



Peach Fuzzer Professional Developer Guide

Peach Fuzzer, LLC

Version 0.0.0

Copyright © 2016 Peach Fuzzer, LLC. All rights reserved.

This document may not be distributed or used for commercial purposes without the explicit consent of the copyright holders.

Peach Fuzzer® is a registered trademark of Peach Fuzzer, LLC.

Peach Fuzzer contains Patent Pending technologies.

While every precaution has been taken in the preparation of this book, the publisher and authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

Peach Fuzzer, LLC
1122 E Pike St
Suite 1064
Seattle, WA 98122

Table of Contents

1. Preface.....	1
1.1. What is Peach Fuzzer Professional?.....	1
1.2. A Brief History of Peach	2
1.3. Additional Resources	2
1.4. Bug Reporting Guidelines	2
1.4.1. Peach Forums.....	3
1.4.2. Support Tickets	3
2. Installation	4
2.1. Hardware Requirements.....	4
2.1.1. Local Target	4
2.1.2. Remote Target	5
2.2. Downloading	5
2.2.1. User Account Download	5
2.2.2. Enterprise Download	7
2.3. Windows	9
2.4. Linux	10
2.4.1. Ubuntu/Debian Linux.....	10
2.4.2. Redhat Enterprise Linux (RHEL and CentOS)	12
2.4.3. SUSE Enterprise Linux (SLES)	13
2.4.4. Other Linux Distributions.....	15
2.5. macOS	15
2.6. License Activation	17
2.6.1. Usage Based (Online Synchronization)	18
2.6.2. Usage Based (Offline Synchronization)	18
2.6.3. Node Locked.....	19
2.6.4. Enterprise	20
2.7. Enabling HTTPS And Authentication.....	20
2.7.1. Reverse Proxy with NGINX.....	20
3. What's new in Peach Fuzzer Professional v0.0.....	22
4. Introduction to Fuzzing	23
4.1. Dumb Fuzzing	25
4.2. Smart Fuzzing	26
4.3. When to Stop Fuzzing	27
5. Methodology	31
5.1. Selecting Fuzzing Targets	31
5.2. Developing Fuzzers	32

5.3. Secure Development Lifecycle.....	32
5.4. Deciding How Long to Fuzz	33
6. Introduction to Peach	34
7. Iteration Types	37
7.1. Record Iteration	37
7.2. Control Iteration	37
7.3. Fuzzing Iteration.....	38
8. Pit Files	40
9. Modeling.....	41
10. Data Modeling.....	42
10.1. Data Elements	43
10.1.1. Handling XML Documents	45
10.2. Relationships in Data.....	46
10.2.1. Adjusting Relation Values	46
10.2.2. Arrays of Offsets to Data	47
10.3. Cracking Data into the Data Model.....	50
10.3.1. Features Specific to Cracking	50
10.3.2. Debugging the Cracking Process	50
11. State Modeling.....	72
12. Providing Sample Data	75
12.1. Specifying Initial Values Using the Data Element.....	75
12.2. Specifying Initial Values in Data Files	75
13. Scripting in Peach.....	76
13.1. Python Scripting	76
13.2. In-line Expressions vs. Importing External Files	76
13.3. Importing External Files	77
13.4. Scriptable Areas in Peach.....	77
13.5. Accessing Data.....	79
13.6. Returning Peach Types	81
13.7. Debugging.....	81
13.8. API Reference	81
14. Monitoring The Fuzzing Environment	82
14.1. Detecting Faults.....	84
14.1.1. Memory debuggers.....	84
14.2. Instrumenting the Fuzzing Environment.....	85
15. Test.....	87
16. Configuration Files.....	88
16.1. Internally-defined Keys.....	89

17. Debugging Pit Files	91
17.1. What to do if the Pit Doesn't Parse	91
17.2. The pit doesn't run properly	91
18. Converting Pits from Peach V2.3	93
18.1. Global Changes	93
18.2. Changes to Individual xml Elements	96
18.3. How to Make a Pit Usable by the Peach Web User Interface	98
19. Running Peach	100
19.1. The Peach Web Interface	101
19.1.1. Peach Web Interface Installation Requirements	101
19.1.2. Starting the Peach Web Interface	102
19.1.3. Configuration Menu	111
19.1.4. Fuzzing Session	139
19.1.5. Faults	149
19.1.6. Metrics	151
19.2. The Peach Command Line Interface	157
19.2.1. Peach Web Application	157
19.2.2. Fuzzing from Command Line	157
19.2.3. Debug Peach XML File	159
19.2.4. Display List of Network Capture Devices	159
19.2.5. Display Known Elements	159
19.2.6. Peach Agent	160
19.2.7. Running Analyzers from Command Line	160
19.2.8. Generate XML Schema File	160
19.2.9. Examples	160
19.3. PeachAgent	164
19.3.1. Licensing	164
19.3.2. Syntax	164
19.4. Minset	165
19.4.1. Collect Traces	165
19.4.2. Compute Minimum Set Coverage	165
19.4.3. All-In-One	166
19.4.4. Distributing Minset	166
19.4.5. Examples	166
19.5. Peach Validator	168
19.6. Peach Multi-Node CLI Tool	169
19.6.1. Installation	169
19.6.2. Syntax	169

19.6.3. Commands	169
19.7. Pit Tool	174
19.7.1. Syntax	174
19.7.2. Commands	174
19.8. Pit Tool - Analyzer	175
19.8.1. Syntax	175
19.8.2. Analyzers	175
19.9. Pit Tool - Compile	176
19.9.1. Syntax	176
19.9.2. Parameters	176
19.9.3. Verify PitDefines	177
19.9.4. PitLint Checks	177
19.9.5. Generate Tuning Metadata	178
19.9.6. Generate Sample Ninja Database	178
19.10. Pit Tool - Crack	179
19.10.1. Syntax	179
19.10.2. Parameters	179
19.10.3. Example	179
19.11. Pit Tool - Make XSD	181
19.11.1. Syntax	181
19.11.2. Parameters	181
19.11.3. Example	181
19.12. Pit Tool - Ninja	182
19.12.1. Syntax	182
19.12.2. Parameters	182
19.12.3. Examples	183
20. Reproducing Faults	185
20.1. Iteration Based Targets	185
20.2. Session Based Targets	186
20.2.1. Searching For Reproduction	186
20.3. How to Control the Automated Fault Reproduction	187
20.4. Replay the Fuzzing Session	188
21. Reference	189
21.1. General	189
21.1.1. Peach Workflow	190
21.1.2. Data	191
21.1.3. Defaults	194
21.1.4. Import	196

21.1.5. Include.....	197
21.1.6. Loggers	199
21.1.7. Pluggable Mutation Strategies.....	199
21.1.8. Param	201
21.1.9. PythonPath.....	204
21.1.10. Test.....	204
21.1.11. Exclude	207
21.1.12. Include.....	210
21.1.13. Mutators	212
21.1.14. Web Path.....	214
21.1.15. Web Query.....	218
21.1.16. Web Header.....	220
21.1.17. Web FormData	222
21.1.18. Web Body	224
21.1.19. Web Part	226
21.1.20. Web Response	232
21.2. Analyzers	235
21.2.1. ASN.1 Analyzer	236
21.2.2. Binary Analyzer	247
21.2.3. BSON Analyzer	255
21.2.4. JSON Analyzer	260
21.2.5. Postman Analyzer.....	265
21.2.6. Regex Analyzer	266
21.2.7. String Token Analyzer.....	271
21.2.8. Swagger Analyzer.....	279
21.2.9. Vcr Analyzer	281
21.2.10. Xml Analyzer.....	283
21.2.11. WebRecordProxy Analyzer	292
21.2.12. Zip Analyzer.....	293
21.3. Data Elements	311
21.3.1. Asn1Type	314
21.3.2. Blob	318
21.3.3. Block.....	327
21.3.4. Bool.....	340
21.3.5. Choice	344
21.3.6. DataModel.....	353
21.3.7. Double	361
21.3.8. Flag.....	372

21.3.9. Flags	377
21.3.10. Frag	382
21.3.11. JsonArray	388
21.3.12. JsonBlob	392
21.3.13. JsonBool	398
21.3.14. JsonDouble	402
21.3.15. JsonInteger	411
21.3.16. JsonObject	416
21.3.17. JsonRaw	421
21.3.18. JsonString	425
21.3.19. Null	435
21.3.20. Number	439
21.3.21. Padding	459
21.3.22. Sequence	467
21.3.23. Stream	481
21.3.24. String	491
21.3.25. VarNumber	510
21.3.26. XmlAttribute	516
21.3.27. XmlCharacterData	523
21.3.28. XmlElement	528
21.4. Data Element Children	538
21.4.1. Relation	538
21.4.2. Placement	545
21.5. Fixups	546
21.5.1. CiscoCdpChecksum	548
21.5.2. CopyValue	551
21.5.3. CrcDual	555
21.5.4. Crc	557
21.5.5. Expression	560
21.5.6. FillValue	564
21.5.7. FragSeqIncrement	567
21.5.8. FromFile	570
21.5.9. IcmpChecksum	571
21.5.10. IcmpV6Checksum	574
21.5.11. IsoFletcher16Checksum	577
21.5.12. Lrc	580
21.5.13. Hmac	583
21.5.14. Md5	588

21.5.15. Script	591
21.5.16. SequenceIncrement	594
21.5.17. SequenceRandom	603
21.5.18. Sha1	606
21.5.19. Sha224	609
21.5.20. Sha256	612
21.5.21. Sha384	615
21.5.22. Sha512	618
21.5.23. Sspi	621
21.5.24. TCPChecksum	624
21.5.25. UDPChecksum	627
21.5.26. UnixTime	630
21.6. Transformers	635
21.6.1. Example	635
21.6.2. Aes128	637
21.6.3. Base64Decode	641
21.6.4. Base64Encode	645
21.6.5. Bz2Compress	649
21.6.6. Bz2Decompress	652
21.6.7. Des	656
21.6.8. GzipCompress	660
21.6.9. GzipDecompress	664
21.6.10. Hex	668
21.6.11. Hmac	672
21.6.12. HtmlDecode	675
21.6.13. HtmlEncode	679
21.6.14. IntToHex	684
21.6.15. Ipv4StringToOctet	688
21.6.16. Ipv6StringToOctet	692
21.6.17. JsEncode	696
21.6.18. Md5	699
21.6.19. NetBiosDecode	702
21.6.20. NetBiosEncode	706
21.6.21. Sha1	710
21.6.22. Sha256	713
21.6.23. SidStringToBytes	716
21.6.24. TripleDes	720
21.6.25. Truncate	724

21.6.26. UrlEncode	727
21.7. State Modeling	731
21.7.1. StateModel	731
21.7.2. State	740
21.7.3. Action	744
21.8. Agents	863
21.8.1. Agent Channels	865
21.9. Monitors	883
21.9.1. Android Monitor	885
21.9.2. AndroidEmulator Monitor	891
21.9.3. APC Power Monitor	896
21.9.4. ButtonClicker Monitor	900
21.9.5. CanaKitRelay Monitor	905
21.9.6. CAN Capture Monitor	911
21.9.7. CAN Error Frame Monitor	912
21.9.8. CAN Send Frame Monitor	913
21.9.9. CAN Timing Monitor	918
21.9.10. CAN Threshold Monitor	921
21.9.11. CleanupFolder Monitor	925
21.9.12. CleanupRegistry Monitor (Windows)	930
21.9.13. CrashReporter Monitor (OS X)	934
21.9.14. CrashWrangler Monitor (OS X)	936
21.9.15. Gdb Monitor (Linux, OS X)	939
21.9.16. GdbServer Monitor (Linux, OS X)	945
21.9.17. IpPower9258 Monitor	951
21.9.18. LinuxCoreFile Monitor (Linux)	956
21.9.19. Memory Monitor	962
21.9.20. NetworkCapture Monitor	966
21.9.21. PageHeap Monitor (Windows)	971
21.9.22. Ping Monitor	974
21.9.23. PopupWatcher Monitor (Windows)	978
21.9.24. Process Monitor	983
21.9.25. ProcessKiller Monitor	993
21.9.26. Run Command	996
21.9.27. SaveFile Monitor	1002
21.9.28. Serial Port Monitor	1004
21.9.29. SNMP Power Monitor	1012
21.9.30. Socket Monitor	1016

21.9.31. SshCommand Monitor	1018
21.9.32. SshDownloader Monitor	1023
21.9.33. Syslog Monitor	1025
21.9.34. TcpPort Monitor	1031
21.9.35. Vmware Monitor	1038
21.9.36. WindowsDebugger Monitor (Windows)	1042
21.9.37. WindowsKernelDebugger Monitor (Windows)	1047
21.9.38. WindowsService Monitor (Windows)	1049
21.10. Publishers	1050
21.10.1. Network Publishers	1050
21.10.2. Publishers	1050
21.10.3. Syntax	1051
21.10.4. Examples	1051
21.10.5. AndroidMonkey Publisher	1054
21.10.6. CAN Publisher	1057
21.10.7. Com Publisher	1060
21.10.8. Console Publisher	1063
21.10.9. ConsoleHex Publisher	1065
21.10.10. File Publisher	1068
21.10.11. FilePerIteration Publisher	1071
21.10.12. Http Publisher (Deprecated)	1073
21.10.13. I2C Publisher	1080
21.10.14. RawEther Publisher	1083
21.10.15. RawIPv4 Publisher	1086
21.10.16. RawV4 Publisher	1090
21.10.17. RawV6 Publisher	1094
21.10.18. Remote Publisher	1098
21.10.19. Rest Publisher (Deprecated)	1101
21.10.20. SerialPort Publisher	1114
21.10.21. Ssl Publisher	1117
21.10.22. SslListener Publisher	1120
21.10.23. Tcp Client Publisher	1123
21.10.24. TcpListener Publisher	1126
21.10.25. Udp Publisher	1128
21.10.26. USB Publisher	1133
21.10.27. WebApi Publisher	1137
21.10.28. WebService Publisher	1147
21.10.29. WebSocket Publisher	1149

21.10.30. Zip Publisher	1152
21.11. Mutators	1155
21.11.1. ArrayEdgeCase	1156
21.11.2. ArrayRandomizeOrder	1157
21.11.3. ArrayReverseOrder	1158
21.11.4. ArrayVarianceMutator	1159
21.11.5. BlobChangeFromNull	1160
21.11.6. BlobChangeRandom	1161
21.11.7. BlobChangeSpecial	1162
21.11.8. BlobChangeToNull	1163
21.11.9. BlobExpandAllRandom	1164
21.11.10. BlobExpandSingleIncrementing	1165
21.11.11. BlobExpandSingleRandom	1166
21.11.12. BlobExpandZero	1167
21.11.13. BlobReduce	1168
21.11.14. ChoiceSwitch	1169
21.11.15. DataElementBitFlipper	1170
21.11.16. DataElementDuplicate	1171
21.11.17. DataElementRemove	1172
21.11.18. DataElementSwapNear	1173
21.11.19. DoubleRandom	1174
21.11.20. DoubleVariance	1175
21.11.21. ExtraValues	1176
21.11.22. NumberEdgeCase	1177
21.11.23. NumberRandom	1178
21.11.24. NumberVariance	1179
21.11.25. SampleNinja	1180
21.11.26. SizedDataEdgeCase	1181
21.11.27. SizedDataVariance	1182
21.11.28. SizedEdgeCase	1183
21.11.29. SizedVariance	1184
21.11.30. StateChangeRandom	1185
21.11.31. StringAsciiRandom	1186
21.11.32. StringCaseLower	1187
21.11.33. StringCaseRandom	1188
21.11.34. StringCaseUpper	1189
21.11.35. StringLengthEdgeCase	1190
21.11.36. StringLengthVariance	1191

21.11.37. StringList	1192
21.11.38. StringStatic	1193
21.11.39. StringUnicodeAbstractCharacters	1194
21.11.40. StringUnicodeFormatCharacters	1195
21.11.41. StringUnicodeInvalid	1196
21.11.42. StringUnicodeNonCharacters	1197
21.11.43. StringUnicodePlane0	1198
21.11.44. StringUnicodePlane1	1199
21.11.45. StringUnicodePlane14	1200
21.11.46. StringUnicodePlane15And16	1201
21.11.47. StringUnicodePlane2	1202
21.11.48. StringUnicodePrivateUseArea	1203
21.11.49. StringUtf8BomLength	1204
21.11.50. StringUtf8BomStatic	1205
21.11.51. StringUtf8ExtraBytes	1206
21.11.52. StringUtf8Invalid	1207
21.11.53. StringUtf16BomLength	1208
21.11.54. StringUtf16BomStatic	1209
21.11.55. StringUtf32BomLength	1210
21.11.56. StringUtf32BomStatic	1211
21.11.57. StringXmlW3C	1212
21.12. Common Attributes and Parameters	1213
21.12.1. Constraint Attribute	1213
21.12.2. Endian Attribute	1214
21.12.3. Field	1214
21.12.4. fileName	1215
21.12.5. Hint	1216
21.12.6. Length Attribute	1216
21.12.7. Length Type Attribute	1216
21.12.8. Maximum Occurrence Attribute	1217
21.12.9. Minimum Occurrence Attribute	1218
21.12.10. Mutable Attribute	1218
21.12.11. Name Attribute	1219
21.12.12. Occurs Attribute	1220
21.12.13. State onStart Attribute	1220
21.12.14. State onComplete Attribute	1226
21.12.15. Action onComplete Attribute	1232
21.12.16. Action onStart Attribute	1238

21.12.17. Ref Attribute	1244
21.12.18. Signed Attribute	1244
21.12.19. Size Attribute.....	1244
21.12.20. Token Attribute.....	1245
21.12.21. Value Attribute	1245
21.12.22. ValueType Attribute	1246
21.12.23. Action when Attribute.....	1249
21.12.24. xpath	1254
22. Extending Peach.....	1255
22.1. Peach Plug-ins.....	1255
22.1.1. Examples.....	1257
22.2. Fixup	1259
22.2.1. Examples.....	1260
22.3. Monitor.....	1262
22.4. Publisher	1264
22.5. Transformer	1268
22.6. Mutator.....	1269
22.7. Agent	1272
23. Tutorials.....	1280
23.1. Tutorial: Dumb Fuzzing	1281
23.1.1. The Peach Development Environment	1281
23.1.2. Creating Data Models.....	1281
23.1.3. Create State Model	1282
23.1.4. Configure Publisher.....	1283
23.1.5. Agent and Monitor	1284
23.1.6. Running the Fuzzer	1287
23.2. Tutorial: File Fuzzing	1288
23.2.1. The Peach Development Environment	1288
23.2.2. Creating Wave Data Models.....	1288
23.2.3. Finishing the Wav Model	1300
23.2.4. Create State Model	1301
23.2.5. Configure Publisher.....	1301
23.2.6. Agent and Monitor	1302
23.2.7. Optimize Testing.....	1305

1. Preface

This document is one of two books that together form the official documentation of Peach Fuzzer Professional v0.0. This documentation set is written by the Peach Fuzzer Professional team, and represents a concerted effort to document fully all of the Peach Fuzzer Professional features.

This book, the Peach Fuzzer Professional Developer Guide, focuses on developer needs such as advanced configurations, building custom fuzzing definitions, and extending the various areas of Peach. Topics of interest include Modeling, Providing Sample Data, Running Peach, Tutorials, and Extending Peach. The reference section is comprehensive.

The primary audience for this guide is a software developer who is familiar with C, Python, and XML. Also needed is intimate knowledge of the test target because that drives design decisions.

Day-to-day activities are located in the Peach Fuzzer Professional User Guide that focuses on running Peach with the Web user interface, reading fuzzing results, and setting up monitors from a set of recipes.

Peach Fuzzer has been in active development through three major revisions since 2004. Until 2014, no complete documentation existed. The current Developer Guide and User Guide bear witness that documenting Peach is an on-going effort.

1.1. What is Peach Fuzzer Professional?

Peach Fuzzer Professional (Peach) is a fuzzer for data consumers. Peach is a smart fuzzer that operates by performing the following actions:

- Understand the data structure and the flows of the test target
- Create and feed malformed data to the test target
- Monitor the test target to record interesting information when unintended or undesirable behavior occurs (monitors include debuggers and network packet sniffers)

Peach is a versatile product and has been used to fuzz a wide range of products and devices:

- web browsers (file consumers)
- web servers (network servers)
- mobile devices (such as Android iOS)
- robots
- SCADA systems
- Semiconductor chips

Because Peach can easily extend its interfacing and monitoring, it is the most adaptable fuzzer that exists.

1.2. A Brief History of Peach

Peach Fuzzer has been in active development since 2004. Since then it has gone through three major versions:

Peach v1

Peach v1 was written in 2004 during the PH-Neutral conference in Berlin Germany. Peach was the first fuzzer to take a more object-oriented approach. Because Peach was originally written in Python, this version required writing Python code to target and utilize.

Peach v2

Peach v2 was a complete redesign of Peach based on the lessons learned developing and using the first version. It introduced all of the core concepts still used in Peach today.

Because Peach v2 was the first true modeling fuzzer available, it became widely used in the industry. This was the first Peach version to use the three M's approach to fuzzing: Model, Mutate, and Monitor. Peach v2 still required writing Python code to target and utilize.

Peach v3

Peach had outgrown Python as a language so Peach v3 was rewritten using the Microsoft .NET environment and the C# language.

In Peach v3, the core concepts remain the same, but the internals were re-architected to allow for the next generation of features. The fuzzing definitions (Pits) are largely backwards compatible with the prior version. Peach v3 introduced Peach Farm, the tool set for scaling fuzzing and for controlling fuzzing server farms.

Peach is currently developed and maintained by Peach Fuzzer, LLC, in Seattle, WA.

1.3. Additional Resources

More information about Peach is available on the Web:

- [Peach Fuzzer website](#)
- [Current Pits and Pit Packs](#)
- [Peach User Forums](#)
- [Installing Peach Fuzzer Software video](#)

1.4. Bug Reporting Guidelines

Support for Peach Fuzzer is available in two ways:

- The Peach Forums site
- Using the Peach ticketing system to open a support ticket

1.4.1. Peach Forums

There are two sets of forums for Peach, the community forums and the professional forums. Both forums are hosted at <https://forums.peachfuzzer.com>.

Peach Fuzzer Professional users should access the private forums to receive support for the commercial versions of Peach. Responses on the commercial forums are prioritized over the public community forums.

To access the Peach Fuzzer Professional forums, follow these steps:

1. Register an account on the forums site.
2. Send an email to support@peachfuzzer.com with your license email and forum username.
3. Your account will be granted access to the commercial forums within 24 business hours.

Forums are monitored by the team at Peach Fuzzer, LLC; however, there is no guarantee of response time.

1.4.2. Support Tickets

You can open a support ticket by sending an email to support@peachfuzzer.com. You will receive an initial response within 24 business hours of opening the ticket. Peach support is available Monday through Friday. Peach support is not currently available on weekends or holidays.

When opening a ticket, please provide the following information in your email:

- Operating system(s) in use by Peach and any agents
- Exact version of Peach being used. This is available from the console output and in the *status.txt* log file
- Detailed description of the issue and expected behavior
- Console output using the *--trace* argument
- (if possible) the full Pit file and configuration files

2. Installation

The following list contains links to the sections that describe in detail how to download, install and activate Peach Fuzzer.

Recommended Hardware

This section describes the minimum and recommended hardware requirements for common fuzzing scenarios.

Product Download

This section contains the steps required to download the product for your desired platform.

Product Installation

This section lists the software prerequisites, OS specific configuration, and steps for installing the product on each of the three supported platforms:

- [Windows](#)
- [Linux](#)
- [OSX](#)

Product Activation

The list below contains links to the steps for activating the different types of Peach Fuzzer licenses.

- [Usage Based \(Online Synchronization\)](#)
- [Usage Based \(Offline Synchronization\)](#)
- [Node Locked](#)
- [Enterprise](#)

Optional Configuration

The list below contains links to optional post-install configurations.

- [Enabling HTTPS And Authentication](#)

2.1. Hardware Requirements

The following are generic hardware recommendations. Adjust based on your needs.

2.1.1. Local Target

When fuzzing a local target (software running on the same machine as Peach), additional resources are required for the target process.

Target Type	Architecture	Cores	Ram	Disk
Network	64-bit	4	8GB	60GB SSD
File	64-bit	4	16GB	60GB SSD
Other	64-bit	4	8GB	60GB SSD

2.1.2. Remote Target

Remote target fuzzing occurs when the target is not located on the machine running Peach.

	Architecture	Cores	Ram	Disk
Minimum	64-bit	2	4GB	60GB Any
Recommended	64-bit	2	8GB	60GB SSD



It's possible to run Peach on 32-bit systems, but it's not recommended as it places severe limits on memory usage (max 2GB).

2.2. Downloading

The first step of the installation is to download the Peach Fuzzer distribution files from the Peach download site. Once the appropriate files have been downloaded, follow the instructions for your specific operating system found in the next section.

User Account Download

If you were assigned an account with a username/password, follow these instructions to sign in and download Peach Fuzzer.

Enterprise Download

If you were provided an enterprise license file, follow the instructions to sign in and download Peach Fuzzer.

2.2.1. User Account Download

1. When your Peach Fuzzer welcome email arrives, click the link to reset your initial password.
2. Navigate to <https://portal.peachfuzzer.com> with your preferred web browser.
3. At the login prompt under the Portal Login section, enter the new username/password that was recently reset and click **Sign In**.

PEACHTECH

Portal Login

Username
Password
Sign In

Enterprise Users

If you were provided a license file 'Peach.license', upload that file here to log in.

No file chosen

Sign In with License



Contact support@peach.tech if you need your account password reset.

4. On the [Downloads](#) page, select the Peach Fuzzer release version and operating system to install.
 - a. Choose a release version from the items on the left side.
 - b. Click the download icon on the right side after deciding which OS and architecture is needed.

PEACHTECH Solutions Forums Tickets **Downloads** Licensing Logout

PEACH FUZZER

- v4.3** 
- v4.2

PEACH API SECURITY

- v1.5
- v1.4
- v1.3
- v1.2
- v1.1
- v1.0

File Name	Size
Client_Peach_Trial_Guide-ICS.pdf	1 MB
Client_Peach_Trial_Guide-Network.pdf	1 MB
Hosted_Peach_Trial_Guide-Fileformat.pdf	1 MB
Hosted_Peach_Trial_Guide-Healthcare.pdf	1 MB
Hosted_Peach_Trial_Guide-ICS.pdf	1 MB
Hosted_Peach_Trial_Guide-Network.pdf	1 MB
Peach_Pro_Changes.pdf	0 MB
Peach_Pro_Installation_Guide.pdf	1 MB
Peach_Pro_User_Guide.pdf	7 MB
peach-pro-4.3.284-linux_x86_64_release.zip	69 MB
peach-pro-4.3.284-linux_x86_release.zip	69 MB
peach-pro-4.3.284-osx_release.zip	73 MB
peach-pro-4.3.284-sdk.zip	36 MB
peach-pro-4.3.284-win_x64_release.zip	83 MB
peach-pro-4.3.284-win_x86_release.zip	80 MB

5. If your organization has multiple entitlements, you may need to select an Activation ID that corresponds to the license the download should be tied to. This selection page will not be displayed

if there is only one entitlement for your organization. Contact licensing@peach.tech for more information if you are unsure which Activation ID to select.

6. After a few moments, an End User License Agreement acceptance page appears. Click **I ACCEPT** to continue.

The screenshot shows a blue header bar with the PeachTech logo and navigation links: Solutions, Forums, Tickets, Downloads, Licensing, and Logout. Below the header is a white content area with a light gray border. At the top of the content area, it says "END-USER LICENSE AGREEMENT: PEACH FUZZER™". Below this, a paragraph of text states: "BY DOWNLOADING, ACCESSING OR USING THE SOFTWARE, DEFINITION FILES OR FEATURES, YOU AUTOMATICALLY ACKNOWLEDGE, ACCEPT AND AGREE TO ALL THE TERMS OF THE PEACH FUZZER™ END-USER LICENSE AGREEMENT LOCATED AT <https://www.peach.tech/eula/flex/>. IF YOU DO NOT AGREE TO THESE TERMS AND CONDITIONS, YOU MAY NOT DOWNLOAD, ACCESS OR OTHERWISE USE THE SOFTWARE AND ACCOMPANYING FEATURES." Underneath this text is a section titled "CONTACT INFORMATION" with the instruction: "If you have any questions about this EULA, or if you want to contact Peach Fuzzer, LLC for any reason, please direct all correspondence to: contact@peach.tech". At the bottom of the content area are two buttons: a blue "I ACCEPT" button and a dark gray "Cancel" button. At the very bottom of the page, outside the main content area, is a small copyright notice: "Copyright © 2019 Peach Fuzzer, LLC . All rights reserved."

7. The download will begin. Depending on your network connection, this could take a few minutes.

2.2.2. Enterprise Download



You need a copy of your Peach Fuzzer license on your system to perform the download.

1. Using a web browser, navigate to <https://portal.peachfuzzer.com>

PEACHTECH

Portal Login

Username
Password

Sign In

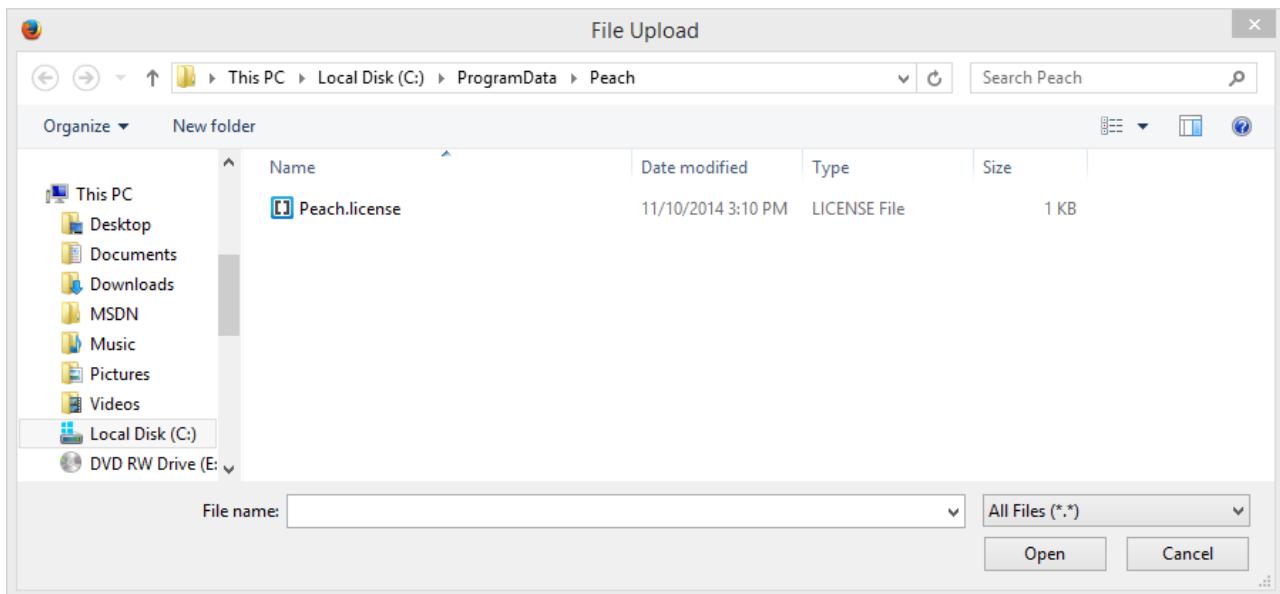
Enterprise Users

If you were provided a license file 'Peach.license', upload that file here to log in.

No file chosen

Sign In with License

- a. Click the **Choose File** button. The upload dialog display allows you to select the **Peach.license** file.



- b. Navigate to the location of the license
 - c. Select the license (**Peach.license**)
 - d. Click **Open** to return to the download home page.
2. Upon returning to the Peach download home page, click **Sign In with License**.
 3. On the **Downloads** page, select the Peach Fuzzer release version and operating system to install.
 - a. Choose a release version from the items on the left side.
 - b. Click the download icon on the right side after deciding which OS and architecture is needed.

PEACH FUZZER

[v4.3](#)[v4.2](#)

PEACH API SECURITY

[v1.5](#)[v1.4](#)[v1.3](#)[v1.2](#)[v1.1](#)[v1.0](#)

v4.3.284 (Thursday, November 15, 2018)

 Client_Peach_Trial_Guide-ICS.pdf	1 MB
 Client_Peach_Trial_Guide-Network.pdf	1 MB
 Hosted_Peach_Trial_Guide-Fileformat.pdf	1 MB
 Hosted_Peach_Trial_Guide-Healthcare.pdf	1 MB
 Hosted_Peach_Trial_Guide-ICS.pdf	1 MB
 Hosted_Peach_Trial_Guide-Network.pdf	1 MB
 Peach_Pro_Changes.pdf	0 MB
 Peach_Pro_Installation_Guide.pdf	1 MB
 Peach_Pro_User_Guide.pdf	7 MB
 peach-pro-4.3.284-linux_x86_64_release.zip	69 MB
 peach-pro-4.3.284-linux_x86_release.zip	69 MB
 peach-pro-4.3.284-osx_release.zip	73 MB
 peach-pro-4.3.284-sdk.zip	36 MB
 peach-pro-4.3.284-win_x64_release.zip	83 MB
 peach-pro-4.3.284-win_x86_release.zip	80 MB

4. After a few moments, an End User License Agreement acceptance page appears. Click **I ACCEPT** to continue.

END-USER LICENSE AGREEMENT: PEACH FUZZER™

BY DOWNLOADING, ACCESSING OR USING THE SOFTWARE, DEFINITION FILES OR FEATURES, YOU AUTOMATICALLY ACKNOWLEDGE, ACCEPT AND AGREE TO ALL THE TERMS OF THE PEACH FUZZER™ END-USER LICENSE AGREEMENT LOCATED AT <https://www.peach.tech/eula/flex/>. IF YOU DO NOT AGREE TO THESE TERMS AND CONDITIONS, YOU MAY NOT DOWNLOAD, ACCESS OR OTHERWISE USE THE SOFTWARE AND ACCOMPANYING FEATURES.

CONTACT INFORMATION

If you have any questions about this EULA, or if you want to contact Peach Fuzzer, LLC for any reason, please direct all correspondence to:
contact@peach.tech

I ACCEPT**Cancel**

Copyright © 2019 Peach Fuzzer, LLC . All rights reserved.

5. The download will begin. Depending on your network connection, this could take a few minutes.

2.3. Windows

Peach is officially supported on the following Windows® Operating Systems:

- Windows 7 SP1 (x86 and x64)
- Windows 8 (x86 and x64)

- Windows 8.1 (x86 and x64)
- Windows 10 (x64)
- Windows Server 2008 SP2 (x86 and x64)
- Windows Server 2008 R2 SP1 (x64)
- Windows Server 2012 (x64)
- Windows Server 2012 R2 (x64)

The only required software is the Microsoft .NET Framework v4.5.

1. Download and install the [Microsoft .NET Framework v4.5.2 \(Installer\)](#).
2. Install the [Microsoft Debugging Tools for Windows](#) (optional).



This is only required if you want to use a debugger to detect crashes in fuzzed programs.

3. Install [Wireshark](#) (optional).



This is only required if you want to collect network captures during fuzzing runs.

4. Unzip the Peach distribution to the appropriate folder. The file is a zip file with the extension `.zip`. Use the filename that begins with `peach-pro` and contains the appropriate architecture for your system, such as `peach-pro-0.0.0-win_x64_release.zip`.
5. When fuzzing, many security products (such as anti-virus programs) can interfere or slowdown fuzzing. For network fuzzing, make sure none of the network or host-based network intrusion detection systems (IDS) are running. For file fuzzing, disable anti-virus software; or mark Peach, the target application, and any directories that might have files used in fuzzing, as out of scope for real time monitoring.

2.4. Linux

Peach is supported on three distributions of Linux; Peach may run on other Linux distributions, but are not officially supported. This section provides instructions for installing Peach on the following supported Linux systems, and includes a checklist for installing Peach on other Linux systems:

- Ubuntu/Debian Linux
- Redhat Enterprise Linux (RHEL and CentOS)
- SUSE Enterprise Linux (SLES)

2.4.1. Ubuntu/Debian Linux

When installing Peach on an Ubuntu or Debian Linux system, the operating system is ready for Peach without modification. The installation starts with the Mono .NET runtime, then Peach. If you want to

attach a debugger to a target process, install GDB when the Mono installation completes.

Peach Fuzzer, LLC, recommends using Ubuntu Linux version 16.04 LTS, and Mono .NET runtime version 4.8.1 from the mono project. The mono project has [apt](#) packages for Ubuntu.

Peach will not run with Mono version 5.0 or newer due to incompatibilities with IronPython. If you have Mono 5.0 installed, you must downgrade to 4.8.1.

 Peach will not run with version 4.4 of the Mono runtime as there are known handle leaks which can cause Peach to run out of memory during long fuzzing runs. If you have Mono 4.4 installed, you can either upgrade to 4.6+ or downgrade to 4.2.

Some Linux kernel versions have known issues with the Mono runtime. When using Ubuntu 14.04 LTS, avoid using kernel versions 3.13.0-48 through 3.13.0-54 inclusive. Version 14.04 LTS might require that you update the Linux kernel. If so, perform the following to update the kernel: [sudo apt-get install linux-image-generic](#).

The Peach installer checks for compatibility and alerts the user if an incompatibility has been detected.

The following steps will prepare Peach to run properly:

1. Install the latest [mono-complete](#) package.

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys  
3FA7E0328081BFF6A14DA29AA6A19B38D3D831EF  
  
echo "deb http://download.mono-project.com/repo/debian wheezy/snapshots/4.8.1.0 main"  
| sudo tee /etc/apt/sources.list.d/mono-xamarin.list  
  
sudo apt-get update  
  
sudo apt-get install mono-complete
```

2. Install libpcap using the following command:

```
sudo apt-get install libpcap
```

3. Optionally, install the GNU Debugger (GDB) to enable debugging of local processes.

```
sudo apt-get install gdb
```

4. Unzip the Peach distribution to the appropriate folder. The file is a zip file with the extension [.zip](#).

Use the filename that begins with `peach-pro` and contains the appropriate architecture for your system, such as `peach-pro-0.0.0-linux_x86_64_release.zip`.

2.4.2. Redhat Enterprise Linux (RHEL and CentOS)

Installing Peach on a RHEL CentOS platform requires additional steps. Begin by installing Extra Packages for Enterprise Linux (EPEL), followed by the Mono package and Peach.

Peach Fuzzer, LLC, recommends using Mono .NET runtime version 4.8.1 from the mono project. The mono project has `yum` packages for RHEL and CentOS distributions.

 Peach will not run with Mono version 5.0 or newer due to incompatibilities with IronPython. If you have Mono 5.0 installed, you must downgrade to 4.8.1.

Peach will not run with version 4.4 of the Mono runtime as there are known handle leaks which can cause Peach to run out of memory during long fuzzing runs. If you have Mono 4.4 installed, you can either upgrade to 4.6+ or downgrade to 4.2.

 The following Mono installation steps are taken from the [Mono Project](#).

The following steps provide the needed details:

1. Install `yum-utils` using the following command:

```
sudo yum install yum-utils
```

2. Install Extra Packages for Enterprise Linux (EPEL) using the following command:

```
sudo yum install epel-release
```

3. Import the GPG signing key for the mono package using the following command. Note the long search key:

```
sudo rpm --import  
"http://keyserver.ubuntu.com/pks/lookup?op=get&search=0x3FA7E0328081BFF6A14DA29AA6A19B  
38D3D831EF"
```

4. Add and enable the mono project repository for CentOS using the yum configuration manager:

```
sudo yum-config-manager --add-repo http://download.mono-project.com/repo/centos/
```

5. Install the latest version of Mono using the following command:

```
sudo yum install mono-complete-4.8.1.0-0.xamarin.1
```

6. Install libpcap using the following command:

```
sudo yum install libpcap
```

7. Unzip the Peach distribution to the appropriate folder. The file is a zip file with the extension **.zip**. Use the filename that begins with **peach-pro** and contains the appropriate architecture for your system, such as **peach-pro-0.0.0-linux_x86_64_release.zip**.

If you receive an error regarding libMonoPosixHelper the **/etc/mono/config** file may need to be edited. To edit locate a line that looks like the following (the path may be different):

```
<dllmap dll="MonoPosixHelper" target="/usr/lib/libMonoPosixHelper.so" os="!windows" />
```



Once found use the Linux *find* command to locate the shared library:

```
find /usr -name "*libMonoPosixHelper.so"
```

And finally, update the **/etc/mono/config** entry to the correct path.

i For more information, see the following resources:
* <http://www.mono-project.com/docs/getting-started/install/linux#centos-fedora-and-derivatives>
* https://fedoraproject.org/wiki/EPEL#How_can_I_use_these_extra_packages.3F

2.4.3. SUSE Enterprise Linux (SLES)

To install Peach on a SUSE Enterprise Linux platform, use the 1-click SUSE mono-complete installation file. If you want to attach a debugger to a target process, install GDB when the Mono installation completes.

Peach Fuzzer, LLC, recommends using Mono .NET runtime version 4.8.1 from the mono project. The mono project has packages for SLES distributions.

 Peach will not run with Mono version 5.0 or newer due to incompatibilities with IronPython. If you have Mono 5.0 installed, you must downgrade to 4.8.1.

Peach will not run with version 4.4 of the Mono runtime as there are known handle leaks which can cause Peach to run out of memory during long fuzzing runs. If you have Mono 4.4 installed, you can either upgrade to 4.6+ or downgrade to 4.2.

The following steps provide the needed details:

1. Import the GPG signing key for the mono package using the following command. Note the long search key:

```
sudo rpm --import  
"http://keyserver.ubuntu.com/pks/lookup?op=get&search=0x3FA7E0328081BFF6A14DA29AA6A19B  
38D3D831EF"
```

2. Add and enable the mono project repository using the zypper configuration manager:

```
sudo zypper ar -f http://download.mono-project.com/repo/centos/ mono
```

3. Install the latest supported version of Mono using the following command:

```
sudo zypper in mono-complete=4.8.1.0-0.xamarin.1
```

4. Install libpcap using the following command:

```
sudo zypper in libpcap
```

5. Optionally, install the GNU Debugger (GDB) for debugging local processes.

```
sudo yum install gdb
```

6. Unzip the Peach binary distribution to the appropriate folder. The file is a zip file with the extension **.zip**. Use the filename that begins with **peach-pro** and contains the appropriate architecture for your system, such as **peach-pro-0.0.0-linux_x86_64_release.zip**.

If you receive an error regarding libMonoPosixHelper the `/etc/mono/config` file may need to be edited. To edit locate a line that looks like the following (the path may be different):



```
<dllmap dll="MonoPosixHelper" target="/usr/lib/libMonoPosixHelper.so" os="!windows" />
```

Once found use the Linux `find` command to locate the shared library:

```
find /usr -name "*libMonoPosixHelper.so"
```

And finally, update the `/etc/mono/config` entry to the correct path.

2.4.4. Other Linux Distributions

For other Linux versions, the installation steps are a checklist, not specific commands. The checklist follows:

1. Install the Mono runtime. Version 4.8.1 is recommended.
2. Unzip the Peach distribution to an appropriate folder. The file is a zip file with the extension `.zip`. Use the filename that begins with `peach-pro` and contains the appropriate architecture for your system, such as `peach-pro-0.0.0-linux_x86_64_release.zip`.

Peach Fuzzer, LLC, recommends using Mono .NET runtime version 4.8.1 from the mono project.



Peach will not run with Mono version 5.0 or newer due to incompatibilities with IronPython. If you have Mono 5.0 installed, you must downgrade to 4.8.1.

Peach will not run with version 4.4 of the Mono runtime as there are known handle leaks which can cause Peach to run out of memory during long fuzzing runs. If you have Mono 4.4 installed, you can either upgrade to 4.6+ or downgrade to 4.2.

2.5. macOS

To install on macOS, follow the installation steps provided below. Installation will require installing the Mono .NET runtime, then Peach. To enable support for the `CrashWrangler` monitor, install CrashWrangler and Xcode. Note that installing CrashWrangler is optional; it is only needed when running the target locally.

Peach Fuzzer, LLC, recommends using Mono .NET runtime version 4.8.1 from the mono project.



Peach will not run with Mono version 5.0 or newer due to incompatibilities with our Python runtime. If you have Mono 5.0 installed, you must downgrade to 4.8.1.

Peach will not run with version 4.4 of the Mono runtime as there are known handle leaks which can cause Peach to run out of memory during long fuzzing runs. If you have Mono 4.4 installed, you can either upgrade to 4.6+ or downgrade to 4.2.

1. Install the [Mono package](#).
2. Unzip the Peach distribution to an appropriate folder. The file is a zip file with the extension [.zip](#). Use the filename that begins with `peach-pro` and contains the appropriate architecture for your system, such as `peach-pro-0.0.0-osx_release.zip`.
3. Install CrashWrangler.

CrashWrangler **MUST** be compiled on each macOS machine. Peach includes the CrashWrangler source files in the peach distribution. Here are instructions to install and compile CrashWrangler from the peach zip.

- a. Ensure XCode is installed.
- b. Open [Terminal.app](#).
- c. Navigate to the folder where you extracted `peach-pro-0.0.0-osx_release.zip`.
- d. Finish installing CrashWrangler using the following commands.

```
# Navigate to the folder containing CrashWrangler distribution
cd CrashWrangler

# Extract CrashWrangler sources
unzip 52607_crashwrangler.zip

# Navigate to the folder containing the extracted CrashWrangler sources
cd crashwrangler

# Compile CrashWrangler
$ make

# Ensure installation directory exists
sudo mkdir -p /usr/local/bin

# Install CrashWrangler
sudo cp exc_handler /usr/local/bin

# Navigate to the folder containing peach
cd ../../

# Verify CrashWrangler can run
exc_handler
```

2.6. License Activation

The list below contains links to the steps for activating the different types of Peach Fuzzer licenses.

Usage Based (Online Synchronization)

The most common method for activating a usage based license. A Cloud License Server will be automatically provisioned and managed for you. However, Peach Fuzzer will require a persistent connection to the Internet.

Usage Based (Offline Synchronization)

Users who wish to use Peach Fuzzer in an offline without having access to the Internet can deploy a Local License Server onsite to provide offline activation and synchronization of licensing information.

Node Locked

The license is tied to the physical machine running Peach Fuzzer. No license server is required and internet connectivity is only needed for activation.

Enterprise

No activation is required for enterprise customers.

2.6.1. Usage Based (Online Synchronization)

A Cloud License Server provides functionality for serving and monitoring a counted pool of licenses for Peach Fuzzer. A persistent connection to the Internet is required so that usage data can be uploaded to the Cloud License Server while a Peach Fuzzer job is running. Peach Fuzzer will automatically activate the first time it is run.



If a proxy server is required to connect to the Internet, it must be configured as described below.

Windows Proxy configuration

On Windows, the system proxy setting is the correct way to configure the proxy peach will use to connect to the licensing server.

Windows 10

The system proxy settings are configured at Settings > Network & Internet > Proxy. From there you will be able to enter the IP and Port of the proxy server.

Windows 8

The system proxy settings are configured at PC Settings > Network Proxy. From there you will be able to enter the IP and Port of the proxy server.

Windows 7

The system proxy settings are configured through the Internet Settings dialog. Open the Internet Options window located at Control Panel > Network and Internet > Internet Options.

1. Click the "Connections" tab at the top of the Internet Options window.
2. Click the "LAN Settings" button at the bottom of the window.
3. Click the "Advanced" button under Proxy Server will allow you to change advanced settings and enable a manual proxy server.

Linux Proxy Configuration

The proxy configuration on Linux is controlled via two environment variables `http_proxy` and `https_proxy`. Ensure both variables are set prior to starting Peach Fuzzer.

