide the container details:
 - **Server URL** of the Docker server
 - **Container ID** of the container to start

 **Tip:** Parameterize this value by linking this property to the **Container ID** property of a **Create Container** activity.

Run an image

After you have downloaded the image of your application to the Docker server, you can run an image of the application on the container.

Running the image starts the image in its own container, and starts the specified application using the command provided.

1. In the Toolbox pane, from the Docker node, add a **Run Image** activity to the canvas.
2. In the **Input/Checkpoints** tab  of the Properties pane, provide the access details for the Docker container:
 - **URL** of the Docker server
 - **Image** to run from the Docker hub on the Docker server
 - The **Command** used to run the application inside the Docker container
3. Below the main input properties, in the **Port Bindings** row, click the **Add** button  to add a port binding array to the step.
4. In the array, provide the following ports:
 - **Container port:** the port inside the container that receives the output from the application
 - **Host port:** the port on the Docker host machine

Once these ports are mapped to one another, all data that arrives at the container port is forwarded to the host port.

Add additional test steps

Once the container is started, the application is available for testing.

From the Toolbox pane, expand any necessary nodes and add activities to the canvas.

Stop the Docker container image

When you finish running the test of the application, stop the container that is currently running.

1. In the Toolbox pane, from the Docker node, add a **Stop Container** step to the canvas.
2. In the **Input/Checkpoints** tab  of the Properties pane, set the properties used to identify the container you want to stop:
 - **URL** of the Docker server
 - **Container's ID** to stop
 - The **Time** to delay (in seconds) before stopping the container

Set array checkpoints

Relevant for: API testing only

Enable active content on your computer

1. In Internet Explorer, select **Tools > Options**.
2. Select the **Advanced** tab.
3. Enable the option **Allow active content to run in files on My Computer** in the **Security** section.
4. Click **OK** and close the browser.

Add a step with an array output

Add a test step with output properties in the form of an array.

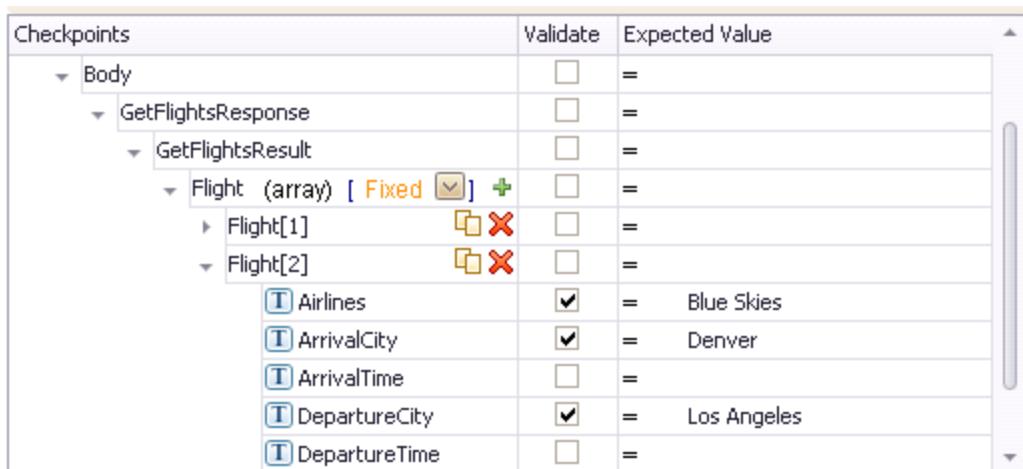
Select an array validation method

1. In the Properties pane, select the **Input/Checkpoints** tab .
2. In the Checkpoints section of the Input/Checkpoints tab, expand the drop down adjacent to the name of the parent array node.
3. In the cell for the parent node of the array, 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.

Validate individual array elements

1. Below the parent node of the array, click the plus button .
2. In the row for the array element to validate, select the **Validate** box:



The screenshot shows the Checkpoints pane with the following structure and data:

Checkpoints	Validate	Expected Value	
Body	<input type="checkbox"/>	=	
GetFlightsResponse	<input type="checkbox"/>	=	
GetFlightsResult	<input type="checkbox"/>	=	
Flight (array) [Fixed] +	<input type="checkbox"/>	=	
Flight[1]	 	<input type="checkbox"/>	=
Flight[2]	 	<input 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"/>	=	

3. Provide validation values for the array elements.

Validate the element count - optional

In the **Expected Value** column of the parent row of the array, enter a desired count number and the evaluation expression, such as `=`, `>`, and so forth.

Set XPath checkpoints

Relevant for: API testing only

This task describes how to validate your test results against XPath expressions.

Add a step with XML output

1. From the Toolbox pane, add a test step with XML output, such as **String to XML**.
2. In the Properties pane, select the **Input/Checkpoints** tab .
3. 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.

Set the namespace setting

In the bottom of the Input/Checkpoints tab, 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.

Add an XPath checkpoint

1. In the XPath tab, click the **Add** button .
2. Type or paste the expression into the **XPath Checkpoints** column:
 - 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.
3. Select the check box in the **Validate** column, select a comparison operator, and provide an expected value.

Use Flow Control activities

Relevant for: API testing only

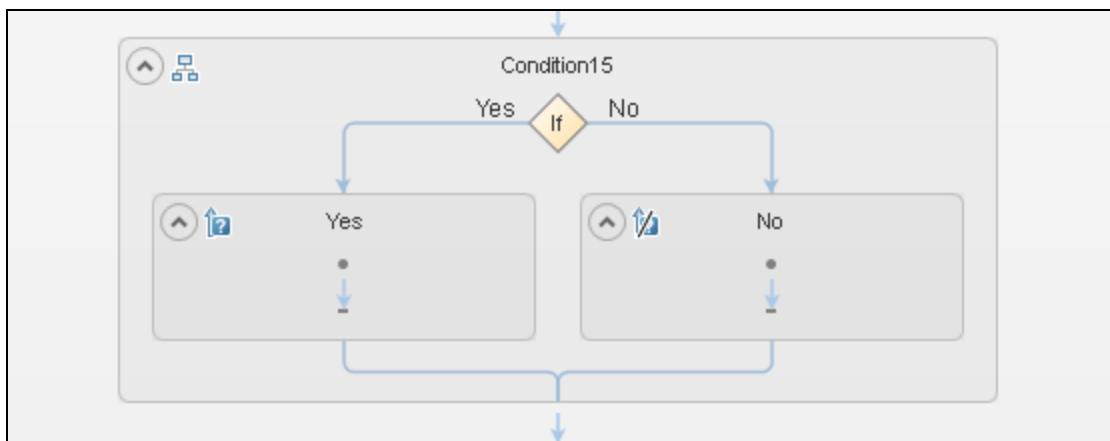
Add a conditional step

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:



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:

Specify a condition for the output of a test step	<ol style="list-style-type: none">In the Condition tab  in the Properties pane, select the Use condition option. The condition properties are displayed.In the Variable field, click the Link to data source button .In the Select Link Source dialog box, link to the output of a previous step. You must link the Variable field to a previous step to set the value to meet the trigger condition.Specify the expected Value and the Operator for the step selected in the Variable field.
--	---

Use an event handler	<ol style="list-style-type: none"> In the Condition tab  in the Properties pane, select the Use event option. In the Events tab , in the Condition row, click the drop-down arrow and select Create a default handler. The <code>TestUserCode.cs</code> file opens in the document pane. In the IfElse<#>_OnCondition section of the <code>TestUserCode.cs</code> file, replace the TODO section with your event handler code: <pre> 34: /// <summary> 35: /// Handler for the IfElse9 Activity's Condition event. 36: /// </summary> 37: /// <param name="sender">The activity object that raised the event. 38: /// <param name="args">The event arguments passed to the event. 39: /// Use this.IfElse9 to access the IfElse9 Activity's condition. 40: public bool IfElse9_OnCondition(object sender, STAActivityEventArgs args) 41: { 42: //TODO: Add your code here... 43: return false; 44: } 45: 46: 47: </pre>
-----------------------------	---

- 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

- In the Toolbox pane, expand the **Flow Control** activities node and drag a **Loop** step to the canvas.
- 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:

Specify a condition	i. In the Condition tab  in the Properties pane, select the Use condition option. The condition properties are displayed. ii. In the Variable field, click the Link to data source button  . iii. In the Select Link Source dialog box, link to the output of a previous step. You must link the Variable field to a previous step to set the value to meet to trigger the condition. iv. Specify the Value and the Operator for the value of the step selected in the Variable field.
Use an event handler	i. In the Condition tab  in the Properties pane, select the Use event option. ii. In the Events tab  , in the Condition row, click the drop-down arrow and select Create a default handler . The TestUserCode.cs file opens in the document pane. iii. In the Loop_OnCondition section of the TestUserCode.cs file, replace the TODO section with your event handler code.

- **'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 470.

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.

Send a multipart HTTP or REST Service request

Relevant for: API testing only

Add an HTTP or REST Service step

From the Toolbox pane, add one of the following to the canvas:

- From the **Network** activities, an **HTTP Request**
- From the **Local Activities** section, a REST service method.

Set the properties for the first part of the request

Set the request property values in the relevant tab:

- 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 

Set the properties for the other parts

1. In the Properties pane, open the **Multipart** tab 
2. Select the **Enable Multipart** option.
3. In the Input section, set the multipart type.
4. Below the Type, provide the details for the general header:

- a. Click the **Add** button  to add a header.
- b. Expand the **Headers (array)** element.
- c. In the Headers[<number>] array property, enter the **Name** and **Value** for the general header.
5. Click the **Add** button  and add the necessary number of parts in the multipart request.
6. Set the properties for the parts:
 - a. In the **Headers (array)** cell, click the **Add** button  to add an request header.
 - b. Expand the **Headers[<number>]** array
 - c. In the Headers array, enter the value for the **Name** and **Value** properties for the part request header.
 - d. 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.

Create a call to a Java class

Relevant for: API testing only

The Call Java Class activity lets you to add Java steps to your test script. This feature enables you to incorporate existing Java code into your test.

Implement the UFT API Java interface

Open the `<installation_folder>\addins\ServiceTest\JavaCall\JavaInterface\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.

Compile the Java source code

In your IDE, compile the java files located in the `<installation_`

<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.

Package your custom step - optional

Package your java classes into a **.jar** file.

Set up the Java environment - optional

1. In the canvas, select a Start or End step.
2. 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.
3. In the Toolbox pane, expand the **JMS** node in the Toolbox pane and drag a **JMS** activity onto the canvas.
4. 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**
5. If you are using a **Send** activity, specify a message.
6. 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.

Add a Call Java Class activity

In the toolbox pane, expand the **Java** node, and drag the **Call Java Class** activity onto the canvas.

Set the Java step property values

1. In the canvas, select the Java step.
2. In the Properties, open the **Input/Checkpoints** tab .
3. In the Input/Checkpoints tab, click the **Java Class** button to open the Java Class Dialog Box.

4. In the Java Class dialog box, do the following:
 - a. **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.
 - b. 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.
 - c. 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.
 - d. Click **OK** to save the Java Call settings.

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

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.



Example:

- 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.

Call an API Test or Action or Service Test test

1. Make sure the action or test you want to call has been saved and run successfully at least once.
2. In the **Standard Activities** section of the Toolbox pane, expand the **HP Automated Testing Tools** node.
3. Add a **Call API Action or Test** activity to the canvas.
4. In the **Input/Output Properties** tab in the Properties pane, click the **Select Action or Test** button.
5. In the Select Action or Test Dialog Box, select a test last modified with Service Test 11.10 or higher, or with UFT.
6. 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.

7. 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..
8. 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.
9. 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.

Call a GUI Test or QuickTest action or test

1. Make sure the action or test you want to call has been saved and run successfully at least once.

2. In the Toolbox pane, in the **Standard Activities** section, expand the **HP Automated Testing Tools** node.
3. 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.

4. 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.
5. 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.

6. 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.

Add a LoadRunner script activity

1. Make sure the action or test you want to call has been saved and run successfully at least once.
2. In the Toolbox pane, in the **Standard Activities** section, expand the **HP Automated Testing Tools** node.
3. Add a **Call Virtual User Generator Script** activity to the canvas.

4. In the Properties pane, select the General tab, and click the script selection button .
5. Navigate to the directory where your VuGen script file (**.usr**) is saved.

Prepare and run a Load test

Relevant for: API testing only

This task describes how to prepare a test for load testing in LoadRunner.

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.

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 .

Add test steps

Drag activities from the Toolbox pane onto the canvas to add steps to the test.

Prepare for load testing

To measure the performance of a group of steps, define a transaction.

Mark the beginning of a transaction	<ul style="list-style-type: none">• 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.
--	---

Mark the end of a transaction	<ul style="list-style-type: none">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. <p>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.</p>
Set the think time	<ul style="list-style-type: none">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.

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 .

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.

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.

Incorporate the test into LoadRunner

Add the test to the LoadRunner Controller console to include it in a load test.

Test Web sockets communication

Relevant for: API testing only

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. In the Web Sockets section of the Toolbox pane, add a **OpenSocket** activity to the canvas.
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. In the Web Sockets section of the Toolbox pane, add a **SendMessage** activity to the canvas.
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. In the Web Sockets section of the Toolbox pane, add a **ReceiveMessage** activity to the canvas.
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. In the Web Sockets section of the Toolbox pane, add a **OpenSocket** activity to the canvas.
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.

Known Issues- Standard Activities

Relevant for: API testing only

System activities

- **End Program.** You cannot specify **Window Title** as an input method for a windowless process, even if you are using a wildcard expression.
- **End Program.** If you are running on a 64-bit machine, the **End Program** activity will not be able to terminate 64-bit applications. However, it can terminate other 32-bit applications.

Java activities

- **Call Java Class.** Supports only Java primitive types.
- **Call Java Class.** Once you select a Java file for the call, the **Java Class** button is disabled. As a result, you cannot replace or update the Java file.
Workaround: Remove the step containing the **Call Java Class** step and add a new one using the new Java file.
- **Call Java Class.** Java code loaded by the JVM (Java Virtual Machine) cannot be modified or updated when the JVM is running.

Network

- **HTTP Request/Receiver.** Nested transactions are not supported.
Workaround: 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.
- **HTTP Request/SOAP Request.** XML files that use XSL, are not supported.
- **SOAP Request.** Switching between **Text** and **Grid** views, may cause the grid to display an element added below the Body's **Any** node, under the Header's **Any** node.
Workaround: Open the **Text** view to view the correct XML..
- **SOAP Request.** 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.
Workaround: Import the SOAP 1.2 schema for your SOAP Request step's response body. This schema is available with the UFT installation at
`<UFT installation folder>\Addins\ServiceTest\WSImportTechnology\envelop1.2.xsd.`

Database

- Nested transactions are not supported in database transaction activities.
Workaround: 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.
- Databases which are supported by ODBC, but not by OLEDB, cannot be accessed by UFT.

FTP

- **FTP:** When working with FTP activities that specify paths, you need to enter the full path.
- **FTP Download:** When downloading from Linux servers, you cannot download an empty folder.

IBM Websphere MQ

- **Get Message from MQ Queue.** The MQGMO_MARK_SKIP_BACKOUT option is not supported.
- **Put Message to MQ Queue, Publish Message to MQ Topic.** 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.
- **MQ Steps.** Automatic linking of IBM Websphere MQ steps to the most recent **Connect to MQ Queue Manager** 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.
Workaround: Manually link to a MQManager property using the Select Link Source dialog box.

JSON

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.

Load Testing

- Related data mapping in a load-enabled test is not supported for the LoadRunner parameter advance policy of **Each Occurrence**.
- Tests created as Business Process Testing (BPT) components, cannot be used

in load testing.

- If your tests use IBM's MQ client, make sure to install the MQ client on all machines running these tests.
- 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.
- The data assignment method, **Use a unique value for each Virtual User when load testing** is not supported in all environments.

Web Sockets

Using a web socket open between actions in a test is not supported.

XPath Activity Checkpoints

XPath aggregate functions are not supported.

Chapter 42: Custom Activities

Relevant for: API testing only

Custom activities enable you to create or import your service models, and then create activities for use in an API test.

Web Services

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.

After you import the WSDL, UFT represents the data differently depending on the type of Web service:

Document/Literal Web services	The Properties pane displays the input and output properties for a Web service method in the grid, enabling you to assign values to the properties.
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. As a result, 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 "[Import a WSDL-based Web service](#)" on page 421.

REST Services

To create REST service models in UFT, you have multiple options:

- Define the service's **Service**, **Resource**, and **Method** manually using the REST service editor. This model is stored as a prototype activity within your test and the test's methods are added as test steps.

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 416.

- Import a service model from a Swagger API or OData REST Service API. UFT reads the service description from the file or URL and creates the corresponding services, resources and models.

For more details, see "[Create a REST service model](#)" on page 424.

Web Application Services

Web Application services provide a description of an HTTP-based Web application in 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.

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 is 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, perform a network capture, and use the captured information as a basis for your test.

Using a network capture program, capture the network traffic for your application or Web service into a saved file and 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 more details, see "[Import a Network Capture file](#)" on page 434.

.NET Assemblies

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.

Begin by importing the .NET assembly into your test. The Toolbox pane then displays the assembly as an activity and you can 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 more details, see "[Import and create a .NET Assembly API test step](#)" on page 436.

SAP-based Services

Create additional 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 be 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 more details, see "[Create an SAP API test step](#)" on page 436.

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.

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.

Perform activity sharing

Relevant for: API testing only

This task describes how to save activities to a repository, update them, and view their properties.

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 displays the service's methods in the **Local Activities** node.

By default, the test stores these activities in its **EmbeddedJavaResources** subfolder.

4. Move the activity to a repository

In the Toolbox pane, 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.

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.

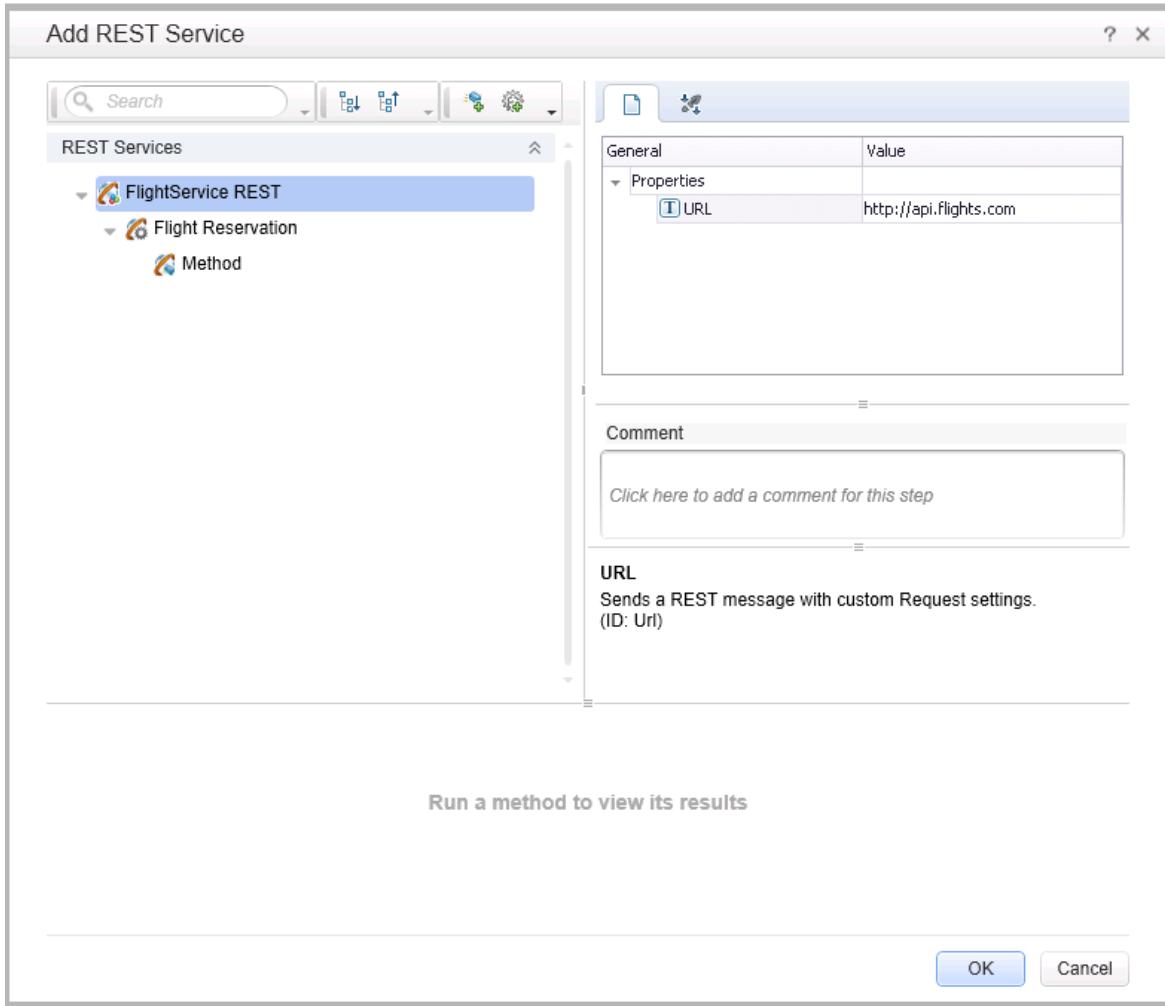
The Properties pane lets you provide values for the SOAP fault header and body. You can enter **faultcode**, **faultstring**, and **faultactor** values as well as custom properties using **Any** type parameters. Using the Checkpoint mechanism, you can validate these values and view the results in the 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 425](#).



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.



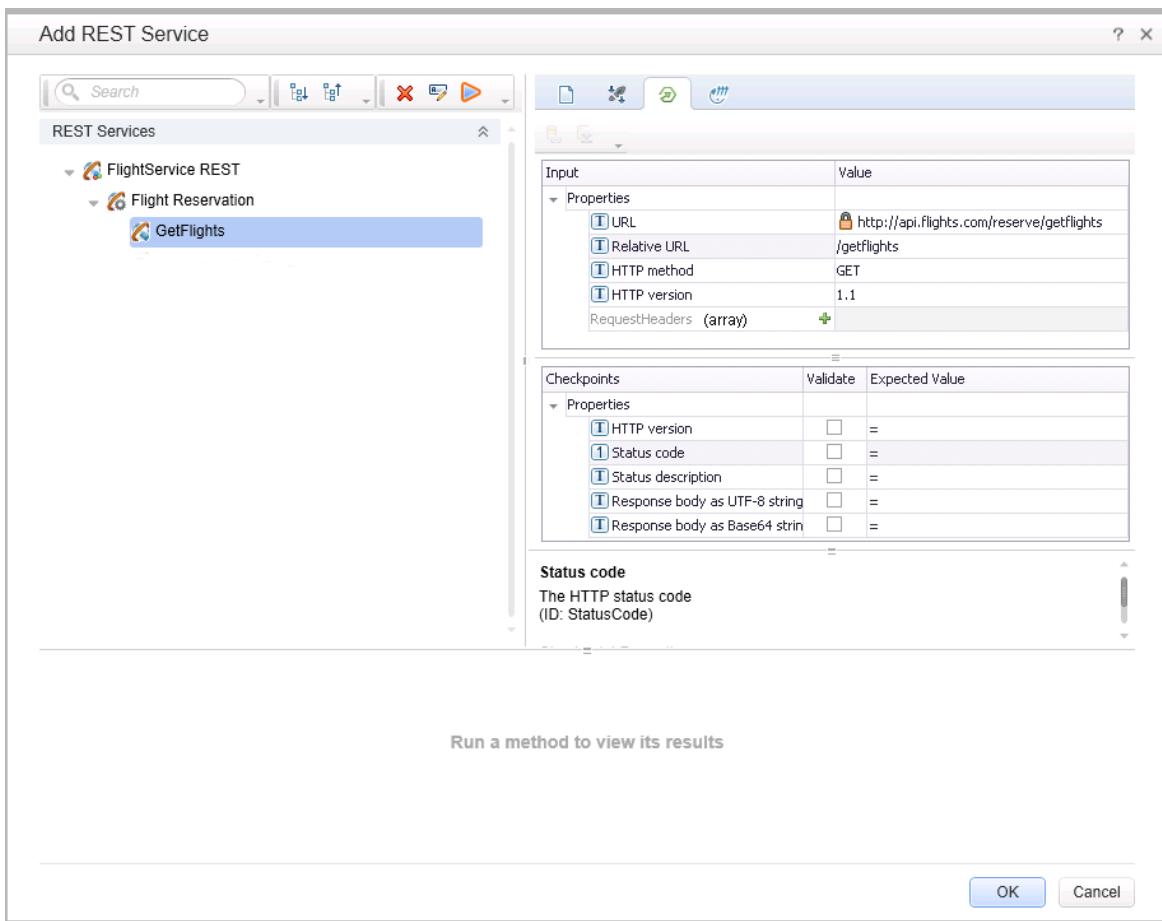
Example:

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.

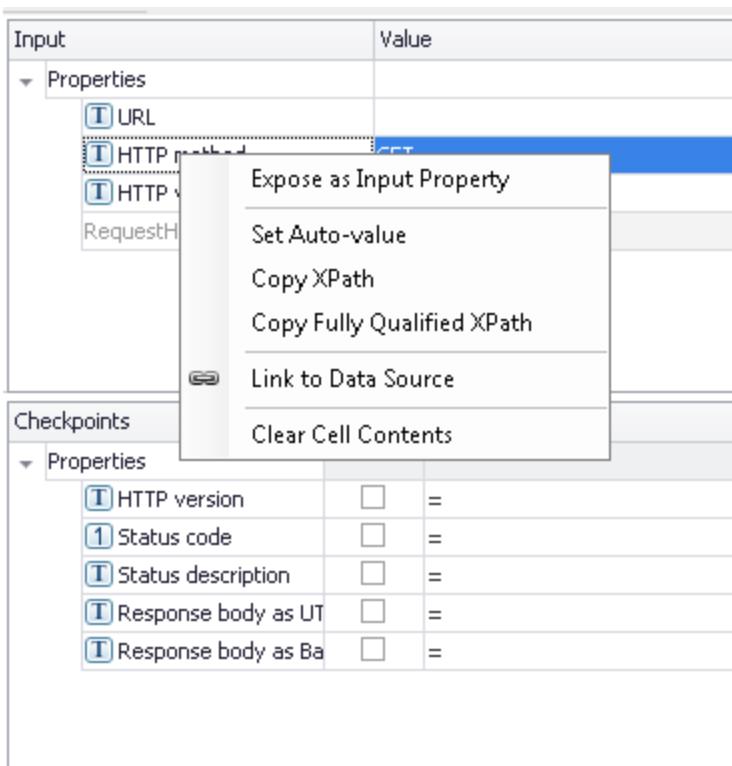
Exposing REST service properties

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.

Import a WSDL-based Web service

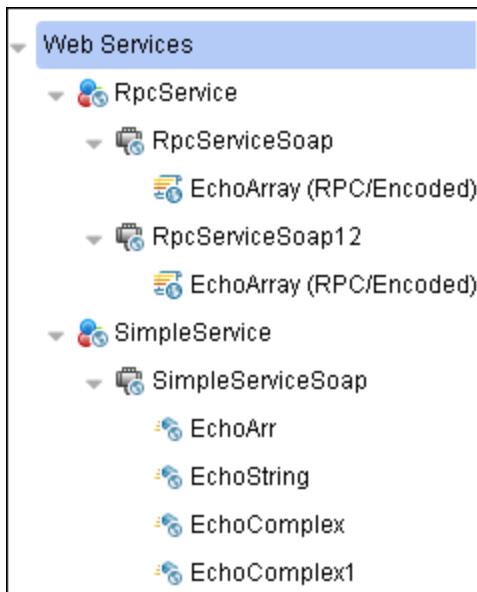
Relevant for: API testing only

Import your service

- In the toolbar, click the **Import WSDL** button  and select **Import WSDL from File or ALM Application Component** OR **Import WSDL from URL or UDDI**.
- Select the source:

For File System or ALM Application Components imports	In the Import WSDL dialog box, navigate to the location of the WSDL file and select it.
For URL imports	<ol style="list-style-type: none">In the Import WSDL dialog box, select URL.Click the Browse button adjacent to the Address field.In the browser window that opens, navigate to the URL containing the WSDL file.Close the browser window. The URL is automatically entered in the Address field.
For UDDI imports	<ol style="list-style-type: none">In the Import WSDL dialog box, select UDDI.Click the Browse button adjacent to the Address field.In the Select Service from UDDI dialog box that opens, specify the UDDI address and click Search. A list of all WSDL files saved in this UDDI location is displayed.From the list of WSDL files, select the file(s) to import and click OK. The address and file is automatically added in the Address field.

3. If your WSDL must be accessed through a secure server or proxy machine set the connection information:
 - a. 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):
 - 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. Click **OK** to import the service. If the import succeeds, the **Toolbox** displays the service, its ports, and operations under the **Web Services** node.



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**.

Update your WSDL information - optional

1. 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:

Update WSDL:	This updates the service details from the source provided when you first imported the WSDL file.
Update WSDL from:	This updates the service URL and details from one of the following sources: <ul style="list-style-type: none">• URL or UDDI: Enables you to reimport the WSDL file (containing the service model details) from a different URL or UDDI source.• File or ALM Component: Enables you to reimport the WSDL file from the file system or ALM.

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.

Add input attachments to the test - optional

1. Select the Web Service call step in the canvas and open the **Attachments** tab in the Properties pane.
2. In the upper pane, select an attachment Type: **DIME** or **MIME**.
3. Click in the **Attachments** row and click the **Add** button  to add an array element.
4. Select a file as the **Origin** of the attachment using the **Browse** button .
5. Select a **Content Type**. Specify a **Content ID** or keep the default value, **Auto**.

Validate output attachments

1. Select the Web Service call step in the canvas and open the **Attachments** view in the Properties pane.
2. 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).
3. 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**.
4. 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.

Configure SOAP Fault information - optional

To apply negative testing to a Web service:

1. In the canvas, select the step to which to apply the SOAP fault.
2. In the Properties pane, open the **SOAP Fault** tab .
3. Select **Fault is expected**.
4. Provide SOAP Fault checkpoint values for the negative testing:

To work In the XML layout:	Expand the SOAP nodes and define Any elements for the SOAP Header and Body. If relevant, provide values for faultcode , faultstring , or faultactor .
To work with XPath expressions:	Click the XPath tab and use the Add button to add new XPath entries. Copy the XPATH entry into the cell.

Create a REST service model

Relevant for: API testing only

Prerequisite

Study the structure of the REST Service body and determine which resources and methods you need to define.

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:

1. In the toolbar, click the **Add REST Service** button  and select **REST Service Editor**.
2. In the Add REST Service editor, give the service a name.
3. Add resources and methods as needed:

For new resources	Click the Add Resource toolbar button  and provide a meaningful name for the resource.
--------------------------	--

For new methods

Click the **Add Method** toolbar button  and provide a meaningful name for the method.

Set the General properties

1. Select the service, resource or method for which you want to set its properties.
2. In the right pane's **Properties** list, open the **General**  tab.
3. Enter values for the properties.

Set the URL property values

1. In the REST Service editor, select a REST service, resource, or method.
2. 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 416](#)

Define the method's HTTP properties

1. Select a REST service method.
2. In the REST service editor's right pane, click the **HTTP Input/Checkpoints** tab.
3. For each method, modify the **HTTP method** and **HTTP version**.
4. Use the plus sign  in the **RequestHeaders** parent node to add name and value pairs for the request header array.

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:

1. In the right pane's **Properties** list, click the **Custom Input/Checkpoints** tab .
2. Expand the plus button  and select **Add Input/Output Property**.

3. Provide a name, data type, and description (optional) for each property.
4. Enter the property values in the **Value** column.

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.)

1. Return to the design document for the REST service and copy the Request body onto the clipboard.
2. 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.
3. Modify the element values within the XML body as required.

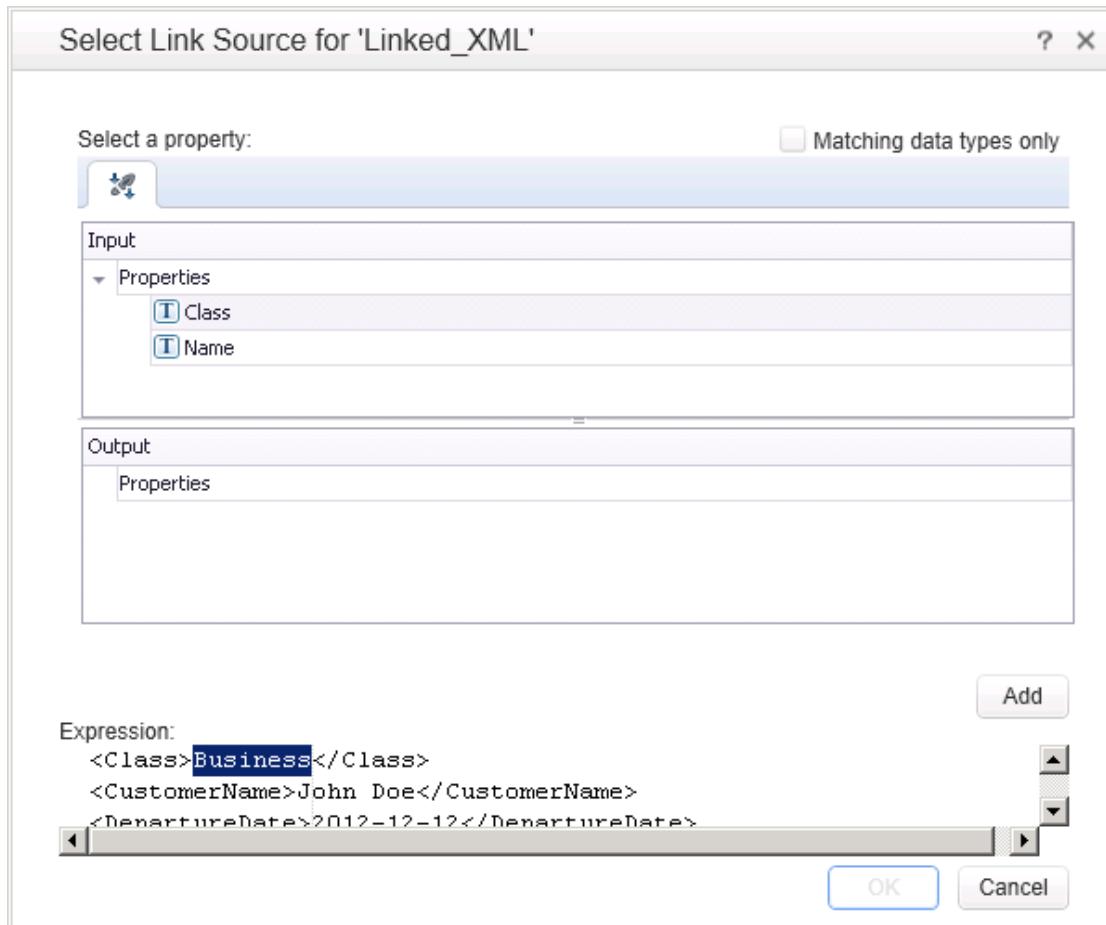
Note: If you need to use a multipart request with your REST Service methods, this is done after dragging the REST method to the canvas. For details, see "[Send a multipart HTTP or REST Service request](#)" on page 397.

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.)

1. Return to the design document for the REST service and copy the request body to the clipboard.
2. 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.
3. Click the **Link Body** button  to the right of the pane, to open the Select Link Source dialog box.
4. 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.

5. 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.



6. 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>
```

7. 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 "[Send a multipart HTTP or REST Service request](#)" on page 397.

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.

Save the service model to your test

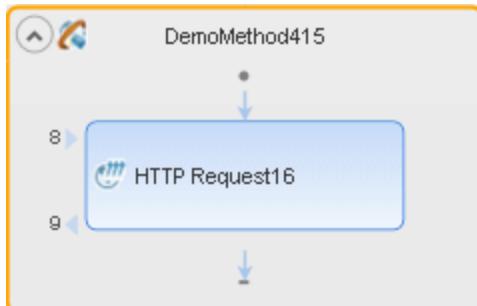
In the REST Service editor, click **OK** in the Design REST Service dialog box. UFT adds the REST service along with its resources and methods to the Toolbox pane, under the **Local Activities** category.

Expose input and output properties

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.

1. Double-click a REST method from the Toolbox pane to add it to the canvas. Expand the method to show the **HTTP Request** frame.



2. 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.
3. Provide a name for the property in the New Exposed Property dialog box.
4. Click the REST method wrapper in the canvas, and view the newly exposed property in the Properties pane's **Custom Input/Checkpoints** tab

Import a REST service model

UFT supports importing REST service model descriptions from WADL, Swagger, or OData REST APIs.

When you import service model descriptions, UFT automatically parses the service description and creates any relevant methods and resources.

Import the service

1. In the toolbar, click the **Add REST Service** drop-down button  and select one of the following:
 - **Import Swagger Service from URL/File**
 - **Import OData Service from URL/File**

The Import WADL/Swagger/OData Service from URL/File dialog box opens.

2. Do one of the following:

When importing from a file	In the Import dialog box, navigate to the JSON file (for Swagger services) or EDMX file (for OData services) that contains the service description.
When importing from a URL	In the Import Service from URL dialog box, 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.

If the schema you import is missing values for some elements, one of the following occurs:

Swagger	UFT automatically generates the missing values. To do this it uses the values defined in the Autovalues pane of the Options dialog box (Tools > Options > API Testing tab > Autovalues pane).
WADL/OData	These values remain empty.

Set authentication and response settings for a SAP HANA service

If you are importing an OData REST service model from a URL for a SAP HANA service, you must provide the log-on credentials as part of the import.

1. In the **Import OData Service from URL** dialog box, click **Advanced Settings**.
2. In the lower part of the dialog, select the **Use authentication settings** option.
3. Provide the log-on credentials for the SAP HANA service.
4. Select the **Use SAP HANA SAML** option.
5. Click **OK** to import the service.

6. From the **Local Activities** section of the Toolbox pane, add a service method to the canvas.
7. In the Properties pane, select the **General** tab .
8. In the General tab, set the **Use SAP SAML Authentication** cell value to **True** or **False**.

Use the service's methods in your test

After you import the service model into UFT, the methods are displayed in the Toolbox pane under the **Local Activities** node.

Add these methods to your test as needed.

Send and receive a JSON request for a REST service

Relevant for: API testing only

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 "["Create a REST service model" on page 424](#)".

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**.

Load the request body

1. In the Properties pane, open the **HTTP** tab .
2. Select **Body type** **JSON**.
3. 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.

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.

1. In the Properties pane, open the **Input/Checkpoints** tab .
2. In the Input/Checkpoints tab, click the plus sign to add a **RequestHeader** array element.
3. Add a custom request header named `Accept` with the value `application/json`.

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.

In the Input/Checkpoints tab, in the Input area, display 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":  
[\  
 {"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.

Import a Web Application service

Relevant for: API testing only

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>
        <resource path="readers">
            <method name="GET"/>
        </resource>
    </resources>
</application>
```

For details on WADL elements, see <http://www.w3.org/Submission/wadl/>.

Import the WADL document

1. 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**.
2. 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"	WADL service name
resource path=<resource name>"	WADL resource If multiple resources have the same name, UFT numbers the resources sequentially to differentiate between resources.
method name=<method name>"	WADL method <ul style="list-style-type: none">• 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 <"method name="GET"/>, 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. 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>"	<p>WADL resource or method parameters. 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.</p> <p>If a "param name = <name>" string also contains a "default=<value>" string, the value defined in the XML is displayed with the parameter.</p>
resources base="http://example.com/api"	<p>WADL Service URL. This is displayed on the HTTP tab for the service.</p> <p>The URL is also passed to all resources and methods included in the service. For details, see "Passing REST service properties" on page 416.</p> <p>You cannot change the URL property for an individual resource or method.</p>

Import a Network Capture file

Relevant for: API testing

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.

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.

Import the network capture file

1. In UFT, with an API test open or selected, select **Tools > Import Network Capture File**. The Import Network Capture Dialog Box opens.
2. 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.

3. 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.
4. 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.
5. 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.

Create an SAP API test step

Relevant for: API testing only

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.

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.

Select an iDoc from your SAP server

1. Select **Tools > Import** from SAP.
2. In the SAP dialog box, select one of the connections that you added in the SAP Connections pane of the Options dialog box.
3. (Optional) Accept the default credentials entered in the SAP Connections pane or click **Override connection** to provide different credentials.
4. Select **IDoc** or **RFC** and specify a search string using an asterisk (*).
5. Click **Search**.
6. 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.

Import and create a .NET Assembly API test step

Relevant for: API testing only

Prerequisite

Prepare a **.dll** assembly file and store it in an accessible location.

Import a .NET assembly

1. In the toolbar, click the **Import .NET Assembly** button .
2. In the Import .NET Assembly dialog box, select the **.NET Assembly Browser** tab.
3. Navigate to the direction containing the assembly .dll or .exe file and select the file.
4. Click **Select** to add it to the **Selected References** list.
5. 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.

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.

1. In the Import .NET Assembly dialog box, select the **GAC** tab.
2. In the list of references, select one or more GAC assemblies and click **Select** to add them to the **Selected References** list.

Add an ExecuteEvent event handler

1. In the Properties pane, open the **Events** tab .
2. In the **ExecuteEvent** row, in the Handler column, click the down arrow and select Create a default handler.
3. In the **TestUserCode.cs** file that opens, locate the **CodeActivity<#>_ExecuteEvent** section of the code.
4. 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_ExecuteEvent(object sender,
STActivityEventArgs args)
{
    this.CodeActivity4.Input.Property1...
```

```
...  
}
```

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:

1. After adding a step, in the Properties pane, open the Input/Output Properties tab .
2. In the Input/Output Properties tab, click the **Add Input/Output Property** button and add input and output properties.

Known Issues- Custom Activities

Relevant for: API testing only

Web services

- 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.**

Workaround: Rename the WSDL file before the import.

- The following WSDLs are not supported:
 - WSDL version 2.0
 - WSDLs containing a `<appInfo>` element.
 - RPC Encoded WSDLs configured as Asynchronous Web services.
 - WSDLs authenticated by HTTP digest on Apache servers.
- When opening many tests in the same solution using one or more of the same Web services, UFT may develop a memory leak.

Workaround: Move the Web service to the file system or ALM repository. To do this, import a Web service and then, in the **Toolbox** pane, right-click the Web Service activity and select **Move to > File System Activities** or **ALM Activities**.

- For a Web service imported as a server response:
 - 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.
 - 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.

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 saved with the test.

Workaround: Add the certificate to the local machine store before the listener starts, and remove it at the end of the listening process.

REST services

- Importing a schema to a REST, HTTP, or SOAP checkpoint may remove the links of the input properties.
- XPath checkpoints are not supported for HTTP and REST activities.

- 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.

Workaround: Delete the link explicitly either before or immediately after deleting its source property.

- 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 **Validate** column, in the Checkpoints pane.
- When running a REST method using the **Run Step** command, checkpoints of dynamic property values in the method linked to other property values (input or output) are ignored.
- When building JSON content and loading it in the XML Grid Model for OData and Swagger, all data types other than **String** are included as values in the JSON content.

Web Application Services

The following elements are not supported when importing your WADL. UFT does not import these elements into the WADL hierarchy inside your test:

- **Grammars** element
- **Resource_type** element
- **Link** element

Any child elements of these elements are also ignored by UFT and not added to your WADL inside the test.

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.

Network Capture Activities

Steps added to your test from a network capture file do not include:

- Security settings for Web service calls and SOAP requests
- Web authentication information for any type of step
- Cookies data for HTTP request
- Attachments sent as part of the Web Service call

.NET Assembly Activities

Importing or adding references to 64-bit .NET assemblies is not supported.

Chapter 43: Actions for API Tests

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.

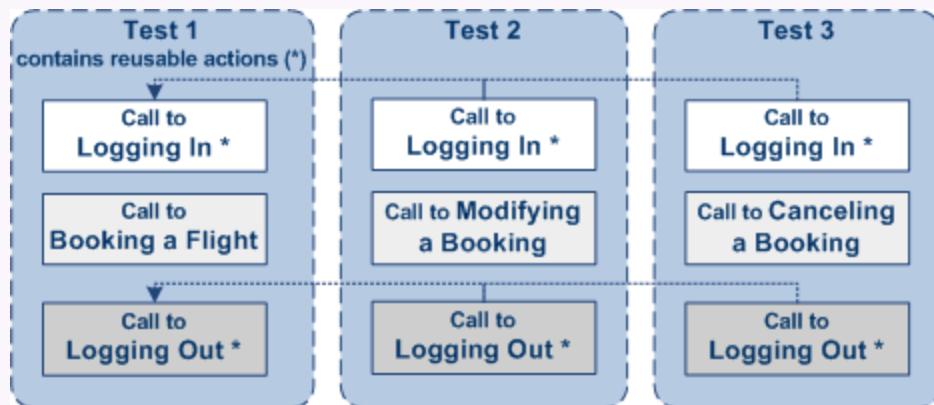
You can call both actions defined within your current test or defined in other tests. When you call an action from another test, by default its data is read-only. If you call an action from an external test that has data assigned to its properties, you can indicate whether you want the data from action's 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.

Actions and data sources

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 "["Data Usage in API Tests" on page 445](#).

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 "[Use actions in an API test](#)" below.

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.

Use actions in an API test

Relevant for: API testing only

Create 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. From the Toolbox pane, expand the **HP Automated Testing Tools** node and add a **Call API Action or Test** or **Call GUI Action or Test** to 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 test and action and 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 "[Assign data to API test/component steps](#)" on page 454.
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.
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.

Chapter 44: Data Usage in API Tests

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:

Link to a data source.	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.
Link to another test step:	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.
Link to multiple sources.	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.
Link to a test variable value.	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.

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.

For task details on using data with test steps, see "[Assign data to API test/component steps](#) on page 454".

Data assignment in arrays

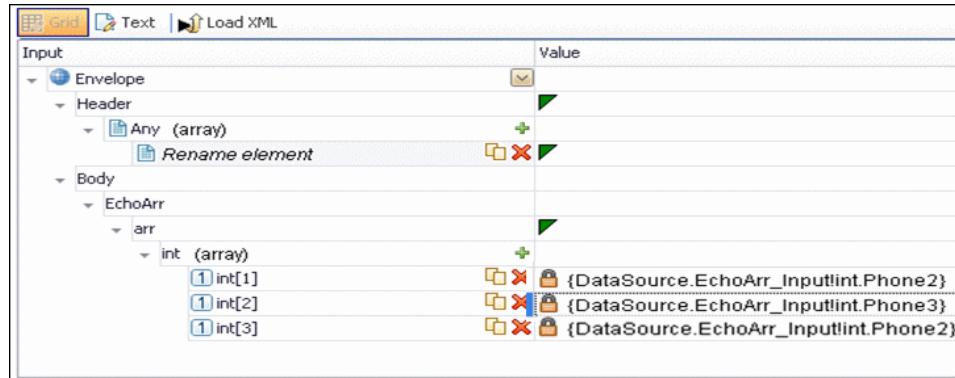
Relevant for: API testing only

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	<p>When you have one or more data relations defined, as described in "Create a new child relation" on page 471, UFT assigns data from a single column. You link the first element of the array to a column in a data source.</p> <p>In order for UFT to assign data based on a data relation, the following conditions must be present:</p>
---------------------------------------	---

- 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 'Input' section of the configuration. An array element 'int[1]' under 'EchoArr/arr/int (array)' is linked to the 'MainDetailsKey' column of a data source named 'DataSource.EchoArr_Input!int'. The 'Value' column for this row contains the expression '{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' section. A data source 'EchoArr_Input>MainDetails' is selected. To its right is a data grid with columns 'A1' and 'MainDetailsKey'. The 'MainDetailsKey' column contains the value 'MainDetailsKey' for row 1 and '1' for row 2. Rows 3 through 11 are empty.

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: <ul style="list-style-type: none">• 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.• 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.
--	--

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 82](#).

Relevant for: API testing only

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**.
7. In the Data pane, select the data source under the **Excel** node. In the right pane, you can view and edit the values.

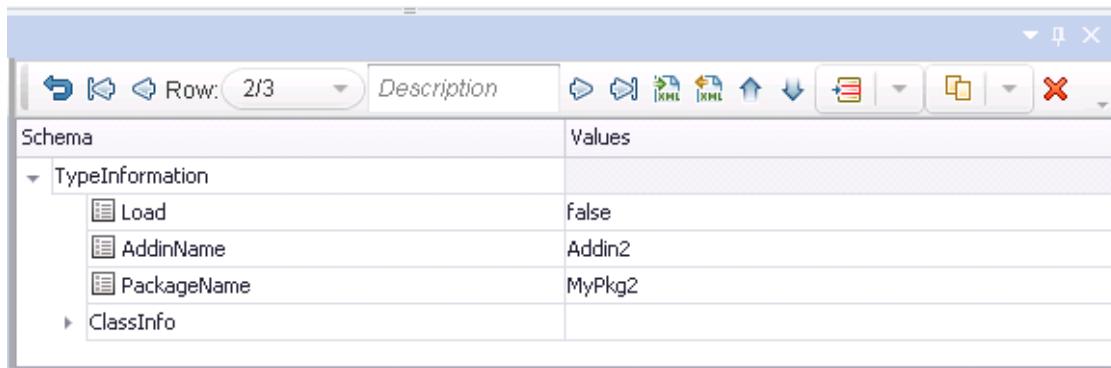


Caution: When entering values into the Excel sheet, you should use the Excel General format for values.

Add an XML data source

1. In the Data pane, click the **New Data Source** down arrow  and select **XML**. The New XML Data Source Dialog Box opens.
2. Provide a **Data Source name**.
3. 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.
4. Click **OK**.

5. Select the data source's node in the Data pane.



The screenshot shows the Data pane interface. At the top, there are navigation buttons (Back, Forward, Row: 2/3, Description) and a toolbar with various icons. Below is a table with two columns: Schema and Values. A tree view under Schema shows a node 'TypeInformation' expanded, revealing four rows: Load (Value: false), AddinName (Value: Addin2), PackageName (Value: MyPkg2), and ClassInfo (empty). There are also buttons for adding, deleting, and saving rows.

Schema	Values
TypeInformation	
Load	false
AddinName	Addin2
PackageName	MyPkg2
ClassInfo	

6. Edit the XML data in the grid. Click the **Add rows** button  to add row values.

7. To load XML values, click the **Load data from an XML file** button .

Add a database data source

1. 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.
2. In the Set Database Connection pane, add the database connection information as follows:

For an OleDb type connection:	<ol style="list-style-type: none">a. Select OleDb as the Connection type.b. Set the connection string in one of the following ways:<ul style="list-style-type: none">◦ Paste an OLE DB connection string into the text area.◦ Select a previously defined connection string from the drop down.◦ Click the Build Connection String button  to use Microsoft's Data Link Properties dialog box.c. Click Test Connection to verify that the connection is valid.d. Click Next. This button is only available for valid connections.
--------------------------------------	--

For an ODBC type connection:	<ol style="list-style-type: none">a. Select ODBC as the Connection type.b. Click the Build Connection String button  to select a DSN type.c. Select a DSN type and provide the necessary credentials. To create or edit a DSN entry, click the Manage ODBC Data Sources button. For details, click the Help button in the ODBC Data Source Administrator dialog box. After you close the administrator dialog box, click the Reload DNS list button to refresh the list.d. Test the connection. After the confirmation, click OK.
-------------------------------------	--

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:

Paste an existing statement into the text area	Take your preexisting statement and enter it.
Select a previously defined statement	From the drop-down list, select a previously-used statement.

Use the Query Designer	<ol style="list-style-type: none">i. Click the Build Query Statement button  to open the Query Designer Dialog Box.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.iii. Enable Auto execute to view the results of your query as you make changes in the upper pane.iv. By default, the Query Designer selects all columns in the table. To remove a column from the query, clear the Output check box for that column.v. To set a sorting order, select Ascend or Descend from the Sort column drop-down. If you have multiple rows that you are sorting, provide the Sort Order beginning with 1. The Query Designer adds an <code>ORDER BY</code> clause to the preview pane.vi. To add a <code>GROUP BY</code> or <code>DISTINCT</code> clause to your statements, select the appropriate check box.vii. Click OK to accept the query and return to the wizard.
-------------------------------	--

- 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.

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

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.



Tip:

- 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.

Link your test step to multiple sources

1. If necessary, add a data source. For details on adding data sources, see "[Add data sources to an API test](#)" on page 449.
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.
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 unmatching 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.

Add data keywords

You can use keywords to customize the test run and validation. For example, **SKIP** omits a property in the request or in the validation.

1. In the Value column for a step property, do one of the following:
 - 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).
2. Select the appropriate keyword:

Input Keywords	#SKIP#	Omits this element from the XML of the request. This is useful for elements of SOAP requests, for which minOccurs = 0
	#NIL#	Adds a nil=true attribute to the property's XML in the request. <pre><name John Doe nil="true"></name></pre> If the XML element is not nillable, this is reported to the log.
Checkpoint Keywords	#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 minOccurs = 0 .
	#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 minOccurs = 0 .
	#SKIP#	Informs the run engine to ignore this value when evaluating checkpoints.

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 "[Assign data to API test/component steps](#)" on page 454.

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:

1. Open the Flight API application from the **Start** menu or screen or the file system (**<UFT installation folder\samples\Flights Application\Flights API.exe**).
2. Copy the URL for the Web Service of the Flight API Application.
3. Import the Web Service into UFT.

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\Flights Application** directory.

1. In the Data pane, click the **New Data Source** button  and select **Excel**.
 2. 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.

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)

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:

1. In the canvas, make sure that the **GetFlights** step is selected.
2. In the Properties pane, open the **Input/Checkpoints** tab .
3. In the Input/Checkpoints tab, manually select the following values from the property value drop-down lists:
 - **DepartureCity:** Denver
 - **Arrival City:** Los Angeles

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.

1. In the canvas, select the **CreateFlightOrder** step.
2. In the Properties pane, open the **Input/Checkpoints** tab .
3. In the Input/Checkpoints tab, in the **Value** cell for the **FlightNumber** input property, select the **Link to data** source button .
4. 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 455.

Note: By default, the **FlightNumber** property is not visible. Expand the **GetFlightsResult** node and click the **Add** button  to expand all the output properties.

5. 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.

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.

1. In the canvas, select the **Report Message** step.
2. In the Properties pane, open the **Input/Checkpoints** tab .
3. In the Input/Checkpoints tab, in the **Value** cell of the **Message** property, select the **Link to data source** button .

4. In the Select Link Source dialog box, click the **Custom Expression** button to display the expression area.
5. In the expression area, enter the text "Your flight order number is (with an extra space at the end).
6. In the Select Link Source dialog box, select the Available step option.
7. In the list of available steps (left pane), select the **CreateFlightOrder** step.
8. In CreateFlightOrder properties list in the right pane, in the **Output Properties** section, expand the **CreateFlightOrderResult** node.
9. In the output properties list, select the **OrderNumber** property.
10. Click **Add** to add this to the custom expression.
11. Click **OK** in the Select Link Source dialog box to add this expression.
12. The expression Your flight order number is
`{Step.OutputProperties.StServiceCallActivity4.Body.CreateFlightOrderResponse.CreateFlightOrderResult.OrderNumber}` appears in the Value column of the Message property

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.

Define API test properties or user/system variables

Relevant for: API testing only

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.

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.
3. Give the variable a description and a default value.

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 . In the Manage Profiles dialog box, 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.

Parameterize XML data

Relevant for: API testing only

This task describes how to parameterize XML data. This enables you to replace a constant value in the XML schema with a data source expression.

1. Prerequisites

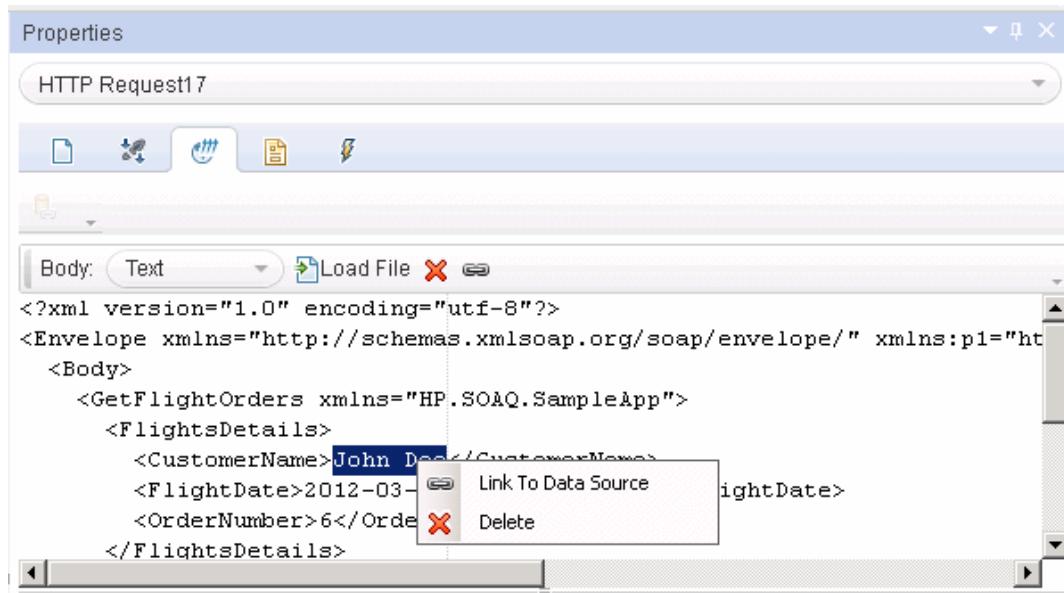
Create a test step that uses XML text in the request body. This applies to Network and XML type activities. If you intend to parameterize the data with a data source, import or create the data source.

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.

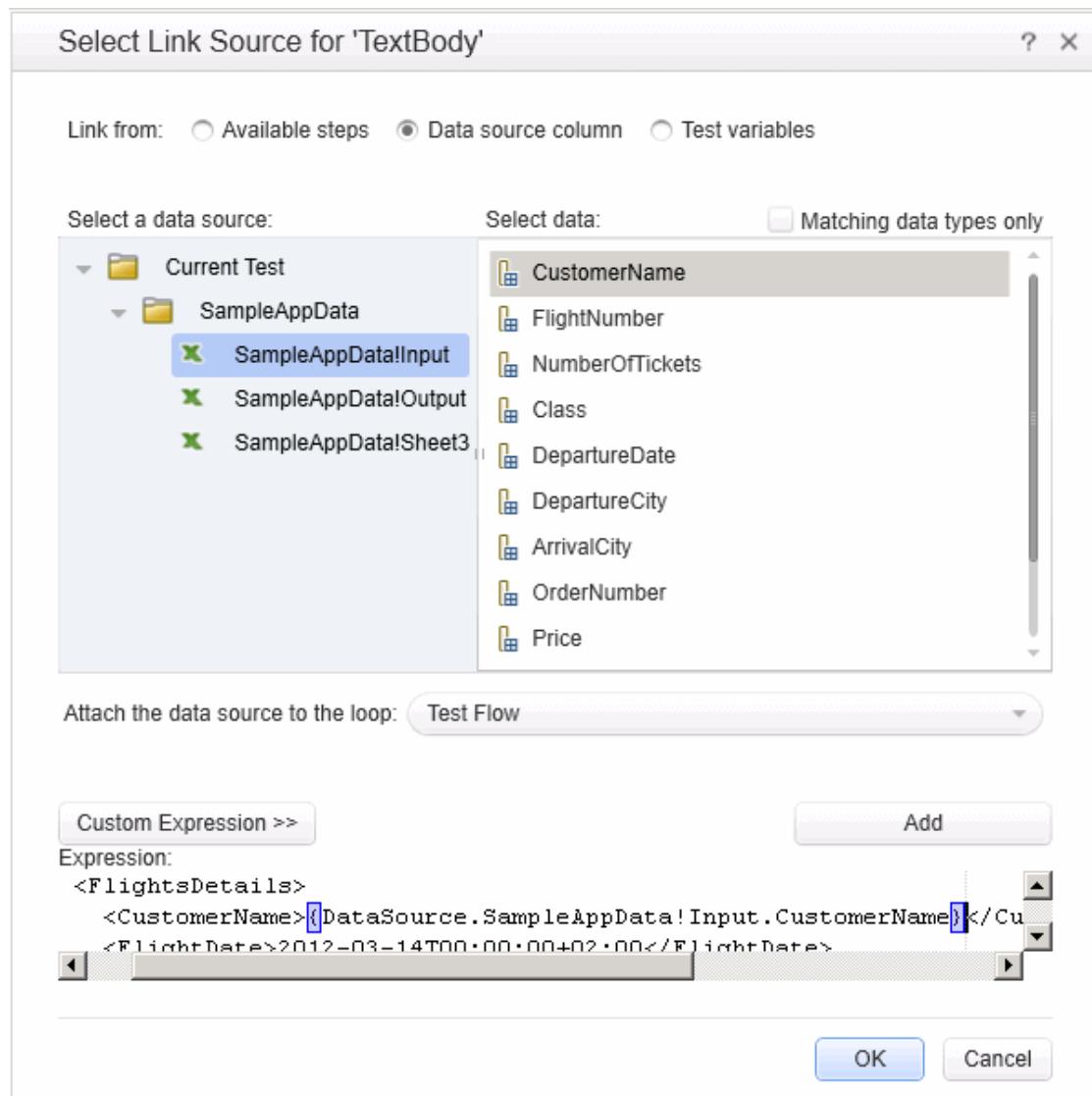
3. Select the text to parameterize

- 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.



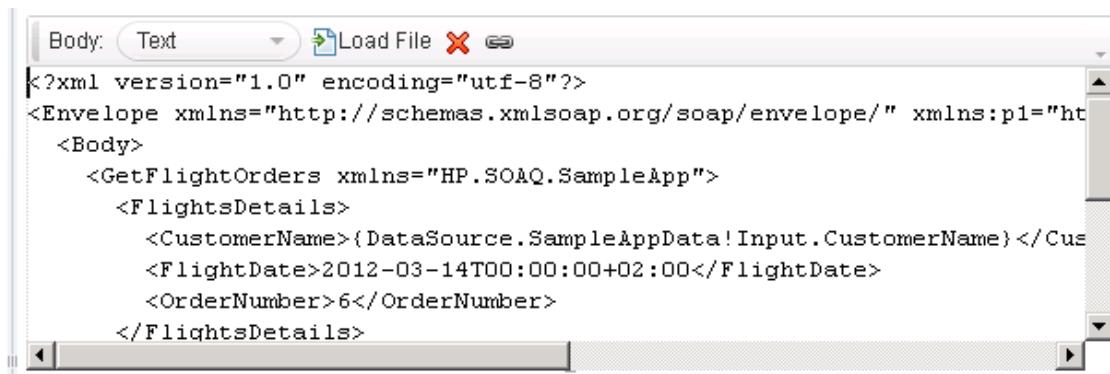
4. Select a link source

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 contains 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>
```

Data drive array checkpoints

Relevant for: API testing only

This task describes how to set checkpoints for elements of an array through data driving.

Enable active content on your computer

1. In Internet Explorer, select **Tools > Options**.
2. Select the **Advanced** tab.
3. Enable the option **Allow active content to run in files on My Computer** in the **Security** section.
4. Click **OK** and close the browser.

Add a step with an array output

Add a test step with output properties in the form of an array.

Data drive the array

1. In the Properties pane, open the **Input/Checkpoints** tab .
2. In the Checkpoints area, use the plus button  in the row of the parent node, to add an array element.
3. 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. Select the array's parent node and click the **Data Drive** button .

5. In the Data Driving Dialog Box, select **Only Input, Checkpoints** or **Both Input and Checkpoints as a Data Driven Section** option.
6. Click **OK**. The data driving mechanism informs you that it succeeded.

Set the evaluation expression

In the Properties pane, adjacent to the data driving expression, select an evaluation operator, such as =, <, and so forth.

Provide data for the array

1. 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, click the node in the left pane, corresponding to the array element that is to be validated.
2. In the **<array_name>** and **MainDetails** tables, enter the iteration number to which to checkpoint should be applied in the **MainDetails** column. 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.

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.

Set the number of iterations - optional

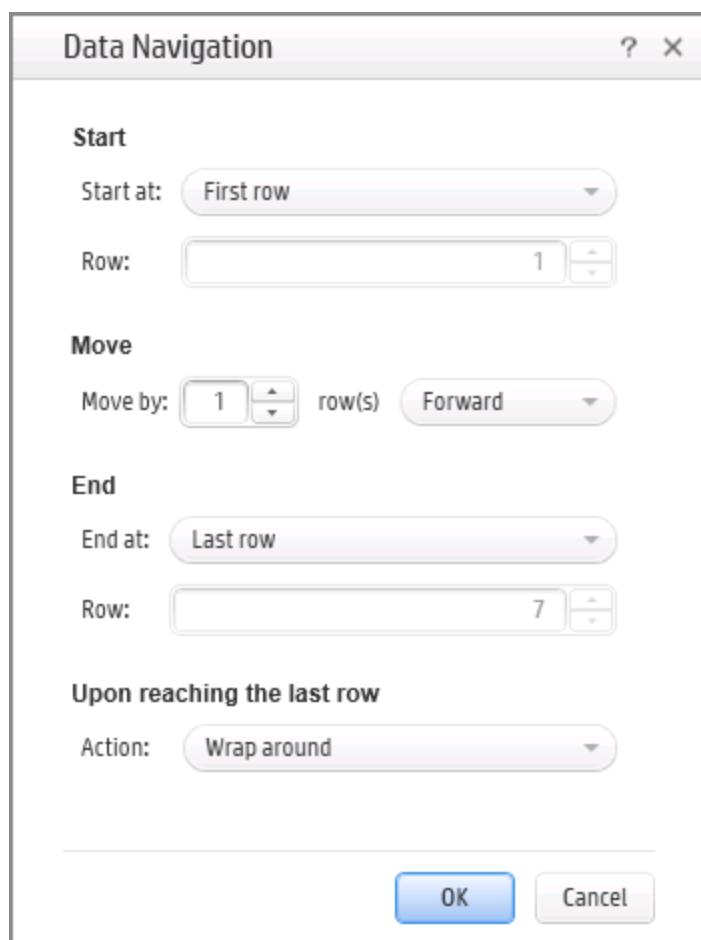
1. Select the **Test Flow** or **Loop** box in the canvas.
2. In the Properties pane, open the Input/Checkpoints tab.

3. Set the number of iterations for the test run.

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).

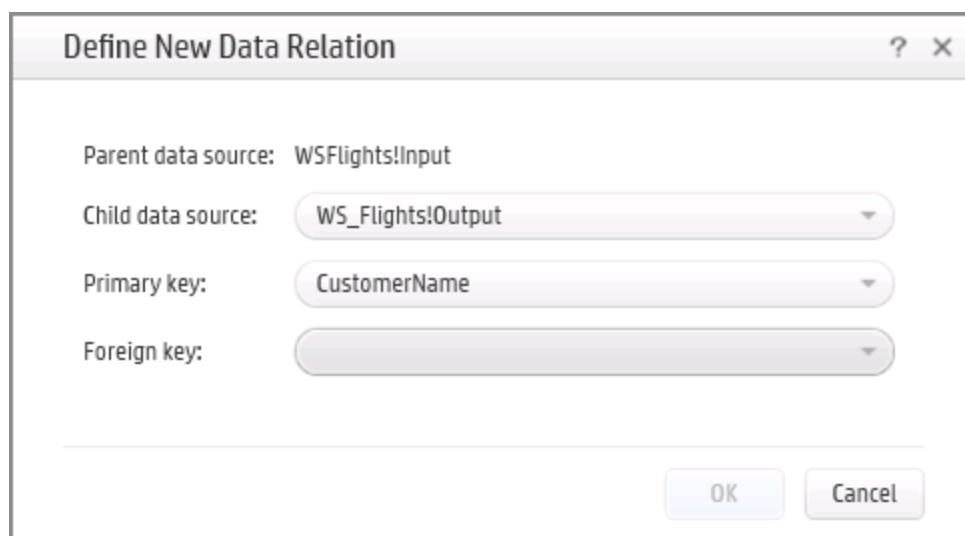
For task details, see "[Set the data source navigation properties](#)" on the next page.

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 the next page.

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.

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.

Known Issues- Using data in API tests and components

Relevant for: API testing only

This section describes troubleshooting and limitations for working with data.

General	<ul style="list-style-type: none">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.Keywords such as #SKIP#, and so forth, are not supported in the Input property grid. Workaround: Link to a data source that contains the keyword.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.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.Linking to file names is not supported for HTTP Request and HTTP Receiver activities that use the File type message body.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.
Data driving	<ul style="list-style-type: none">Data driving for JSON requests or responses, is only supported for Excel.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.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.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.Keyword data driving to an XML data source is not supported.

Chapter 45: Updating Services and Assemblies

Relevant for: API testing only

You can update services that exist in the Toolbox pane, from their original locations or from alternate locations.

For Web Services	<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 (<code><service name:port name></code> combination) do not match, and security was set for on the port level, UFT opens the Update Port Security Wizard.</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 ""Update a Web Service" on the next page.</p>
-------------------------	---

For REST services	<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"> • Adding or removing an input or output property • Renaming an input or output property • A change in the REST service URL • A change in the HTTP request/response body type or schema • 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) • HTTP methods <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>
For SAP IDocs or RFCs	<p>You can update SAP RFCs and IDocs from their original location or from another connection or location.</p> <p>The Update 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 Update from 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 "Update an SAP RFC or IDoc" on page 478.</p>

For task details, see "[Update a Web Service](#)" below, "[Resolve conflicts in a REST service test step](#)" on page 477, or "[Update an SAP RFC or IDoc](#)" on page 478.

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.

Prerequisites

If you want to update the WSDL from ALM, make sure you have an open connection to the ALM server.

Update the service

To update a service from its original location	In the Toolbox pane, right-click the main service node and select Update WSDL .
To update the service from another location or if the Update WSDL option is not available	<ol style="list-style-type: none">1. Select the service's main node in the Toolbox pane.2. Choose Update WSDL from > URL or UDDI or Update WSDL from > File or ALM Application Component from the right-click menu.3. Navigate to the WSDL and click OK. IMPORTANT: The service or service location URL must be accessible when updating the service.4. 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.

Run the Update Port Security wizard - optional

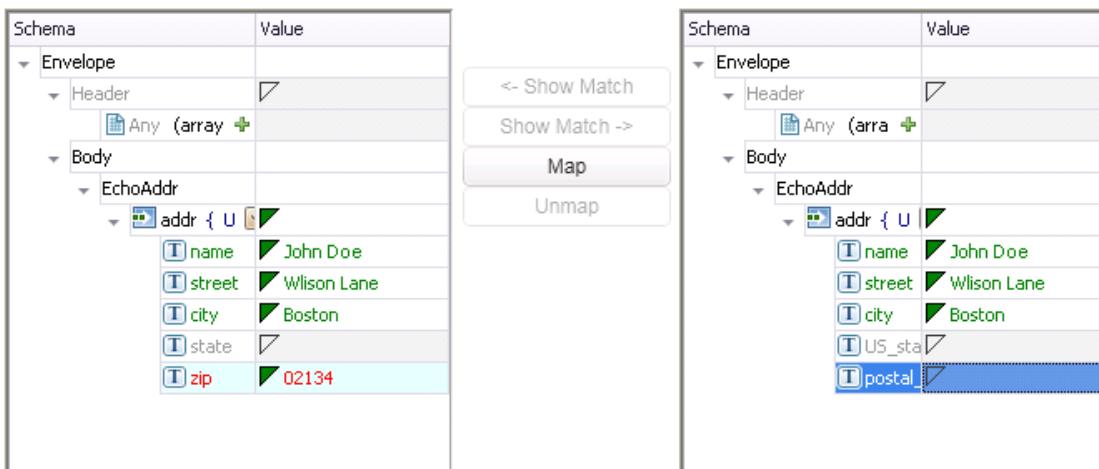
If UFT detects a change in the port path (<**service name:port name**> combination) the **Update Port Security** Wizard opens. The wizard only opens if you configured security settings for the original service. Follow the steps of the wizard to resolve all of the Service/Port conflicts.

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.

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.

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 **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.
3. In the page, 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. 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**.
5. Click **Finish** to save your changes and close the wizard.

Update a .NET Assembly

Relevant for: API testing only

This task describes how to update a .NET assembly with a newer version.

Select the assembly to update

In the Toolbox pane, expand the **.NET Assemblies** node and select the assembly you want to update.

Import a new .NET assembly

1. In the Toolbar, click the Import .NET Assembly button .
2. In the Import .NET Assembly dialog box, click the **.NET Assembly Browser** tab.
3. In the .NET Assembly Browser tab, click **Browse** and navigate to the the .dll or .exe file.
4. Click **Open** to add it to the **Selected References** list.
5. Click **OK** to begin the update.

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.

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.

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.

Modify the step as required

In the Properties pane, 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.

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**.

Run the wizard

1. 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.

2. Proceed with the wizard, resolving or keeping the detected differences.

Update an SAP RFC or IDoc

Relevant for: API testing only

Update the RFC/IDoc from its original location

In the Toolbox pane, right-click the RFC or iDoc and select Update.

Update the RFC/IDoc from a different location

1. In the Toolbox pane, right-click the RFC or IDoc and select **Update from**. The Update <Item_Name> dialog box opens.
2. To change the connection or server information, do one of the following:
 - In the Update dialog box, select **Override connection** and specify the details
 - In the SAP Connections pane in the Options dialog (**Tools > Options > API Testing** tab > **SAP Connections** node), change the default connection information.
3. In the bottom pane, search for and select a different RFC or IDoc.
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. In the canvas, 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.

Known Issues- Updating Services

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

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

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:

Default Web Service Scenario	A default Web Service 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 " Web Service scenario " below. You use the default Web Service scenario for: <ul style="list-style-type: none"> • Simple Web Services where no advanced standards are required. • Web services using HTTP transport level security. • Web services using message level security (WS-Security) for SOAP 1.1
WCF-type scenario	WCF scenarios enables you to configure security for HTTP or custom bindings and work with advanced specifications, such as WS-SecureConversation . You use a WCF scenario for: <ul style="list-style-type: none"> • WCF Services that utilize advanced security and WS-Specifications. • Web services using message level security (WS-Security) for SOAP 1.2

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.

Web Service scenario

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:

Transport Level Security	The transport level security includes the authentication and proxy server information. You can also specify Keep Alive preferences and connection timeout.
---------------------------------	--

Message Level Security	<p>The WS-Security tab lets you set the message level security through tokens, signatures and encryption.</p> <p>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: UserName, X509 Certificate, Kerberos, Kerberos2, and SAML:</p> <ul style="list-style-type: none">• UserName: The User Name token contains user identification information for the purpose of authentication: User Name and Password. You can also specify Password Options, indicating how to send the password to the server for authentication: Text, None, or Hash. and indicate whether to include a timestamp.• X509 Certificate: 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.• Kerberos/Kerberos 2: 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. <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">• SAML Token: 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.

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.

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

In addition to the default Web service scenario, you can select from one of the following WCF-type service scenarios:

**WCF Service
(CustomBinding)
Scenario**

The **WCF Service (CustomBinding)** 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.

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:

- **Transport.** **HTTP**, **HTTPS**, **TCP**, or **NamedPipe**
- **Encoding.** **Text**, **MTOM**, or **WCF Binary**

You can also provide additional security information:

- **Authentication mode.** The type of authentication, such as **None**, **AnonymousForCertificate**, and so forth. The options are available from the drop down list.
- **Bootstrap policy.** For the **SecureConversation** authentication mode, you can specify a bootstrap policy.
- **Net Security.** The network security for **TCP** and **NamedPipe** type transports. Typical values are **None**, **Windows stream security**, or **SSL stream security** available from the field's drop down list.
For services with HTTP transport, you should set the value to **None**. To enable SSL for HTTP, select **HTTPS transport**.

For task details, see "["Customize security for WCF-type Web services" on page 493](#)".

Note: For WSE3 security configurations, use the WCF Service(CustomBinding) Scenario. For details, see "["Customize security for WCF-type Web services" on page 493](#)".

**WCF Service
(Federation)
Scenario**

In the **WCF Service (Federation)** 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:

- Provide security details for the application server's security scenario in the following areas:
 - **Server.** The transport and encoding methods.
 - **Security.** The authentication mode, such as **IssuedToken**, **SecureConversation**, and so forth.
 - **Identity.** Information about the server certificate and expected DNS.
 - **STS.** Settings related to the STS, such as the endpoint address and binding.
- Define an STS binding in the **Referenced binding** field.

For task details, see "[Customize security for WCF-type Web services](#)" on page 493.

WCF Services (WSHttpBinding) Scenario	In the WCFSERVICE (WSHttpBinding) scenario, you can select from several types of authentication: None, Windows, Certificate, or Username (message protection).
--	---

No Authentication (Anonymous)

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:

- Client uses the server's X.509 certificate for encryption.
- Client is not authenticated.
- Communication may utilize advanced standards such as secure conversation or MTOM.

Windows Authentication

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.

Use this scenario to test Web Services where the:

- Client and server use Windows authentication.
- Security is based on Kerberos or SPNEGO negotiations.
- Communication may utilize advanced standards such as secure conversation or MTOM.

Certificate Authentication

In this **WCF WSHttpBinding** scenario, the client uses the server's X.509 certificate to encrypt the message and its own certificate for a signature.

Use this scenario to test Web Services where the:

- Client uses the server's X.509 certificate for encryption.
- Client uses its own X.509 certificate for signatures.
- Communication may utilize advanced standards such as secure conversation or MTOM.

Username Authentication (Message Protection)

In the **WCF WSHttpBinding** scenario, the client uses the server's X.509 certificate to encrypt the message, and sends a user name and password to authenticate itself.

Use this scenario to test Web Services where the:

- Client uses the server's X.509 certificate for encryption.
- Client is authenticated with a username and password.
- Communication may utilize advanced standards such as secure conversation or MTOM.

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.

Create a Web Service scenario

1. 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.
2. In the Security Settings dialog box, select **Web Service** from the **Service Details** dropdown list (default).

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.

Add message level security with a Username Token

To send a message level username/password token (a UserName token):

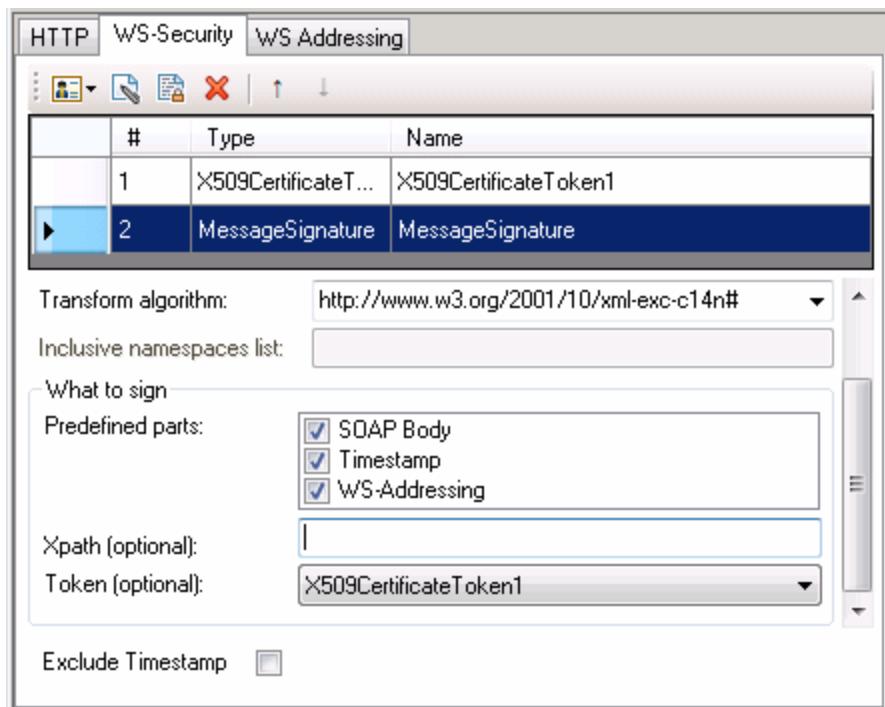
1. In the Security Settings dialog box, from the Service Details dropdown list, select the **Web Service** scenario.
2. In the main part of the Security Settings dialog box, click the **WS-Security** tab.
3. In the WS-Security tab, click the **Add Token**  button and add a **Username** token.

4. In the lower pane, customize the token details, such as username and password.

Add message level security by signing with an X.509 Certificate

1. In the Security Settings dialog box, select the **Web Service** scenario from the **Service Details** dropdown list.
2. In the main part of the Security Settings dialog box, click the **WS-Security** tab.
3. In the WS-Security tab, click the **Add Token** button  and select **X509 Certificate** from the dropdown list
4. In the lower pane, enter the token name.
5. Click the **Browse** button to navigate to your certificate file. The certificate must be installed in the Windows certificate store.
6. Select a **Reference type**. Since this token is used for a signature, the most common type is **BinarySecurityToken**.
7. Click the **Add Message Signature** button .
8. In the **Signing Token** dropdown list, select the token you entered in the previous steps.
9. 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.

Encrypt a Web service message using a Certificate

To encrypt a message using the service's X.509 certificate:

- In the Security Settings dialog box, select the **Web Service** scenario from the **Service Details** dropdown list,
- In the main part of the Security Settings dialog box, select the **WS-Security** tab.
- In the WS-Security tab, click the **Add Token** button and select **X509 Certificate** token from the **Security Token** drop down list.
- Enter token name.
- Since this token is used for encryption, use **Reference** as the **Reference type**.
- Click the **Add Message Encryption** button . In the drop down list, select the token you created in the previous steps.
- Scroll down to the **XPath** field. Enter an XPath expression to the elements to encrypt, for example: // *[local-name()='Body'].

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:

1. In the Security Settings dialog box, select the **Web Service** scenario from the **Service Details** dropdown list.
2. In the main part of the Security Settings dialog box, select the **WS-Security** tab.
3. In the WS-Security tab, click the **Add Token** button  and select **Username Token** from the **Security Token** drop down list.
4. In the lower pane, Provide the token details for the Username token.
5. Click the **Add Token** button  again and select **X509 Certificate Token** from the **Security Token**  drop down list.
6. 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**.
7. Click the **Add Message Encryption** button  . In the drop down list, select the X.509 token you created in the previous steps.
8. To encrypt a specific message, scroll down to the **XPath** field. Enter an XPath expression, for example:, `// *[local-name()='Body']`.

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.

1. In the Security Settings dialog box, select the **Web Service** scenario from the **Service Details** list.
2. In the main part of the Security Settings dialog box, select the **WS-Security** tab.
3. In the WS-Security tab, click the **Add Token** button  and select **X509 Certificate Token** from the **Security Token** drop down list.
4. Enter the token details to reference your private certificate. Select **BinarySecurityToken** as a **Reference type**, since this token is used for a signature.
5. Click the **Add Token** button  again and select **X509 Certificate Token** from the **Security Token** drop down list to add another X.509 token.
6. Enter the token details to reference your private certificate. Select **Reference** as a **Reference type**, since this token is used for a encryption.