```
export http_proxy=http://xxxxx  
export https_proxy=https://xxxxx
```

2.6.2. Usage Based (Offline Synchronization)

The Local License Server provides functionality for serving and monitoring a counted pool of licenses for Peach Fuzzer. Users who wish to use Peach Fuzzer without having access to the Internet can deploy

a Local License Server onsite to provide offline activation and synchronization of licensing information.

The instructions for installing and activating a Local License Server can be found on the [Peach Portal](#) by navigating to the "Licensing" tab and clicking the "Local License Server" button for the desired license.

The screenshot shows a screenshot of a web browser displaying the [Peach Portal](#). The navigation bar includes links for Solutions, Forums, Tickets, Downloads, **Licensing**, and Logout. The main content area shows a license entry for "Peach Fuzzer - Premium". It includes fields for Activation ID (8264-92dc-cde5-48c7-9f3a-db62-bb15-fad3), Expiration Date (2/1/2021), License Type (Local License Server), and Products (listing Peach Fuzzer - Premium, PeachPitPack-Premium, and Peach Fuzzer - Test Cases (Limited)). At the bottom of the card, there are two buttons: "View details »" and "Local License Server Instructions", with the second button being highlighted with a red box. The footer of the page contains the copyright notice "Copyright © 2019 Peach Fuzzer, LLC . All rights reserved."

2.6.3. Node Locked

No license server is required, as node locked licenses are tied to an individual machine. Peach Fuzzer will automatically activate the first time it is run.

If your license has changed and you want Peach Fuzzer refresh its license, run the following command:

```
peach --activate
```

If you wish to move your license to a new machine, you must first deactivate the existing instance by running the following command:

```
peach --deactivate
```



Peach Fuzzer requires an internet connection in order to perform activation and deactivation. Once activated, no further internet connectivity is required.

2.6.4. Enterprise

No activation is required for enterprise customers. The enterprise license is automatically embedded in the Peach Fuzzer download.

2.7. Enabling HTTPS And Authentication

Peach Fuzzer Professional uses a web interface for configuration and control of the fuzzing engine. By default, the web interface is accessible with no encryption (SSL/TLS) and no authentication. If the use of HTTPS or authentication is required, a reverse proxy (apache/nginx/traefik) can be used to provide both SSL/TLS and authentication.

2.7.1. Reverse Proxy with NGINX

The following steps will configure NGINX as a reverse proxy for Peach adding TLS and authentication:

1. Install nginx using your Linux package manager
2. Create required key. For self signed keys this [online self-signed certificate generator](#) can be used.
3. Install the included NGINX configuration file to /etc/nginx/sites-available/peach
4. Install certificate and key and update configuration file if needed
5. Create .htpasswd with username/passwords replacing **USERNAME** with your username

```
sudo sh -c "echo -n 'USERNAME:' >> /etc/nginx/.htpasswd"
sudo sh -c "openssl passwd -apr1 >> /etc/nginx/.htpasswd"
```

6. Add a firewall rule to block external access to Peach's port 8888. Make sure this rule is enabled on bootup.
7. Link /etc/nginx/sites-available/peach to /etc/nginx/sites-enabled/peach
8. Restart NGINX and verify configuration is working

NGINX Configuration File

```
# HTTPS server
#
server {
    listen 443;
    server_name localhost;

    root html;
    index index.html index.htm;

    ssl on;
    ssl_certificate /etc/ssl/certs/ssl.crt;
    ssl_certificate_key /etc/ssl/private/ssl.key;

    ssl_session_timeout 5m;

    ssl_protocols TLSv1.1 TLSv1.2;
    ssl_ciphers "HIGH:!aNULL:!MD5 or HIGH:!aNULL:!MD5:!3DES";
    ssl_prefer_server_ciphers on;

    location / {
        # First attempt to serve request as file, then
        # as directory, then fall back to displaying a 404.
        #try_files $uri $uri/ =404;
        # Uncomment to enable naxsi on this location
        # include /etc/nginx/naxsi.rules

        auth_basic "Restricted";
        auth_basic_user_file /etc/nginx/.htpasswd;
        proxy_pass http://127.0.0.1:8888/;
    }
}
```

3. What's new in Peach Fuzzer Professional v0.0

This section provides a high-level view of the changes introduced this release of Peach Fuzzer Professional.

4. Introduction to Fuzzing

Welcome to the world of fuzzing!

What is fuzzing?

Fuzzing is the art of performing unexpected actions that result in target misbehavior.

What is our goal when we fuzz?

Our goal is to find new, previously unknown security vulnerabilities. Finding vulnerabilities is limited only by our ability to detect them automatically.

The most common target for fuzzing is a data consumer such a network service or a web browser. For a data consumer, the unexpected actions consist of sending data to the consumer that is malformed, fuzzed, in some way. Consuming fuzzed data can trigger vulnerabilities in the target. The likelihood that a single change causes a vulnerability might not be high, but when the act of sending fuzzed data repeats over 100,000 times, a very good chance exists that the target will misbehave in some interesting manner.

Fuzzers submit thousands, hundreds of thousands, and even millions of data samples malformed in some way; fuzzers call the malformed data mutations. Once written, fuzzers typically run for long time periods finding more and more vulnerabilities, called faults.

Consider the following C structure and corresponding data shown in hex:

```
struct Header
{
    ushort ver;
    ushort len; // 33
    char* data; // 33
}

char* msg_data = malloc(len);
strcpy(msg_data, data);
```

59	4D	00	21	41	4F	41	42
44	B1	45	55	56	0F	43	44
45	50	2D	4A	38	39	C0	80
62	68	5F	70	65	61	63	68
5F	66	75	7A	00	32	C0	80
35	39	C0	80	59	09	76	3D
31	26	6E	3D	64	31	74	38
75	69	70	38	6B	70	64	6C
6F	26	6C	3D	31	37	5F	66
34	30	32	37	5F	00	6B	70
70	73	2F	6F	26	70	3D	6D
32	54	63	78	4D	44	55	2D

The data consumer or test target parses data into the structure. The data and structure are color coded

to show which parts of the data are loaded into each member of the structure *Header*. Following the structure, two lines of code occur that use the *Header* structure. Looking at that code, we can identify several issues that could likely lead to crashing our program that we could trigger by changing our input data.

The first thing we could do is change the *len* in our data from 0x0021 to 0x0001. This would cause the memory allocation to return a smaller amount of memory than what is needed for copying data using the *strcpy* function. Once *strcpy* executes, we will have written past the end of *msg_data*, possibly causing the target process to crash.

```
struct Header
{
    ushort ver;
    ushort len; // 1
    char* data; // 33
}

char* msg_data = malloc(len);
strcpy(msg_data, data);
```

59	4D	00	01	41	4F	41	42
44	B1	45	55	56	0F	43	44
45	50	2D	4A	38	39	C0	80
62	68	5F	70	65	61	63	68
5F	66	75	7A	00	32	C0	80
35	39	C0	80	59	09	76	3D
31	26	6E	3D	64	31	74	38
75	69	70	38	6B	70	64	6C
6F	26	6C	3D	31	37	5F	66
34	30	32	37	5F	00	6B	70
70	73	2F	6F	26	70	3D	6D
32	54	63	78	4D	44	55	2D

Another change we could make to the data is to remove or change the null byte that terminates the string *data*. In C, string functions operate on data up to the null character. If the null byte is changed, the *strcpy* function copies more than 33 bytes into *msg_data*. This could cause the target process to crash.

```

struct Header
{
    ushort ver;
    ushort len; // 33
    char* data; // 74
}

char* msg_data = malloc(len);
strcpy(msg_data, data);

```

59	4D	00	21	41	4F	41	42
44	B1	45	55	56	0F	43	44
45	50	2D	4A	38	39	C0	80
62	68	5F	70	65	61	63	68
5F	66	75	7A	55	32	C0	80
35	39	C0	80	59	09	76	3D
31	26	6E	3D	64	31	74	38
75	69	70	38	6B	70	64	6C
6F	26	6C	3D	31	37	5F	66
34	30	32	37	5F	00	6B	70
70	73	2F	6F	26	70	3D	6D
32	54	63	78	4D	44	55	2D

The two changes we made to trigger two similar but different bugs in the code is something we look to automate with fuzzing. Taking this input data and randomly changing a byte to a random value, would, over the course of many attempts, eventually find both of these issues.

One of the goals with fuzzing is to find as many faults in targets with the least amount of human time.

4.1. Dumb Fuzzing

Several types of fuzzing technology exist; one of the most common technologies is dumb fuzzing. Dumb fuzzers are a class of fuzzers that operate with the following characteristics:

- Limited or no information or understanding of the target.
- Limited or no understanding of the data they provide the target.

Dumb fuzzers are popular because they take very little effort to get running and can produce good results with certain targets. Since they lack any knowledge about the data that the target is consuming, they have limited practical use.

The most common mutation that dumb fuzzers perform is bit flipping. Bit flipping selects a part of data to change and modifies it in a simple manner. One might make a single change, or multiple changes depending on the fuzzer. The resulting data is sent to the data consumer to see whether it causes the data consumer to misbehave—such as crash.

You can build a dumb fuzzer with Peach. For an example, see the [dumb file fuzzing tutorial](#).

Dumb fuzzing makes a great starting point when first fuzzing as it is the easiest method to use. However, in many cases, dumb fuzzing does not work. One example that prevents a dumb fuzzer from

working is the use of checksums in the data format. Checksums validate integrity of the data during transmission or storage. Any change to the data will result in a checksum that does not match, as shown in the following image:

CRC-32:

98 1B A7 75



New CRC-32:
DA 38 80 19

59	4D	53	47	00	0F	00	00
02	B1	00	55	00	00	00	00
00	50	2D	4A	38	39	C0	80
62	68	5F	FF	65	61	63	68
5F	66	75	7A	7A	32	C0	80
35	39	C0	80	59	09	76	3D
31	26	6E	3D	64	31	74	38
75	69	70	38	6B	70	64	6C
6F	26	6C	3D	31	37	5F	66
34	30	32	37	5F	35	6B	70
70	73	2F	6F	26	70	3D	6D
32	54	63	78	98	1B	A7	75

The highlighted FF in the data stream is a byte that was changed during fuzzing. The new checksum value does not match the one that already exists in the file.

If the data consumer validates the checksum, the validation failure will cause the data consumer to stop the test case. If the failure is early in the data consumer's processing cycle (such as upon receipt of the data), the fuzzed data would not deeply test the data consumer; in turn, this action would limit the number of faults that can be found in the test target.

Smart fuzzers allow updating the data to correct for things like checksums, encryption, encoding, and compression so that the data passes through the initial system checks and is processed by the data consumer.

Peach makes it easy to shift from dumb fuzzing into smart fuzzing.

4.2. Smart Fuzzing

Smart fuzzers are a class of fuzzers that operate with some knowledge and understanding of the target,

and of data that the target consumes. The amount of knowledge depends on the fuzzer used.

A typical smart fuzzer does the following things:

- Understand the data format used by the target application to consume incoming data.
- Monitor the target for fault conditions.
- Modify the data to gain better coverage or increase the ability to detect certain types of issues.

Data understanding includes:

- Type information (string, integer, byte array).
- Relationships between fields in the data (length, offset, count).
- Ability integrity fields such as a checksum or a CRC.

At this level, understanding the data structures and data types allows the fuzzer to make more-informed changes (mutations) to the data. Smart fuzzers use this understanding level to find more bugs.

Smart fuzzers can control and monitor the fuzzing environment. Environment and instrumentation controls start all the components of the system so they are ready to fuzz; on a faulting condition, they reset the environment to a known, good state. Smart fuzzers can detect a faulting condition and collect any interesting data in the system at the time of the fault (including output from a debugger, a network capture, or files on the file system), and log the data for later review. High-quality smart fuzzers can run unattended for long periods and capture enough information to allow a resource to reasonably reproduce and investigate the faults that occurred.

Smart fuzzers also perform bug bucketing and basic risk analysis. Fuzzing commonly finds the same issue multiple times during a long run. Bucketing is an industry term for associating similar (and possibly duplicate) issues together. Bucketing implementations can use a simple set of categories to group issues, or use a tiered set of categories. The Peach Fuzzer buckets issues using two tiers: major and minor levels.

- Minor differences between faults mean that the issues are basically the same, but worth reviewing to make sure.
- Major differences are generally distinct issues.

Along with buckets, initial risk analysis allows you to direct your attention first on higher risk faults before spending time on lower risk issues. Risk analysis is not always possible, but useful when it can be performed.

For an example of building a smart fuzzer with Peach, see the [smart file fuzzing tutorial](#).

4.3. When to Stop Fuzzing

A fuzzing bar sets requirements for a fuzzing job and answers the question, “How long do we fuzz?”

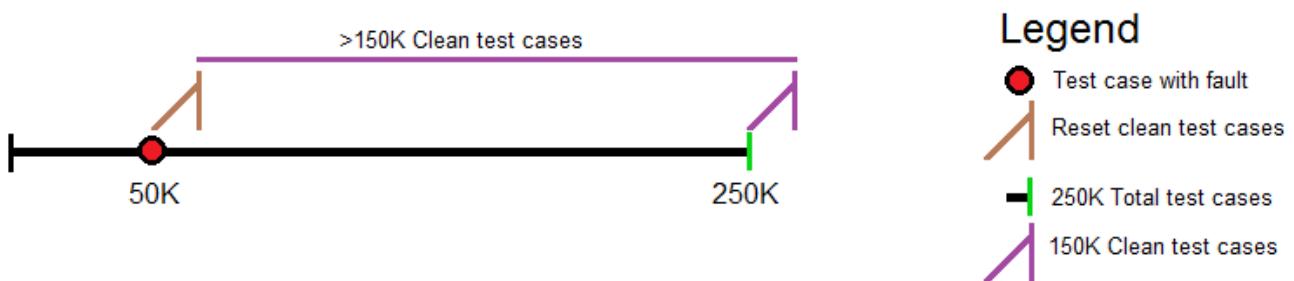
Here is one set of exiting criteria that consists of two requirements:

- The fuzzer must run a set number of test cases.
The number of test cases performed must meet or exceed the specified threshold.
- The fuzzer must have at least Y consecutive clean test cases.
A clean test case generates zero new faults.

As soon as both requirements occur, fuzzing can stop.

For example, when fuzzing a new product, the release criteria might be set to 250,000 total fuzzing test cases on the product and yield 150,000 consecutive clean test cases.

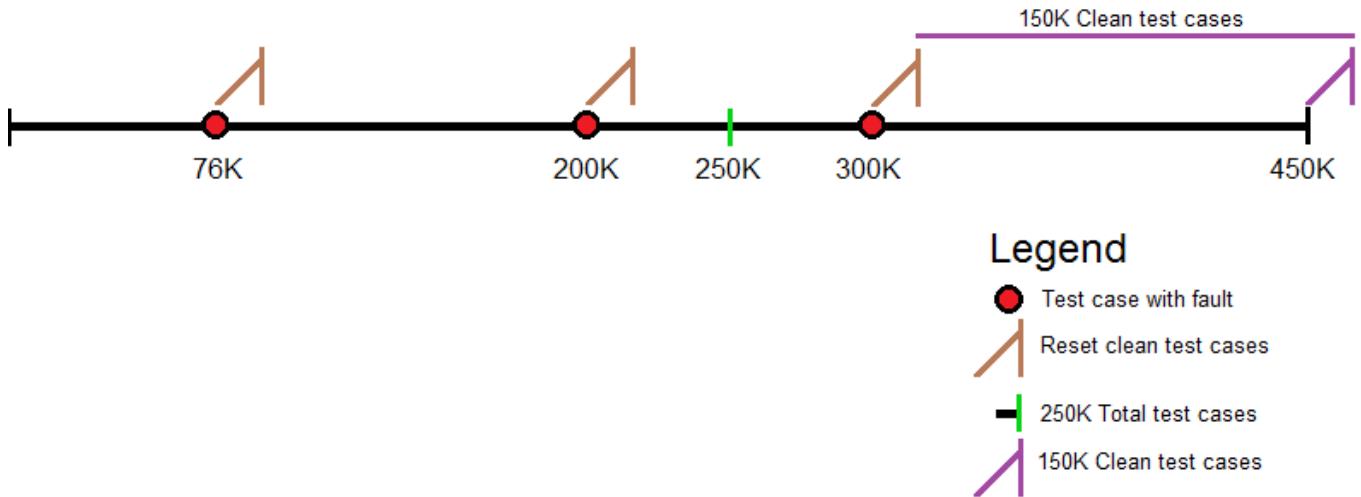
The following diagram shows one way of meeting this criteria and focuses on the number of test cases performed, test case failures, and the number of consecutive clean test cases.



A few items are worth noting:

- The total number of test cases performed is 250K.
- The number of consecutive clean test cases is 200K, surpassing the requirement of 150K clean test cases. The last 50K test cases were needed to meet the first requirement.
- The count of consecutive clean test cases reset to zero when a new fault was found at the 50,000th test case.

The next diagram shows another way of meeting this criteria.



Again, a few of items are worth noting:

- The total number of test cases performed is 450K.
- The number of consecutive clean test cases is 150K.
- The count of the consecutive clean test cases restarted three times due to new faults occurring in test cases: after 76K, 200K, and 300K.

The requirements used in the previous example are reasonable for a first release of a product. However, each successive version of a product should become more and more stable. This can be realized by increasing the total number of test cases performed by the fuzzer, and the number of consecutive clean test cases that result from fuzzing. The following table provides increasing requirements as a product matures.

Year	Required Iterations	Clean Iterations
Year 1	250,000	150,000
Year 2	500,000	250,000
Year 4	High 750,000 Medium 500,000 Low 500,000	High 500,000 Medium 250,000 Low 250,000
Year 5	High 1,000,000 Medium 750,000 Low 500,000	High 750,000 Medium 500,000 Low 250,000

What was once the Software Development Life Cycle (SDLC) has evolved into the Secure Development Lifecycle (SDL) to protect corporate assets from zero-day attacks. Fuzzing is part of the SDL, specifically in the security push and the Verification Phase. Peach Fuzzer can operate on non-executable file formats, protocol stacks, and data that originates from a lesser-privileged trust boundary.



5. Methodology

The following sections introduce a basic methodology for fuzzing. Our goal is to find previously unknown vulnerabilities with a high return on investment (ROI). We approach this by intelligently picking our targets and finding a balance between the amount of human power required to build and maintain our fuzzers versus the number of faults we will find. To accomplish this goal, we need to do the following:

- Select the correct targets
- Develop adequate fuzzers that achieve good coverage
- Understand when to fuzz
- Understand how long to fuzz

5.1. Selecting Fuzzing Targets

Selecting good fuzzing targets is critical to success. For teams just starting down the fuzzing road, it is important to get good early results to prove the program and to build confidence in the technology and process. Once the team becomes confident with the process and tools, they should chose more complex targets. When picking a fuzzing target, a number of factors play into the choice.

For each possible target, consider and evaluate the following things:

What is the risk of issues found in the target?

If issues are identified in this target, are they important to fix? Would they be seen as a high priority? Ideally, a target should be picked because the resulting issues would have a high diagnosis priority.

Does the development of the target pre-date a secure development lifecycle (SDL) being in place?

Targets developed prior to an SDL process are more likely to contain security issues. Additionally, this characteristic is true of applications that contain complex parsers and logic.

What types of issues (bugs) would we expect to find in this target?

The types of issues we are looking to identify should map well to fuzzing. The easiest issues to find are those that crash the target.

Does the fuzzer support tools that can be used out-of-the-box to identify issues in this target?

If Peach already has the tools to identify the issues intended to be found without any extensions, it will be easier and faster to create/configure our fuzzer. For Windows, Linux, and OS X, this could be debugger support. With embedded devices, support tools should be able to ping the device, or otherwise detect that the device has stopped working.

Does the data format contain cryptographic components—such as encryption or signatures, or custom complex authentication?

This question eliminates targets that would require custom extensions. Initially, we want to avoid the more complex and sophisticated parts of fuzzing. Formats that include a cryptographic component, but are not available as licensed Pits can be challenging to create.

When we select a fuzzing target, our goal is to find something that is not very difficult to fuzz, yet will provide a good payout based on the number or type of issues found.

5.2. Developing Fuzzers

Having an appropriate fuzzing model is crucial to achieving good results. Peach Fuzzer has many off-the-shelf fuzzing models (Pits) that eliminate development time. Individual Pits and groups of themed Pits are available from Peach Fuzzer, LLC.

If you have custom extensions or need to fuzz a proprietary item, such as a custom file format or protocol, you'll need to develop the custom portion of the fuzzing model. Peach Fuzzer includes the components needed to build a fuzzing definition, as well as documentation to extend Peach where custom components are needed.

For information on developing fuzzing definitions, see the following sections:

- [Pit Files](#)
- [Monitoring the Test Environment](#)
- [Test](#)
- [Debugging Pit Files](#)
- [Dumb File Fuzzing Tutorial](#)
- [File Fuzzing Tutorial](#)

For information on Peach components, see [Reference](#).

For information on building custom components, see [Extending Peach](#).

5.3. Secure Development Lifecycle

A robust secure development lifecycle (SDL) program should include requirements for fuzzing. Using the Microsoft SDL as an example, fuzzing typically integrates into the validation phase. After the first iteration with fuzzing is integrated, the project team should decide if fuzzing can be started earlier on the next revision of the deliverable. **Fuzzers are most effective when the target is reasonably stable and fully featured.** If the target is unstable or not usable, basic stability issues limit the ability to identify security issues. For example, if the data formats or features are in flux, the amount of human effort to maintain the fuzzers will increase. That, in turn, can lower the return on investment.

As a team gains experience with fuzzing, they become better equipped to identify the best time to start fuzzing. Typically, more mature products can fuzz earlier in the development cycle than newer products.

5.4. Deciding How Long to Fuzz

The best way to measure how long to fuzz is by counting iterations. An iteration is a single test performed by the fuzzer. The number of iterations required to fully test a target vary based on the data format complexity and quality of the target. In a perfect world, you would fuzz until you stop finding issues by hitting a plateau. In reality, projects have ship dates and limited resources. Setting a bar to reach for each cycle is a good method to use. For more information, see [When to Stop Fuzzing](#).

6. Introduction to Peach

Peach is a hybrid fuzzer, combining part of both mutation and generational fuzzing.

Fuzzing with Peach centers on the three M's of Peach: model, mutate, and monitor.

- Model - We model the data format and data/state flow.
- Mutate - The fuzzing engine uses the data and state models to mutate data intelligently, in preparation of sending the malformed data to the target.
- Monitor - We monitor our test target and the test environment to identify fault conditions and to collect information used to investigate and remediate the faults.

A Peach fuzzing definition consists of the data and state models, monitoring declarations, and specifics about the test environment. Even though fuzzing definitions included with Peach have all of these components, you still need to describe the monitoring details, based on your test target and operating system.

Peach specifically targets data consumers. The data could be a file format, a network protocol, or an argument to an RPC interface.

Peach has the following high-level concepts:

Pit

Fuzzing definitions in Peach are Pits. Peach Pits are XML files that contain all the information Peach needs to perform a fuzzing operation. This includes data models, state models, agent configurations (agents host the monitors), and test details.

Peach includes one or more licensed Pits purchased individually or in themed groups called Pit Packs. Peach offers Pits for more than 25 network protocols, and for some file formats. To fuzz with a Pit, you can run the Peach Web interface to configure the Pit, and then immediately use it.

If your needs extend beyond the available Pits offered by Peach, you can create a fuzzing definition for a custom file format, a proprietary network protocol, an embedded device, a kernel driver, and more.

Modeling

Peach operates by fuzzing data and state models of the test targets. Because security testing (fuzzing) focuses on what happens when inappropriate data enters or appears in a system or when inappropriate state transitions occur, Peach focuses heavily on data modeling and state modeling.

For the average Peach user, most of the human time and energy now goes into analyzing fuzzing job results and fixing bugs. This is due to the availability of pre-made, off-the-shelf Pits, and the Peach Web UI.

In most cases, gone are the days of time-consuming activities, such as defining and solidifying the

data and state models of a fuzzing definition.

Publisher

Publishers are I/O interfaces. They take abstract concepts such as input, output, and call (used in state modeling) and provide the actual transport or implementation.

Peach includes a number of Publishers so your pits and test configurations can perform the following tasks:

- write to files,
- connect to other devices using TCP, UDP or other network protocols,
- make web requests, or
- call COM objects.

If you need a custom Publisher, it is one of the simplest Peach components to create.

Fuzzing Strategy

The fuzzing strategy is the logic that describes the test approach for a fuzzing session. The fuzzing strategy addresses some fundamental questions that impact a fuzzing session that include the following:

- Do we want Peach to test each element of the data and state models individually?
- Do we want Peach to test more than one element at a time?
- How many mutators do we want Peach to apply to each fuzzing element?
- Do we want to limit the number of iterations in the test session?

Peach includes several fuzzing strategies. They should be sufficient for the majority of users.

Mutators

Mutators produce altered or mutated data. Some mutators use a reference data value and create new values based on the reference value. Other mutators generate new values from scratch, from a formula, or from a random number generator.

Mutators tend to contain very simple logic and should perform a single type of mutation. Some examples follow:

- Produce a number that is within 50 units from the supplied reference value.
- Produce a string that has a length between 1 and 10,000.
- Produce 500 random numbers between 0 and sizeof(int32).

Agents

Agents are special Peach processes that can run locally or remotely, and host one or more Monitors or remote Publishers. Agents are the basis for the robust monitor facility provided by the Peach

Fuzzing Platform and allow for monitoring simple fuzzing configurations through very complex systems that have many tiers.

A Peach fuzzer can use zero or more agents.

Agents can be used for fault detection, data collection, and instrumentation of targets involved in the fuzzing run. They do not host fuzzing engines themselves, nor do they play a part in parallel or distributed fuzzing. Agents can also host Remote Publishers.

Agents can be written in any language. Peach Professional includes a built in agent and also basic implementations in several languages in the SDK.

Monitors

Monitors run within Peach Agent processes and perform utility tasks such as taking captures of network traffic during a fuzzing iteration, or attaching a debugger to a process to detect crashes, or even re-starting a network service if it crashes or stops.

Peach includes a number of monitors. Like Publishers, monitors are easy to write and integrate into the Peach environment.

Logger

A logging facility saves crashes and fuzzing run information.

Peach comes with a file system logger by default.

The following sections dive deeper into all of these concepts and more.

7. Iteration Types

Peach performs several different types of iterations during a fuzzing session. The following sections introduce each iteration type and describe when the iteration type is most effectively used.

7.1. Record Iteration

The record iteration performs a baseline test that a Peach Pit is operating correctly. The record iteration is the first iteration of a fuzzing session. Peach does not fuzz any data: no mutations. The expected outcome is that the state model executes with no errors.

Every time Peach starts, it performs a record iteration to record and evaluate the information it receives for this fuzzing session. Information captured from the record iteration will be used during later iterations. The captured information includes the following items:

- Applicable data models and data elements, to estimate the total iteration count required for [fuzzing iterations](#)
- Which mutators can act on which data elements
- Choices and arrays
- Data flows, to compare with the [control iterations](#)
- States and actions flows

[Control iterations](#), if configured, verify that the fuzzer executes the same set of actions as in the record iteration to determine whether the target is still operational.

When running Peach with one test, you will actually see two iterations performed—a record iteration and a single fuzzing iteration.

See also [control iterations](#) and [fuzzing iterations](#).

7.2. Control Iteration

Control iterations verify the correct target operation for a long running target (one that is operational across multiple test cases).

An example of a long running target is a network service that starts once at the beginning of the fuzzing session; Peach monitors the service to see if it faults. For targets that are not long running, such as a process that restarts for every test case, control iterations are normally not necessary.



Control iterations must be explicitly enabled in the pit file.

Control iterations work in conjunction with [record iterations](#). The state-model execution sequence is recorded during a record iteration. A control iteration will perform a test case with no mutations

occurring and compare the execution sequence with the recorded sequence. The two sequences should match. If not, Peach generates a fault.

For example, if 10 actions occur on the record iteration and only 6 occur on the control, Peach assumes that the target is in a bad state and generates a fault since the bad state is likely interesting (as a denial of service vulnerability) from a security standpoint. When a fault occurs, actions can be taken to restart the system.

Control iterations are configured on the [Test element](#) using the *controlIteration* attribute. The value provided is a positive integer that defines how often to perform a control iteration. A value of 1 will cause control iterations to be performed after each iteration.

1. Syntax example

```
<Test name="TheTest" controlIteration="1"> .... </Test>
```

A second attribute, *nonDeterministicActions*, allows Peach to continue fuzzing when the execution flow of a control iteration might not match the execution flow of the record iteration.

For example, some protocols that use a challenge-response authentication process negotiate the number of iterations needed to gain authentication. In this case, the sequence flow is indefinite and control iteration sequence flow should not be compared to that of the record iteration.

When the value of *nonDeterministicActions* is ‘false’, the default action occurs, in which Peach enforces a matching execution flow sequence between the record iteration and each control iteration. If the flows do not match, Peach issues a fault.

When the value of *nonDeterministicActions* is [true](#), the Peach skips the check of the execution flow sequence. If a control iteration state flow differs from the state flow of a record iteration, Peach does not produce a fault.

2. Syntax example

```
<Test name="TheTest" controlIteration="1" nonDeterministicActions="true"> .... </Test>
```

See also [control iterations](#) and [fuzzing iterations](#).

7.3. Fuzzing Iteration

Fuzzing iterations are the most common iteration type. The majority of iterations are usually fuzzing iterations.

A fuzzing iteration mutates the models to produce a fuzz test case to send to the target.

- During a fuzzing iteration, we assume the target will behave in unexpected ways, such as errors from our I/O interface (time outs, short writes or reads).

- After a fuzzing iteration, we probably will not be able to fully execute our state models or correctly crack the returned data. Because error conditions generated by fuzzing iterations are likely to happen (even if no issue has occurred), they are usually ignored.

An expected error seen during a fuzzing iteration will cause Peach, at most, to skip to the next iteration.

For targets that are long running (such as network services), configure the [control iterations](#) to validate such that the target can maintain correct operation.

See also [record iterations](#) and [control iterations](#).

8. Pit Files

Peach Pit files are XML files that contain all of the information needed for Peach to perform a fuzzing run. When starting a fuzzing session, Peach uses a Peach Pit file and, sometimes, an associated configuration file.

Peach provides several Pit files that, with some configuration settings, run out-of-the-box. If your needs extend beyond the Pit library, you can create a new Pit file from scratch or by modifying an existing Pit file.

Peach Pit files contain the following elements:

- [General Configuration elements](#)
- [Data Modeling elements](#)
- [State Modeling elements](#)
- [Agents and Monitors](#)
- [Test Configuration elements](#)

9. Modeling

Modeling describes the format and flow of the data to fuzz, whether in a file format, a network protocol, an embedded system, or a kernel driver. Accurate models are at the core of a successful Peach Fuzzer experience.

The two types of models used in Peach are the data model that describes the format of the data we fuzz and the state model that provides the flow of data to and from the target. A dumb fuzzer in Peach is a simplistic data model, while a smart (or fully fleshed out) Peach fuzzer has a full data model and state model for both the test data and the system under test.

When modeling with Peach, remember to model enough depth to enable the fuzzer to work but not more. Instead of worrying about business rules or constraints, focus on the information type, relationships between data elements, and updating items such as checksums.



Modeling is about data structure, not logic.

When creating custom Peach fuzzers, we recommend initially creating a dumb or simple version of the fuzzer and then extending it. This allows you to run the dumb version of the fuzzer as you develop the fuller, fleshed-out version. This stepping stone approach is also easier for novices to implement.

10. Data Modeling

Data models describe the data structures of the test target. A data model includes type information, basic relationships, and the ability to update elements (such as checksums). A data model should not include business rules or business-related constraints, such as those that appear in specifications.

The goal of building a data model data is to find a balance between effort and fuzzing ability. Spending much effort beyond the balancing point does not measurably increase bug detection in the target. This extra effort lowers the fuzzing return on investment. On the other hand, spending too little effort and not reaching the balancing point decreases the number of found bugs, again hurting the fuzzing return on investment.

Data models contain the following concepts:

Data elements

Data elements describe the intrinsic data types (such as number and string).

Since mutators are written to operate on data elements, providing the correct data element information is critical to fuzzing success. Incorrect data elements mean less effective mutations.

Relations

Peach models three basic types of relationships between data elements: size, count, and offset.

Data elements sometimes provide modeling information used by other element information like array counts or string length. Modeling relationships provides several benefits:

1. It allows the fuzzer to update the relationships during fuzzing.
2. If a string is lengthened, the associated length field can be updated to match. That way, the fuzzing data can penetrate deeper into the logic of the target application.
3. Mutators can mutate several fields together, providing much smarter data mutations.

Fixups

Fixups are special utility classes that calculate new checksums, CRC's and other operations that rely on data from another data element.

By fixing data features such as checksums, the values in the data model have a greater chance to pass initial validation checks and exercise the data consumer's core logic.

Transformers

In addition to fixups, Peach provides the ability to perform static transformations such as encoding/decoding and compression/decompression on data. Static transformations can be added and removed; the transformers can decode on input and encode on output.

Transformers are useful when a format contains data that is encoded or compressed before it reaches the fuzzer. The transformer can decode the data, apply the fuzzing elements to the data,

then encode the result so that the fuzzing target receives encoded data as it normally would.

Analyzers

In many cases (like structured data such as ASN.1 or XML), it is easier to write code to generate a data model than to manually define a data model. Analyzers let you write code helpers to generate or modify the data model.

Cracking occurs when loading a sample file or when performing on input actions (such as *input*, *getProperty*, and the result from a *call* action). We use Analyzers to crack data into a model.

Although Analyzers are moderately easy to add since they directly deal with the data model, implementing an analyzer requires greater knowledge of Peach internals than other data modeling components.

10.1. Data Elements

Data elements describe the intrinsic types that begin to model the data. Typically data are comprised of several data types (such as numbers and string). Providing the correct data elements is critical to the success of a fuzzer. Since mutators operate on specific types of data elements, if a data element is incorrect, less effective mutations will occur.

Common data elements:

DataModel

The *DataModel* element is a top-level element that defines a data model.

Multiple *DataModels* can exist in a pit file. *DataModels* by themselves have no size and are containers for other data elements.

Blob

A Blob is a container for unstructured data (think of a byte array). The term Blob is borrowed from the relational database field for a column that holds raw binary data.

Data held in Blobs will be dumb fuzzed.

Block

The Block element is a container for other data elements.

By themselves, Blocks do not have size nor contain other data. The main use of a Block is to group other data elements so they can be referenced or operated on as a unit. For example, if two data elements need to be included in a checksum calculation, they can be grouped in a block element. Then, by placing the fixup element as a child element in the block, both data elements will be included in the fixup calculation.

Choice

The choice element allows switch-like conditional statements in Peach data models. Choice is used

when modeling type-length-value (TLV) metaphors, or in other situations when the data format changes.

Flags

The Flags element defines a set of flags.

Flags is a container for Flag elements that define the individual flags. This can be a nice shortcut when dealing with a flag set that has many unused positions.

The *Number* element supports unaligned sizes and can also be used to define flags.

Flag

Defines a specific flag in a flag set. Flag elements have a bit position and bit length.

Number

Defines a binary integer.

Numbers are packed to bit/byte representation with byte order (endian-ness). For ASCII strings, use the String element.

Padding

Defines a padding for a block or data model.

Padding supports various options and is used when the padding size is variable (pad to 8-bit or byte boundary).

Stream

A Stream represents a group of data files that contain other files (like zip files) in a single model. Stream-aware publishers can use the stream metadata and content to combine all streams in a data model into a single file.

String

Defines a string type with encoding. Supported character encodings include ASCII and various Unicode formats.



You can add custom data elements. New data elements should be containers that expose existing types such as Blob, Number, and String. This approach allows your custom data element to use the existing mutators. If you add a custom data type that is not based on any of the existing types, you will need to add mutators that can fuzz the new data type. Otherwise, fuzzing for the new type won't be effective.

Example 1. UDP Packet Example

This example describes how to format a UDP packet.

This example does not provide any business logic or constraints on the data. It contains just enough information to describe the types and relations, and to update the checksum field. When fuzzing this data model, Peach will be able to keep the *Length* field in sync with the *Data* field changes, unless it specifically decides to mutate the *Length*. The same is true of the *CheckSum* field.

```
<DataModel name="UdpPacket">
    <Number name="SrcPort" size="16" endian="big" />
    <Number name="DestPort" size="16" endian="big" />
    <Number name="Length" size="16" endian="big">
        <Relation type="size" of="UdpPacket" />
    </Number>
    <Number name="CheckSum" size="16" endian="big">
        <Fixup class="UDPChecksumFixup">
            <Param name="ref" value="UdpPacket"/>
        </Fixup>
    </Number>

    <Blob name="Data" />
</DataModel>
```

10.1.1. Handling XML Documents

When fuzzing XML documents, you have a choice of fuzzing targets:

- The consumer of the data stored in the XML document.
- The XML parser.

The first target is the most common and is the focus of this section. The second target fuzzes the XML parser.

When targeting the data consumer, two special data modeling elements apply to the XML document to modify both the document structure and the document content.

XmlElement

Defines an XML element used to fuzz consumers of XML documents.

XmlAttribute

Defines an XML attribute and is used to fuzz consumers of XML documents. *XmlAttribute* is only valid as a child of *XmlElement*.



The [XML analyzer](#) automatically converts XML to data models. When fuzzing XML documents, use [XML analyzer](#) so you don't have to manually enter the document structure into Peach.



When using `XmlAttribute` or `XmlElement`, the XML parser is not targeted. The target is only the consumer of the data stored in the XML document.

10.2. Relationships in Data

Data formats usually have related values. If you understand the basic data relationships types, it will be easier for you to fuzz your data correctly.

The basic relationships supported in Peach are *size of* (sometimes called “length of”), *count of*, and *offset of*. These basic relationships often produce some of the most interesting mutations during fuzzing as they directly tie to parsing the target data. These basic relationships also form the smallest set of relationships required to correctly output, mutate, and crack most data formats.

When fuzzing relationships, Peach can either fuzz each part of the relationship separately, or fuzz them together. This can lead to more powerful and interesting mutations.

[size](#)

Size relations are the most common relationship found in data formats. Size models length and size fields. The relation is always placed on the size or length field and references the data element that needs a run-time size definition.

[count](#)

Count relations are the second most common relations found in data formats. Count relations model the array metaphor with a count specifier. Like size relations, the relationship element is always placed on the field that contains the count (the runtime value), not the array.

[offset](#)

Offset relations are the least common data format relationship. Offset relations are used in data formats where one element provides the offset to some data.

When offsets appear, they are usually arrays of offsets to arrays of data. Peach has a special way of handling this. See the [arrays of offsets to data](#) for more information.

10.2.1. Adjusting Relation Values

At some point, you will come across a format that will require a modification to the relation value. Perhaps the value requires a mask, or perhaps you need to double the value specified in the relation.

Relations in Peach allow you to associate Python expressions when reading and when setting the value of the relation. The attributes `expressionGet` and `expressionSet` perform this function. The relation value is exposed as `size` for a size relation, `count` for a count relation, or `offset` for an offset relation.

Example 2. Example of expressionGet and expressionSet

```
<Number name="HalfLength" size="32">
  <Relation type="size" of="Data" expressionGet="size * 2" expressionSet="size / 2"
  />
</Number>
<Blob name="Data" />
```

For more information about *expressionGet* and *expressionSet* with each relation type, see [Relation](#).

10.2.2. Arrays of Offsets to Data

...Or, the story of [Placement](#).

A common data pattern seen is arrays of offsets to data.

When cracking this type of data, Peach cracks both the offset relation and its corresponding data when parsing the array element. This creates a situation where the corresponding data will end up in the wrong place—not at the offset. Peach provides *Placement*, a child-element, to move the parent data element to the correct location post-cracking.

Consider the following data model:

```
<DataModel name="ArrayOfOffsets">

  <Block name="Item" maxOccurs="1024">

    <Number name="StringLength" size="32">
      <Relation type="size" of="Data" />
    </Number>

    <Number name="OffsetOfData" size="32">
      <Relation type="offset" of="Data"/>
    </Number>

    <String name="Data" />

  </Block>

</DataModel>
```

The structure of data formed when cracking data into the model looks like:

- ArrayOfOffsets

- Item-0
 - StringLength
 - OffsetOfData
 - Data
- Item-1
 - StringLength
 - OffsetOfData
 - Data
- Item-N
 - StringLength
 - OffsetOfData
 - Data

However, the data does not actually look like this. The format should instead look like the following layout, with the Data element located where the *OffsetOfData* specifies. In this second layout, the data directly follows the array of lengths and offsets.

- ArrayOfOffsets
- Item-0
 - StringLength
 - OffsetOfData
- Item-1
 - StringLength
 - OffsetOfData
- Item-N
 - StringLength
 - OffsetOfData
- Data-0
- Data-1
- Data-N

Peach can achieve this format using the *Placement* element. Here is the modified XML that produces the correct data model:

```

<DataModel name="ArrayOfOffsets">

  <Block name="Item" maxOccurs="1024">

    <Number name="StringLength" size="32">
      <Relation type="size" of="Data" />
    </Number>

    <Number name="OffsetOfData" size="32">
      <Relation type="offset" of="Data"/>
    </Number>

    <String name="Data">
      <Placement before="PlaceDataHere" />
    </String>

  </Block>

  <Block name="PlaceDataHere"/>

</DataModel>

```

The resulting data model looks like the following:

- ArrayOfOffsets
- Item-0
 - StringLength
 - OffsetOfData
- Item-1
 - StringLength
 - OffsetOfData
- Item-N
 - StringLength
 - OffsetOfData
- Data-0
- Data-1
- Data-N
- PlaceDataHere

The block named PlaceDataHere is a zero-length data element and does not produce any actual data. It

serves solely as a point to place our data.

10.3. Cracking Data into the Data Model

In Peach, cracking is the process of parsing data into a data model where data elements have initial values set. The two most common examples of cracking are: loading a sample file from disk using the [Data](#) element, and data received through a state model's [input action](#).

Cracking sample data into a model is a good way to verify that the model works correctly. Peach provides several features to support cracking. When building a Pit, Peach also provides methods to debug the cracking process.

10.3.1. Features Specific to Cracking

token attribute

A token is a string of one or more characters that are significant as a group. In Peach, one function of tokens is to identify delimiters that separate strings in an input stream. When cracking data into a model, the input stream must include elements marked as tokens. Strings that have unknown lengths, but are separated by tokens, can be cracked at the appropriate locations.

constraint attribute

Constraints are scripting expressions that are evaluated and return true or false. The constraint is executed when cracking data into a data element. Constraints provide similar functionality to tokens, but are more flexible. For example, a token attribute used on a [String](#) element would be case sensitive. A constraint can be used with a scripting comparison that is not case sensitive.

Placement element

Placement is used in conjunction with the [offset](#) relation to move an element inside of the model. See [the story of placement](#) for more information.

Analyzers element

Analyzers provide code that generate or modify a data model. When compared to generating a *DataModel* manually, analyzers usually reduce the time and effort needed to create a data model (especially in self-describing formats such as ASN.1 and XML).

--debug argument

The Peach **--debug** command line argument is the main method of debugging the cracking process. The debug argument causes Peach to provide detailed output for every step of the cracking process. See [Debugging the Cracking Process](#) for information on how to use this output.

10.3.2. Debugging the Cracking Process

When writing data models, it is useful to understand, in detail, how cracking occurs for both debugging and data validation. Peach provides two tools for debugging and validating that data is cracking correctly, the **--debug** argument to Peach and the [Peach Validator](#) tool.

- The `--debug` argument enables verbose debugging messages from Peach, including the cracking process.
- The Peach Validator tool provides a graphical view of the data model before and after data is cracked into the model.

The combination of these tools provides enough information to debug and to validate your pit.

Understanding the `--debug` Output

The following example explains the output of `--debug`, and the debugging process.

The focus of this example is the first part of a PNG Pit, and writing the debug output to the console. PNG is a common image file format. A full fuzzing definition of this format is available from Peach as a stand-alone Pit or as part of the Image Pit Pack.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

  <DataModel name="PngModel">
    <Blob />
  </DataModel>

  <StateModel name="State" initialState="Initial">
    <State name="Initial">

      <Action type="output">
        <DataModel ref="PngModel" />
        <Data fileName="##Peach.Pwd##\samples_png\snail.png" />
      </Action>

    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="State"/>

    <Publisher class="File">
      <Param name="FileName" value="fuzzed.png" />
    </Publisher>
  </Test>
</Peach>
```

When run with the `--debug` argument, it produces the following output:

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 28899.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Cracker.DataCracker -----①
Peach.Core.Cracker.DataCracker DataModel 'PngModel' Bytes: 0/33546, Bits: 0/268368 ②
Peach.Core.Cracker.DataCracker getSize: -----> DataModel 'PngModel'
Peach.Core.Cracker.DataCracker scan: DataModel 'PngModel'
Peach.Core.Cracker.DataCracker scan: Blob 'PngModel.DataElement_0' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: DataModel 'PngModel' Size: <null>, Bytes: 0/33546,
Bits: 0/268368
Peach.Core.Cracker.DataCracker -----③
Peach.Core.Cracker.DataCracker Blob 'PngModel.DataElement_0' Bytes: 0/33546, Bits:
0/268368 ④
Peach.Core.Cracker.DataCracker getSize: -----> Blob 'PngModel.DataElement_0'
Peach.Core.Cracker.DataCracker scan: Blob 'PngModel.DataElement_0' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker lookahead: Blob 'PngModel.DataElement_0'
Peach.Core.Cracker.DataCracker getSize: <----- Last Unsized: 268368
Peach.Core.Cracker.DataCracker Crack: Blob 'PngModel.DataElement_0' Size: 268368, Bytes:
0/33546, Bits: 0/268368 ⑤
Peach.Core.Dom.DataElement Blob 'PngModel.DataElement_0' value is: 89 50 4e 47 0d 0a 1a
0a 00 00 00 0d 49 48 44 52 00 00 01 00 00 00 01 00 08 02 00 00 00 d3 10 3f.. (Len: 33546
bytes) ⑥
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(33546 bytes) ⑦
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.FilePublisher stop()

[*] Test 'Default' finished.

```

① Debug messages are prefixed with the full class name they originate from. For cracking messages, you see a combination of DataCracker and DataElements.

The long dashed line indicates the start of a new data element

② The element type and element name are provided. This is the current data offset in bytes and in bits.

③ The long dashed line indicates the start of a new data element

- ④ Again, the element type and element name are provided. This is the current data offset in bytes and in bits. Since we have not yet read any bytes, we are still at offset zero.
- ⑤ The amount of data, in bits, to load into our element; and the current position, in bytes and in bits.
- ⑥ After cracking data into an element, the value is displayed. If the value is long, the display of the value might be truncated. Also, the total size in bytes is provided.
- ⑦ The total size sent as output to the Publisher. This should match the size of our input file.

As part of data validation, verify that the amount of data being sent and cracked matches the size of the file.

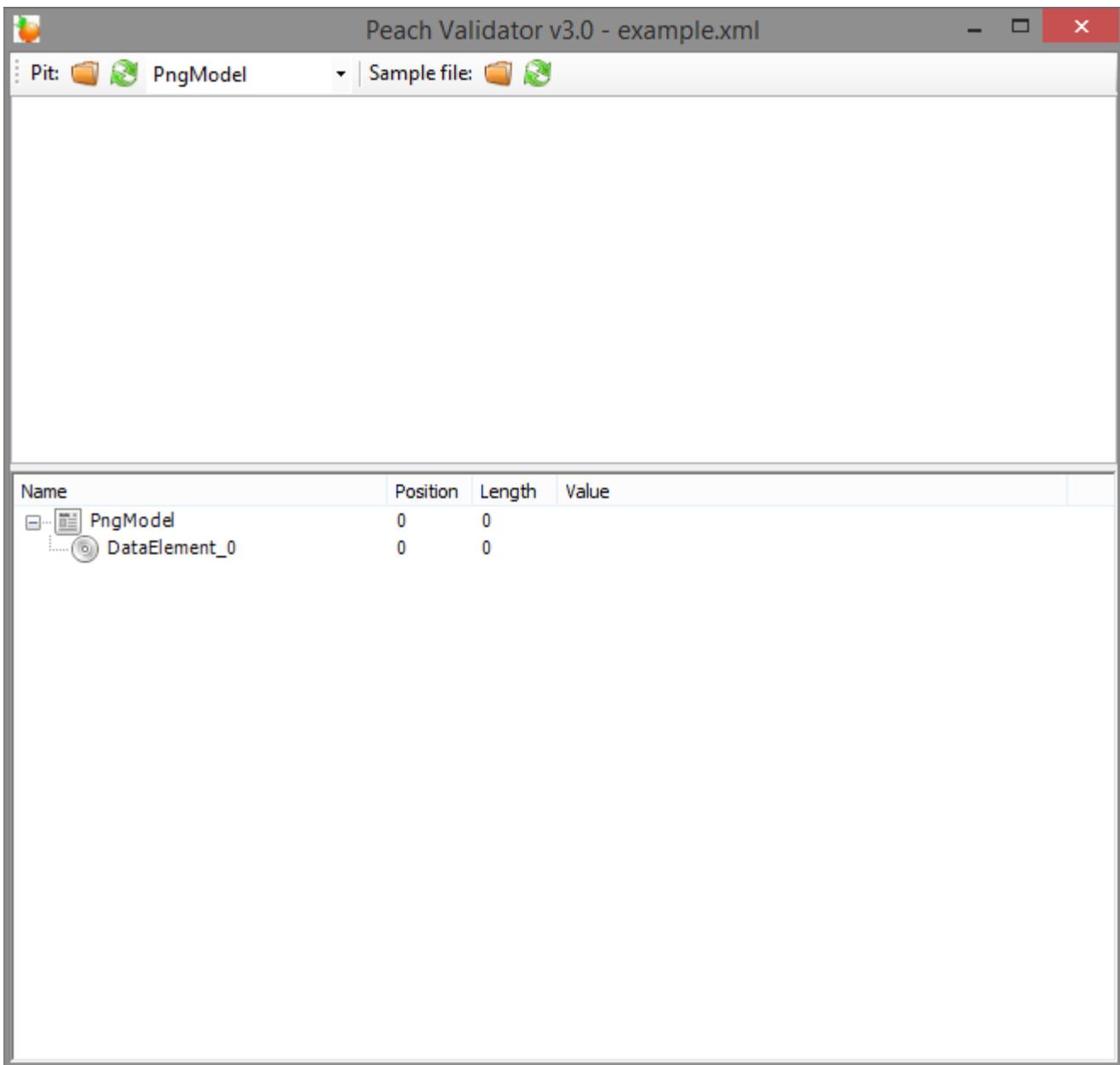
```
> dir samples_png\snail.png
Volume in drive C has no label.
Volume Serial Number is