7. Click the **Add Message Signature** button  . In the drop down list, select the first X.509 token you created.
8. Click the **Add Message Encryption** button  . In the drop down list, select the second token you created.

Configure the WS-Addressing (optional)

1. In the main pane of the Security settings dialog box, click the **WS-Addressing** tab.
2. In the WS-Addressing tab, select the relevant version or **None** if WS-Addressing is not used.
3. In the **Reply to** field, provide an alternate destination.

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.

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 W3E3 service.
2. Add a method from the Web service to the canvas.
3. In the Properties pane, select the **Security Settings** tab  , or in the Toolbox pane right-click the Web service node 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.

Set up Advanced Standards testing

Relevant for: API testing only

This section provides guidelines for using UFT in advanced standards testing.

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.

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.

Known Issues- Web Service Security

Relevant for: API testing only

This section describes troubleshooting and limitations for working with Web services security.

Security on Webservices imported from UDDI	Authentication and proxy security are not supported for Web Services imported from a UDDI.
WCF-type Web services	<ul style="list-style-type: none">Configuring different security settings for operations residing on the same port is not supported.Some of the user event handlers (such as the AfterProcessRequestSecurity, BeforeProcessResponseSecurity, OnSendRequest, and OnReceiveResponse events) will not be invoked.When working with Federation type scenarios that use STS (Security Token Service), you cannot change the SOAP version.
Web services with message level security	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.
SAML security tokens	<ul style="list-style-type: none">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.

SOAP 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

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:

WS-Addressing	<p>WS-Addressing 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 replyTo 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>
----------------------	---

HTTP Receiver	<p>In the HTTP Receiver pattern, the server 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>
Web Service Publish Subscribe	<p>In the Web Service Publish Subscribe pattern, the server sends an HTTP request to the client, reversing the typical roles of the client and server. It is similar to the HTTP Receiver, 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>

Web Service Solicit Response	<p>The Web Service Solicit Response pattern is a variation of the Web Service Publish Subscribe 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>
Dual WSDL Files	<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"> Import the Request WSDL file, using the Import WSDL from import command. Import the Response WSDL file, using the Import WSDL from import command, enabling the Import as Server Response option..

For task details, see "[Test an asynchronous Web service](#)" below.

Test an asynchronous Web service

Relevant for: API testing only

This task describes how to create an API test for testing an asynchronous Web service.

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.

Known Issues- 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

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 "[Use the Wizard to create a custom Activity - C#](#)" on page 520.

Advanced users can build custom activities manually, without the wizard. For details, see "[Manually create a custom Activity in C#](#)" on page 524.

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:

"Runtime files"	The DLL that UFT invokes to run the activity.
"Signature files"	An XML file that defines the input and output properties, events, and the runtime class that executes the activity.
"Addin files"	An XML file that references all of the activity component data.
"Resource files" on page 519	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>\addons\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 "[Manually create a custom Activity in C#](#)" on page 524.

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 "[Create a runtime file](#)" on page 526.

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
---------	------------	-------------

Resource Element	type	The type of entity. In this case, the type is Activity .
	id	A unique string that identifies the activity. This string is used when writing event handler code for the activity.
	version	The version of the current addin mechanism, for example 1.0.0
	group	The parent group (in the Toolbox) pane to which to add the activity.
	shortname	The short name of the activity displayed in the hint area of the Toolbox pane.
	description	The description of the activity displayed in the hint area of the Toolbox pane.
	assembly	The DLL file to call when running the activity. This .dll file is stored in the same folder as the signature and addin files.
	className	The class implemented by the activity. This class must inherit from the STActivityBase class.
	image	An image file for the icon representing the activity. This image is stored in the same folder as the signature file.
	visible	A boolean value indicating whether to display the activity in the Toolbox pane
	xmlns	The namespace defining the schema for the signature file. Keep the default value.
	xmlns:xsi	The schema instance used for the signature file. Keep the default value.
	xmlns:Location	The URL of the signature file's schema (Signature.xsd), referenced by the name space. Keep the default value for this.

Section Element	name	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" .
	source	A boolean value indicating the source of the section.
	dest	A bollean value indicating the destination path of the section.
	checkpoint	A boolean value indicating whether to display a checkpoint checkbox in the Validate column for the activity.
	isSharedMetadata	A boolean value indicating whether to share the section's meta data with other sections.
	propertiesType	The type of the properties in the section, for example, "XML"
	showXmlControls	A boolean value indicating whether to display the Text and XPath tabs in the Input/Checkpoints tab.
	displayName	The name of the section as it will be displayed in the Properties pane.

Section Element - Sub Attributes	Tab	<p>The tabs to display in the Properties pane for the activity. This sub-element can include the following attributes:</p> <ul style="list-style-type: none"> • name. The internal name of the tab. Some of the built-in ones are General, InputOutput, Events, Attachments, and SOAPFault. • id. 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. • CanBeInToolbox. A boolean value indicating if the activity can be shown in the Toolbox pane's toolbar • CanBeInPropertySheet. A boolean value that indicates if the activity's tab is displayed in the Properties pane. • CanBeInDataLinkDialog. A boolean value that indicates if the activity can be displayed in the Select Link Source dialog box. <p>To use the default tabs - General, Input/Checkpoints, and Events, 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 Tabs sub-element and specify the desired tabs.</p>
	Alert	<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"> • constraint. The reason to show the alert, for example, NullValueConstraint. • target. The Xpath of the property to which to apply the constraint. • section. The internal name of the section containing the properties. • type. The type of alert, such as <code>error</code> or <code>warning</code>

	Events	<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">• name. The internal name of the event. Use one of the built-in names or define a custom one.• CodeCheckpointEvent. Enables you to create an event handler to run when the test is verifying checkpoints.• BeforeExecuteStepEvent. Enables you to create an event handler to run before executing the activity.• AfterExecuteStepEvent. Enables you to create an event to run after executing the activity.• <custom event>. A custom event that you define.• description. A textual description of the event.• eventArgs. The source of the arguments for the event. The standard argument for BeforeExecuteStepEvent and AfterExecuteStepEvent event is STActivityBaseEventArgs. The built-in value for the CodeCheckpointEvent is CheckpointEventArgs. <p>To access the default events: CodeCheckpoint, BeforeExecute, and AfterExecute, you need to include only the Events tab in the Tab sub-element, but you do not need to use the Events 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
Elements/subelements The elements and attributes are defined in the standard XML schema file, http://www.w3.org/2001/XMLSchema , 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.	xs:schema	The schema namespaces for the properties, as described by the xml:ns attribute. Keep the default value for this attribute.
	xs:import	The namespace to import using the namespace and schemaLocation attributes. Keep the default value for this attribute.
	xs:element	The element to define, using the attributes described in the table below.
	xs:simpleType	A tag indicating the beginning of definitions of a simple type property. 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 xs:simpleType tag. <pre><xs:element name="ClientCertificate" type="types:Certificate" types:displayName="Client certificate" /></pre>

	xs:complexType	A tag indicating the beginning of definitions for a node of multiple properties.
	xs:sequence	A tag indicating the beginning of a list of properties in a complex type property.
	xs:restriction	A tag restricting the value of the enumeration values of a property, using the base attribute. To restrict String type values, use base="xs:string" .
	xs:enumeration	A tag indicating the beginning of list of values in the drop-down list for a property, using the value attribute.
	xs:annotation	An annotation for the element Use an xs:documentation sub-element to compose text that will appear below the properties grid in the Properties pane.
Element Attributes The following table describes the primary attributes of the xs:element . For attributes in the standard XML schema, use an xs: prefix in the value, for example standard types use type=xs:string or type=xs:int . For types defined in the Types.xsd schema, use a types: prefix in the attribute name. For example types:displayName .	name	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 Name column.

	type	The type of property. Some common values are: xs:string , xs:int , xs:boolean , Multipart , Header , Part . For a value defined in the Types schema, use the types: prefix. For example type="types:filePath" .
	minOccurs	The minimum number of array elements for which the user must provide. For none, specify " 0 ".
	maxOccurs	The maximum number of array elements the user may provide. To allow an unlimited amount, specify " unbounded "
	types:visible	When true , enables the parameter to be visible even before being expanded by the Add Array Element command.
	types:argType	The type of the property: " XML " or " Object ".
	types:displayName	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.

```
<xss:element name="Status" default="Done" types:displayName="Status">
    <xss:simpleType>
        <xss:restriction base="xss:string">
            <xss:enumeration value="Done"/>
```

```
<xs:enumeration value="Pass"/>
<xs:enumeration value="Fail"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
```

Complex Array Elements

The following sample defines a complex property, with a `Key` and `Value` pair of values.

```
<xs:complexType name="NameValueType">
<xs:sequence>
<xs:element name="Key" type="xs:string"/>
<xs:element name="Value" type="xs:string"/>
</xs:sequence>
</xs: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
---------	---------	-------------

Addin Section	<Addin>	The basic details about the section, including any of the following attributes: <ul style="list-style-type: none"> • name. the name of the addin. • author. the creator of the activity. • copyright. the full path of a text file with the copyright information. • description. a textual description of the activity. • version. The addin file version, set to 1.0.
	Path	Details about the location of the activity, including any of the following attributes: <ul style="list-style-type: none"> • name. The logical path scanned by the framework, to identify addons. The physical location of this folder is addons\CustomerAddins<addin_name>
	Activity (sub-element of the Path attribute)	Additional information about the activity, including any of the following attributes: <ul style="list-style-type: none"> • id. An identifying string corresponding to the ID in the signature file. • displayName. The activity's display name in the Toolbox pane. • signatureFile. The name of the XML signature file.
Manifest Section	Identity	Basic details about the addin, including any of the following attributes: <ul style="list-style-type: none"> • name. The activity name corresponding to the ID in the signature file. When referring to this addin as a dependency, use this name.
	Dependency	The activity on which the current activity is dependent, including any of the following attributes: <ul style="list-style-type: none"> • addin. The identity name of the dependent activity, from the name attribute in the Manifest section of its addin file. • requirePreload. A boolean value indicating whether to preload the dependent addin before loading the current one.

Runtime Section	Import	<p>An assembly to import when running the activity.</p> <p>You can use the assembly, which is the name of an assembly. Use the DLL name without the DLL extension.</p> <p>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.
```

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.

Run the Activity Wizard

1. Open the Activity Wizard (**Start > All Programs > HP Software > HP Unified Functional Testing > Tools > Activity Wizard** or **<UFT installation folder>\bin\ActivityWizard.exe**).
2. In the wizard's **General Properties** pane, select the **C#** as the **Language**.
3. Follow the steps of the wizard to create your activity.

Add execution code

1. 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.

2. 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 //*****
...
```

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");
...
}
```

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", "<BR>");
    this.Report("Message", DetailsReport);      ;
...
}
```

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.

Deploy the activity in UFT

1. In the final wizard screen, click **Deploy in UFT**.
2. Click **Finish** to close the wizard and restart UFT.

Use the Wizard to create a custom Activity - Java

Relevant for: API testing only

This task describes how to create a new activity using Java code, and deploy it in UFT.

Prerequisite

Make sure you have a **JAVA_HOME** environment variable defined on your machine indicating the parent JDK folder.

Run the Activity Wizard

1. 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**).
2. In the wizard's **General Properties** pane, select the **Java** as the **Language**.
3. Follow the steps of the wizard to create your activity.

Edit the code

1. 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.

2. 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;
}
...
```

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;
}
```

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;
}
```

Compile the Java into a class

1. In your design IDE, add the **ServiceTestCall.jar** file to the build path.
2. 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.

Deploy the activity

1. In the final wizard screen, click **Deploy in UFT**.
2. Click **Finish** to close the wizard and restart UFT.

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.

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 "[Create a runtime file](#)" on page 526.

Create a signature file

1. 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.
2. 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"
    >
```

3. Add the required sections, such as GeneralProperties, InputProperties, Tabs, Events and so forth

4. 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.

5. Specify any external resource files.

6. Close the file with the **</Resource>** tag.

For more details about the structure of the signature file, see "[Signature files](#)" on page 509.

Create an addin file

1. Create a new file with an **.addin** extension in the **<installation directory>\addins\CustomerAddins\<addin_name>** folder, together with the signature file.
2. 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>
```

3. Create a unique **Addin** file for each activity—do not define multiple activities in a single **Addin** file.
4. Define post-build tasks such as **resgen**.
5. Compile the project and copy the DLL to the **<installation_Folder>\addins\CustomerAddins\<addin_name>** folder.

For additional details, see "[Runtime files](#)" on page 508.

Provide a graphic for your activity - optional

1. Copy an icon image for your activity into the **<installation_Folder>\addins\CustomerAddins\<addin_name>** folder. This file should meet the following requirements:
 - a .png extension
 - sized at 16 x 16 pixels
 - 8-bit color depth
2. Specify the name of the image file in the signature file's **Resource Element**.

Check the implementation

1. Reopen the application and drag the new activity into the Test Flow. Verify that the activity and its properties appear as expected.
2. Provide property values.
3. Run the test and observe the Output log and the run results.
4. Enable checkpoints to verify the results and rerun the test.

Create a runtime file

Relevant for: API testing only

Note: You must compile the DLL with a **Target Framework** of Framework 4.0.

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;
```

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
    {
```

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 670](#).

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;
}
```

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.

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);
    }
}
```

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", "<BR>");
    DetailsReport = DetailsReport.Replace("\n", "<BR>");
    this.Report("Message", DetailsReport);
```

If you defined a custom event, invoke it after the call to **ExecuteStep**.

```
...
protected override STExecutionResult ExecuteStep()
{
    InvokeCustomerEvent();
}
```

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);
}
```

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.

Known Issues - 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: The Editor

Relevant for GUI tests and components

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](#)" below and "[Automatic code completion](#)" on page 535.
- **Bookmarks.**
- **Search and Replace.** For details, see "[Searching and replacing](#)" on page 536
- **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 in the Text Editor tab of the Options dialog box (**Tools > Options > Text Editor** tab > **General** node.)

For a description of using the editor for GUI tests and components, see "[Programming Tests](#)" on page 542. For a description of using the editor for API tests, see "[Event Handlers for API Test Steps](#)" on page 616.

Statement completion

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

Statement completion enables you to increase programming speed and accuracy when typing in the Editor by providing dynamic lists of items in the form of tooltips, drop-down lists, or popup windows.

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.

The available information differs depending on what character you type:

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 ","	<p>The operation syntax, bolding the next argument for which you need to enter a value.</p> <p>Relevant if you enter a comma after any argument value other than the last one in a step.</p> <p>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.</p>
An operation or function name	Ctrl+Shift+Space or Select Edit > Format > Argument Info	The statement completion (argument syntax) tooltip for that item.

If you enter:	Followed by:	UFT displays:
Ctrl+Space or Edit > Format > Complete Word		A dynamic list of the relevant: <ul style="list-style-type: none">• operations• properties• user-defined functions• constants and local variables relevant to the current programming scope• functions and methods relevant to the current programming scope• options relevant to your GUI test or component, including:<ul style="list-style-type: none">• test objects• utility objects• collections <p>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.

Automatic code completion

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 "[Use code snippets and templates](#)" on page 538.

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

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.

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:

GUI tests and scripted components	<p>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.</p> <p>When you search in scripted components, the search is performed in any defined function libraries.</p>
API tests	<p>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.</p> <p>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.</p> <p>You can search for the following types of items in API tests:</p> <ul style="list-style-type: none">• 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 details on searching and replacing, see "[Search for references or classes](#)" on [page 540](#).

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.

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 Edit > Code Snippet menu	SPACE	Inserts the first VBScript snippet defined for the keyword you entered, as listed in the Edit > Code Snippet menu.
The keyword for a code template defined in the Options dialog box	TAB	Inserts the code snippet defined in the relevant template in the Code Templates Pane. If multiple templates are defined for the keyword you entered, the first template in the list is inserted.

If you enter:	Followed by:	UFT does the following:
Part of a code template keyword	CTRL+SPACE	<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 Edit > Code Snippet 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.

Search for references or classes

Relevant for: User code files

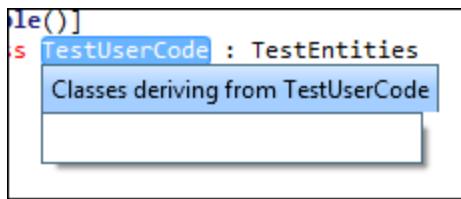
Search for references to the currently selected function or method

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.
2. Select a class and then select **Search > Find Derived Symbols**.

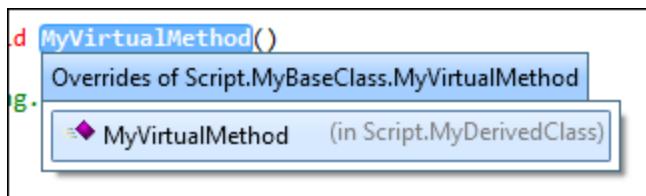
The derived classes are displayed in a small drop-down box under the selected class. For example:



Search for methods that override a virtual method

1. Create a new user code file, or open an existing one.
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.
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:



Known Issues - Statement Completion

Relevant for: GUI actions, scripted GUI components, and function libraries

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 Tests

Relevant for: GUI tests, 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 "[The Editor](#)" on page 533.

To learn about working with VBScript, you can view the VBScript documentation available from the **UFTHelp** 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.

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.

However, you can also instruct UFT to perform operations on objects without referring to the object repository or to the object's name. 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 or a file containing an image of the control as a description property (for an Insight object).

Such a **programmatic description** can be very useful in a number of scenarios:

The object is not stored in the object repository	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.
A number of objects have common identical properties	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.
The object is created dynamically during the run session	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.

The object differs between different versions of the application	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.
---	---

**Example:**

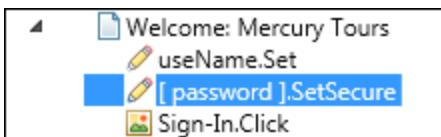
Suppose you are testing a Web site that generates a list of potential employers based on biographical information you provide, and offers to send your resume to the employer names you select from the list. You want your test to select all the employers displayed in the list, but when you design your 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**.

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](#)" on the next page.
- **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 548.

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.



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:

TestObject	The test object class.
PropertyName:=PropertyValue.	<p>The identification property and its value. Each property:=value pair should be separated by commas and quotation marks.</p> <p>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>).</p>

For example, 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
```

Note the following special scenarios for static programmatic descriptions:

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.
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: <pre>MyVar="some text string" Browser("Hello").Page("Hello").Webtable("table").Webedit("name:=" & MyVar)</pre>

**Finding
parent
test
objects
with
progra
mmatic
descript
ions**

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.

**Example:**

- 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"
```

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: <ul style="list-style-type: none">• 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):<ul style="list-style-type: none">• 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 *UFT Object Model Reference for GUI Testing*.

Note the following when using dynamic programmatic descriptions:

Regular expressions and programmatic descriptions	<p>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" on page 357.</p> <p>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 <i>UFT Object Model Reference for GUI Testing</i>.</p>
--	---

Programmatic descriptions and the object hierarchy	<p>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.</p> <p>For example, you can use Browser(Desc1).Page(Desc1).Link(desc3), since it uses programmatic descriptions throughout the entire test object hierarchy.</p> <p>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).</p> <p>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).</p>
---	--

Retrieving child objects with programmatic descriptions

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)
```

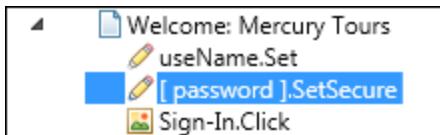


Example:

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 *UFT Object Model Reference for GUI Testing*.

Using the Index property

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
```

Opening and closing applications

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 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:

- 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.
- 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
```

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.

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](#)" on page 563
- "[Calculations](#)" on page 567
- "[For...Next Statement](#)" on page 571
- "[For...Each Statement](#)" on page 570
- "[Do...Loop statement](#)" on page 569
- "[While...Wend Statement](#)" on page 573
- "[If...Then...Else Statement](#)" on page 571
- "[With Statement](#)" on page 573

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 **GetTOProperty**, **GetTOProperties**, and **SetTOProperty** methods in your action, component, or function library to set and retrieve the identification property values of the test object.

- The **GetTOProperty** and **GetTOProperties** methods enable you to retrieve a specific property value or all the properties and values that UFT uses to identify an object.
- The **SetTOProperty** 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 **SetTOProperty** 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").SetTOProperty  
"Name", "my button"  
ButtonName=Browser("QA Home Page").Page("QA Home Page").WebButton  
("Submit").GetTOProperty("Name")
```

- You use the **GetROPProperty** 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").GetROPProperty("href")
```

For a list and description of identification properties supported by each object, and for more details on the **GetROPProperty**, **GetTOProperty**, **GetTOProperties**, and **SetTOProperty** methods, see the **Common Methods and Properties** section of the *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")
```

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.

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:

Retrieving native properties	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
```

Activating native operations

Use the **Object** property to activate the internal operations of any runtime object. For example, you can activate the native **focus** method of the edit box as follows:

```
Dim MyWebEditSet  
MyWebEdit=Browser("Mercury Tours").Page("Mercury Tours").WebEdit  
("username").Object  
MyWebEdit.focus
```

After you insert the **.Object** method, 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*.

For more details on the **Object** property, see your object's properties in the *UFT Object Model Reference for GUI Testing*.

Use the Windows API in test steps

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.

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_HWNDDNEXT**, **GW_HWNDDPREV** 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_HWNDDLAST    1
#define GW_HWNDDNEXT    2
#define GW_HWNDDPREV    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.

See also:

• General syntax rules and guidelines	560
• Formatting text	562
• Comments	563
• Parameter indications	564
• Parentheses	565
• Calculations	567
• Variables	568
• Do...Loop statement	569
• For...Each Statement	570
• For...Next Statement	571
• If...Then...Else Statement	571
• While...Wend Statement	573
• With Statement	573

General 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:

Case-sensitivity	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. For example, the two statements below are identical in VBScript: <pre>Browser("Mercury").Page("Find a Flight:").WebList("toDay").Select "31" browser("mercury").page("find a flight:").weblist("today").select "31"</pre>
-------------------------	---

Text strings	<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 argument (specifying the timeout) is a numeric value, which also does not need quotation marks.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"><pre>Browser("Mercury").Page("Find a Flight:").WaitProperty("items count", Total_Items, 2000)</pre></div>
Variables	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 "Variables" on page 568 .
Parentheses	To achieve the desired result and to avoid errors, it is important that you use parentheses () correctly in your statements. For details, see "Parentheses" on page 565 .
Indentation.	You can indent or outdent your script to reflect the logical structure and nesting of the statements. For details, see "Formatting text" on the next page .
Comments.	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 "Formatting text" on the next page , and "Comments" on page 563 .
Spaces.	You can add extra blank spaces to your script to improve clarity. These spaces are ignored by VBScript.

Reserved Words. Certain words are reserved by UFT or VBScript. You cannot use these words as variables, constants, or procedure names.

- Reserved 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).
- VBScript reserved words can be found in various online VBScript guides.

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:

```
Set Window = Window("Calculator")
```

or

```
WinButton = Window("Calculator").GetROProperty("hwnd")
```

A run error also occurs when running the following statement because it uses a reserved F-key as a variable:

```
Set F1 = CreateObject("Scripting.FileSystemObject")
```

Formatting 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	<p>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" below.</p> <ul style="list-style-type: none">• 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. 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. You can remove the comments from a selected block or line of text by choosing Edit > Format > Uncomment.
Indentation	<p>Use indentation to reflect the logical structure and nesting of your statements.</p> <ul style="list-style-type: none">• 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).• 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

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).



Tip:

- You can comment or uncomment a block of text by selecting **Edit > Format > Comment/Uncomment**.
- 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

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
Object_Hierarchy	The hierarchical definition of the test object, consisting of one or more objects separated by a dot.
Method	The name of the method that UFT executes on the parameterized object.
DataTable	The reserved object representing the data table.
parameterID	The name of the column in the data table from which to take the value.

Item	Description
<code>sheetID</code>	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.

For more details on using and defining parameter values, see "[Parameterizing Object Values](#)" on page 330.

Parentheses

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 **O** 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

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

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").GetROProperty("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.

Example: 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)
```

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
counter	The variable used as a counter for the number of iterations.
start	The start number of the counter.
end	The last number of the counter.
step	The number to increment at the end of each loop. Default = 1. Optional.
statement	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.

The **With** statement has the following syntax:



Example: **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: 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 "[Generate With statements](#)" on page 602.

Chapter 51: User-Defined Functions

Relevant for GUI tests and components

Note: The terms function, method, and operation are used interchangeably in this chapter.

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.

Associated function libraries

Relevant for: GUI tests and components

After you create your function libraries, you associate them with your test or application area. This enables you to insert a call to a public function or subroutine contained in the associated function library from a test or 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.

For details, see "[Manage function library associations](#)" on page 585.

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. 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.

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.

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 or you want to add additional functionality, you may want to create a **user-defined function**.

You create the functions in VBScript.

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.

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: <ul style="list-style-type: none"> 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. <p>For more details, see "Registered user-defined functions" on the next page and "Create and register a user-defined function using the Function Definition Generator" on page 591.</p>

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**).

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.

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:

For tests	<p>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.</p> <p>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:</p> <ul style="list-style-type: none">• 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. <p>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:</p> <ul style="list-style-type: none">• 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.
For components	<p>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.</p> <p>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).</p> <p>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.</p>

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 589.

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 591](#)".

In certain situations, you must pay special care when unregistering user-defined methods:

Unregistering functions for reusable actions	<p>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.</p> <p>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.</p> <p>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.</p>
---	---

**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.



Example: 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 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..

The following table describes the differences between using each of these statements:

LoadFunctionLibrary	ExecuteFile
<p>In a test: After you run a LoadFunctionLibrary statement, the functions in the file are available to your entire test, until the end of the run session.</p> <p>In a component: LoadFunctionLibrary works in the same way as ExecuteFile. After you run the statement, the functions in the file are available only within the scope of the calling component.</p>	<p>After you run an ExecuteFile statement, you can call the functions in the loaded file only within the scope of the calling action or component.</p>

LoadFunctionLibrary	ExecuteFile
LoadFunctionLibrary enables you to debug the functions in the function library during run-time.	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.

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.

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.

View the list of associated function libraries

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

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.
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 using the Test Settings dialog box

1. Create or open a test.
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 the Solution Explorer pane

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.

Associate a function library with an application area

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

- 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. 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

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.)

Load a function library dynamically during a run session

Add a the **LoadFunctionLibrary** statement to your action, scripted component, or associated function library.



Tip: To include the same **LoadFunctionLibrary** statement in every action you create, you can add the statement to an action template.

Create and work with a user-defined function

Relevant for: GUI tests and components

Prerequisites - Open the function library or test

1. 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.
2. 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.

Create the function

You can define functions manually or using the Function Definition Generator, which creates the basic function definition for you automatically.



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:

The screenshot shows the HP Unified Functional Testing interface. On the left, there's a tree view with a node labeled 'Main'. Below it, a code editor window displays the following VBA-like script:

```
1  '@Description This function displays a Hello World message box.
2  Function Hello_World
3
4      MsgBox "Hello World"
5  End Function
6
7  h
```

To the right of the code editor, a tooltip window appears, containing the description: "This function displays a Hello World message box." A scrollable list box below the tooltip contains several items, with "Hello_World" highlighted in blue.

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.

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:

Manually	<p>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.</p> <p> Example:</p> <pre>RegisterUserFunc "WebEdit", "MySet", "MySetFunc", True</pre> <p>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).</p> <p>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.</p>
Using the Function Definition Generator	<p>If you use the Function Definition Generator Dialog Box 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.</p> <p>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.</p>

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.

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.

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.

Navigate to the function's definition - optional

You can navigate directly from a function call to the function's definition.

1. In the Editor, in an action, click in the step containing the relevant function.
 2. 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.

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.

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.

Create and register a user-defined function using the Function Definition Generator

Relevant for: GUI tests and components

Open the function library/test and the Function Definition Generator

1. Make sure that the function library or action in which you want to insert the function definition is the active document. This is because the Function Definition Generator inserts the function in the currently active document after you finish defining it.
2. Select **Design > Function Definition Generator**.

Specify the details for the function definition

1. In the **Function definition** section, give the function a unique name.
2. Select the type for the function: **Function** or **sub** (subroutine).
3. Specify the type for the function:
 - **Public:** The function can be called by any test or component (via the application area) associated with the function library
 - **Private:** The function can be called only from this function library.

Note: If you are overriding an existing test object class method, select the **Public** option.

Register the function to a test object class - optional

1. Below the function name, select the **Register to a test object** check box.
2. Select the test object from the list of available objects.
3. 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.
4. If necessary, 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. You can also add additional **RegisterUserFunc** statements, to register the function to additional test object classes.

Add arguments to the function - optional

1. In the Arguments box, click the **Add** button  to add the necessary arguments for the function.
2. For each argument, in the Pass mode cell, specify how the value is passed to the function:
 - **By value:** A value entered into the function
 - **By reference:** The function references the value.

Add documentation details to the function - optional

In the Additional information box, enter the details of the function:

- **Description:** The description of the function displayed as a tooltip when you hover over the function name.
- **Documentation:** The description of the function displayed in the Documentation column of the Keyword View after you enter the necessary arguments.

Insert the function in your active document

Click **OK** or **Insert**. UFT inserts the generated VBScript code in the active document.

Add the content (code) of the function

To finalize the function, add content to the function code, as required, replacing the TODO comment.

The function is now available to your tests, components, or function libraries (depending on where the function was generated and the context in which you are working). If you registered the function to a test object, this function is displayed in the list of available functions for the test object (in the Keyword View and Editor).

Known Issues- Function Libraries and user-defined functions

Relevant for: GUI tests and components

Defining a VBScript class in a function

If you define a VBScript class, it can be called only within the UFT action or function library in which you defined it.

Workaround:

1. You can use an **ExecuteFile** statement to call a VBScript class defined in an external function library.
2. 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**.

Calling a test object method that has been overwritten by a function	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.
User-defined functions in the run results	<p>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.</p> <p>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.</p>
Partial test runs and method registration	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.
Using the Option Explicit method	<ul style="list-style-type: none"> 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. <p>In test actions, you can use Option Explicit statements directly without any restrictions.</p>
Unique variables in function libraries	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 <code>Dim</code> 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 <code>Dim</code> statement only in the last function library (since function libraries are loaded in the reverse order).

Class definitions in a function library	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.
Modifying function libraries	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.
Cyrillic characters in function libraries	Function libraries that contain the Cyrillic 'њ' character and are saved in ANSI encoding may be interpreted incorrectly in UFT. For example, the 'њ' 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

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.

Message statements

Relevant for: GUI tests and scripted GUI components

Message statements 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.

```
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.

```
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:

micPassed	Causes this step to pass and sends the message to the report.
micFailed	Causes this step (and therefore the test) to fail, and sends the message to the report.
micDone	Sends a message without passing or failing the step.
micWarning	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.

MessageBox VBScript function	Displays a message that pauses the run session until the message box is closed.
Print Utility statement	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

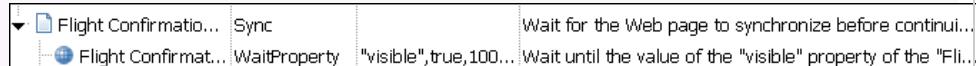
Handle these anticipated timing problems by synchronizing your test to ensure that UFT waits until your application is ready before performing a certain step.

There are several options that you can use to synchronize your test:

Add a synchronization point	If you do not want UFT to perform a step or checkpoint until an object in your application achieves a certain status, insert a synchronization point to instruct UFT to pause the test. 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. 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.
------------------------------------	---



Example: After you insert a synchronization point for the **Flight Confirmation** button, it may look something like this:



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	<p>Use Exist and/or Wait statements to instruct UFT to wait for a window to open or an object to appear. You can combine these statements within a loop to instruct UFT to wait until the object exists before continuing with the test.</p> <p>For example, the following statements instruct UFT to wait up to 20 seconds for the Flights Table dialog box to open.</p> <pre>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</pre> <p>For details, see Add Synchronization Point Dialog Box.</p>
Timeout Settings	<p>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, increase the default object synchronization timeout values for your test and the browser navigation timeout values for your test.</p> <ul style="list-style-type: none">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.

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 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.

Generate With statements

Relevant for: GUI tests and scripted GUI components

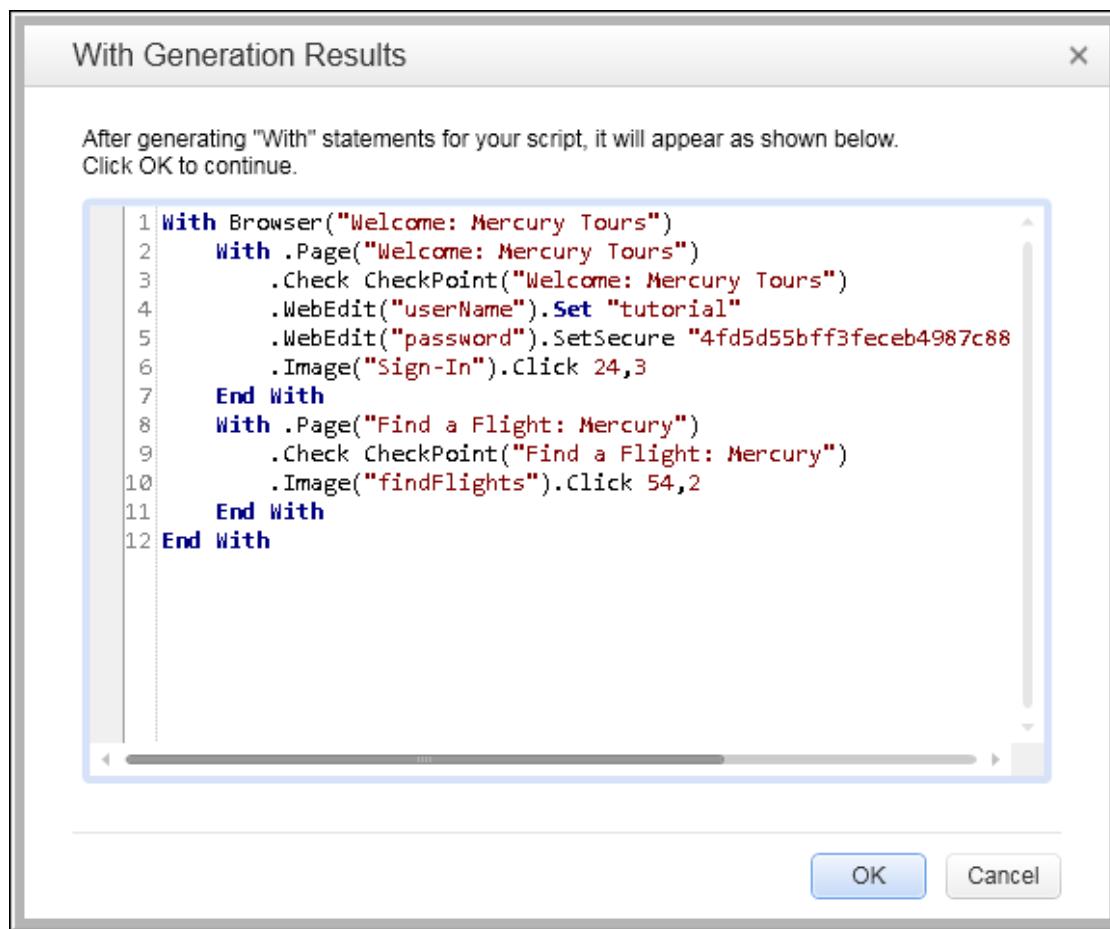
Instruct UFT to generate With statements while recording

1. In the **General** pane of the GUI Testing tab of the Options dialog box (**Tools > Options > GUI Testing tab > General tab**), select the **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.
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

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.

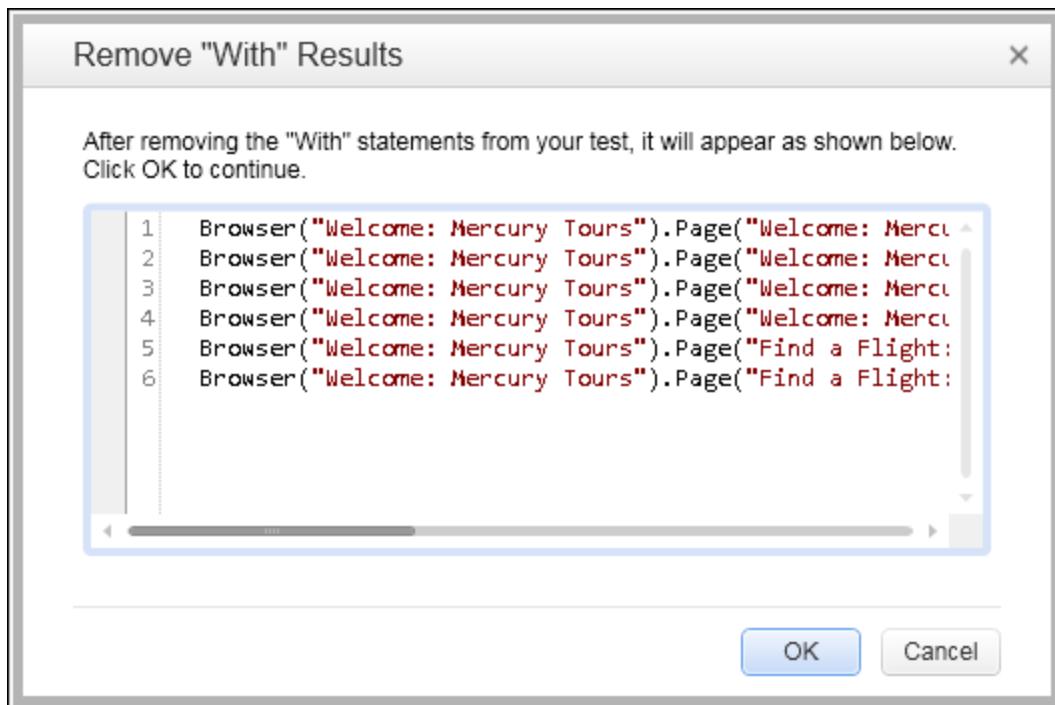
To confirm the generated results, click **OK**. The **With** statements are applied to the action.



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

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.

Chapter 53: UFT Automation Scripts

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: 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.** The script automatically starts UFT and configures the options and the settings required for testing a specific environment.
- **Maintaining your tests and components.** The script iterates over your collection of tests and components to accomplish a certain goal. For example:
 - **Updating values.** The script 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 to a new version of UFT, the new version 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.** 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 **UFT Application** 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 188, "[Global Options](#)" on page 98, and "[Document Settings](#)" on page 100.

Create a UFT automation script

Relevant for: GUI tests and components

Prerequisites

Decide whether to use UFT Automation Scripts	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.

Create the Application object

The procedure for creating the **Application** object differs slightly from language to language. Below are some examples for creating the UFT **Application** object and starting UFT in visible mode, using different programming languages

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
```

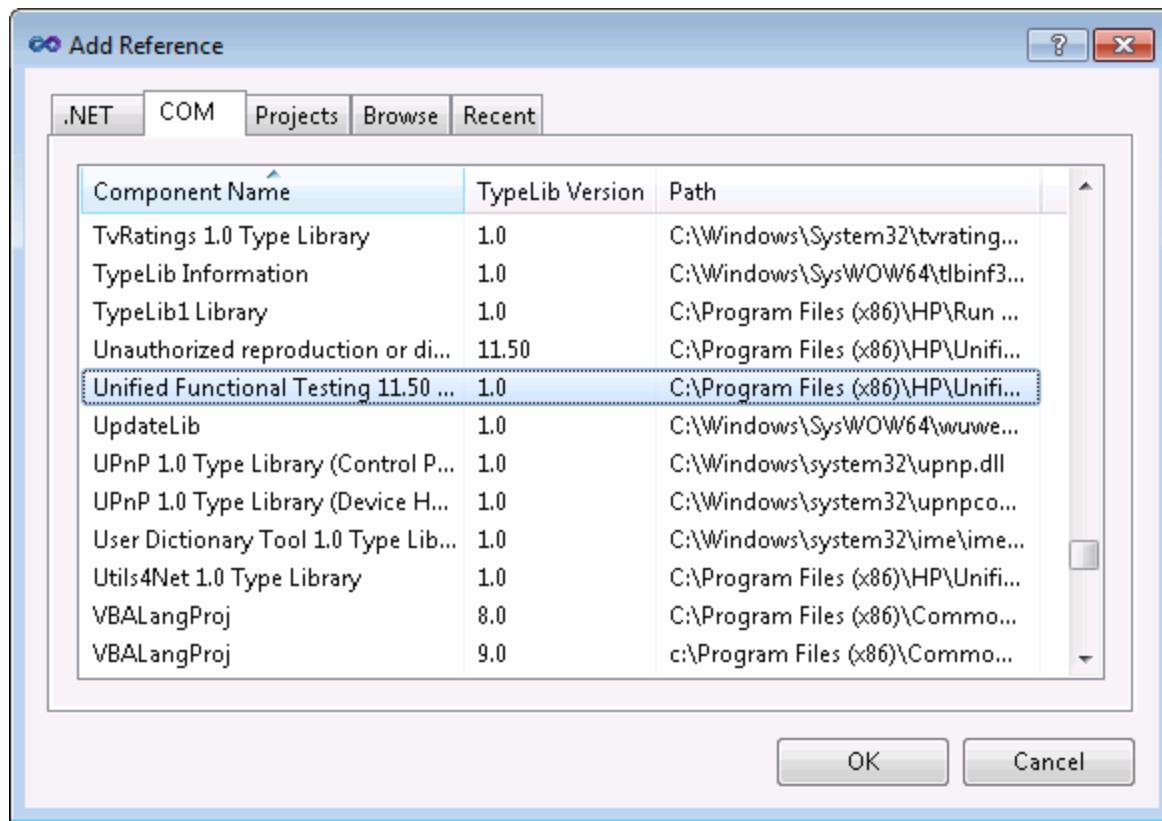
```
// Declare the application pointer
QuickTest::_ApplicationPtr spApp;
// Create the application object
spApp.CreateInstance("QuickTest.Application");
spApp->Launch(); // Launch the application
spApp->Visible = VARIANT_TRUE; // Make it visible
```

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).



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 608.

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.

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
```

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.