Directory of samples_png

03/12/2014  07:00 PM           33,546 snail.png
               1 File(s)        33,546 bytes
```

In this case, the sizes match.

We can also use the Peach Validator to get a visual representation of what is happening. Launch Peach Validator and load *example.xml* via the toolbar. Notice the tree view on the bottom half of the window. This is the data model and data elements. Because no data has been loaded, the values are all empty. Notice the *DataElement_0* element; this is the unnamed *Blob* element. Peach assigned it a default name.



Next, load the sample file *snail.png* and see the result of the crack.

Peach Validator v3.0 - example.xml - snail.png

Pit: PngModel | Sample file:

000000000	89 50 4E 47 0D 0A 1A 0A 00 00 00 00 0D 49 48 44 52	.PNG.....IHDR
000000010	00 00 01 00 00 00 01 00 08 02 00 00 00 D3 10 3FÓ.?
000000020	31 00 00 80 00 49 44 41 54 78 DÀ EC 9D 65 7C 1E	1...IDATxÚí.e .
000000030	47 96 AF 9F EA EE 97 C5 CC 96 64 90 99 ED 98 13	G. .éí.Àì.d..i..
000000040	B3 1D 07 1C 87 39 0E 33 33 33 33 93 ED 24 B6	.,.9.33333.i§¶
000000050	93 38 31 C5 CC CC 96 64 49 16 B3 F4 4A 2F 43 77	.81ÀíÌ.di.ºJ/Cw
000000060	D7 FD E0 CC EE EC BD BB 73 27 BB 99 71 B2 E3 E7	*yàíii¤»s'».q*áç
000000070	9B E5 DF DB 5D 55 A7 FE 5D A7 4E 55 9D 12 52 4A	.åÙ]U\$þ]SNU..RJ
000000080	8E F1 2F 42 10 94 16 74 13 0B 18 E8 A6 2A 08 0B	.ñ/B...t..è!*..
000000090	61 51 34 50 AC 61 1F 75 87 C2 01 FF 56 47 BC AF	aQ4P-a.u.À.ÿVG¾
0000000A0	B0 67 40 68 93 64 38 C9 DD 6A EA 9D 75 16 E7 0E	°g@h.d8ÉÝjê.u.ç.
0000000B0	47 7A BC DD 91 6F CA 4C 45 28 E1 90 2D D4 EE 8B	Gz¤Ý.oÊLE(á.-Óí.
0000000C0	49 6F 37 65 A2 6A A2 B8 14 48 3D DA 75 FB 6F 22	Io7eo¡o.,.H=Úuûo"
0000000D0	8E 09 E0 5F 8E 00 38 9B 21 8D 30 68 6D 66 D0 EE	..à_.8!.0hmfDi

Name	Position	Length	Value
PngModel	0	33546	
DataElement_0	0	33546	89 50 4e 47 0d 0a 1a 0a 00 00 00 0d 49 48 44 52 00 00

The data model now contains data. Click on the *DataElement_0* item to highlight the corresponding data in the hex view.

The screenshot shows the Peach Validator interface. At the top, it says "Peach Validator v3.0 - example.xml - snail.png". Below the title bar, there are tabs for "Pit" (selected), "PngModel", and "Sample file".

The main area displays a hex dump of the file. The left column lists addresses from 00000000 to 000000D0. The right column shows the corresponding hex values. Above the hex dump, there is some ASCII representation of the file's content.

Name	Position	Length	Value
PngModel	0	33546	89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52 00 00 ...
DataElement_0	0	33546	89 50 4e 47 0d 0a 1a 0a 00 00 00 0d 49 48 44 52 00 00 ...

This definition is considered a dumb fuzzing model for PNG. The next step expands the model based on the PNG specification.

Adding Some Smarts

PNG files consist of a piece of file magic and then multiple T-L-V (type, length, value) blocks that the PNG specification calls *chunks*. TLV's are common structures in data formats. The file magic is a unique marker identifying PNG files. The next revision of the PNG model includes the file magic and a TLV structure set up as an array.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="PngModel">
        <Blob name="Magic" length="8" />

        <!-- TLV -->
        <Block name="Chunk" maxOccurs="100">
            <Number name="Length" size="32">
                <Relation type="size" of="Data" />
            </Number>
            <Block name="DataToCrc">
                <!-- Types: IHDR, IDAT, etc.-->
                <String name="Type" length="4" />
                <Blob name="Data" />
            </Block>
            <Number name="Checksum" size="32">
                <Fixup class="Crc">
                    <Param name="ref" value="DataToCrc" />
                </Fixup>
            </Number>
        </Block>
    </DataModel>

    <StateModel name="State" initialState="Initial">
        <State name="Initial">

            <Action type="output">
                <DataModel ref="PngModel" />
                <Data fileName="##Peach.Pwd##\samples_png\snail.png" />
            </Action>

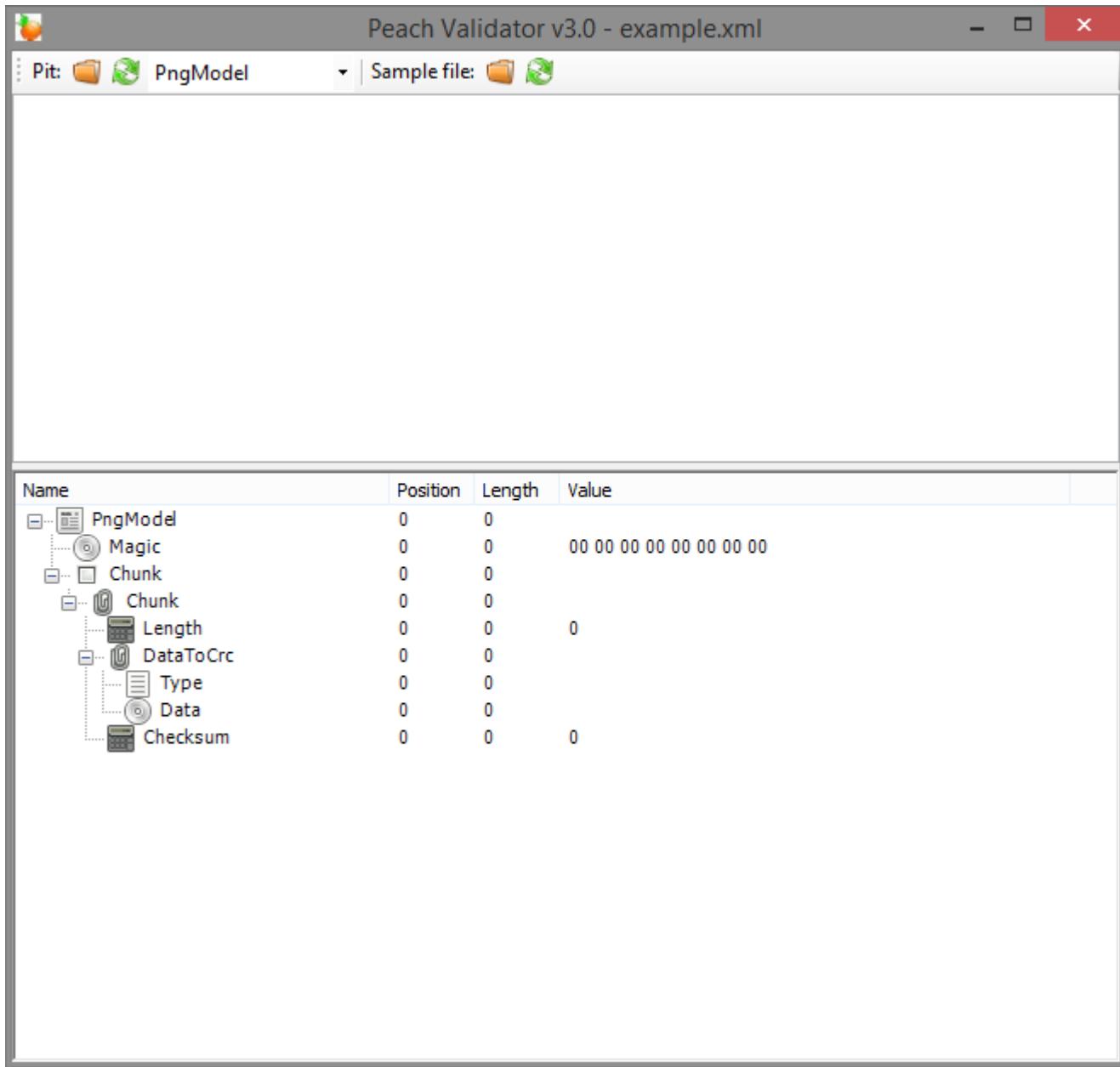
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="State"/>

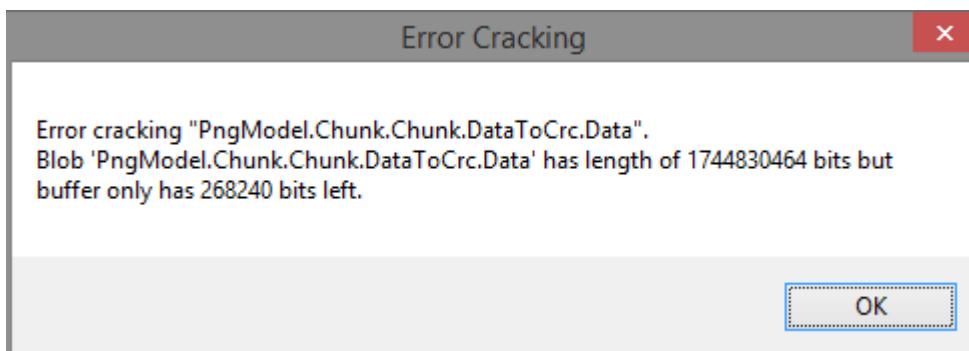
        <Publisher class="File">
            <Param name="FileName" value="fuzzed.png" />
        </Publisher>
    </Test>
</Peach>

```

The preceding Pit definition produces the following in Peach Validator before cracking any data.



When we try and crack the sample, an error message states that cracking failed. An error message also displays that might help in locating and resolving the issue.



The next step is to review the `--debug` output and see if we can spot the issue.

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 44055.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'PngModel' Bytes: 0/33546, Bits: 0/268368
Peach.Core.Cracker.getSize: -----> DataModel 'PngModel'
Peach.Core.Cracker.DataCracker scan: DataModel 'PngModel'
Peach.Core.Cracker.DataCracker scan: Blob 'PngModel.Magic' -> Pos: 64, Length: 64
Peach.Core.Cracker.getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: DataModel 'PngModel' Size: <null>, Bytes: 0/33546,
Bits: 0/268368
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Blob 'PngModel.Magic' Bytes: 0/33546, Bits: 0/268368
Peach.Core.Cracker.getSize: -----> Blob 'PngModel.Magic'
Peach.Core.Cracker.DataCracker scan: Blob 'PngModel.Magic' -> Pos: 64, Length: 64
Peach.Core.Cracker.getSize: <----- Size: 64
Peach.Core.Cracker.DataCracker Crack: Blob 'PngModel.Magic' Size: 64, Bytes: 0/33546,
Bits: 0/268368
Peach.Core.Dom.DataElement Blob 'PngModel.Magic' value is: 89 50 4e 47 0d 0a 1a 0a ①
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Array 'PngModel.Chunk' Bytes: 8/33546, Bits: 64/268368
Peach.Core.Cracker.getSize: -----> Array 'PngModel.Chunk'
Peach.Core.Cracker.DataCracker scanArray: Array 'PngModel.Chunk'
Peach.Core.Cracker.DataCracker scan: Block 'PngModel.Chunk.Chunk'
Peach.Core.Cracker.DataCracker scan: Number 'PngModel.Chunk.Chunk.Length' -> Pos: 32,
Length: 32
Peach.Core.Cracker.DataCracker scanArray: Array 'PngModel.Chunk' -> FirstSized
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: Array 'PngModel.Chunk' Size: <null>, Bytes:
8/33546, Bits: 64/268368
Peach.Core.Dom.Array Crack: ===== ②
Peach.Core.Dom.Array Crack: Block 'PngModel.Chunk.Chunk' Trying #1 ③
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Block 'PngModel.Chunk.Chunk' Bytes: 8/33546, Bits:
64/268368
Peach.Core.Cracker.DataCracker getSize: -----> Block 'PngModel.Chunk.Chunk'
Peach.Core.Cracker.DataCracker scan: Block 'PngModel.Chunk.Chunk'
Peach.Core.Cracker.DataCracker scan: Number 'PngModel.Chunk.Chunk.Length' -> Pos: 32,
Length: 32
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: Block 'PngModel.Chunk.Chunk' Size: <null>, Bytes:
8/33546, Bits: 64/268368
```

```
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Number 'PngModel.Chunk.Chunk.Length' Bytes: 8/33546, Bits: 64/268368
Peach.Core.Cracker.DataCracker getSize: -----> Number 'PngModel.Chunk.Chunk.Length'
Peach.Core.Cracker.DataCracker scan: Number 'PngModel.Chunk.Chunk.Length' -> Pos: 32, Length: 32
Peach.Core.Cracker.DataCracker getSize: <----- Size: 32
Peach.Core.Cracker.DataCracker Crack: Number 'PngModel.Chunk.Chunk.Length' Size: 32, Bytes: 8/33546, Bits: 64/268368
Peach.Core.Dom.DataElement Number 'PngModel.Chunk.Chunk.Length' value is: 218103808 ④
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Block 'PngModel.Chunk.Chunk.DataToCrc' Bytes: 12/33546, Bits: 96/268368
Peach.Core.Cracker.DataCracker getSize: -----> Block 'PngModel.Chunk.Chunk.DataToCrc'
Peach.Core.Cracker.DataCracker scan: Block 'PngModel.Chunk.Chunk.DataToCrc'
Peach.Core.Cracker.DataCracker scan: String 'PngModel.Chunk.Chunk.DataToCrc.Type' -> Pos: 32, Length: 32
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: Block 'PngModel.Chunk.Chunk.DataToCrc' Size: <null>, Bytes: 12/33546, Bits: 96/268368
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker String 'PngModel.Chunk.Chunk.DataToCrc.Type' Bytes: 12/33546, Bits: 96/268368
Peach.Core.Cracker.DataCracker getSize: -----> String 'PngModel.Chunk.Chunk.DataToCrc.Type'
Peach.Core.Cracker.DataCracker scan: String 'PngModel.Chunk.Chunk.DataToCrc.Type' -> Pos: 32, Length: 32
Peach.Core.Cracker.DataCracker getSize: <----- Size: 32
Peach.Core.Cracker.DataCracker Crack: String 'PngModel.Chunk.Chunk.DataToCrc.Type' Size: 32, Bytes: 12/33546, Bits: 96/268368
Peach.Core.Dom.DataElement String 'PngModel.Chunk.Chunk.DataToCrc.Type' value is: IHDR ⑤
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Blob 'PngModel.Chunk.Chunk.DataToCrc.Data' Bytes: 16/33546, Bits: 128/268368
Peach.Core.Cracker.DataCracker getSize: -----> Blob 'PngModel.Chunk.Chunk.DataToCrc.Data'
Peach.Core.Cracker.DataCracker scan: Blob 'PngModel.Chunk.Chunk.DataToCrc.Data' -> Pos: 1744830464, Size relation: 1744830464
Peach.Core.Cracker.DataCracker getSize: <----- Size: 1744830464
Peach.Core.Cracker.DataCracker Crack: Blob 'PngModel.Chunk.Chunk.DataToCrc.Data' Size: 1744830464, Bytes: 16/33546, Bits: 128/268368
Peach.Core.Cracker.DataCracker Blob 'PngModel.Chunk.Chunk.DataToCrc.Data' failed to crack. ⑥
Peach.Core.Cracker.DataCracker Blob 'PngModel.Chunk.Chunk.DataToCrc.Data' has length of 1744830464 bits but buffer only has 268240 bits left.
Peach.Core.Cracker.DataCracker Block 'PngModel.Chunk.Chunk.DataToCrc' failed to crack.
Peach.Core.Cracker.DataCracker Block 'PngModel.Chunk.Chunk' failed to crack.
Peach.Core.Dom.Array Crack: Array 'PngModel.Chunk' Failed on #1
Peach.Core.Cracker.DataCracker Array 'PngModel.Chunk' failed to crack.
```

```
Peach.Core.Cracker.DataCracker DataModel 'PngModel' failed to crack.
```

```
[*] Test 'Default' finished.  
Peach.Core.PeachException: Error, failed to crack  
"c:\peach\win_x64_release\bin\samples_png\snail.png" into "PngModel":Blob  
'PngModel.Chunk.Chunk.DataToCrc.Data' has length of 1744830464 bits but buffer only has  
268240 bits left. ---> Peach.Core.Cracker.CrackingFailure: Blob  
'PngModel.Chunk.Chunk.DataToCrc.Data' has length of 1744830464 bits but buffer only has  
268240 bits left.
```

- ① *Magic* element cracked correctly.
- ② The double line separator indicates an array is being expanded to fit the incoming data.
- ③ The array position (starting with 1) is indicated along with the name of the element that makes up the array.
- ④ Notice the very large value cracked into the *Length* field. This is much larger than the file size.
- ⑤ The *Type* field correctly cracked with a value of *IHDR*.
- ⑥ The *Data* field failed to crack with a huge size (shown in bits) provided by the *Length* field.

The debug output shows that the elements are cracking at the correct byte offsets and only *Length* and *Data* appear to be incorrect. In fact, the *Length* field is the real issue here. A closer look at the PNG specification shows all numbers should be big endian byte order. By default Peach uses little endian. This can be adjusted using the *endian* attribute, or a *Defaults* element to change to default endian-ness.

The following XML has corrected the issue:

```
<?xml version="1.0" encoding="utf-8"?>  
<Peach xmlns="http://peachfuzzer.com/2012/Peach"  
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">  
  
    <Defaults>  
        <Number endian="big" />  
    </Defaults>  
  
    <DataModel name="PngModel">  
        <Blob name="Magic" length="8" />  
  
        <!-- TLV -->  
        <Block name="Chunk" maxOccurs="100">  
            <Number name="Length" size="32">  
                <Relation type="size" of="Data" />  
            </Number>  
            <Block name="DataToCrc">  
                <!-- Types: IHDR, IDAT, etc.-->  
                <String name="Type" length="4" />
```

```

        <Blob name="Data" />
    </Block>
    <Number name="Checksum" size="32">
        <Fixup class="Crc">
            <Param name="ref" value="DataToCrc" />
        </Fixup>
    </Number>
</Block>
</DataModel>

<StateModel name="State" initialState="Initial">
    <State name="Initial">

        <Action type="output">
            <DataModel ref="PngModel" />
            <Data fileName="##Peach.Pwd##\samples_png\snail.png" />
        </Action>

    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="State"/>

    <Publisher class="File">
        <Param name="FileName" value="fuzzed.png" />
    </Publisher>
</Test>
</Peach>

```

With this change, let's review the `--debug` output again.

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 59855.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'PngModel' Bytes: 0/33546, Bits: 0/268368
Peach.Core.Cracker.DataCracker getSize: -----> DataModel 'PngModel'
Peach.Core.Cracker.DataCracker scan: DataModel 'PngModel'
Peach.Core.Cracker.DataCracker scan: Blob 'PngModel.Magic' -> Pos: 64, Length: 64
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: DataModel 'PngModel' Size: <null>, Bytes: 0/33546,
Bits: 0/268368
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Blob 'PngModel.Magic' Bytes: 0/33546, Bits: 0/268368
Peach.Core.Cracker.DataCracker getSize: -----> Blob 'PngModel.Magic'
Peach.Core.Cracker.DataCracker scan: Blob 'PngModel.Magic' -> Pos: 64, Length: 64
Peach.Core.Cracker.DataCracker getSize: <----- Size: 64
Peach.Core.Cracker.DataCracker Crack: Blob 'PngModel.Magic' Size: 64, Bytes: 0/33546,
Bits: 0/268368
Peach.Core.Dom.DataElement Blob 'PngModel.Magic' value is: 89 50 4e 47 0d 0a 1a 0a
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Array 'PngModel.Chunk' Bytes: 8/33546, Bits: 64/268368
Peach.Core.Cracker.DataCracker getSize: -----> Array 'PngModel.Chunk'
Peach.Core.Cracker.DataCracker scanArray: Array 'PngModel.Chunk'
Peach.Core.Cracker.DataCracker scan: Block 'PngModel.Chunk.Chunk'
Peach.Core.Cracker.DataCracker scan: Number 'PngModel.Chunk.Chunk.Length' -> Pos: 32,
Length: 32
Peach.Core.Cracker.DataCracker scanArray: Array 'PngModel.Chunk' -> FirstSized
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: Array 'PngModel.Chunk' Size: <null>, Bytes:
8/33546, Bits: 64/268368

```

Next is the first element of the array that caused issues in the prior run.

```

Peach.Core.Dom.Array Crack: =====
Peach.Core.Dom.Array Crack: Block 'PngModel.Chunk.Chunk' Trying #1
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Block 'PngModel.Chunk.Chunk' Bytes: 8/33546, Bits:
64/268368
Peach.Core.Cracker.DataCracker getSize: -----> Block 'PngModel.Chunk.Chunk'
Peach.Core.Cracker.DataCracker scan: Block 'PngModel.Chunk.Chunk'
Peach.Core.Cracker.DataCracker scan: Number 'PngModel.Chunk.Chunk.Length' -> Pos: 32,
Length: 32
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???

```

```
Peach.Core.Cracker.DataCracker Crack: Block 'PngModel.Chunk.Chunk' Size: <null>, Bytes: 8/33546, Bits: 64/268368
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Number 'PngModel.Chunk.Chunk.Length' Bytes: 8/33546, Bits: 64/268368
Peach.Core.Cracker.DataCracker getSize: -----> Number 'PngModel.Chunk.Chunk.Length'
Peach.Core.Cracker.DataCracker scan: Number 'PngModel.Chunk.Chunk.Length' -> Pos: 32, Length: 32
Peach.Core.Cracker.DataCracker getSize: <----- Size: 32
Peach.Core.Cracker.DataCracker Crack: Number 'PngModel.Chunk.Chunk.Length' Size: 32, Bytes: 8/33546, Bits: 64/268368
Peach.Core.Dom.DataElement Number 'PngModel.Chunk.Chunk.Length' value is: 13 ①
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Block 'PngModel.Chunk.Chunk.DataToCrc' Bytes: 12/33546, Bits: 96/268368
Peach.Core.Cracker.DataCracker getSize: -----> Block 'PngModel.Chunk.Chunk.DataToCrc'
Peach.Core.Cracker.DataCracker scan: Block 'PngModel.Chunk.Chunk.DataToCrc'
Peach.Core.Cracker.DataCracker scan: String 'PngModel.Chunk.Chunk.DataToCrc.Type' -> Pos: 32, Length: 32
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: Block 'PngModel.Chunk.Chunk.DataToCrc' Size: <null>, Bytes: 12/33546, Bits: 96/268368
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker String 'PngModel.Chunk.Chunk.DataToCrc.Type' Bytes: 12/33546, Bits: 96/268368
Peach.Core.Cracker.DataCracker getSize: -----> String 'PngModel.Chunk.Chunk.DataToCrc.Type'
Peach.Core.Cracker.DataCracker scan: String 'PngModel.Chunk.Chunk.DataToCrc.Type' -> Pos: 32, Length: 32
Peach.Core.Cracker.DataCracker getSize: <----- Size: 32
Peach.Core.Cracker.DataCracker Crack: String 'PngModel.Chunk.Chunk.DataToCrc.Type' Size: 32, Bytes: 12/33546, Bits: 96/268368
Peach.Core.Dom.DataElement String 'PngModel.Chunk.Chunk.DataToCrc.Type' value is: IHDR
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Blob 'PngModel.Chunk.Chunk.DataToCrc.Data' Bytes: 16/33546, Bits: 128/268368
Peach.Core.Cracker.DataCracker getSize: -----> Blob 'PngModel.Chunk.Chunk.DataToCrc.Data'
Peach.Core.Cracker.DataCracker scan: Blob 'PngModel.Chunk.Chunk.DataToCrc.Data' -> Pos: 104, Size relation: 104
Peach.Core.Cracker.DataCracker getSize: <----- Size: 104
Peach.Core.Cracker.DataCracker Crack: Blob 'PngModel.Chunk.Chunk.DataToCrc.Data' Size: 104, Bytes: 16/33546, Bits: 128/268368
Peach.Core.Dom.DataElement Blob 'PngModel.Chunk.Chunk.DataToCrc.Data' value is: 00 00 01 00 00 00 01 00 08 02 00 00 00 ②
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Number 'PngModel.Chunk.Chunk.Checksum' Bytes: 29/33546, Bits: 232/268368
Peach.Core.Cracker.DataCracker getSize: -----> Number 'PngModel.Chunk.Chunk.Checksum'
```

```

Peach.Core.Cracker.DataCracker scan: Number 'PngModel.Chunk.Chunk.Checksum' -> Pos: 32,
Length: 32
Peach.Core.Cracker.DataCracker getSize: <----- Size: 32
Peach.Core.Cracker.DataCracker Crack: Number 'PngModel.Chunk.Chunk.Checksum' Size: 32,
Bytes: 29/33546, Bits: 232/268368
Peach.Core.Dom.DataElement Number 'PngModel.Chunk.Chunk.Checksum' value is: 3541057329

```

- ① The *Length* field looks correct with a value of 13.
- ② The *Data* field cracked successfully this time.

Now that the first chunk cracks correctly, it's time to expand the array to pick up the other chunks in the file as well.

```

Peach.Core.Dom.Array Crack: =====
Peach.Core.Dom.Array Crack: Block 'PngModel.Chunk.Chunk' Trying #2
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Block 'PngModel.Chunk.Chunk_1' Bytes: 33/33546, Bits:
264/268368
Peach.Core.Cracker.DataCracker getSize: -----> Block 'PngModel.Chunk.Chunk_1'
Peach.Core.Cracker.DataCracker scan: Block 'PngModel.Chunk.Chunk_1'
Peach.Core.Cracker.DataCracker scan: Number 'PngModel.Chunk.Chunk_1.Length' -> Pos: 32,
Length: 32
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: Block 'PngModel.Chunk.Chunk_1' Size: <null>, Bytes:
33/33546, Bits: 264/268368
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Number 'PngModel.Chunk.Chunk_1.Length' Bytes: 33/33546,
Bits: 264/268368
Peach.Core.Cracker.DataCracker getSize: -----> Number 'PngModel.Chunk.Chunk_1.Length'
Peach.Core.Cracker.DataCracker scan: Number 'PngModel.Chunk.Chunk_1.Length' -> Pos: 32,
Length: 32
Peach.Core.Cracker.DataCracker getSize: <----- Size: 32
Peach.Core.Cracker.DataCracker Crack: Number 'PngModel.Chunk.Chunk_1.Length' Size: 32,
Bytes: 33/33546, Bits: 264/268368
Peach.Core.Dom.DataElement Number 'PngModel.Chunk.Chunk_1.Length' value is: 32768 ①
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Block 'PngModel.Chunk.Chunk_1.DataToCrc' Bytes: 37/33546,
Bits: 296/268368
Peach.Core.Cracker.DataCracker getSize: -----> Block 'PngModel.Chunk.Chunk_1.DataToCrc'
Peach.Core.Cracker.DataCracker scan: Block 'PngModel.Chunk.Chunk_1.DataToCrc'
Peach.Core.Cracker.DataCracker scan: String 'PngModel.Chunk.Chunk_1.DataToCrc.Type' ->
Pos: 32, Length: 32
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: Block 'PngModel.Chunk.Chunk_1.DataToCrc' Size:
<null>, Bytes: 37/33546, Bits: 296/268368
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker String 'PngModel.Chunk.Chunk_1.DataToCrc.Type' Bytes:

```

```

37/33546, Bits: 296/268368
Peach.Core.Cracker.getSize: -----> String
'PngModel.Chunk.Chunk_1.DataToCrc.Type'
Peach.Core.Cracker.scan: String 'PngModel.Chunk.Chunk_1.DataToCrc.Type' ->
Pos: 32, Length: 32
Peach.Core.Cracker.getSize: <----- Size: 32
Peach.Core.Cracker.Crack: String 'PngModel.Chunk.Chunk_1.DataToCrc.Type'
Size: 32, Bytes: 37/33546, Bits: 296/268368
Peach.Core.Dom.DataElement String 'PngModel.Chunk.Chunk_1.DataToCrc.Type' value is: IDAT
②
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.Blob 'PngModel.Chunk.Chunk_1.DataToCrc.Data' Bytes:
41/33546, Bits: 328/268368
Peach.Core.Cracker.getSize: -----> Blob
'PngModel.Chunk.Chunk_1.DataToCrc.Data'
Peach.Core.Cracker.scan: Blob 'PngModel.Chunk.Chunk_1.DataToCrc.Data' -> Pos:
262144, Size relation: 262144
Peach.Core.Cracker.getSize: <----- Size: 262144
Peach.Core.Cracker.Crack: Blob 'PngModel.Chunk.Chunk_1.DataToCrc.Data' Size:
262144, Bytes: 41/33546, Bits: 328/268368
Peach.Core.Dom.DataElement Blob 'PngModel.Chunk.Chunk_1.DataToCrc.Data' value is: 78 da
ec 9d 65 7c 1e 47 96 af 9f ea ee 97 c5 cc 96 64 90 99 ed 98 13 b3 1d 07 1c 87 39 0e 33
33.. (Len: 32768 bytes) ③
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.Number 'PngModel.Chunk.Chunk_1.Checksum' Bytes:
32809/33546, Bits: 262472/268368
Peach.Core.Cracker.getSize: -----> Number 'PngModel.Chunk.Chunk_1.Checksum'
Peach.Core.Cracker.scan: Number 'PngModel.Chunk.Chunk_1.Checksum' -> Pos: 32,
Length: 32
Peach.Core.Cracker.getSize: <----- Size: 32
Peach.Core.Cracker.Crack: Number 'PngModel.Chunk.Chunk_1.Checksum' Size: 32,
Bytes: 32809/33546, Bits: 262472/268368
Peach.Core.Dom.DataElement Number 'PngModel.Chunk.Chunk_1.Checksum' value is: 4205918359

```

① The *Length* value is somewhat large, but is still smaller than the total file size. The type of the next chunk is **IDAT**. This chunk contains the main image data, so the model looks good.

② The *Type* field correctly cracks as **IDAT**.

③ The *Data* field correctly cracks correctly.

Now examine the third chunk.

```

Peach.Core.Dom.Array Crack: =====
Peach.Core.Dom.Array Crack: Block 'PngModel.Chunk.Chunk' Trying #3
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.Block 'PngModel.Chunk.Chunk_2' Bytes: 32813/33546, Bits:
262504/268368

```

```

Peach.Core.Cracker.DataCracker getSize: -----> Block 'PngModel.Chunk.Chunk_2'
Peach.Core.Cracker.DataCracker scan: Block 'PngModel.Chunk.Chunk_2'
Peach.Core.Cracker.DataCracker scan: Number 'PngModel.Chunk.Chunk_2.Length' -> Pos: 32,
Length: 32
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: Block 'PngModel.Chunk.Chunk_2' Size: <null>, Bytes:
32813/33546, Bits: 262504/268368
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Number 'PngModel.Chunk.Chunk_2.Length' Bytes: 32813/33546,
Bits: 262504/268368
Peach.Core.Cracker.DataCracker getSize: -----> Number 'PngModel.Chunk.Chunk_2.Length'
Peach.Core.Cracker.DataCracker scan: Number 'PngModel.Chunk.Chunk_2.Length' -> Pos: 32,
Length: 32
Peach.Core.Cracker.DataCracker getSize: <----- Size: 32
Peach.Core.Cracker.DataCracker Crack: Number 'PngModel.Chunk.Chunk_2.Length' Size: 32,
Bytes: 32813/33546, Bits: 262504/268368
Peach.Core.Dom.DataElement Number 'PngModel.Chunk.Chunk_2.Length' value is: 709 ①
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Block 'PngModel.Chunk.Chunk_2.DataToCrc' Bytes:
32817/33546, Bits: 262536/268368
Peach.Core.Cracker.DataCracker getSize: -----> Block 'PngModel.Chunk.Chunk_2.DataToCrc'
Peach.Core.Cracker.DataCracker scan: Block 'PngModel.Chunk.Chunk_2.DataToCrc'
Peach.Core.Cracker.DataCracker scan: String 'PngModel.Chunk.Chunk_2.DataToCrc.Type' ->
Pos: 32, Length: 32
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: Block 'PngModel.Chunk.Chunk_2.DataToCrc' Size:
<null>, Bytes: 32817/33546, Bits: 262536/268368
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker String 'PngModel.Chunk.Chunk_2.DataToCrc.Type' Bytes:
32817/33546, Bits: 262536/268368
Peach.Core.Cracker.DataCracker getSize: -----> String
'PngModel.Chunk.Chunk_2.DataToCrc.Type'
Peach.Core.Cracker.DataCracker scan: String 'PngModel.Chunk.Chunk_2.DataToCrc.Type' ->
Pos: 32, Length: 32
Peach.Core.Cracker.DataCracker getSize: <----- Size: 32
Peach.Core.Cracker.DataCracker Crack: String 'PngModel.Chunk.Chunk_2.DataToCrc.Type'
Size: 32, Bytes: 32817/33546, Bits: 262536/268368
Peach.Core.Dom.DataElement String 'PngModel.Chunk.Chunk_2.DataToCrc.Type' value is: IDAT
②
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Blob 'PngModel.Chunk.Chunk_2.DataToCrc.Data' Bytes:
32821/33546, Bits: 262568/268368
Peach.Core.Cracker.DataCracker getSize: -----> Blob
'PngModel.Chunk.Chunk_2.DataToCrc.Data'
Peach.Core.Cracker.DataCracker scan: Blob 'PngModel.Chunk.Chunk_2.DataToCrc.Data' -> Pos:
5672, Size relation: 5672
Peach.Core.Cracker.DataCracker getSize: <----- Size: 5672
Peach.Core.Cracker.DataCracker Crack: Blob 'PngModel.Chunk.Chunk_2.DataToCrc.Data' Size:

```

```

5672, Bytes: 32821/33546, Bits: 262568/268368
Peach.Core.Dom.DataElement Blob 'PngModel.Chunk.Chunk_2.DataToCrc.Data' value is: ad 2a
41 22 6d a6 b7 29 a7 ac 60 37 cb 5e d3 b4 c7 97 93 bd b3 7c 53 0d ec 6b a0 12 08 99 d9
9d.. (Len: 709 bytes) ③
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Number 'PngModel.Chunk.Chunk_2.Checksum' Bytes:
33530/33546, Bits: 268240/268368
Peach.Core.Cracker.DataCracker getSize: -----> Number 'PngModel.Chunk.Chunk_2.Checksum'
Peach.Core.Cracker.DataCracker scan: Number 'PngModel.Chunk.Chunk_2.Checksum' -> Pos: 32,
Length: 32
Peach.Core.Cracker.DataCracker getSize: <----- Size: 32
Peach.Core.Cracker.DataCracker Crack: Number 'PngModel.Chunk.Chunk_2.Checksum' Size: 32,
Bytes: 33530/33546, Bits: 268240/268368
Peach.Core.Dom.DataElement Number 'PngModel.Chunk.Chunk_2.Checksum' value is: 2357285555

```

- ① The length again looks good.
- ② The type is another *IDAT* field
- ③ The data also looks correct.

Now for the final chunk. The type should be *IEND* according to the specification.

```

Peach.Core.Dom.Array Crack: =====
Peach.Core.Dom.Array Crack: Block 'PngModel.Chunk.Chunk' Trying #4
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Block 'PngModel.Chunk.Chunk_3' Bytes: 33534/33546, Bits:
268272/268368
Peach.Core.Cracker.DataCracker getSize: -----> Block 'PngModel.Chunk.Chunk_3'
Peach.Core.Cracker.DataCracker scan: Block 'PngModel.Chunk.Chunk_3'
Peach.Core.Cracker.DataCracker scan: Number 'PngModel.Chunk.Chunk_3.Length' -> Pos: 32,
Length: 32
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: Block 'PngModel.Chunk.Chunk_3' Size: <null>, Bytes:
33534/33546, Bits: 268272/268368
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Number 'PngModel.Chunk.Chunk_3.Length' Bytes: 33534/33546,
Bits: 268272/268368
Peach.Core.Cracker.DataCracker getSize: -----> Number 'PngModel.Chunk.Chunk_3.Length'
Peach.Core.Cracker.DataCracker scan: Number 'PngModel.Chunk.Chunk_3.Length' -> Pos: 32,
Length: 32
Peach.Core.Cracker.DataCracker getSize: <----- Size: 32
Peach.Core.Cracker.DataCracker Crack: Number 'PngModel.Chunk.Chunk_3.Length' Size: 32,
Bytes: 33534/33546, Bits: 268272/268368
Peach.Core.Dom.DataElement Number 'PngModel.Chunk.Chunk_3.Length' value is: 0 ①
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Block 'PngModel.Chunk.Chunk_3.DataToCrc' Bytes:
33538/33546, Bits: 268304/268368

```

```

Peach.Core.Cracker.DataCracker getSize: ----> Block 'PngModel.Chunk.Chunk_3.DataToCrc'
Peach.Core.Cracker.DataCracker scan: Block 'PngModel.Chunk.Chunk_3.DataToCrc'
Peach.Core.Cracker.DataCracker scan: String 'PngModel.Chunk.Chunk_3.DataToCrc.Type' ->
Pos: 32, Length: 32
Peach.Core.Cracker.DataCracker getSize: <---- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: Block 'PngModel.Chunk.Chunk_3.DataToCrc' Size:
<null>, Bytes: 33538/33546, Bits: 268304/268368
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker String 'PngModel.Chunk.Chunk_3.DataToCrc.Type' Bytes:
33538/33546, Bits: 268304/268368
Peach.Core.Cracker.DataCracker getSize: ----> String
'PngModel.Chunk.Chunk_3.DataToCrc.Type'
Peach.Core.Cracker.DataCracker scan: String 'PngModel.Chunk.Chunk_3.DataToCrc.Type' ->
Pos: 32, Length: 32
Peach.Core.Cracker.DataCracker getSize: <---- Size: 32
Peach.Core.Cracker.DataCracker Crack: String 'PngModel.Chunk.Chunk_3.DataToCrc.Type'
Size: 32, Bytes: 33538/33546, Bits: 268304/268368
Peach.Core.Dom.DataElement String 'PngModel.Chunk.Chunk_3.DataToCrc.Type' value is: IEND
②
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Blob 'PngModel.Chunk.Chunk_3.DataToCrc.Data' Bytes:
33542/33546, Bits: 268336/268368
Peach.Core.Cracker.DataCracker getSize: ----> Blob
'PngModel.Chunk.Chunk_3.DataToCrc.Data'
Peach.Core.Cracker.DataCracker scan: Blob 'PngModel.Chunk.Chunk_3.DataToCrc.Data' -> Pos:
0, Size relation: 0
Peach.Core.Cracker.DataCracker getSize: <---- Size: 0
Peach.Core.Cracker.DataCracker Crack: Blob 'PngModel.Chunk.Chunk_3.DataToCrc.Data' Size:
0, Bytes: 33542/33546, Bits: 268336/268368
Peach.Core.Dom.DataElement Blob 'PngModel.Chunk.Chunk_3.DataToCrc.Data' value is:
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Number 'PngModel.Chunk.Chunk_3.Checksum' Bytes:
33542/33546, Bits: 268336/268368
Peach.Core.Cracker.DataCracker getSize: ----> Number 'PngModel.Chunk.Chunk_3.Checksum'
Peach.Core.Cracker.DataCracker scan: Number 'PngModel.Chunk.Chunk_3.Checksum' -> Pos: 32,
Length: 32
Peach.Core.Cracker.DataCracker getSize: <---- Size: 32
Peach.Core.Cracker.DataCracker Crack: Number 'PngModel.Chunk.Chunk_3.Checksum' Size: 32,
Bytes: 33542/33546, Bits: 268336/268368
Peach.Core.Dom.DataElement Number 'PngModel.Chunk.Chunk_3.Checksum' value is: 2923585666
Peach.Core.Dom.Array Crack: =====
Peach.Core.Dom.Array Crack: Block 'PngModel.Chunk.Chunk' Trying #5
Peach.Core.Dom.Array Crack: Consumed all bytes. Bytes: 33546/33546, Bits: 268368/268368
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(33546 bytes) ③

```

```

Peach.Core.Publisher.FilePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publisher stop()

[*] Test 'Default' finished.

```

- ① Length is zero. This is correct for the IEND chunk.
- ② Type is IEND.
- ③ Output length is the same size as the input file!

The output looks good and the model output the correct number of bytes. Time to review things inside the Peach Validator.

Peach Validator v3.0 - example.xml - snail.png

Pit:	File	Sample file:
	PngModel	

Name	Position	Length	Value
PngModel	0	33546	
Magic	0	8	89 50 4e 47 0d 0a 1a 0a
Chunk	8	33538	
Chunk	8	25	
Length	8	4	13
DataToCrc	12	17	
Type	12	4	IHDR
Data	16	13	00 00 01 00 00 00 01 00 08 02 00 00 00
Checksum	29	4	3541057329
Chunk_1	33	32780	
Length	33	4	32768
DataToCrc	37	32772	
Type	37	4	IDAT
Data	41	32768	78 da ec 9d 65 7c 1e 47 96 af 9f ea ee 97 c5 cc 96 64 90
Checksum	32809	4	4205918359
Chunk_2	32813	721	
Length	32813	4	709
DataToCrc	32817	713	
Type	32817	4	IDAT
Data	32821	709	ad 2a 41 22 6d a6 b7 29 a7 ac 60 37 cb 5e d3 b4 c7 97
Checksum	33530	4	2357285555
Chunk_3	33534	12	
Length	33534	4	0
DataToCrc	33538	4	
Type	33538	4	IEND
Data	33542	0	
Checksum	33542	4	2923585666

Reviewing the cracked model, it looks like everything is good.

Next Steps

To complete this fuzzing definition, the following would need to be done:

1. Finish flushing out the data models. For each chunk type, create a custom *Data* model.
2. Configure logging. For simplicity, this pit does not have logging configured.
3. Configure monitoring. Once a target is selected, monitoring is needed to detect faulting conditions.
4. Sample sets. Peach needs a large, non-redundant set of PNG images to use. Once the set or collection exists, run `minset` on the collection to pare it down to the optimal set of images to use for fuzzing.

11. State Modeling

Once the data model is complete, modeling shifts to describe the flow of that data to the target system. In Peach Fuzzer Professional this is called *state modeling*. State models can range from very simple to very complex.



Use the minimal amount of state model complexity needed for successful testing.

State models consist of three elements:

StateModel

Top level element that defines a complete state model. The *StateModel* element defines the entry state to start the execution.

State

A child of *StateModel* that defines a specific state that occurs. The *StateModel* element defines the entry, or *initial* state. Execution of a state model begins with the initial state. If the initial state does not trigger other states into occurring using the [changeState action](#), then the initial state also ends the execution.

Action

Action elements perform an action on a publisher (I/O adapter), agent/monitor, or data model. A State consists of one or more Action elements executed in order. Examples of actions include: sending output, receiving input, and changing to another state.

Scripting expressions can be used to indicate whether an action is performed by specifying the *when* attribute. Actions also support executing scripting expressions before an action begins and after an action completes.



Many fuzzing tasks can be accomplished using a single [State](#) with a series of [Actions](#).

Example 3. File State Model

This example shows a typical pattern for fuzzing files that are consumed by a target application. The state model produces a fuzzed output file that, in turn, loads into a target application running under a debugger monitor. The debugging monitor is used to trigger faults when the target program crashes.

This example is configured for Windows and requires an installed copy of Windows Debugger.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">
```

```

<!-- TODO: Define a full data model. -->
<DataModel name="DumbFuzzerModel">
    <Blob />
</DataModel>

<!-- Define a simple state machine that will write the file
     and then launch a program. -->
<StateModel name="State" initialState="Initial">
    <State name="Initial">

        <!-- Write out contents of file -->
        <Action type="output">
            <DataModel ref="DumbFuzzerModel" />

            <!-- Provide a sample set using files from the samples_png folder.
                 this folder is located in your Peach folder. The
                 ##Peach.Pwd## provides the path to the peach folder. -->
            <Data fileName="##Peach.Pwd##\samples_png" />
        </Action>

        <!-- We must close the file before launching our
             target process. -->
        <Action type="close" />

        <!-- Launch the target. This call will send a message to our agent and
             monitors. The WindowsDebugger monitor is configured to listen
             for this message and launch our target when received. Notice that
             the StartOnCall parameter for the monitor matches our method
             attribute. The publisher attribute of Peach.Agent tells Peach
             to send this call action out to all agents. -->
        <Action type="call" method="ScoobySnacks" publisher="Peach.Agent"/>

    </State>
</StateModel>

<!-- Setup a local agent that will monitor for faults -->
<Agent name="LocalAgent">
    <Monitor class="WindowsDebugger">

        <!-- The command line to run. Notice the filename provided matched up
             to what is provided below in the Publisher configuration -->
        <Param name="Executable" value="c:\windows\system32\mspaint.exe" />
        <Param name="Arguments" value="fuzzed.png" />

        <!-- If needed, uncomment this line and provide the path to windbg.exe
-->
        <!-- By default peach will try and locate the program in the default

```

```

locations -->
    <!--<Param name="WinDbgPath" value="C:\Program Files (x86)\Debugging
Tools for Windows (x86)" /-->

    <!-- This parameter will cause the debugger to wait for an action-call in
        the state model with a method="ScoobySnacks" before running
        program.
    -->
    <Param name="StartOnCall" value="ScoobySnacks" />

    </Monitor>
</Agent>

<Test name="Default">

    <Agent ref="LocalAgent" />
    <StateModel ref="State"/>

    <!-- Configure a publisher to write our file -->
    <Publisher class="File">
        <Param name="FileName" value="fuzzed.png" />
    </Publisher>

    </Test>

</Peach>

```

Run the following command line from the command-line interface. You should see mspaint.exe open and close over and over.

```
> peach example.xml
```

12. Providing Sample Data

To fuzz effectively with Peach, create a set of sample data to use as a fuzzing mutation base. Peach provides two ways to load sample data into a fuzzing session:

- Specify initial values using the [Data](#) element
- Specify initial values in data files

12.1. Specifying Initial Values Using the Data Element

You can provide sample data by specifying the information for each element, one at a time. This can be useful when no sample exists with the wanted values or for testing formats with a small number of fields, such as some network protocols. In this case, the [Data](#) element is used along with the *Field* child-element. See the [Data](#) reference section for examples.

12.2. Specifying Initial Values in Data Files

The most common way to provide sample data is via disk files. For example, when fuzzing a graphics format such as PNG you might use a set of 1,000 sample images. This method can also be used for network fuzzing by providing samples of each output as files and referencing them.

For targets that consume files, perform code coverage metrics to choose the minimum set of sample files needed for the best code coverage.

Research has shown that using a minimum set of samples increases the likelihood of finding new faults while fuzzing. Peach provides a tool called [minset](#) to help you create a minimum sample set.



The minset tool only works for targets that accept a filename from the command line.

The minset tool runs each sample file through the target application, collecting code coverage data for each file. It then selects a set of files that use or cover different parts of the code base. No duplicate samples are included in the resulting set of files.



Assume that only the features and code exercised by the provided sample data will be thoroughly fuzzed. While Peach may discover additional paths to code by mutating certain values, those paths will not gain the same level of test coverage compared to fuzzing robust sample data. To achieve the best test coverage, provide data samples that fully exercise all code paths reasonably possible.

13. Scripting in Peach

Peach provides a number of areas that expose scripting hooks. Python is the primary language used when scripting. Ruby is a second scripting language for Peach; however, support for Ruby is limited.

The following sections describe how scripting works in Peach and common script uses.

13.1. Python Scripting

Peach supports Python scripts with IronPython, the .NET framework implementation of Python 2.7. IronPython enables Peach to be flexible when creating fuzzers by allowing a small amount of code to be added to a pit when needed. The scripting engine within Peach has access to all DataModels, StateModels, and Tests.

13.2. In-line Expressions vs. Importing External Files

Scripting in Peach can be done either by using in-line expressions or importing a Python file into the Peach name space.

Python functions can be imported into a pit and used for in-line expressions in the StateModel.

3. Importing the Python module random

```
<Import import="random"/>

<StateModel name="TheState" initialState="initial">
    <State name="initial">
        <Action type="output" when="random.randrange(10) % 2 is 0">
            <DataModel ref="DataModel" />
        </Action>
    </State>
</StateModel>
```

External Python files can be included into a pit so that predefined functions may be called.

4. Importing a Local Python File

```
<PythonPath path="Path/To/PyFile"/>
<Import import="myPyFile" />

<StateModel name="TheState" initialState="initial">
    <State name="initial">
        <Action type="output" onStart="myPyFile.myAwesomeFunc(self)">
            <DataModel ref="DataModel" />
        </Action>
    </State>
</StateModel>
```

13.3. Importing External Files

When importing Python files into the Peach name space, define both the `PythonPath` and the `Import` elements. The `PythonPath` element defines where to find the Python files. The `Import` element specifies the file to import; the `.py` filename extension is not used in the declaration, as it is implied. Multiple files can be imported.

```
<PythonPath path="Path/To/PyFile"/>
<Import import="myPyFile" /> ①

<StateModel name="TheState" initialState="initial">
    <State name="initial">
        <Action type="output" onStart="myPyFile.myAwesomeFunc(self)"> ②
            <DataModel ref="DataModel" />
        </Action>
    </State>
</StateModel>
```

- ① Notice that the `.py` extension is missing from the `import` attribute even though the file is saved on disk as `myPyFile.py`.
- ② Calling a function from the `myPyFile` name space.

13.4. Scriptable Areas in Peach

When fuzzing, Peach can run a script for a state or for an action. Further, the script can run at the beginning or at the end of the action or state. The `onStart` and `onComplete` attributes specify when the script runs. Include the attribute in the `Action` or `State` definition to associate the script with the `action` or `state`. The most common script usages alter data or set a program into an expected state before or after fuzzing.

You can control when an action executes with the evaluation expression `when`. The `when` attribute

allows the action to execute when the given expression evaluates to true. This attribute is very useful when state changes depend on received data.

Other places evaluation expressions occur in relations:

- expressionSet/expressionGet attributes
- constraint attribute used for cracking
- ExpressionFixup
- valueType literals
- Godel state modeling

5. Example using onStart and onComplete

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

<Import import="time"/>

<DataModel name="Ex1">
  <String name="TransformMe" value="supersupersecret" >
    <Transformer class="Aes128">
      <Param name="Key" value="ae1234567890aeaffeda214354647586"/>

      <Param name="IV" value="aeaeaeaeaeaeaeaeaeaeaeaeae"/>
    </Transformer>
  </String>
</DataModel>

<StateModel name="TheState" initialState="initial">
  <State name="initial">
    <!-- Encrypted Output -->
    <Action type="output" publisher="ConsolePub" onStart="time.sleep(2)"> ①
      <DataModel ref="Ex1" />
    </Action>

    <!-- Write Encrypted Output to File -->
    <Action type="output" publisher="FilePubWrite" onComplete="time.sleep(2)"> ②
      <DataModel ref="Ex1" />
    </Action>

    <Action type="close" publisher="FilePubWrite" />

    <!-- Read and decrypt encrypted file and slurp output to console -->
    <Action type="input" publisher="FilePubRead" >
```

```

<DataModel name="InputModel" ref="Ex1" />
</Action>

<Action type="slurp" valueXpath="//InputModel//TransformMe" setXpath=
"//OutputModel//StringValue"/>

<Action type="output" publisher="ConsolePub">
  <DataModel name="OutputModel">
    <String name="StringValue" />
  </DataModel>
</Action>
</State>
</StateModel>

<Test name="Default">
  <StateModel ref="TheState"/>

  <Publisher class="ConsoleHex" name="ConsolePub"/>

  <Publisher class="File" name="FilePubWrite">
    <Param name="FileName" value="encrypted.bin" />
  </Publisher>

  <Publisher class="File" name="FilePubRead">
    <Param name="FileName" value="encrypted.bin" />
    <Param name="Overwrite" value="false" />
  </Publisher>
</Test>
</Peach>
```

- ① The expression is executed before the output action starts.
- ② The expression is executed after the output action completes.

13.5. Accessing Data

Scripts running in Peach have access to all DataModels, StateModels and Tests. Data is accessed by traversing the DOM to locate the target element.

6. SNMP Python File for Copying Incoming Data

```
#!/usr/bin/env python

import clr
clr.AddReferenceByPartialName('Peach.Core')

import Peach.Core ①

# Sets the type and value for all empty Value fields (The value for the Object Name
pair):
def set_variables(ctx): ②
    vars_block = ctx.parent.actions[0].dataModel.find('VariableBindings') ③
    output = ctx.dataModel.find('VariableBindings').parent
    if vars_block:
        #Targets the Value field(s) inside of the VariableBindings Value
        #x[0] == Type, x[1] == Size, x[2] == Value
        variable_bindings = vars_block[0][2][0][0][2]
        for x in variable_bindings:
            if x[2].DefaultValue.ToString() == "":
                x[0].DefaultValue = Peach.Core.Variant(0x06) ④
                x[2].DefaultValue = Peach.Core.Variant((0x2b,0x06, 0x01, 0x04, 0x01, 0x8f, 0x51,
0x01, 0x01, 0x01, 0x82, 0x29, 0x5d, 0x01, 0x1b, 0x02, 0x02, 0x01))
            output['VariableBindings'] = vars_block.Clone()
```

① The Peach name space is inserted into the file.

② The ctx variable starts at the action from where it was called.

③ Traversing the DOM to find the target Block element.

④ Setting the DefaultValue of an element to the Peach Variant type.

Data elements expose ways of accessing data associated with a data element.

DefaultValue

This property provides access to the default, unmutated value. The default value is produced when this element is not being fuzzed.

InternalValue

This property contains the value (fuzzed or default) to use when generating the final value. The field type is a variant of the internal typing, such as "int" or "string". The InternalValue property is used during scripting.

Value

This is the final, generated binary value. The Value property is the InternalValue packed into binary form. For a *Number* element of size 32, this is 4 contiguous bytes of data with correct endian-ness.

The data and state models in Peach are tree-object graphs with a parent-child relationship. Parents are accessible using the `_parent` property.

13.6. Returning Peach Types

When a script overwrites the `DefaultValue` for any element, the value type must be a Peach Variant. A Peach Variant accepts integers, strings, and arrays.

```
import clr
clr.AddReferenceByPartialName('Peach.Core')
import Peach.Core
import code

def some_func(ctx):
    code.InteractiveConsole(locals=locals()).interact()
    elementOne = ctx.parent.actions[0].dataModel.Find('TargetOne')
    elementOne.DefaultValue = Peach.Core.Variant("Hello")

    element = ctx.parent.actions[0].dataModel.Find('TargetTwo')
    element.DefaultValue = Peach.Core.Variant(31337)
```

13.7. Debugging

You can debug Python code loaded from a file by using the `InteractiveConsole` from the `code` module. When the function executes and reaches the `code.InteractiveConsole` line, control over the function transfers to the user until the function ends. This approach allows each line in the function to be manually entered to ensure the expected behavior is happening.

```
import clr
clr.AddReferenceByPartialName('Peach.Core')
import Peach.Core
import code

def some_func(ctx):
    code.InteractiveConsole(locals=locals()).interact()
    datamodel = ctx.parent.actions[0].dataModel
```

13.8. API Reference

The Peach distribution includes HTML API documentation. The elements in the XML map directly to the `Peach.Core.Dom` namespace. All public properties and methods are available to be called.

14. Monitoring The Fuzzing Environment

Monitoring the fuzzing environment to detect faults, collect interesting information and control the environment is a crucial step in building or configuring a Peach pit. [Agents](#) and [Monitors](#) provide the means to accomplish this.

- Agents are processes, local or remote, that host Monitors.
- Monitors provide the logic to detect faults, collect information, or control the environment.

To detect a fault, attach a debugger to a process to monitor it for exceptions. Or ping a device to verify it's active.

The following sections describe how agents and monitors work. Detailed descriptions of each monitor are located in the [monitors](#) section of the documentations.

Remote channels are used for both remote agents and remote publishers. Remote channels have two uses:

- Install Peach on the remote device. Remote agents and remote publishers can run and exchange data with the controlling Peach process.
- Install a small remote agent on the target machine that can receive and send communications to Peach. This implementation is necessary when it is not possible for Peach to run on the target machine. The remote agent implementation allows Peach to detect crashes more intelligently, thereby increasing the usefulness of fuzzing.

Remote publishers run publishers on a remote machine. This is useful when running OS specific publishers (such as [Com](#), which only runs on Windows).

One reason to remote publish is when Peach runs on a local machine that is much faster than the remote target; by using remote publishers, all fuzzing can be done on the local machine and the results sent to the target to be output. Another reason to remote publish is if you fuzz a system that expects to communicate with two different machines. By having one publisher speaking from the main machine and another from the remote machine you can trick the target application into thinking it was talking to two different systems

When fuzzing a virtual machine as a remote target, you may want to run a remote Agent on the target to monitor the internal state and a local Agent on the target to monitor the external state and to verify that the target is still alive. The local agent has the ability to start the virtual machine and the remote agent can run monitors from within the virtual machine once it has fully started.

Your current target may require OS-specific monitors. If you define an agent for each platform, you can use platform-specific monitors and run the same pit on each of the platforms.

Agents and monitors run in the order listed in a pit. The order can be very important when multiple agents are used to get the fuzzing target into a working state. An example of this is when one agent

starts a virtual machine and the second agent is a remote agent that controls things inside of the virtual machine. If the remote agent tries to run before the first agent starts the virtual machine, the Pit configuration will fail.

Shutting down monitors should occur in reverse order of their startup sequence. The shutdown sequence is important when you want to complete various actions before restarting a virtual machine.

Agents automatically try to reconnect when they become disconnected. When a device (like a virtual machine) restarts when a fault occurs, the remote agent will reconnect once the virtual machine is running again.

When the target opens fuzzing files from a command line, the target process must not open the file until fuzzing completes. To verify that fuzzing is complete, before we start the program, we use the monitor that starts the process and opens the file once we are done fuzzing; that is, fuzzed data file is on disk and has closed. Doing this causes the program to start and stop with every iteration, but it guarantees fuzzing is complete before the target process opens the fuzzed file.

Monitors are agent child processors; monitors get the target into a working state and monitor the target system. They can execute scripts, run processes, and monitor for crashes.

A variety of different monitors are supplied with Peach; select those you wish to use based on both the target platform and type of target. Is the target Windows software? Or Linux software? Or an embedded device? They each require different monitors. As an example, if I'm targeting an image-editing application, it's pointless to use a network sniffer.

Detecting crashes is challenging and important. If you aren't properly monitoring the environment, you will miss crashes that occur in unexpected places. In Peach's automated fuzzing process, you will need a combination of monitors that set up the environment and monitors that watch for crashes. For remote targets, both local and remote monitors are necessary.

When fuzzing remote devices, the more items that Peach monitors and the more ways that Peach automates the environment, the better the results will be. Peach can do very basic remote device monitoring (such as pinging a device with the Ping monitor) to make sure the device is still alive, or running a command on the box (via SSH) to check device system status.



The Ping and SSH monitors are basic monitors. When targeting a remote device, use additional monitors whenever possible.

Carefully select the monitors you use to fuzz because when a crash is detected, the monitor in use determines what and how information is logged. Running the target process inside of one of the debugger monitors (such as WindowsDebugger) places crashes into buckets, making it easier to identify crashes that are caused by unique bugs. When debugger monitors are unavailable, all crashes are logged in buckets based on which monitor detected the crashes, making it more difficult to identify unique crashes.

14.1. Detecting Faults

Faults result from Peach detecting interesting behavior such as an application crashing or entering an unexpected state. The places to look for faults are many; from making sure a program has not crashed, to using optical character recognition (OCR) on a screen for a specific dialog window.

Peach has two ways to detect faults:

- From Peach monitors that check for irregular behavior such as a program crashing or excessive memory usage.
- With [Godel](#) state modeling, Peach can detect that an application enters an unexpected state such as a login successful state when the expected result is a failed login state.

You can improve the chances that Peach will uncover faults by having Peach monitor more areas and use an appropriate number of robust tools. For example, many applications contain proper error handling that hide memory leaks from Peach when dumb or very simple fuzzing is done. These memory leaks are vulnerabilities within an application that might not be discovered unless additional tools (such as memory debuggers) are used to monitor multiple areas of application-accessed memory.

File fuzzing is performed by giving a valid sample file to Peach that produces a fuzzed version of the file. Depending on the target program, different sample files will touch different code paths. For complete code coverage and the best fuzzing results, use multiple sample files to touch all code paths of the target. Once you have a large number of sample files, the [minset](#) tool (included with Peach) can traverse all sample files and trace the code paths they exercise so that Peach does not use redundant samples.

Sometimes visual cues (such as pop-ups or a blinking LED) are the only way to monitor a system for faults. Peach can be extended to monitor the test target using visual cues (such as monitors that can use OCR on a screen and fault on a specific pop-up or when a specific LED on a board is lit) when no way exists within the system for Peach to detect the behaviors.

One example is fuzzing an iPhone application. In some cases, gathering the actual debug data from the iPhone application is very difficult; yet, when the application crashes, a pop-up window displays and notifies the user that a crash occurred. Using the [Open Vision Control](#) library, you can detect crashes that Peach can reproduce using only visual screen monitoring.

14.1.1. Memory debuggers

A memory debugger monitors the memory at the boundary of each memory allocation to detect when a program attempts to access memory past what was allocated and forces the application to crash instead of allowing it to handle the error. The use of a memory debugger while fuzzing will greatly increase Peach's ability to detect and reproduce faults.

All the memory debuggers make detecting faults much easier. There are many commonly used memory debuggers [including eFence, DUMA, Page Heap (Windows), and Guard Malloc (OS X)] and they are used in a variety of ways. Page Heap on Windows can be enabled via the GFlags tool for any

Windows executable. Guard Malloc on OS X is a debug option in Xcode for both Mac and iOS apps running in the simulator. DUMA (a forked version of eFence) is a cross-platform library that needs to be linked into the target's code when it's compiled.

14.2. Instrumenting the Fuzzing Environment

The first step in fuzzing is to send fuzzed data at a target. Peach doesn't need to stop here. Peach can orchestrate the entire environment. Controlling the fuzzing environment with Peach can be as simple as starting and stopping an application or as complex as automating external devices such as phones. The key to fully automating the Peach fuzzing environment is two-fold:

- detect all irregular behavior
- reset the environment to a known working state when needed

When fully configured, Peach can start and stop the target process, monitor the target process and the environment of the target, and restart the process or the entire environment as needed.

Targeting a single process (such as MS Paint with fuzzed images) is an example of a very basic fuzzing environment. In order to control the target completely, Peach needs to open and close Paint and monitor Paint for crashes. You can supply this level of control to Peach by using two monitors: WindowsDebugger and PageHeap. WindowsDebugger opens and closes Paint with the fuzzed files and monitors the environment for crashes. PageHeap (a memory debugger) makes detecting crashes easier. With both monitors configured, Peach can fuzz and monitor Paint indefinitely.

Fuzzing a target within a virtual machine (VM) is more complex. For virtual machine fuzzing, Peach is external to the VM and needs to start and stop the VM. Further, Peach needs to connect to two components within the VM: a remote agent and a publisher. Peach can monitor the environment by using a combination of local agents (to control the state of the VM) and remote agents (to monitor the environment inside of the VM).

Before using the remote agents, the VM needs to be in a started state; in the pit, call the local agent first so the VM can start fully before attempting to talk to the agents inside of the VM. Once the VM is started, the remote agents can start to control the system within the VM. If the fuzzing target within the VM is the Paint application, use both the WindowsDebugger and PageHeap monitors with the remote agent.

The best way to use VMs with Peach (since rebooting a VM can be a slow process) is to get the VM into a good state and take a snapshot. Peach can use this snapshot to start the VM and to revert to the snapshot on fault, so the system is always in a known, good state. Having the system in a known, good state ensures reliable fault reproduction.

Another complex example is mobile phone fuzzing. For mobile phone fuzzing, Peach must perform several actions:

- send both touch input and data input to the phone

- monitor the phone
- simulate NFC bumps
- spawn dynamic WIFI direct networks

Not every mobile application uses all features of the phone, but to fuzz and control any application on a phone, Peach needs to talk to a mobile device on any and all media. In order to fuzz an application that uses NFC to transfer data between phones, Peach needs a configuration that supports the following:

- opening the application on two phones
- touching buttons to get the phones ready to send and receive NFC, and
- transmitting the NFC.

When fuzzing some battery-powered devices, the target can enter an unresponsive state. To restart the device to a known, good state, the power button on the device has to be pressed. This is not good from a software automation standpoint since Peach needs to continue to fuzz without human interaction. Peach can generate a manual reset (pushing the power button) by issuing commands to a device with arms attached to a servo motor, that in turn, responds to the commands by extending an arm to press the button.

15. Test

The final step to create a Peach Pit is the [Test](#) element section. Here, the state model, agents, publishers, and logger all come together. A Test element represents a fuzzing test session that Peach can run. A Pit can contain one or more Test elements.

If a Pit contains multiple Test elements, you can specify the Test element to use for a specific fuzzing session at runtime.

See the references for: [Test](#), [Agent](#), [Include](#), [Exclude](#), [Mutators](#), and [Publisher](#).

16. Configuration Files

Most Peach pits contain configurable information. For example, a network Pit might store source and destination IP addresses for the network. Some configuration settings apply to a specific operating system.

Peach uses configuration files that allow different settings based on operating system. The information stored in a configuration file is typically used many times in a single definition and rarely stays the same. The presence of a configuration file eases the burden of supplying configurable settings by hand, an awkward and error-prone task.

You can create configuration files with default values and reuse them as needed. Peach configuration files use placeholders that are replaced at runtime either through a configuration file or via the command line. Peach automatically tries to load a configuration file along with the Pit file at the start of a fuzzing session. The naming convention of a Pit configuration file is `PIT.xml.config`, where `PIT.xml` is the name of your fuzzing definition.

```
<PitDefines>
  <All>    ①
    <Define key="TargetIPv4" value="127.0.0.2"/>
    <Define key="TargetPort" value="22"/>
  </All>
  <Linux>   ②
    <Define key="Interface" value="eth0"/>
  </Linux>
  <OSX>    ③
    <Define key="Interface" value="em0"/>
  </OSX>
</PitDefines>
```

① This section is common to all operating systems.

② This section is only for Linux.

③ This section is only for OS X.

Using a defined key consists of encasing the key with the two-character sequence `\#\#`. So, the key *Interface* becomes `##Interface##`, as shown in the following example.

```

<Monitor class="NetworkCapture">
  <Param name="Device" value="##Interface##"/>
</Monitor>

<Publisher class="Tcp" name="TcpHandler">
  <Param name="Host" value="##TargetIPv4##"/>
  <Param name="Port" value="##TargetPort##"/>
</Publisher>

```

Overriding one configuration file with a second (or another) configuration file requires using the **--config** command-line argument. The values in the file specified on the command line override the values listed in the default .config file.

```
Peach.exe --config=CommonConfig.xml MyPit.xml
```

You can specify individual configuration items (keys) on the command line by using the **-D** argument. Values provided via the command line override those provided by file.

```
Peach.exe -DTargetIp=10.1.1.1 MyPit.xml
```

16.1. Internally-defined Keys

Peach includes several predefined configuration keys that can be used in your Pits.

Peach.Cwd

Peach current working directory. While this directory is usually the directory containing Peach, the directory can be another location on your system. The Peach Working Directory is set by launching Peach from the shell command line. The value is the current working directory of the command shell when you start Peach.

PitLibraryPath

The Pit library path is the full path of the folder in the Peach installation directory that contains your licensed Peach Pits and Pit Packs. The folder name is "pits", and contains subdirectories that hold your licensed Pits, Pit configurations, and Peach-supplied sample files.

Peach.LogRoot

Full path of the directory where Peach stores logging information from each fuzzing session. The default log root location is a subdirectory of the Peach Installation Directory. The default name of this directory is "Logs".

Peach.OS

Operating system that hosts the Peach Fuzzer platform. The value is selected when downloading the

Peach distribution image.

Peach.Pwd

Peach installation directory. This directory contains the Peach executable, and usually is named "peach".

17. Debugging Pit Files

A PIT is NOT valid unless...

- The first iteration is successful against the target.
- The target is isolated from other input.
- You can demonstrate the target has parsed your data by checking the following items:
 - GUI opens the image data.
 - Network service logs your data.
 - You get the correct response packet from the target for the packet you sent.
 - The status lights blink appropriately.
- Logging is enabled.
- Monitoring is enabled.
- Monitoring has been tested and fault on a demo/test fault.

17.1. What to do if the Pit Doesn't Parse

Peach provides a tool to perform validation and lint checking that can assist in tracking down basic syntax issues in your pit file. See [PitTool](#) for additional information.

```
pittool compile pit_file.xml
```

17.2. The pit doesn't run properly

During the testing and building phase, run the first iteration only. The `-1` command-line argument runs the record iteration on the Pit. No fuzzing occurs.

```
Peach.exe -1 pit_file.xml
```

Reduce the pit to basic parts in order make sure the following items are specified correctly:

1. Test - The references have the right names.
2. Log Path - The folder path is valid.
3. Publisher - The functionality works:
 - Communicating (network Publisher)
 - Sending or receiving data or

- Outputting data
4. Agent - The Monitor setup is correct and the Monitor has connectivity.
 5. State Machine (StateModel) - The references to the data models are correct.
 6. DataModel - Loads and displays the appropriate layout in the Peach Validator.

The Peach debug argument (`--debug` on the command line) provides verbose output that can help in debugging a Pit file.

```
Peach.exe -1 --debug pit_file.xml
```

18. Converting Pits from Peach V2.3

Fuzzing definitions (pits) from Peach version 2.3 do not run "as-is" in the current version. The changes needed to enable these pits to run in the current version include a few global issues; advancements in methodology that translate to changes in the description language; and product improvements, bug fixes, and enhancements.

Changing a pit to run in Peach version 3.x means that you can run the pit through the command-line interface. However, Peach version 3.x introduced a web-based UI that requires some additional work to have compatibility between a pit and the new web UI.

The remainder of this section addresses the global changes, changes to individual commands, and how to set up a pit to run with the Peach web UI.

18.1. Global Changes

The global changes consist of the following items that you need to change for all version 2.3 pits:

- The `<Peach>` element `xmlsru`ns and `xsi:schemaLocation` attribute values have changed. The simplest fix is to replace the v2.3 `<Peach>` element with the following:

```
<!-- Peach v3.x -->
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">
```

- The `<Include>` element for `defaults.xml` is no longer used. Remove the include statement from the pit. An example of the statement to remove follows:

```
<!-- Peach v2.3 -->
<Include ns="default" src="file:defaults.xml"/>
```

- The relationship `from` is no longer used. Remove all `from` relations. An example statement follows:

```
<!-- Peach v2.3 -->
<Relation type="size" from="Length" />
```

Peach v2.3 provided `from` relations to boost performance by specifying both sides of a relationship using `of` and `from` parameters, as in the following example.

```

<!-- Peach v2.3 -->
<Number name="Length" size="32" endian="network" signed="false">
    <Relation type="size" of="Data" />
</Number>
<Blob name="Data">
    <Relation type="size" from="Length" />
</Blob>

```

Peach v3.x does not use the `from` parameter. The following example, written for Peach v3.x, provides identical functionality to the previous example.

```

<!-- Peach v3.x -->
<Number name="Length" size="32" endian="network" signed="false">
    <Relation type="size" of="Data" />
</Number>
<Blob name="Data" />

```

- The `<Logger>` element is now part of the `<Test>` element. Move the Logger element block into the Test element block. For more information, see the next item.
- The functionality of the `<Run>` element is now part of the `<Test>` element. Upon moving the Logger element block into the Test element block, remove the Run element block from the pit. An example follows.

```

<!-- Peach v2.3 Test and Run elements sample -->
<Test name="UdpResp">
    <Agent ref="LocalAgent" />
    <StateModel ref="UdpTransaction" />
    <Publisher class="Udp">
        <Param name="host" value="192.168.1.3" />
        <Param name="port" value="53" />
    </Publisher>
</Test>

<Run name="DefaultRun">
    <Logger class="logger.Filesystem">
        <Param name="Path" value="logs" />
    </Logger>
    <Test ref="UdpResp" />
</Run>

```

In Peach v3.x, the Test element block identifies the Agent, StateModel (and, by implication, the DataModel), Publisher, and Logger for a fuzzing session. A v3.x Test element block follows. The

block is functionally identical the previous example.

```
<!-- Peach v3.x Test element sample -->
<Test name="Default">
  <Agent ref="LocalAgent" />
  <StateModel ref="UdpTransaction" />
  <Publisher class="Udp">
    <Param name="Host" value="192.168.1.3" />
    <Param name="Port" value="53"/>
  </Publisher>
  <Logger class="logger.Filesystem">
    <Param name="Path" value="logs" />
  </Logger>
</Test>
```



In the example, the name for the Test element is **Default**. This is the default name for the Test element. At runtime, Peach v3.x automatically looks for and runs the default Test, unless you specify a Test name on the command line.

- The parameter names for Monitors and Publishers now use CamelCasing. In the previous example, the parameter names **"Host"** and **"Port"** for the publisher have changed slightly due to CamelCasing. You can use the Peach DOM reference or the developer's guide to check parameters that fail validation. Use the following command to generate the DOM reference.

```
peach --showenv
```

- You can place the data and state model definitions in separate xml files to improve re-use of these definitions. Once defined, you can pull these definitions into different pits to perform different tests on the same data and state models. Specify the file containing the models using the include xml element.

The following example shows file fragments of a pit and definition files that contain the state and data models. Two include elements are used: 1) the main pit file includes the state model, and 2) the state model file includes the data model.

```

<!-- Peach v3.x Pit file -->
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <!-- Pull the StateModel into the pit. -->
    <Include ns="FTP" src="file:FTP_State.xml" />

    <Test name="Default">
        <StateModel ref="FTP:Client" />
    </Test>
</Peach>

<!-- Peach v3.x StateModel definitions (FTP_State.xml) -->
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <!-- Pull the DataModel into the StateModel. -->
    <Include ns="FTP" src="file:FTP_Data.xml" />

    <StateModel name="Client" initialState="Initial">
        <!-- FTP Client State Model -->
    </StateModel>
</Peach>

```

18.2. Changes to Individual xml Elements

The following Peach v2.3 xml elements require changes when used with Peach v3.x.

<Defaults>

The Defaults element contains default values for parameter definitions. If an individual element does not specify an optional parameter, Peach uses the value specified in this element block. Values for optional attributes and parameters are defined in this element.

Note that the **Size** attribute of the number element is a required attribute that must be specified with each number instance.

<Import>

This xml element has one attribute, import, that names the python file containing code. Note the **.py** postfix is not used.

In v3.x, you must specify each python file you want to use. Wild card characters (*) are not supported.

The from attribute is now a top-level element named <PythonPath> that specifies the search path for all python modules. Note that a trailing \ or / for the path is not used. Use multiple <PythonPath> elements to tell Peach to search in more than one place.

<DataModel>

Remove all `from` relation statements from all data model elements.

<Flag>

A multiple-bit Flag that uses the value parameter accepts a value expressed as a hexadecimal integer or a sequence of hexadecimal digits.

- A value expressed as a hexadecimal integer must fit into the bits allocated for the flag.
- A value expressed as a sequence of hexadecimal digits must have sufficient length to span the number of bits in the flag.

<Number>

value Attribute

When specifying a value for a number, you can use an integer value, a hexadecimal integer value or a sequence of hexadecimal digits. A value expressed as a sequence of hexadecimal digits (where `valueType="hex"`) must match lengthwise with the allocated size of the number or a validation error occurs. For example, initially setting a 64-bit number to one can be specified as seven digits of zeros and one digit of one:

```
<!-- Peach v3.x -->
<Number size="64" valueType="hex" value="00 00 00 00 00 00 00 01" />
```

For values expressed as hex integers prefix the value with `0x`.

```
<!-- Peach v3.x -->
<Number size="64" value="0x01" />
```

Size is a required attribute. You cannot use a default size specified in the Defaults element block for number elements.

<StateModel>

No changes.

<Test>

Now includes logger definitions, and performs the functionality of the v2.3 Run xml element.

<Run>

This section is obsolete. Move the Logger into the test section.

<Publisher>

RawIpv4

- *Host* and *Protocol* are required parameters.
- The *Protocol* parameter is required and must have a valid value.
 - "17" is the value for UDP.
 - "6" is the value for TCP.
- The *Host* parameter is required and must have a valid value, specified as a hostname or an IP address.
- The *Interface* parameter now uses CamelCase with a capitalized first letter. This parameter is optional.



While the old publisher name is valid, the current name `RawIpv4` is the name used in the documentation and in log messages from the Peach engine.

18.3. How to Make a Pit Usable by the Peach Web User Interface

In order to use a Peach pit with the Web UI, each pit requires an associated configuration file. The configuration file contains all parameters considered configurable. Peach automatically loads the pit and the associated configuration file. The values contained in the configuration file are exposed in the web UI.

Here are the steps to follow to make a Pit usable by the web UI. The information contained in this example is sufficient as a complete configuration file.

1. Create a configuration file and give it a name.

The name of the configuration file uses the following form:

Name.xml.config

Name is same as the base name of the pit. `xml` and `config` are literals. For example, the pit `XXX.xml` would have a configuration file named `XXX.xml.config`.

2. Add parameter definitions for the pit.

The target address and port are common values to define.

```
<Ipv4 key="TargetIPv4"  
      value="127.0.0.1"  
      name="Target IPv4 Address"  
      description="The IPv4 address of the target machine or device." />
```



On Windows, run `ipconfig` and look for the *IPv4 Address* field.

On Linux, run `ifconfig` and look for the *inet addr* field.

On OS X, run `ifconfig` and look for the *inet* field." />

```
<Range key="TargetPort"  
      value="21"  
      min="0"  
      max="65535"  
      name="Target Port"  
      description="The target or destination port to send the network packet." />
```

These values are ready for use in the pit as `##TargetIPv4##` and `##TargetPort##`. They can be used anywhere in the pit file. For example, the Publisher definition would be updated as follows:

```
<Publisher class="Udp">  
  <Param name="Host" value="##TargetIPv4##"/>  
  <Param name="Port" value="##TargetPort##"/>  
</Publisher>
```

3. Remove all agent element blocks from the PIT.

You can specify the agent and the monitoring options using the web UI.

4. Create a folder to hold the converted pits.

Create a sub-folder in the pits folder, such as peach/pits/converted.

5. Place the converted pits and associated configuration files in the newly created folder.

For example, after placing, the XXX pit and configuration file in the converted folder, the file locations would be as follows:

```
peach/pits/converted/XXX.xml  
peach/pits/converted/XXX.xml.config
```

6. Your pit is ready to configure, test, and then use with the Peach web UI.



If the need arises, using the command-line interface, you can manually override a configuration setting using the `-D` switch. An example follows: `peach.exe XXX.xml -DTargetIPv4=10.0.0.1`

19. Running Peach

Peach Fuzzer Professional includes a number of executable files. In most instances, using the web interface will meet your needs. Peach can also be used from the command line. This includes using the Peach Web Interface, which launches by running Peach from the command line without any parameters or switches.

The following list identifies the support applications included with Peach Fuzzer Professional.

Program	Executable	Description
Peach Web Interface	Peach.exe	The Peach Web Interface for Peach Fuzzer.
Peach Command Line	Peach.exe	The Peach Command Line Interface for Peach Fuzzer.
Peach Agent	PeachAgent.exe	The Agent process for Peach Fuzzer.
Minset	PeachMinset.exe	Find the minimal set of sample files for use during fuzzing with the greatest code coverage for a given target.
Validator	PeachValidator.exe	Graphical tool for debugging pit files. Provides a graphical view of the data model graph before and after cracking a file.
PitTool	PitTool.exe	Utilities useful for Pit development.
Peach Multi-Node CLI Tool	sdk\tools\peachcli	Control and coordinate multiple Peach instances.

19.1. The Peach Web Interface

The Peach Web Interface is an interactive interface to Peach Fuzzer Professional that simplifies monitoring of local and remote fuzzing jobs. Using Peach Web Interface, you can select, configure, and run fuzzing definitions (Pits). The Peach Web Interface is operating system agnostic. This means that you can use the same interface to run Peach, whether on Windows, Linux, or OS X. When a Pit is running, you can view the state of the fuzzing job and see faults that result.

The Peach Web Interface works with Pits from the included Pit Library. With a little configuration, the Pits will be ready to run. Once you settle on a Pit that you want to use, you can configure the Pit to detect faults, to collect data, or to automate the fuzzing session.

19.1.1. Peach Web Interface Installation Requirements

The Peach Web Interface uses modern web technologies, such as HTML 5. Yet, Peach Web Interface requires two things to run:

- JavaScript, enabled in your web browser
- A supported browser
 - Internet Explorer: version 9 and newer
 - Safari: version 6 and newer
 - Firefox: version 4 and newer
 - Chrome: version 12 and newer
 - Opera: version 12 and newer



Other browsers might work; however, they are not officially supported

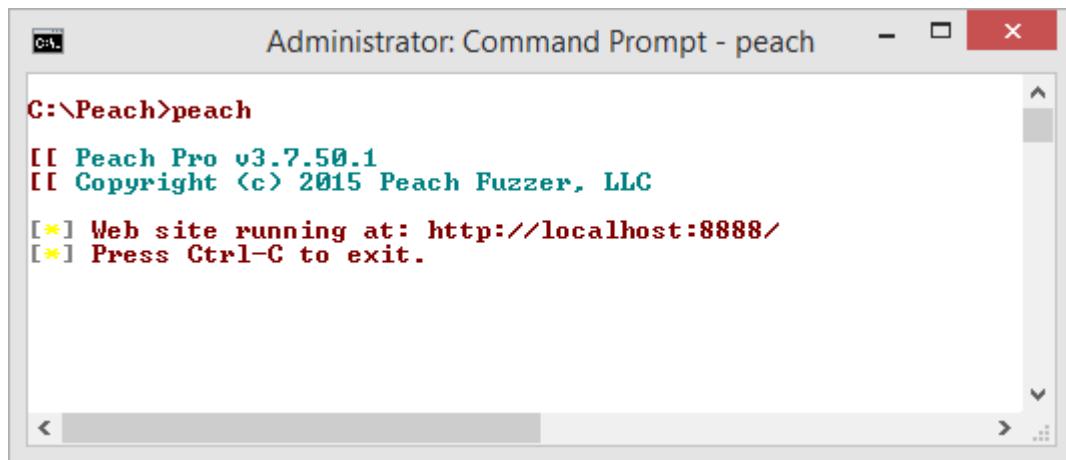
19.1.2. Starting the Peach Web Interface

You can start the Peach Web Interface using the GUI File Manager or from the command line. Both methods include the same functionality, use the method you prefer.

We recommend running Peach with heightened or administrative privileges. Some applications that provide monitoring functions, such as debuggers, need heightened access to run.

You can start Peach from a command line with two actions:

1. Open a command shell with administrative privileges.
2. On the command line, type Peach and press RETURN. The following illustration shows starting Peach in Windows.



The image shows an Administrator Command Prompt window titled "Administrator: Command Prompt - peach". The window contains the following text output:

```
C:\Peach>peach
[[ Peach Pro v3.7.50.1
[[ Copyright <c> 2015 Peach Fuzzer, LLC
[*] Web site running at: http://localhost:8888/
[*] Press Ctrl-C to exit.
```

The Peach Web Interface is the default mode of operation for the command line. Launching the Peach Web Interface performs the following tasks:

1. Starts Peach Fuzzer Professional
2. Loads your default web browser with the Peach Web Interface URL

On the first launch of Peach, a licensing page displays.

File Edit View History Bookmarks Tools Help

localhost:8888/eula

END-USER LICENSE AGREEMENT: PEACH FUZZER(TM) ENTERPRISE SOLUTION

IMPORTANT: CAREFULLY READ THIS END USER LICENSE AGREEMENT BEFORE ACCESSING OR USING THE SOFTWARE, DEFINITION FILES OR FEATURES. BY ACCESSING OR USING THE SOFTWARE, DEFINITION FILES OR FEATURES, YOU AUTOMATICALLY ACKNOWLEDGE, ACCEPT AND AGREE TO ALL THE TERMS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO THESE TERMS AND CONDITIONS, YOU MAY NOT ACCESS OR OTHERWISE USE THE SOFTWARE AND ACCOMPANYING FEATURES.

This End-User License Agreement (the "Agreement") is a legally binding and enforceable agreement between Peach Fuzzer, LLC, a Washington limited liability company (the "Company") and the customer, individual or entity, set forth on the applicable Customer Invoice (the "Customer"), governing the Customer's license to and use of the Software, Definition Files and Features, as well as the Customer's license to create and use Customer Peach Pits.

1. DEFINITIONS.

(a) "Customer Invoice" means that certain final invoice issued by the Company to the Customer that identifies the specific license to products and services of the Company that the Customer has purchased.

(b) "Definition File(s)" means any Peach Pit files, development tools, and software programs developed by the Company and provided in the Peach Pit library subject to a current Peach Pit License.

BY CLICKING "I ACCEPT" YOU ACKNOWLEDGE THAT YOU HAVE READ, UNDERSTAND, AND AGREE TO BE BOUND BY THE TERMS ABOVE.

Accept Reject

When you have read and understood the End User License Agreement, click *Accept*.

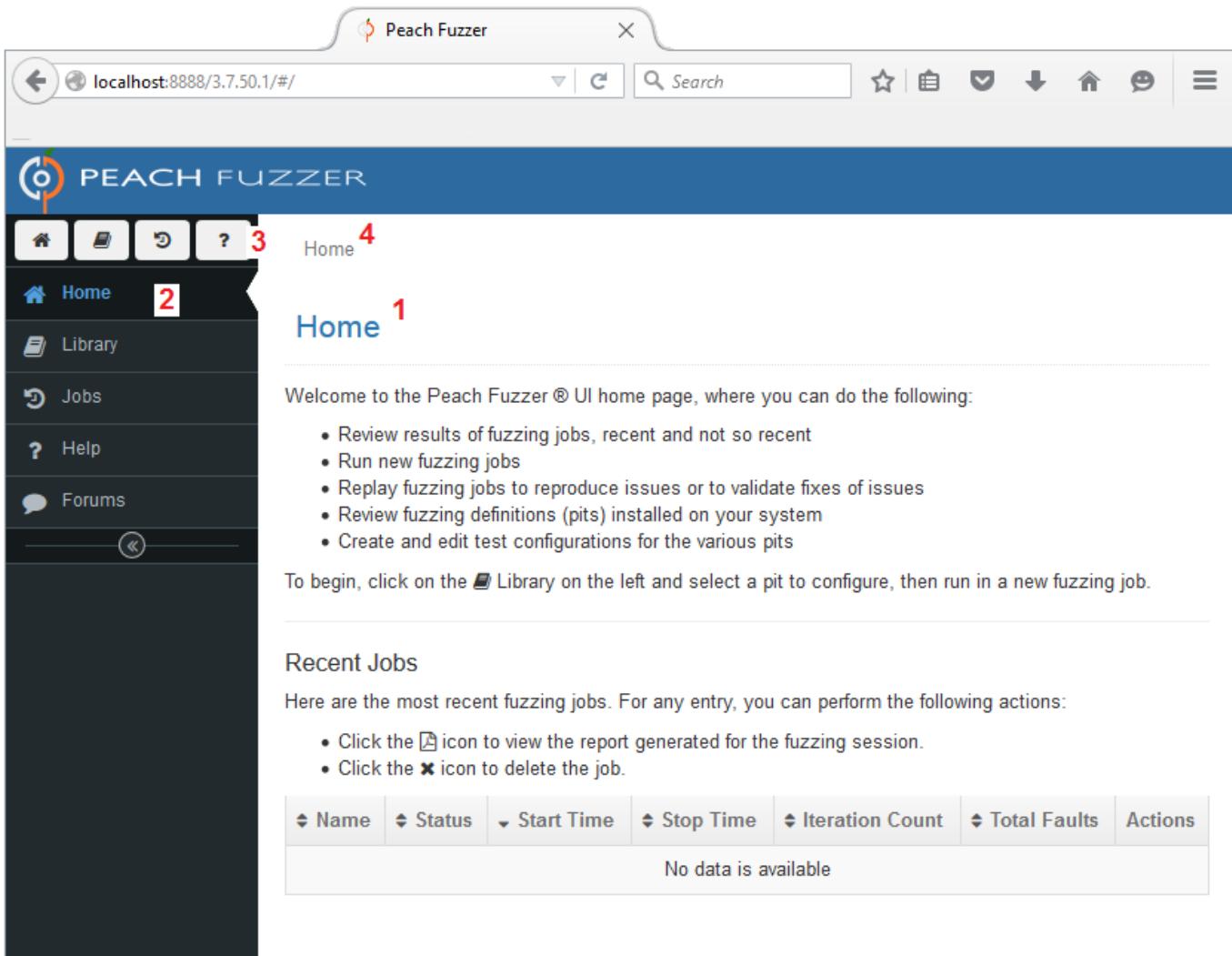
3. Displays the Peach Home Page

The screenshot shows a web browser window with the title "Peach Fuzzer". The address bar displays "localhost:8888/3.7.50.1/#/". The main content area is titled "PEACH FUZZER" with a logo. A sidebar on the left contains icons for Home, Library, Jobs, Help, and Forums. The "Home" icon is highlighted. The main content area has a "Home" button at the top. Below it, the title "Home" is displayed in blue. A welcome message says: "Welcome to the Peach Fuzzer ® UI home page, where you can do the following:" followed by a bulleted list: • Review results of fuzzing jobs, recent and not so recent • Run new fuzzing jobs • Replay fuzzing jobs to reproduce issues or to validate fixes of issues • Review fuzzing definitions (pits) installed on your system • Create and edit test configurations for the various pits. A note below says: "To begin, click on the Library on the left and select a pit to configure, then run in a new fuzzing job." A section titled "Recent Jobs" follows, with a note: "Here are the most recent fuzzing jobs. For any entry, you can perform the following actions:" and a bulleted list: • Click the icon to view the report generated for the fuzzing session. • Click the icon to delete the job. A table header with columns "Name", "Status", "Start Time", "Stop Time", "Iteration Count", "Total Faults", and "Actions" is shown, with a message "No data is available" below it.

From Home, you can use click entries and buttons on the left side of the page to work with your licensed Pits, view reports and details of previous fuzzing sessions, and to interact with the Peach forums.

Parts of the User Interface

The Web User Interface includes many useful components, the most prominent are called out in the following illustration:



1. Page Title

The Page title identifies the page you are on and establishes the context of your current task.

2. Menu Entries

Menu entries are located along the left edge of the browser window. These buttons identify the functionality immediately at your disposal. Whether you want to explore your Pit Library, investigate a completed fuzzing job, or get some assistance from Help or the Forums, the menu is there for you.

The entries in the menu change according to the work context. For example, when you start working with a Pit, the menu entries change to accommodate editing a Pit definition. Click an entry in the menu to start working on that item. There is also an option to collapse the menu.

3. Menu Icons

Located above the menu entries, the menu icons are always present and do not change. The icons are always:

- Home Page

- Your Pit Library
- Your Fuzzing Job Historical Results
- Peach User's Guide (HTML Help)

Click a menu icon to move there.

4. Breadcrumb menu

The breadcrumb is located above the Page Title and identifies your position within the Peach Web UI. Portions of the breadcrumb are links and marked appropriately by color.

A Quick Tour of the Peach Web Interface

This section provides a brief glimpse of the main areas of Peach that are accessible from the menu and buttons on the left edge of the page. The tour is quick: about a slide per menu item.

Library

The *Library* menu is where you select and configure Pits (fuzzing definitions), and run fuzzing jobs. The page lists the licensed Peach Pits first, then follows with the defined configurations that you create.

The screenshot shows the Peach Fuzzer application interface. At the top, there's a header bar with the title "Peach Fuzzer" and a URL "10.0.1.57:8888/#/library". The main content area has a blue header "PEACH FUZZER". On the left, a sidebar menu includes "Home", "Library" (which is selected), "Jobs", "Help", and "Forums". Below the sidebar is a search bar with the placeholder "Search for pits...". The main content area is titled "Library" and contains a welcome message: "Welcome to the Peach Pit library. This page consists of the following sections: Pits, Configurations, Legacy". To the right of the welcome message are three sections: "Pits" (All Peach Pits (test modules) that are present on your system), "Configurations" (Ready to use, saved Pit configurations), and "Legacy" (Pits from an older version of Peach that require migration so they can be used in the latest version). The "Pits" section is expanded, showing categories like "Application" (HL7, PPTX), "Image" (BMP, DICOM File, GIF, ICO, JPG, JPG2000, PNG), and "Net".

Jobs

The *Jobs* menu provides access to your fuzzing job results. Click on an entry to see the report status, summary, metrics, and drill down detail of individual findings.

The screenshot shows a web browser window titled "Peach Fuzzer". The address bar displays "localhost:8888/3.7.50.1/#/jobs". The main content area is titled "PEACH FUZZER" with a logo. A sidebar on the left contains links for "Home", "Library", "Jobs" (which is selected and highlighted in blue), "Help", and "Forums". The main content area shows the "Jobs" page with the title "Jobs". It includes a descriptive text about fuzzing jobs and a list of actions for each entry. Below this is a table header with columns: Name, Status, Start Time, Stop Time, Iteration Count, Total Faults, and Actions. The table body contains a single row with the message "No data is available".

Help

The *Help* Menu provides access to the online Peach User Guide. This instructional piece provides workflows for installing Peach, recipes for setting up and running various configurations, and descriptions of the Peach monitors for detecting faults (issues), collecting data, and automating the test environment.

The screenshot shows a web browser window displaying the "Peach Fuzzer Professional" documentation. The title bar reads "File Edit View History Bookmarks Tools Help" and "Peach Fuzzer Professional - X". The address bar shows "localhost:8888/docs/index.html". The page header features the "PEACH FUZZER" logo and navigation links for "SIDEBAR" and "NEXT ▶". The main content area is titled "Peach Fuzzer Professional" and includes a "User Guide" section for "Peach Fuzzer, LLC". A "Table of Contents" sidebar on the left lists chapters such as Preface, Installation, What's new in Peach Platform v3.7 (2015 Q3), The Peach Web Interface, Monitor Recipes, Agents, and Monitors, each with further sub-links.

Peach Fuzzer Professional

User Guide

Peach Fuzzer, LLC

Table of Contents

- [Preface](#)
 - [What is Peach Fuzzer Professional?](#)
 - [Additional Resources](#)
 - [Bug Reporting Guidelines](#)
- [Introduction to Fuzzing](#)
 - [Dumb Fuzzing](#)
 - [Smart Fuzzing](#)
- [Installation](#)
 - [Windows](#)
 - [Linux](#)
 - [OS X](#)
 - [Peach Fuzzer Professional License](#)
- [What's new in Peach Platform v3.7 \(2015 Q3\)](#)
 - [Installation](#)
 - [Fuzzing Definitions \(Pits\)](#)
 - [WindowsKernelDebug Monitor](#)
 - [Documentation](#)
- [The Peach Web Interface](#)
 - [Peach Web Interface Installation Requirements](#)
 - [Starting the Peach Web Interface](#)
 - [Quick Start Wizard](#)
 - [Configuration Menu](#)
 - [Fuzzing Session](#)
 - [Faults](#)
 - [Metrics](#)
 - [Documentation](#)
 - [Forums](#)
 - [Switching Pits](#)
- [Monitor Recipes](#)
 - [Recipe: Monitoring a Network Device](#)
 - [Recipe: Monitoring a File Consumer \(File Fuzzing\)](#)
- [Agents](#)
 - [Agent Channels](#)
- [Monitors](#)
 - [Monitor Parameters When and Wait When](#)
 - [Android Monitor](#)
 - [AndroidEmulator Monitor](#)
 - [ButtonClicker Monitor](#)



Additionally, the *User Guide* is available in PDF format.

Peach Fuzzer Professional: User Guide

Table of Contents

Bookmarks

- 1. Preface
 - 1.1. What is Peach Fuzzer Professional?
 - 1.2. Additional Resources
 - 1.3. Bug Reporting Guidelines
- 2. Getting Started with the Peach Fuzzer Platform
 - 2.1. Picking your fuzzing target
 - 2.2. Installing Peach
 - 2.3. Launching Peach
 - 2.4. Selecting a Peach Pit
 - 2.5. Specifying a Test Configuration
 - 2.6. Running the Fuzzing Job
 - 2.7. Interpreting Results
 - 2.8. Fixing Issues and Re-running Test Cases
- 3. Introduction to Fuzzing
 - 3.1. Dumb Fuzzing
 - 3.2. Smart Fuzzing
 - 3.3. When to Stop Fuzzing
- 4. Installation
 - 4.1. Peach Fuzzer Professional License
 - 4.2. Downloading the Peach Fuzzer Distribution Files
 - 4.3. Windows
 - 4.4. Linux
 - 4.5. OS X
- 5. What's new in Peach Platform v3.8 (2015 Q4)
 - 5.1. Fuzzing Definitions (Pits)
 - 5.2. Canbus Publisher
 - 5.3. User Interface Enhancements
 - 5.4. Documentation
- 6. The Peach Web Interface
 - 6.1. Peach Web Interface Installation Requirements
 - 6.2. Starting the Peach Web Interface
 - 6.3. Quick Start Wizard
 - 6.4. Configuration Menu
 - 6.5. Test Pit Configuration
 - 6.6. Fuzzing Session
 - 6.7. Faults
 - 6.8. Metrics
 - 6.9. Switching Pits
- 7. Monitoring Recipes
 - 7.1. Recipe: Monitoring a File Consumer (File Fuzzing)
 - 7.2. Recipe: Monitoring a Linux Network Service Client

Forums

Peach has user forums that serve as a knowledge base of user questions, and as an active platform to raise questions of current need or interest. Feel free to explore the forums. Note that the professional forum provides a service venue to licensed users of Peach Professional and Peach Enterprise solutions.

Forums

Today's Posts Mark Channels Read

You may have to [register](#) before you can post: click the register link above to proceed. To start viewing messages, select the forum that you want to visit from the selection below.

Forums

FORUMS	LATEST ACTIVITY	MY SUBSCRIPTIONS
Directory	Topics Posts Last Post	
Peach Community Edition		
General Support Peach Community Edition general support forum.	445 1,575  by DataModel to... by TeraFuzz Yesterday, 07:01 AM	
Peach Pits A place to post and discuss fuzzing definitions.	12 17  by Not getting st... by mike 10-20-2015, 03:14 AM	
Mark Channels Read		
Peach Fuzzer Forums Statistics		
Topics: 503 Posts: 1,790 Members: 467 Active Members: 20		

Latest Topics

-  [ValidValues / Choice - how to rea...](#)
mischcon
Hi there, in my bachelor thesis I nee...
10-20-2015, 04:56 PM
-  [Almost a year now without updat...](#)
shai.sarfaty
hi, it's almost a year now without an...
10-19-2015, 08:56 PM
-  [DataModel to load Data line by li...](#)
TeraFuzz
Hi, I have a file that has to be used a...
10-18-2015, 09:16 PM

[View All](#)

[Tag Cloud](#)

19.1.3. Configuration Menu

You can configure a Pit by using the Configuration menu. This menu consists of four parts: variables, monitoring, tuning, and test. Only the Variable and Monitoring parts need to be completed to pass a test of the configuration.

To begin from the Home Page,

1. Click the Library menu entry.

- From the Pit Library, select a Pit or an existing configuration.
 - Selecting a configuration means that you are revising the settings of an existing configuration. Peach displays the start screen for the configuration.
 - Selecting a Pit means that you are creating a new configuration. You will need to name the configuration, optionally provide a description, and click "Submit" to reach the start screen for the configuration.

The screenshot shows the Peach Fuzzer web application interface. At the top, there's a navigation bar with links for File, Edit, View, History, Bookmarks, Tools, and Help. Below that is a browser-style header with a back button, a search bar containing 'Search', and various icons for refresh, download, and navigation. The main title is 'PEACH FUZZER'. On the left, a sidebar menu includes 'Pit' (selected), 'Quick Start', 'Configure', 'Help', and 'Forums'. The main content area shows the title 'BMP_Demo'. A yellow warning box says: 'Warning! The currently selected Pit should be configured for monitoring the environment. Pit Configuration Quick Start'. Below it, text states: 'This configuration uses the `BMP_Demo` pit and includes configuration data for your test setup.' There are three buttons: 'Quick Start Wizard', 'Configure Variables', and 'Configure Monitoring'. Under 'Start Options', it says: 'The Start Options specify the test cases that start and end a fuzzing job, and identify a seed for generating mutated data. For a new fuzzing job, the default values are usually appropriate to use, although you can select other values to use.' It continues: 'If you want to replay a fuzzing run, you need to use the same Start Options as in the original fuzzing job. You can obtain these values by 1) selecting the fuzzing run you want to replay, and 2) click the Replay button on that job page.' It then says: 'For replay, changing the Start Options has the following effects:' followed by a bulleted list. At the bottom, there are fields for 'Seed' (set to 'Random Seed') and 'Start Test Case' (set to '1').

File Edit View History Bookmarks Tools Help

Peach Fuzzer

localhost:8888/3.7.50.1/#/pit/5376991a9f3fe90d308

Search

PEACH FUZZER

Home / Library / BMP_Demo

BMP_Demo

Warning! The currently selected Pit should be configured for monitoring the environment. Pit Configuration Quick Start

This configuration uses the `BMP_Demo` pit and includes configuration data for your test setup.