Set DCOM Configuration Properties on the Remote Computer

UFT automation enables UFT to act as a COM automation server. 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.

1. On the computer where you want to run the automation script, select **Start > Run**. The Run dialog box opens.
2. 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.
3. 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.)
4. In the QuickTest Professional Automation Properties dialog box, click the **Security** tab.
5. In the launch permissions section, select the custom option and click **Edit**.
6. 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.
7. 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.
8. In the QuickTest Professional Automation Properties dialog box, click the **Identity** tab and select the **interactive user** option.
9. Click **OK** to save the QuickTest Professional Automation Properties settings.
10. Click **OK** to close the Distributed COM Configuration Properties dialog box or the Component Services window.

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.

Known Issues- Automation Scripts

Relevant for: GUI tests and components

This section describes troubleshooting and limitations for automation scripts.

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: Event Handlers for API Test Steps

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.

Writing code for API test events

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 test events structure](#)" on page 662.

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](#)" on page 628.



Example:

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 623
- "Standard application" on page 626

Web Service application

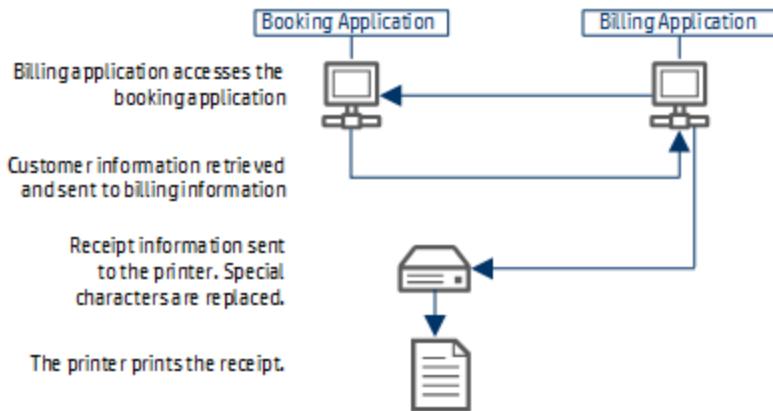
Scenario

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

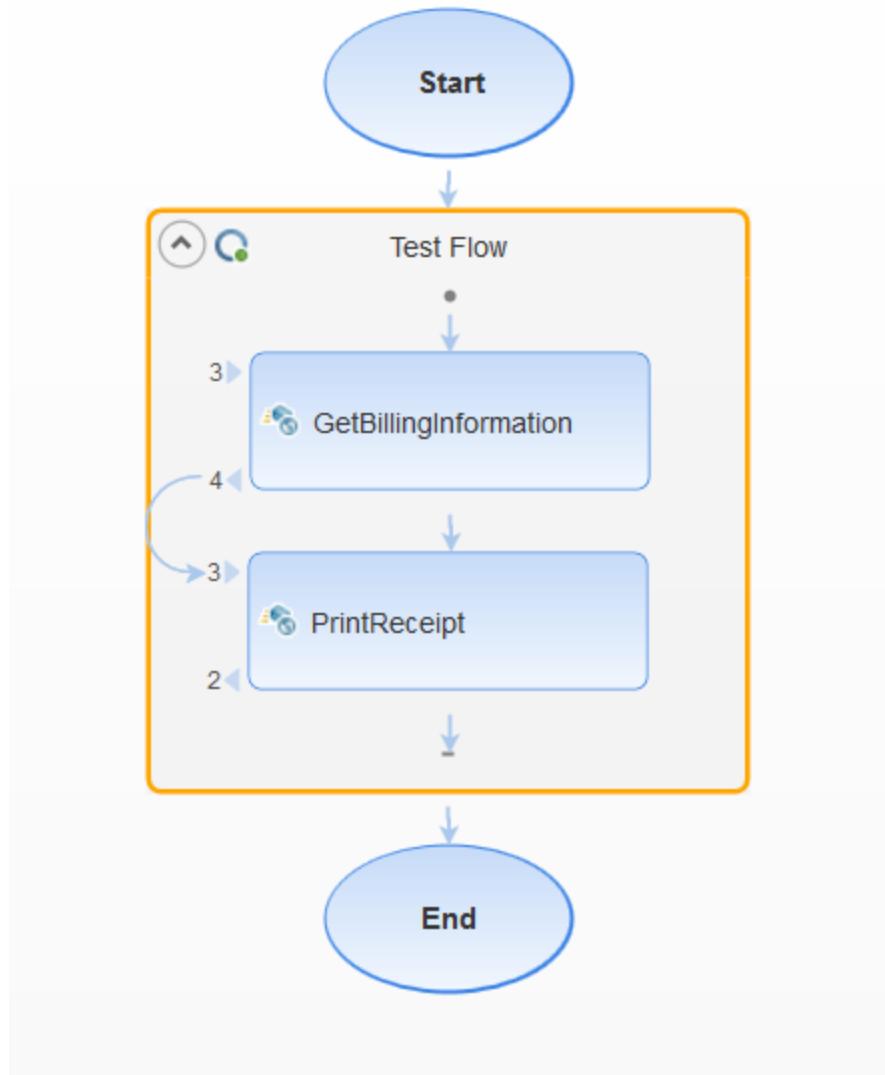
The billing application has the following workflow:



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:

Input Properties	Web service address for the hotel booking application.
-------------------------	--

Output Properties	<ul style="list-style-type: none"> • <i>Customer Name</i> • <i>Customer credit card number</i> <p>These properties are contained in a <code>description</code> element in the Web service response.</p> <ul style="list-style-type: none"> • <i>Price</i>
Checkpoints	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:

Input Properties	<i>Description</i> . This property is linked to the data source imported in the first step. <i>Price</i> . The total price to charge to the customer.
Output Properties	Response file containing the receipt from the
Checkpoints	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

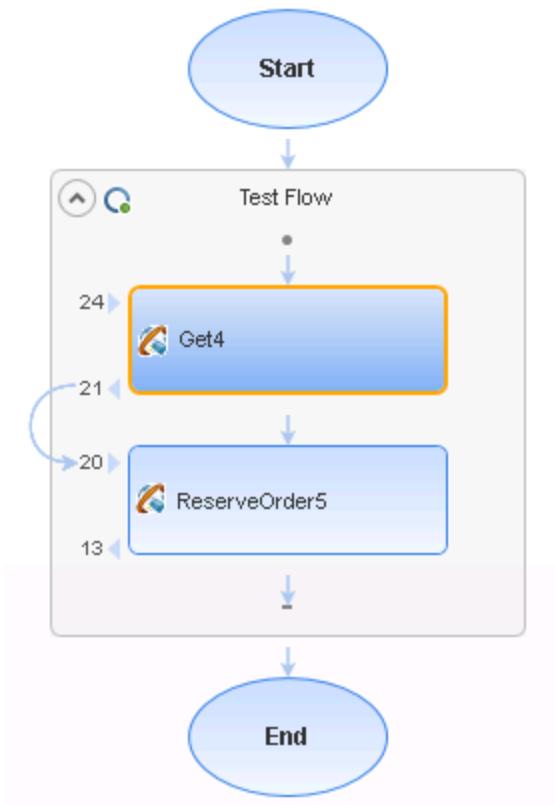
The billing application has the following workflow:

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:

Input Properties	<ul style="list-style-type: none"> • <i>Airlines</i>. • <i>ArrivalCity</i> • <i>ArrivalTime</i> • <i>DepartureCity</i> • <i>DepartureTime</i> • <i>FlightNumber</i>
Output Properties	<ul style="list-style-type: none"> • <i>Class</i> • <i>DepartureDate</i> • <i>FlightNumber</i> <p>Note: The output properties are defined by importing a ResponseXML file, which details the required response parameters.</p>
Checkpoints	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:

Input Properties	<ul style="list-style-type: none"> • <i>Class</i>. This property is linked to the <i>Class</i> output property of the <code>Get</code> step. • <i>CustomerName</i> • <i>DepartureDate</i>. This property is linked to the <i>DepartureDate</i> output parameter from the <code>Get</code> step. • <i>FlightNumber</i>. This property is linked to the <i>FlightNumber</i> output property from the <code>Get</code> step. • <i>NumberofTickets</i>
Output Properties	<ul style="list-style-type: none"> • <i>OrderNumber</i> • <i>TotalPrice</i>

Checkpoints	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.

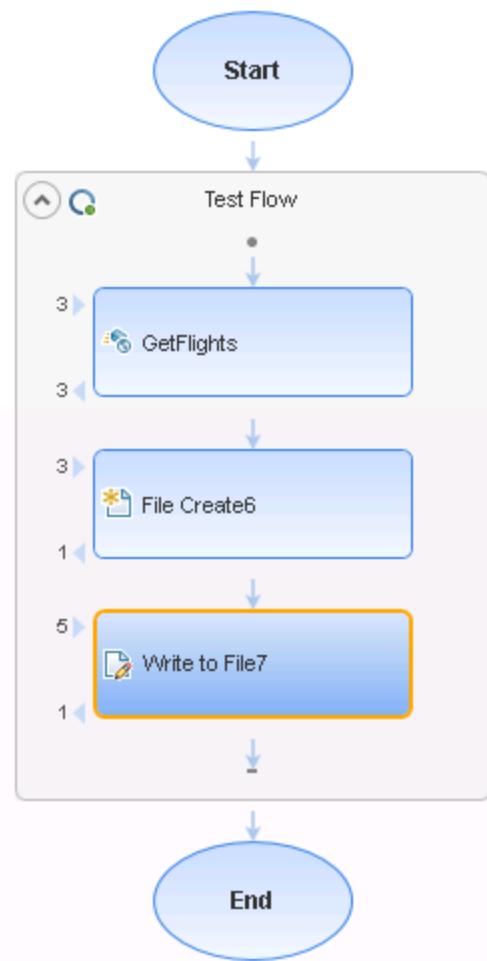
Application Flow

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:

Input Properties	<ul style="list-style-type: none">• <i>Folder Path</i> This property is linked to the <i>Class output</i> property of the <i>Get</i> step.• <i>File Name</i> <p>Both of these properties are entered manually in the test.</p>
------------------	---

Output Properties	None
Checkpoints	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:

Input Properties	<ul style="list-style-type: none"> • <i>File Path</i>. This property is linked to the <i>Folder Path</i> property of the <code>CreateFile</code> 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. • <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.
Output Properties	None
Checkpoints	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

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.

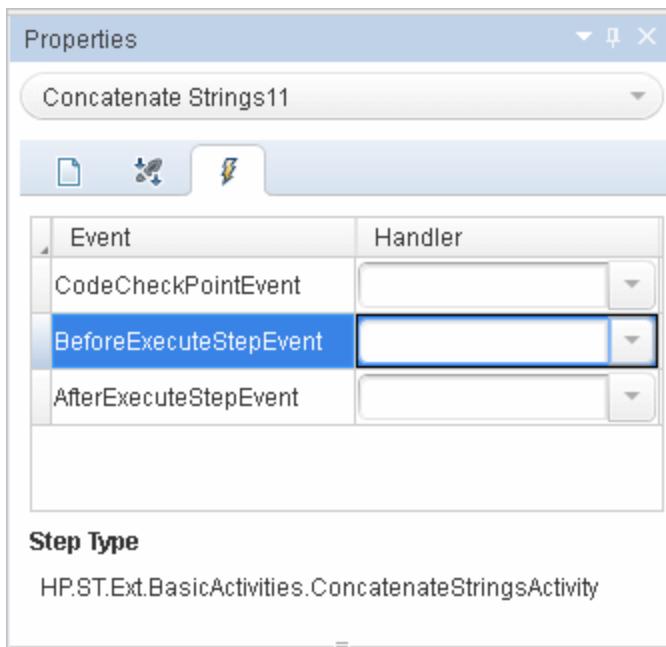
Open a window for writing custom code

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.

Open the Events tab

1. In the canvas, select an activity.
2. 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.

Select an event

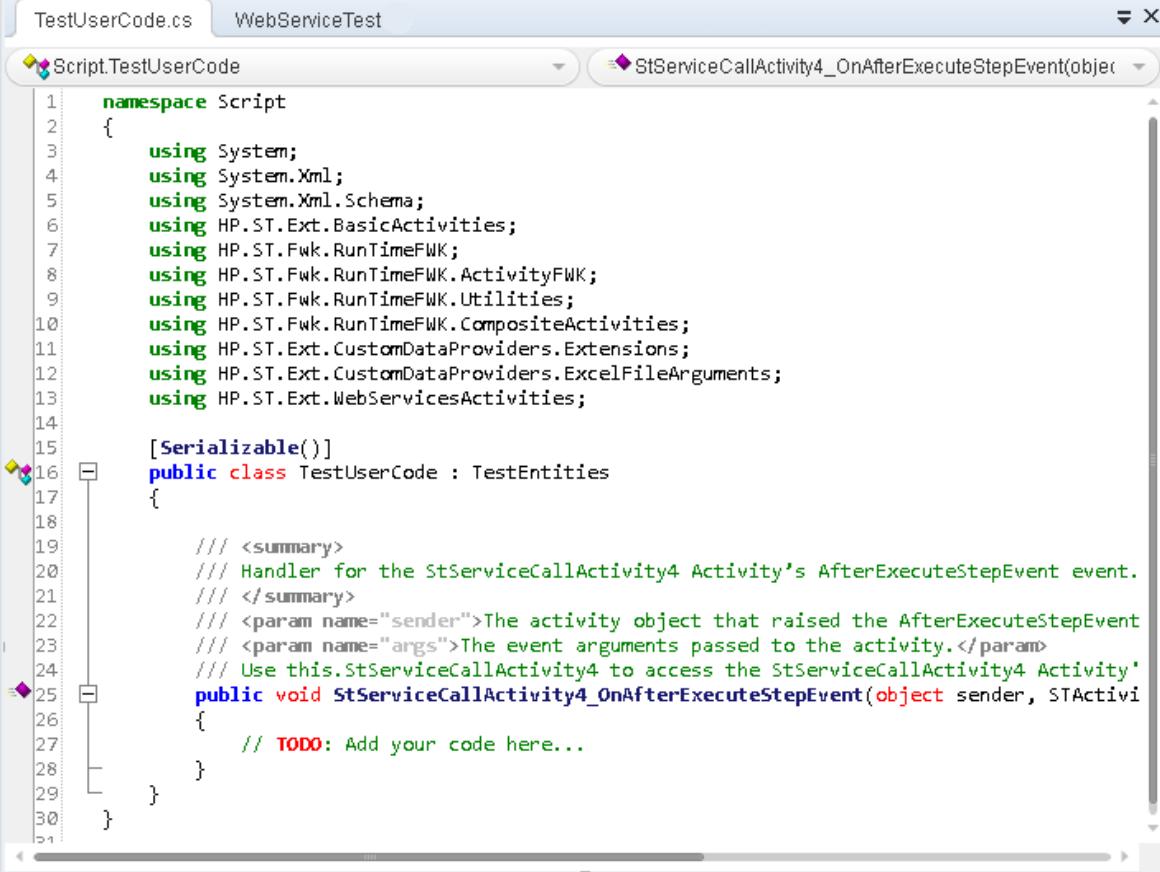
In the Handler column, double-click the row for the event to which you want to provide code.

The **TestUserCode.cs** file opens as a separate tab.

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.



```
TestUserCode.cs  WebServiceTest
Script.TestUserCode
StServiceCallActivity4_OnAfterExecuteStepEvent(object sender, STActivityEventArgs args)
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 HP.ST.Ext.CustomDataProviders.Extensions;
    using HP.ST.Ext.CustomDataProviders.ExcelFileArguments;
    using HP.ST.Ext.WebServicesActivities;

    [Serializable()]
    public class TestUserCode : TestEntities
    {
        /// <summary>
        /// Handler for the StServiceCallActivity4 Activity's AfterExecuteStepEvent event.
        /// </summary>
        /// <param name="sender">The activity object that raised the AfterExecuteStepEvent
        /// <param name="args">The event arguments passed to the activity.</param>
        /// Use this.StServiceCallActivity4 to access the StServiceCallActivity4 Activity.
        public void StServiceCallActivity4_OnAfterExecuteStepEvent(object sender, STActivityEventArgs args)
        {
            // TODO: Add your code here...
        }
    }
}
```

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 "Access and set the value of step input, output, or checkpoint properties" on page 650.

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 675. For details on the SelectSingleNode method, see "SelectSingleNode Method" on page 690.



Example: 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 655.
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 655.

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 "[Assign data to API test/component steps](#)" on page 454.
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.

Example:

- 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:

- In the canvas, select the Web service or SOAP Request step for which you want to save the Web service call.
- In the Properties pane, open the **Events** tab .
- 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;
```

Stop an API test run

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();
```

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.

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.

Example:

- 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: 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



Example:

```
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 470](#).
2. Set the Data Navigation policy for the data source. For details, see ["Set the data source navigation properties" on page 470](#).
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;
```

Access and set the value of step input, output, or checkpoint properties

Relevant for: API testing only

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.

The screenshot shows the HP Unified Functional Testing IDE interface. The main window displays a C# code editor for a file named 'TestUserCode.cs'. The code defines a class 'TestUserCode' that contains a method 'CodeActivity5_OnExecuteEvent'. The cursor is positioned at the start of the method body, and a code completion dropdown menu is open, listing various properties and methods of the 'Step' class, such as 'GetRootLoop', 'GetType', 'IconPath', and 'Prefix'. The 'Prefix' item is highlighted. The status bar at the bottom shows 'Errors: 0' and 'Warnings: 0'.

```
[Serializable()]
public class TestUserCode : TestEntities
{
    /// <summary>
    /// Handler for the CodeActivity5 Activity's ExecuteEvent event.
    /// </summary>
    /// <param name="sender">The activity object that raised the Ex
    /// <param name="args">The event arguments passed to the activi
    /// Use this.CodeActivity5 to access the CodeActivity5 Activity's
    public void CodeActivity5_OnExecuteEvent(object sender, STActivit
    {
        this.ConcatenateStringsActivity4.|
```

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 the previous page](#) 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: 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:

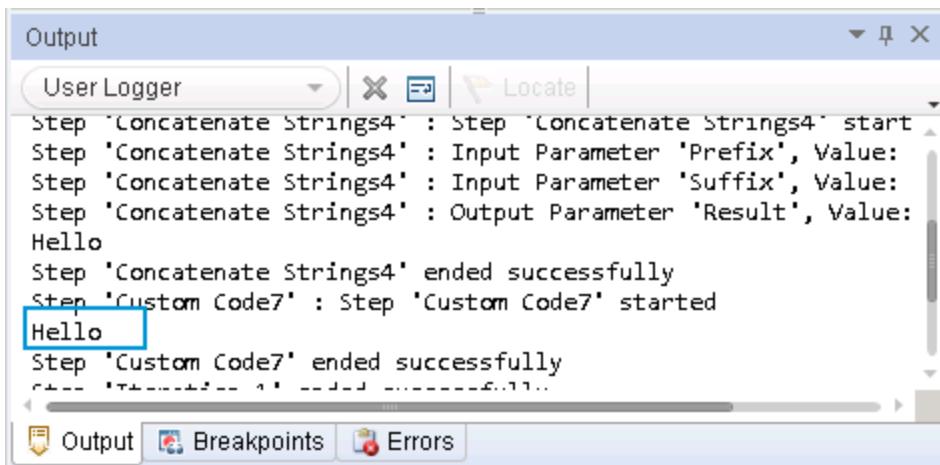
1. Add an event handler for the **OnAfterExecuteStep** event to the step for which you want to watch a property/parameter value.
2. Use the `UserLogger` object to report the value to the compilation log in the Output pane, using the following syntax:

```
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](#)" on the previous page.
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 670.



Example: 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");
```

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.

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:



Example:

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.

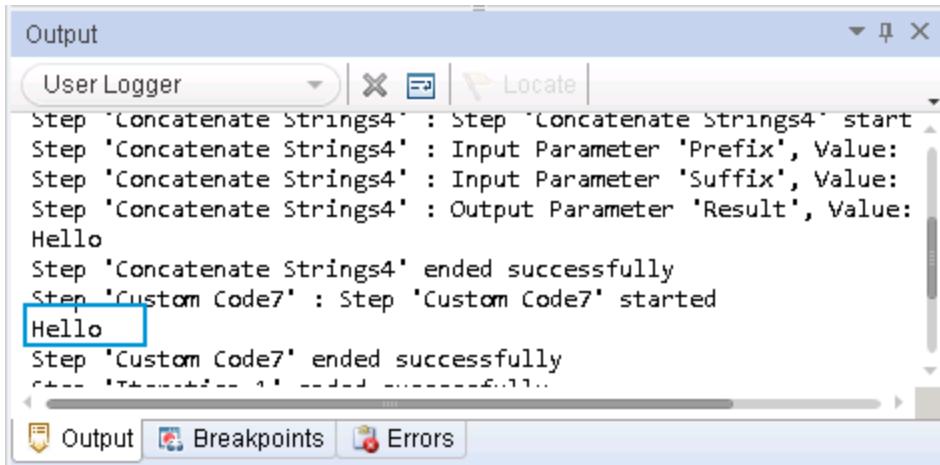
1. Select the step for which you want to report information.
2. In the Properties pane, open the **Events** tab .
3. In the Events tab, create an event handler. The **TestUserCode.cs** file opens.

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:



Example:

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);
```

Retrieve and set test or user variables

Relevant for: API testing only

Prerequisite - create user variables.

See "[Define user variables](#)" on page 462 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 462

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>");
```

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>");
```

or



Example: this.<activity name>.Context.EnvironmentProfile.SetVariableValue("<variable name>","<variable value>");



Example: 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");
```

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.

Encrypt the password

Use the activity's context's **EncryptionMngr** method, and select **Encrypt** from the autocomplete drop-down. The following example encrypts a password.

```
string plainText = "myPassword";
string encryptedText =
this.StServiceCallActivity4.Context.EncryptionMngr.Encrypt(plainText);
```

Decrypt the password - optional

Use the **Decrypt** method from the autocomplete 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 test events structure

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:

- [Standard event structure](#) 662
- [Web Service event structure](#) 665

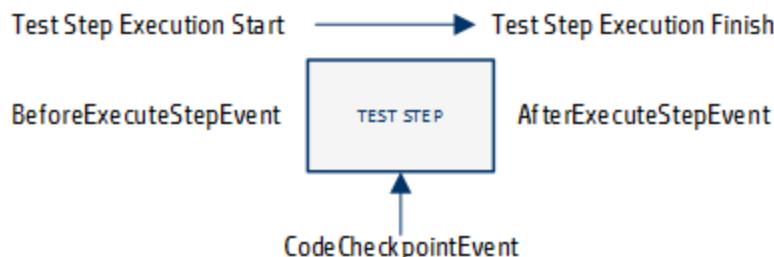
Standard event structure

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 the next page

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

```
/// <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, STActivityEventArgs 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

```
/// <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

```
/// <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 );
    }
```

Web Service event structure

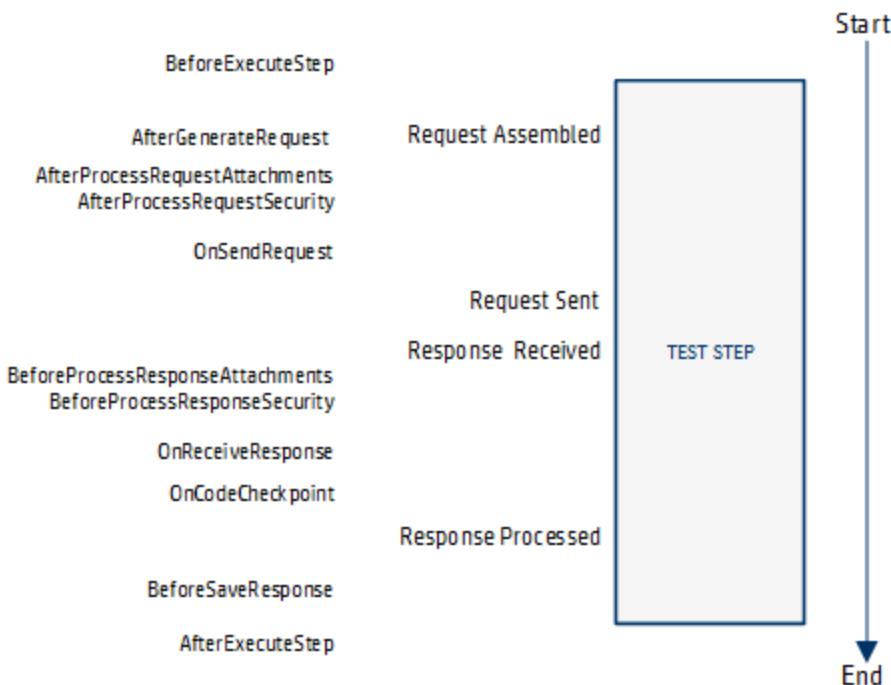
Relevant for: API testing only

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](#)" on the next page
- "[AfterExecuteStepEvent](#)" on the next page
- "[CodeCheckpointEvent](#)" on the next page
- "[AfterGenerateRequest](#)" on the next page
- "[AfterProcessRequestSecurity \(WCF services only\)](#)" on page 668
- "[OnReceiveResponse](#)" on page 668
- "[BeforeProcessResponseSecurity \(WCF Security Scenarios only\)](#)" on page 668
- "[BeforeSaveResponse](#)" on page 668

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" on page 481](#).

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" on page 481](#).

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" on page 481](#).

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	669
• Assert Object	670
• Checkpoint Object	671
• CurrentIterationNumber Object	671
• EncryptionMngr Object	672
• EnvironmentProfile Object	673
• InputAttachment Object	673
• InputEnvelope Object	675
• OutputAttachment Object	676
• OutputEnvelope Object	678
• Parent Object	678
• TestProfile Object	679
• UserLogger Object	680

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
- LesserEqual
- 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:

• Export Method	681
• ExportToExcelFile Method	682
• GetDataSource Method	683
• GetValue Method	684
• GetVariableNames Method	685
• GetVariableValue Method	686
• Import Method	687
• ImportFromExcelFile Method	688
• Info Method	689
• Report Method	690
• SelectSingleNode Method	690
• SetValue Method	691
• SetVariableValue Method	692

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. <p>Note: 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.</p>

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>	Required. The row from which to take the value. 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>	Required. 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 Test Variables 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);
```

ImportFromFile Method

Relevant for: API testing only

Description

Imports the selected Excel file to your test.

Class

DataSource

Syntax

```
GetDataSource("<data source name>").ImportFromFile(@"<path to Excel file>","  
"<test data source name>", header row);
```

Parameters

Parameter	Description
Name	Name assigned to the data source.
Path to data source	The Windows path to the file for the data source.
Data source name in the test	The name the test gives the data source after importing.
Header row	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").ImportFromFile  
(@"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
Message	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
Fully qualified XPath	The XPath to the property node in the array. Right-click the property/parameter name in the Input/Checkpoints tab  in the Properties pane and select Copy Fully Qualified XPath 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>	Required. The row from which to take the value. 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>	Required. 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.SetValue("<variable name>" , "<variable value>");  
this.<activity>.Context.EnvironmentProfile.SetValue("<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 Test Variables 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");
```

Part 8: Run / Debug Tests

Chapter 55: Running Tests and Components

Relevant for: GUI tests and components and API testing

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 "[Create and maintain business process tests and flows in UFT](#)" on page 839.

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
Run the entire test or component from the beginning	UFT runs each step, starting from the first step in the first action, in sequential order.
Run only a part of a test	<p>You can run sections of a test using these commands:</p> <ul style="list-style-type: none">• Run from Action: Runs the test from the specified action to the end of the test.• Run to Action: Runs the test from the beginning of the test to a specific action• Run from Step: Runs the test from a specified step to the end of the test.• Run to Step: Runs the test from the beginning of the test to the specified step

Run an iteration of a single action (tests only)	Using the Run Current Action command, you can run a single iteration of an individual action. If the action contains nested actions, UFT runs the nested actions for the defined number of iterations.
Debug a section of a test or component	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">• Debug from Step: Starts the run/debug session from a specified step.• Debug from Action: Starts the run/debug session from the first step of the selected action. Make sure the application is open to the relevant section when using these commands.
Update your test or component to change test object descriptions	You can use the Update Run 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 " "Maintaining Tests or Components" on page 147
Run a set of tests	Using the Test Batch Runner, you can create a batch of tests and then run all of these tests together in a setp.
Skip certain steps if necessary	You can designate certain steps as optional, which enables UFT to skip them if they fail.
Specify the number of iterations for a test or action (tests only)	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" on page 702](#)

API tests and components

UFT runs the test steps contained in the test flow in sequence, unless specified otherwise. You can:

Debug the test	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 General pane of the Options Dialog Box.
Stop the test run if a checkpoint fails	In the Test Settings tab for the test, you can instruct UFT to stop a test run if a test fails.
Run a set of tests	Using the Test Batch Runner, you can create a batch of tests and then run all of these tests together in a step.

Run a test or component

Relevant for: **GUI tests and scripted GUI components**

Prerequisites

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.

Open the test or component you want to run.

Set the number of iterations for the test

Do one of the following:

For GUI tests	In the Run tab of the Test Settings dialog box, specify the number of iterations: <ul style="list-style-type: none">• Run one iteration only. Runs the test only once, using only the first row in the global data table.• Run on all rows. Runs the test with iterations using all rows in the global data table.• Run from row __ to row __. Runs the test with iterations using the values in the global data table for the specified row range.
For API tests	<ol style="list-style-type: none">1. In the canvas, select the Test Flow or Loop box2. In the Properties Pane, open the Input/Checkpoints tab .3. Set the number of iterations.

Run an entire test or component

1. In the toolbar, click the **Run** button .
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:

For tests	<ul style="list-style-type: none">• Select Run > Run to Step.• Right-click a step and select Run to Step.• Right-click an action in the canvas and select Run to Action.
For components	Select Run > Run to Step .

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.

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)

1. Make sure your application is in a state matching the step or action you want to run.
2. 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.

3. Do one of the following:

For tests	<ul style="list-style-type: none">• Select Run > Run from Step.• Select Run > Run Current Action.• Right-click the step and select Run from Step.• Right-click the action in the canvas and select Run from Action.
For components	Select Run > Run from Step

4. 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).

Run an API test from the command line

You can also run API tests using the **ServiceTestExecuter.exe** application, located in the product's **bin** directory.

Note: To run a test from the command line, you must save and run the test at least once.

Use the following syntax to call this utility:

```
%ProgramFiles%\HP\Unified Functional Testing\bin> ServiceTestExecuter.exe -test
```

You can use any of the following arguments:

Parameter name	Description
-test	The full path of the test (required). Specify the test directory—not the solution directory.
-inParams	The full path of an XML file containing the input property values (optional).
-outParams	The full path of an XML file containing the output property values (optional).
-profile	The name of an Test profile (optional). For details, see " Define API test properties or user/system variables " on page 461.
-report	The directory in which to store the report.

Note: If you use the -inParams or -outParams arguments, the XML file must have this structure:

```
<TestParameters>
  <Values>
    <Arguments>
      <a>1</a>
      <b>2</b>
    </Arguments>
  </Values>
</TestParameters>
```

View the run results

By default, when the run session ends, the run results open.

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*.

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 "[Run a GUI test with a disconnected Remote Desktop Connection](#)" below

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.

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

1. 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
2. Set the data for this value to **2**.
3. Restart your remote session in order for this setting to take effect.

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.

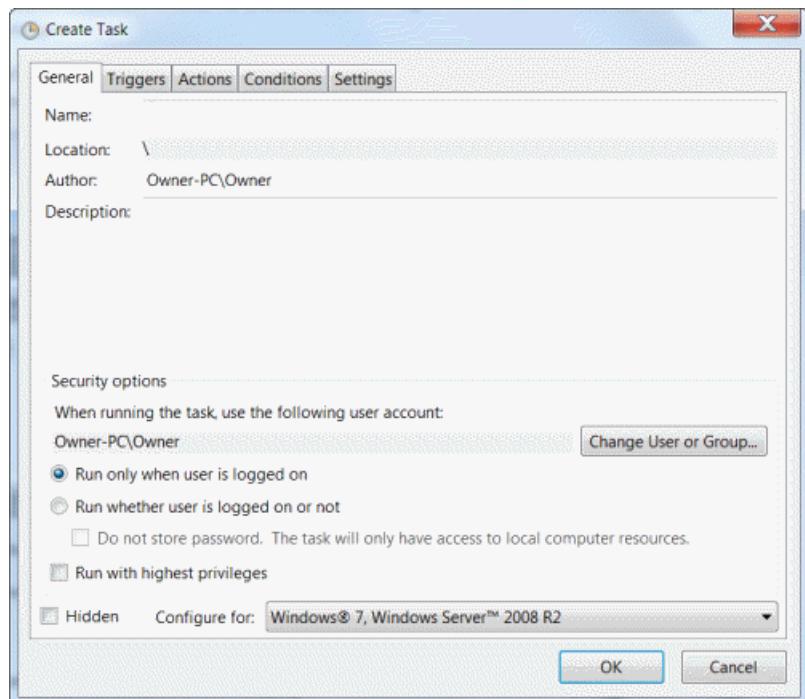
Configure the Remote Desktop Connection options in the remote computer

You must configure the Remote Connection options before beginning to run the test:

1. In the Options dialog box, open the **Run Sessions** pane (**Tools > Options > General** tab > **Run Sessions** node).
2. 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.
3. Enter the user name and password of the user connecting to the remote computer with the Remote Desktop connection.

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:



Start the test and disconnect from the Remote Desktop connection

1. On the remote computer, start the test run.
2. 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

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

The UFTRuntime 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:

Running tests and components from ALM	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).
Running tests from automation:	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
Running tests using the Jenkins plug-in:	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.

Running tests using external UFT tools	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**):

Add-ins	You can specify add-ins to be loaded.
Run options	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.
Remote connection options	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.
Run result export options	You can specify how and what the Runtime Engine should export from the run results after a run session.
Text Recognition options	You can specify how the Runtime Engine works with text in your application when running a GUI test.
Web and Windows Application options	You can specify how the Runtime Engine runs tests for specific scenarios against a Web application or Windows application.

Test Batch Runner

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.

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.

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.

Create and run a test batch

Relevant for: GUI tests and components and API testing

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.



Note: You do not need to have UFT open to use Test Batch Runner.

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.

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.

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.

Run the test batch via the command line

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"
```

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.

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:

Dialog Box / Message Box Title Bar
AutoComplete
File Download
Internet Explorer
Enter Network Password
Error
Security Alert
Security Information
Security Warning
Username and Password Required

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.

To set an optional step:

- 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
```

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).

Known Issues - Run Sessions

Relevant for: GUI and API tests

UFT run sessions	<ul style="list-style-type: none">• 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:<ul style="list-style-type: none">• 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)

Running API Tests	<ul style="list-style-type: none">• 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. <p>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.</p> <ul style="list-style-type: none">• 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.
--------------------------	--

Running steps that contain Insight test objects	<ul style="list-style-type: none">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. <p>Workaround: Use a different program (for example, Virtual Network Computing) instead of Remote Desktop Connection to run steps on a remote computer.</p> <ul style="list-style-type: none">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. <p>Workaround: Do one or both of the following:</p> <ul style="list-style-type: none">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. <ul style="list-style-type: none">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.
--	--

Chapter 56: Using Run Results

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:

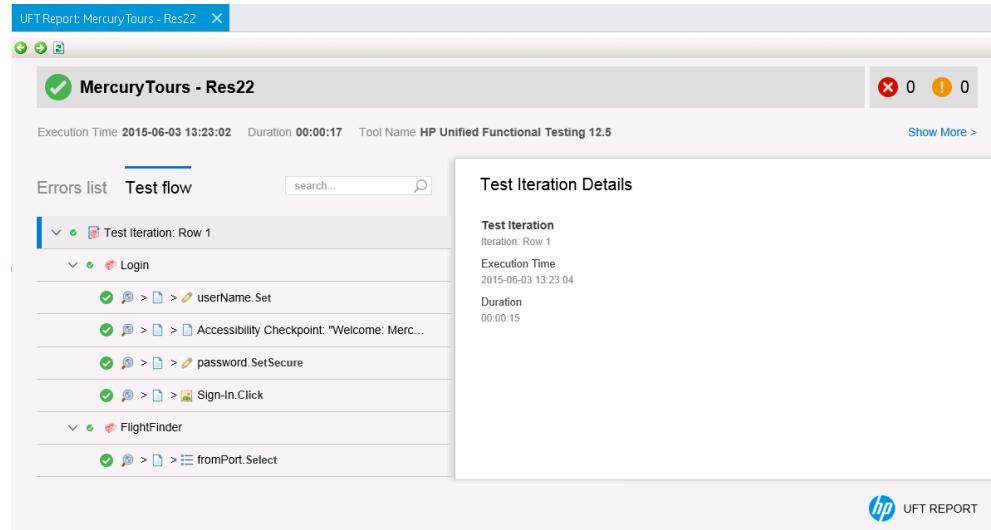
- Internet Explorer 10 or higher (with Compatibility Mode disabled)
- The latest supported version of Chrome.

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**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 area displays the "Test flow" section, which lists the steps of the test iteration. The steps shown are:

- Test Iteration: Row 1
 - Login
 - username.Set
 - Accessibility Checkpoint: "Welcome: Merc..."
 - password.SetSecure
 - Sign-In.Click
 - FlightFinder
 - fromPort.Select

On the right side, there is a "Test Iteration Details" panel showing the following information:

 - Test Iteration**: Iteration: Row 1
 - Execution Time**: 2015-06-03 13:23:04
 - Duration**: 00:00:15

At the bottom right of the report window, the HP logo and the text "UFT REPORT" are visible.

You can also choose to filter what items are shown in the test flow.

**Error
and
warning
informati
on 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\brojerem\Desktop\UFT_Tutorial\Tests\NET Tutorial\Tutorial_Object Repositories>Select Flight.tsr

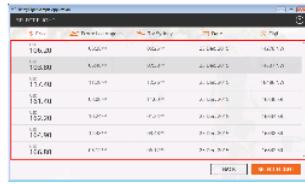
Object Path
WpfWindow("HP MyFlight Sample Application").WpfTable("flightsDataGrid")

Properties

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]

Screenshot of the captured data:



USD106.20	116.00	116.00	116.00	116.00
116.00	116.00	116.00	116.00	116.00
116.00	116.00	116.00	116.00	116.00
116.00	116.00	116.00	116.00	116.00
116.00	116.00	116.00	116.00	116.00
116.00	116.00	116.00	116.00	116.00
116.00	116.00	116.00	116.00	116.00

Run-time screen captures of your application on (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

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	<p>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:</p>
	<p>Cannot identify the object "home" (of class Image). Verify that this object's properties match an object currently displayed in your application.</p> <p>Description</p> <p>Cannot identify the object "home" (of class Image). Verify that this object's properties match an object currently displayed in your application.</p> <p>Line (12): "Browser("Flight Confirmation: Mercury").Page ("Flight Confirmation: Mercury").Image("home").Click".</p> <p>Execution Time</p> <p>2015-04-01 11:30:28</p> <hr/>
	<p>Test Object</p> <p>Cannot identify the object "home" (of class Image). Verify that this object's properties match an object currently displayed in your application.</p> <p>Repository</p> <p>C:\Users\bjerem\Desktop\UFT_Tutorial_Tests\Tutorial\Tutorial_ObjectRepositories\MercuryToursFlightCo</p> <p>Object Path</p> <p>Browser("Flight Confirmation: Mercury")</p> <hr/>
	<p>Stacktrace</p>
	<p>BookFlight Line 12: "Browser("Flight Confirmation: Mercury").Page ("Flight Confirmation: Mercury").Image("home").Click"</p>
Movies of the run session (GUI testing only)	<p>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:</p> <div style="background-color: #f0f0f0; padding: 5px; margin-top: 10px;"> <small>Execution Time: 2015-04-01 11:28:37 Duration: 00:03:10 Tool Name: HP Unified Functional Testing Tool Version: 12.5</small> Show More > Video </div> <p>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.</p>

Data resources	<p>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:</p> <div style="background-color: #f0f0f0; padding: 10px;"> <table border="1" style="width: 100%; border-collapse: collapse;"> <tbody> <tr> <td>Execution Time 2015-11-18 13:11:10</td><td>Duration 00:00:36</td><td>Tool Name HP Unified Functional Testing 12.52</td><td>Result Name Res4</td><td>< Show Less</td></tr> <tr> <td>Operation System Windows 7 Service Pack 1 (x64)</td><td>CPU Intel(R) Xeon(R) CPU E5-2670 0 @ 2.80GHz</td><td>Core Number 2</td><td>Total Memory 4096MB</td><td></td></tr> <tr> <td>Host NM13</td><td>User broje</td><td>Timezone +02:00:00</td><td>Locale English (United States)</td><td></td></tr> </tbody> </table> <p style="text-align: right;">Test Data</p> </div> <p>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.</p>	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 @ 2.80GHz	Core Number 2	Total Memory 4096MB		Host NM13	User broje	Timezone +02:00:00	Locale English (United States)																
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 @ 2.80GHz	Core Number 2	Total Memory 4096MB																												
Host NM13	User broje	Timezone +02:00:00	Locale English (United States)																												
Custom report messages (GUI testing only)	<p>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 <i>UFT Object Model Reference for GUI Testing</i>.</p>																														
Captured data for test steps (API testing only)	<p>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.</p> <p>This grid also includes any custom messages send to the run results with event handler code.</p> <p>For each API test step in the test flow, the run results provide the captured values as part of the test step summary:</p> <div style="border: 1px solid #ccc; padding: 10px;"> <p>Step Details</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tbody> <tr> <td>Step</td> <td>ReserveOrder10</td> </tr> <tr> <td>Execution Time</td> <td>2015-11-18 13:17:52</td> </tr> </tbody> </table> <hr/> <p>Captured Data</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Name</th> <th>Step ID</th> <th>Status</th> <th>Message</th> </tr> </thead> <tbody> <tr> <td>ReserveOrder10</td> <td>RESTActivityV210</td> <td>Done</td> <td>Successfully invoked an HTTP request</td> </tr> </tbody> </table> <p>More</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tbody> <tr> <td>RequestBody</td> <td><FlightOrderDetails ... /FlightOrderDetails></td> </tr> <tr> <td>HttpRawRequest</td> <td>POST http://localhost:8000/FlightOrderDetails</td> </tr> <tr> <td>RequestHeader_Content-Type</td> <td>text/xml</td> </tr> <tr> <td>RequestHeader_Host</td> <td>localhost:8000</td> </tr> <tr> <td>RequestHeader_Content-Length</td> <td>263</td> </tr> <tr> <td>RequestHeader_Expect</td> <td>100-continue</td> </tr> <tr> <td>ResponseHeader_Content-Length</td> <td>183</td> </tr> <tr> <td>ResponseHeader_Content-Type</td> <td>text/xml; charset=utf-8</td> </tr> <tr> <td>ResponseHeader_Date</td> <td>Wed, 18 Nov 2015 11:17:52 GMT</td> </tr> </tbody> </table> </div>	Step	ReserveOrder10	Execution Time	2015-11-18 13:17:52	Name	Step ID	Status	Message	ReserveOrder10	RESTActivityV210	Done	Successfully invoked an HTTP request	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
Step	ReserveOrder10																														
Execution Time	2015-11-18 13:17:52																														
Name	Step ID	Status	Message																												
ReserveOrder10	RESTActivityV210	Done	Successfully invoked an HTTP request																												
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																														

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 for 'Errors list' (selected), 'Test flow', and a search bar. The main area is a tree view of test steps:

- 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

The 'Flow Details' pane on the right provides specific information for the selected step (sc1):

- Flow: MixedFlow (#2)
- Execution Time: 2015-12-16 08:24:08
- Duration: 00:01:04

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.