Configuration Options

Quick Start Wizard Configure Variables Configure Monitoring

Start Options

The Start Options specify the test cases that start and end a fuzzing job, and identify a seed for generating mutated data. For a new fuzzing job, the default values are usually appropriate to use, although you can select other values to use.

If you want to replay a fuzzing run, you need to use the same Start Options as in the original fuzzing job. You can obtain these values by 1) selecting the fuzzing run you want to replay, and 2) click the Replay button on that job page.

For replay, changing the Start Options has the following effects:

- Change the Seed. This changes the mutated data values used in all test cases. The result is definitely a new job.
- Change the Start or Stop Test Case. This changes the test cases (first and/or last) to execute in the job, thereby lengthening or shortening the total number of test cases executed in the job. Note that changing the test cases (first and last) for the job can help validate an issue or the fix for an issue. However, the result might not produce the intended results.

Click "Start" to begin the fuzzing job.

Seed: Random Seed

Start Test Case: 1

3. Click the "Configure Variables" button to define or edit the variables and their values.

Switching Pit Configurations



The active Peach Pit configuration can be changed in the Peach Web Interface by clicking on the Home menu icon above the menu along the left side of the screen. Then, click on the Library menu item to choose a Pit or Pit configuration in your library.

Variables

The variables data entry screen lists the information needed by the selected pit, as in the following illustration. Some information is pit-specific, such as file names used to fuzz file formats and port addresses used to fuzz network protocols. Other information applies to the Peach environment; two examples are the Peach Installation Directory and the Pit Library Path.

Variables

This page lists the information needed by the selected pit. Some of the information applies to the Peach environment; two examples are the Peach Installation Directory and the Pit Library Path. Other information is pit specific, such as port addresses for a network protocol or source files for a file format.

Save + Add Variable

▼ All

Name	Key	Value
Fuzzed Data File	FuzzedFile	? fuzzed.bmp
Seed File	Seed	? *.bmp
Sample Path	SamplePath	? ##PitLibraryPath##/_Common/Samples/Image

▼ System Defines

Name	Key	Value
Peach OS	Peach.OS	? windows
Peach Installation Directory	Peach.Pwd	? C:\Peach
Peach Working Directory	Peach.Cwd	? C:\Peach
Root Log Directory	Peach.LogRoot	? C:\Peach\Logs
Pit Library Path	PitLibraryPath	? C:\Peach\pits

Pit-specific Variables

In the illustration, three variables are specific to the **BMP** Pit. Because the variables have default values, you can use the supplied values during the fuzzing job.

- If a variable lacks a default value and is not marked optional, you need to supply a value for the variable before you can fuzz the target.
- If a variable is labeled optional or has a default value, you need not supply a value to enable the configuration to run.

When entering a data value, type the value into the appropriate text box of the form. If you'd like to use the value of another variable, type `##`, the name of the variable that contains the value you want to use, and `##` to end the variable name.

The default value of the `Sample Path` variable in the previous illustration uses another variable (`PitLibraryPath`) as part of its value.

System-defined Variables

The Peach environment variables includes the following entries:

Peach OS

The Peach OS identifies the operating system that the Peach Fuzzer is using. The value is selected when downloading the Peach distribution image.

Peach Installation Directory

The Peach Installation Directory is the directory that contains the peach executable file. This directory was created when Peach was installed on the computer system.

Peach Working Directory

The Peach working directory is the current working directory for fuzzing. While this directory is usually the directory containing Peach, the directory can be another location on your system. The Peach Working Directory is set by launching Peach from the shell command line. The value is the current working directory of the command shell when you start Peach.

Root Log Directory

The default name of this directory is Logs. The default location is a subdirectory of the Peach Installation Directory. You can specify another location for it, such as a subdirectory of the Peach Working Directory.



Each pit has its own logs that appear as subdirectories of the logger Path.

Pit Library Path

The Pit library path is the full path of the folder in the Peach installation directory that contains your licensed Peach Pits and Pit Packs. The folder name is "pits", and contains subdirectories that hold your licensed Pits, Pit configurations, and Peach-supplied sample files.

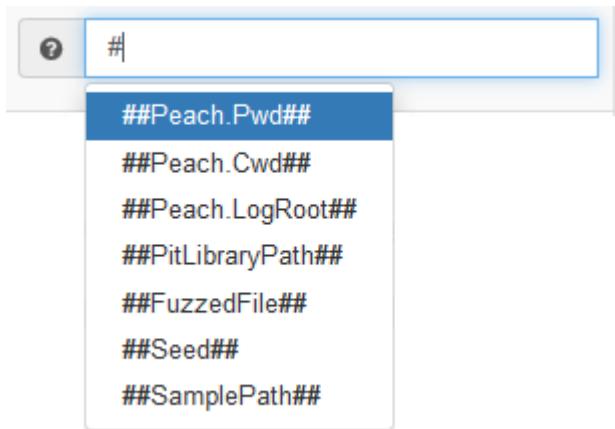
For descriptions on the pit-specific variables, see the documentation for the individual pit.

Custom Variables

You can create your own set variables for Peach. Perhaps you want to identify where the image repository is located. Or, perhaps you want to identify the MAC addresses, interface names, and IP addresses for the network interfaces on your system.

Whatever the resource you'd like to specify, once you've created a variable and given it a value, you can use the variable wherever appropriate when supplying values for Pit variables and monitor variables.

Additionally, once defined, the custom variable is added to the dropdown variable list. The list of variables displays during data entry when a # is typed as the sole character in a data field, as shown in the following illustration.



Creating a custom variable

Here are the steps to add a custom variable to Peach:

1. Begin on the Variables page.

This page lists the information needed by the selected pit. Some of the information applies to the Peach environment; two examples are the Peach Installation Directory and the Pit Library Path. Other information is pit specific, such as port addresses for a network protocol or source files for a file format.

Variables

Name	Key	Value
Fuzzed Data File	FuzzedFile	fuzzed.bmp

Save **Add Variable**

- Click the "Add Variable" button immediately above and to the right of list of variables.

The "Add Variable" pop-up dialog displays.

Add Variable

Name	<input type="text" value="Name"/>
Required	
Key	<input type="text" value="Key"/>
Required	
Value	<input type="text" value="Value"/>

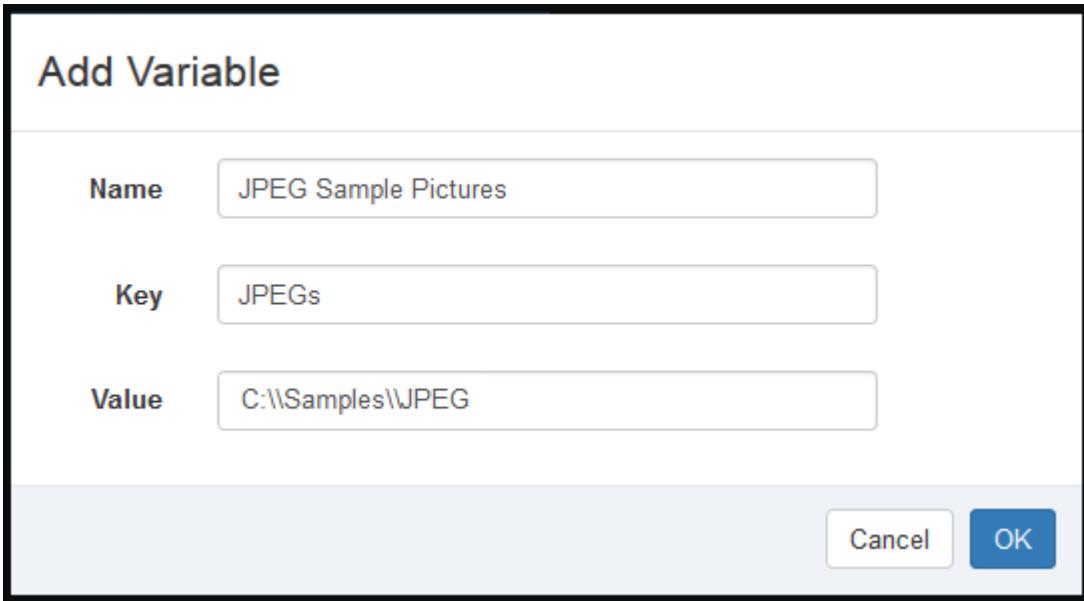
Cancel **OK**

- Fill in values for the name, key, and value fields. The following illustration shows a variable locating a JPEG repository.

- The **Name** parameter is the name of the custom variable, and allows spaces and punctuation. This is the name to search on if you want to edit the value of the variable.
- The **Key** parameter is the value, such as **JPEGs**, that you plug into values in the Peach Web UI.

When Peach processes the pit, the fuzzer replaces the key parameter with the value of the **Value** parameter.

- The **Value** parameter stores the value that Peach uses when processing the pit.



4. Click OK.

5. Click save.

The new value is now part of the list, and has been saved for future use.

This page lists the information needed by the selected pit. Some of the information applies to the Peach environment; two examples are the Peach Installation Directory and the Pit Library Path. Other information is pit specific, such as port addresses for a network protocol or source files for a file format.

Saved successfully.

Name	Key	Value
Fuzzed Data File	FuzzedFile	fuzzed.bmp
Seed File	Seed	*.bmp
Sample Path	SamplePath	##PitLibraryPath##/_Common/Sa

Name	Key	Value	Remove
JPEG Sample Pictures	JPEGs	C:\\Samples\\JPEG	X

> System Defines

Using a custom variable

Using a custom variable consists of typing **##**, the variable name, and **##** in the value of another variable. In the following illustration, the "Sample Path" variable refers to the "JPEG Sample Pictures". When parsing the configuration information, Peach inserts the value **C:\\Samples\\JPEG** for the value of the "Sample Path".

The screenshot shows the Peach Fuzzer web application running at localhost:8888/3.7.50.1/#/pit/5376991a9f3fe90d30876a0. The left sidebar has a navigation menu with items like Pit, Quick Start, Configure, Variables (which is selected), Monitoring, Test, Help, and Forums. The main content area shows the 'Variables' configuration page. At the top, there's a breadcrumb trail: Home / Library / BMP_Demo / Configure / Variables. Below that, the title 'Variables' is displayed. A descriptive text explains that this page lists information needed by the selected pit, including Peach environment variables and pit-specific variables like port addresses. There are two buttons at the top right: 'Save' and '+ Add Variable'. The main content area contains three tables under sections 'All', 'User Defines', and 'System Defines'. The 'All' section contains three rows: 'Fuzzed Data File' with key 'FuzzedFile' and value 'fuzzed.bmp'; 'Seed File' with key 'Seed' and value '* bmp'; and 'Sample Path' with key 'SamplePath' and value '# JPEGS##'. The 'User Defines' section contains one row: 'JPEG Sample Pictures' with key 'JPEGs' and value 'C:\Samples\JPEG'. The 'System Defines' section is collapsed.

Monitoring

The Monitoring data entry screen defines one or more Agents and one or more Monitors for the Pit.

Agents are host processes for monitors and publishers. Local agents can reside on the same machine as Peach, and can control the test environment through monitors and publishers. Remote agents reside on the test target, and can provide remote monitors and publishers.

Monitors are components that perform one or more of the following functions: detect faults (issues),

collect data associated with faults, and help manage the fuzzing job to reduce the level of human interaction throughout the job.

From the Home Page

To begin configuring from the Home Page:

1. Click the Library menu entry.
2. From the Pit Library, select a Pit or an existing configuration.
 - Selecting a configuration means that you are revising the settings of an existing configuration. Peach displays the start screen for the configuration.
 - Selecting a Pit means that you are creating a new configuration. You will need to name the configuration, optionally provide a description, and click "Submit" to reach the start screen for the configuration.

The screenshot shows the Peach Fuzzer web application running at localhost:8888/3.7.50.1/#/pit/5376991a9f3fe90d308. The left sidebar has a dark theme with icons for Home, Library, Pit, Quick Start, Configure, Help, and Forums. The main content area has a light blue header with the Peach Fuzzer logo and title. It displays the path [Home / Library / BMP_Demo](#). A yellow warning box says: "Warning! The currently selected Pit should be configured for monitoring the environment. Pit Configuration Quick Start". Below it, a message states: "This configuration uses the **BMP_Demo** pit and includes configuration data for your test setup." There are three buttons: "Quick Start Wizard", "Configure Variables", and "Configure Monitoring". The "Configure Monitoring" button is highlighted. A "Start Options" section follows, with a "Hide" link. It explains that start options specify test cases for a fuzzing job and includes instructions for replaying runs. It lists effects of changing start options and provides input fields for "Seed" (set to "Random Seed") and "Start Test Case" (set to "1").

3. Click the "Configure Monitoring" button to define or edit agents, monitors, and the data values associated with them. The Monitoring data entry screen displays and is initially empty.

The screenshot shows a web browser window titled "Peach Fuzzer". The address bar displays "localhost:8888/3.7.50.1/#/pit/5376991a9f3fe90d30876a0". The main content area is titled "PEACH FUZZER" and shows the "Monitoring" page. On the left, there is a sidebar with icons for Home, Library, BMP_Demo, Configure, Monitoring, Test, Help, and Forums. The "Monitoring" icon is highlighted. The main content area has a breadcrumb navigation path: Home / Library / BMP_Demo / Configure / Monitoring. Below the path, the title "Monitoring" is displayed. A text block explains: "The Monitoring data entry screen defines one or more Agents and one or more Monitors for the Pit. Agents are host processes for monitors and publishers. Local agents can reside on the same machine as Peach, and can control the test environment through monitors and publishers. Remote agents reside on the test target, and can provide remote monitors and publishers." A yellow warning box contains the message "Warning! No agents have been configured." At the bottom right of the content area are two buttons: "Save" and "+ Add Agent".

The workflow for this data entry screen has you declare an Agent. Then, you can branch out and do the following in any order:

- Declare one or more monitors for the agent
- Fill in details for a monitor
- Switch focus from one monitor to another
- Declare additional agents, as needed
- Switch focus from one agent to another

In this instance, we're going to complete the agent, then add a monitor and fill in the monitor settings.

Specifying an Agent

Here are the steps to add an agent to a configuration:

1. Click the "Add Agent" button. Peach adds a new agent to the Monitors page, as in the following illustration.

An agent has a name and location. The location can be local or remote.

- Local agents run in the same process space as the Peach fuzzing engine.
- Remote agents are separate processes that can reside on the same hardware as the test target. Remote agents act as intermediaries between Peach and test targets, sending test cases to the test target and replying with test case results and data back to Peach.

If you use a remote agent, you need to supply location information that conforms to a URL with the following parts: `channel://host:port`

channel

Specify one of the following for the channel type: *local*, *tcp*, or *http*. Typical remote agent configurations should use the *tcp* channel.

host

Specify the hostname of the agent to be used. This value is not required for the *local* channel.

port

Specify the port number of the agent to be used. This value is not required for the *local* channel.

- Example agent using the `tcp` channel: `tcp://192.168.127.128:9001`

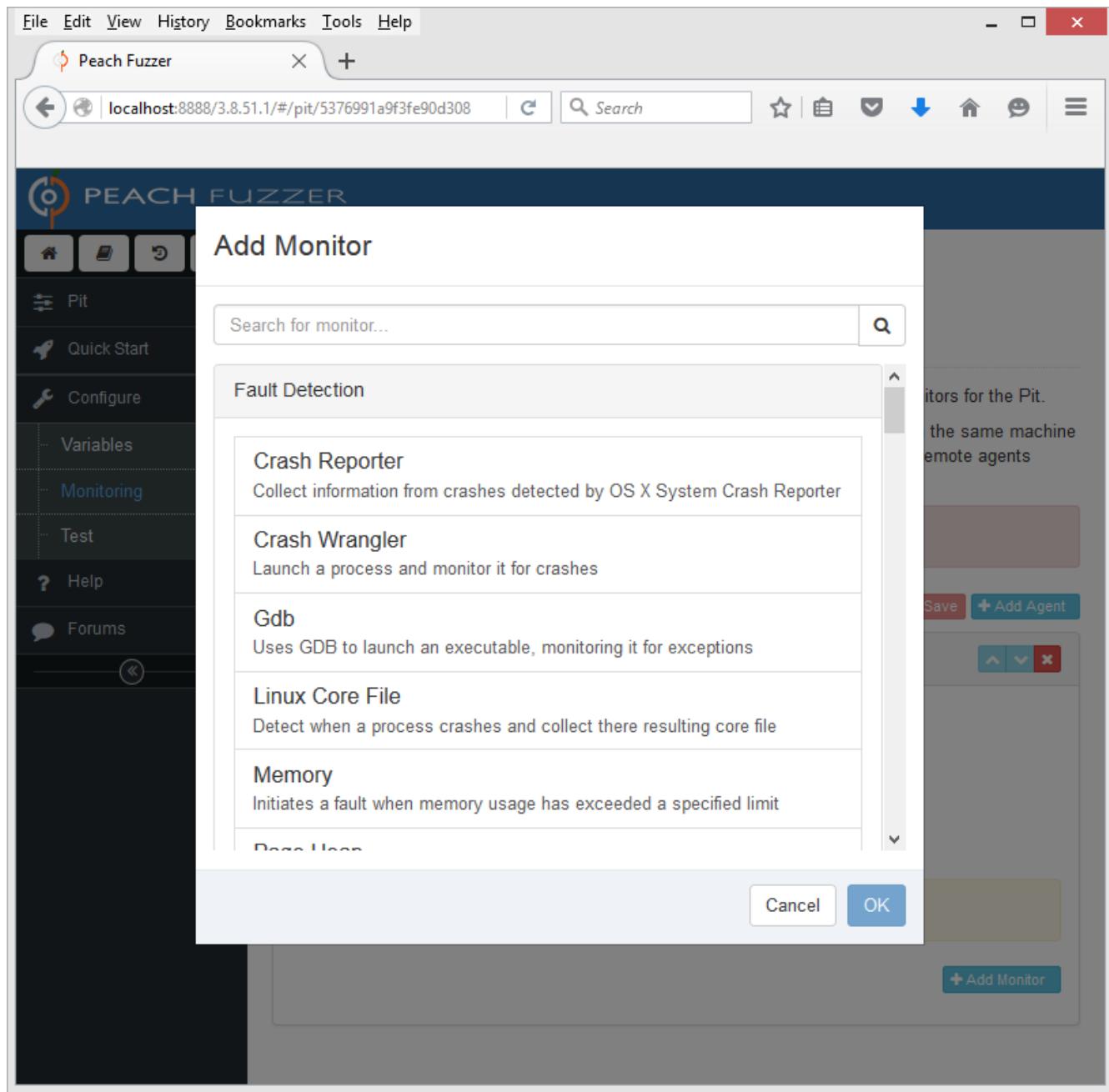
2. Give the agent a name, such as `LocalAgent` and click "Save".

Peach saves the Agent information, provides a visual cue with a "Saved successfully." message in a banner near the top of the page.

Adding a Monitor

1. click the "Add a monitor..." button.

Peach displays a list of monitors that you can use in your configuration. The monitors are categorized by usage. Fault detection monitors appear first, then data collection, automation, android, and lastly, other monitors.



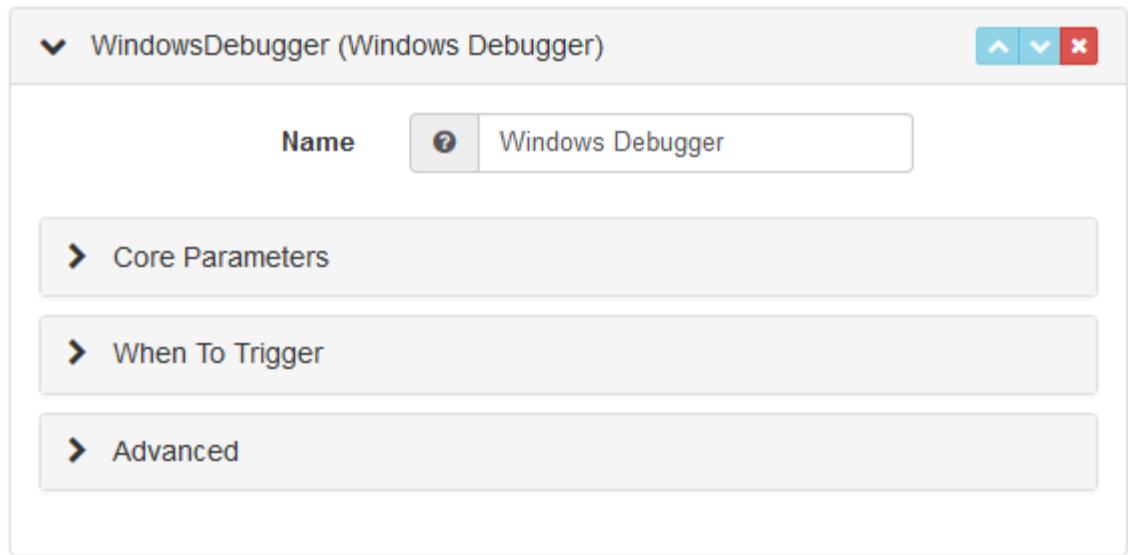
2. Select the **WindowsDebugger** entry, located under the **Fault Detection** section, and click "OK".

The screenshot shows the Peach Fuzzer web application interface. The title bar reads "Peach Fuzzer". The address bar shows the URL "localhost:8888/3.7.50.1/#/pit/5376991a9f3fe90d30876a0". The main content area is titled "Monitoring". On the left, there is a sidebar with icons for Home, Library, BMP_Demo, Configure, Monitoring (which is selected), Test, Help, and Forums. The main content area displays two monitoring agents: "local:// (LocalAgent)" and "WindowsDebugger (Windows Debugger)". Each agent has fields for Name (LocalAgent, Windows Debugger) and Location (local://). Below these, under "Core Parameters", are fields for Executable, Arguments, and Process Name. A "Save" button is located at the top right of the monitoring section.

3. Fill in the details of monitor.

Configuration information for each monitor is available in the [Monitors](#) reference section. A list of monitors appears at the start of the section that links to the individual entries.

The monitor parameters divide into three groups: "Core", "When To Trigger", and "Advanced".

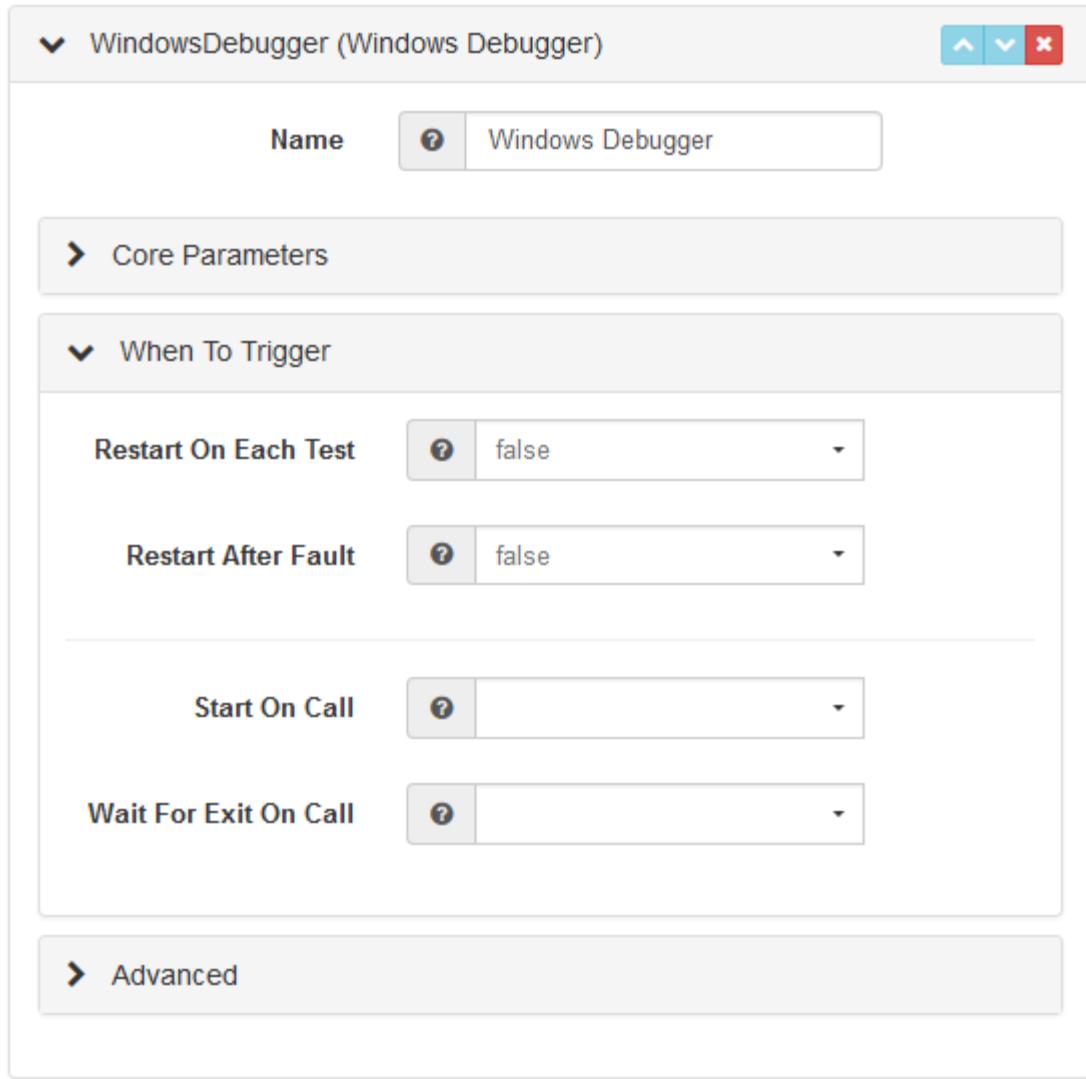


- The "Core" parameters consist of parameters that you should check when creating or editing a configuration. This monitor provides a choice of what to monitor: an executable file, a running process, or a service.

▼ WindowsDebugger (Windows Debugger) ^ ▾ ✕

Name	? Windows Debugger
Core Parameters	
Executable	? <input type="text"/>
Arguments	? <input type="text"/>
Process Name	? <input type="text"/>
Service	? <input type="text"/>
Win Dbg Path	? <input type="text"/>
When To Trigger	
Advanced	

- The "When To Trigger" addresses timing-related issues, such as restarting the test target at the end of an iteration. This is common for file fuzzing, where Peach creates a fuzzed data file, then starts the target with the fuzzed file as input.



- The "Advanced" parameters are items that seldom need to be specified; the default values of these parameters are usually sufficient as is. Yet, once in a while you might need to access one of these parameters. When fuzzing a network client, the "No Cpu Kill" parameter needs to be set to true to give the client an opportunity to close cleanly.

▼ WindowsDebugger (Windows Debugger) ▲ ▼ ×

Name	? Windows Debugger
Core Parameters	
When To Trigger	
Advanced	
No Cpu Kill	? false
Fault On Early Exit	? false
Wait For Exit Timeout	? 10000
Symbols Path	? SRV*http://msdl.microsoft.com
Ignore First Chance Guard Page	? false
Ignore Second Chance Guard Page	? false
Cpu Poll Interval	? 200

Sample Agent with Multiple Monitors

The following illustration is of an agent with multiple monitors. Note that you can show or hide the details for a monitor by clicking the chevron preceding the monitor name. In fact, this is true for agents, too.

The screenshot shows the Peach Fuzzer web interface. The title bar reads "Peach Fuzzer". The URL in the address bar is "localhost:8888/3.7.50.1/#/pit/1137da12d3364fc7". The navigation menu on the left includes links for Home, Library, Pit, Quick Start, Configure, Variables, Monitoring (which is selected), and Test. The main content area is titled "Monitoring". It contains a descriptive text about monitoring agents and monitors, followed by a form for defining an agent. The agent is named "Agent0" and is located at "local://". Below the agent definition, there is a list of four monitors: "PageHeap", "WindowsDebugger", "SaveFile", and "Process". Each monitor entry has up/down arrows and a delete button.

File Edit View History Bookmarks Tools Help

Peach Fuzzer

localhost:8888/3.7.50.1/#/pit/1137da12d3364fc7

Search

PEACH FUZZER

Home / Library / JPG_Test1 / Configure / Monitoring

Monitoring

The Monitoring data entry screen defines one or more Agents and one or more Monitors for the Pit.

Agents are host processes for monitors and publishers. Local agents can reside on the same machine as Peach, and can control the test environment through monitors and publishers. Remote agents reside on the test target, and can provide remote monitors and publishers.

Save + Add Agent

local:// (Agent0)

Name: Agent0

Location: local://

+ Add Monitor

PageHeap

WindowsDebugger

SaveFile

Process

Sample Remote Agent

The following illustration is of a remote agent. The location of the agent is the IP address of the remote machine, and that the address is stored in a custom variable. The agent manages multiple monitors.

The screenshot shows the Peach Fuzzer web application running at localhost:8888/3.7.50.1/#/pit/ed9a357583baa23c. The title bar says "Peach Fuzzer". The left sidebar has icons for Home, Library, IPv4, Configure, and Monitoring, with "Monitoring" selected. The main content area is titled "Monitoring". It displays a description of monitoring agents and monitors, and a configuration form for "tcp://##TargetIPv4## (Agent0)". The form includes fields for "Name" (Agent0) and "Location" (tcp://##TargetIPv4##), and a list of monitors: PageHeap, WindowsDebugger, Pcap, and Process. A "Save" button and a "+ Add Agent" button are at the top right of the configuration form.

Tuning

Tuning allows control of how testing is performed on a field-by-field basis.

Fields are shown hierarchically below. A search feature is provided to quickly find fields of interest. Each field, or set of fields, can be excluded from testing or have its testing focus turned up or down. As a field's focus is turned up (High or Highest), test cases will be generated more often for that field. If a

field is turned down (Low, Lowest), fewer test cases will be generated for that field. Fields that are excluded will not have any test cases generated.

Excluding fields from testing should be used judiciously as it can lead to undiscovered faults.



Tuning is not required. In fact it's recommended to use the defaults and let Peach decide how often to test fields.

From the Home Page

To begin configuring from the Home Page:

1. Click the Library menu entry.
2. From the Pit Library, select a Pit or an existing configuration.
 - Selecting a configuration means that you are revising the settings of an existing configuration. Peach displays the start screen for the configuration.
 - Selecting a Pit means that you are creating a new configuration. You will need to name the configuration, optionally provide a description, and click "Submit" to reach the start screen for the configuration.

The screenshot shows the Peach Fuzzer web application running at localhost:8888/3.7.50.1/#/pit/5376991a9f3fe90d308. The left sidebar has a 'Pit' section expanded, showing 'Quick Start', 'Configure', 'Help', and 'Forums'. The main content area is titled 'BMP_Demo'. A yellow warning box says: 'Warning! The currently selected Pit should be configured for monitoring the environment. Pit Configuration Quick Start'. Below it, text states: 'This configuration uses the **BMP_Demo** pit and includes configuration data for your test setup.' There are three buttons: 'Quick Start Wizard', 'Configure Variables', and 'Configure Monitoring'. A 'Start Options' section explains how to specify test cases for a fuzzing job, mentioning a 'Replay' feature. It lists effects of changing start options and provides input fields for 'Seed' (set to 'Random Seed') and 'Start Test Case' (set to '1').

3. Click the "Configure" menu item on the left of the screen. It will expand to show several configuration options.
4. Click the "Tuning" sub-menu item to access the Tuning page. The Monitoring data entry screen displays and is initially empty.

Peach Fuzzer

10.0.1.57:8888/#/pit/3f8913878cbe2695222e837c3c3d20d3/advanced

PEACH FUZZER

Home / Library / HTTP_Server-Demo / Configure / Tuning

Tuning

Tuning allows control of how testing is performed on a field-by-field basis.

Fields are shown hierarchically below. A search feature is provided to quickly find fields of interest. Each field, or set of fields, can be excluded from testing or have its testing focus turned up or down. As a field's focus is turned up (High or Highest), test cases will be generated more often for that field. If a field is turned down (Low, Lowest), fewer test cases will be generated for that field. Fields that are excluded will not have any test cases generated.

Excluding fields from testing should be used judiciously as it can lead to undiscovered faults.

Search for fields...

Exclude Lowest Low Normal High Highest

- Request-Line
- Method
- URI
- Version
- CRLF
- Headers

Tuning Fields

The Tuning page shows fields in a hierarchy. For file pits such as media formats or application formats, this will be the format of the file being generated. For network pits, the hierarchy will show different packets or messages being transmitted, assuming the protocol has this concept. When modifying a field that contains children, by default all of the children will also be changed. This allows quickly tuning entire messages or sections of a message.

For each field there are several tuning options:

Exclude

Don't perform any testing of this field.



Excluding fields from testing should be used judiciously, as it may lead to missed faults. It's our recommendation that all fields get tested. Instead of excluding fields, consider tuning them to lower or lowest.

Lowest

Produce the least number of test cases for this field. When selected, the field will still receive some testing, but much less than fields marked as normal.

Low

Produce fewer test cases for this field. Fields tuned to low will still be tested, but fewer test cases will be generated compared to fields marked as normal.

Normal

No changes to how this field is tested. Peach will decide how often to generate test cases for fields marked as normal. This is the recommended setting for all fields.

High

Produce more test cases for this field. Fields tuned to high will receive more testing than those marked as normal.

Highest

Produce even more test cases for this field. Fields tuned to highest will receive more testing than those marked as high.

Once you have completed tuning the fields, click the Save button in the upper right of the screen.

Test

In the Test section, Peach performs a test on the selected Pit configuration using settings provided for variables, agents, and monitors. Peach identifies the readiness of the Pit configuration for testing by tracking and reporting the progress of completing settings for the variables, agents, and monitors.

The screenshot shows the Peach Fuzzer application interface. On the left is a sidebar with various navigation links: Home, Library, HTTP_Server-Demo, Configure, Test, Pit, Quick Start, Configure, Variables, Monitoring, Tuning, and Test. The 'Test' link is currently selected. The main content area has a blue header bar with the Peach Fuzzer logo and the word 'PEACH FUZZER'. Below the header, the URL in the address bar is 10.0.1.57:8888/#/pit/3f8913878cbe2695222e837c3c3d20d3/advanced. The main content area has a title 'Test' and a descriptive paragraph: 'This section validates the configuration by executing one test case and exposes issues that would hinder a fuzzing session. The output will display any warnings and errors that surface in the control test case. Detailed log messages are provided to help diagnose issues with the configuration.' Below this text is a message: 'Click the Begin Test button to validate the configuration.' At the bottom right of the content area are two buttons: a green 'Continue' button and an orange 'Begin Test' button.



The Test section runs a single test case without any fuzzing.

The test requires that the target device, service, or application be available for use.



This screen issues a warning if the pit is not configured, but lets the user run the test.

- Click the Begin Test button to run the test.

When the test completes, Peach reports whether the Pit configuration passes the test. If the configuration passes the test, the following message displays:

The screenshot shows the Peach Fuzzer web application running in a browser window titled "Peach Fuzzer". The URL is 10.0.1.57:8888/#/pit/3f8913878cbe269522e837c3c3d20d3/advanced. The left sidebar has a dark theme with white icons and text. It includes links for Pit, Quick Start, Configure, Variables, Monitoring, Tuning, and Test (which is currently selected). Below these are Help and Forums. The main content area has a light blue header with the Peach Fuzzer logo. The breadcrumb navigation shows Home / Library / HTTP_Server-Demo / Configure / Test. The main title is "Test". A descriptive text block explains that this section validates configuration by executing one test case and exposing issues. It also says to click the Begin Test button to validate the configuration. There are two buttons at the bottom: a green "Continue" button and an orange "Begin Test" button. A green message box below the buttons says "Testing passed, click Continue.". The "Test Output (12:30 pm)" section shows a table with a "Summary" tab selected and a "Log" tab. The Log table has a single row under the "Message" column, listing various log entries with green checkmarks.

	Message
✓	Loading configuration file 'c:\pits\output\pits\Assets\Net\HTTP_Server.xml.config'
✓	Loading pit file 'c:\pits\output\pits\Assets\Net\HTTP_Server.xml'
✓	Starting fuzzing engine
✓	Connecting to agent 'local://'
✓	Starting monitor 'SshCommand'
✓	Notifying agent 'local://' that the fuzzing session is starting
✓	Connecting to agent 'tcp://192.168.48.129:9001'

You can start a fuzzing job with your pit.

19.1.4. Fuzzing Session

With your Pit configured and tested, you're ready to start fuzzing!

From the Home Page

1. From the Home screen, click the Library menu.
2. From your Pits Library screen, select a configuration.

The configurations are listed in the section that follows the Pits.

Once selected, the configured Pit displays, as in the following illustration.

File Edit View History Bookmarks Tools Help

Bing × Peach Fuzzer ×

localhost:8888/3.7.57.1/#/pit/889c168ecb45814200992a0b751b318e

 PEACH FUZZER

Home / Library / BMP_1_Simple

BMP_1_Simple

The Pit is configured and ready for use. Click the START button below to begin fuzzing.

This configuration uses the **BMP_1_Simple** pit and includes configuration data for your test setup.

Configuration Options

Set or change the test configuration using the following buttons:

- Quick Start Wizard** Leads a structured question & answer session that covers typical configurations. This choice is recommended for novice users and for simple configurations.
- Configure Variables** A list of items needed by the configuration, including global pit variables and pit-specific information, such as target IP addresses.
- Configure Monitoring** A list of the agents and monitors defined for the test configuration. Click this button to select and update the agents, monitors and associated monitor settings for the test configuration. This choice provides the most flexibility in implementing fault detection, data collection, and automation.

Start Options

The Start Options specify the test cases that start and end a fuzzing job, and identify a seed for generating mutated data. For a new fuzzing job, the default values are usually appropriate to use, although you can select other values to use.

If you want to replay a fuzzing run, you need to use the same Start Options as in the original fuzzing job. You can obtain these values by 1) selecting the fuzzing run you want to replay, and 2) click the Replay button on that job page.

For replay, changing the Start Options has the following effects:

- Change the Seed. This changes the mutated data values used in all test cases. The result is definitely a new job.
- Change the Start or Stop Test Case. This changes the test cases (first and/or last) to execute in the job, thereby lengthening or shortening the total number of test cases executed in the job. Note that changing the test cases (first and last) for the job can help validate an issue or the fix for an issue. However, the result might not produce the intended results.

Click "Start" to begin the fuzzing job.

Seed	Random Seed
Start Test Case	1
Stop Test Case	Default
	

If needed, you can change configuration settings or set some other parameters at the button of the page (typically used in replaying a fuzzing session).

3. Click Start to begin the fuzzing session. The Peach Dashboard displays.

The screenshot shows a web browser window titled "Peach Fuzzer" displaying the "BMP 1" job page. The dashboard interface includes a left sidebar with links for Home, Jobs, Faults, Metrics, Help, Forums, and a navigation bar with icons for File, Edit, View, History, Bookmarks, Tools, and Help. The main content area shows the job name "BMP 1" and a status message "Peach is currently fuzzing...". Below this is a summary table with the following data:

Start Time	10/9/15 3:07PM	Running Time	00h 00m 24s
Test Cases/Hour	3450	Seed	6023
Test Cases Executed	23	Total Faults	0

Below the table are three control buttons: "Start" (green), "Pause" (blue), and "Stop" (red). The "Recent Faults" section is shown below, with a header row containing columns for #, When, Monitor, Risk, Major Hash, Minor Hash, and Download. A message "No data is available" is displayed in this section.

The dashboard allows you to monitor progress as your fuzzing job runs and from it, you can pause, stop, resume, and replay your fuzzing session. The Peach Dashboard provides the following information:

- The Configuration name, above the colored status bar
- The time the job started
- The duration that the job has been running
- The number of fuzzing test cases per hour
- The seed ID for random number generation, so you can replicate the test, if needed
- Number of test cases completed
- Total number of faults found in this run
- A summary of the most recent faults.

NOTES

1. The seed ID influences the fuzzing that occurs during a fuzzing job. If you want to replicate a test, the seed value is required to reproduce the exact sequence of values from the random-number generator used in fuzzing.
2. The STOP button does NOT close Peach. The STOP button only allows you to stop the currently running job.
3. If you have stopped a job and wish to start a new job using a different pit, choose one of the Pits or Pit configurations in your Pit Library. You'll need to re-visit the Home page, and then choose the appropriate entry from the library.
4. The fault summary lists the most recent faults. For information about the faults generated during this fuzzing session, click the Metrics menu, then Faults on the left side of the screen.



From a Configuration Test

When your Pit configuration passes the validation test, the following screen displays with the green banner and the message "Testing Passed, click Continue."

The screenshot shows the Peach Fuzzer web application running in a browser window titled "Peach Fuzzer". The URL in the address bar is 10.0.1.57:8888/#/pit/3f8913878cbe2695222e837c3c3d20d3/advanced. The user is logged in as "Michael". The main content area displays the "Test" configuration page for an "HTTP_Server-Demo" library. The left sidebar includes links for Pit, Quick Start, Configure, Variables, Monitoring, Tuning, and Test. The "Test" link is currently selected. The main content area has a heading "Test" and a descriptive paragraph about validating configuration. It also includes a note to click the "Begin Test" button and two buttons: "Continue" (green) and "Begin Test" (orange). A green message box states "Testing passed, click Continue.". Below this is a section titled "Test Output (12:30 pm)" with tabs for "Summary" (selected) and "Log". The "Summary" tab shows a list of successful messages:

Message
✓ Loading configuration file 'c:\pits\output\pits\Assets\Net\HTTP_Server.xml.config'
✓ Loading pit file 'c:\pits\output\pits\Assets\Net\HTTP_Server.xml'
✓ Starting fuzzing engine
✓ Connecting to agent 'local://'
✓ Starting monitor 'SshCommand'
✓ Notifying agent 'local://' that the fuzzing session is starting
✓ Connecting to agent 'tcp://192.168.48.129:9001'

From here, you can start a fuzzing session with two clicks of the mouse.

1. Click Continue. Peach displays the Pit configuration page.

File Edit View History Bookmarks Tools Help

Bing × Peach Fuzzer ×

localhost:8888/3.7.57.1/#/pit/889c168ecb45814200992a0b751b318e

 PEACH FUZZER

Home / Library / BMP_1_Simple

BMP_1_Simple

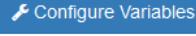
The Pit is configured and ready for use. Click the START button below to begin fuzzing.

This configuration uses the **BMP_1_Simple** pit and includes configuration data for your test setup.

Configuration Options

Set or change the test configuration using the following buttons:

- Quick Start Wizard** Leads a structured question & answer session that covers typical configurations. This choice is recommended for novice users and for simple configurations.
- Configure Variables** A list of items needed by the configuration, including global pit variables and pit-specific information, such as target IP addresses.
- Configure Monitoring** A list of the agents and monitors defined for the test configuration. Click this button to select and update the agents, monitors and associated monitor settings for the test configuration. This choice provides the most flexibility in implementing fault detection, data collection, and automation.

Start Options

The Start Options specify the test cases that start and end a fuzzing job, and identify a seed for generating mutated data. For a new fuzzing job, the default values are usually appropriate to use, although you can select other values to use.

If you want to replay a fuzzing run, you need to use the same Start Options as in the original fuzzing job. You can obtain these values by 1) selecting the fuzzing run you want to replay, and 2) click the Replay button on that job page.

For replay, changing the Start Options has the following effects:

- Change the Seed. This changes the mutated data values used in all test cases. The result is definitely a new job.
- Change the Start or Stop Test Case. This changes the test cases (first and/or last) to execute in the job, thereby lengthening or shortening the total number of test cases executed in the job. Note that changing the test cases (first and last) for the job can help validate an issue or the fix for an issue. However, the result might not produce the intended results.

Click "Start" to begin the fuzzing job.

Seed	Random Seed	
Start Test Case	1	
Stop Test Case	Default	
		

2. Click Start in the "Start Options" section at the bottom of the page to start a fuzzing job.

The Peach dashboard displays and the fuzzing job starts.

The screenshot shows the Peach Fuzzer web interface. The top navigation bar includes File, Edit, View, History, Bookmarks, Tools, and Help. The title bar says "Peach Fuzzer". The address bar shows "localhost:8888/3.7.59.1#/job/9e7a0197-16e1-46cc-ad98-204c6111ae88". The search bar contains "office online". The main content area has a blue header "PEACH FUZZER". On the left is a sidebar with icons for Home, Dashboard, Faults, Metrics (selected), Help, and Forums. The main content area shows "BMP 1" and a message "Peach is currently fuzzing...". Below this is a table with the following data:

10/9/15 3:07PM	00h 00m 24s
3450	6023
Test Cases/Hour	Seed
23	0
Test Cases Executed	Total Faults

At the bottom are three buttons: "Start" (green), "Pause" (blue), and "Stop" (red). Below this is a section titled "Recent Faults" with a table header and the message "No data is available".

Re-running a Fuzzing Job

Peach allows you to re-run fuzzing sessions in whole or in part, meaning that exactly the same tests run using exactly the same data values. If you run an entire fuzzing job, all of the test cases are performed.

If you run a partial job, the test cases you specify run, again in exactly the same order as in the original fuzzing job using exactly the same data values.

What is needed to re-run a fuzzing job?

- Seed value
- Start test case number
- Stop test case number

Supply the seed value and appropriate Start and Stop Test Case values in the "Start Options" on the Pit Configuration page that follows, and click Start to begin a repeat of the fuzzing session.

File Edit View History Bookmarks Tools Help

Bing Peach Fuzzer

localhost:8888/3.7.57.1/#/pit/889c168ecb45814200992a0b751b318e

PEACH FUZZER

Home / Library / BMP_1_Simple

BMP_1_Simple

The Pit is configured and ready for use. Click the START button below to begin fuzzing.

This configuration uses the [BMP_1_Simple](#) pit and includes configuration data for your test setup.

Configuration Options

Set or change the test configuration using the following buttons:

- Quick Start Wizard** Leads a structured question & answer session that covers typical configurations. This choice is recommended for novice users and for simple configurations.
- Configure Variables** A list of items needed by the configuration, including global pit variables and pit-specific information, such as target IP addresses.
- Configure Monitoring** A list of the agents and monitors defined for the test configuration. Click this button to select and update the agents, monitors and associated monitor settings for the test configuration.
This choice provides the most flexibility in implementing fault detection, data collection, and automation.

[Quick Start Wizard](#) [Configure Variables](#) [Configure Monitoring](#)

Start Options

The Start Options specify the test cases that start and end a fuzzing job, and identify a seed for generating mutated data. For a new fuzzing job, the default values are usually appropriate to use, although you can select other values to use.

If you want to replay a fuzzing run, you need to use the same Start Options as in the original fuzzing job. You can obtain these values by 1) selecting the fuzzing run you want to replay, and 2) click the Replay button on that job page.

For replay, changing the Start Options has the following effects:

- Change the Seed. This changes the mutated data values used in all test cases. The result is definitely a new job.
- Change the Start or Stop Test Case. This changes the test cases (first and/or last) to execute in the job, thereby lengthening or shortening the total number of test cases executed in the job. Note that changing the test cases (first and last) for the job can help validate an issue or the fix for an issue. However, the result might not produce the intended results.

Click "Start" to begin the fuzzing job.

Seed: Random Seed

Start Test Case: 1

Stop Test Case: Default

▶ Start

The next two sections provide information about the Seed Value and the Test Case number used in recreating a fuzzing job.

Seed Value

The seed value of a fuzzing job is located in the following places:

- For a running job, the seed is part of the dashboard display when a fuzzing job is running. See entry 2 of the right column of the dashboard.
- For a completed job, click on the Jobs menu and again on the entry in the Jobs list. This brings up the dashboard for the fuzzing job, where the seed is entry 2 of the right column.
- For a completed job, view the generated report, a [.pdf](#) file. The seed is the last table entry in the Summary section (section 1) of the report.

A seed is a common technique used in scientific and computer experiments to provide reproducible results when a random set of data values is needed. The seed feeds into a random-number generator that produces a sequence of "random numbers". Each time the seed is used, the same sequence of "random numbers" is generated.

Peach uses the "random numbers" in determining the mutators to use in a test case, the sequence of mutators, the data elements to fuzz, and the fuzzed data values. Having the seed value guarantees that the same sequence of numbers is generated and used throughout a fuzzing job.

Start/Stop Test Case

The Start Test Case number identifies the first test case to perform in a fuzzing job. The Stop Test Case number identifies the last test case to perform in a fuzzing job. Together, the Start and Stop Test Case numbers identify a range or a sequence of fuzzing test cases to run, whether the fuzzing job is new or is a re-run.

The number of a specific test case is present in the detail or drill-down report for a specific fault. In this report, the value is at the top of the report and is labeled "Iteration". The value represents the iteration number (or test case number) in a fuzzing job.

Main uses of specifying start and/or stop test case numbers in a fuzzing job are to confirm that an issue reliably occurs, to assist in tracking down an issue, or to verify that an issue is fixed.

If the issue does not reproduce, the issue will be more difficult to solve and might be a HEAP-related memory issue in which the addressable memory layout can have a large impact on the bug occurrence. In short, tracking down the root cause and verifying a fix for an issue will require running Peach for a long time to see whether the issue recurs. There is no easy way to guarantee an effective fix in this case.



Once a fix is in place, run a new fuzzing job to regress around the fix and to determine whether any residual faults surface.

19.1.5. Faults

While Peach Fuzzer Professional is running, you can view all the faults generated during the session by clicking the Faults menu option on the left.

Faults displays the total number of generated faults. There are two Faults views: the Summary view and the Detail view:

The screenshot shows the Peach Fuzzer Professional application window. The title bar says "Peach Fuzzer" and the address bar shows "10.0.1.57:8888/#/job/2f086ba2-ad7c-47e3-963f-2ab52933b0e3/faults/all". The main header is "PEACH FUZZER". The left sidebar has icons for Home, Dashboard, Faults (with 13 notifications), Metrics, Help, and Forums. The "Faults" item is selected. The top navigation bar shows "Home / Jobs / HTTP Server-Demo / Faults". The main content area is titled "Faults" and contains a summary: "For each session, the Faults view lists a summary of information about a fault such as:" followed by a bulleted list. Below this is a table of fault details.

#	When	Monitor	Risk	Major Bucket	Minor Bucket	Download
3188	1/31/16 8:58 PM	Gdb	EXPLOITABLE	047C1B7E	B4C61093	Download
5701	1/31/16 9:07 PM	Gdb	EXPLOITABLE	0B0DDD19	310A8210	Download
15739	1/31/16 9:43 PM	Gdb	EXPLOITABLE	047C1B7E	B4C61093	Download
15767	1/31/16 9:44 PM	Gdb	EXPLOITABLE	04EC23BE	C78DF99E	Download
18631	1/31/16 9:55 PM	Gdb	EXPLOITABLE	047C1B7E	244E0C17	Download
18784	1/31/16 9:56 PM	Gdb	EXPLOITABLE	14F145F9	F3F8E68E	Download
20296	1/31/16 10:01 PM	Gdb	EXPLOITABLE	047C1B7E	33E26825	Download
24220	1/31/16 10:15 PM	Gdb	EXPLOITABLE	AC60250E	393FF2B1	Download
30558	1/31/16 10:39 PM	Gdb	EXPLOITABLE	047C1B7E	33E26825	Download
32952	1/31/16 10:49 PM	Gdb	EXPLOITABLE	047C1B7E	33E26825	Download
33692	1/31/16 10:52 PM	Gdb	EXPLOITABLE	14F145F9	00D7125B	Download
37007	1/31/16 11:03 PM	Gdb	EXPLOITABLE	047C1B7E	C1D79156	Download

For each session, the Faults Summary view lists a summary of information about the fault such as:

- Identified fault iteration count
- Time and date
- Monitor that detected the fault
- Risk (if known)

- Unique identifiers of the fault (major and minor hashes), if available

Clicking on one of the listed faults from the Summary view opens the Details view for the selected fault.

The screenshot shows the Peach Fuzzer web application interface. The title bar says "Peach Fuzzer" and the URL is "10.0.1.57:8888/#/job/2f086ba2-ad7c-47e3-963f-2ab52933b0e3/faults/all/41400". The user is logged in as "Michael".

The main content area displays the "Iteration: 41400" details. On the left is a sidebar with navigation links: Dashboard, Faults (13), Metrics, Help, Forums, and a search bar. The "Faults" link is highlighted.

The main content area has the following details:

Test Case	41400				
Reproducible	Yes				
Title	PossibleStackCorruption (7/22), AccessViolation (21/22)				
When	1/31/16 11:20 PM				
Source	Gdb				
Risk	EXPLOITABLE				
Major Bucket	B9973A01				
Minor Bucket	00507C8A				
Tested Fields	<table border="1"> <thead> <tr> <th>Field</th> <th>Mutator</th> </tr> </thead> <tbody> <tr> <td>HTTP:Request.request-line.uri</td> <td>StringLengthVariance</td> </tr> </tbody> </table>	Field	Mutator	HTTP:Request.request-line.uri	StringLengthVariance
Field	Mutator				
HTTP:Request.request-line.uri	StringLengthVariance				

The "Description" section contains a large block of assembly code and stack trace information:

```

'exploitable' version 1.3
Linux ubuntu 3.13.0-29-generic #53-Ubuntu SMP Wed Jun 4 21:00:20 UTC 2014
Signal si_signo: 11 Signal si_addr: 140737488351232
Nearby code:
    0x00007ffff789685e <+1070>: add    rdi,0x40
    0x00007ffff7896862 <+1074>: add    rsi,0x40
    0x00007ffff7896866 <+1078>: movdqu XMMWORD PTR [rdi-0x40],xmm4
    0x00007ffff789686b <+1083>: movaps xmm2,XMMWORD PTR [rsi]
    0x00007ffff789686e <+1086>: movdqa xmm4,xmm2
=> 0x00007ffff7896872 <+1090>: movdqu XMMWORD PTR [rdi-0x30],xmm5
    0x00007ffff7896877 <+1095>: movaps xmm5,XMMWORD PTR [rsi+0x10]
    0x00007ffff789687b <+1099>: pminub xmm2,xmm5
    0x00007ffff789687f <+1103>: movaps xmm3,XMMWORD PTR [rsi+0x20]
    0x00007ffff7896883 <+1107>: movdqu XMMWORD PTR [rdi-0x20],xmm6
Stack trace:
# 0 __strcat_sse2_unaligned at 0x7ffff7896872 in /lib/x86_64-linux-gnu/libc.so.6
# 1 CrashMe at 0x401ffe in /home/mike/httpd
# 2 None at 0x6974736f702f7469 in ?

```

Here's where you can find details about the selected fault. Additional information (such as any files collected during the data collection phase) are located in the disk log folder.

19.1.6. Metrics

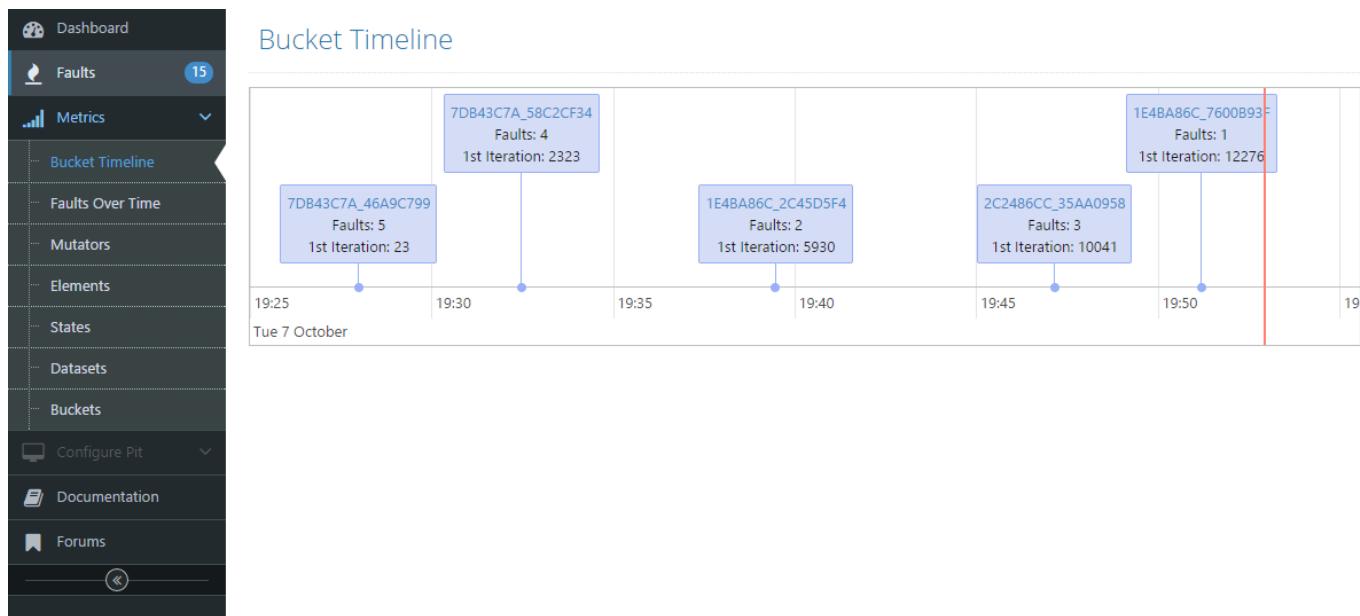
A number of metrics are available for viewing while Peach Fuzzer Professional is running.



The data grids used on many of the metrics displays support multi-column sorting using the *shift* key and clicking on the different columns to sort.

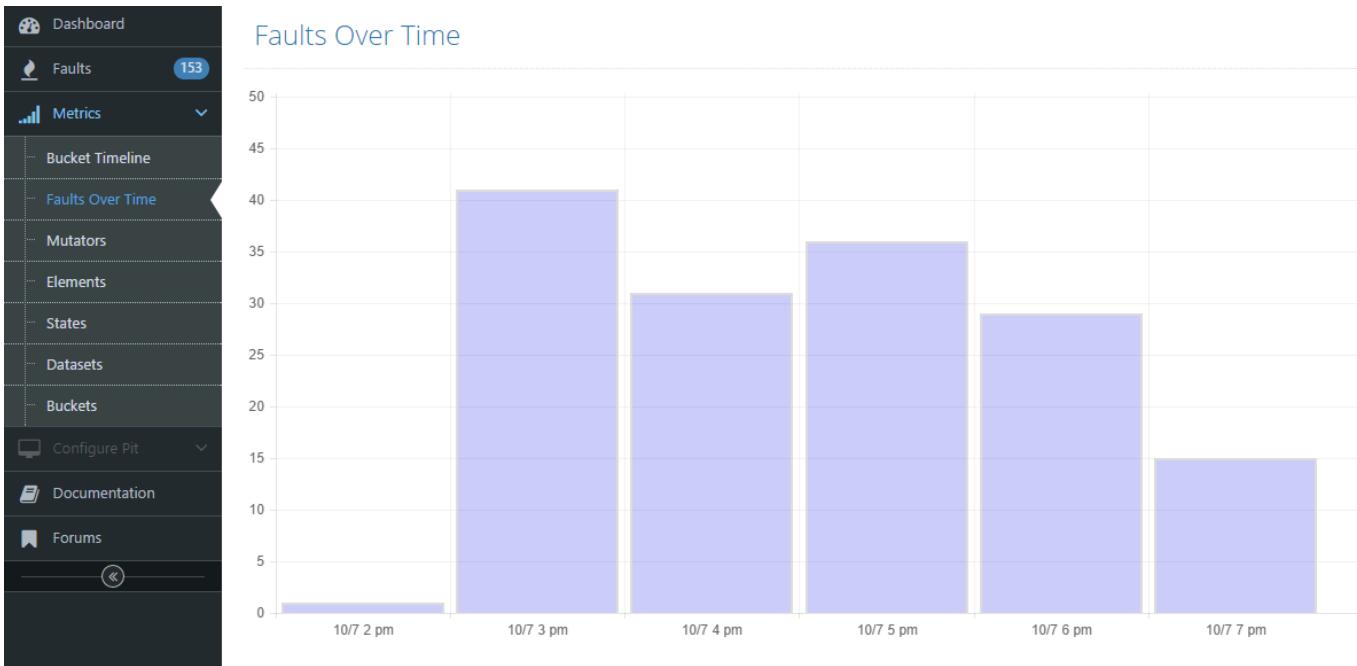
Bucket Timeline

This metric display shows a timeline with new fault buckets listed, and total number of times the bucket was found during the fuzzing session.



Faults Over Time

This metric display shows the count of faults found by hour over the course of the fuzzing run. This is the count of all faults found, not just unique buckets.



Mutators

This metric display shows statistics for each mutator by arranging the information into columns:

Element Count

The number of elements this mutator touched with mutated data.

Iteration Count

The number of iterations this mutator was used during the fuzzing job.

Bucket Count

The number of unique buckets found while this mutator was in use.

Fault Count

The number of faults found while this mutator was in use.

The screenshot shows the PEACH FUZZER application interface. The top navigation bar includes the logo, the title "PEACH FUZZER", and a user dropdown "randofaulter ▾". The left sidebar contains a navigation menu with items: Dashboard, Faults (27), Metrics (selected), Bucket Timeline, Faults Over Time, Mutators (selected), Elements, States, Datasets, Buckets, Configure Pit, Documentation, and Forums. The main content area is titled "Mutators" and displays a table of mutation statistics:

Mutator	Element Count	Iteration Count	Bucket Count	Fault Count
BlobExpandAllRandom	175	1019	2	2
BlobExpandSingleIncrementing	176	1047	2	2
BlobExpandZero	176	1027	2	2
DataElementBitFlipper	139	192	1	1
NumberEdgeCase	80	24893	5	13
NumberRandom	80	6012	5	7
NumberVariance	80	24446	5	20
SizedDataEdgeCase	16	2110	2	3
SizedDataVariance	16	4127	2	4

Elements

This metric display shows statistics for all of the elements in your Pit.

This display shows several columns of information:

State

The state this element belongs to

Action

The action this element belongs to

Parameter

The parameter this action belongs to (if any). Parameters are used only with actions of type *call*.

Element

The full name of the element and its associated DataModel.

Mutations

The number of mutations generated from this element.

Buckets

The number of unique buckets found by sending mutating data to this element.

Faults

The number of faults found from the mutated data sent to this element.

Elements

State	Action	Parameter	Element	Iterations	Buckets	Faults
S1	A2		TheDataModel.Headers.Height	456	1	1
S1	A2		TheDataModel.Length	808	1	1
S1	A3		TheDataModel.Data.Count	325	1	1
S1	A3		TheDataModel.Data.DataSegment.DataSegment_1	15	1	1
S1	A3		TheDataModel.Data.Type	455	1	1
S1	A3		TheDataModel.Headers.Depth	453	1	1
S1	A3		TheDataModel.Headers.Height	459	1	1
S1	A3		TheDataModel.Length	781	1	1

States

This metric display presents statistics that are relevant for pits that have state models with more than two or more states. This display shows the number of times a specific state occurred during the fuzzing session. Seldom-used states might hide issues or indicate a problem.

For example, not all states always execute. If an early-occurring state is fuzzed, the outcome of the fuzzing could prevent states that are used late in the state flow from occurring.



Over time, the number of occurrences for most states should trend towards equality.

States

State	Executions
S1	14640
S2	14640
S3	14640
S4	14639

Data Sets

This metric display shows statistics related to the use of two or more data sets in the fuzzing session. This is useful to determine the origin of unique buckets and also faults in terms of the data sources

used in mutating.

This display shows several columns of information:

Data Set

Name of the data set

Iterations

Number of fuzzing iterations performed using this data set

Buckets

Number of unique buckets found with this data set

Faults

Number of faults found with this data set

Data Set	Iterations	Buckets	Faults
Data	55734	3	19

Buckets

This metric display shows the buckets encountered during the fuzzing job. Several columns of information show:

Fault bucket

Identifier of the fault that occurred

Mutator

The mutator that generated the fault

Iteration count

The number of iterations that used the mutator

Faults count

The number of faults that occurred while using the mutator

Fault Bucket	Mutator	Element	Iteration Count	Fault Count
1E4BA86C	BlobExpandSingleIncrementing	S1.A3.TheDataModel.Data.Da...	3	1
1E4BA86C	NumberEdgeCase	S1.A2.TheDataModel.Headers...	230	1
1E4BA86C	NumberEdgeCase	S1.A2.TheDataModel.Length	212	1
1E4BA86C	NumberEdgeCase	S1.A3.TheDataModel.Headers...	224	1
1E4BA86C	NumberEdgeCase	S3.A2.TheDataModel.Headers...	212	1
1E4BA86C	NumberVariance	S1.A3.TheDataModel.Data.Co...	229	1
1E4BA86C	NumberVariance	S1.A3.TheDataModel.Headers...	198	1
1E4BA86C	NumberVariance	S1.A3.TheDataModel.Length	156	1
1E4BA86C	NumberVariance	S2.A2.TheDataModel.Data.Co...	215	1

Accessing Raw Metrics Data

Each job has its own SQLite database that contains metrics and other information about the job. The database is stored with other log assets under the logs folder in the peach application folder. Each job will have its own folder of assets. While we don't document the database schema, advanced users are welcome to mine the database to utilize the metrics data in different ways. The database format may change between versions, though typically changes are small.

19.2. The Peach Command Line Interface

This is the core Peach application which provides the core fuzzing capabilities and also some utility functions for the custom pit developer. This application can be used to start the Peach Web Application and also to launch fuzzing jobs from the command line.

Some options may be disabled depending on your Peach License options.

Please submit any bugs to support@peachfuzzer.com.

19.2.1. Peach Web Application

Starts Peach and provides a web application for configuring, running, and viewing results of a fuzzing job.

Syntax

```
peach [options]
```

--nobrowser

Disable launching browser on start.

--webport=PORT

Specified port the web application runs on.

--plugins=PATH

Change the plugins folder location. Defaults to the *Plugins* folder relative to the Peach installation.

19.2.2. Fuzzing from Command Line

A fuzzing run is started by specifying the Peach Pit Configuration or Peach XML file and the name of a test to perform.

If a run is interrupted, it can be restarted by providing the last successful test case and the seed of the test session. Use the --skipto and --seed parameters to provide this information.

Syntax

```
peach [options] <PEACH_PIT.xml | PEACH_CONFIG.peach> [test_name]
```

PEACH_CONFIG.peach

Peach Pit Configuration generated by Peach Application.

PEACH_PIT.xml

The Peach Pit XML file.

test_name

Name of test to run (defaults to *Default*).

-1

Perform a single test case

--debug

Enable debug messages. Useful when debugging Peach Pit Files.

-DKEY=VALUE

Define a configuration variable via the command line. Multiple defines can be provided as needed.

Example: -DTargetIPv4=127.0.0.1 -DTargetPort=80

--duration=DUR

Duration of fuzzing run. Peach will run for DUR length of time. Commonly integrating Peach into an automated test cycle or continuous integration environment. Argument format is DD.HH:MM:SS.

Examples

- **--duration=12** Duration of 12 days
- **--duration=0:20** Duration of 20 min
- **--duration=5:00** Duration of 5 hours
- **--duration=1.5:00** Duration of 1 day, 5 hrs

--noweb

Disable the Peach Web Application

--plugins=PATH

Change the plugins folder location. Defaults to the *Plugins* folder relative to the Peach installation.

--polite

Disable interactive console mode

--range=S,F

Perform a range of test cases starting at test case S and ending with test case F. Typically combined with the --seed argument.

--seed=SEED

Set the fuzzing jobs seed. The same seed will always produce the same test case sequence. Should only be set when reproducing a historical fuzzing job. Default is a random seed.

--skipto=NUM

Skip to NUM test case and start fuzzing. Normally combined with --seed to reproduce a specific sequence of test cases.

--trace

Enable even more verbose debug messages.

--webport=PORT

Specified port the web application runs on

19.2.3. Debug Peach XML File

This will perform a single iteration (-1) of your pit file while displaying a lot of debugging information (--debug). The debugging information is intended for custom pit developers.

Syntax

```
peach -1 --debug <PEACH_PIT.xml | PEACH_CONFIG.peach> [test_name]
```

19.2.4. Display List of Network Capture Devices

Display a list of all known devices Peach can perform network capture on.

Syntax

```
peach --showdevices
```

19.2.5. Display Known Elements

Print a list of all known:

- Actions
- Agent Channels
- Analyzers
- DataElements
- Fixups
- Loggers
- Monitors
- Mutation Strategies
- Mutators

- Publishers
- Relations
- Transformers

The list includes any associated parameters along with a description and default values. This can be used to verify that custom extensions are found.

Syntax

```
peach --showenv
```

19.2.6. Peach Agent

The Peach Agent functionality has been moved to a separate executable. See [PeachAgent](#) for more information.

19.2.7. Running Analyzers from Command Line

This functionality has been moved to a separate executable. See [PitTool - Analyzer](#) for more information.

19.2.8. Generate XML Schema File

This functionality has been moved to a separate executable. See [PitTool - Makexsd](#) for more information.

19.2.9. Examples

Example 4. Running a Pit Configuration (.peach)

This example shows how to run a fuzzing job from a configuration file (.peach). The following command line launches Peach and fuzzes using `pit_config.peach` as the configuration file.

```
> peach pit_config.peach
```

Example 5. Running a Pit

This example shows how to run a fuzzing definition. The following command line launches Peach and fuzzes using `pit.xml` (and if it exists, `pit.xml.config`) as the configuration file.

```
> peach pit.xml
```

Example 6. Single Iteration with Debug Output

When testing a definition, we recommend running a single non-mutating iteration with debug output.

The following command line launches Peach and fuzzes using `pit.xml` (and if it exists, `pit.xml.config`) as the configuration file. The command line combines the `-1` and `--debug` arguments to run a single iteration; the debugging information is included in the output. Even more verbose output can be enabled by using `--trace` instead of `--debug`.

```
> peach -1 --debug samples\DebuggerWindows.xml

[*] Test 'Default' starting with random seed 27886.
Peach.Core.Agent.Agent StartMonitor: Monitor WindowsDebugger
Peach.Core.Agent.Agent StartMonitor: Monitor_1 PageHeap
Peach.Core.Agent.Agent StartMonitor: Monitor_2 NetworkCapture
Peach.Core.Agent.Agent SessionStarting: Monitor
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid SessionStarting
Peach.Core.Agent.Agent SessionStarting: Monitor_1
Establishing the listener...
Waiting for a connection...
Peach.Core.Agent.Agent SessionStarting: Monitor_2

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpClientPublisher start()
Peach.Core.Publishers.TcpClientPublisher open()
Accepted connection from 127.0.0.1:51466.
Peach.Core.Publishers.TcpClientPublisher output(12 bytes)
Peach.Core.Publishers.TcpClientPublisher

00000000  48 65 6C 6C 6F 20 57 6F  72 6C 64 21          Hello World!
Received 12 bytes from client.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
```

```
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpClientPublisher output(12 bytes)
Peach.Core.Publishers.TcpClientPublisher

00000000  48 65 6C 6C 6F 20 57 6F  72 6C 64 21          Hello World!

Received 12 bytes from client.
Peach.Core.Publishers.TcpClientPublisher close()
Peach.Core.Publishers.TcpClientPublisher Shutting down connection to 127.0.0.1:4244
Connection closed by peer.
Shutting connection down...
Connection is down.
Peach.Core.Publishers.TcpClientPublisher Read 0 bytes from 127.0.0.1:4244, closing
client connection.
Waiting for a connection...
Peach.Core.Publishers.TcpClientPublisher Closing connection to 127.0.0.1:4244
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid DetectedFault()
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid DetectedFault() - No fault detected
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.TcpClientPublisher stop()
Peach.Core.Agent.Agent SessionFinished: Monitor_2
Peach.Core.Agent.Agent SessionFinished: Monitor_1
Peach.Core.Agent.Agent SessionFinished: Monitor
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid SessionFinished
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _StopDebugger
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _FinishDebugger
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _StopDebugger
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _StopDebugger
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _FinishDebugger
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _StopDebugger

[*] Test 'Default' finished.
```

Example 7. Replay Existing Test Sequence

Once you find a faulting condition, you may want to replicate the exact test (or sequence of tests) to recreate the issue. Peach can reproduce exact test sequences given the following information:

1. Exact version of Peach. This is found in the log file `status.txt`.
2. Seed number used. This is also found in the log file `status.txt`.
3. Same/similar pit file. Data and state models must be the same.
4. If datasets are used, they must be the same set and have the same contents.

7. `status.txt`

Peach Fuzzing Run

=====

```
Date of run: 3/20/2014 1:58:58 PM
Peach Version: 3.1.40.1          ①
Seed: 51816                      ②
Command line: samples\DebuggerWindows.xml
Pit File: samples\DebuggerWindows.xml
. Test starting: Default
```

① Version of Peach used. Must match when reproducing.

② Seed used. Must match when reproducing.

We can use the first command line to skip directly to a specific iteration and start fuzzing. This lets you run a series of iterations starting from a certain point.



The `--seed` argument matches the value from the `status.txt` file.

```
> peach --seed 51816 --skipto 37566
```

We can use the second command line to perform either a specific iteration or a small number of iterations.

```
> peach --seed 51816 --range 37566,37566
```

19.3. PeachAgent

Starts a Peach Agent server process.

A Peach Agent can be started on a remote machine (remote to Peach) to accept connections from a Peach instance. Agents run various utility modules called Monitors and also host remote Publishers. Peach Agents do not need any specific configuration outside of which port to listen on. All configuration is provided by a Peach instance.

19.3.1. Licensing

The Peach Agent server process does not require a license. In a typical deployment, only the machine running the core Peach process (Peach.exe) requires a license.

19.3.2. Syntax

```
peachagent [--port=9001]
```

-h, --help

Display this help and exit

-V, --version

Display version information and exit

-v, --verbose

Increase verbosity, can use multiple times

--plugins=VALUE

Specify the plugins path

--port=VALUE

Port to listen for incoming connections on (defaults to 9001).

--debug

Enable debug messages. Useful when debugging your Peach Pit file. Warning: Messages are very cryptic sometimes.

--trace

Enable even more verbose debug messages.

19.4. Minset

This tool is used when adding additional samples to an existing Pit or creating a custom file fuzzing Pit.

Peach Minset is used to identify the minimum number of sample files required to provide the greatest code coverage for a given target. This process can be distributed across multiple machines to decrease the run time.

There are two steps to the process:

1. Collect traces
2. Compute minimum set coverage

The first step can be distributed and the results collected for analysis by the second step.

19.4.1. Collect Traces

Performs code coverage using all files in the *samples* folder. Collects the .trace files for later analysis. This process can be run in parallel across multiple machines or CPU cores.

Syntax

```
PeachMinset [-k] -s samples -t traces command.exe args %s
```

-k

Kill target when CPU usage drops to near zero. This is used when taking traces of GUI programs.

-s samples

Folder to load sample files from

-t traces

Folder to write traces to

command.exe args %s

Executable and arguments. **%s** is replaced with the path and file name of a file from the samples folder.

19.4.2. Compute Minimum Set Coverage

Analyzes all .trace files to determine the minimum set of samples to use during fuzzing. This process cannot be parallelized.

Syntax

```
PeachMinset -s samples -t traces -m minset
```

-s samples

Folder to load sample files from

-t traces

Folder to write traces to

-m minset

Folder to write minimum set of files to

19.4.3. All-In-One

Both tracing and computing can be performed in a single step.

Syntax

```
PeachMinset [-k] -s samples -m minset -t traces command.exe args %s
```

-k

Kill target when CPU usage drops to near zero. This is used when taking traces of GUI programs.

-s samples

Folder to load sample files from

-m minset

Folder to write minimum set of files to

-t traces

Folder to write traces to

command.exe args %s

Executable and arguments. **%s** is replaced with the path and file name of a file from the samples folder.

19.4.4. Distributing Minset

Minset can be distributed by splitting up the sample files and distributing the collecting of traces to multiple machines. The final compute minimum set coverage cannot be distributed.

19.4.5. Examples

Example 8. Example Run

```
> PeachMinset.exe -s pinsamples -m minset -t traces bin\pngcheck.exe %%  
  
[*] Running both trace and coverage analysis  
[*] Running trace analysis on 15 samples...  
[1:15] Coverage trace of pinsamples\basn0g01.png...done.  
[2:15] Coverage trace of pinsamples\basn0g02.png...done.  
[3:15] Coverage trace of pinsamples\basn0g04.png...done.  
[4:15] Coverage trace of pinsamples\basn0g08.png...done.  
[5:15] Coverage trace of pinsamples\basn0g16.png...done.  
[6:15] Coverage trace of pinsamples\basn2c08.png...done.  
[7:15] Coverage trace of pinsamples\basn2c16.png...done.  
[8:15] Coverage trace of pinsamples\basn3p01.png...done.  
[9:15] Coverage trace of pinsamples\basn3p02.png...done.  
[10:15] Coverage trace of pinsamples\basn3p04.png...done.  
[11:15] Coverage trace of pinsamples\basn3p08.png...done.  
[12:15] Coverage trace of pinsamples\basn4a08.png...done.  
[13:15] Coverage trace of pinsamples\basn4a16.png...done.  
[14:15] Coverage trace of pinsamples\basn6a08.png...done.  
[15:15] Coverage trace of pinsamples\basn6a16.png...done.  
  
[*] Finished  
[*] Running coverage analysis...  
[-] 3 files were selected from a total of 15.  
[*] Copying over selected files...  
[-] pinsamples\basn3p08.png -> minset\basn3p08.png  
[-] pinsamples\basn3p04.png -> minset\basn3p04.png  
[-] pinsamples\basn2c16.png -> minset\basn2c16.png  
  
[*] Finished
```

19.5. Peach Validator

The Peach Validator tool provides a visual view of data models before and after data is cracked into them. It's a useful to verify data is correctly cracking into your model in a visual manner.

Peach Validator is used with the `--debug` parameter to debug custom pit files. When an error does occur some debugging information is provided via Peach Validator; for a more complete picture, review the `--debug` output.

To use Peach Validator:

1. Open the application
2. Using the icons in the toolbar, open the pit file.
3. In the drop down menu, use the toolbar to select one of the data models found in the file.
4. Once you select the data model, it is displayed in the lower half of the window.
5. A sample file can be loaded and cracked by using the right-most file-open icon in the toolbar.

Once the file is loaded, it is automatically cracked into the model. As elements are selected in the lower half of the screen, the byte range is shown in the hex view on top.

19.6. Peach Multi-Node CLI Tool

This tool is used to control multiple Peach instances at once. The tool can be used via the command line or as an interactive tool.

The tool utilizes the Peach REST API to perform all actions.

19.6.1. Installation

Installation of this tool has two steps.

Install Python 2.7

Python v2.7 is recommended. Other versions may also work.

Install dependencies

```
easy_install requests  
easy_install cmd2
```

Populate instances.py

The instances.py file contains a list of all Peach instances that will be controlled from this tool. Instances can be placed into groups. An instance can be part of more than one group.

Configurations are pulled from a master instance configured in instances.py. Typically the master instance is running locally, but it can also be one of the fuzzing instances.



Only one master instance can be configured!

19.6.2. Syntax

```
peachcli  
peachcli "jobs all" quit
```

19.6.3. Commands

jobs: View all jobs

```
jobs <group>  
jobs all
```

Show basic job information from each instance in a group.

Example 9. Example

```
# peachcli.py

| Peach CLI v0.1
| Copyright (c) Peach Fuzzer, LLC

>> jobs all

-- http://192.168.48.128:8888 --
Name          Status      Start           Stop        Count
Faults
-----
-----
HTTP Server-Test    stopped   2016-01-12T23:13:11Z 2016-01-12T23:13:30Z 200
-
HTTP Server-Test    stopped   2016-01-13T00:27:02Z 2016-01-13T00:54:05Z 781
-

-- http://192.168.48.129:8888 --
Name          Status      Start           Stop        Count
Faults
-----
-----
HTTP Server-Test    stopped   2016-01-12T23:13:14Z 2016-01-12T23:13:25Z -
-
HTTP Server-Test    stopped   2016-01-13T00:27:04Z 2016-01-13T00:48:35Z 13400
112

>>
```

pause: Pause a set of jobs

```
pause <pit-config> <group>
pause HTTP_Server-Test all
```

Pause pit configuration jobs running on a specified group.

pull: Pull fault details

```
pull faults <pit-config> <group>  
  
pull faults HTTP_Server-Test all
```

Pull fault details for a specific pit configuration. Faults from all jobs in group will be collected. The resulting faults are placed into a **faults** folder, organized by risk, bucket, and node/test case number.

Example 10. Example

```
# peachcli.py  
  
| Peach CLI v0.1  
| Copyright (c) Peach Fuzzer, LLC  
  
>> pull faults HTTP_Server-Test all  
Pulling faults of HTTP_Server-Test for group all:  
  
Pulling 112 faults from http://192.168.48.129:8888...  
  
>> quit  
  
# ls faults\HTTP_Server-Test\EXPLOITABLE\F0FF8D9D\1395E24B\  
192.168.48.129_13290 192.168.48.129_13309 192.168.48.129_13328  
192.168.48.129_13347 192.168.48.129_13366 192.168.48.129_13385  
192.168.48.129_13291 192.168.48.129_13310 192.168.48.129_13329  
192.168.48.129_13348 192.168.48.129_13367 192.168.48.129_13386  
192.168.48.129_13292 192.168.48.129_13311 192.168.48.129_13330  
192.168.48.129_13349 192.168.48.129_13368 192.168.48.129_13387  
192.168.48.129_13293 192.168.48.129_13312 192.168.48.129_13331  
192.168.48.129_13350 192.168.48.129_13369 192.168.48.129_13388  
192.168.48.129_13294 192.168.48.129_13313 192.168.48.129_13332  
192.168.48.129_13351 192.168.48.129_13370 192.168.48.129_13389  
192.168.48.129_13295 192.168.48.129_13314 192.168.48.129_13333  
192.168.48.129_13352 192.168.48.129_13371 192.168.48.129_13390  
192.168.48.129_13296 192.168.48.129_13315 192.168.48.129_13334  
192.168.48.129_13353 192.168.48.129_13372 192.168.48.129_13391  
192.168.48.129_13297 192.168.48.129_13316 192.168.48.129_13335  
192.168.48.129_13354 192.168.48.129_13373 192.168.48.129_13392  
192.168.48.129_13298 192.168.48.129_13317 192.168.48.129_13336  
192.168.48.129_13355 192.168.48.129_13374 192.168.48.129_13393  
192.168.48.129_13299 192.168.48.129_13318 192.168.48.129_13337  
192.168.48.129_13356 192.168.48.129_13375 192.168.48.129_13394  
192.168.48.129_13300 192.168.48.129_13319 192.168.48.129_13338
```

192.168.48.129_13357	192.168.48.129_13376	192.168.48.129_13395
192.168.48.129_13301	192.168.48.129_13320	192.168.48.129_13339
192.168.48.129_13358	192.168.48.129_13377	192.168.48.129_13396
192.168.48.129_13302	192.168.48.129_13321	192.168.48.129_13340
192.168.48.129_13359	192.168.48.129_13378	192.168.48.129_13397
192.168.48.129_13303	192.168.48.129_13322	192.168.48.129_13341
192.168.48.129_13360	192.168.48.129_13379	192.168.48.129_13398
192.168.48.129_13304	192.168.48.129_13323	192.168.48.129_13342
192.168.48.129_13361	192.168.48.129_13380	192.168.48.129_13399
192.168.48.129_13305	192.168.48.129_13324	192.168.48.129_13343
192.168.48.129_13362	192.168.48.129_13381	192.168.48.129_13400
192.168.48.129_13306	192.168.48.129_13325	192.168.48.129_13344
192.168.48.129_13363	192.168.48.129_13382	192.168.48.129_13401
192.168.48.129_13307	192.168.48.129_13326	192.168.48.129_13345
192.168.48.129_13364	192.168.48.129_13383	
192.168.48.129_13308	192.168.48.129_13327	192.168.48.129_13346
192.168.48.129_13365	192.168.48.129_13384	

push: Push pit configuration

```
push <pit-config> <group>
```

```
push HTTP_Server-Test all
```

The push command will copy a configuration from the master instance to a group of remote instances. The pit configuration name must follow a specific naming convention to use this command. The name has two parts. The first part is the source pit name, for example "HTTP_Server". The second part is the configuration name, for example "Test". They are joined with a hyphen (-). The resulting name would be "HTTP_Server-Test".

status: Status of all related jobs

```
status <pit-config> <group>
```

```
status HTTP_Server-Test all
```

Collect information about all jobs for a specific pit configuration.

Example 11. Example

```
# peachcli.py

| Peach CLI v0.1
| Copyright (c) Peach Fuzzer, LLC

>> status HTTP_Server-Test all
Status of HTTP_Server-Test for group all:

Nodes     Running   Stopped   Paused   Count   Faults
-----
2          0         4         0        14381   112

>>
```

start: Start a new set of jobs

```
start <pit-config> <group>

start HTTP_Server-Test all
```

Start a new job using the specified pit configuration on all instances in the specified group.

stop: Stop a set of jobs

```
stop <pit-config> <group>

stop HTTP_Server-Test all
```

Stop pit configuration jobs running on a specified group.

19.7. Pit Tool

PitTool is provided as an additional set of utilities useful for developing Peach pits.

19.7.1. Syntax

Usage:

```
PitTool.exe <command> [options]
```

Commands:

analyzer	Run a Peach analyzer.
compile	Validate and compile pit into .meta.json and .ninja files.
crack	Crack a sample file.
help	Show help for commands.
makexsd	Generate a peach.xsd file.
ninja	Create a sample ninja database.

General Options:

-h, --help	Display this help and exit
-V, --version	Display version information and exit
-v, --verbose	Increase verbosity, can use multiple times
--plugins=VALUE	Specify the plugins path
--pits=VALUE	Specify the PitLibraryPath.

19.7.2. Commands

analyzer

Run a Peach analyzer.

compile

Validate and compile pit into .meta.json and .ninja files.

crack

Crack a sample file.

makexsd

Generate a peach.xsd file.

ninja

Create a sample ninja database.

19.8. Pit Tool - Analyzer

Run a Peach analyzer.

19.8.1. Syntax

```
PitTool.exe analyzer <analyzer> [options]
```

19.8.2. Analyzers

Asn1

Converts ASN.1 data in Blobs into a full data model.

Json

Generate a data model based on a JSON document.

Postman (beta)

Convert Postman API collection to Peach Pit.

Regex

Break up a string using a regex. Each group will become strings. The group name will be used as the element name.

StringToken

Generate a data model by tokenizing a text document.

Swagger (beta)

Convert Swagger API definition to Peach Pit.

Xml

Generate a data model based on an XML document.

19.9. Pit Tool - Compile

The Pit compiler is used by Pit developers to provide the following features:

- Validation of Pit files against the [Peach XML Schema](#).
- Verification of [PitDefines](#) in [.xml.config](#) files.
- Lint checks to enforce basic rules for [.xml](#) files.
- Generation of [.meta.json](#) files used for [tuning mutations](#) in the Peach Web Interface.
- Generation of [.ninja](#) databases used by the [Sample Ninja mutator](#).

19.9.1. Syntax

Usage:

```
PitTool.exe compile [options] <pitPath>
```

Description:

Validate and compile pit into .meta.json and .ninja files.

compile Options:

--no-verify	Don't verify PitDefines.
--no-lint	Don't perform lint checks.
--no-meta	Don't generate metadata used for tuning.
--no-ninja	Don't generate a sample ninja database.

General Options:

-h, --help	Display this help and exit
-V, --version	Display version information and exit
-v, --verbose	Increase verbosity, can use multiple times
--plugins=VALUE	Specify the plugins path
--pits=VALUE	Specify the PitLibraryPath.

19.9.2. Parameters

PitPath

The path to a PIT.xml.

--no-verify

Don't verify PitDefines.

--no-lint

Don't perform lint checks.

--no-meta

Don't generate metadata used for tuning.

--no-ninja

Don't generate a sample ninja database.

19.9.3. Verify PitDefines

This step enforces the following rules on each PitDefine:

- A name is defined.
- A description is defined.
- A PitDefine is used by the Pit.

19.9.4. PitLint Checks

The following lint checks are provided to enforce basic rules on Pits:

- The Pit must have an xml declaration.
- The first xml element must be `<Peach>`.
- `<Peach>` must have an `xmlns` attribute.
- The `xmlns` attribute must be equal to <http://peachfuzzer.com/2012/Peach>.
- Only a single `<Test>` element is allowed.
- `<Test>` element must have a `maxOutputSize` attribute.
- `<Test>` element must have a `targetLifetime` attribute.
- `<Publisher>` parameters must not be hard-coded, use a PitDefine (suppress with `Allow_HardCodedParamValue`)
- The `class` attribute of a `<Publisher>` element must match the known Publisher plugins.
- `<Publisher>` must not be referenced with deprecated name.
- All required parameters on a publisher must be configured.
- `<StateModel>` must not have unexpected call actions.
- `<StateModel>` must have a `StartIterationEvent` call `<Action>` at the start.
- `<StateModel>` must have a `ExitIterationEvent` call `<Action>` at the end.
- `<Action>` with a `when` attribute must not contain `controlIteration`.
- `<Action>` with a `when` attribute force `<Test>` to have `nonDeterministicActions` attribute set to `true`.
- Elements with a `value` attribute must not contain embedded newline.

Some lint checks may be ignored by using an xml comment with the format:

```
<!-- PitLint: RULE -->
```

The following rules may be ignored:

- Skip_StartIterationEvent
- Allow_WhenNonDeterministicActions
- Allow_WhenControlIteration
- Allow_MissingParamValue=<ParamName>
- Allow_HardCodedParamValue

For example, the following can be used to ignore errors about the `Timeout` and `Filter` params required by the `RawEther` publisher.

```
<Publisher class='RawEther'>
    <!-- Pit is send only, don't need to expose timeouts or filter -->
    <!-- PitLint: Allow_MissingParamValue=Timeout -->
    <!-- PitLint: Allow_MissingParamValue=Filter -->
    <Param name='Interface' value='##Interface##'/>
</Publisher>
```

19.9.5. Generate Tuning Metadata

Peach offers the ability to tune the fuzzing process by allowing users to manually set a relative mutation weight on a field-by-field basis. To accomplish this task, `PitTool compile` is used to generate metadata which includes the structure of the fields in a Pit. This metadata is used by the Peach Web Interface so that the tree of fields can be displayed and manipulated.

19.9.6. Generate Sample Ninja Database

For more information on the sample ninja database, see [PitTool - Ninja](#).

19.10. Pit Tool - Crack

19.10.1. Syntax

Usage:

```
PitTool.exe crack [options] <pitPath> <DataModel> <SamplePath>
```

Description:

Crack a sample file.

General Options:

-h, --help	Display this help and exit
-V, --version	Display version information and exit
-v, --verbose	Increase verbosity, can use multiple times
--plugins=VALUE	Specify the plugins path
--pits=VALUE	Specify the PitLibraryPath.

19.10.2. Parameters

PitPath

Fuzzing definition that refers to sample files.

DataModel

The DataModel used to crack each sample.

SamplesPath

The path to a folder containing the sample files to be used during fuzzing.

19.10.3. Example

```
$ pittool crack \
pits/Net/HTTP_Client.xml \
HTTP:HTTP:Request \
pits/_Common/Samples/Net/HTTP/requests/get.bin
Parsing: 'pits/Net/HTTP_Client.xml'
Looking for data model: 'HTTP:HTTP:Request'
+- DataModel 'Request'
| +- DataModel 'request-line'
| | -- String 'method' [GET]
| | -- String 'sep1' [ ]
| | -- String 'uri' [/]
| | -- String 'sep2' [ ]
| | +- DataModel 'version'
| | | -- String 'http' [HTTP/]
| | | -- String 'major-version' [1]
| | | -- String 'period' [.]
| | | -- String 'minor-version' [1]
| | -- String 'crlf' [\r\n]
| +- Array 'header-array'
| | +- Choice 'header-array_0'
| | | +- DataModel 'host-header'
| | | | -- String 'field-name' [Host]
| | | | -- String 'delim' [:]
| | | | -- String 'field-value' [localhost]
| | | | -- String 'crlf' [\r\n]
| | +- Choice 'header-array_1'
| | | +- DataModel 'misc-header-analyzer'
| | | | -- String 'field-name' [User-Agent]
| | | | -- String 'delim' [:]
| | | | +- Block 'field-value'
| | | | | +- Block 'value'
...
| | | | -- String 'crlf' [\r\n]
| -- String 'header-crlf' [\r\n]
| +- Block 'end'
```

19.11. Pit Tool - Make XSD

Generate XML Schema file (*peach.xsd*). This file is used for pit file validation and also intelliSense in XML editors. The XSD only needs to be generated if custom extensions are added to Peach.

The Peach schema file, *peach.xsd*, provides intelliSense in supported editors (like Visual Studio and oXygen XML Editor). When adding extensions to Peach, generate a new schema file that includes these extensions.

19.11.1. Syntax

Usage:

```
PitTool.exe makexsd
```

Description:

Generate a *peach.xsd* file.

General Options:

-h, --help	Display this help and exit
-V, --version	Display version information and exit
-v, --verbose	Increase verbosity, can use multiple times
--plugins=VALUE	Specify the plugins path
--pits=VALUE	Specify the PitLibraryPath.

19.11.2. Parameters

None

19.11.3. Example

```
> pittool makexsd
Successfully generated peach.xsd
```

The output of this command is a new *peach.xsd* file that contains any custom extensions.

19.12. Pit Tool - Ninja

This tool is used when adding samples to an existing Pit or creating a custom file fuzzing Pit.

Sample Ninja reads sample files used in fuzzing, runs them through the Peach data crackers (which applies sample files to the data models), and places them into a database. During fuzzing, a sample ninja mutator mixes-and-matches file sections to create a new file.



Sample Ninja is not available in the Peach Community version.

The `pittool ninja` command produces and maintains the Sample Ninja database used by the Sample Ninja mutator.

To fuzz using Sample Ninja:

- Generate a Sample Ninja database using `pittool ninja`.
- The Sample Ninja mutator automatically uses the database the next time Peach is run.
- When new or modified sample files are needed, re-run `pittool ninja` to update the database.

The generated database has a name like `PIT.ninja`. This means that if your pit file is `png.xml`, the generated database file is `png.ninja`.

19.12.1. Syntax

Usage:

```
PitTool.exe ninja <pitPath> <dataModel> <samplesPath>
```

Description:

Create a sample ninja database.

General Options:

<code>-h, --help</code>	Display this help and exit
<code>-V, --version</code>	Display version information and exit
<code>-v, --verbose</code>	Increase verbosity, can use multiple times
<code>--plugins=VALUE</code>	Specify the plugins path
<code>--pits=VALUE</code>	Specify the PitLibraryPath.

19.12.2. Parameters

PitPath

Fuzzing definition that refers to sample files.

DataModel

The DataModel used to crack each sample.

SamplesPath

The path to a folder containing the sample files to be used during fuzzing.

19.12.3. Examples

Example 12. Creating a Sample Ninja database

1. Open a new terminal window.
2. Run `pittool < PitPath > < DataModel > < SamplesPath >`.
3. The Sample Ninja database will be generated.

```
> pittool ninja pits\Image\PNG.xml PNG:PNG:File samples_png
Processing: samples_png\ajou_logo.png
Processing: samples_png\apollonian_gasket.png
Processing: samples_png\aquarium.png
Processing: samples_png\baboon.png
...
Processing: samples_png\z00n2c08.png
Processing: samples_png\z03n2c08.png
Processing: samples_png\z06n2c08.png
Processing: samples_png\z09n2c08.png
```

Generated database:

03/17/2014 08:39 PM	9,035	PNG.xml
03/20/2014 03:52 PM	9,651,200	PNG.ninja

Example 13. Adding new samples to an existing Samples Ninja database

1. Put new samples into your samples folder.
2. Open a terminal window.
3. Run `pittool <FilePath> <DataModel> <SamplesPath>`.
4. The new and modified files will be added to the database.

```
> pittool ninja pits\Image\PNG.xml PNG:PNG:File samples_png
Skipping: samples_png\ajou_logo.png
Skipping: samples_png\apollonian_gasket.png
Skipping: samples_png\aquarium.png
Skipping: samples_png\baboon.png
...
Skipping: samples_png\z00n2c08.png
Skipping: samples_png\z03n2c08.png
Skipping: samples_png\z06n2c08.png
Skipping: samples_png\z09n2c08.png
Processing: samples_png\zzzz.png
```

Generated database:

03/17/2014 08:39 PM	9,035	PNG.xml
03/20/2014 03:52 PM	9,651,200	PNG.ninja

20. Reproducing Faults

A fault that occurs in a fuzzing session needs to be reproducible so that it can be investigated, understood, and mitigated. Peach has the following features that aid in reproducing faults that occur during fuzzing:

- Automatic fault reproduction
- Replay the fuzzing session

When a fault is detected at the end of a test case, Peach automatically enters reproduction mode. How Peach implements fault reproduction depends on the type of target being fuzzed.

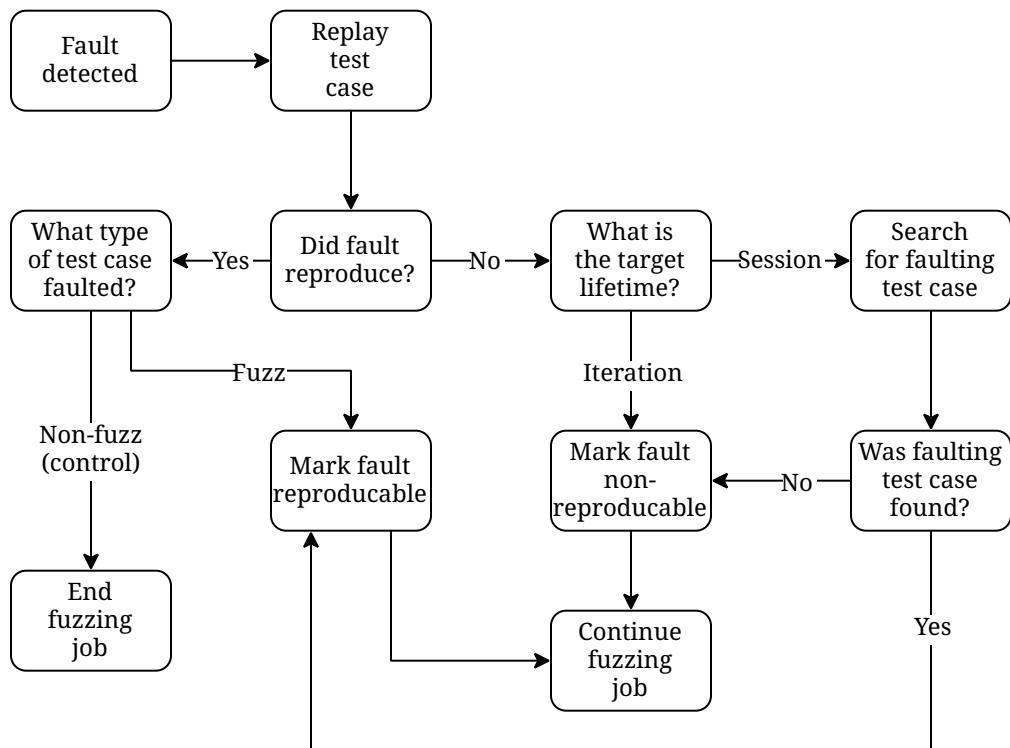


Figure 1. Fault Reproduction Flow Chart

20.1. Iteration Based Targets

Iteration based test targets are targets that restart with every test case. File fuzzing targets are typically iteration based targets. For example, an image viewer that is started during every test case would be considered an iteration target. When reproducing a fault found on an iteration target, Peach will only try the initial faulting test case. Peach will not try and reproduce the fault using previous test cases.



There are rare cases in which an iteration based target retains enough state through a data store that a prior test case may impact the crash. As faults are triaged if you find a significant number of faults that fall into this category it may make sense to reconfigure the Pit to be session based. Currently this is only possible if you have developer access to the pit in question.