You can also view run results in the Run Results Viewer. For details, see the the *HP Run Results Viewer User Guide*.

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 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:

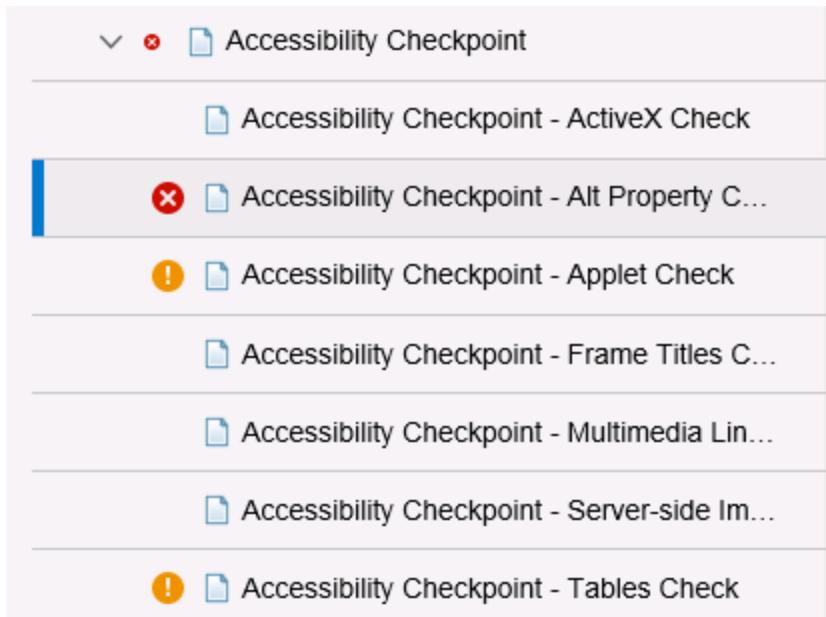
- The name of the checkpoint
- The properties being checked
- A screen capture of the object used in the checkpoint (if available depending on the selected options in the **Screen Capture** pane of the Options dialog box)

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' and 'Standard Checkpoint: "CheckName"'. The 'Standard Checkpoint: "CheckName"' entry is selected, opening a detailed view on the right. This view includes a summary section with the checkpoint name, execution time (2015-04-05 22:27:36), and a screenshot of the checked object. Below this is a 'Captured Data' table comparing expected and actual values for an 'INPUT' tag named 'value'.

Name	Expected	Actual
html tag	INPUT	INPUT
value	John	Bethami

Accessibility checkpoints	<p>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).</p> <p>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:</p>
----------------------------------	--



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	html1	[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.

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:

- **When Comparing Expected Bitmaps with Actual Bitmaps:** The step details show the expected and actual bitmaps that were compared during the run session, and a **Difference** 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.

Details

Bitmap CheckPoint

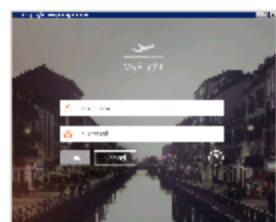
"HP MyFlight Sample Application"

Execution Time

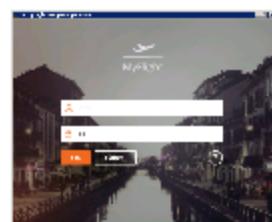
2015-04-05 22:01:49

Captured Bitmap

Expected



Actual



Difference



- **When Locating Specified Bitmaps in Actual Bitmaps:** 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.

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 **Save still image captures to results** option in the **Screen Capture** pane (**Tools > Options > GUI Testing tab > Screen Capture** node) of the Options dialog box.

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](#)

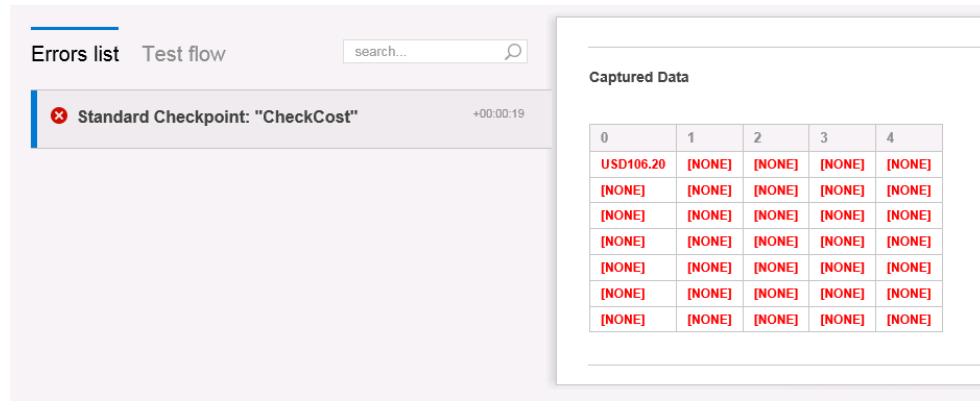
Context Information

 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: <ul style="list-style-type: none">• 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 HP Unified Functional Testing interface. On the left, there is an 'Errors list' pane titled 'Test flow' with a search bar. A red error icon is next to the text 'Standard Checkpoint: "CheckCost"' with a duration of '+00:00:19'. On the right, there is a 'Captured Data' grid. The grid has columns labeled 0, 1, 2, 3, and 4. The data rows show various values, mostly [NONE], with some red text indicating failures. The grid has a light gray background with alternating row colors.

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]
[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, there is a tree view of a test flow named "CheckCost". Under this flow, there are several checkpoint types: Database Checkpoint, Action: Flight Confirmation, Standard Checkpoint, WpfWindow.WpfButton.Click, WpfWindow.WpfProgressBar.W..., Text Checkpoint, Standard Checkpoint, and another WpfWindow.WpfButton.Click. The "Text Checkpoint" node is expanded, showing its details. The "Description" field contains the text "Text Checkpoint: captured \"completed\" Match case: OFF Exact match: OFF Ignore spaces: ON". To the right of the description is a small screenshot of a Windows application window showing a progress bar and some text. Below the description, the "Execution Time" is listed as "2015-04-05 22:18:54".

XML Checkpoints	<p>The step details display the checkpoint step results. If the checkpoint or schema validation failed, the reasons for the failure are also shown.</p> <p>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.</p> <p>The following is an example of the results for an XML checkpoint:</p> <p>Checkpoint Details</p> <p>Xml Checkpoint "file.xml"</p> <p>Description XML Checkpoint failed.</p> <p>Execution Time 2015-12-01 17:21:16</p> <hr/> <p>Test Object</p> <p> "file.xml"</p> <p>Repository Local</p> <p>Object Path XMLFile("file.xml")</p> <p>Properties</p> <table border="1"><tr><td>filename</td><td>C:\QTP.QA.Automation\TestOutput\HP. QTP.Tests.HTMLReport.Basic.HTMLReportGuiTestSanityFull\UITest_Test.html Report Pass\file.xml</td></tr></table> <hr/> <p>Captured Data</p> <p>Content Comparison</p> <hr/> <p>Context Information</p> <p> file.xml</p> <p>XMLFile</p>	filename	C:\QTP.QA.Automation\TestOutput\HP. QTP.Tests.HTMLReport.Basic.HTMLReportGuiTestSanityFull\UITest_Test.html Report Pass\file.xml
filename	C:\QTP.QA.Automation\TestOutput\HP. QTP.Tests.HTMLReport.Basic.HTMLReportGuiTestSanityFull\UITest_Test.html Report Pass\file.xml		

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.

Set run result reporting options

Before you begin a test run, set the options to change the information included with the test results in the Options dialog box (**Tools > Options**):

Option	Options Dialog pane	Description
Instruct UFT to automatically open the run results	Run Sessions pane (General tab > Run Sessions node)	Select the View results when run session ends option.
Select the format of the run results	Run sessions pane (General tab > Run Sessions node)	Select either the HTML Report or Run Results Viewer Report option.
Take screen captures of steps	Screen Capture pane (GUI Testing tab > Screen Capture pane)	Select the Save still image captures to results option. You can specify to save screen captures for: <ul style="list-style-type: none"> • Always (all steps) • For errors • For errors and warnings
Capture movies of the run session	Screen Capture pane (GUI Testing tab > Screen capture pane)	Select the Save movie to results option. You can specify to capture the movie: <ul style="list-style-type: none"> • Always (all steps) • For errors • For errors and warnings

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, view information about the checkpoint:

For a checkpoint that succeeded	<ol style="list-style-type: none">1. In the lower part of the run results, display the Test Flow.2. Select the checkpoint step from the step list. A summary of the step details is displayed.3. Use the information in the summary to view the checkpoint, including:<ul style="list-style-type: none">• The properties checked in the checkpoint• The test object used in the checkpoint• The identification properties of the test object used in the step
--	--

For a checkpoint that failed	<ol style="list-style-type: none">1. In the lower part of the run results, display the Errors.2. Select the failed checkpoint from the list. A summary of the error is displayed.3. Use the information in the summary to find the source of the error, including:<ul style="list-style-type: none">• The properties checked in the checkpoint• The expected and actual values of the checkpoint• The test object used in the checkpoint• The identification properties of the test object used in the step
-------------------------------------	---

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	A link named Data displayed in the See More section above the Test Flow and Details area of the run results. The data table is opened as an Excel file. You can also find the Excel file in the run results folder: <run results folder>\Report\Default.xls
API test	<ul style="list-style-type: none">• Excel• Local table• XML• Database	A link named Data displayed in the See More section above the Test Flow and Details 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. Note: The actual run-time values used in each step (taken from the data source) are displayed in the Details section for each test step.

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

For GUI tests	Use the Reporter object to send custom messages to the run results. These messages appear in the Test Flow in the step in which you inserted the <code>statement.</code>
----------------------	---

For API tests	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">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. The details of the test step are displayed in the right pane.3. In the details, view the Captured Data. In the captured data, you can see the custom field added in the data grid.
----------------------	--

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

When running tests in UFT, you can use a virtualized service with your test in place of your application's real service, or deploy an emulated network, to see how the network performs under certain conditions when an application is running on the network. Both these virtual/emulated services help you gain a fuller and deeper level of testing in your application.

Virtualized services

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 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 *Service Virtualization User Guide*.

For task details, see "[Use a virtualized service for a UFT test](#)" on page 741.

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	<p>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.</p> <p>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.</p> <p>There are a number of different types of performance models for a virtualization project:</p> <ul style="list-style-type: none">• 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. <p>For details on setting and defining performance models for your service, see the <i>Service Virtualization User Guide</i></p>
---------------------------	--

Data Models	<p>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.</p> <p>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.</p> <p>All data models are user-defined.</p> <p>For details on setting and defining data models for your service, see the <i>Service Virtualization User Guide</i>.</p>
--------------------	---

Network emulation

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.



Example: 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 "[Run a test using an emulated network](#)" on page 745.

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.

Use a virtualized service for a UFT test

Relevant for: GUI tests and API tests

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 *Service Virtualization User Guide*.

Add services from a virtualization project to your test in UFT

1. In UFT, click the **Virtualized Services Settings** button  in the toolbar.
2. In the Virtualized Services Settings Dialog Box, click the **Add Services** button.
3. 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.

4. Click **Next**.
5. 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.
6. 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.

7. 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.

Add virtualized services from a server to the test in UFT

1. In UFT, click the **Virtualized Services Settings** button  in the toolbar.
2. In the Virtualized Services Settings Dialog Box, click the **Add Services** button.
3. In the Add New Services dialog box, select the **Running Server** radio button and enter the name of your Service Virtualization server.
4. In the next window, provide the user name and password for the server.
5. Click **Next**.
6. 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.
7. 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.

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:

1. In the Virtualized Services Settings Dialog Box, click the **Show Runtime** button. The Service Virtualization Runtime dialog box opens
2. In the Runtime dialog box, select the project you want to deploy.
3. 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.

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

1. 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.
2. In the Service Settings dialog box, update any of the following:

Service Detail	Where?	How to update
Server address and credentials	SV Server tab	In the SV Server tab, enter: <ul style="list-style-type: none">• The revised Service Virtualization address• The revised user name or password for the Service Virtualization server
Virtualization project details	SV Project Location tab	Enter the path to the project (on the file system or ALM) and the password (if necessary) for the project.

3. Click **Save Configuration** to save the details. These revised details are used when UFT runs the test again.

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:

1. In the Virtualized Services Settings Dialog Box, select the virtualization project to use for the current test run.
2. 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.

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:

1. In the main Service Virtualization Setup dialog, click the **Show Runtime** button. The Service Virtualization Runtime dialog box opens.
2. In the Service Virtualization Runtime dialog, select the services you want to pause.
3. 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**.

4. 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.

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:

1. In the Service Virtualization setup window, select the service you want to put on standby.
2. 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.

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 *Service Virtualization User Guide*.

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

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.

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.

To see a blog post about this, see the [UFT All About the Apps blog!](#)

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*.

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:

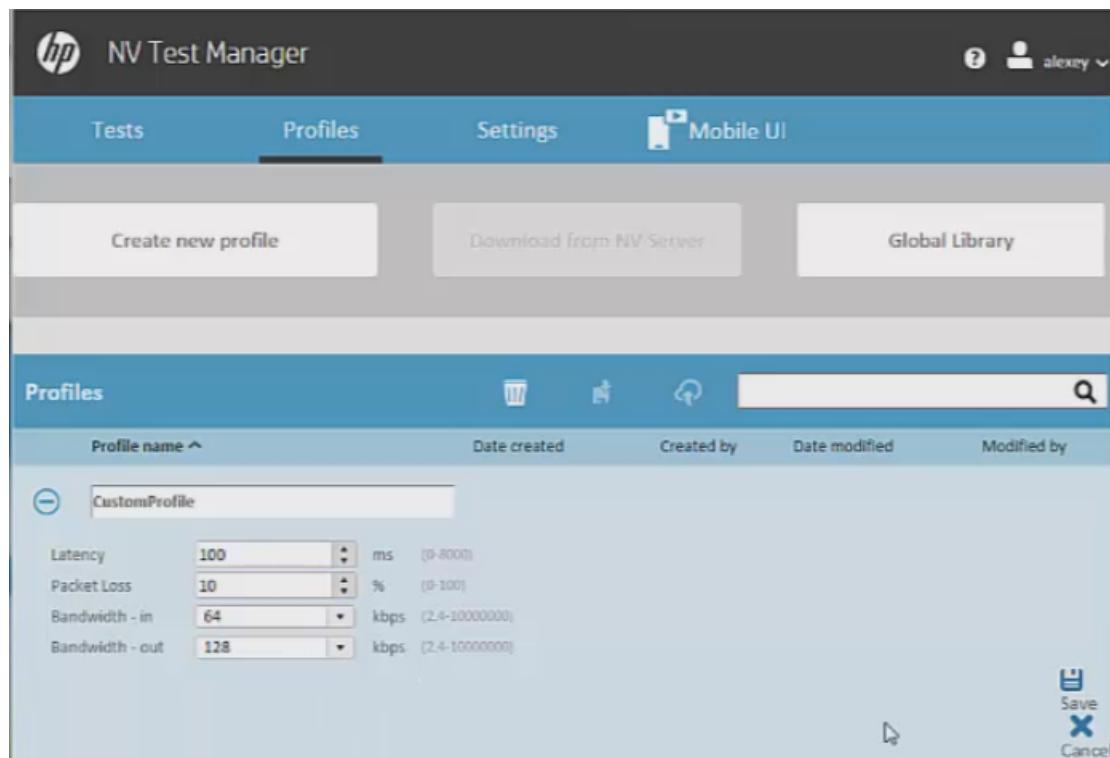
- The URL of the NV Test Manager, in the format **http://<NV Test Manager Address>:<NV Test Manager Port>**
- User name and password

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.



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.

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:

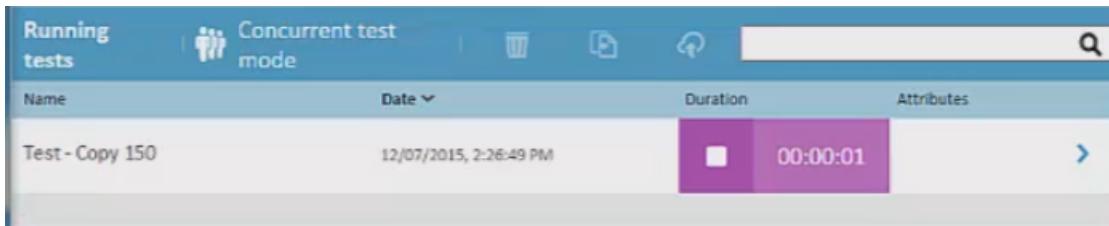
In the Network Virtualization pane of the Options dialog box	In the Excluded IPs section add the IPs to exclude. These IP addresses are excluded from all network emulation sessions launched from UFT.
Using the NV.StartEmulationExcludeIPs method	Enter a step using the .Start EmulationExcludeIPs method: <code>NV.StartEmulationExcludeIPs("profile name", array of excluded IPs)</code> These IP address are excluded only from the current emulation session.

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

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.

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:

Step Into	Runs only the current step in the active document. 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.
Step Out	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).
Step Over	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). 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 Step Into).

- Run to or from a specific step or action.

Modifying and watching the values of variables and properties of objects

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 run-time 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 753](#).

Debug a 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 ["Debug a function - Exercise" on page 755](#) (for GUI testing) or ["Debug an API user code file - Exercise" on page 759](#) (for API testing).

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**.)
- 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. To slow the test run speed to enable more effective debugging, 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

Step Into

In the toolbar, press the **Step Into** button .

Step Out	In the toolbar, press the Step Out button  .
Step Over	In the toolbar, press the Step Over button  .

Start or pause your debugging session at a specific step or action

- Select the step in your document at which you want UFT to stop and select **Run > Run to Step**.
- 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

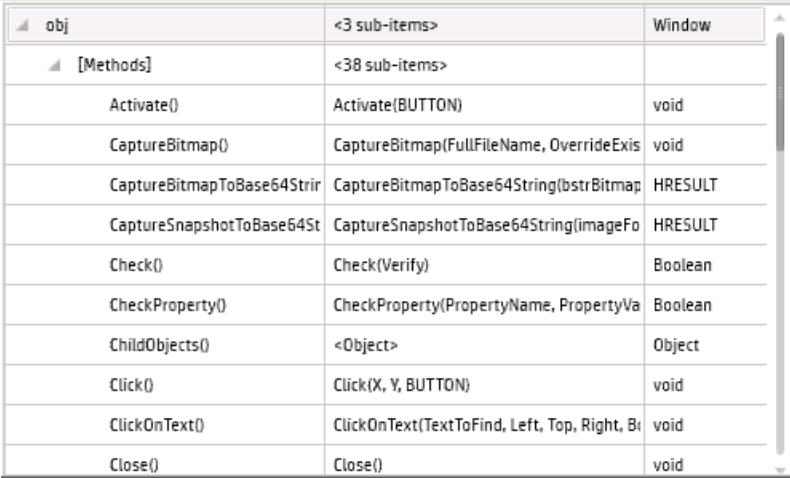
For details, see "[Use breakpoints](#) " on page 763.

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

In the Watch Pane	<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"> Click the Add New Watch Expression button  and enter the name of the expression in the Add New Watch dialog box. For GUI actions, scripted GUI components, and function libraries only: Highlight the selected expression and select Run > Add to Watch right-click the expression and select Add to Watch from the context menu. <p>To add an identification property to the Watch pane, you must use an expression that calls GetROPProperty. 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:</p> <pre>Window("Calculator").WinEdit("Edit").GetROPProperty("text")</pre> <p>You can add an expression to the Watch pane from the Editor or from a function library, but not from the Keyword View.</p>
In the Quick Watch	<ol style="list-style-type: none"> 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"> Right-click and select Quick Watch Select Run > Quick Watch In the Quick Watch, do one of the following: <ul style="list-style-type: none"> In the Expression field, enter the name of the object, variable, or expression and click Evaluate. UFT displays the value in the current context of the test or component. Enter the name of the object, variable, or expression and click Add to Watch. UFT adds it to the list in the Watch Pane
In the Local Variables pane	<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>

In the Editor	Hover over an object, variable, or expression when the test run is paused. UFT displays a floating tooltip with the value:
 <p>The screenshot shows a floating tooltip window titled 'obj'. It contains a table with two columns: 'Method' and 'Value'. The methods listed are: [Methods], Activated(), CaptureBitmap(), CaptureBitmapToBase64String(), CaptureSnapshotToBase64String(), Check(), CheckProperty(), ChildObjects(), Click(), ClickOnText(), and Close(). The values for these methods are: <3 sub-items>, Activate(BUTTON), CaptureBitmap(FullFileName, OverrideExis, void), CaptureBitmapToBase64String(bstrBitmap), CaptureSnapshotToBase64String(imageFo, HRESULT), Check(Verify), CheckProperty(PropertyName, PropertyVa, Boolean), Object, Click(X, Y, BUTTON), ClickOnText(TextToFind, Left, Top, Right, Bi, void), and Close().</p>	

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.

View currently running threads

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.

View the loaded modules associated with the run session

Select **View > Debug > Loaded Modules**. UFT displays the list of currently loaded modules (depending where in the test you have paused).

Debug 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 "Debug a test, component, function library, or user code file" on page 751.

Create a new action or function

1. Create or open a function library.
2. Create a new function called **SetVariables**.

Enter the VBScript code in the Editor of your action or the function library, as follows:

```
Function SetVariables  
  
Dim a  
a="hello"
```

```
b="me"  
MsgBox a  
  
EndFunction
```

Associate the function library with a test)

1. Bring the function library into focus.
2. Right-click the function library document tab and select **Associate Library <Function Library Name> with <Test Name>**. UFT associates the function library with your test or application area.

Add a call to the function in your test

Add a call to the function by typing the following in the Editor:

```
SetVariables
```

Add breakpoints

Click in the left margin of the lines containing the text **b="me"** and **MsgBox a**.

Begin running the test or component

Run the test. The test stops at the first breakpoint, before executing that step (line of script).

Check the value of the variables in the debug panes

1. Select **View > Debug > Watch** to open the Watch pane.
2. 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.
3. 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.

4. Select **View > Debug > Local Variables** to open the Local Variables pane.

Only variable **a** is displayed (with the value "**hello**"), because **a** is the only variable that was initiated up to this point.

Variable **b** is not displayed because the test or component stopped before variable **b** was declared.

Check the value of the variables at the next breakpoint

Click the **Run** button to continue running the test.

The test stops at the next breakpoint. Note that the values of variables **a** and **b** were both updated in the Watch and Local Variables panes.

Modify the value of a variable using the Console pane

1. Select **View > Debug > Console**.
2. 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.
3. 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**".
4. Click the **Run** button to continue running the test.

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.

5. Click **OK** to close the message box.

Repeat a command from the command history

1. Remove the first breakpoint and run the test or component again.

When the test 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"**.

2. 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**.
3. In the command prompt, right-click and select **Paste**.
4. 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.

Step Into, Out of, or Over a specific step - Exercise

Relevant for: GUI actions, scripted GUI components, and function libraries

In this exercise, you create a sample function library and run it using the **Step Into**, **Step Out**, and **Step Over** commands.

Note: For a task related to this exercise, see "Debug a test, component, function library, or user code file" on page 751.

Create the sample function library and test

1. Select **File > New > Function Library** to open a new function library.
2. 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.

3. Save the function library to your ALM project or to the file system, with the name **SampleFL.qfl**.
4. Select **File > New > Test** and select **GUI Test** to open a new test.
5. Click the document tab for the **SampleFL.qfl** function library to bring it into focus.
6. Right-click the function library document tab and select **Associate Library 'SampleFL.qfl' with 'Test'** to associate the function library with your test.
7. 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"
```

Run the function library and use the commands

1. Select the first step of the action (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 .

2. Run the test. The test pauses at the breakpoint.
3. Press **F11 (Step Into)**. The execution arrow points to the first line (`msgbox "one"`) of the function in the function library.
4. Press **F11 (Step Into)** again. A message box displays the text `one`.
5. Click **OK** to close the message box. The execution arrow moves to the next line in the function.
6. 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 (the second call to the `myfunc` function).
7. Press **F11 (Step Into)** to enter the function again. The execution arrow points to the first `msgbox` line within the function.
8. 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 (the third call to the `myfunc` function).
9. 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 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.

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 "["Debug a test, component, function library, or user code file" on page 751](#)".

Create test steps

1. Create an API test.
2. From the Toolbox Pane, in the Math section, drag the **Add** activity, **Multiply** activity, and the **Custom Code** activity to the canvas.

Set properties for the math steps

1. In the **Input/Checkpoints** tab , in the Input pane, enter the values for the **Add** operation.

- In the **Value** column for **A**, enter 10.
 - In the **Value** column for **B**, enter 6.
2. 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.

Create parameters for the Custom Code activity

1. 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.
2. 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.
3. Click the **Add** button again and select **Add Output Property**.
4. Enter the details for the output parameter.
 - In the **Name** field, enter **Result**.
 - In the **Type** field select **Decimal** from the drop-down list.
5. In the **Checkpoints** pane, enter the value of **128**.

Link the Custom Code activity to existing steps

1. 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.
2. 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**.

Create events for the Custom Code activity

1. In the Properties pane, select the **CustomCode** activity from the drop-down list or by clicking the **CustomCode** activity in the canvas.
2. In the Properties pane, select the **Events** tab.
3. 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.
4. 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;
```

5. Enter a breakpoint on the last line of this method.
6. In the **TestUserCode.cs** file, enter the following text in the **CodeActivity<number>_** **OnAfterExecuteStepEvent**:

```
decimal result = CodeActivity<number>.Output.Result;
```

7. Enter a breakpoint on the last line of this method.
8. Save the test.

Run the test

Select **Run > Run** or press F5.

Check the value of the variables at the first breakpoint

1. When the test run stops at the first breakpoint, select **View > Debug > Local Variables** to open the Local Variables pane.
2. 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.

Add a variable to the Watch Pane

1. In the **TestUserCode.cs** tab, highlight the text **AddResult**.
2. Open the Watch pane by selecting **View > Debug > Watch**.

3. 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.
4. 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**.

Check the value of the variables at the next breakpoint

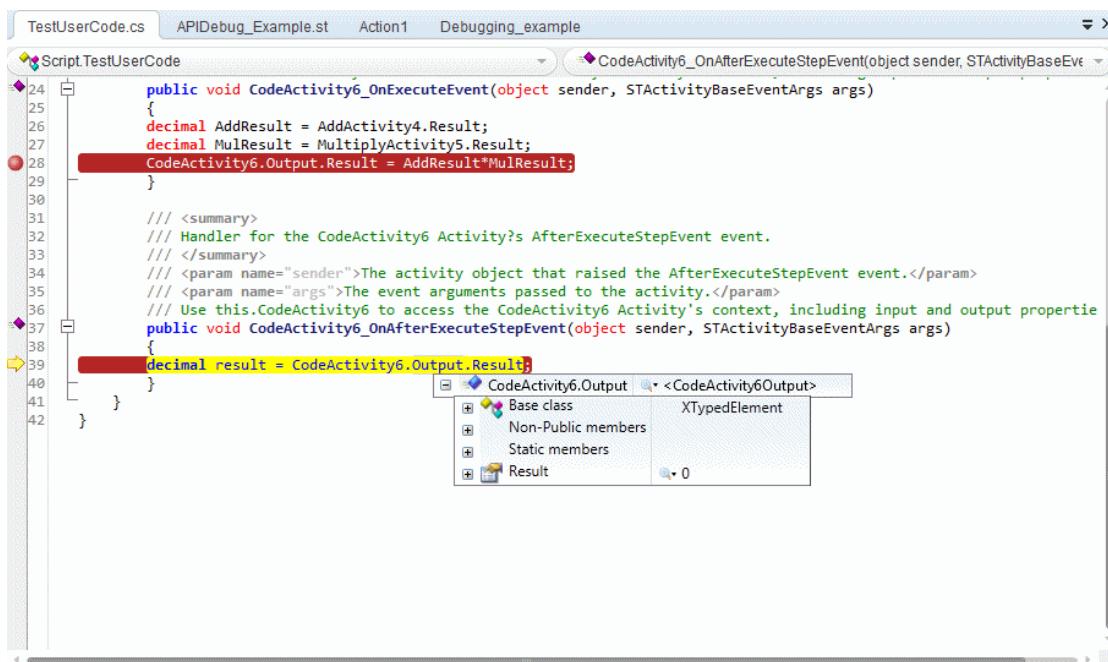
1. Continue the run session by selecting **Run > Continue** or pressing F5.
2. When the run session pauses at the next breakpoint, highlight the text **CodeActivity<number>.Output.Result** and add it to the Watch pane.

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.

3. 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.



4. Select **Run > Continue** to complete the run session and view the run results.

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.

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 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.

Known Issues - Debugging

Relevant for: GUI actions and scripted GUI components, and function libraries

"Out of memory" exception when debugging	<p>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.</p> <p>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\CLSID\{78A51822-51F4-11D0-8F20-00805F2CD064}\InprocServer32.</p> <p>To register a newer version of the pdm.dll, do the following:</p> <ol style="list-style-type: none">1. 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.2. 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.)3. Run the following commands:<ul style="list-style-type: none">• regsvr32 <full path to pdm.dll>\pdm.dll• regsvr32 <full path to msdbg2.dll>\msdbg2.dll
Adding action automation objects to the Watch pane	<p>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.</p> <p>Workaround: Restart UFT and open your test.</p>
Breakpoints on a line after a run-time error	<p>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.</p> <p>Workaround: Set the breakpoint at least two lines after the line that causes the run-time error.</p>

Opening a function library after pausing to debug	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.
Debugging from ALM	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.
Debugging Windows-based applications	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 . <p>Workaround: Do one of the following:</p> <ul style="list-style-type: none"> • In the MicIPC section of the <code>mercury.ini</code> file (located in <code>%SYSTEMROOT%</code>) add these entries: <pre>devenv.exe=0 msdev.exe=0 msscrdbg.exe=0</pre> <ul style="list-style-type: none"> • Open the debugging program from a user account other than the one running UFT.
Using the Quick Watch debugger	<ul style="list-style-type: none"> • 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 <code>arr1(2)</code>, UFT will display the value of the third element in the tooltip unless you highlight the <code>arr1</code> expression without the <code>2</code>.

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.

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:

Create tests, components, and resources and save them in your ALM project.	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.
View the contents of your tests.	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.

Run your tests and components and view the results in ALM.	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.
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.	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.
Associate external files for all tests or for a single test.	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.
Take advantage of all the features provided with the Resources and Dependencies model.	For details, see "Resources and Dependencies for ALM assets" on page 774 .
Use the QCUtil object to access and use the full functionality of the ALM OTA (Open Test Architecture) (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>UFT Object Model Reference for GUI Testing</i> and the ALM Open Test Architecture documentation.
Use the TDOTA object in your automation scripts to access the ALM OTA (GUI testing only)	For details, see the <i>UFT Object Model Reference for GUI Testing</i> or the HP UFT GUI Testing Automation and Schema References Help > Automation Object Model Reference).

For a list of required access permissions for working with ALM, see ["UFT program use" on page 39](#).

Work with tests and components in ALM

Relevant for: GUI tests and components and API testing

Prerequisites

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 "[UFT program tools](#)" on page 45.

For details on setting up secure connections to your ALM server, see your ALM documentation.

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 server name and login credentials for your users.

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.

3. Click **Connect** to connect to the ALM server. UFT pauses for a few moments to connect with the ALM server.
4. In the lower part of the ALM Connection dialog box, select the domain and project to access.
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, in the **Test Runs** pane of the Options dialog box (**Tools > Options > GUI Testing tab > Test Runs** node), select the **Allow other HP products to run tests and components** option.

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 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>.

Enable the Remote Agent

If the **Windows Firewall** is **turned on** on your computer, and you want to enable tests to be run on your computer from a remote ALM client, then you must manually 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.

Install an external certificate for your ALM server

This is necessary when your ALM server is 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.

Note: 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**.

Create a template test

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 "Create a template GUI test" on page 778.

Set UFT Remote Agent Preferences

1. Select **Start > All Programs > HP Software > HP Unified Functional Testing > Tools > Remote Agent** or **<UFT installation folder>\bin\UFTRemoteAgent.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.

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.

Resources and Dependencies for ALM assets

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:

Replaces the use of attachments with linked assets.	<p>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.</p> <p>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.</p>
Supports versioning	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 811.
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.
Improved 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).
Supports baselines	You can view baseline history in UFT, and you can view and manage baselines in ALM.
View/compare assets	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.

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.
Easy deletion of assets	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.

Relative paths for tests/resources saved in ALM

When you save a test and its resources in ALM, ALM creates a path to 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.

UFT deals with the absolute vs. relative path mappings differently depending on where in ALM the resource is saved:

- If you associate resources from 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.

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).

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 and comments or steps to include in all tests. For example, a template test might specify the UFT add-ins, associated function libraries, and recovery scenarios that are associated with a 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. 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).

For details on how to create and work with template tests, see "[Create a template GUI test](#)" below.

Create a template GUI test

Relevant for: **GUI tests only**

In UFT	<ol style="list-style-type: none">1. Open an existing test with the required add-ins loaded. 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).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.
In ALM	<ol style="list-style-type: none">1. 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.

Data drive a test 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.

Prerequisites

1. Connect to ALM.
2. Make sure that your test is saved in your ALM project.
3. **For GUI testing:** Make sure you have a test that uses data table parameters from the **Global** sheet.

Import data into a test (API testing only)

1. In the Data pane, click the **New Data Source** button  and select **Excel**.
2. 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.
3. Click **OK** to import the data source into your test.

Data drive the test steps (API testing only)

For details, see "[Assign data to API test/component steps](#)" on page 454.

Create a data resource file in your ALM project

1. In ALM, in the Test Resources module, expand the Resources tree and select the required node.
2. Select **Resources > New Resource** to add a resource under that node.
3. 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.
4. In the Resource Viewer tab, click **Upload File**. Then browse to and upload the relevant .xls or .xlsx file.



Tip: 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.

Specify a default data table resource for all new test configurations

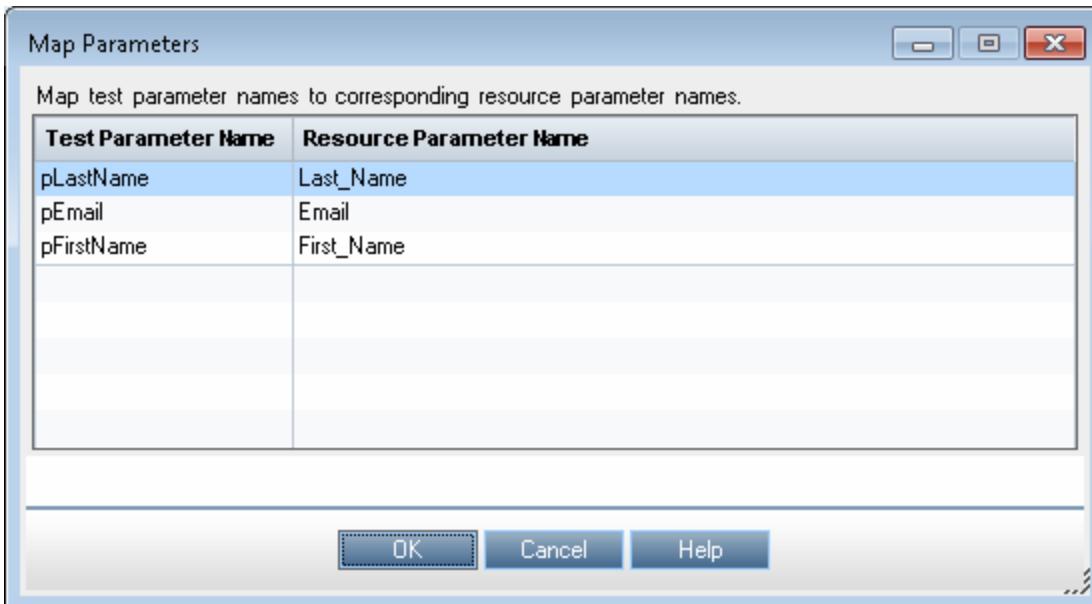
1. In the Parameters tab of the Test Plan module, select the data table resource you want to use as the default for all test configurations.
For a GUI test, if you do not specify a data table resource, the data specified in the Resources pane of the Test Settings dialog box (**File > Settings**) is used instead.



Note: In this tab, only the Parameter Name column is relevant for GUI

tests.

2. 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.

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.

1. In the ALM Test Plan module, browse to and select the test to associate with your data table resource.
2. With the test selected, 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.
3. In the bottom pane of the Configurations tab, click the **Data** tab.
4. 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.

5. 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)
6. 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.

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.

1. In the Test Plan module, click the **Req Coverage** tab.
2. Click the **Select Req** button. The Requirement Tree tab is displayed in the right pane.
3. 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.
4. Select the test configurations that cover this requirement.

Run your test configuration

1. 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.
2. In the ALM Test Lab module, select or create a test set.
3. In the right pane, select the **Execution Grid** tab.
4. Click the **Select Tests** button to display the **Test Plan Tree** and **Requirements Tree** tabs in the right pane.

5. 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.)
6. Click the **Run** button to run the selected configurations.
7. After the run session, click the **Launch Report** button in the Last Run Report tab to view the results.

Known Issues- Resources and Dependencies

Relevant for: GUI tests and components and API testing

- 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.

Known Issues- General ALM integration

Relevant for: GUI tests and components and API testing

ALM and Windows security issues	<ul style="list-style-type: none">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 "Known Issues - UFT Tools and Programs" on page 45. 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.
--	---

Connecting to ALM on Windows 8.x	<ul style="list-style-type: none">In Windows 8.x, when trying connecting to ALM (for ALM versions earlier than 12.53) 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
Connecting to ALM on Windows 10	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.
First connection to ALM	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.
ALM authentication	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: <ol style="list-style-type: none">1. Go the following location on the ALM client machine: http://\<ALM Platform server name>[<port number>]/qcbin/Apps/2. Click on the Webgate Customization Tool link and select Run.3. 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.4. Click Save and then Close.
Switching ALM servers	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.

Running tests remotely from ALM	<p>Before you run tests remotely from ALM, you must perform the following prerequisites:</p> <ol style="list-style-type: none">1. You must enable COM+ access. For details, see the Installation Guide.2. Change DCOM permissions and open firewall ports. For details, see the Installation Guide.3. 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. <p>This is due to a problem with opening DCOM permissions on Windows 7 and Windows Server 2008 R2.</p> <ol style="list-style-type: none">4. Disable User Account Control (UAC) in Windows and restart your computer before you first connect to ALM. For details, see "Known Issues - UFT Tools and Programs" on page 45.
Working with documents saved on ALM	<ul style="list-style-type: none">• 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.• 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.

QuickTest tests saved in ALM	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).
Adding documents stored in ALM to a solution	Testing documents that are part of the same solution cannot be stored in different ALM projects, domains, or servers.
Working with non-Unicode ALM projects	ALM supports non-Unicode projects. If you are working with an ALM project that is not Unicode compliant: <ul style="list-style-type: none">• 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 " Known Issues- Multilingual Applications " on page 43.

Known Issues- ALM Integration with GUI Testing

Relevant for: GUI tests and components

Running a GUI test from ALM	To run a test or component from ALM, you must first invoke UFT at least once. Otherwise, ALM may be unable to open UFT.
------------------------------------	---

Parameters names in the Data table	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.
Renaming parameters	Renaming a UFT test parameter from UFT will cause any run-time parameter values already set for this parameter in ALM to be lost.

Known Issues- ALM integration with API Testing

Relevant for: API testing

Changing licenses	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.
Linking API test parameters	<ul style="list-style-type: none">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.

Chapter 61: Running Tests from ALM

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

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

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. 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

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. By 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 "[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 use in tests that run using server-side execution. When ALM runs the tests, ALM 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, link UFT 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.

For details on how to link test parameters to AUT parameters, see "[Run a test using Server-Side Execution](#)" on the next page

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 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.

Run a test using Server-Side Execution

Relevant for: GUI tests and API testing

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.

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.

Create tests and save them in ALM

Create your tests in UFT or ALM, and save them in ALM.

Create functional test sets in ALM

1. In ALM, in the **Testing > Test Lab** module, create a functional type test set and specify the required information.
2. Select the **Execution Grid** tab and click **Select Tests**.
3. In the Test Plan tree, select the tests you want to add to the test set.

Set up AUT Parameters in ALM and link your test parameters to them in UFT - optional

1. 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*.
2. 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 .
3. Click **Add > Add Input Parameter**.
4. In the Add Input Parameter dialog box, click the **Select ALM application parameters** button .
5. In the Select AUT Parameter dialog box, expand the **AUT Environment** node and select a parameter. Click **OK**.
6. 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

Set up hosts in ALM for the UFT tests

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.

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.

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.

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*.

Note:

If you do not configure the Remote Agent as an exception as described in "[Enable the Remote Agent](#)" on page 772, a Windows Security Alert message will display while running a test remotely.

Click **Unblock** to solve this problem.

Test parameterization and test configurations

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

For each test saved in ALM, a test configuration can be applied for the test to enable you to run the test with variable data. This enables you to use the same data file or data files repeatedly to provide changing data for a particular test run. Furthermore, you maintain fewer sets of data, in a central location in your ALM project, which is accessible for all tests in the project.

When you save a test, a default **configuration** with the same name as the test is created simultaneously. A **test configuration** is specific setup of test data that represents a business process use-case. 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:

Share common data sources across different tests	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.
Test various use-cases, each time with a different set of data	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.

When running test configurations, there are different ways in which to associate data with the test configuration:

Static data association	<p>Static data is created 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.</p>								
	<p>In UFT, this data is displayed as read only in the Test Configurations tab in the Properties pane:</p>  <p>Test Configurations:</p> <p>Flight_Application Delete New</p>								
	<p>Selected Iterations: All</p> <p>Static Test Data:</p> <table border="1" data-bbox="425 853 1176 958"> <thead> <tr> <th>toCity</th> <th>fromCity</th> <th>NumberOfTickets</th> <th>Class</th> </tr> </thead> <tbody> <tr> <td> </td> <td> </td> <td> </td> <td> </td> </tr> </tbody> </table> <p>Static data association is recommended when you only need a small amount of data for the test run.</p>	toCity	fromCity	NumberOfTickets	Class				
toCity	fromCity	NumberOfTickets	Class						
Dynamic data association	<p>Dynamic data association is created with 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.</p> <p>This approach is recommended if you have large amounts of data that is maintained in an external file.</p>								

To help parameterize your test configurations, add Microsoft Excel (**.xls**) files in stored the Test Resources module. Once the Excel file is in the Test Resources module, map its column names to the data table parameters (for GUI and API tests) or test parameters (for business process tests). By mapping the column names to data table parameters, you enable UFT to identify and use the correct parameter value during a run session.

Depending on the location in ALM you want to use for the test configuration's data, mapping is performed differently:

With the data resource in the test's Parameters tab in ALM	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 use a different data resources or override the data with another resource	you need to map the data table parameters to the column names in the .xls or .xlsx file.



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.

Test Combinations Generator

Relevant for: business process tests

The Test Combinations 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 Combinations Generator to prepare this data for you.

The Test Combinations Generator creates configurations based on the following algorithms:

Linear	Includes all possible combinations of parameters and values. May result in a very large amount of data.
Pairwise	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.
Triplewise:	Includes all combinations of valid values for triples of input parameters.

Furthermore, you can specify values be added to special paths:

Happy Path:	No error or exception is expected to be raised by using this particular value
Error Path:	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.

Use-Case Scenario: Use the Test Combinations 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 Combinations 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
Login	Login window
FlightFinder Page	Window to specify flight details (departure, arrival, etc.)

Select Flights Page	Window to select an available flight
Flight Confirmation Page	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.

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 Combinations Generator to generate the combinations automatically.

1. Provide the possible parameter values.

After [opening the Test Combinations Generator](#) in the Test Configurations pane, UFT displays the parameters in the Test Combinations Generator main window.

Class	fromCity	NumberofTickets	toCity
Business	Los Angeles	2	Denver

You enter the possible values for each of the parameters.



Tip: If you click Generate Parameters in the upper left corner of the Test



Combinations Generator window, UFT can automatically generate parameter values for these parameters (depending on the required format of the value).

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.

The screenshot shows the 'Test Cases Generator' interface. The configuration name is 'Flight_Application'. The 'PAIRWISE' algorithm is selected. The table below lists the generated combinations:

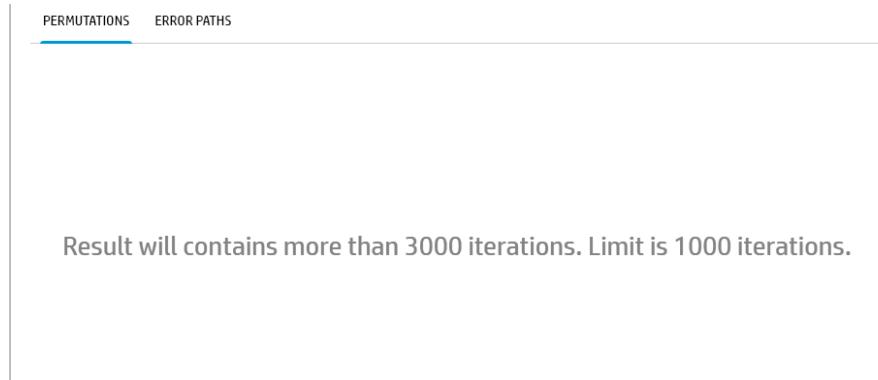
Class	fromCity	NumberofTickets	toCity
Business	Los Angeles	1	Denver
Business	Denver	2	San Francisco
Business	Seattle	3	Portland
Business	San Francisco	4	Seattle
Business	Portland	5	Los Angeles
Business	Paris	6	Zurich
Business	London	7	Frankfurt
Business	Sydney	8	Paris
Total 112			

At the bottom left is a 'BACK TO PARAMETERS' link, and at the bottom right is a 'GENERATE' button.

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).

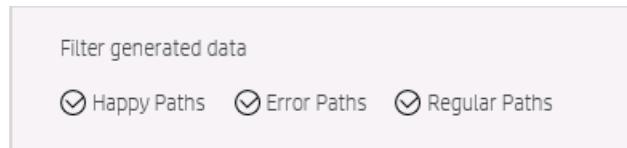


In this case, because the total number of combinations when using Triplewise or Linear algorithms is huge, the **Pairwise** algorithm is the one to select. You can select others, depending on your testing needs, and UFT will only generate 1000 combinations.

4. Generate the configurations.

Now that you have entered the parameters and selected the algorithm to use, you can generate the different combinations.

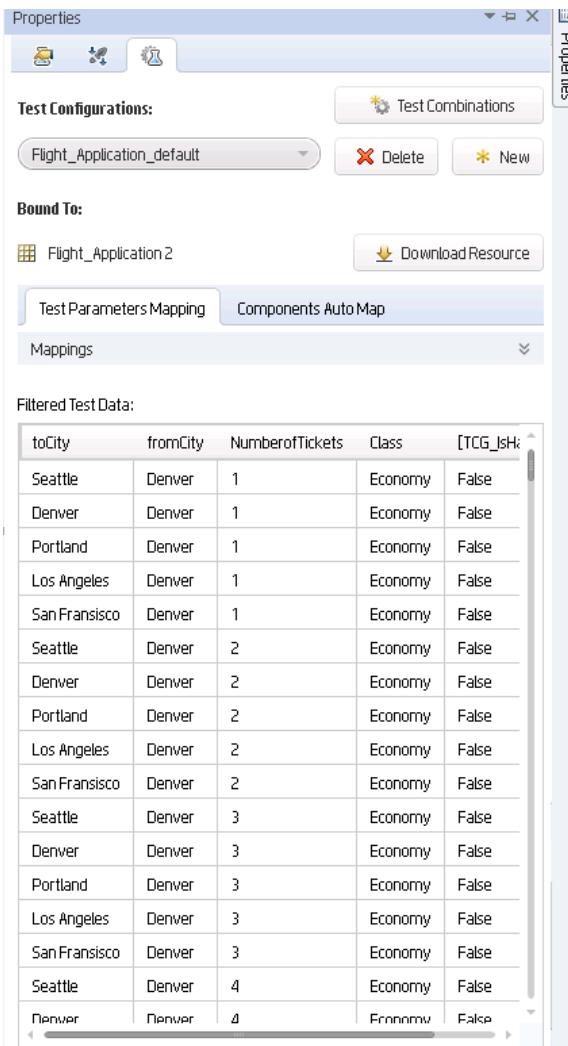
First, instruct UFT which configurations to generate. If you click **Filter**, you can choose from **Regular Path** (the standard combinations), **Happy Path**, or **Error Path** combinations:



For this scenario, you only need the **Regular Path** configurations.

Then, in the bottom part of the Test Combinations 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.



Set up and run test configurations

Relevant for: business process tests

Note: This task is part of a higher-level task. For details, see ["Create and maintain business process tests and flows in UFT" on page 839](#).

Prerequisite

Create an Excel file containing your test data, with the following:

- **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 components, the columns with the parameter names must use 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 880](#)".

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:

- In the Properties pane, open the **Test Configurations** tab
- Do one of the following:

From a data set	<ol style="list-style-type: none">In the Test Configurations tab, click the New button.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.Click OK to associate the Excel file with the test configuration. UFT displays the test configuration tabs.
From the Test Cases Generator	<ol style="list-style-type: none">In the Test Configurations tab, click the Test Combinations button.Generate your test configurations as described in ""Use the Test Combinations Generator to create test configurations" on page 803"

Enter static data configuration values

In UFT, static data configuration values are read-only. To add data to the static configuration, 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*.

Map test parameters to the Excel file

After you associate the Excel file with a specific test configuration, instruct UFT where to provide the values for your test parameters:

- In the Test Configurations tab, select the **Test Parameters Mapping** tab. A list of all test parameters is displayed.

2. In the **Resource Column**, from the drop-down list, select the name of the column to which you want to map the parameter.
3. (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.

View component parameter mapping details

1. In the Test Configurations tab, select the **Components Automap** tab. A list of all components in the test is displayed.
2. In the components list, select the component you want to view.
3. 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.

Run the test with the selected configuration

1. In the Run dialog, expand the **Options** node.
2. From the Options area, select the **Test Configurations** tab.
3. 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.

Use the Test Combinations Generator to create test configurations

Relevant for: business process tests

This task describes how to use the Test Combinations 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 "["Set up and run test configurations" on page 800](#).

Tip: For a use-case scenario related to this task, see "["Use-Case Scenario: Use the Test Combinations Generator" on page 796](#).

Prerequisites

Before using the Test Combinations Generator, create test parameters.

Open the Test Combinations Generator

In the Properties pane, select the **Test Configurations** tab  , and then click the **Test Combinations** button.

Set the value of your parameters

Enter the possible values for your parameters as follows:

Manually	Click on a cell and enter the values for the parameters as needed or cut and paste into the grid as needed.
-----------------	---

Import

Click the **Import** button to automatically import values for the parameters. Import an Excel file from the file system or an ALM project.

Automatically generate values for parameters

1. In the Parameter Values pane of the Test Combinations Generator, in the upper right hand corner, click **Generate Parameters**.
2. In the Generate Parameters pane, from the Parameter drop-down list, select the **Parameter** to generate.
3. From the **Generation Type** drop-down list, select the type of value to generate.
4. For certain types of values, select the other options as needed. For example, for the First Name or Full Name value types, you can select **English** or **Non-English** types.
5. Specify the number of entries to generate.

The Test Combinations Generator only generates unique items, without duplicating values. For example, if you instruct UFT to generate 20 entries, but limit the available values to the numbers between 95 and 100, only 6 entries are generated.

6. In the lower right corner of the pane, click **Generate**. The values are added to your list of parameter values used to generate test cases.

Add a custom data generator

In addition to using UFT's built-in dictionaries, you can create a custom data generator for the parameter types in your test:

1. Create an **.xml** file for your data generator.
2. In the XML file, add the following XML:

```
<Generators>
    <DataGenerator name="[CustomGeneratorName]" type="[GeneratorType]" title="[GeneratorTitle]" returnType="[GeneratorResultType]">
        ...
    </DataGenerator>
</Generators>
```

name

A unique name for the generator type.

<i>type</i>	The generator type. Possible values: <ul style="list-style-type: none"> • HP.UFT.TCG.DataGeneration.Generators.RegExpGenerator: to specify the value format using a regular expression • HP.UFT.TCG.DataGeneration.Generators.BaseRepositoryGenerator: to specify the values with standard strings, numbers, etc.
<i>title</i>	The title of the Generation Type as it is displayed in the Test Configurations Generator.
<i>returnType</i>	The format of the values to return. Supported types include: <ul style="list-style-type: none"> • String • Password • Number • Date



Note: You can add multiple generator types in this XML file.

3. In the Data Generator attribute, specify the parameter details using the **Parameter** element:

```
<Generators>
    <DataGenerator name="[CustomGeneratorName]" type="[GeneratorType]" title="[GeneratorTitle]" returnType="[GeneratorResultType]">
        <Parameters>
            <Parameter .... />
        </Parameters>
    </DataGenerator>
</Generators>
```

The attribute values for the Parameter element differ if you are using the regular expression type (**HP.UFT.TCG.DataGeneration.Generators.RegExpGenerator**) or the regular tyle (**HP.UFT.TCG.DataGeneration.Generators.BaseRepositoryGenerator**):

Regular expression	<ul style="list-style-type: none"> • name: must be the string pattern • defaultValue: the default regular expression pattern • type: must be the string System.String
Non-regular expression	<ul style="list-style-type: none"> • name: a unique name for the parameter • defaultValue: the default parameter value • type: must be the string System.Boolean • title: the parameter title

- For non-regular expression-based parameter types, add the **Repositories** element:

```
<Generators>
    <DataGenerator name="[CustomGeneratorName]" type="[GeneratorType]" title="[GeneratorTitle]" returnType="[GeneratorResultType]">
        <Parameters>
            <Parameter .... />
        </Parameters>
        <Repositories>
            <Repository.... />
        </Repository>
    </Repositories>
</DataGenerator>
</Generators>
```

The Repository element must contain the following attributes:

<i>name</i>	A unique name for the repository.
<i>path</i>	The path to the Excel file containing the possible values.
<i>useWhenCheck</i>	The name of the parameter entered in the Parameter element. This instructs UFT to use the repository when a checkbox for one of the parameter values is checked.
<i>rule</i>	The format for data generation.

Example

```
<DataGenerator name ="FullNameGenerator" type
="HP.UFT.TCG.DataGeneration.Generators.BaseRepositoryGenerator"  title="Full
Name" >
    <Parameters>
        <Parameter name ="english" defaultValue ="true" type ="System.Boolean" title
= "English"/>
        <Parameter name ="non-english" defaultValue ="false" type ="System.Boolean"
title = "Non-English"/>
    </Parameters>
    <Repositories>
        <Repository name ="fr_full_names" path ="FrenchFirstLastNames.xlsx"
useWhenCheck ="non-english">
            <Rule>[MaleName] [Surname]</Rule>
            <Rule>[FemaleName] [Surname]</Rule>
        </Repository>
    </Repositories>
</DataGenerator>
```

```
<Repository name ="en_full_names" path ="EnglishFirstLastNames.xlsx"
useWhenCheck ="english">
  <Rule>[MaleName] [Surname]</Rule>
  <Rule>[FemaleName] [Surname]</Rule>
</Repository>
<Repository name ="jp_full_names" path ="JapaneseFirstLastNames.xlsx"
useWhenCheck ="non-english">
  <Rule>[MaleName][Surname]</Rule>
  <Rule>[FemaleMaleName][Surname]</Rule>
  <Rule>[FemaleName][Surname]</Rule>
</Repository>
<Repository name ="ru_full_names" path ="RussianFirstLastNames.xlsx"
useWhenCheck ="non-english">
  <Rule>[MaleName] [MaleSurname]</Rule>
  <Rule>[MaleName] [FemaleMaleSurname]</Rule>
  <Rule>[FemaleName] [FemaleSurname]</Rule>
  <Rule>[FemaleName] [FemaleMaleSurname]</Rule>
</Repository>
</Repositories>
</DataGenerator>
```

Set values as Happy Path or Error Path

Select a component value and click one of the following:

Happy path	The value is expected not to cause an exception or error.
Error path	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).

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:

Regular/Happy path values	All possible combinations of the values are listed in the Permutations tab.
Error path values	All possible combinations are listed in the Error Paths tab, using the error path values.

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.

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.

1. In the Permutations tab, click the **Filter** button.
2. In the filter, select the combinations to generate.
3. If necessary, enter a name for the generated configuration.

Generate the test configurations

In the bottom of the Test Combinations 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.

Known Issues - Running tests from ALM

Run results

Fast run results	<p>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.</p> <p>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.</p>
-------------------------	---

ALM versions earlier than 12.50	If you run tests from UFT that are stored in an ALM version lower than 12.50, the Run Sessions > Report Format option is automatically set to Run Results Viewer Report . Run results are opened in the Run Results Viewer (RRV).
--	--

ALM versioning

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.

External authentication

Accessing the ALM Test Plan	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. For details about setting up certificates and the Webgate Customization tool, see the documentation for the Webgate Customization tool included with your ALM server.
Unable to connect	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.

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 details, see the documentation for the Webgate Customization tool for your ALM server.

Stopping a test in the middle of a run

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.

Test Combinations Generator

The Test Combinations Generator may work unexpectedly if BPT test input parameters have names beginning with numbers.

For example: **1_name**

To avoid issues, always start BPT test input parameter names with letters.

Running API tests on Windows 2012 R2

Network COM+ access is not enabled by default on Windows 2012 R2, and is required for running API tests from ALM.

Enable it manually before running your API tests as follows:

1. Browse to the Windows **Add or Remove Programs** control panel.
2. Click **Turn Windows features on or off**.
3. In the **Add Roles and Features Wizard**, click **Server Roles**, and then select **Application Server**.
4. Expand the details of the child features, and select the **COM+ Network Access** option.
5. Click **Next** until you complete the Windows Components wizard, and then click **Install** to install the features.

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*.

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.

You can do any of the following:

Add assets	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. 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	<p>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.</p> <p>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.</p> <p>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.</p>
View version control information	<p>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.</p> <p>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.</p>
View and compare asset versions	<p>You can view and compare the versions of an asset using the Asset Comparison Tool.</p> <p>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.</p>

View baseline history	<p>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.</p> <p>Baselines can be created for ALM projects that are enabled for version control, and for projects for which version control is not enabled.</p> <p>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.</p> <p>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.</p>
------------------------------	---

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:

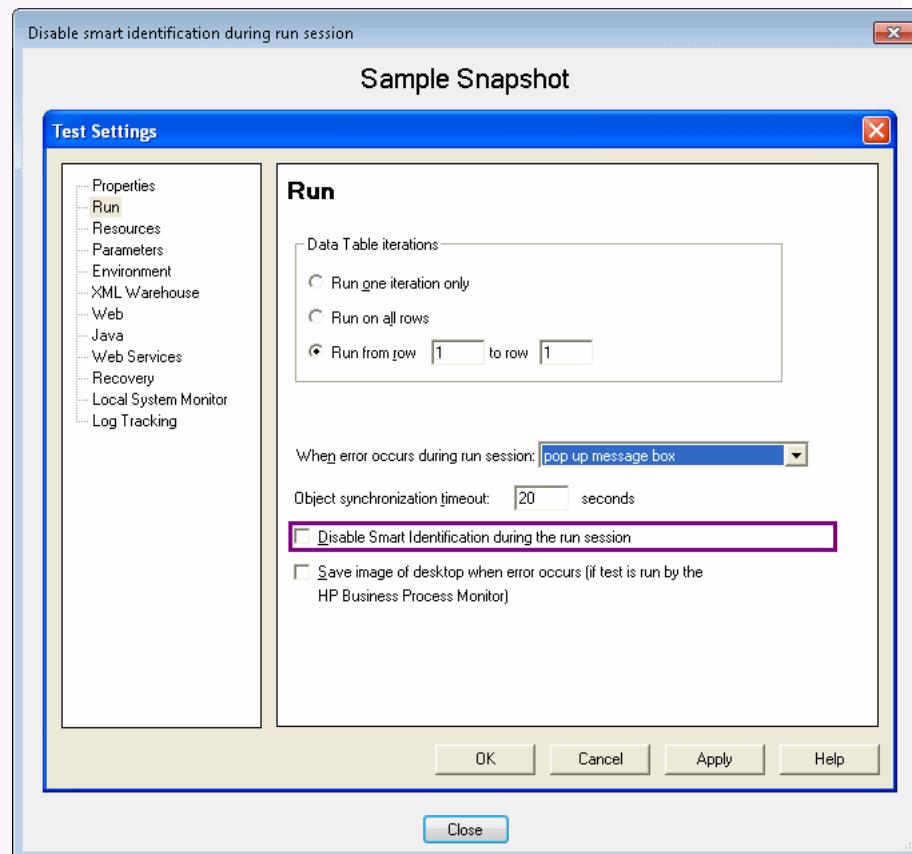
View any version of an asset using the Asset Viewer	After you select a specific version in the Version History dialog box, you can see specific details about that version of the asset, even if it is different from the working copy of the asset. Using this feature enables you to see where specific settings and details of the asset have been changed.
Compare two versions of an asset using the Asset Comparison Tool	After you have selected the specific versions to view in the Version History dialog box, you can view differences between the two versions.
Drill down to view or compare versions	<ul style="list-style-type: none">• View or compare versions of an integral element. 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.• View or compare versions of associated external assets. 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). <p>To view or compare by drilling down, the resource file must be associated via an absolute or ALM path.</p>

View a screen capture depicting the UFT location of an element (GUI testing only)

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.



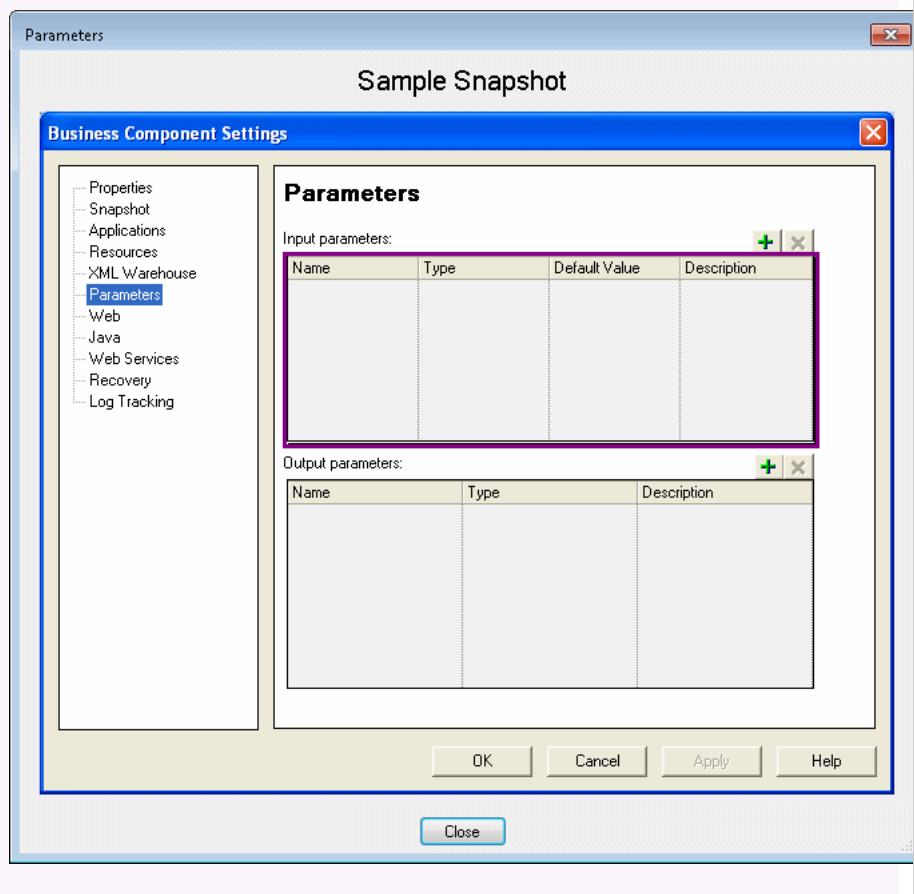
Example: Suppose you are viewing a comparison of a test and you notice that the **Disable Smart Identification during the run session** node is highlighted, indicating that it was modified. If you are not sure where this option is located in UFT, you can right-click the node in the comparison tree and 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 "[Work with the Asset Comparison Tool and Asset Viewer](#)" on page [818](#).

Use ALM version control

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

Check in the currently open asset

1. Confirm that the currently open asset is checked out to you.
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.
2. Open the resource, as follows:

If the asset is a:	Do this:
Test, Component or Function Library	In the main UFT window, select File > Open > and select Test, Business Component, Function Library, or Application Area or click the Open down arrow and select the asset type from the list.
Shared Object Repository (GUI testing only)	In the Object Repository Manager, select File > Open or click the Open button.
Recovery Scenario (GUI testing only)	In the Recovery Scenario Manager, click the Open 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.

Work with the Asset Comparison Tool and Asset Viewer

Relevant for: GUI tests and components and API testing

Open the Asset Viewer

You can open the Asset Viewer from any of the following:

The main UFT window	<ol style="list-style-type: none">1. Open the test, component, or function library for which you want to view an earlier version.2. Select ALM > Version History. The Version History dialog box opens.3. Select a version and click View. The Asset Viewer opens.
The Object Repository Manager (GUI Testing only)	<ol style="list-style-type: none">1. Open the Object Repository Manager (Resources > Object Repository Manager).2. Browse to and open the shared object repository for which you want to view an earlier version. For details, see "Create and manage shared object repositories" on page 223.3. Select ALM > Version History. The Version History dialog box opens.4. Select a version and click View. The Asset Viewer opens.
The Recovery Scenario Manager (GUI Testing only)	<ol style="list-style-type: none">1. Open the Recovery Scenario Manager (Resources > Recovery Scenario Manager).2. Open the recovery scenario file for which you want to view an earlier version.3. Click the Version Control down arrow and select Version History.4. Select a version and click View. The Asset Viewer opens.

From within ALM	<p>Connect to the project containing the asset you want to view, and do one of the following:</p> <ul style="list-style-type: none">• To view the current version of an asset: In the Test Resources module, select the resource and click the Resource Viewer tab.• To view the current or earlier version of an asset:<ol style="list-style-type: none">a. Do one of the following:<ul style="list-style-type: none">◦ In the sidebar, click the Test Plan button (for tests) or the Business Components button (for components) to open the Test Plan or Business Components module.◦ Click the Test Resources button to open the Test Resources 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.b. In the tree, select the file for which you want to view an earlier version.c. Click the History tab, and then click the Versions or Baselines tab.d. In the grid, select a version, and then click the View 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.
The Command Line Interpreter (cmd.exe) e) (tests only)	<ol style="list-style-type: none">1. Open the Command Line Interpreter.2. Enter the command using the following syntax:<pre>"<UFT installation folder>\bin\UFTDiffApplication.exe" P1:<file path 1>"</pre>where P1 = the file system path to the asset.<p> Example: "%ProgramFiles%\HP\Unified Functional Testing\bin\UFTDiffApplication.exe" P1: "%ProgramFiles%\HP\Unified Functional Testing\Tests\Test1"</p>

Open the Asset Comparison Tool

You can open the Asset Comparison Tool from any of the following:

The main UFT window	<ol style="list-style-type: none">1. Open the test, component, or function library (GUI testing only) whose versions you want to compare.2. Select ALM > Version History or Baseline History. The Version History or Baseline History dialog box opens.3. Select two versions (using the CTRL key) and click Compare. The Asset Comparison Tool opens.
The Object Repository Manager (GUI testing only)	<ol style="list-style-type: none">1. Open the Object Repository Manager (Resources > Object Repository Manager).2. Browse to and open the shared object repository whose versions you want to compare.3. Select ALM > Version History or Baseline History. The Version History or Baseline History dialog box opens.4. Select two versions (using the CTRL key) and click Compare. The Asset Comparison Tool opens.
Recovery Scenario Manager (GUI testing only)	<ol style="list-style-type: none">1. Open the Recovery Scenario Manager (Resources > Recovery Scenario Manager).2. Open the recovery scenario file whose versions you want to compare.3. Click the Version Control down arrow and select Version History or Baseline History.4. Select two versions (using the CTRL key) and click Compare. The Asset Comparison Tool opens.

- | | |
|------------------------|--|
| From within ALM | <ol style="list-style-type: none">1. In ALM, connect to the project containing the asset you want to compare.2. Do one of the following:<ul style="list-style-type: none">• In the sidebar, click the Test Plan button (for tests) or the Business Components button (for components) to open the Test Plan or Business Components module.• Click the Test Resources button in the sidebar to open the Test Resources 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.3. In the tree, select the file whose versions you want to compare.4. Click the History tab, and then click the Versions or Baselines tab.5. In the grid, select two versions to compare (using the CTRL key), and then click the Compare button.6. In the sidebar of the window that opens, click the QTP Comparison/ST Comparison or Automation button. The Asset Comparison Tool opens. |
|------------------------|--|



Tip: You can also compare baselines from the **Management** module. Click the **Management** button in the side bar to open the **Management** module. Select a baseline in the tree and click the **Compare To**  button. For details, see the ALM user guide.

The Command Line Interpreter (cmd.exe) tests only)	<ol style="list-style-type: none">1. Open the Command Line Interpreter.2. Enter the command using the following syntax:<pre>"<UFT installation folder>\bin\UFTDiffApplication.exe" P1: "<file path 1>" P2: "<file path 2>"</pre>where P1 = the file system path to the first asset, and P2 = the file system path to the second asset. <p>Note: 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>Example: "%ProgramFiles%\HP\Unified Functional Testing\bin\UFTDiffApplication.exe" P1: "%ProgramFiles%\HP\Unified Functional Testing\Tests\Test1" P2: "%ProgramFiles%\HP\Unified Functional Testing\Tests\Test2"</p>
---	--

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

- 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

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 813.

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.



Example: In the following example, three sub-elements were modified, one was deleted, and seven were added:

Test Settings [3] [1] [7]

Known Issues - ALM Version Control

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

For GUI tests and components	<ul style="list-style-type: none">• 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:<ol style="list-style-type: none">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.<p>Workaround: If this occurs, close the Asset Comparison Tool, and then open it again after you save your changes.</p>• 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	<ul style="list-style-type: none">• 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:<ol style="list-style-type: none">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.<p>Workaround: If this occurs, close the Asset Comparison Tool, and then open it again after you save your changes.</p><ul style="list-style-type: none">• 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.
-------------------------------------	---

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:

Run ALM manual tests and Business Process tests with new step display	<ul style="list-style-type: none">Clear steps display. Steps are presented in a clear, organized, and user-friendly design, making it easier to view step information and update step status.Move easily between tests in your run. 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.Edit actual parameter values during your test run. You can easily edit the actual values of parameters in your test, during your test run.Multiple views. Change the way you view your steps depending on your testing needs. View in normal mode when more details are needed, or view in Subtitles mode if you need to see more of your application.Actual value including screen captures. Attach a plain or annotated screen capture of your application to the step's actual value.
Create formal tests from exploratory tests with no pre-defined steps.	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.
Submit defects to ALM	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.
Create and annotate screen captures of your application.	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.

Let Sprinter perform some manual testing tasks for you.	You can create and run macros to automate a set of actions in your application. Sprinter can also inject data automatically into fields in your application.
Replicate your actions on another computer.	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.
View test results.	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.
Use Sprinter tests to create UFT tests and business components	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

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.

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 (including keyword GUI components, scripted GUI components, or manual 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.

Note: Business Process Testing is available with ALM Edition and Quality Center Enterprise Edition. For more information about the HPBusiness Process Testing editions and their functionality, see the *HP Application Lifecycle Management User Guide*. To find out what edition of HPBusiness Process Testing you are using, ask your ALM site administrator.

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.

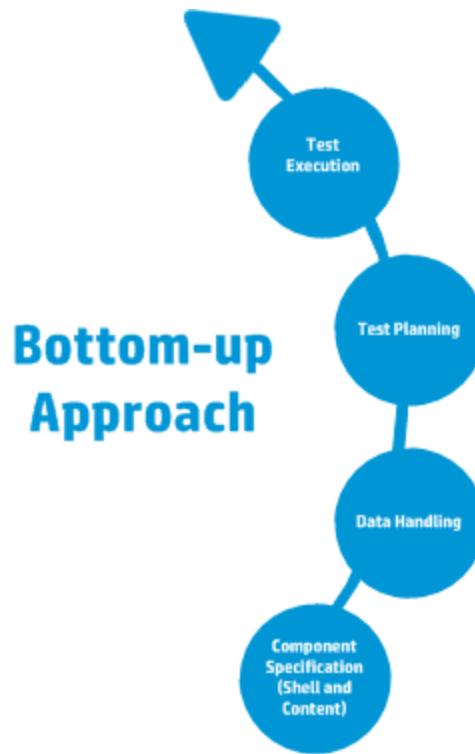
You can use any of the following methodologies:

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

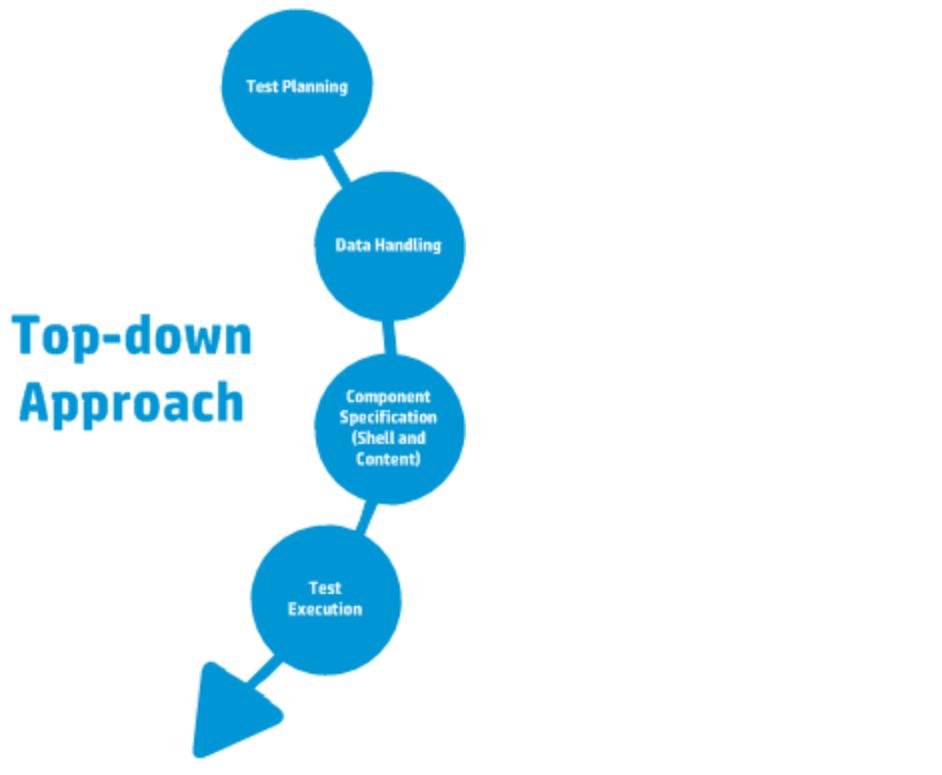
Using the approach enables you to create resources - components, application areas, and object repositories that can be reused across tests. For example, you can use the same component in a number of tests. Likewise, you can design many different components that contain the same application area (which is based upon a specific area of your application).

The bottom-up methodology is based on the following design phases:



Top-down Methodology	<p>The top-down methodology advocates the creation of business process testing entities according to the following hierarchy:</p> <ul style="list-style-type: none">• 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:



Agile Methodology	<p>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.</p> <p>This approach encourages:</p> <ul style="list-style-type: none">• 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. <p>The following presents the Agile Development-centric approach.</p>
--------------------------	---

The diagram illustrates the Agile Development-Centric Approach. It features a central circle containing the text "Agile Development-Centric Approach". This central circle is connected by lines to four smaller circles arranged in a square pattern around it. Above this central group is a horizontal bar divided into two sections: "Development Activities" on the left and "Testing Activities" on the right. The left side of the bar is labeled "Sprint Starts" and the right side is labeled "Sprint Ends". The entire diagram is set against a light gray background.

The chapters in this guide are structured according to the bottom-up methodology.

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

Function	ALM	UFT
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	
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

Set up UFT for Business Process Testing

Relevant for: business process tests and flows

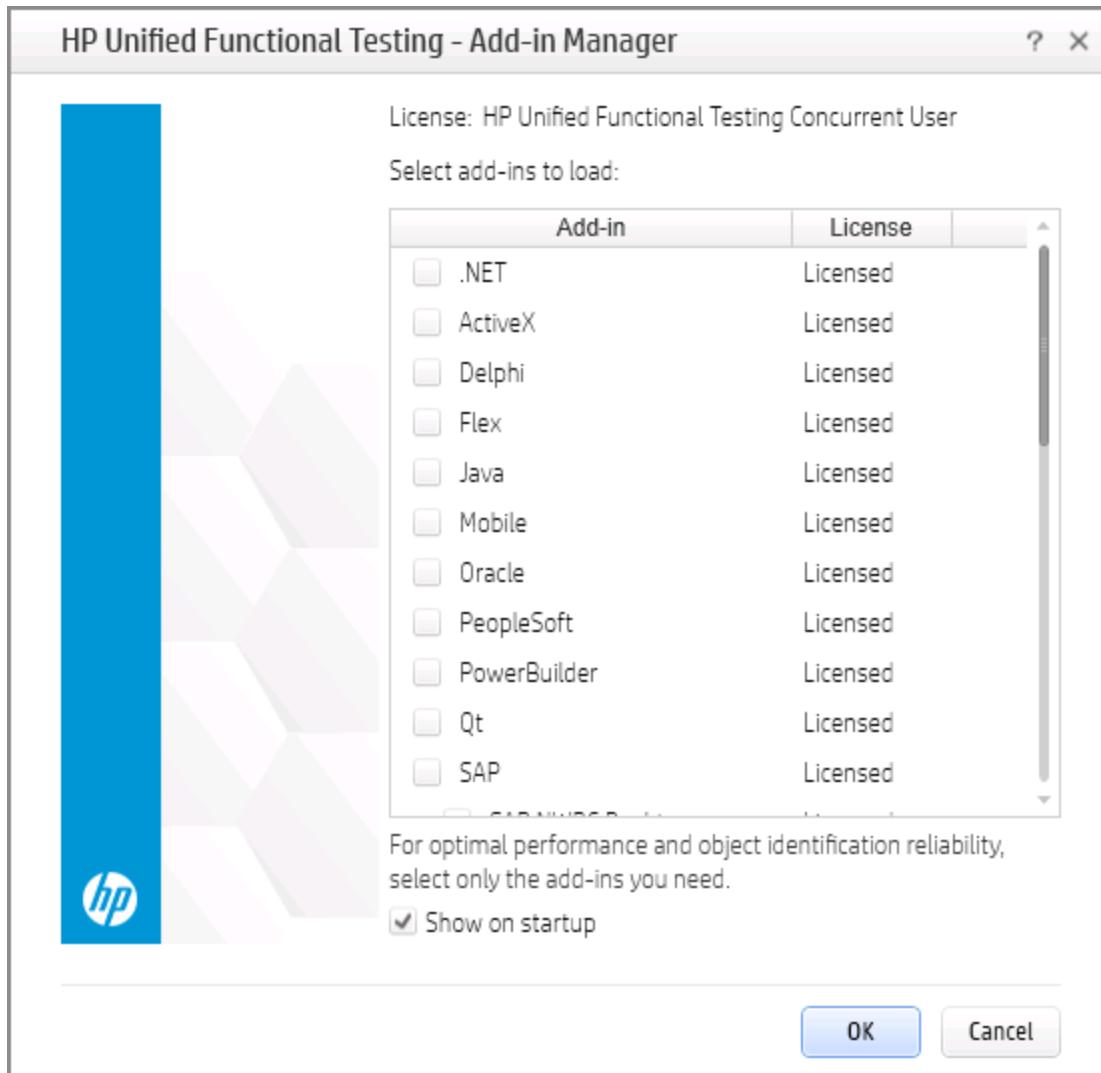
Before you create business process tests and steps, you must set up UFT. This enables you to create your tests without having to stop frequently to troubleshoot UFT program issues.

Prerequisites

- 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

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 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:

The screenshot shows the 'Additional Settings' tab selected in the left sidebar of the UFT interface. Under the 'Applications' section, the 'Record and run only on:' option is selected, with the 'Applications opened by UFT' checkbox checked. A table titled 'Application details' lists 'C:\Program Files (x86)...' under 'Application' and 'C:\Program Files (x86)...' under 'Working Folder'. The 'Program Arg' column is empty.

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:

Component-centered (or bottom-up)	<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 business processes of your application may require additional planning.</p>
Test flow-centered (or top-down)	<p>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.</p> <p>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.</p>

Create and maintain business process 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.