20.2. Session Based Targets

Session based test targets are targets that do not restart with every test case. Server or service targets are typically session based targets. For example, an FTP server that is started at the beginning of a fuzzing session would be considered a session target. With session targets, it is possible that a detected fault is the byproduct of previously executed test cases. Therefore, when reproducing a fault found on a session target, Peach will replay sequences of previous test cases. Peach will then locate the exact test case or range of test cases required to reproduce the fault. If a fault is reproduced on a non-fuzz (control) test case that runs after one or more fuzz test cases, Peach will record the range of test cases executed and consider the fault to be reproducible.

20.2.1. Searching For Reproduction

When Peach detects a fault, it always attempts to reproduce the fault. This reproduction involves searching through previously executed test cases. The following steps describe the major elements of the search:

1. Replay the most recent test. This typically is where the fault surfaces.
If the fault reproduces, the system logs the information and the search finishes.
If the test target restarts with each fuzzing iteration, the search finishes because the test started in a known, clean state. Otherwise, continue with step 2.
2. Replay the last 10 iterations, running them in the same sequence as in the fuzzing session.
If the fault reproduces, the system logs the information and the search finishes. Otherwise, continue with step 3.
3. Double the number of iterations, and run them in the same sequence as in the fuzzing session.
If the fault reproduces, the system logs the information and the search finishes. If not, continue the search, each time doubling the number of iterations to run. The criteria for stopping follows:
 - Reproduce the fault.
 - Encounter a critical point in the data; the effort to recreate the fault encountered another fault that peach found and logged during the test session.
 - Encounter a user-specified limit for the search.

Using a search limit of 200, the following sets of iterations would run until the limit is reached:

- 1 iteration
- 10 iterations

- 20 iterations
- 40 iterations
- 80 iterations
- 160 iterations
- 200 iterations (greater than 160 iterations and less than 320 iterations, which is the next cutoff point)

All told, a maximum of 511 iterations would run ($1 + 10 + 20 + \dots + 200$) without human intervention; and some of the iterations would be repeated on subsequent passes.

Some additional considerations about automated fault reproduction include the following:

- Control iterations are treated as in a normal fuzzing session. That is, the results of a control iteration serves as a standard of comparison for the results of record iterations. Also, the frequency of performing a control iteration is determined by the *controlIteration* attribute of the *Test* element.
- Record iterations are still compared to control iterations. If the results of a record iteration matches the results of a control iteration, all is well and good, and processing advances to the next iterations. If the results of a record iteration do not match the results of a control iteration, the results are logged and the search to reproduce the fault ends.

20.3. How to Control the Automated Fault Reproduction

If you use a licensed Peach Pit, the appropriate automated settings are already included in the fuzzing definition.

If you need to change the setting or if you are defining your own pit, the *Test* element in the pit has two attributes that apply to Automated Fault Reproduction: *targetLifetime* and *maxBackSearch*.

- *targetLifetime* is an enumeration that indicates when the test target restarts.
 - session** means that the target restarts at the start of a session; and, when a fault occurs, Peach performs the search to recreate the fault, going back several iterations as needed.
 - iteration** means that the target restarts each iteration and that the search consists of re-running the current iteration.
- *maxBackSearch* indicates the maximum number of iterations to include in the search to reproduce the fault. The default value is 80. In the previous example, the search reaches into the fuzzing session a maximum of 200 iterations that precede the fault in question.



If the test target restarts every iteration (as in file fuzzing), set *targetLifetime* to "iteration".

20.4. Replay the Fuzzing Session

Any test case generated by Peach can be replayed. This allows an engineer to more easily perform root cause analysis of the fault discovered in the device under test. In order for Peach to replay a test case, you need:

- The exact version of Peach
- The pit used in the fuzzing session (the DataModel and the StateModel must be identical to those used in the fuzzing session)
- The seed value used in the fuzzing job
- The test case that caused the fault

With this information, you can re-run the appropriate part of the fuzzing session. For an example, see the [example](#) listed in the [The Peach Command Line Interface](#).

If the command line does not specify an iteration range or a starting iteration, the entire fuzzing session runs.

21. Reference

This section provides reference documentation for Peach.

21.1. General

This section documents general elements used in creating a fuzzing definition.

21.1.1. Peach Workflow

This section will walk through the internal workflow Peach follows when fuzzing. This is provided to give a deeper understanding of how Peach works and when you can expect various activities to occur.

Highlevel Overview

This is the 10,000 ft view of what occurs to perform a fuzzing. This disregards the UI and focus just on the internal workflow.

1. Parse the XML into a DOM (object model). This is when we create objects to represent elements in the pit such as DataModel, Test, Publisher, etc. During this parse time any components of your DataModel such as data elements, fixups and transformers are likely to be called.
2. Run the requested Test. Here we start the fuzzing job which will:
 - a. Find the specified Test
 - b. Connect to all agents and start all monitors
 - c. Initialize some internal context
 - d. Fuzzing loop
 - i. Identify mutations that will occur
 - ii. Generate copy of DOM for use in this test case
 - iii. Run StateModel applying mutations as needed
 - e. As the state model executes it will run actions. Some actions such as start, stop, open, close, accept, input, output, call, set/getProperty are passed directly to the publisher to handle
3. When job completed or is canceled generate the pdf report

Agent, StateModel and Publisher Workflow

This section covers what happens during the "Run StateModel applying mutations as needed" step from the prior section.

Peach is for the most part synchronous in operation. The only asynchronous calls occur in the publishers themselves. For instance, a base publisher exists to deal with Stream objects which makes use of the async read/write methods. That async behavior is entirely contained in the publisher itself. All interactions between Agent, StateModel and Publishers are performed via synchronous blocking calls.

The interaction between the action type *input* may not appear this way as we offload the actual reading of bytes into the data cracker via the Publishers "WantBytes" call. So for the *input* action you will see something like this if you are the publisher:

```
publisher.input()
loop until data cracker done:
    publisher.wantBytes(count)
```

Because some of the publishers have an async read or a worker thread handling a read loop you may see data coming in and displayed in Hex (with --debug) prior to the *input* action occurring. In that case the publisher is buffering received data until it is needed.

A fuzzing iteration would look like this:

```
agent.iterationStarting()

stateModel.Run()          ①
    initialState.Run()
        Loop over actions:
            action.Run()

agent.iterationFinished()
agent.detectedFaults()
if fault then
    agent.GetMonitorData()

agent.mustStop()
```

① Once inside of stateModel.Run() the only agent interaction would be broadcasting the events when using action type call where publisher is **Peach.Agent**. In that case you will see an "agent.message(event)" occur.

21.1.2. Data

At the start of Pit processing, all data sets are empty or have their values defined in the data model (using the *value* attribute). Peach uses the *Data* element to override a *value* attribute or to assign a value to create and load the data into a DataModel.

Peach uses the cracking subsystem to map the data to the correct elements. This process fails if the data does not fit.

Data is cracked in a DataModel the same way that we crack an *input* Action in a StateModel. Currently, the cracking subsystem uses three ways to load data:

- using values from a file
- via a python expression
- by setting values field by field

The *Data* element is a valid top-level element (child of `<Peach>`) when defining data sets that will be referenced during later processing. It is also a valid child of [Action](#) and [Param](#) when used to define data or to reference an existing top level definition.

Data and *switchCount*

If you specify multiple XML blocks in a data file, use the *switchCount* attribute with the [random strategy](#) to specify the number of iterations performed before switching to the next data file.

If there are multiple *Data* elements, after a specified number of fuzzing iterations (using *switchCount*), Peach switches the data element that it uses to populate the fuzzing file.

The following example illustrates this using two *Data* elements. Peach fuzzes with the first *Data* element for the *switchCount* number of iterations. Then, Peach switches *Data* elements and continues fuzzing using the second *Data* element.

```
<DataModel name="MyDataModel">
  <Block name="FooBlock">
    <String name="Value" />
  </Block>
</DataModel>

<Data name="HelloWorldDataSet">
  <Field name="FooBlock.Value" value="Hello World!" />
</Data>

<Data name="LoadFromFile" fileName="sample.bin" />
```

Attributes

Required:

None.

Optional:

name

Name of the Data.

fileName

Name of file to load, or folder with files to cycle through.

Child Elements

- [Field](#)

Examples

Example 14. Basic Usage Example

Multiple Data elements switch after *switchCount* iterations.

```
<StateModel name="TheState">
  <State name="initial">
    <Action type="output">
      <DataModel ref="TheDataModel" />
      <Data name="SampleData1">
        <Field name="Block1.Value" value="Hello World!" />
      </Data>
      <Data name="SampleData2">
        <Field name="Block1.Value" value="Good Afternoon World!" />
      </Data>
    </Action>
  </State>
</StateModel>
```

21.1.3. Defaults

The *Defaults* element changes the default values of the specified data-element attributes.



Items specified in the *Defaults* section must be optional element attributes.



Required attributes do not belong in the *Defaults* section. Instead, place required attribute values in the definition of each element.

Syntax

```
<Defaults>
  <Number endian="big" signed="false" />
</Defaults>
```

Examples

Example 15. Changing default Byte Order

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <Defaults>
        <Number endian="big" signed="false" />
    </Defaults>

    <DataModel name="TheDataModel">
        <Number size="32" value="0xffff" />
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState">
        <State name="InitialState">
            <Action type="output">
                <DataModel ref="TheDataModel" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel" />
        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>

```

Produces the following output:

```

[*] Test 'Default' starting with random seed 54950.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(4 bytes)
00000000  00 00 FF FF                         ****
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

21.1.4. Import

The Import element applies to only Python files. It allows you to import custom Python modules to use in your Pit file.

Import works like the Python import keyword.

Syntax

```
<Import import="MyCode" />
```

Attributes

Required:

import

Same as python import keyword. A file must exist with a `.py` postfix.

Optional:

None.

21.1.5. Include

The Include element allows you to include other Pit files in namespaces (the name that is prepended to everything in the pit) so they can be used by the current Pit file.

When referencing included Pit files, prefix element names with the namespace and a colon (:)
foo:datamodel.

Syntax

```
<Include ns="foo" src="file:foo.xml" />
```

Attributes

Required:

ns

Namespace prefix.

src

Source URL, use "file:" prefix for filenames.

Optional:

None.

Examples

Example 16. Basic Include Example

This example uses the Include element to use a data model defined in another file.

8. example_include.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String value="Hello From Example_Include.xml!\n" />
    </DataModel>
</Peach>
```

9. example.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <Include ns="ex" src="example_include.xml" />

    <StateModel name="TheStateModel" initialState="InitialState">
        <State name="InitialState">
            <Action type="output">
                <DataModel ref="ex:TheDataModel" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel" />
        <Publisher class="Console"/>
    </Test>
</Peach>
```

Produces the following output:

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 35703.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(32 bytes)
Hello From Example_Include.xml!
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

21.1.6. Loggers

Loggers allow adding additional logging methods into Peach. Every Peach fuzzing job will always receive the default Peach logger. Typical reasons for adding custom loggers include:

- Integration into end-user systems
- Custom notification (SMS, Email, Pager)



It may be easier to use the Peach REST API for performing integration work.

File

The file system logger needs a specified path. Peach creates a folder containing the run name and time stamp. Inside of this folder are the actual logs. Until the first fault emerges, Peach logs very little information to conserve disk space.



The source to this monitor is provided in the SDK as an example.

```
<Logger class="File">
    <Param name="Path" value="logfolder" />
</Logger>
```

Parameters

Required:

Path

The relative or absolute path to create log files.

Optional:

None.

21.1.7. Pluggable Mutation Strategies

Peach has pluggable mutation strategies to deal with two common situations:

- Peach usually fuzzes DataModels using a top-down, sequential method. While this guarantees that every data element is fuzzed with each test case, this may not be the best solution for larger, complex systems that can produce millions of test case variations.
- Peach needs a simple mechanism to allow you to change how fuzzing is performed. This helps you determine the best fuzzing methods and strategies for your situation.

If you implement a single C# class, you can fully control how Peach fuzzes a target (including state transitions) because:

- The C# class has full control over data mutations and state transition mutations.
- Data mutations are changing the value of a data element.
- You can easily change the state. If the pit says to transition to state named "RxPacket1," the mutation strategy can change the state so Peach transitions to the state named "TxPacket4" instead of state "RxPacket1".

Peach supplies three user-extensible mutation strategies:

- Random (default)
- Sequential
- RandomDeterministic

Random

Random is the default fuzzing strategy.

This strategy selects one or more elements to mutate at a time. For each selected element, Peach randomly selects a mutator appropriate for that element. The *MaxFieldsToMutate* sets the maximum number of elements that simultaneously receive mutations.



Peach derives the randomness of these selections from a randomly-generated seed.

You can repeat a test a test run using identical values and mutations in sequence by passing the same **seed** value with the Peach --seed command line option. This option is useful for replaying fuzzing iterations to reproduce a previous Fault.

This strategy is most useful for larger data models or for use after performing a sequential fuzzing run.



The random strategy can have an infinite number of fuzzing iterations; it can run forever.

Parameters

MaxFieldsToMutate

Maximum fields to mutate at once. The default value is 6.

SwitchCount

Number of iterations to perform before switching **Data** sets. The default value is 200.

Examples

```

<Test name="Default">
  <StateModel ref="TheStateModel"/>

  <Publisher name="writer" class="File">
    <Param name="FileName" value="fuzzed.tmp"/>
  </Publisher>

  <Strategy class="Random">
    <Param name="MaxFieldsToMutate" value="15" />
    <Param name="SwitchCount" value="100" />
  </Strategy>
</Test>

```

Sequential

If you select the sequential strategy, Peach fuzzes each element in the DataModel in order, one at a time. Peach starts from the top of the DataModel and applies all valid mutations to each data element until all possible mutations have been used.

The sequential strategy has a finite number of fuzzing iterations.



The seed for this strategy is not configurable and is always 31337.

Examples

```

<Test name="Default">
  <!-- ... -->
  <Strategy class="Sequential" />
</Test>

```

RandomDeterministic

This fuzzing strategy is deterministic (has a start and end). It is similar to the Sequential strategy in that Peach systematically works through the DataModel from top to bottom, and for each data element, Peach applies all valid mutations before moving to the next data element. Unlike the Sequential strategy, the RandomDeterministic strategy shuffles the order of mutations for each element.

The RandomDeterministic strategy provides consistent repeatability using a seed value that is useful for repeating a previous fuzzing session with identical fuzzing.

21.1.8. Param

The *Param* child element specifies a configuration setting for its parent element. The structure of a *Param* is a key-value parameter pair. *Params* are common elements used by [Monitors](#), [Publishers](#), and [Fixups](#).



A second usage of Param is with the Action_Call statement. This usage follows the attribute descriptions.

```
<Publisher class="Tcp">
  <Param name="Host" value="127.0.0.1" />
  <Param name="Port" value="80" />
</Publisher>

<Fixup class="Crc32Fixup">
  <Param name="ref" value="MyDataStuff" />
</Fixup>

<Monitor class="WindowsDebugger">
  <Param name="Executable" value="CrashableServer.exe" />
  <Param name="Arguments" value="127.0.0.1 4244" />
</Monitor>
```

Attributes:

Required:

name

Name of parameter.

value

Default value.

Optional:

valueType

Format of value attribute.

Child Elements

None.

Param and the Call Action

Param can also be an argument to the call Action. Call actions can have an array of parameters (defined by Param) used in different ways by the Publisher.

In this usage, the element is solely a container for a DataModel and Data. The Data described in Param is fuzzed by Peach.

Syntax

```
<Action type="call" method="Players[1].OpenUrl">
  <Param name="P1">
    <DataModel ref="TheDataModel" />
    <Data>
      <Field name="Value" value="video.mov"/>
    </Data>
  </Param>
</Action>
```

Attributes

Required:

name

Name of parameter

Optional:

None.

Child Elements

DataModel

Reference to a DataModel that acts as a source for fuzzed data.

Data

Set of initial data to be cracked into the above DataModel before fuzzing.

21.1.9. PythonPath

The *PythonPath* top-level element adds a path to the Python module search path list. It extends Peach and includes custom code locations.

Syntax

```
<PythonPath path="c:/peach/mycode">
```

Attributes

Required:

path

The path to add.

Optional:

None.

21.1.10. Test

The Test element defines the tests that Peach runs during a fuzzing session. Peach requires at least one Test element for fuzzing to occur.

Test configures a specific fuzzing occurrence that combines a StateModel with a Publisher and other configuration options (such as including/excluding elements from being mutated, Agents, and fuzzing strategies).

One pit can contain multiple Test elements; simply provide the test element names on the Peach command line.



If the command-line for launching Peach does not include a test name, Peach uses the Test element named "Default" for the fuzzing session.

Syntax

```

<Test name="Default">

    <!-- Optionally exclude some elements from mutation -->
    <Exclude xpath="//Reserved" />
    <Exclude xpath="//Magic" />

    <!-- Optional agent references -->
    <Agent ref="LocalWindowsAgent" platform="windows" />
    <Agent ref="LocalOsxAgent" platform="osx" />
    <Agent ref="LocalLinuxAgent" platform="linux" />

    <Agent ref="RemoteAgent" />

    <!-- Indicate which state model to use (required) -->
    <StateModel ref="TheState" />

    <!-- Configure the publisher to use (required) -->
    <Publisher class="Tcp">
        <Param name="Host" value="127.0.0.1" />
        <Param name="Port" value="9001" />
    </Publisher>

    <!-- Use a different fuzzing strategy -->
    <Strategy class="Random" />

    <!-- Log output to disk -->
</Test>

```

Attributes

Required:

name

Name of the test, use "Default" for the default test.

Optional:

waitTime

Time to wait between adjacent test cases—The default value is zero (0).

faultWaitTime

Time to wait for a fault to occur before starting next iteration. The default value is zero (0).

controlIteration

Specifies the number of test cases that run before Peach performs the next control iteration. The default value is 0, a special value that means Peach is not using control iterations.

targetLifetime

Specifies when the target restarts to a known, stable state.

- "session" indicates the original target is used throughout the entire fuzzing session. This is the default value.
- "iteration" indicates the target restarts every iteration. This is common in file fuzzing.

maxBackSearch

Sets the maximum number of iterations to include in the search to reproduce a fault. This attribute is used when *targetLifeTime* is "session". The default value is 80.

For example, if the default value is used, the search potentially re-runs the 80 iterations that precede the most recent fault.

maxOutputSize

Sets the maximum size of data that Peach generates. This is on an Action by Action basis, meaning that if your StateModel contains two actions, each action can produce data up to maxOutputSize. The default value is unlimited.

When using Publishers that have a maximum output size, such as UDP, it's recommended this attribute be used to set maximum size to correspond with this limit.

nonDeterministicActions

If true, checks actions in the state model flow for exceptions.

If false, checks actions in the state model flow for exceptions and fully analyzes the state model for consistency. The value false specifies legacy behavior and is the default.

Child Elements

Required:

- [StateModel](#)
- [Publisher](#)

Optional:

- [Agent](#)
- [Include](#)
- [Exclude](#)
- [Strategy](#)
- [Logger](#)
- [Mutators](#)

21.1.11. Exclude

When used in the *Test* element of a Pit, the **Exclude** and **Include** elements allows for including or excluding various data elements from mutation. One use case of these elements is to fuzz encapsulated data without fuzzing the container.

By default, Peach fuzzes all data elements. **Exclude** prevents Peach from fuzzing all or specific data elements. XPath is used to identify the excluded elements.



Even though a data element is excluded, it still may appear to be fuzzed. This is due to other elements being fuzzed. However, the excluded element does not have any specific mutations performed directly to it.

XPath Further Reading

The following links provide additional information regarding XPath.

- [XPath Tutorial](#)
- [XPath Syntax](#)
- [XPath Specification](#)

Syntax

```
<!-- Exclude all data elements -->
<Exclude />

<!-- Exclude specific elements -->
<Exclude xpath="//Value" />
```

Attributes

Required:

None.

Optional:

xpath

Provides an xpath query to select elements to exclude.



Peach interprets the statement `<Exclude />` as all exclusive, and fuzzes no data elements.

Examples

Example 17. Exclude via xpath

This example selects elements to exclude based on an xpath.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

  <DataModel name="TheDataModel">
    <String value="1" />
    <String value="2" />
    <String value="3" />
    <String value="4" />
    <String value="5" />

    <Block name="NotThese">
      <String value="6" />
      <String value="7" />
      <String value="8" />
    </Block>
  </DataModel>

  <StateModel name="TheStateModel" initialState="InitialState">
    <State name="InitialState">
      <Action type="output">
        <DataModel ref="TheDataModel" />
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheStateModel" />
    <Publisher class="ConsoleHex"/>S

      <Exclude xpath="//NotThese" />
    </Test>
  </Peach>
```

Example 18. Exclude and Include

This example excludes all data elements, then includes specific ones.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

  <DataModel name="TheDataModel">
    <String value="1" />
    <String value="2" />
    <String value="3" />
    <String value="4" />
    <String value="5" />

    <Block name="FuzzJustThese">
      <String value="6" />
      <String value="7" />
      <String value="8" />
    </Block>
  </DataModel>

  <StateModel name="TheStateModel" initialState="InitialState">
    <State name="InitialState">
      <Action type="output">
        <DataModel ref="TheDataModel" />
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheStateModel" />
    <Publisher class="ConsoleHex"/>

    <Exclude />
    <Include xpath="//FuzzJustThese" />
  </Test>
</Peach>
```

21.1.12. Include

When used in the *Test* element of a Pit, the **Include** and **Exclude** elements allows for including or excluding various data elements from mutation. One use case of these elements is to fuzz encapsulated data without fuzzing the container.

XPath is used to identify the included elements.

XPath Further Reading

The following links provide additional information regarding XPath.

- [XPath Tutorial](#)
- [XPath Syntax](#)
- [XPath Specification](#)

Syntax

```
<!-- Include all data elements -->
<Include />

<!-- Include specific elements -->
<Include xpath="//Value" />
```

Attributes

Required:

None.

Optional:

xpath

Provide an xpath query to select elements to include.



Peach interprets the statement `<Include />` as all-inclusive, and fuzzes all data elements. === Examples

Example 19. Include via xpath

This example excludes all data elements, then includes specific ones.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

  <DataModel name="TheDataModel">
    <String value="1" />
    <String value="2" />
    <String value="3" />
    <String value="4" />
    <String value="5" />

    <Block name="FuzzJustThese">
      <String value="6" />
      <String value="7" />
      <String value="8" />
    </Block>
  </DataModel>

  <StateModel name="TheStateModel" initialState="InitialState">
    <State name="InitialState">
      <Action type="output">
        <DataModel ref="TheDataModel" />
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheStateModel" />
    <Publisher class="ConsoleHex"/>

    <Exclude />
    <Include xpath="//FuzzJustThese" />
  </Test>
</Peach>
```

21.1.13. Mutators

The Mutators element, a child element of Test, specifies either a list of mutators to include in a fuzzing test run or a list of mutators to exclude from a fuzzing test run.

By default, Peach includes all mutations in a fuzzing run.

When specifying mutators to include in a test, Peach uses the supplied list of mutators in the test. If a mutator name is not listed, peach does not include that mutator in the test run.

When specifying the mutators to exclude from a test, Peach uses all mutators in the test except those supplied in the list. If a mutator name is not listed, peach keeps the mutator included in the test session.



A test can include at most one list of mutators: an include list or an exclude list.



The mutator names are listed in the Peach DOM, accessible by running the **peach --showenv** command.

Syntax

```
<Test>
  <Mutators mode="include_or_exclude">
    <Mutator class="Xyz" />
  </Mutators>
</Test>
```

Attributes

Required:

mode

Either "include" or "exclude" must be specified. The list of mutators must contain at least one mutator entry.

Optional:

None.

Examples

Example 20. Test a single mutator

This example uses a single mutator in the test run.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String type="utf16" value="1" />
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState">
        <State name="InitialState">
            <Action type="output">
                <DataModel ref="TheDataModel" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel" />
        <Publisher class="ConsoleHex"/>

        <!-- Use StringUtf16BomLength mutator in the test. -->
        <!-- No other mutator is in the test run. -->

        <Mutators mode="include">
            <Mutator class="StringUtf16BomLength" />
        </Mutators>
    </Test>

</Peach>
```

Example 21. Omit a single mutator from testing

This example excludes one mutator from the test run.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String type="utf16" value="2" />
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState">
        <State name="InitialState">
            <Action type="output">
                <DataModel ref="TheDataModel" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel" />
        <Publisher class="ConsoleHex"/>

        <!-- Exclude the StringCaseLower mutator from the test. -->
        <!-- All other mutators are in the test run. -->

        <Mutators mode="exclude">
            <Mutator class="StringCaseLower" />
        </Mutators>
    </Test>

</Peach>
```

21.1.14. Web Path

The *Path* child element specifies an HTTP path identifier used by the parent `web` action. All *Path* elements must have a matching substitution identifier in the `web` action URL. The identifier must match the value of the *Path id* attribute.

```
<Action type="web" method="GET" url="http://www.google.com/product/{id}">
    <Path key="id" value="100"/>
</Action>
```

Attributes:

Required:

key

Substitution identifier

Optional:

name

Name of parameter. When not specified, a sanitized version of *key* will be used to generate a name. The *name* field is shown in the job metrics.

value

Default value.

valueType

Format of value attribute.

mutable

Mutable

Child Elements

DataModel

Reference to a DataModel that acts as a source for fuzzed data.

Data

Set of initial data to be cracked into the above DataModel before fuzzing.

Examples

Example 22. Simple Example

A simple web api request with a single *Path* element.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<StateModel name="Default" initialState="FirstState">
    <State name="FirstState">

        <Action type="web" method="GET" url="http://www.example.com/product/{id}">
            <Path key="id" value="1"/>

            <Response />
        </Action>

    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="Default" />
    <Publisher class="WebApi" />
</Test>

</Peach>
```

Example 23. Path with DataModel

A simple web api request with a single *Path* element.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="PathValue">
    <String value="foo"/>
    <String value="-" />
    <String value="bar" />
</DataModel>

<StateModel name="Default" initialState="FirstState">
    <State name="FirstState">

        <Action type="web" method="GET" url="http://www.example.com/product/{id}">
            <Path key="id">
                <DataModel ref="PathValue" />
            </Path>
            <Response />
        </Action>

    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="Default" />
    <Publisher class="WebApi" />
</Test>

</Peach>
```

Example 24. Path with Data sets

A simple web api request with a single *Path* element.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="PathValue">
    <String name="Value"/>
</DataModel>

<StateModel name="Default" initialState="FirstState">
    <State name="FirstState">

        <Action type="web" method="GET" url="http://www.example.com/product/{id}">
            <Path key="id">
                <DataModel ref="PathValue" />
                <Data>
                    <Field name="Value" value="100" />
                </Data>
                <Data>
                    <Field name="Value" value="101" />
                </Data>
                <Data>
                    <Field name="Value" value="102" />
                </Data>
            </Path>
            <Response />
        </Action>

    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="Default" />
    <Publisher class="WebApi" />
</Test>

</Peach>
```

21.1.15. Web Query

The *Query* element models an HTTP querystring parameter. This element is only valid when used with

a parent element of [web](#).

Syntax

```
<Action type="web" method="GET" url="http://www.google.com/users">
  <Query key="first" value="John"/>
  <Query key="last" value="Smith"/>
</Action>
```

Attributes

Required:

key

Substitution identifier

Optional:

name

Name of parameter. When not specified, a sanitized version of *key* will be used to generate a name.
The *name* field is shown in the job metrics.

value

Default value. When used, a DataModel is automatically created and attached to this element.
Cannot be used in conjunction with a DataModel child element.

valueType

Format of value attribute.

mutable

When *value* attribute is used, this attribute will mark all elements in the automatically generated model with this mutable value.

Child Elements

DataModel

Reference to a DataModel that acts as a source for fuzzed data.

Data

Set of initial data to be cracked into the above DataModel before fuzzing.

Examples

Example 25. Simple Example

A simple web api request with a two *Query* elements.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="LastName">
    <String name="value" value="Smith">
        <Hint name="Peach.TypeTransform" value="false" />
    </String>
</DataModel>

<StateModel name="Default" initialState="FirstState">
    <State name="FirstState">

        <Action type="web" method="GET" url="http://www.example.com/users">

            <Query key="first" value="John"/>

            <Query key="last">
                <DataModel ref="LastName" />
            </Query>

            <Response />
        </Action>

    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="Default" />
    <Publisher class="WebApi" />
</Test>

</Peach>
```

21.1.16. Web Header

The *Header* element models an HTTP header key-value pair. This element is only valid when used with a parent element of *web*.

Syntax

```
<Action type="web" method="GET" url="http://www.google.com/users">
    <Header key="Content-Type" value="application/json"/>

    <Body name="json">
        <DataModel ref="JsonBody" />
    </Body>
</Action>
```

Attributes

Required:

key

Substitution identifier

Optional:

name

Name of parameter. When not specified, a sanitized version of *key* will be used to generate a name.
The *name* field is shown in the job metrics.

value

Default value. When used, a DataModel is automatically created and attached to this element.
Cannot be used in conjunction with a DataModel child element.

valueType

Format of value attribute.

mutable

When *value* attribute is used, this attribute will mark all elements in the automatically generated model with this mutable value.

Child Elements

DataModel

Reference to a DataModel that acts as a source for fuzzed data.

Data

Set of initial data to be cracked into the above DataModel before fuzzing.

Examples

Example 26. Simple Example

Example setting the Content-Type header field.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="NewUser">
    <JsonObject>
        <JsonString propertyName="user" value="jsmith"/>
        <JsonString propertyName="name" value="John Smith" />
    </JsonObject>
</DataModel>

<StateModel name="Default" initialState="FirstState">
    <State name="FirstState">

        <Action type="web" method="POST" url="http://www.example.com/users">

            <Header key="Content-Type" value="application/json"/>

            <Body name="json">
                <DataModel ref="NewUser" />
            </Body>

            <Response />
        </Action>

    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="Default" />
    <Publisher class="WebApi" />
</Test>

</Peach>
```

21.1.17. Web FormData

The *FormData* element models a key-value pair sent via the HTTP body. Normally this is paired with a content type of *application/x-www-form-urlencoded*. This element is only valid when used with a parent element of *web*. Multiple *FormData* elements can be used, but cannot be mixed with *Body* elements.

Syntax

```
<Action type="web" method="GET" url="http://www.google.com/users">
  <Header key="Content-Type" value="application/x-www-form-urlencoded"/>
  <FormData key="first" value="John"/>
  <FormData key="last" value="Smith"/>
</Action>
```

Attributes

Required:

key

Substitution identifier

Optional:

name

Name of parameter. When not specified, a sanitized version of *key* will be used to generate a name.
The *name* field is shown in the job metrics.

value

Default value. When used, a DataModel is automatically created and attached to this element.
Cannot be used in conjunction with a DataModel child element.

valueType

Format of value attribute.

mutable

When *value* attribute is used, this attribute will mark all elements in the automatically generated model with this mutable value.

Child Elements

DataModel

Reference to a DataModel that acts as a source for fuzzed data.

Data

Set of initial data to be cracked into the above DataModel before fuzzing.

Examples

Example 27. Simple Example

Example setting the Content-Type header field.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<StateModel name="Default" initialState="FirstState">
    <State name="FirstState">

        <Action type="web" method="POST" url="http://www.example.com/users">

            <Header key="Content-Type" value="application/x-www-form-urlencoded"/>

            <FormData key="user" value="jsmith"/>
            <FormData key="first" value="John"/>
            <FormData key="last" value="Smith"/>

            <Response />
        </Action>

    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="Default" />
    <Publisher class="WebApi" />
</Test>

</Peach>
```

21.1.18. Web Body

The *Body* element models the web api/HTTP request body. Typically it is paired with a *Header* element to define the *Content-Type* of the request. *Body* elements can only be used with method verbs that allow request bodies such as *POST* and *PUT*. This element is only valid when used with a parent element of *web*. *Body* and *FormData* elements cannot be used together.

```
<Action type="web" method="GET" url="http://www.google.com/users">
    <Header name="ct" key="Content-Type" value="application/x-www-form-urlencoded"/>
    <Body name="JsonBody">
        <DataModel ref="JsonBody" />
    </Body>
</Action>
```

Attributes

Required:

name

Name of parameter.

Optional:

None.

Child Elements

DataModel

Reference to a DataModel that acts as a source for fuzzed data.

Data

Set of initial data to be cracked into the above DataModel before fuzzing.

Examples

Example 28. Simple Example

Example setting the Content-Type header field.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="NewUser">
    <JsonObject>
        <JsonString propertyName="user" value="jsmith"/>
        <JsonString propertyName="name" value="John Smith" />
    </JsonObject>
</DataModel>

<StateModel name="Default" initialState="FirstState">
    <State name="FirstState">

        <Action type="web" method="POST" url="http://www.example.com/users">

            <Header name="ct" key="Content-Type" value="application/json"/>

            <Body name="json">
                <DataModel ref="NewUser" />
            </Body>

            <Response />
        </Action>

    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="Default" />
    <Publisher class="WebApi" />
</Test>

</Peach>
```

21.1.19. Web Part

The *Part* element allows modeling of HTTP multipart requests. Each section of a multipart request can include its own set of HTTP headers and a body composed of *FormData* elements or a single *Body* element. This element is only valid when used with a parent element of *web*.



A *Content-Type* header does not need to be provided when modeling a multipart HTTP request. Peach will automatically set the correct content-type header when a child of *Part* is used.

Syntax

```
<Action type="web" method="POST" url="http://api.company.com/api/v1/resource">

    <Part name="metadata">
        <Header key="Content-Disposition" value="form-data; name="metadata"" />
        <Header key="Content-Type" value="application/json"/>
        <Body name="json">
            <DataModel ref="JsonBody" />
        </Body>
    </Part>

    <Part name="file">
        <Header key="Content-Disposition" value="form-data; name="file"; filename="filename.zip"" />
        <Header key="Content-Type" value="binary/octet-stream" />
        <Body name="fileContents">
            <DefaModel ref="FileData" />
        </Body>
    </Part>

</Action>
```

Attributes

Required:

name

Name of parameter.

Optional:

value

Default value. When used, a DataModel is automatically created and attached to this element. Cannot be used in conjunction with a DataModel child element.

valueType

Format of value attribute.

mutable

When *value* attribute is used, this attribute will mark all elements in the automatically generated

model with this mutable value.

Child Elements

Header

Define an HTTP header.

FormData

Define a key/value pair of form data. These values are transmitted via the request body. It is not possible to combine FormData with a Body child.

Body

Define the request body. Only one Body child element is allowed. Body elements cannot be mixed with FormData elements. Only one type of body is allowed.

Examples

Example 29. Uploading a file with JSON metadata

The following example models a request that uploads a file along with some metadata in JSON format.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<StateModel name="Default" initialState="FirstState">
    <State name="FirstState">
        <Action type="web" method="POST" url="http://api.company.com/api/user/{id}">

            <Path key="id" value="1" />

            <Part name="UserPictureMeta">
                <Header key="Content-Disposition" value="form-data; name=
"metadata" />
                <Header key="Content-Type" value="application/json"/>

                <Body name="json">
                    <DataModel ref="JsonBody" />
                </Body>
            </Part>

            <Part name="File">
                <Header key="Content-Disposition" value="form-data; name=
"file" />
                <Header key="Content-Type" value="binary/octet-stream" />

                <Body name="fileContents">
                    <DefaModel ref="FileData" />
                </Body>
            </Part>

        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="Default" />
    <Publisher class="WebApi" />
</Test>

</Peach>
```

Example 30. Uploading file with form data

The following example models a request that uploads a file along with some metadata as form data.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<StateModel name="Default" initialState="FirstState">
    <State name="FirstState">
        <Action type="web" method="POST" url="http://api.company.com/api/user/{id}">

            <Path key="id" value="1" />

            <Part name="UserPictureMeta">
                <Header key="Content-Disposition" value="form-data; name=
"quot;metadataquot;" />
                <Header key="Content-Type" value="application/x-www-form-urlencoded
"/>

                <FormData key="createor" value="Josh Smith" />
                <FormData key="category" value="misc" />
                <FormData key="icon" value="zip" />
            </Part>

            <Part name="File">
                <Header key="Content-Disposition" value="form-data; name="quot;
filequot; filename="quot;filename.zipquot;" />
                <Header key="Content-Type" value="binary/octet-stream" />

                <Body name="fileContents">
                    <DefaModel ref="FileData" />
                </Body>
            </Part>

        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="Default" />
    <Publisher class="WebApi" />
</Test>

</Peach>

```

21.1.20. Web Response

The *Response* element is used to model HTTP response bodies. It's an optional child element of the [web](#) action. If a *Response* element is not specified, the *web* action will automatically create a default one. When manually specifying a *Response* element a child DataModel is typically specified to capture the body.

A *Response* element will modify the resulting DataModel to also contain the status code, status message and headers. The resulting data model will look as follows:

```
<Response>
  <DataModel name="Response">
    <String name="StatusCode" />
    <String name="StatusDescription"/>
    <Block name="Headers">
      <String name="Content-Type" value="application/json"/>
    </Block>
    <Block name="Body">
      <!-- Custom DataModel contents here -->
      <!-- ... OR ... -->
      <!-- Auto generated Data Model -->
    </Block>
  </DataModel>
</Response>
```

The automatically generated will be one of the following:

Model generated for text based content types:

```
<DataModel name="WebApiResponse">
  <Choice name="ResultOfEmpty">
    <String name="Result">
      <!-- Analyzer is only added for known content-types -->
      <Analyzer class="Json|Xml"/>
    </String>
    <Block name="Empty" />
  </Choice>
</DataModel>
```

Model generated for unknown content types:

```
<DataModel name="WebApiResponse">
  <Choice name="ResultOfEmpty">
    <Blob name="Result" />
    <Block name="Empty" />
  </Choice>
</DataModel>
```

Syntax:

```
<Action type="web" method="GET" url="http://www.google.com/users">
  <Response>
    <DataModel ref="CustomBody" />
  </Response>
</Action>
```

Attributes

Required:

name

Name of parameter.

Optional:

None.

Child Elements

DataModel

Reference to a DataModel that acts as a source for fuzzed data.

Data

Set of initial data to be cracked into the above DataModel before fuzzing.

Examples

Example 31. Simple Example

Example setting the Content-Type header field.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="User">
    <JsonObject>
        <JsonString propertyName="user" value="jsmith"/>
        <JsonString propertyName="name" value="John Smith" />
    </JsonObject>
</DataModel>

<StateModel name="Default" initialState="FirstState">
    <State name="FirstState">

        <Action type="web" method="GET" url="http://www.example.com/user/1">

            <Response>
                <DataModel ref="User" />
            </Response>

        </Action>

    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="Default" />
    <Publisher class="WebApi" />
</Test>

</Peach>
```

21.2. Analyzers

Analyzers are classes that parse some form of data and build a Peach Document Object Model (DOM) or a partial *DataModel*. Examples of data that can be easily parsed are XML and ASN.1.

Using an *Analyzer* can incredibly reduce the time needed to begin smart fuzzing since the time spent creating a precise *DataModel* can be avoided.

Asn1

Converts ASN.1 data in *Blobs* into a full data model.

Binary

Breaks up unknown data when strings are found.

JSON

Converts JSON data in a *String* into a full data model.

Postman

Converts Postman API Catalog into Pit.

Regex

Use regular expressions to parse strings. Supports regular expression named groups.

StringToken

Separates strings using a precedence based list of punctuation. The strings are used to form the resulting *DataModel*.

Swagger

Converts Swagger API JSON into Pit.

Vcr

Converts Vcr JSON cassette into Pit.

Xml

Converts XML contained in a *string* element into a data model of *XmlElement* and *XmlAttribute* types.

WebRecordProxy

Record HTTP requests and generate Peach Pit.

Zip

Converts zipped data in a *blob* into a data model of *stream* elements.

21.2.1. ASN.1 Analyzer

This analyzer converts the Abstract Syntax Notation One (ASN.1) specification into a full data model. This analyzer includes the following parts of the ASN.1 specification in the data model:

- Basic Encoding Rules (BER)
- Canonical Encoding Rules (CER)
- Distinguished Encoding Rules (DER)

When used in the DataModel section of a Peach Pit, the Fuzzer walks the ASN.1 data and creates the appropriate elements.

When used from the command line, the Fuzzer walks the ASN.1 data, creates the appropriate elements, and saves the results of the generated model to disk. Once saved, you can use and modify the results as needed.



The ASN.1 analyzer requires data from a Blob data type.

Syntax

```
<Blob name="BinaryData">
    <Analyzer class="Asn1" />
</Blob>
```

```
pittool analyzer Asn1 input.bin output.xml
```

Command Line Syntax

```
pittool analyzer Asn1 <input file> <output file>
```

input file

File containing ASN.1 structured data.

output file

File creating containing generated data model

Attributes

Required:

None

Optional:

None

Examples

Example 32. Command line data model generation

This example uses the Asn.1 analyzer on the command line to automatically generate a data model. The sample file contains an ASN.1 bit string that is comprised of two concatenated ASN.1 bit strings.

Hex dump of sample file `asn1.bin`

```
00000000: 2380 0303 000a 3b03 0504 5f29 1cd0 0000 #.....;....)....
```

Output from running peach on the command line

```
> pittool analyzer Asn1 asn1.bin asn1.xml  
[*] Starting Analyzer
```

Contents of generated pit `asn1.xml`

```

<Peach>
  <DataModel name="example_asn1">
    <Asn1Type class="0" pc="1" tag="3" name="BIT_STRING">
      <Block name="Value">
        <Asn1Type class="0" pc="0" tag="3" name="BIT_STRING">
          <Block name="Value">
            <Number size="8" signed="false" name="UnusedLen" value="0">
              <Relation type="size" of="UnusedBits" />
            </Number>
            <Blob name="Value" valueType="hex" value="0a3b" />
            <Blob name="UnusedBits" valueType="hex" value="" />
            <Padding name="Padding" />
          </Block>
        </Asn1Type>
        <Asn1Type class="0" pc="0" tag="3" name="BIT_STRING_1">
          <Block name="Value">
            <Number size="8" signed="false" name="UnusedLen" value="4">
              <Relation type="size" of="UnusedBits" />
            </Number>
            <Blob name="Value" valueType="hex" value="5f291c" />
            <Blob name="UnusedBits" valueType="hex" value="" />
            <Padding name="Padding" />
          </Block>
        </Asn1Type>
        <Asn1Type class="0" pc="0" tag="0" name="EOC">
          <Blob name="Value" valueType="hex" value="" />
        </Asn1Type>
      </Block>
    </Asn1Type>
  </DataModel>
</Peach>

```

Example 33. Certificate Inline Example

This example uses the Asn.1 analyzer on inline Blob data.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

  <DataModel name="TheModel">
    <Blob valueType="hex" value="30 82 03 85 30 82 02 6D A0 03 02 01 02 02 09 00
E2 5B 91 05 F2 8F AB AA 30 0D 06 09 2A 86 48 86 F7 0D 01 01 05 05 00 30 59 31 0B 30
09 06 03 55 04 06 13 02 55 53 31 13 30 11 06 03 55 04 08 0C 0A 57 61 73 68 69 6E 67

```

```

74 6F 6E 31 10 30 0E 06 03 55 04 07 0C 07 53 65 61 74 74 6C 65 31 0D 30 0B 06 03 55
04 0A 0C 04 44 65 6A 61 31 14 30 12 06 03 55 04 03 0C 0B 74 65 73 74 69 6E 67 2E 63
6F 6D 30 1E 17 0D 31 34 30 33 31 37 30 30 32 32 32 30 5A 17 0D 31 35 30 33 31 37 30
30 32 32 32 30 5A 30 59 31 0B 30 09 06 03 55 04 06 13 02 55 53 31 13 30 11 06 03 55
04 08 0C 0A 57 61 73 68 69 6E 67 74 6F 6E 31 10 30 0E 06 03 55 04 07 0C 07 53 65 61
74 74 6C 65 31 0D 30 0B 06 03 55 04 0A 0C 04 44 65 6A 61 31 14 30 12 06 03 55 04 03
0C 0B 74 65 73 74 69 6E 67 2E 63 6F 6D 30 82 01 22 30 0D 06 09 2A 86 48 86 F7 0D 01
01 01 05 00 03 82 01 0F 00 30 82 01 0A 02 82 01 01 00 A2 9F 5E 21 EE 45 4A 0A AB CB
D9 35 42 7C A9 5C 9C 59 8D 72 78 0A A0 49 63 C2 FE 36 42 9B 43 CC 05 41 49 26 3B 37
2D BC 10 10 B8 57 43 AF 6B 2B 7E 97 87 FC CB 00 EC 03 0B D6 58 55 71 C1 B0 6A 1D 38
9E EB 4C 5F D0 25 2E C6 20 AF 68 92 0E DB 8B 3D 97 61 89 3B 6A 0D 50 77 26 0A 60 0D
11 B3 82 F7 DF 30 8D F9 45 7F CD C0 88 B8 82 3F 24 A3 86 17 0E 19 60 E7 98 71 27 CE
63 49 F9 E0 95 47 E3 A6 A6 CC 9B DB 19 92 C0 58 23 90 11 C1 A6 F5 34 02 9A DD 09 FF
D7 59 E7 E4 48 91 92 5C 17 EA 86 84 1D A9 57 26 13 76 F4 F7 8F 29 5A 10 FD E4 BD AE
E3 CC AD 5E 64 03 E7 B6 A1 48 0E 2A D2 6B 24 95 EC 42 AE FB 79 B9 C0 9F 49 5C 2B 10
D8 A1 CE 44 8C 89 97 9B 97 45 96 5D 24 C6 3E E6 79 9F 2B 25 4A C5 21 41 0B 55 18 90
15 A7 56 C1 69 A9 90 B2 73 C6 35 47 53 4D F4 88 6F D7 E2 59 90 DB 02 03 01 00 01 A3
50 30 4E 30 1D 06 03 55 1D 0E 04 16 04 14 36 F2 B5 D1 62 F1 F8 BF B7 1C F7 70 DD B6
D9 32 2E B6 99 5E 30 1F 06 03 55 1D 23 04 18 30 16 80 14 36 F2 B5 D1 62 F1 F8 BF B7
1C F7 70 DD B6 D9 32 2E B6 99 5E 30 0C 06 03 55 1D 13 04 05 30 03 01 01 FF 30 0D 06
09 2A 86 48 86 F7 0D 01 01 05 05 00 03 82 01 01 00 4F C7 70 55 D7 74 7F 12 50 78 D1
14 77 4D 05 6C D3 5E 56 F2 84 1A D8 BC 59 BC D3 B7 63 4D F3 5F 44 1C 2C 8C A9 66 89
07 23 4D 5A 1D F8 C0 DD E7 D2 38 9A 0F 1C 56 B6 F9 FF 50 85 BA C6 09 2C 80 A6 A9 B0
47 ED 9B DF 8E 53 B6 DB 4A 4A 05 58 DC 7E 98 E5 DF B0 C7 6B A2 01 67 DA AE 6A 1E 26
8D 33 B0 17 BD 5D C3 B6 12 D5 80 A8 16 CA B6 A2 AF DD D1 80 32 89 6E 1A 7A C3 9F 7A
15 1F 35 36 EC 85 D6 B2 84 91 AD 8D 7D 40 51 8B 5A 3B 5D C9 89 9D 74 13 77 86 7A ED
59 60 89 D0 35 71 07 3E 84 2B 44 5D 26 D3 19 EE 92 F9 49 FF C9 76 BA 43 6B A7 A9 0C
2C A1 6D C3 0B 98 AB 92 99 3C C8 76 DE 7D 14 50 45 68 84 7F E9 B0 FE 90 7B 10 A7 9C
9A 40 9F 0A 49 B5 0D 0C 86 21 9B F3 49 B1 9E 55 88 9B 76 6F DC 00 F5 35 11 A0 F2 EB
49 9D 8C 5A 78 2F 98 CB FE 77 E8 C2 91 95 FA C4 87 88 E3 F5 D7 ">
    <Analyzer class="Asn1" />
</Blob>
</DataModel>

<StateModel name="TheState" initialState="initial">
    <State name="initial">
        <Action type="output">
            <DataModel ref="TheModel" />
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>
    <Publisher class="ConsoleHex" />
</Test>
</Peach>

```

Output from inline Blob data example.