Prerequisites

- Load the necessary add-ins in the Add-in Manager on UFT startup.
- Connect to an ALM server and project.

Create application areas for each area of your application

Before creating tests and their components, create application areas for each area of your application. The application area contains the object repositories with the test objects, function libraries, and specific settings to use for the component.

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.

From the Toolbox pane	<ol style="list-style-type: none"> 1. Open the Toolbox pane. 2. Under the Components tree, expand the nodes and navigate to the component or flow you want to add. 3. Double-click or drag and drop the component to the test grid or canvas. 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.
While recording	<ol style="list-style-type: none"> 1. In the toolbar, click the Record button . 2. In the New Business Component dialog box, specify a Name, Location, and default Application Area for the test. 3. Record user actions on your application. 4. As your record, in the Record toolbar, click the New Business Component button . 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. 6. When you are finished recording, click the Stop button . UFT adds all the recording components to the test with the recording steps (from the steps you performed on your application).

Add steps to your component

For details, see "[Create test steps in a business process test](#)" on page 863.

Group components and flows

In the document pane (grid view or canvas view), select the components or flows you want to group, and in the toolbar, click **Group** .

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 ["Use data in a business process test" on page 877](#)

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 880](#) and ["Set data values for the parameters for each iteration" on page 880](#).

Add a test configuration

You can add test configurations to your test to enable you to run the test with varying sets of data. For details, see ["Set up and run test configurations" on page 800](#).

Debug and run your test

To debug a test or component	Insert breakpoints in specific components or flows in your test, and then run your test. The run session stops at each breakpoint. 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. However, before running the BPT test, you must open the business components with the breakpoints and add them to the solution in UFT.
To run your test:	In the toolbar, click the Run button  . <ul style="list-style-type: none">Before running a test from ALM, you must enable the Allow other HP products to run tests and components option in the Test Runs pane (Tools > Options > GUI Testing tab > Test Runs node)For improved performance when running business process tests or flows, UFT creates and runs a hosting test, named Test Runtime. The Test Runtime test is recreated each time the test or flow runs, and is not saved with the run.

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**).

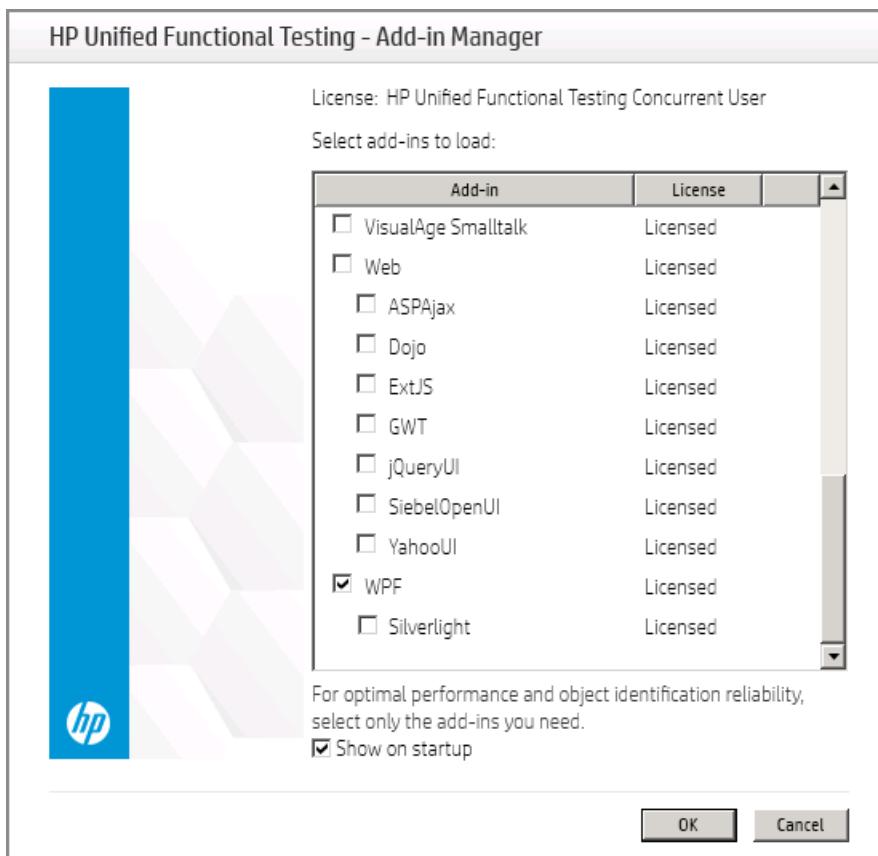
1. Analyze your application

Determine the technologies used in your applications.	<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>
Determine the functionality that you want to test.	<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">a. Login window: The user logs in to the flight reservation applicationb. Flight search window: The user searches for available flights or booked flights, including:<ul style="list-style-type: none">◦ Find a flight based on departure and arrival cities and departure date◦ Find a flight based on the seating class of the passenger◦ Search for booked flights by customer name, flight number, or flight datec. Flight selection window: Select a flight from the available flightsd. Flight confirmation and booking window: Confirm the selected flight and entering the customer details, including:<ul style="list-style-type: none">◦ Entering the customer's name◦ Entering the exact number of tickets needed◦ Booking the flight◦ Waiting for confirmation of the order from the application◦ Restarting the reservation process

Decide how to divide the testable functionality into smaller units	By dividing the test into smaller chunks, you can make it more manageable and easier to follow. Based on this list of application windows and tasks, you can create four different components for the different areas of the application.
---	--

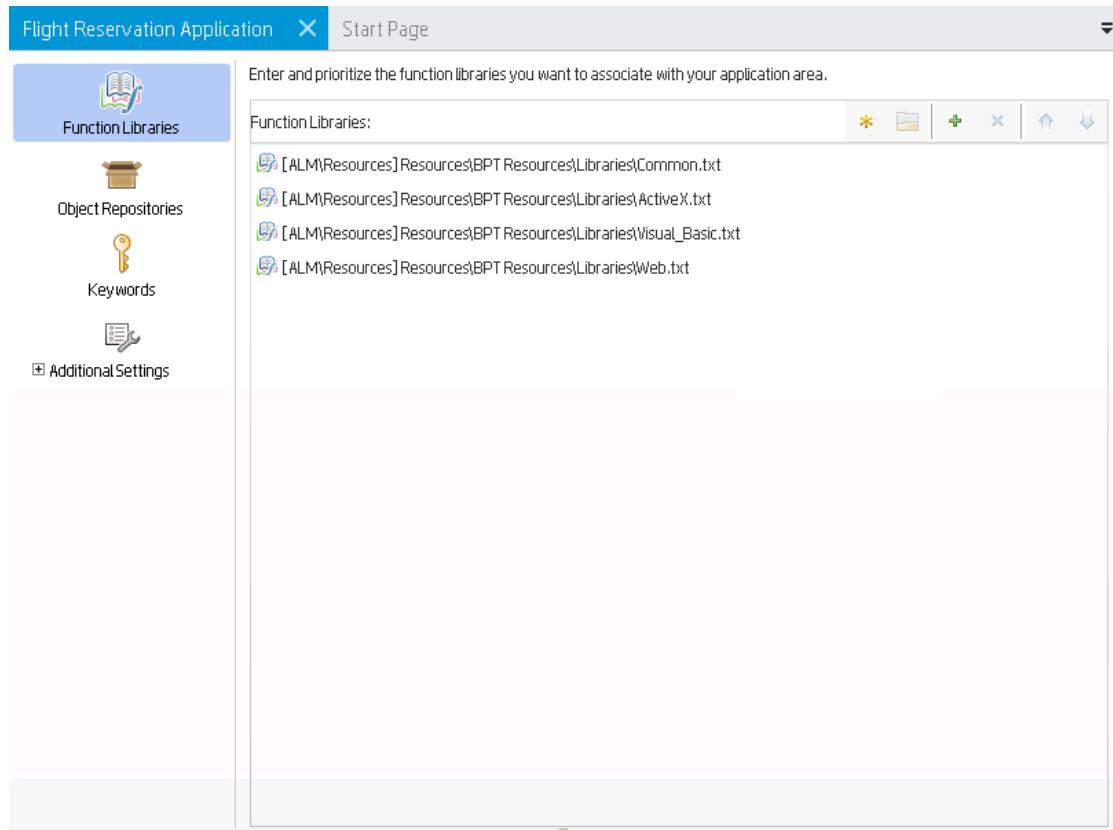
2. Prepare your test infrastructure.

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.

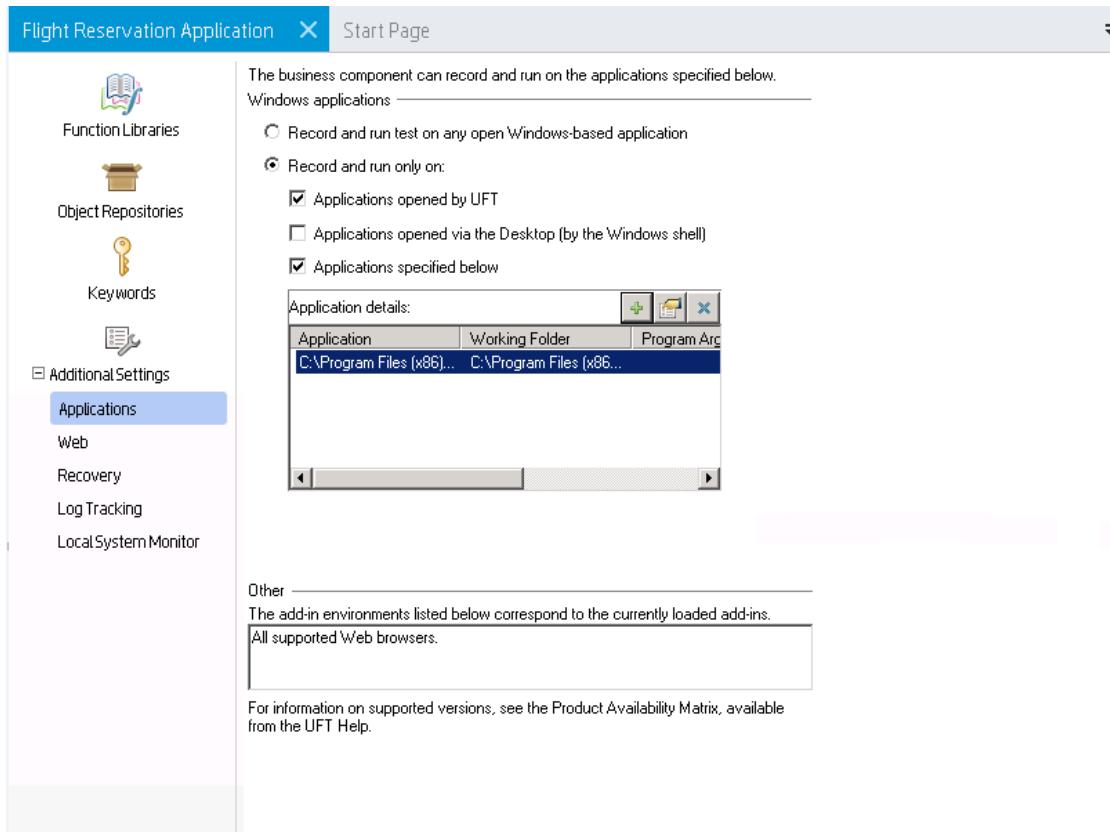


Second, after UFT starts, [you must then connect to ALM](#) to enable yourself to use Business Process Testing with UFT. Once you are successfully connected with ALM, UFT displays an icon  in the lower right corner of the main window.

Third, you must create or have a default application area. This application area contains the object repositories, custom function libraries, and application settings:



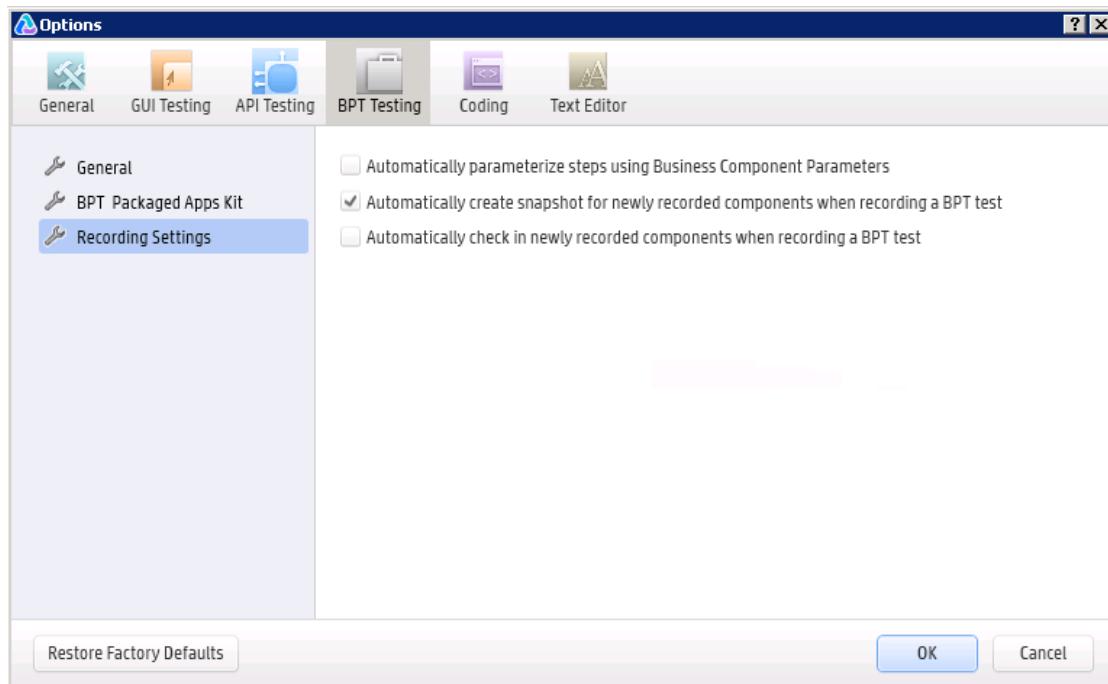
Lastly, in the application area, settings you must configure UFT to work with the flight reservation application:



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.

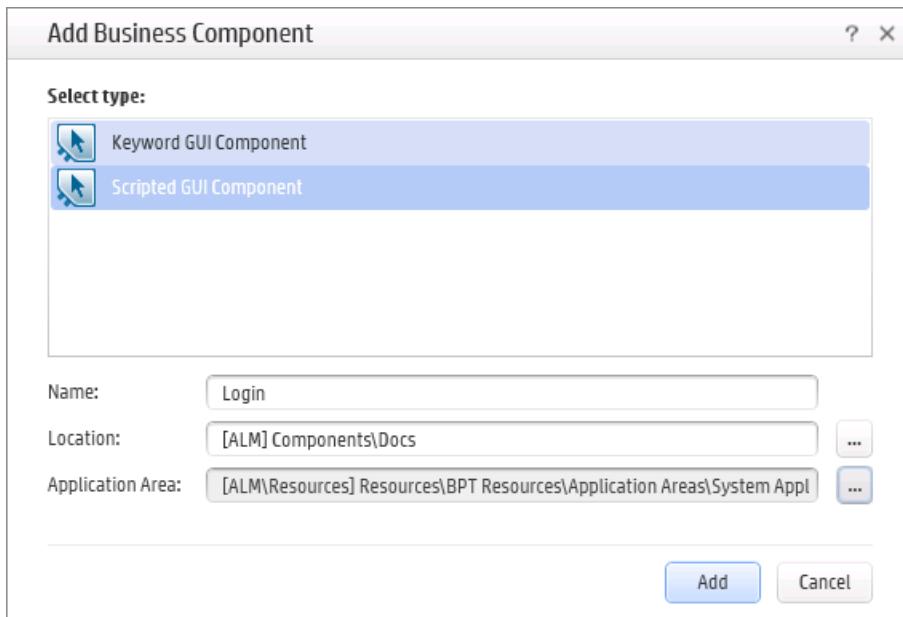
Before recording, set the options for how UFT records a business process test in the **Recording Settings** pane of the Options dialog box (**Tools > Options > BPT Testing tab > Recording Settings node**):



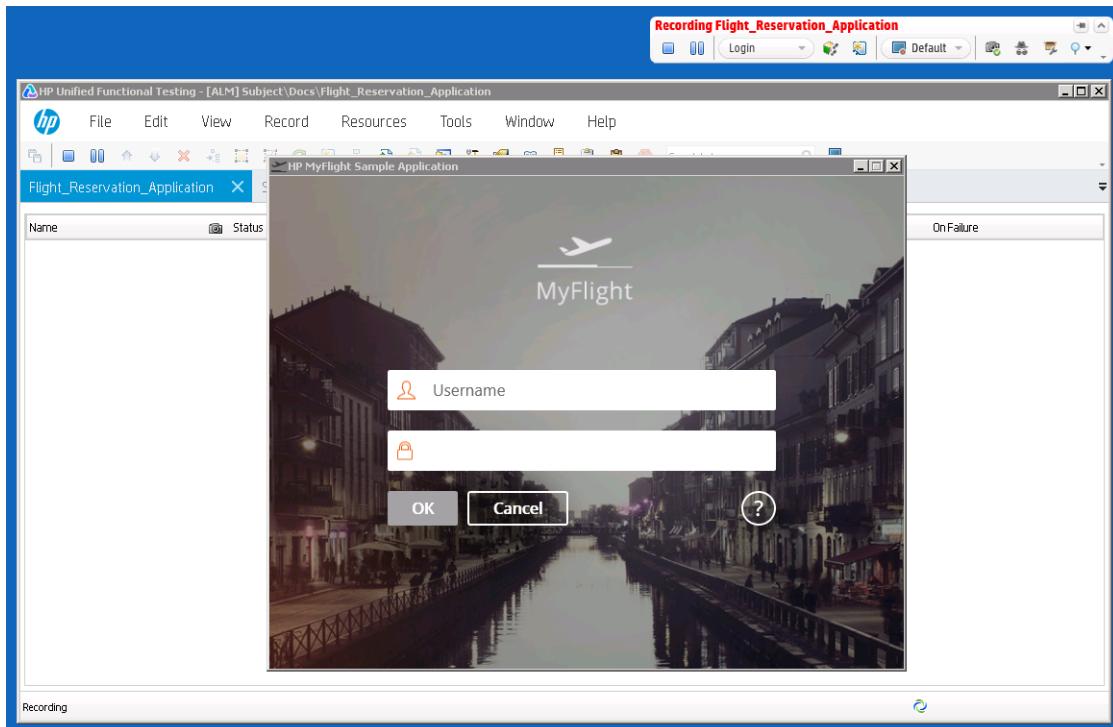
For this scenario, you instruct UFT to **Automatically parameterize steps using Business Component Parameters**. This enables UFT to create a parameter for each object in your application. After recording your test, you can later parameterize steps in the test using these parameters.

However, before you can begin adding steps to 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:



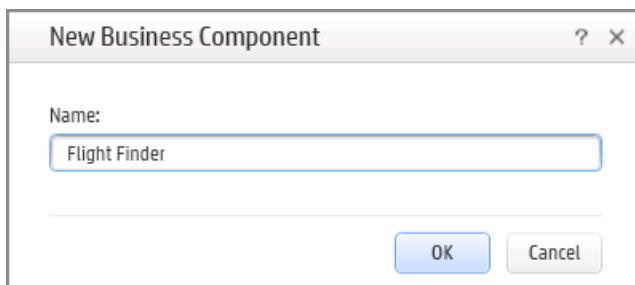
After you add this business component, the UFT window goes into the background and the Record Toolbar is minimized. Open the flight reservation application:



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:

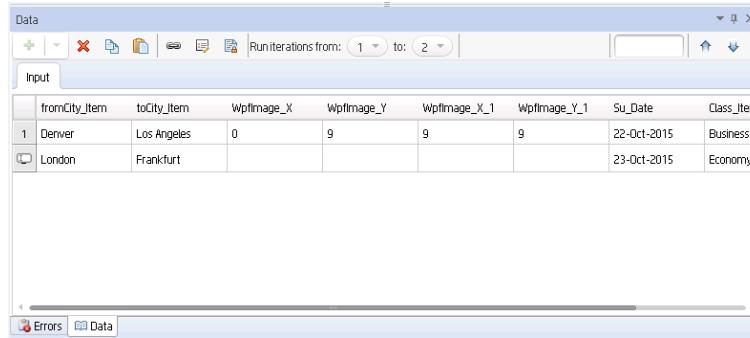
Name	Status	I/O Parameters	Iterations	Run Condition	On Failure
Login [1]	Under Development	8 In	1 Iterations	Continue	
Flight Finder [1]	Under Development	9 In	1 Iterations	Continue	
Select Flight [1]	Under Development		1 Iterations	Continue	
Flight Confirmation [1]	Under Development	1 In	1 Iterations	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:

Group components	Select components and group them together to enable certain functionalities (like common parameterization, running, etc.) for all components in the group:																																				
	<table border="1"> <thead> <tr> <th>Name</th><th>Status</th><th>I/O Parameters</th><th>Iterations</th><th>Run Condition</th><th>On Failure</th></tr> </thead> <tbody> <tr> <td>Login [1]</td><td>Under Development</td><td>8 In</td><td>1 Iterations</td><td>Continue</td><td></td></tr> <tr> <td>Group 1</td><td></td><td></td><td>2 Iterations</td><td></td><td></td></tr> <tr> <td>Flight Finder [1]</td><td>Under Development</td><td>9 In</td><td></td><td>Continue</td><td></td></tr> <tr> <td>Select Flight [1]</td><td>Under Development</td><td></td><td></td><td>Continue</td><td></td></tr> <tr> <td>Flight Confirmation [1]</td><td>Under Development</td><td>1 In</td><td></td><td>Continue</td><td></td></tr> </tbody> </table>	Name	Status	I/O Parameters	Iterations	Run Condition	On Failure	Login [1]	Under Development	8 In	1 Iterations	Continue		Group 1			2 Iterations			Flight Finder [1]	Under Development	9 In		Continue		Select Flight [1]	Under Development			Continue		Flight Confirmation [1]	Under Development	1 In		Continue	
Name	Status	I/O Parameters	Iterations	Run Condition	On Failure																																
Login [1]	Under Development	8 In	1 Iterations	Continue																																	
Group 1			2 Iterations																																		
Flight Finder [1]	Under Development	9 In		Continue																																	
Select Flight [1]	Under Development			Continue																																	
Flight Confirmation [1]	Under Development	1 In		Continue																																	

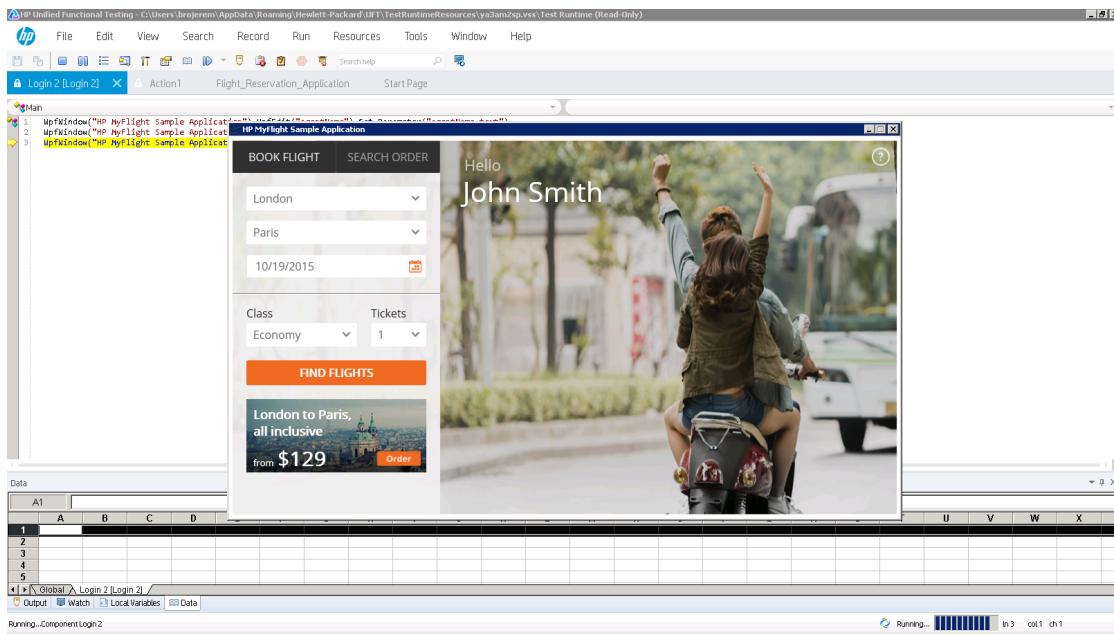
Change parameter values across iterations	<p>If you select any component in the test, and then view the Parameters tab in the Properties pane, you see that UFT has automatically created component parameters for the objects used in the component steps.</p> <p>In the data pane, add additional iterations and change the values of the parameters in other iterations:</p>  <p>The screenshot shows the UFT Data pane with the following data:</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																				
Set run conditions for components in the test	If you select a component, in the Properties tab of the Properties pane, you can specify to stop (Exit) or continue (Continue) the test if a component fails on a particular test run.																											
Create a test configuration to change the parameterization of your components across test runs	<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



Chapter 65: Business Components and 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.

When building a business process test, your test consists of business components. Inside of each business component is an application area that contains the necessary resources to run the component.

What are business components?

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 one or more of the following methods to categorize components:

Logical Components	<p>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.</p> <p>Example:</p> <ul style="list-style-type: none">• 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	<p>An application object component might represent an object on the screen or a call to a single API.</p> <p>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.</p> <p> Example:</p> <ul style="list-style-type: none"> • 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	<p>A generic component performs actions outside of the context of the application under test. It can be reused in tests of different applications.</p> <p> Example: A Launch component represents the launching of a browser.</p>

When you create a component, you can select one of the following types:

Keyword GUI components	<p>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.</p> <p>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.</p>
-------------------------------	--

Scripted GUI components	<p>Scripted components are maintainable, reusable scripts that perform a specific task.</p> <p>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:</p> <ul style="list-style-type: none">• 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.
API components	API components enable you the same basic functionality as an API test with limitations on the types of activities that are useable.

What are application areas?

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.

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.

Converting manual components to UFT 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. 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.

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"

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 convert a manual component to an automated component from within ALM. For details, see the *HP Business Process Testing User Guide*.

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.

Prerequisites

- Connect UFT to an ALM project.
- Ensure you have the required ALM permissions. 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.

Update a component from an earlier QuickTest version

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.

All components last modified in versions of QuickTest later than 9.5 can be opened in UFT without conversion.

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.

Convert a manual component to an automated component



Caution: Converting manual 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 it
 - 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.

Convert the keyword GUI component to a scripted GUI component

1. Open the keyword component you want to convert.
2. Select **File > Convert to Scripted Component**. When prompted, click **OK** to proceed with the conversion.

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.

Manage application areas

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.

Application areas contain four different areas in which to manage application area resources:

	Function Libraries pane. Enables you to create, associate, or open associated function libraries with your component.
	Object Repositories pane. Enables you to create, associate, or open associated object repositories.
	Keywords pane. Enables you to select the available objects available for each loaded add-in and the available methods for these objects.
	Additional Settings pane. Enables you to configure behavior for the application area and component for: <ul style="list-style-type: none">• Settings for how UFT runs and records on Windows-based applications• Settings for running Web tests• Recovery scenario details• Log tracking and system monitor configuration

Known Issues- Business Components and Application Areas

Relevant for: GUI components only

This section describes troubleshooting and limitations for business components.

- 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.

- You can work with API components in much the same way as API tests, with some limitations:
 - 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.

Chapter 66: Creating Business Process Test Steps

Relevant for: business process tests

After you create the structure of your business process test or flow, you must then add test steps inside each component. These test steps perform user actions on your application and actually test your application.

In order to create test steps, you must do a number of things:

Create object repositories	<p>To create test steps, you must have test objects. These test objects are saved in object repositories which are then associated with the component (if you record the test steps) or the component's application area.</p> <p>Add test objects to an object repository:</p> <ul style="list-style-type: none">• In a component's local object repository: Using the Capture toolbar, scan all the objects in your application or in a specified area of your application. UFT saves these objects to the component's local object repository and the test objects are available for use in the associated component. <p>For details on how to capture test objects from your application, see "Add test objects to a component with Capture" on page 867.</p> <ul style="list-style-type: none">• In a shared object repository: In the Object Repository Manager, learn the objects in your application. Then, the shared object repositories is associated with one or more application areas. <p>For more detail on creating shared object repositories, see "Test Objects in Object Repositories" on page 206.</p>
Add object repositories to your application areas	<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. 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 and use the objects in test steps.</p> <p>For details on how to associate an object repository with your application area, see "Add object repositories to an application area" on page 864.</p>

Add test steps to the component	After you have the test objects available to use in your component's steps, you can add the test steps to the component: <ul style="list-style-type: none">By recording: 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).Manually: You can manually add test objects to the component by dragging them from the Toolbox pane, selecting them from the list of available objects in the Keyword View, or adding statements in the Editor.
Enhance your component's test steps	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: <ul style="list-style-type: none">Checkpoints: Checkpoint steps check the state of your application or its objects in the course of the test run. For additional details on checkpoints, see "Checkpoints in GUI Testing" on page 262.Functions: Function steps enable you to run a custom process within the test progress. For additional details on functions, see "User-Defined Functions" on page 575.Output values: 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 "Output Values in GUI Testing" on page 322.

For task details on how to add steps to your business process test, see "[Create test steps in a business process test](#)" below.

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 "[Create and maintain business process tests and flows in UFT](#)" on page 839.

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

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 206.

Add test objects using Capture

For details, see "[Add test objects to a component with Capture](#)" on page 867.

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

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. Select the component tab in the document pane.
2. In the component's tab, click in the **Item** column and from the dropdown list, select the relevant object or select **Object from repository**.
3. In the **Operation** column, select the necessary method (action) to perform on the object.
4. Add the necessary argument values to use when performing the method (action) on the object. If you hover over the Value cell you can see a description of the necessary arguments.

Add steps to your component by recording

For details, see "[Record a business process test](#)" below

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 automatically created.

Note: This task is part of a higher-level task. For details, see "[Create test steps in a business process test](#)" on page 863.

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.

Set recording options

1. 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:

Automatically parameterize steps using Business Component Parameters	Instruct UFT to create component parameters for each operation recorded and link the step values to these parameters.
Automatically check in newly recorded components when creating a BPT test	If you are working with a version-controlled ALM project, this instructs UFT to prompt you to check in all new components after recording.
Automatically create snapshot for newly recorded components when recording a BPT test	Instructs UFT to take a snapshot of the application window when recording a BPT test.

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

Start the test record

1. 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.
2. In the New Business Component dialog box, provide the **Name**, **Component**, and default **Application Area** for your component.
3. Click **Create**. UFT is minimized and the Record toolbar is displayed.

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.

Add additional components to the test (optional)

1. In the Record toolbar, click the **New Business Component** button .
2. In the New Business Component dialog box, provide a name for the component. (The component is saved in the same location you entered when beginning the recording session.)
Subsequent steps are recorded in the created component.

Stop recording

1. In the Record Toolbar, press the **Stop** button . UFT loads and creates the components, displaying them in the test canvas or grid.
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.
2. If necessary, when prompted, in the Check In dialog box, check in the new components to your version controlled ALM project.

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, "Create test steps in a business process test" on page 863.

Prerequisites

Add a business component to the solution and open it.

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.

Capture the test objects in the application

1. In the Solution Explorer pane, select the component in which you want to include the captured objects.
2. In the toolbar, click the **Capture** button . The Capture toolbar opens.
3. (Optional) In the Capture toolbar, press the **Define Object Filter** button .
4. (Optional) In the Define Object Filter dialog box, select the level of detail to capture:

Selected object only	Captures the selected object's properties and values, without its descendant objects.
Default object types	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 Selected object types , clicking the Select button, and then clicking the Default button.
All object types	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.
Selected object types	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 Select button and selecting the required items.

5. Select the top level object for which you want to capture its test objects (i.e. a Browser window, or an application screen).
6. 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.

7. 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.

Capture a selected area of the application

1. In the Solution Explorer pane, select the component in which you want to include the captured objects.
2. In the toolbar, click the **Capture** button . The Capture toolbar opens.
3. (Optional) In the Capture toolbar, press the **Define Object Filter** button .
4. (Optional) In the Define Object Filter dialog box, select the level of detail to capture:

Selected object only	Captures the selected object's properties and values, without its descendant objects.
Default object types	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 Selected object types , clicking the Select button, and then clicking the Default button.
All object types	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.
Selected object types	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 Select button and selecting the required items.

5. Select the top level object for which you want to capture its test objects (i.e. a Browser window, or an application screen).
6. In the Capture toolbar, press the **Capture Area** button. The mouse pointer turns into a crosshairs.
7. 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.
8. 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.

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.

1. In the Solution Explorer, expand the node of your component.
2. Under the component node, double-click the **Local** node. The Object Repository window opens.
3. In the Object Repository, edit the names and properties of the objects as needed.

Export the local object repository

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.

1. In the Object Repository window, select **File > Export Local Objects**.
2. 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.)
3. 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 67: Using Data in Business Process Testing

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 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).

Parameterization is performed at a number of different levels in a business process test:

Test parameters	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).
Flow parameters	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).
Component parameters	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: 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
Input parameters for a component	<ul style="list-style-type: none">• LoginName, entered as input by the banker when logging in• AccountNo, entered by the banker, perhaps from a written inquiry. <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>
Output parameters for a component	<ul style="list-style-type: none">• SessionNo, a number for the login session, outputted by the business component when the banker successfully logs in• SelectedAccountNo, outputted by the business component after the banker selects a loan from a list
Test Parameters	CustomerLoans , a comma-delimited list of all loans for a particular customer, accessed from the test level.

For task details, see ["Use data in a business process test" on page 877](#).

Linking parameters

Relevant for: business process testing

Parameter linkage enables you to pass data from test or flow parameters to component parameters, or between components. This tests the ability of your application to pass a value from one API to another in the course of the application's work.

**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.

Linking parameters when the component has iterations

As part of data driving a business process test, you can set components (or groups of components) to run multiple times in different iterations.

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).
- **Condition 2.** The source and the target have the same number of iterations (an “n-to-n” relationship).



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.

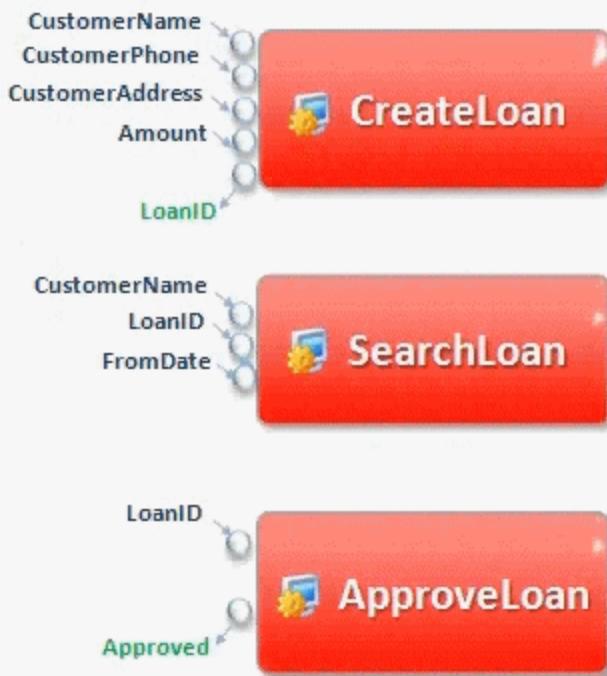
When you use the output parameter of a previous component as the value for an input parameter of a subsequent component, the link between the output and input parameter applies to all component iterations of the input parameter. Likewise, when iterations of a source component in a business process test 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.



Example: 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**).

**Linking Input and Output Parameters**

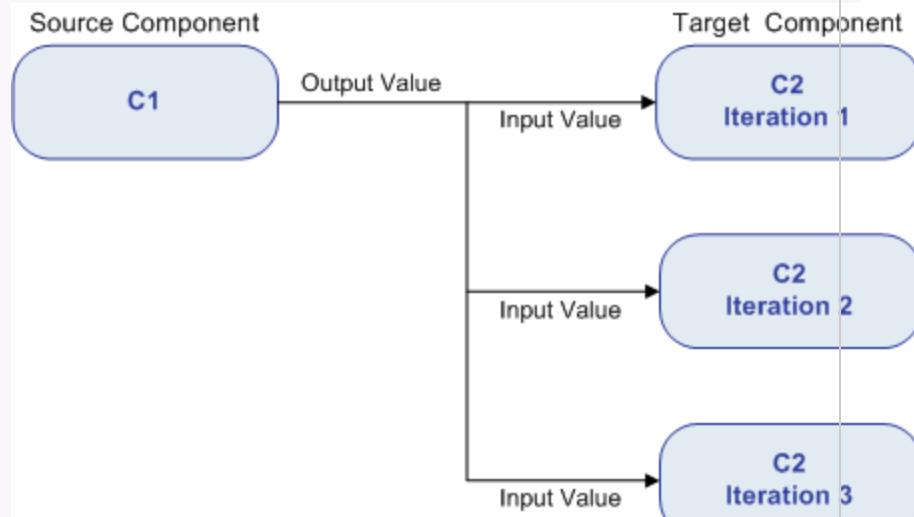
When you create a business process test, you arrange the components in order to test the entire loan approval process workflow from receiving the request through approving the request.

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, 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. The value of the **LoanID** output parameter is used for each iteration of the **SearchLoan** or **ApproveLoan** component:

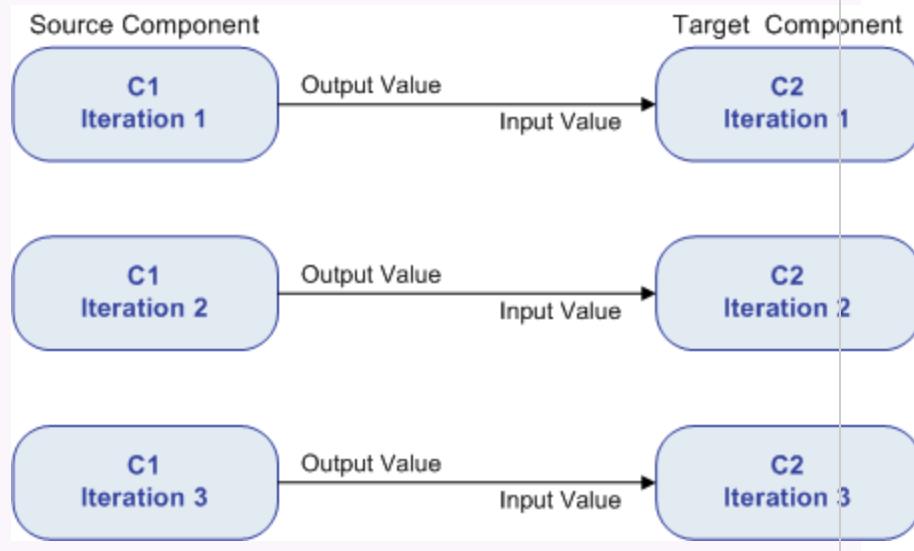




Linking Parameters when using Iterations ("n-to-n" Relationship)

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.

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:



For task details, see ["Link parameters" on page 878](#).

Promoting parameters

Relevant for: business process testing

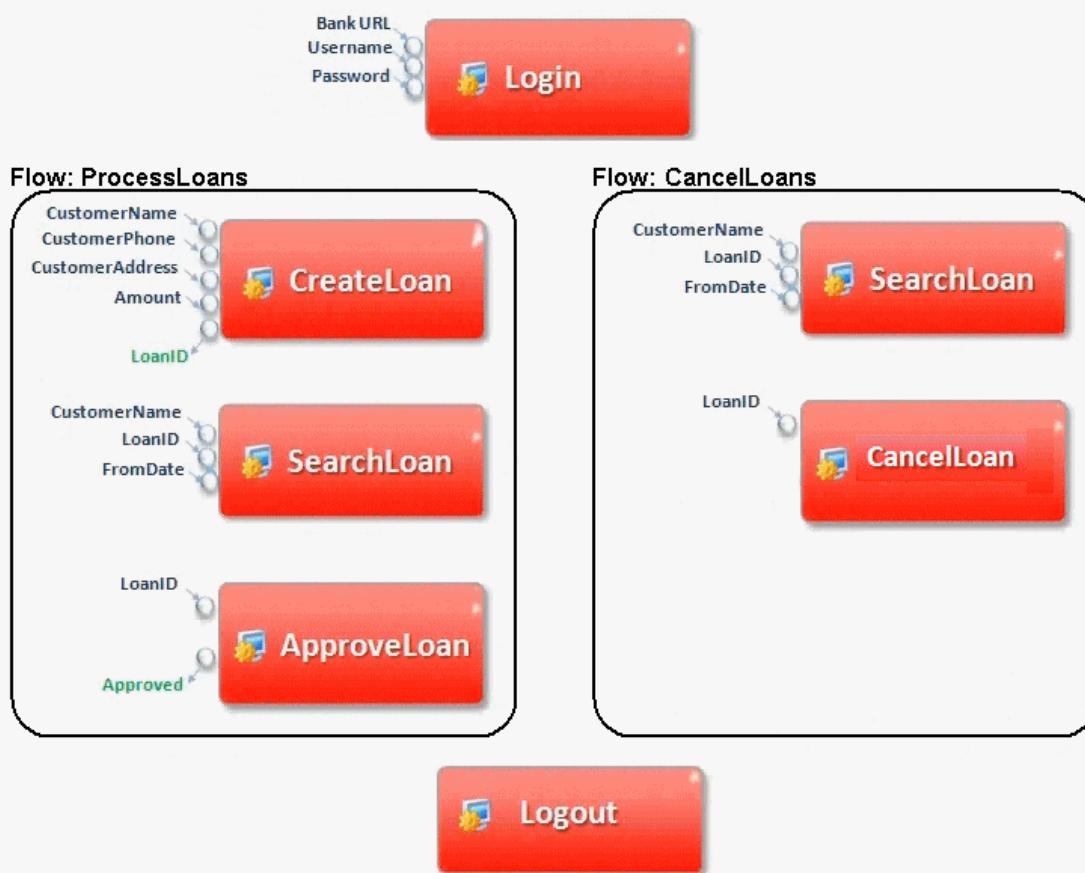
When editing a business process test, you can promote flow or component parameters to the flow or test level at the same time as you add a component to a flow or test.



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 879.

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.

Note: This task is part of a higher-level task. For details, see "[Create and maintain business process tests and flows in UFT](#)" on page 839.

Design data

Consider the following before using parameters:

- Determine which parameters are linked.
- Determine which parameters should be available at the component, flow, and test levels.
- Determine how many times each component, flow, and business process test should run, and with what parameter values.

Create parameters and set default values

1. In the Solution Explorer, select the test, flow, or component to which you want to add a parameter.
2. In the Properties pane, open the **Test Parameters or Parameters** tab .
3. In the Test Parameters/Parameters tab, click the **Add** button and specify the type of parameter to add (either input or output).
4. 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 ####-## ####).

Use component parameters in component steps

For details on using the parameters as step values, see "[Parameterize values for operations](#)" on page 339.

Link parameters

1. In the document pane, select a business process test or flow.
2. In the business process test or flow tab, select a flow or component.
3. In the Properties pane, open the **Test Parameters or Parameters** tab.
4. Select the parameter you want to link.

5. In the Value cell of the parameter, click the **Link** button. The Select Link Source dialog box opens.
6. In the left pane of the Select Link Source dialog box, select the test or flow from which you want to use a parameter.
7. 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:

In the Options dialog box	<ol style="list-style-type: none">1. Open the BPT Testing tab (Tools > Options > BPT Testing tab).2. In the BPT Testing tab, select the Promote component parameters option. Any parameters in components or flows added after this option is selected are promoted to test or flow parameters. If you do not want to automatically promote parameters for a selected component or flow, clear this option before adding the flow or component.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 Promote Parameters button  in the Parameters tab of the Properties pane. The selected parameter is promoted to the next level and the parameter value is linked to the value of the newly created parameter.
In the Properties pane	<p>In the Parameters tab, click the Promote Parameters 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>
In the Toolbox pane	<p>In the title node, click the down arrow and then click the Promote parameters to test level 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:

Add New Iteration:	Adds a new iteration without any values for the component/group parameters.
Copy Iteration.	Copies the values for the component/group parameters from the previously entered iteration.
Create Iteration with Default Values	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. 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.

3. 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.

Using data with Business Process Testing in UFT - Known Issues

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 68: Running Business Process Tests

Relevant for: business process testing

Running business process tests is much the same as running regular UFT GUI tests. UFT opens the specified application, and opens each component in the test in sequence. Inside each component, UFT runs the steps contained in the component.

However, when running a business process test, there are specific issues to consider:

- **The number of iterations:** You can specify different numbers of iterations for any flow, component, or component group included in your test. Furthermore, the number of iterations does not have to be the same for all members of the test.
- **The run conditions:** You can instruct UFT how to run a test, depending on the condition of the component and your application before or after a specific component's steps are complete.
- **The data to use for a test run:** You can change the data used for a specific business process test run by creating test configurations and selecting a particular test configuration for each test run.

Running iterations

Relevant for: business process testing

When running a business process test, you can vary the number of times a selected flow, component group, or component runs. This enables you to run flows or components with different sets of data.

This differing values are defined in the Data pane. In the Data pane, specify the number of iterations for each flow or component. Then, for each iteration, specify values for each of the flow or component parameters.

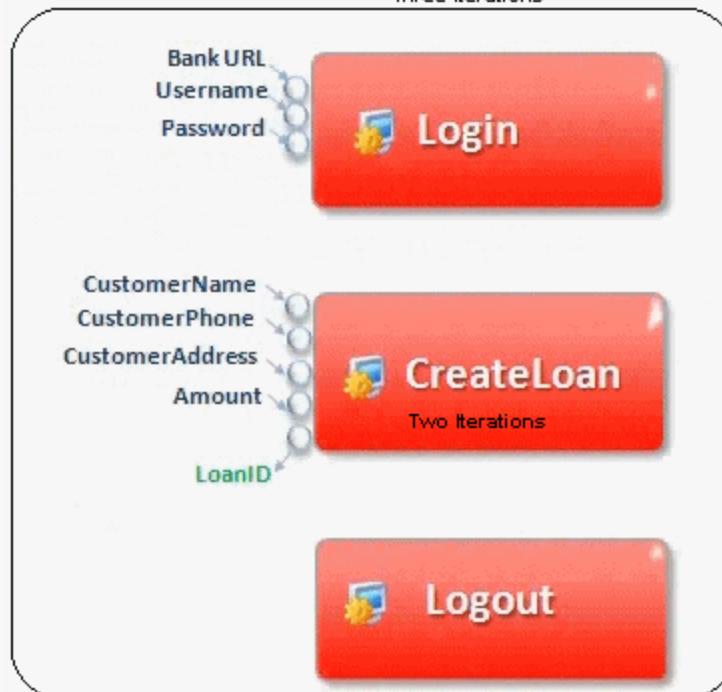
In addition, you can export the component parameter iteration structure to an external Excel and then edit this Excel. After you provide values for the parameters in each iteration inside the Excel file, you import the Excel back into the test and UFT automatically uses the parameter values provided in the Excel file.



Example: You create a business process test with components named **Login**, **CreateLoan**, and **Logout**.

Business Process Test: CreateLoans

Three Iterations



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 [880](#).

Running iterations of component groups

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.

When you group components, all the components in the group must include the same number of iterations.

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.



Example:

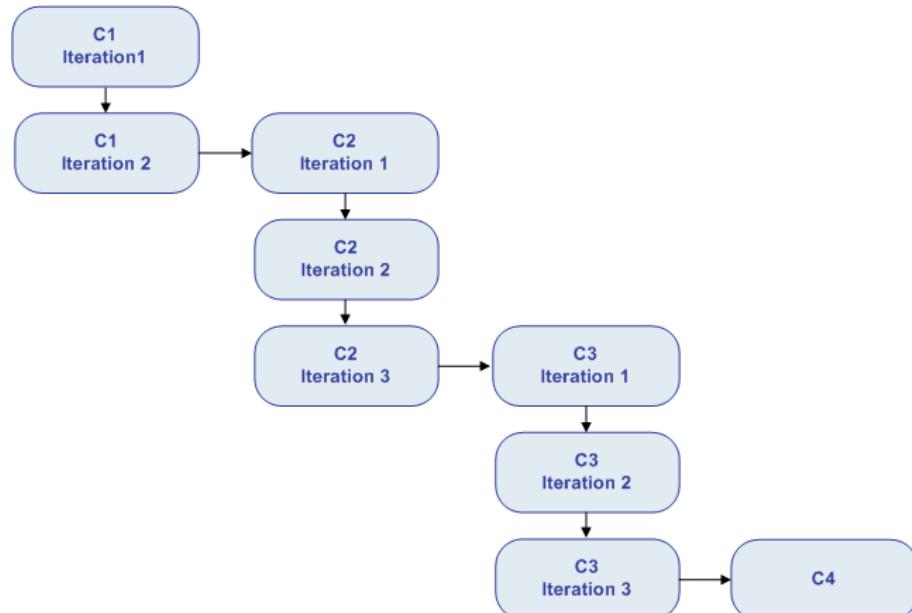
You create a business process test with the following business components: **C1**, **C2**, **C3**, and **C4**. You set the iterations for the components as follows:

- 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:



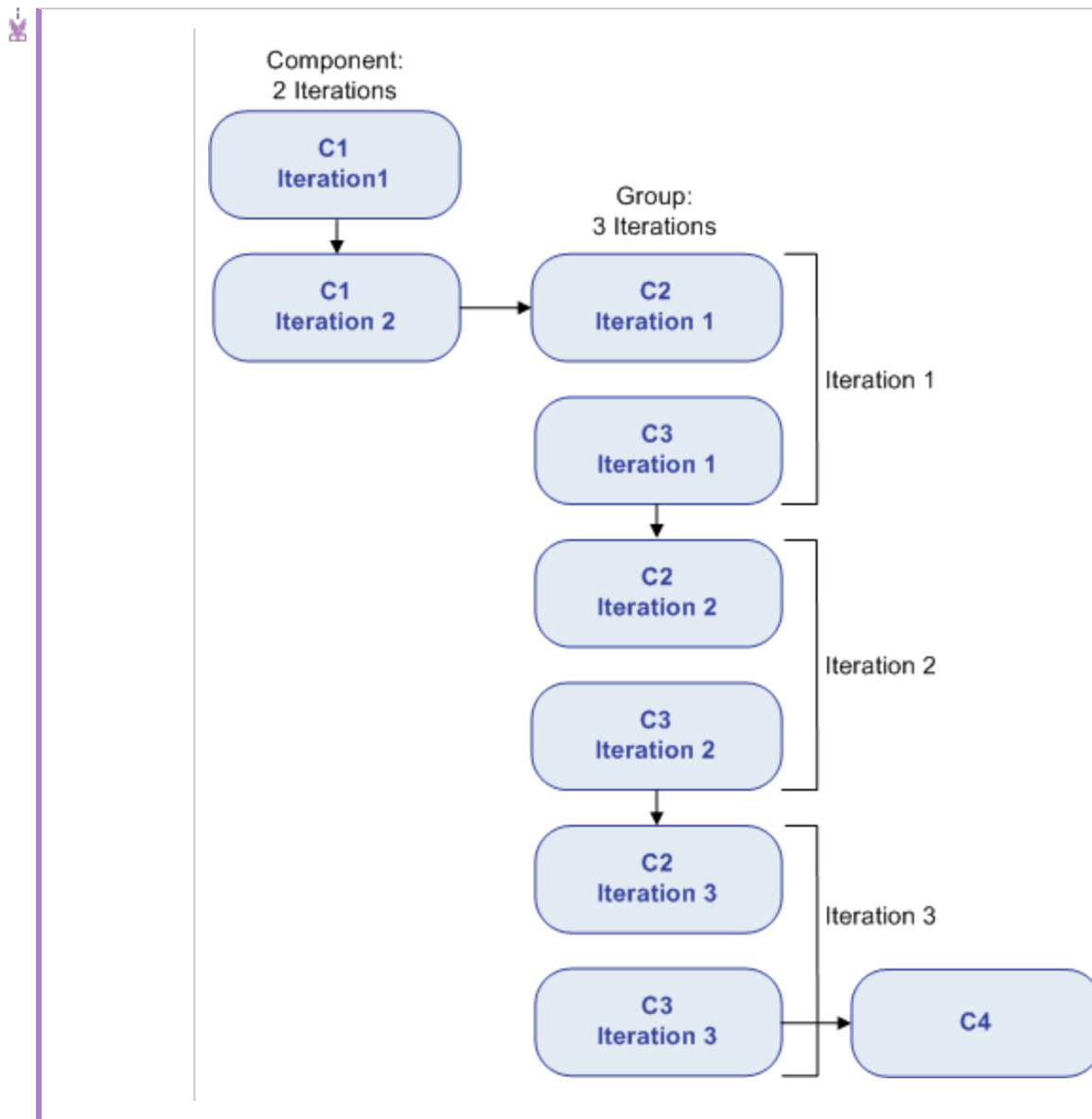
**With
Groupin
g**

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:



Run conditions

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.

Set run conditions

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 "[Create and maintain business process tests and flows in UFT](#)" on page 839.

Prerequisites

Ensure that the component for which you are setting run conditions has parameters or that the flow containing the component has a parameter.

Select the flow or component to set run conditions

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.

Set On Failure settings

If you are editing a test, you can set **On Failure** options for all the flows or components included in the test:

1. In the Solution Explorer or document pane, select the business process test you are editing.
2. In the document pane, select the flow or component for which to set On Failure settings.
3. In the Properties pane, open the **Properties** tab .
4. 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.

Set run conditions

When you are editing a business process flow or component, you can set additional run options for the components included in the flow:

1. In the Solution Explorer or the document pane, select the business process test or flow on which you are working.
2. In the document pane, select the component to which you want to add run conditions.
3. In the **Properties** tab  in the Properties pane, select the **Use run condition** option.
4. 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:

Skip to next component and continue	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.
End component run and fail.	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.

Go To	<p>UFT continues to the specified place in the test. You select the location in the Go To: 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>Note: The GOTO condition is supported only for ALM versions 12.50 patch 1 or higher.</p>
--------------	---

Chapter 69: Business Process Testing with the BPT Packaged Apps Kit

Relevant for: business process tests and flows

Normally, when you create business process tests of your application, you 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.

To make creating business process tests of your SAP applications simpler, you use the BPT Packaged Apps Kit. When you use the BPT Packaged Apps Kit, you can:

- **Automatically learn the steps you perform on the SAP application,** and then generate a test with separate components based on the screens and transactions within your applications.
- **Run tests in Change Detection Mode** 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 "[Learn business process tests and flows](#)" on page 895 and "[Detect and resolve changes using Change Detection Mode](#)" on page 901.

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, UFT can "learn" components automatically from the application.

To learn, you simply perform user actions in your application. After you finish, UFT automatically breaks down the learned areas of the application into individual business components representing 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.

While 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.

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 and displays the changes in a Change Detection report.

Note: Running tests in Change Detection mode is supported only for SAP GUI applications.

For each change in the report, UFT offers possible resolutions. This enables you to update your test or flow automatically without learning or manually updating the component's steps.

For example, you are testing a screen for adding new customer contact information. 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.

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 page 897
6. "Select components" on page 897
7. "Reuse an existing component" on page 897
8. "Remove a learned component" on page 898
9. "Resume learning components" on page 898
10. "Edit table parameters" on page 899

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.
- 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**.

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:

Manually select components for reuse:	This option enables you to select the components in the Learn Summary dialog box after finishing the learning session.
Automatically reuse identical components:	This automatically uses existing components that are identical without prompting you after learning the application.
Ignore comments when comparing scripts	This instructs UFT to ignore comment steps when comparing learned components to existing ones.
Component Type:	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:

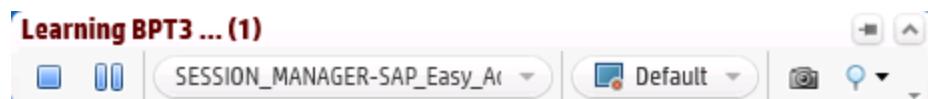
- **Automatically parameterize steps using Business Component parameters:** Enables you to use the Business component parameters as the step value parameters
- **Use the default values from the learned flow:** Uses the values you entered when learning the flow
- **Use the values from the reused components:** 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**

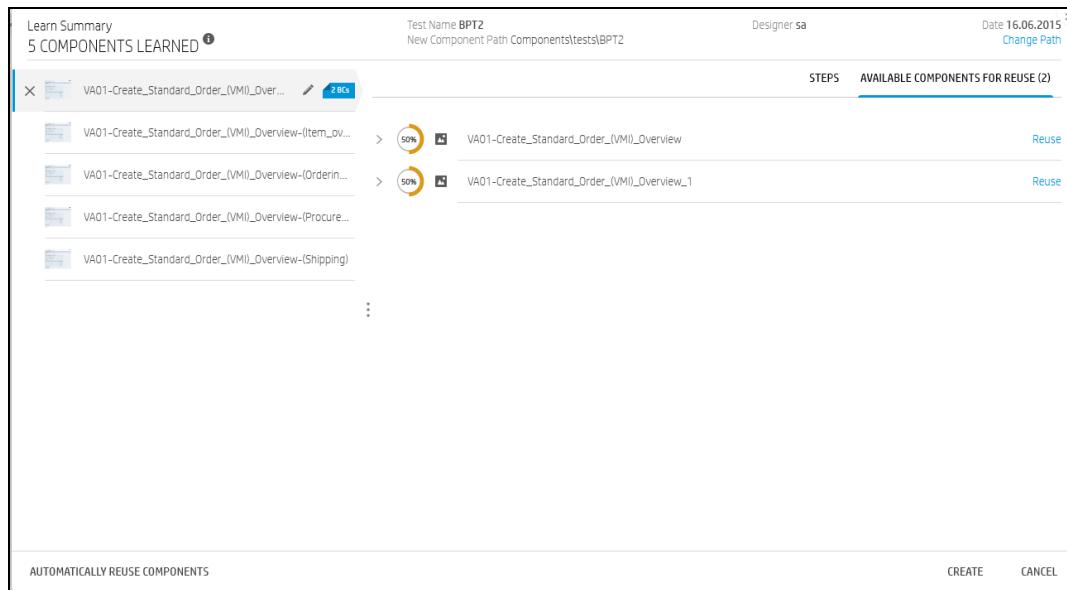
After you finish learning the screens/areas in your application, UFT displays a summary of the learning session in the Learn Summary dialog box.

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**

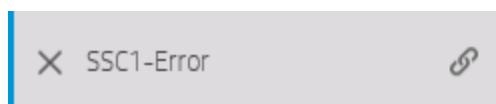
If UFT detects that a learned component is similar or identical to an existing component, UFT displays the number of similar components next to the component name in the component tree.

- 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.



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:

- 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:



- Repeat this process for each component that you wish to reuse.
 - 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.
- Remove a learned component**
 - In the component list, select the component to remove.
 - To the left of the component, click the **X** button.
 - 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.
 - 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:

Yes	If you select Yes, the existing components in the test are removed and newly learned components are inserted instead.
No	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 896](#).

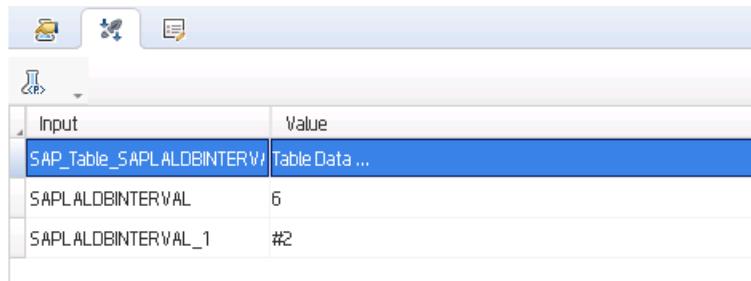
10. **Edit table parameters**

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:

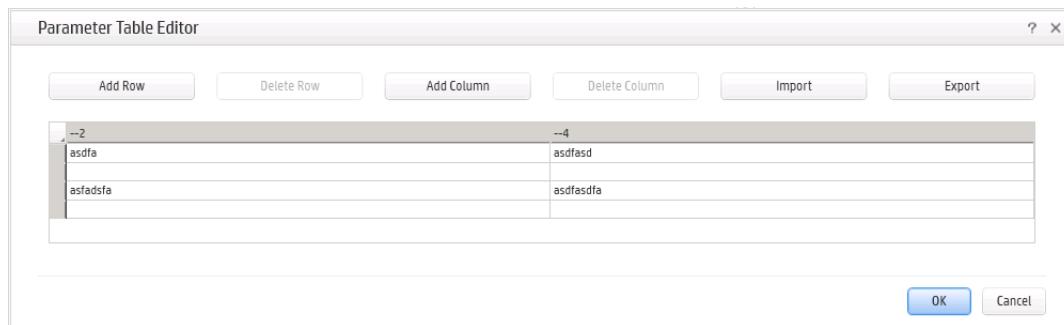
- a. Create or open a GUI test.
- b. In the SAP General pane of the Options dialog box (**Tools > Options > GUI Testing tab > SAP > General node**), select the **Auto-parameterize table and grid controls** option.
- c. Close the GUI test.
- d. Create or open a business process test.
- e. Perform user 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_SAPLAODBINTERVAL	Table Data ...
SAPLAODBINTERVAL	6
SAPLAODBINTERVAL_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.

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" below
4. "Save changed components to your ALM project" on page 903
5. "Optional - view run results for the test run" on page 903

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.
- You must belong to a user group that has permissions for the **Run** task, and permissions to modify tests and business components.

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 test of an SAP Fiori application in Change Detection Mode.

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

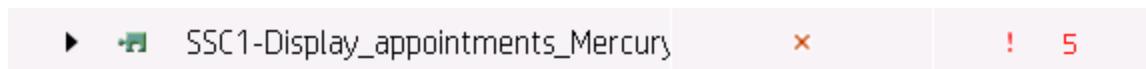
Using the report, you can update your components and steps automatically:

- In the component tree, select the component for which you want to resolve changes.

Components in which you need to resolve changes are displayed 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.

- 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"> + <input type="checkbox"/> Propose items (Ctrl+F11) + <input type="checkbox"/> Reject document (Ctrl+F10) + <input type="checkbox"/> Save (Ctrl+S) + <input type="checkbox"/> Net value + <input type="checkbox"/> Net value_2 + <input type="checkbox"/> PO date + <input type="checkbox"/> Purch.order no. + <input type="checkbox"/> Ship-to party + <input type="checkbox"/> Sold-to party + <input type="checkbox"/> Standard Order (VMI) + <input type="checkbox"/> mbar + <input type="checkbox"/> OKCode - <input type="checkbox"/> / 	<ul style="list-style-type: none"> <input type="checkbox"/> Add step to component: 'Propose items (Ctrl+F11).Click' <input type="checkbox"/> Add step to component: 'Reject document (Ctrl+F10).Click' <input type="checkbox"/> Add step to component: 'Save (Ctrl+S).Click' <input checked="" type="checkbox"/> Add step to component: 'Net value.Set' <input checked="" type="checkbox"/> Add step to component: 'Net value_2.Set' <input type="checkbox"/> Add step to component: 'PO date.Set' <input type="checkbox"/> Add step to component: 'Purch.order no..Set' <input type="checkbox"/> Add step to component: 'Ship-to party.Set' <input checked="" type="checkbox"/> Add step to component: 'Sold-to party.Set' <input type="checkbox"/> Add step to component: 'Standard Order (VMI).Set' <input type="checkbox"/> Add step to component: 'mbar.Select' <input type="checkbox"/> Add step to component: 'OKCode.Set' <input type="checkbox"/> This object is not used in any step 		
<input checked="" type="checkbox"/> Changes to object repository were applied			<input type="button" value="UPDATE STEPS"/>

- If you want to accept the proposed changes, in the lower right corner of the pane, click the **Apply Changes** button.

In addition, the selected component's report row is updated to show that you have resolved the changes:



- In the right pane, select the checkboxes for the steps that require an update.
- 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 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 - Known Issues

- 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:
GUI tests	<ul style="list-style-type: none">• GUI tests• actions• function libraries
GUI components	<ul style="list-style-type: none">• keyword GUI components• scripted GUI components• application areas• function libraries
API testing	<ul style="list-style-type: none">• API tests• API components• user code files

Term (A-Z)	Relevant for:	For details, see:
.NET assembly	API testing	
accessibility checkpoints	<ul style="list-style-type: none">• GUI tests• scripted GUI components	"Accessibility checkpoints" on page 267
actions	<ul style="list-style-type: none">• GUI tests• API testing	<ul style="list-style-type: none">• For GUI testing: "Actions in GUI Testing" on page 121• For API testing: "Actions for API Tests" on page 441
active screen	<ul style="list-style-type: none">• GUI tests• scripted GUI components	"Active Screen Pane" on page 77
activity	API testing	"Standard Activities" on page 386
activity repository	API testing	"Perform activity sharing" on page 414

Term (A-Z)	Relevant for:	For details, see:
add-ins	<ul style="list-style-type: none"> • GUI tests • GUI components 	<i>HP Unified Functional Testing Add-ins Guide</i>
ALM project	<ul style="list-style-type: none"> • GUI tests • GUI components • API testing 	"ALM Integration" on page 769
application area	GUI components	"Business Components and Application Areas" on page 854
array properties	API testing	
automation	<ul style="list-style-type: none"> • GUI tests • GUI components 	"UFT Automation Scripts" on page 605
bitmap checkpoints	<ul style="list-style-type: none"> • GUI tests • scripted GUI components 	"Bitmap checkpoints" on page 267
bookmarks	<ul style="list-style-type: none"> • GUI tests • scripted GUI components • function libraries • API testing 	"Bookmarks Pane" on page 80
breakpoints	<ul style="list-style-type: none"> • GUI tests • scripted GUI components • function libraries • API testing 	
business process test	<ul style="list-style-type: none"> • GUI components • API testing 	

Term (A-Z)	Relevant for:	For details, see:
canvas	<ul style="list-style-type: none"> • GUI tests • API testing 	"The Canvas" on page 81
checkpoint	<ul style="list-style-type: none"> • GUI tests • GUI components • API testing 	<ul style="list-style-type: none"> • For GUI testing: "Checkpoints in GUI Testing" on page 262 • For API testing: "Checkpoint validation" on page 386
checkpoint validation	API testing	"Checkpoint validation" on page 386
classes	API testing	"Search for references or classes" on page 540
code completion	<ul style="list-style-type: none"> • GUI tests • scripted GUI components • function libraries • API testing 	"Automatic code completion" on page 535
code object	<ul style="list-style-type: none"> • GUI tests • GUI components • API testing 	
code snippets	<ul style="list-style-type: none"> • GUI tests • scripted GUI components 	<ul style="list-style-type: none"> • "Automatic code completion" on page 535 • "Use code snippets and templates" on page 538
code templates	<ul style="list-style-type: none"> • GUI tests • scripted GUI components • function libraries • API testing 	<ul style="list-style-type: none"> • "Use code snippets and templates" on page 538
compile	API testing	"Output Pane" on page 88

Term (A-Z)	Relevant for:	For details, see:
conditional statement	<ul style="list-style-type: none"> • GUI tests • scripted GUI components 	"Comments, control-flow, and other VBScript statements" on page 554
conditional steps	API testing'	
data driving	<ul style="list-style-type: none"> • GUI tests • API testing 	<ul style="list-style-type: none"> • For GUI testing: "Data Driver" on page 338 • For API testing: "Assign data to API test/component steps" on page 454
data table	<ul style="list-style-type: none"> • GUI tests • scripted GUI components • API testing 	<ul style="list-style-type: none"> • For GUI testing: "Data Pane" on page 82 • For API testing: "Data Usage in API Tests" on page 445
data table parameters	GUI tests	"Data table parameters" on page 331
data table property	API testing	"Assign data to API test/component steps" on page 454
database checkpoints	<ul style="list-style-type: none"> • GUI tests • scripted GUI components 	"Database checkpoints" on page 271
database query	API testing	
Editor	<ul style="list-style-type: none"> • GUI tests • scripted GUI components • function libraries • API testing 	"The Editor" on page 533
environment variables	<ul style="list-style-type: none"> • GUI tests • GUI components 	"Storing output values" on page 324
event handler	API testing	"Writing code for API test events" on page 618

Term (A-Z)	Relevant for:	For details, see:
file content checkpoints	<ul style="list-style-type: none"> • GUI tests • scripted GUI components 	"File Content checkpoints" on page 272
function library	<ul style="list-style-type: none"> • GUI tests • GUI components 	"User-Defined Functions" on page 575
global data sheet	<ul style="list-style-type: none"> • GUI tests • scripted GUI components 	"Data tables and sheets in GUI tests and components" on page 348
IBM Websphere MQ	API testing	
input/output properties	API testing	"Properties Pane" on page 90
Insight objects	<ul style="list-style-type: none"> • GUI tests • GUI components 	"Identifying objects using Insight" on page 236
JMS transport	API testing	
JSON	API testing	"Send and receive a JSON request for a REST service" on page 430
Keyword View	<ul style="list-style-type: none"> • GUI tests • GUI components 	"Keyword View" on page 126
load testing	API testing	
log tracking	<ul style="list-style-type: none"> • GUI tests • GUI components 	"Log tracking" on page 710
loop statement	<ul style="list-style-type: none"> • GUI tests • scripted GUI components 	"Comments, control-flow, and other VBScript statements " on page 554

Term (A-Z)	Relevant for:	For details, see:
maintenance run mode	<ul style="list-style-type: none"> • GUI tests • GUI components 	"Maintenance Run mode" on page 148
missing resources	<ul style="list-style-type: none"> • GUI tests • GUI components • API testing 	"Errors Pane" on page 86
multipart HTTP requests	API testing	"Send a multipart HTTP or REST Service request" on page 397
negative testing	API testing	"Negative testing of Web services" on page 415
object repository	<ul style="list-style-type: none"> • GUI tests • GUI components 	
object spy	<ul style="list-style-type: none"> • GUI tests • GUI components 	
optional steps	<ul style="list-style-type: none"> • GUI tests • scripted GUI components 	"Optional steps" on page 709
output value	<ul style="list-style-type: none"> • GUI tests • scripted GUI components 	"Output Values in GUI Testing" on page 322
parameter (both input and output)	<ul style="list-style-type: none"> • GUI tests • GUI components 	"Parameterizing Object Values" on page 330
property	<ul style="list-style-type: none"> • GUI tests • GUI components • API testing 	"Properties Pane" on page 90

Term (A-Z)	Relevant for:	For details, see:
recording	<ul style="list-style-type: none"> • GUI tests • GUI components 	"Record GUI Tests and Components" on page 113
recovery scenario	<ul style="list-style-type: none"> • GUI tests • GUI components 	"Recovery Scenarios" on page 159
references	API testing	
regular expression	<ul style="list-style-type: none"> • GUI tests • scripted GUI components 	"Regular expressions" on page 357
repository parameters	<ul style="list-style-type: none"> • GUI tests • GUI components 	"Test Objects in Object Repositories" on page 206
resources	<ul style="list-style-type: none"> • GUI tests • GUI components 	
REST Service	API testing	"Create a REST service model" on page 424
run results	<ul style="list-style-type: none"> • GUI tests • GUI components • API testing 	"Using Run Results" on page 716
run session	<ul style="list-style-type: none"> • GUI tests • GUI components • API testing 	"aRunning Tests and Components" on page 695
run-time object	<ul style="list-style-type: none"> • GUI tests • GUI components 	"How UFT applies the test object model concept" on page 174
SOAP	API testing	"Web Service scenario" on page 482

Term (A-Z)	Relevant for:	For details, see:
solution	<ul style="list-style-type: none"> • GUI tests • GUI components • API testing 	"Solution Explorer Pane" on page 93
statement completion	<ul style="list-style-type: none"> • GUI tests • scripted GUI components • function libraries • API testing 	"Statement completion" on page 533
step generator	<ul style="list-style-type: none"> • GUI tests • scripted GUI components 	
syntax errors	<ul style="list-style-type: none"> • GUI tests • GUI components • API testing 	"Errors Pane" on page 86
system monitor	<ul style="list-style-type: none"> • GUI tests • GUI components 	
Test Batch Runner	<ul style="list-style-type: none"> • GUI tests • API testing 	"Test Batch Runner" on page 707
test loops	API testing	
test object	<ul style="list-style-type: none"> • GUI tests • GUI components 	"The Test Object Model" on page 173
test variables	API testing	"Define API test properties or user/system variables" on page 461
text checkpoint text area checkpoints	<ul style="list-style-type: none"> • GUI tests • scripted GUI components 	"Text and text area checkpoints" on page 274

Term (A-Z)	Relevant for:	For details, see:
text recognition	<ul style="list-style-type: none"> • GUI tests • GUI components 	<ul style="list-style-type: none"> • "Text recognition in run-time" on page 275
TODO comments	<ul style="list-style-type: none"> • GUI tests • scripted GUI components • function libraries • API testing 	"Tasks Pane" on page 94
Toolbox	<ul style="list-style-type: none"> • GUI actions • GUI components function libraries • API testing Business process tests and flows 	"Toolbox Pane" on page 95
Update Run Mode	<ul style="list-style-type: none"> • GUI tests • GUI components 	"Update Run mode " on page 153
Update Service/Assembly	API testing	"Updating Services and Assemblies" on page 473
user code files	API testing	"Writing code for API test events" on page 618
Userlogger	API testing	"UserLogger Object" on page 680
virtual object	<ul style="list-style-type: none"> • GUI tests • scripted GUI components 	"Virtual Objects" on page 260
virtualized services	API testing	"Virtualized services" on page 738
Web services	API testing	

Term (A-Z)	Relevant for:	For details, see:
WSDL	API testing	"Import a WSDL-based Web service" on page 421
XML checkpoints	<ul style="list-style-type: none">• GUI tests• GUI components	"XML checkpoints" on page 279
XPath Checkpoints	API testing	"XPath checkpoints" on page 387

Appendix B: 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 | 917 |
| • Supported Output Values | 921 |

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 page 920](#).

	Accessibility	Bitmap	Database	File Content	Image	Page	Standard	Table	Text	Area	XML (Application)	XML (Resource)
.NET Web Forms ³	S	S	NA	NA	NA	NA	S	S	S	S	S	S
.NET Windows Forms	NA	S	NA	NA	NA	NA	S	S	S	S	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

User Guide

Appendix B: GUI Checkpoints and Output Values Per Add-in

	Accessi bility	Bit ma p	Data base	File Con tent	Im ag e	P age	Stan dard	Ta bl e	T ext	A rea	XML (Applic ation)	XML (Reso urce)
Java	NA	S	NA	NA	NA	NA	S	S	S	S ⁴	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
People Soft	S	S	NA	NA	S	S	S	S	S ¹	NS	S	S
PowerB uilder²	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 Windo ws- based	S ⁵	S	NA	NA	S ⁵	S ₅	S	S	S ₅	NS	S ⁵	NA
Siebel	S	S	NA	NA	S	S	S	S	S	NS	S ⁶	S
SiebelO penUI	S	S	NA	NA	S	S	S	S	S	S	S ⁶	NA
Silverli ght	NA	S	NA	NA	NA	NA	S	S	S	S	NA	NA

User Guide

Appendix B: GUI Checkpoints and Output Values Per Add-in

	Accessi bility	Bit ma p	Data base	File Con tent	Im ag e	P age	Stan dard	Ta bl e	T ext	A re a	XML (Appli cation)	XML (Reso urce)
Standar d Windo ws	NS	S	NA	NA	NS	NA	S	S	S	S	NA	NA
Stingra y	NA	S	NA	NA	NA	NA	S	S	S	S	NA	NA
Termin al Emulat or	NA	S	NA	NA	NA	NA	S	NA	S ⁷	NA	NA	NA
Testing Extensi bility	NA	S	NA	NA	S	NA	S	S	S ₁	S	NA	NA
UI Automa tion	NS	S	NS	NS	NS	NS	S	NS	S	S	NS	NS
VisualA ge for Smallta lk	NA	S	NA	NA	NA	NA	S	S	S	S	NA	NA
Visual Basic	NS	S	NA	NA	NS	NA	S	S	S	S	NA	NA
Web	S	S	NA	NA	S	S	S	S	S ₁	S	S ⁶	NA
Windo ws Runtim e	NA	S	NA	NA	NA	NA	S	S	S	S	NA	NA
WPF	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 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.
- 7 Text and text area checkpoints are supported for Terminal Emulators for the TEScreen and TEText Screen objects.

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 page 923](#).

	Accessi bility	Bit ma p	Data base	File Con tent	Im ag e	P a ge	Stan dard	Ta bl e	T e xt	A re a	XML (Appli cation)	XML (Reso urce)
.NET Web Forms	NA	NA	NA	NA	NA	S	S	S	S5	S5	NA	NA
.NET Windo ws Forms	NA	NA	NA	NA	NA	NA	S	S	S5	S5	NA	NA
ActiveX	NS	NA	NA	NA	NA	NA	S	S	S	S	NA	NA
Delphi	NS	NA	NA	NA	NA	NA	S	S	S	S	NA	NA
Java	NA	NA	NA	NA	NA	NA	S	NA	S	S3	NA	NA

User Guide

Appendix B: GUI Checkpoints and Output Values Per Add-in

	Accessi bility	Bit ma p	Data base	File Con tent	Im ag e	P age	Stan dard	Ta bl e	T ext	A re a	XML (Applic ation)	XML (Reso urce)
Mobile	NA	NA	NA	NA	NA	NA	S	NA	S	NS	NA	NA
Oracle	NA	NA	NA	NA	NA	NA	S	S	NA	NA	NA	NA
People Soft	NA	NA	NA	NA	NA	S	S	S	S ¹	NS	S	S
PowerB uilder²	NA	NA	NA	NA	NA	NA	S	NA	S	S	NA	NA
Qt	NA	NA	NA	NA	NA	NA	S	S	S	S	NA	NA
SAP Web- based	NA	NA	NA	NA	NA	S	S	S	S	NS	S	S
SAP Windo ws- based	NA	NA	NA	NA	NA	S ⁴	S	S	S ⁴	NS	S ⁴	S
Siebel	NA	NA	NA	NA	NA	S	S	S	S	NS	S	S
SiebelO penUI	NA	NA	NA	NA	NA	S	S	S	S ¹	S	S ⁶	NA
Silverli ght	NA	NA	NA	NA	NA	N A	S	S	S	S	NA	NA
Standar d Windo ws	NA	NA	NA	NA	NA	N A	S	S	S	S	NA	NA

User Guide

Appendix B: GUI Checkpoints and Output Values Per Add-in

	Accessi bility	Bit ma p	Data base	File Con tent	Im ag e	P age	Stan dard	Ta bl e	T ext	A re a	XML (Appli cation)	XML (Reso urce)
Stringa y	NA	NA	NA	NA	NA	NA	S	S	S	S	NA	NA
Termin al Emulat or	NA	NA	NA	NA	NA	NA	S ⁸	NA	S ⁷	NA	NA	NA
Testing Extensi bility	NA	NA	NA	NA	NA	NA	S	S	S ¹	S	NA	NA
UI Automa tion	NS	S	NS	NS	NS	NS	S	NS	S	S	NS	NS
VisualA ge for Smallta lk	NA	NA	NA	NA	NA	NA	S	S	S	S	NA	NA
Visual Basic	NA	NA	NA	NA	NA	NA	S	NA	S	S	NA	NA
Web	NA	NA	NA	NA	NA	S	S	S	S ¹	NS	S ⁶	NA
Windo ws Runtim e	NA	NA	NA	NA	NA	NA	S	S	S	S	NA	NA
WPF	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 C: 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:

• Programming in the Editor and function libraries	926
• Can I store functions and subroutines in a function library?	926
• How can I enter information during a run session?	926
• I have a Microsoft Access database that contains data I would like to use in my test.	
How do I do this?	926
• How do I customize the run results?	927
• Working with dynamic content	927
• How can I create and run tests or components on objects that change dynamically from viewing to viewing?	927
• How can I check that an object or child object exists (or does not exist)?	927
• How does UFT record on dynamically generated URLs and Web pages?	928
• How does UFT handle tabs in browsers?	928
• Advanced Web issues	929
• How does UFT handle cookies?	929
• Where can I find a Web page's cookie?	929
• How does UFT handle session IDs?	929
• How does UFT handle server redirections?	929
• How does UFT handle meta tags?	930
• Does UFT work with .asp and .jsp?	930
• How does UFT support AJAX?	930
• Does UFT work with COM?	930
• Does UFT work with XML?	930
• How can I access HTML tags directly?	931
• How can I send keyboard key commands (such as shortcut commands) to objects that do not support the Type method?	931
• Working with Windows applications	931
• How can I record on nonstandard menus?	931
• Can I copy and paste to and from the Clipboard during a run session?	932
• Test and component maintenance	932
• How do I maintain my test or component when my application changes?	932
• Can I increase or decrease Active Screen information after I finish recording a test?	933
• Testing localized applications	934
• Improving GUI testing performance	934
• How can I improve the working speed of UFT when working with GUI testing?	934
• How can I decrease the disk space used by UFT for GUI tests and components?	936

- [Is there a recommended length for tests?](#) 936

Programming in the Editor and function libraries

Relevant for: GUI tests and components

Can I store functions and subroutines in a function library?

You can create function libraries to hold your function and then call the functions from any test action or component by associating them with the test or with the component's application area. You can also register your functions as methods for UFT test objects. Your registered methods 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](#)" on page 575.

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 example:

```
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
```

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 to add an event to the Run Results Viewer-based report or the **AddRunInformation** or **ReportHtmlEvent** methods to add an event to the HTML report. 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.

You can also add custom report statements using the Insert Report dialog box.

For more details, see the **Reporter** object in the **Utility Objects** section of the *UFT Object Model Reference for GUI Testing*.

Working with dynamic content

Relevant for: **GUI tests and components**

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.

Use the **Exist** property to check whether the object exists in the application. This 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.

```
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
```

- 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 *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 *UFT Object Model Reference for GUI Testing*.

Advanced Web issues

Relevant for: GUI tests and components

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](#)" on page 279, and the **XMLUtil** object in the **Utility Objects** section of the *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
```

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 > HPUnified Functional Testing Help > VBScript Reference > Windows Script Host**).

Working with Windows applications

Relevant for: GUI tests and components

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**).

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>"**

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

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.

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.

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 148.

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.

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 330.

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

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:

- 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.
- 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.

- 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. 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 more details, see .
- Try to use the same application area for all components in a business process test.
- Minimize the number of actions in a test. Ideally, a test should not contain more than a few dozen actions.
- Store your functions in function libraries instead of as reusable actions.
- 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.
- 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**.
- 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. Set your options in the Active Screen pane of the Options dialog box (**Tools > Options > GUI Testing tab > Active Screen** node).
- 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.
- 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.
- Save tests on the file system instead of network drives.

How can I decrease the disk space used by UFT for GUI tests and components?

- Save screen captures and movie segments only in certain situations, such as when errors occur, or by not saving them at all. with 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.

- 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.

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 Testing.

Send Us Feedback



Let us know how we can improve your experience with the User Guide.

Send your email to: docteam@hpe.com