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 62676.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(905 bytes)
00000000 30 82 03 85 30 82 02 6D A0 03 02 01 02 02 09 00
00000010 E2 5B 91 05 F2 8F AB AA 30 0D 06 09 2A 86 48 86
00000020 F7 0D 01 01 05 05 00 30 59 31 0B 30 09 06 03 55
00000030 04 06 13 02 55 53 31 13 30 11 06 03 55 04 08 0C
00000040 0A 57 61 73 68 69 6E 67 74 6F 6E 31 10 30 0E 06
00000050 03 55 04 07 0C 07 53 65 61 74 74 6C 65 31 0D 30
00000060 0B 06 03 55 04 0A 0C 04 44 65 6A 61 31 14 30 12
00000070 06 03 55 04 03 0C 0B 74 65 73 74 69 6E 67 2E 63
00000080 6F 6D 30 1E 17 0D 31 34 30 33 31 37 30 30 32 32
00000090 32 30 5A 17 0D 31 35 30 33 31 37 30 30 32 32 32
000000A0 30 5A 30 59 31 0B 30 09 06 03 55 04 06 13 02 55
000000B0 53 31 13 30 11 06 03 55 04 08 0C 0A 57 61 73 68
000000C0 69 6E 67 74 6F 6E 31 10 30 0E 06 03 55 04 07 0C
000000D0 07 53 65 61 74 74 6C 65 31 0D 30 0B 06 03 55 04
000000E0 0A 0C 04 44 65 6A 61 31 14 30 12 06 03 55 04 03
000000F0 0C 0B 74 65 73 74 69 6E 67 2E 63 6F 6D 30 82 01
00000100 22 30 0D 06 09 2A 86 48 86 F7 0D 01 01 01 05 00
00000110 03 82 01 0F 00 30 82 01 0A 02 82 01 01 00 A2 9F
00000120 5E 21 EE 45 4A 0A AB CB D9 35 42 7C A9 5C 9C 59
00000130 8D 72 78 0A A0 49 63 C2 FE 36 42 9B 43 CC 05 41
00000140 49 26 3B 37 2D BC 10 10 B8 57 43 AF 6B 2B 7E 97
00000150 87 FC CB 00 EC 03 0B D6 58 55 71 C1 B0 6A 1D 38
00000160 9E EB 4C 5F D0 25 2E C6 20 AF 68 92 0E DB 8B 3D
00000170 97 61 89 3B 6A 0D 50 77 26 0A 60 0D 11 B3 82 F7
00000180 DF 30 8D F9 45 7F CD C0 88 B8 82 3F 24 A3 86 17
00000190 0E 19 60 E7 98 71 27 CE 63 49 F9 E0 95 47 E3 A6
000001A0 A6 CC 9B DB 19 92 C0 58 23 90 11 C1 A6 F5 34 02
000001B0 9A DD 09 FF D7 59 E7 E4 48 91 92 5C 17 EA 86 84
000001C0 1D A9 57 26 13 76 F4 F7 8F 29 5A 10 FD E4 BD AE
000001D0 E3 CC AD 5E 64 03 E7 B6 A1 48 0E 2A D2 6B 24 95
000001E0 EC 42 AE FB 79 B9 C0 9F 49 5C 2B 10 D8 A1 CE 44
000001F0 8C 89 97 9B 97 45 96 5D 24 C6 3E E6 79 9F 2B 25
00000200 4A C5 21 41 0B 55 18 90 15 A7 56 C1 69 A9 90 B2
00000210 73 C6 35 47 53 4D F4 88 6F D7 E2 59 90 DB 02 03
```

```

00000220 01 00 01 A3 50 30 4E 30 1D 06 03 55 1D 0E 04 16
00000230 04 14 36 F2 B5 D1 62 F1 F8 BF B7 1C F7 70 DD B6
00000240 D9 32 2E B6 99 5E 30 1F 06 03 55 1D 23 04 18 30
00000250 16 80 14 36 F2 B5 D1 62 F1 F8 BF B7 1C F7 70 DD
00000260 B6 D9 32 2E B6 99 5E 30 0C 06 03 55 1D 13 04 05
00000270 30 03 01 01 FF 30 0D 06 09 2A 86 48 86 F7 0D 01
00000280 01 05 05 00 03 82 01 01 00 4F C7 70 55 D7 74 7F
00000290 12 50 78 D1 14 77 4D 05 6C D3 5E 56 F2 84 1A D8
000002A0 BC 59 BC D3 B7 63 4D F3 5F 44 1C 2C 8C A9 66 89
000002B0 07 23 4D 5A 1D F8 C0 DD E7 D2 38 9A 0F 1C 56 B6
000002C0 F9 FF 50 85 BA C6 09 2C 80 A6 A9 B0 47 ED 9B DF
000002D0 8E 53 B6 DB 4A 4A 05 58 DC 7E 98 E5 DF B0 C7 6B
000002E0 A2 01 67 DA AE 6A 1E 26 8D 33 B0 17 BD 5D C3 B6
000002F0 12 D5 80 A8 16 CA B6 A2 AF DD D1 80 32 89 6E 1A
00000300 7A C3 9F 7A 15 1F 35 36 EC 85 D6 B2 84 91 AD 8D
00000310 7D 40 51 8B 5A 3B 5D C9 89 9D 74 13 77 86 7A ED
00000320 59 60 89 D0 35 71 07 3E 84 2B 44 5D 26 D3 19 EE
00000330 92 F9 49 FF C9 76 BA 43 6B A7 A9 0C 2C A1 6D C3
00000340 0B 98 AB 92 99 3C C8 76 DE 7D 14 50 45 68 84 7F
00000350 E9 B0 FE 90 7B 10 A7 9C 9A 40 9F 0A 49 B5 0D 0C
00000360 86 21 9B F3 49 B1 9E 55 88 9B 76 6F DC 00 F5 35
00000370 11 A0 F2 EB 49 9D 8C 5A 78 2F 98 CB FE 77 E8 C2
00000380 91 95 FA C4 87 88 E3 F5 D7

Peach.Core.Publisher.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publisher stop()

[*] Test 'Default' finished.

```

Output in Peach Validator

Name	Position	Length	Value
TheModel	0	0	
DataElement_0	0	0	
SEQUENCE	0	0	
Identifier	0	0	
Class	0	0	0
PC	0	0	1
Tag	0	0	16
Length	0	0	901
Value	0	0	
SEQUENCE	0	0	
Identifier	0	0	
Class	0	0	0
PC	0	0	1
Tag	0	0	16
Length	0	0	621
Value	0	0	
CONTEXT_SPECIFIC	0	0	
Identifier	0	0	
Class	0	0	2
PC	0	0	1
Tag	0	0	0
Length	0	0	3
Value	0	0	
INTEGER	0	0	
Identifier	0	0	
Class	0	0	0
PC	0	0	0
Tag	0	0	2
Length	0	0	1
Value	0	0	02
INTEGER	0	0	
Identifier	0	0	
Class	0	0	0
PC	0	0	0
Tag	0	0	2
Length	0	0	9
Value	0	0	00 e2 5b 91 05 f2 8f ab aa
SEQUENCE	0	0	
SEQUENCE_1	0	0	
SEQUENCE_2	0	0	
SEQUENCE_3	0	0	
SEQUENCE_4	0	0	
CONTEXT_SPECIFIC_	0	0	
Identifier	0	0	
Length	0	0	80
Value	0	0	
SEQUENCE	0	0	
Identifier	0	0	
Length	0	0	78
Value	0	0	
SEQUENCE_1	0	0	
Identifier	0	0	
Length	0	0	13
Value	0	0	
BIT_STRING	0	0	
Identifier	0	0	
Length	0	0	257
Value	0	0	
UnusedLen	0	0	0
Value	0	0	4f c7 70 55 d7 74 7f 12 50 78 d1 14 77 4d 05 6c d3 5e 56
UnusedBits	0	0	
Padding	0	0	

Example 34. Certificate From File Example

This example uses the Asn.1 analyzer on an external file of Blob data (Cert.der).

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="TheModel">
        <Blob>
            <Analyzer class="Asn1" />
        </Blob>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="TheModel" />
                <Data name="Cert" fileName="Cert.der"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>
        <Publisher class="ConsoleHex" />
    </Test>
</Peach>
```

Output from the Blob data file example.

```
> peach -l --debug example.xml

[*] Test 'Default' starting with random seed 18200.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'TheModel' Bytes: 0/905, Bits: 0/7240
Peach.Core.Cracker.DataCracker getSize: -----> DataModel 'TheModel'
Peach.Core.Cracker.DataCracker scan: DataModel 'TheModel'
Peach.Core.Cracker.DataCracker scan: Blob 'TheModel.DataElement_0' -> Offset: 0
    Unsized element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: DataModel 'TheModel' Size: <null>, Bytes:
```

```

0/905, Bits: 0/7240
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Blob 'TheModel.DataElement_0' Bytes: 0/905, Bits
0/7240
Peach.Core.Cracker.DataCracker getSize: -----> Blob 'TheModel.DataElement_0'
Peach.Core.Cracker.DataCracker scan: Blob 'TheModel.DataElement_0' -> Offset: 0
    Unsized element
Peach.Core.Cracker.DataCracker lookahead: Blob 'TheModel.DataElement_0'
Peach.Core.Cracker.DataCracker getSize: <----- Last Unsized: 7240
Peach.Core.Cracker.DataCracker Crack: Blob 'TheModel.DataElement_0' Size: 7240,
Bytes: 0/905, Bits: 0/7240
Peach.Core.Dom.DataElement Blob 'TheModel.DataElement_0' value is: 30 82 03 85
0 82 02 6d a0 03 02 01 02 02 09 00 e2 5b 91 05 f2 8f ab aa 30 0d 06 09 2a 86 48
86.. (Len: 905 bytes)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(905 bytes)
00000000 30 82 03 85 30 82 02 6D A0 03 02 01 02 02 09 00
00000010 E2 5B 91 05 F2 8F AB AA 30 0D 06 09 2A 86 48 86
00000020 F7 0D 01 01 05 05 00 30 59 31 0B 30 09 06 03 55
00000030 04 06 13 02 55 53 31 13 30 11 06 03 55 04 08 0C
00000040 0A 57 61 73 68 69 6E 67 74 6F 6E 31 10 30 0E 06
00000050 03 55 04 07 0C 07 53 65 61 74 74 6C 65 31 0D 30
00000060 0B 06 03 55 04 0A 0C 04 44 65 6A 61 31 14 30 12
00000070 06 03 55 04 03 0C 0B 74 65 73 74 69 6E 67 2E 63
00000080 6F 6D 30 1E 17 0D 31 34 30 33 31 37 30 30 32 32
00000090 32 30 5A 17 0D 31 35 30 33 31 37 30 30 32 32 32
000000A0 30 5A 30 59 31 0B 30 09 06 03 55 04 06 13 02 55
000000B0 53 31 13 30 11 06 03 55 04 08 0C 0A 57 61 73 68
000000C0 69 6E 67 74 6F 6E 31 10 30 0E 06 03 55 04 07 0C
000000D0 07 53 65 61 74 74 6C 65 31 0D 30 0B 06 03 55 04
000000E0 0A 0C 04 44 65 6A 61 31 14 30 12 06 03 55 04 03
000000F0 0C 0B 74 65 73 74 69 6E 67 2E 63 6F 6D 30 82 01
00000100 22 30 0D 06 09 2A 86 48 86 F7 0D 01 01 01 05 00
00000110 03 82 01 0F 00 30 82 01 0A 02 82 01 01 00 A2 9F
00000120 5E 21 EE 45 4A 0A AB CB D9 35 42 7C A9 5C 9C 59
00000130 8D 72 78 0A A0 49 63 C2 FE 36 42 9B 43 CC 05 41
00000140 49 26 3B 37 2D BC 10 10 B8 57 43 AF 6B 2B 7E 97
00000150 87 FC CB 00 EC 03 0B D6 58 55 71 C1 B0 6A 1D 38
00000160 9E EB 4C 5F D0 25 2E C6 20 AF 68 92 0E DB 8B 3D
00000170 97 61 89 3B 6A 0D 50 77 26 0A 60 0D 11 B3 82 F7
00000180 DF 30 8D F9 45 7F CD C0 88 B8 82 3F 24 A3 86 17
00000190 0E 19 60 E7 98 71 27 CE 63 49 F9 E0 95 47 E3 A6
000001A0 A6 CC 9B DB 19 92 C0 58 23 90 11 C1 A6 F5 34 02
000001B0 9A DD 09 FF D7 59 E7 E4 48 91 92 5C 17 EA 86 84
000001C0 1D A9 57 26 13 76 F4 F7 8F 29 5A 10 FD E4 BD AE

```

000001D0	E3 CC AD 5E 64 03 E7 B6 A1 48 0E 2A D2 6B 24 95
000001E0	EC 42 AE FB 79 B9 C0 9F 49 5C 2B 10 D8 A1 CE 44
000001F0	8C 89 97 9B 97 45 96 5D 24 C6 3E E6 79 9F 2B 25
00000200	4A C5 21 41 0B 55 18 90 15 A7 56 C1 69 A9 90 B2
00000210	73 C6 35 47 53 4D F4 88 6F D7 E2 59 90 DB 02 03
00000220	01 00 01 A3 50 30 4E 30 1D 06 03 55 1D 0E 04 16
00000230	04 14 36 F2 B5 D1 62 F1 F8 BF B7 1C F7 70 DD B6
00000240	D9 32 2E B6 99 5E 30 1F 06 03 55 1D 23 04 18 30
00000250	16 80 14 36 F2 B5 D1 62 F1 F8 BF B7 1C F7 70 DD
00000260	B6 D9 32 2E B6 99 5E 30 0C 06 03 55 1D 13 04 05
00000270	30 03 01 01 FF 30 0D 06 09 2A 86 48 86 F7 0D 01
00000280	01 05 05 00 03 82 01 01 00 4F C7 70 55 D7 74 7F
00000290	12 50 78 D1 14 77 4D 05 6C D3 5E 56 F2 84 1A D8
000002A0	BC 59 BC D3 B7 63 4D F3 5F 44 1C 2C 8C A9 66 89
000002B0	07 23 4D 5A 1D F8 C0 DD E7 D2 38 9A 0F 1C 56 B6
000002C0	F9 FF 50 85 BA C6 09 2C 80 A6 A9 B0 47 ED 9B DF
000002D0	8E 53 B6 DB 4A 4A 05 58 DC 7E 98 E5 DF B0 C7 6B
000002E0	A2 01 67 DA AE 6A 1E 26 8D 33 B0 17 BD 5D C3 B6
000002F0	12 D5 80 A8 16 CA B6 A2 AF DD D1 80 32 89 6E 1A
00000300	7A C3 9F 7A 15 1F 35 36 EC 85 D6 B2 84 91 AD 8D
00000310	7D 40 51 8B 5A 3B 5D C9 89 9D 74 13 77 86 7A ED
00000320	59 60 89 D0 35 71 07 3E 84 2B 44 5D 26 D3 19 EE
00000330	92 F9 49 FF C9 76 BA 43 6B A7 A9 0C 2C A1 6D C3
00000340	0B 98 AB 92 99 3C C8 76 DE 7D 14 50 45 68 84 7F
00000350	E9 B0 FE 90 7B 10 A7 9C 9A 40 9F 0A 49 B5 0D 0C
00000360	86 21 9B F3 49 B1 9E 55 88 9B 76 6F DC 00 F5 35
00000370	11 A0 F2 EB 49 9D 8C 5A 78 2F 98 CB FE 77 E8 C2
00000380	91 95 FA C4 87 88 E3 F5 D7

```
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()
```

[*] Test 'Default' finished.

Output in Peach Validator

Name	Position	Length	Value
TheModel	0	0	
DataElement_0	0	0	
SEQUENCE	0	0	
Identifier	0	0	
Class	0	0	0
PC	0	0	1
Tag	0	0	16
Length	0	0	901
Value	0	0	
SEQUENCE	0	0	
Identifier	0	0	
Class	0	0	0
PC	0	0	1
Tag	0	0	16
Length	0	0	621
Value	0	0	
CONTEXT_SPECIFIC	0	0	
Identifier	0	0	
Class	0	0	2
PC	0	0	1
Tag	0	0	0
Length	0	0	3
Value	0	0	
INTEGER	0	0	
Identifier	0	0	
Class	0	0	0
PC	0	0	0
Tag	0	0	2
Length	0	0	1
Value	0	0	02
INTEGER	0	0	
Identifier	0	0	
Class	0	0	0
PC	0	0	0
Tag	0	0	2
Length	0	0	9
Value	0	0	00 e2 5b 91 05 f2 8f ab aa
SEQUENCE	0	0	
SEQUENCE_1	0	0	
SEQUENCE_2	0	0	
SEQUENCE_3	0	0	
SEQUENCE_4	0	0	
CONTEXT_SPECIFIC_	0	0	
Identifier	0	0	
Length	0	0	80
Value	0	0	
SEQUENCE	0	0	
Identifier	0	0	
Length	0	0	78
Value	0	0	
SEQUENCE_1	0	0	
Identifier	0	0	
Length	0	0	13
Value	0	0	
BIT_STRING	0	0	
Identifier	0	0	
Length	0	0	257
Value	0	0	
UnusedLen	0	0	0
Value	0	0	4f c7 70 55 d7 74 7f 12 50 78 d1 14 77 4d 05 6c d3 5e 56
UnusedBits	0	0	
Padding	0	0	

21.2.2. Binary Analyzer

The Binary analyzer applies to Blob data elements and provides an easy way to improve fuzzing of unknown binary data.

The analyzer searches for known types (such as strings) in the Blob data and constructs a DataModel of the Blob. If a string type occurs in the Blob data, Peach runs the StringTokenizer analyzer on each string found.



The Binary analyzer requires data from a Blob data type.

Syntax

```
<Blob name="BinaryData">
    <Analyzer class="Binary" />
</Blob>
```

Attributes

Required:

None.

Optional:

None.

Parameters

Tokens

List of characters to pass to the StringTokenizer analyzer.

AnalyzeStrings

Calls the StringTokenizer analyzer on string elements. The default value is true.

Examples

Example 35. Simple Hello World Example

This example uses the binary analyzer on inline Blob data.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="TheModel">
        <Blob valueType="hex" value="54 65 73 74 69 6E 67 20 48 65 6C 6C 6F 57 6F 72
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 50 65 61 63 68 46 75 7A 7A 65 72 21
FF AA BB CC">
            <Analyzer class="Binary" />
        </Blob>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="TheModel" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>
        <Publisher class="ConsoleHex" />
    </Test>
</Peach>
```

Output from the inline Blob data example.

```

> peach -l --debug example.xml

Peach.Core.Analyzers.Binary Created 12 data elements from binary data.
Peach.Core.Analyzers.Binary Created 12 data elements from binary data.

[*] Test 'Default' starting with random seed 61927.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(48 bytes)
00000000  54 65 73 74 69 6E 67 20  48 65 6C 6C 6F 57 6F 72  Testing HelloWor
00000010  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  ????????????????
00000020  50 65 61 63 68 46 75 7A  7A 65 72 21 FF AA BB CC  PeachFuzzer!???
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

Output in Peach Validator

Name	Position	Length	Value
TheModel	0	0	
DataElement_0	0	0	
DataElement_1	0	0	
DataElement_1	0	0	
DataElement_2	0	0	Testing
DataElement_3	0	0	
DataElement_4	0	0	HelloWor
DataElement_6	0	0	00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
DataElement_5	0	0	
DataElement_5	0	0	
DataElement_7	0	0	PeachFuzzer
DataElement_8	0	0	!
DataElement_9	0	0	
DataElement_10	0	0	ff aa bb cc

Example 36. Load From File Example

This example uses the binary analyzer on an external file of Blob data (example.bin).

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="TheModel">
        <Blob name="BinaryData">
            <Analyzer class="Binary" />
        </Blob>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="TheModel" />
                <Data name="SampleData1" fileName="example.bin"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>
        <Publisher class="ConsoleHex" />
    </Test>
</Peach>
```

Output from the Blob data file example.

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 48471.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'TheModel' Bytes: 0/48, Bits: 0/384
Peach.Core.Cracker.DataCracker getSize: -----> DataModel 'TheModel'
Peach.Core.Cracker.DataCracker scan: DataModel 'TheModel'
Peach.Core.Cracker.DataCracker scan: Blob 'TheModel.BinaryData' -> Offset: 0, Un
sized element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: DataModel 'TheModel' Size: <null>, Bytes:
0/48, Bits: 0/384
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Blob 'TheModel.BinaryData' Bytes: 0/48, Bits: 0/3
84
Peach.Core.Cracker.DataCracker getSize: -----> Blob 'TheModel.BinaryData'
Peach.Core.Cracker.DataCracker scan: Blob 'TheModel.BinaryData' -> Offset: 0, Un
sized element
Peach.Core.Cracker.DataCracker lookahead: Blob 'TheModel.BinaryData'
Peach.Core.Cracker.DataCracker getSize: <----- Last Unsized: 384
Peach.Core.Cracker.DataCracker Crack: Blob 'TheModel.BinaryData' Size: 384, Byte
s: 0/48, Bits: 0/384
Peach.Core.Dom.DataElement Blob 'TheModel.BinaryData' value is: 54 65 73 74 69 6
e 67 20 48 65 6c 6c 6f 57 6f 72 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f.
. (Len: 48 bytes)
Peach.Core.Analyzers.Binary Created 12 data elements from binary data.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(48 bytes)
00000000  54 65 73 74 69 6E 67 20  48 65 6C 6C 6F 57 6F 72  Testing HelloWor
00000010  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  ???????????????
00000020  50 65 61 63 68 46 75 7A  7A 65 72 21 FF AA BB CC  PeachFuzzer!???
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

Output in Peach Validator

Name	Position	Length	Value
TheModel	0	0	
DataElement_0	0	0	
DataElement_1	0	0	
DataElement_1	0	0	
DataElement_2	0	0	Testing
DataElement_3	0	0	
DataElement_4	0	0	HelloWar
DataElement_6	0	0	00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
DataElement_5	0	0	
DataElement_5	0	0	
DataElement_7	0	0	PeachFuzzer
DataElement_8	0	0	!
DataElement_9	0	0	
DataElement_10	0	0	ff aa bb cc

Example 37. Custom String Tokens Example

This example uses the binary analyzer on a value with a custom set of tokens defined.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="TheModel">
        <Blob name="BinaryData" valueType="hex" value="28 54 65 73 74 69 6E 67 20 48 65
6C 6C 6F 57 29 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 7B 50 65 61 63 68 46
75 7A 7A 65 72 7D BB CC ">
        <Analyzer class="Binary">
            <Param name="Tokens" value="{}"/>
        </Analyzer>
    </Blob>
</DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="TheModel" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>
        <Publisher class="ConsoleHex" />
    </Test>
</Peach>

```

Output from the example that uses custom tokens.

```
> peach -1 --debug example.xml
```

```
Peach.Core.Analyzers.Binary Created 18 data elements from binary data.  
Peach.Core.Analyzers.Binary Created 18 data elements from binary data.
```

```
[*] Test 'Default' starting with random seed 9875.
```

```
[R1,-,-] Performing iteration
```

```
Peach.Core.Engine runTest: Performing recording iteration.
```

```
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
```

```
Peach.Core.Dom.Action ActionType.Output
```

```
Peach.Core.Publishers.ConsolePublisher start()
```

```
Peach.Core.Publishers.ConsolePublisher open()
```

```
Peach.Core.Publishers.ConsolePublisher output(47 bytes)
```

```
00000000 28 54 65 73 74 69 6E 67 20 48 65 6C 6C 6F 57 29 (Testing HelloW)
```

```
00000010 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ??????????????????
```

```
00000020 7B 50 65 61 63 68 46 75 7A 7A 65 72 7D BB CC {PeachFuzzer}??
```

```
Peach.Core.Publishers.ConsolePublisher close()
```

```
Peach.Core.Engine runTest: context.config.singleIteration == true
```

```
Peach.Core.Publishers.ConsolePublisher stop()
```

```
[*] Test 'Default' finished.
```

Output in Peach Validator

Name	Position	Length	Value
TheModel	0	0	
BinaryData	0	0	
DataElement_0	0	0	
DataElement_0	0	0	
DataElement_1	0	0	
DataElement_2	0	0	(
DataElement_3	0	0	
DataElement_4	0	0	Testing HelloW
DataElement_5	0	0)
DataElement_6	0	0	
DataElement_8	0	0	00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
DataElement_7	0	0	
DataElement_7	0	0	
DataElement_9	0	0	
DataElement_10	0	0	{
DataElement_11	0	0	
DataElement_12	0	0	PeachFuzzer
DataElement_13	0	0	}
DataElement_14	0	0	
DataElement_15	0	0	bb cc

21.2.3. BSON Analyzer

This analyzer converts BSON documented into data models.

When used in the DataModel section of a Peach Pit, the Fuzzer walks the BSON data and creates the appropriate elements.

When used from the command line, the Fuzzer walks the BSON data, creates the appropriate elements, and saves the results of the generated model to disk. Once saved, the results can be added to a PIT or serve as the basis for a new PIT.



The BSON analyzer requires data from a Blob data type.

Syntax

```
<Blob name="BsonData">
  <Analyzer class="Bson" />
</Blob>
```

```
pittool analyzer Bson input.bin output.xml
```

Command Line Syntax

```
pittool analyzer Bson <input file> <output file>
```

input file

File containing BSON encoded data.

output file

File creating containing generated data model

Attributes

Required:

There are no required attributes.

Optional:

There are no optional attributes.

Examples

Example 38. BSON Inline Example

This example uses the BSON analyzer on inline data.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="TheModel">
        <Blob valueType="hex" value=
'\x16\x00\x00\x00\x02\x68\x65\x6c\x6c\x6f\x00\x06\x00\x00\x00\x77\x6f\x72\x6c\x64\x00
\x00'>
            <Analyzer class="Bson" />
        </Blob>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="TheModel" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>
        <Publisher class="ConsoleHex" />
    </Test>
</Peach>
```

Output from inline example.

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 49484.
2018-03-28 12:36:39.4049 Peach.Core.Engine runTest: Iteration Starting: 1,
=====
[R1,-,-] Performing iteration
2018-03-28 12:36:39.4370 Peach.Core.Engine runTest: Performing control recording
iteration.
2018-03-28 12:36:39.5127 Peach.Core.Dom.StateModel Run(): Changing to state
"initial".
2018-03-28 12:36:39.5192 Peach.Core.Dom.Action Run(Action): Output
2018-03-28 12:36:39.6926 Peach.Pro.Core.Publishers.ConsolePublisher start()
2018-03-28 12:36:39.6926 Peach.Pro.Core.Publishers.ConsolePublisher open()
2018-03-28 12:36:39.6926 Peach.Pro.Core.Publishers.ConsolePublisher output(22 bytes)
00000000 16 00 00 00 02 68 65 6C 6C 6F 00 06 00 00 00 77 ....hello....w
00000010 6F 72 6C 64 00 00
                                orld..
2018-03-28 12:36:39.6926 Peach.Pro.Core.Publishers.ConsolePublisher close()
2018-03-28 12:36:39.7086 Peach.Core.Engine runTest: context.config.singleIteration ==
true
2018-03-28 12:36:39.7086 Peach.Core.Engine All test cases executed, stopping engine.
2018-03-28 12:36:39.7086 Peach.Pro.Core.Publishers.ConsolePublisher stop()
2018-03-28 12:36:39.7086 Peach.Core.Engine EndTest: Stopping all agents and monitors

[*] Test 'Default' finished.
```

This example uses the BSON analyzer on an external file containing BSON encoded data.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="TheModel">
        <Blob>
            <Analyzer class="Bson" />
        </Blob>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="TheModel" />
                <Data fileName="example bson"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>
        <Publisher class="ConsoleHex" />
    </Test>
</Peach>

```

Output from BSON data file example.

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 36094.
2018-03-28 12:41:03.4793 Peach.Core.Engine runTest: Iteration Starting: 1,
=====
[R1,-,-] Performing iteration
2018-03-28 12:41:03.5129 Peach.Core.Engine runTest: Performing control recording
iteration.
2018-03-28 12:41:03.5329 DataCracker -+ DataModel 'TheModel', Bytes: 0/22, Bits: 0/176
2018-03-28 12:41:03.5329 DataCracker | Size: ??? (Deterministic)
2018-03-28 12:41:03.5329 DataCracker |-- Blob 'DataElement_0', Bytes: 0/22, Bits: 0/176
2018-03-28 12:41:03.5464 DataCracker |   Size: 22 bytes | 176 bits (Last Unsized)
2018-03-28 12:41:03.5464 DataCracker |   Value: 16 00 00 00 02 68 65 6c 6c 6f 00 06 00
00 00 77 6f 72 6c 64 00 00
2018-03-28 12:41:03.5464 DataCracker /
2018-03-28 12:41:03.6427 Peach.Core.Dom.StateModel Run(): Changing to state "initial".
2018-03-28 12:41:03.6427 Peach.Core.Dom.Action Run(Action): Output
2018-03-28 12:41:03.7630 Peach.Pro.Core.Publishers.ConsolePublisher start()
2018-03-28 12:41:03.7630 Peach.Pro.Core.Publishers.ConsolePublisher open()
2018-03-28 12:41:03.7655 Peach.Pro.Core.Publishers.ConsolePublisher output(22 bytes)
00000000 16 00 00 00 02 68 65 6C 6C 6F 00 06 00 00 00 77 ....hello....w
00000010 6F 72 6C 64 00 00                           orld..
2018-03-28 12:41:03.7655 Peach.Pro.Core.Publishers.ConsolePublisher close()
2018-03-28 12:41:03.7655 Peach.Core.Engine runTest: context.config.singleIteration ==
true
2018-03-28 12:41:03.7655 Peach.Core.Engine All test cases executed, stopping engine.
2018-03-28 12:41:03.7806 Peach.Pro.Core.Publishers.ConsolePublisher stop()
2018-03-28 12:41:03.7806 Peach.Core.Engine EndTest: Stopping all agents and monitors

[*] Test 'Default' finished.
```

21.2.4. JSON Analyzer

This analyzer converts JSON strings into a full data model.

When used in the DataModel section of a Peach Pit, the Fuzzer walks the JSON data and creates the appropriate elements.

When used from the command line, the Fuzzer walks the JSON data, creates the appropriate elements, and saves the results of the generated model to disk. Once saved, you can use and modify the results as needed.



The JSON analyzer requires data from a String data type.

Syntax

```
<String name="JsonData">
    <Analyzer class="Json" />
</String>
```

```
pittool analyzer Json input.txt output.xml
```

Command Line Syntax

```
pittool analyzer Json <input file> <output file>
```

input file

File containing JSON encoded data.

output file

File creating containing generated data model

Attributes

Required:

There are no required attributes.

Optional:

There are no optional attributes.

Examples

Example 39. JSON Inline Example

This example uses the JSON analyzer on inline data.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="TheModel">
        <String value=
'{"Foo": ["Bar", 1, null], "Null": null, "Bool": false, "Obj": {"Num": 1, "Str": "StringValue"}, "Double": 1.2}'>
            <Analyzer class="Json" />
        </String>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="TheModel" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>
        <Publisher class="ConsoleHex" />
    </Test>
</Peach>
```

Output from inline example.

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 56481.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "initial".
Peach.Core.Dom.Action Run(Action): Output
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(96 bytes)
00000000  7B 22 46 6F 6F 22 3A 5B  22 42 61 72 22 2C 31 2C  {"Foo": ["Bar", 1,
00000010  6E 75 6C 6C 5D 2C 22 4E  75 6C 6C 22 3A 6E 75 6C  null], "Null": nul
00000020  6C 2C 22 42 6F 6F 6C 22  3A 66 61 6C 73 65 2C 22  l, "Bool": false,
00000030  4F 62 6A 22 3A 7B 22 4E  75 6D 22 3A 31 2C 22 53  Obj": {"Num": 1, "S
00000040  74 72 22 3A 22 53 74 72  69 6E 67 56 61 6C 75 65  tr": "StringValue
00000050  22 7D 2C 22 44 6F 75 62  6C 65 22 3A 31 2E 32 7D  "}, "Double": 1.2}
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

This example uses the JSON analyzer on an external file containing JSON encoded data.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="TheModel">
        <String>
            <Analyzer class="Json" />
        </String>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="TheModel" />
                <Data name="Json" fileName="json.json"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>
        <Publisher class="ConsoleHex" />
    </Test>
</Peach>

```

Output from JSON data file example.

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 33175.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'TheModel' Bytes: 0/96, Bits: 0/768
Peach.Core.Cracker.DataCracker getSize: -----> DataModel 'TheModel'
Peach.Core.Cracker.DataCracker scan: DataModel 'TheModel'
Peach.Core.Cracker.DataCracker scan: String 'TheModel.DataElement_0' -> Offset: 0,
Unsized element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: DataModel 'TheModel' Size: <null>, Bytes: 0/96,
Bits: 0/768
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker String 'TheModel.DataElement_0' Bytes: 0/96, Bits: 0/768
Peach.Core.Cracker.DataCracker getSize: -----> String 'TheModel.DataElement_0'
Peach.Core.Cracker.DataCracker scan: String 'TheModel.DataElement_0' -> Offset: 0,
Unsized element
Peach.Core.Cracker.DataCracker lookahead: String 'TheModel.DataElement_0'
Peach.Core.Cracker.DataCracker getSize: <----- Last Unsized: 768
Peach.Core.Cracker.DataCracker Crack: String 'TheModel.DataElement_0' Size: 768, Bytes:
0/96, Bits: 0/768
Peach.Core.Dom.DataElement String 'TheModel.DataElement_0' value is:
>{"Foo": ["Bar", 1, null], "Null": null, "Bool": false, "Obj": {"Num": 1, "S.. (Len: 96 chars)
Peach.Core.Dom.StateModel Run(): Changing to state "initial".
Peach.Core.Dom.Action Run(Action): Output
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(96 bytes)
00000000  7B 22 46 6F 6F 22 3A 5B  22 42 61 72 22 2C 31 2C  {"Foo": ["Bar", 1,
00000010  6E 75 6C 6C 5D 2C 22 4E  75 6C 6C 22 3A 6E 75 6C  null], "Null": nul
00000020  6C 2C 22 42 6F 6F 22 3A  66 61 6C 73 65 2C 22 1, "Bool": false,
00000030  4F 62 6A 22 3A 7B 22 4E  75 6D 22 3A 31 2C 22 53  Obj": {"Num": 1, "S
00000040  74 72 22 3A 22 53 74 72  69 6E 67 56 61 6C 75 65  tr": "StringValue
00000050  22 7D 2C 22 44 6F 75 62  6C 65 22 3A 31 2E 32 7D  "}, "Double": 1.2}
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

21.2.5. Postman Analyzer

This analyzer converts Postman Collections into Peach Pits for fuzzing WebApi style web service endpoints.

[Postman](#) is a popular app used during development and testing of WebApi style web services. APIs are organized into Collections which can be converted into fuzzers using this analyzer.

After converting, some manual work is usually needed to have a fully working pit. This includes:

- Hooking up any authentication/authorization
- Adding slurps for resource identifiers
- Adding cleanup states to remove created resources
- Testing to verify it all works



This analyzer is intended to always be run from the command line.

Syntax

```
pittool analyzer Postman catalog.json web_api_pit.xml
```

Command Line Syntax

```
pittool analyzer Postman <input file> <output file>
```

input file

Postman Catalog file to convert

output file

Generated PIT file

Attributes

Required:

There are no required attributes.

Optional:

There are no optional attributes.

Examples

No examples.

21.2.6. Regex Analyzer

The Regex Analyzer parses a string and breaks it into substrings using a regular expression to define parsing details. This analyzer provides a quick way to parse string-based data.

A regular expression provides a pattern that the parser attempts to match while walking through the input string data. A pattern can consist of a single specification. Or, a pattern can consist of groups where each group describes a substring or a part of the overall pattern. With Peach, if a group is named, you can optionally name the corresponding substring.

When used in the DataModel section of a Peach Pit, the Fuzzer walks the input string data and creates the appropriate substrings.

When used from the command line, the Fuzzer walks the input string data, creates the appropriate substrings, and saves the results of the generated model to disk. Once saved, you can use and modify the results as needed.



The regular expression syntax is consistent with the regular expression parser used by Microsoft.NET. For more information, see the [quick reference](#).

Syntax

```
<String name="Value" value="http://www.google.com/q?q=kitty">
  <Analyzer class="Regex">
    <!-- (?<protocol>\w+)(://)?<host>[^/?]+(?<path>[^?]+)(\?)?(?<query>.*)
    <Param name="Regex" value="(?&lt;protocol&gt;\w+)(://)(?&lt;host&gt;[^/?]+)(?&lt;path&gt;[^?]+)(\?)(?&lt;query&gt;.*)" />
  </Analyzer>
</String>
```

```
pittool analyzer Regex "(\w+)(.)" input.txt output.xml
```

Command Line Syntax

```
pittool analyzer Regex <regex> <input file> <output file>
```

regex

Regular Expression to base tokens on

input file

Text file

output file

Generated PIT file

Attributes

Required:

None.

Optional:

None.

Parameters

Regex

Regular expression. Only groups are kept. Group names are used to name specific elements and cannot be duplicated.

Examples

Example 40. Simple HTTP GET URL Example

This example uses the string tokenizer on an inline value.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

  <DataModel name="TheDataModel">
    <String name="Value" value="http://www.google.com/q?q=kitty">
      <Analyzer class="Regex">
        <!-- (?<protocol>\w+)(://)(?<host>[^/?]+)(?<path>[^?]+)(\?)(?<query>.*)> -->
        <Param name="Regex" value="(?&lt;protocol&gt;\w+)(://)(?&lt;host&gt;[^/?]+)(?&lt;path&gt;[^?]+)(\?)(?&lt;query&gt;.*)" />
      </Analyzer>
    </String>
  </DataModel>

  <StateModel name="State" initialState="State1" >
    <State name="State1" >
      <Action type="output" >
        <DataModel ref="TheDataModel"/>
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="State"/>

    <Publisher class="Console" />
  </Test>
</Peach>
<!-- end -->

```

Output from the example using the string tokenizer and an inline value.

```
> peach -1 --debug RegexAnalyzer.xml

[*] Web site running at: http://localhost:8889/

[*] Test 'Default' starting with random seed 43577.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "State1".
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(31 bytes)
00000000  68 74 74 70 3A 2F 2F 77 77 77 2E 67 6F 6F 67 6C  http://www.googl
00000010  65 2E 63 6F 6D 2F 71 3F 71 3D 6B 69 74 74 79      e.com/q?q=kitty
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

Output in Peach Validator

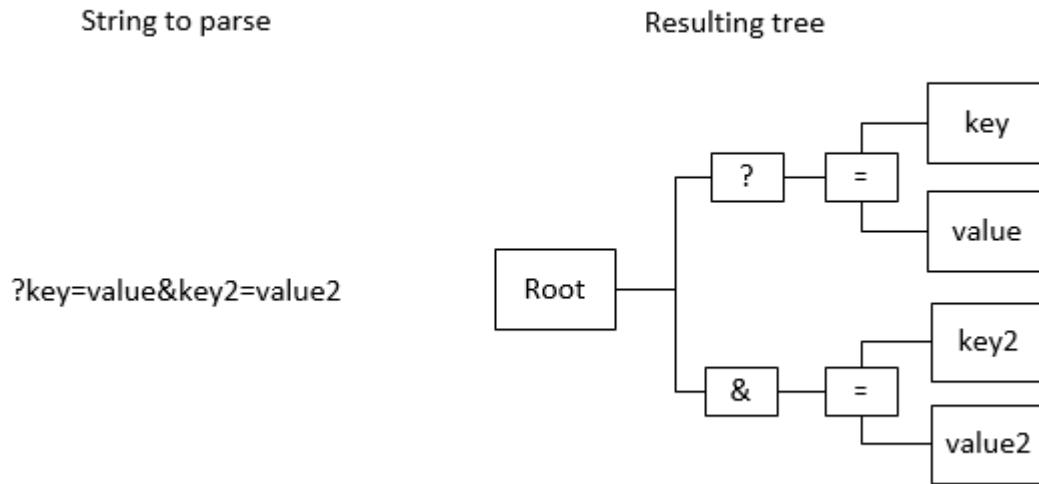
Pit: TheDataModel

Sample file:

Name	Position	Length	Value
TheDataModel	0	0	
Value	0	0	
protocol	0	0	http
1	0	0	::/
host	0	0	www.google.com
path	0	0	/q
2	0	0	?
query	0	0	q=kitty

21.2.7. String Token Analyzer

The String Token Analyzer creates a tokenized tree from an input string based on special characters and punctuation in the string. For example, the string "?key=value&key2=value2" creates a tree similar to the following:



The tree allows Peach to perform mutations that can expose bugs in parsers.

When run using the command line, the results of the generated model are saved to disk. Once saved, you can use and modify the results as needed.

Syntax

```
<String value="GET /index.html?name=peach&testing=true&admin=false&debug=1"
/>
<Analyzer class="StringToken" />
</String>
```

```
pittool analyzer StringTokenizer input.txt output.xml
```

Command Line Syntax

```
pittool analyzer StringTokenizer <input file> <output file>
```

input file

Text file

output file

Generated PIT file

Attributes

Required:

None.

Optional:

None.

Parameters

Tokens

List of characters to pass to the StringTokenizer analyzer.

Examples

Example 41. Simple HTTP GET URL Example

This example uses the string tokenizer on an inline value.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="TheModel">
        <String value="GET /index.html?name=peach&testing=true&admin=false
&debug=1" />
            <Analyzer class="StringToken" />
        </String>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="TheModel" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>
        <Publisher class="ConsoleHex" />
    </Test>
</Peach>

```

Output from the example that uses the string tokenizer on an inline value.

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 26956.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(59 bytes)
00000000  47 45 54 20 2F 69 6E 64  65 78 2E 68 74 6D 6C 3F   GET /index.html?
00000010  6E 61 6D 65 3D 70 65 61  63 68 26 74 65 73 74 69   name=peach&testi
00000020  6E 67 3D 74 72 75 65 26  61 64 6D 69 6E 3D 66 61   ng=true&admin=fa
00000030  6C 73 65 26 64 65 62 75  67 3D 31                   lse&debug=1
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

Output in Peach Validator

Name	Position	Length	Value
TheModel	0	0	
DataElement_0	0	0	
DataElement_0	0	0	
DataElement_1	0	0	GET
DataElement_2	0	0	
DataElement_3	0	0	
DataElement_4	0	0	
DataElement_31	0	0	
DataElement_32	0	0	/
DataElement_33	0	0	index
DataElement_5	0	0	.
DataElement_6	0	0	
DataElement_7	0	0	html
DataElement_8	0	0	?
DataElement_9	0	0	
DataElement_10	0	0	
DataElement_19	0	0	name
DataElement_20	0	0	=
DataElement_21	0	0	peach
DataElement_11	0	0	&
DataElement_12	0	0	
DataElement_13	0	0	
DataElement_22	0	0	testing
DataElement_23	0	0	=
DataElement_24	0	0	true
DataElement_14	0	0	&
DataElement_15	0	0	
DataElement_16	0	0	
DataElement_25	0	0	admin
DataElement_26	0	0	=
DataElement_27	0	0	false
DataElement_17	0	0	&
DataElement_18	0	0	
DataElement_28	0	0	debug
DataElement_29	0	0	=
DataElement_30	0	0	1

Example 42. Custom String Tokens Example

This example uses the StringTokenizer analyzer on a string with a custom set of tokens.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="TheModel">
        <String name="TheString" value="(abc:{123,xyz}, def:{456})">
            <Analyzer class="StringToken">
                <Param name="Tokens" value="{}{},:{}"/>
            </Analyzer>
        </String>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="TheModel" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>
        <Publisher class="ConsoleHex" />
    </Test>
</Peach>

```

Output from the example of the string that contains custom tokens.

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 56835.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(26 bytes)
00000000  28 61 62 63 3A 7B 31 32  33 2C 78 79 7A 7D 2C 20  (abc:{123,xyz},
00000010  64 65 66 3A 7B 34 35 36  7D 29                      def:{456})
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

Output in Peach Validator

Name	Position	Length	Value
TheModel	0	0	
TheString	0	0	
TheString	0	0	
DataElement_0	0	0	
DataElement_1	0	0	(
DataElement_2	0	0	
DataElement_3	0	0	
DataElement_6	0	0	
DataElement_24	0	0	abc
DataElement_25	0	0	:
DataElement_26	0	0	
DataElement_7	0	0	{
DataElement_8	0	0	
DataElement_9	0	0	
DataElement_12	0	0	
DataElement_18	0	0	123
DataElement_19	0	0	,
DataElement_20	0	0	xyz
DataElement_13	0	0	}
DataElement_14	0	0	
DataElement_21	0	0	
DataElement_22	0	0	,
DataElement_23	0	0	
DataElement_10	0	0	{
DataElement_11	0	0	
DataElement_15	0	0	456
DataElement_16	0	0	}
DataElement_17	0	0	
DataElement_4	0	0)
DataElement_5	0	0	

21.2.8. Swagger Analyzer

This analyzer converts Swagger API JSON into Peach Pits for fuzzing WebApi style web service endpoints.

[Swagger](#) is a popular method for representing your RESTful API, especially for documentation purposes. Many frameworks can export Swagger API representations which can then be converted into partial pits using this analyzer.

After converting, some manual work is usually needed to have a fully working pit. This includes:

- Adding default values to models
- Hooking up any authentication/authorization
- Adding slurps for resource identifiers
- Adding cleanup states to remove created resources
- Testing to verify it all works



This analyzer is intended to always be run from the command line.

Syntax

```
pittool analyzer Swagger swagger.json web_api_pit.xml
```

Command Line Syntax

```
pittool analyzer Swagger <input file> <output file>
```

input file

Swagger API JSON

output file

Generated PIT file

Attributes

Required:

There are no required attributes.

Optional:

There are no optional attributes.

Examples

No examples.

21.2.9. Vcr Analyzer

This analyzer converts Vcr cassettes into Peach Pits for fuzzing HTTP requests.

A cassette is a set of recorded interactions serialized to a specific format. A cassette has a list (or array) of interactions and information about the library that recorded it.

After converting, some manual work is usually needed to have a fully working pit. This includes:

- Adding default values to models
- Hooking up any authentication/authorization
- Adding slurps for resource identifiers
- Adding cleanup states to remove created resources
- Testing to verify it all works



This analyzer is intended to always be run from the command line.

Syntax

```
pittool analyzer Vcr cassette.json http_pit.xml
```

Command Line Syntax

```
pittool analyzer Vcr <input file> <output file>
```

input file

Vcr cassette JSON

output file

Generated PIT file

Attributes

Required:

There are no required attributes.

Optional:

There are no optional attributes.

Examples

No examples.

21.2.10. Xml Analyzer

The XML Analyzer consumes an XML document or fragment and converts it into a tree structure of [XmlElement](#) and [XmlAttribute](#) elements.

When run using the command line, the results of the generated model are saved to disk. Once saved, you can use and modify the results as needed.

This analyzer can be attached to a string.

The following XML special characters need to be encoded if they are inline values.



Symbol	Name	Escape Sequence
"	double-quote	"
'	single-quote	'
<	less-than	<
>	greater-than	>
&	ampersand	&

Syntax

```
<String value="&lt;Root&gt;HelloWorld!&lt;/Root&gt;">
    <Analyzer class="Xml" />
</String>
```

```
pittool analyzer Xml input.xml output.xml
```

Command Line Syntax

```
pittool analyzer Xml <input file> <output file>
```

input file

XML file

output file

Generated PIT file

Attributes

Required:

None.

Examples

Example 43. Simple Hello World Example

This example uses the XML analyzer on an inline value.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="TheModel">
        <String value="&lt;Root&gt;HelloWorld!&lt;/Root&gt;">
            <Analyzer class="Xml" />
        </String>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="TheModel" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>
        <Publisher class="ConsoleHex" />
    </Test>
</Peach>
```

Output of the example using an inline value.

```

>peach -1 --debug example.xml

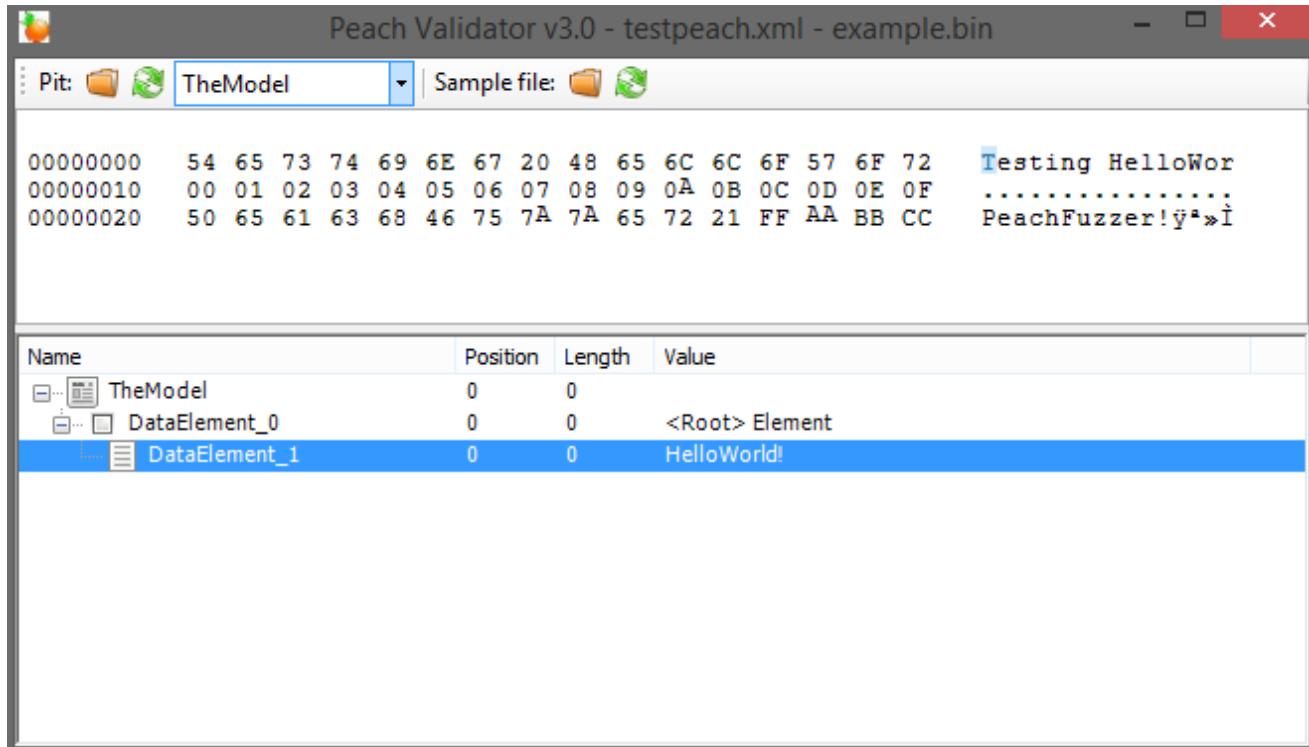
[*] Test 'Default' starting with random seed 22910.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(24 bytes)
00000000  3C 52 6F 6F 74 3E 48 65  6C 6C 6F 57 6F 72 6C 64  <Root>HelloWorld
00000010  21 3C 2F 52 6F 6F 74 3E                                         !</Root>
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

Output in Peach Validator



Example 44. Inline XML Inline Encoding Example

This example uses the Xml analyzer on an inline value with encoded Xml characters.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="TheModel">
        <String value="&lt;Root test="true">HelloWorld!&amp;#
&lt;/Root&gt;">
            <Analyzer class="Xml" />
        </String>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="TheModel" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>
        <Publisher class="ConsoleHex" />
    </Test>
</Peach>

```

Output from the example using an inline value with encoded Xml characters.

```

>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 44192.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(37 bytes)
00000000  3C 52 6F 6F 74 20 74 65  73 74 3D 22 74 72 75 65  <Root test="true
00000010  22 3E 48 65 6C 6C 6F 57  6F 72 6C 64 21 26 3C 2F  ">HelloWorld!&lt;
00000020  52 6F 6F 74 3E                                     Root>
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

Output in Peach Validator

The screenshot shows the Peach Validator v3.0 application window. The title bar reads "Peach Validator v3.0 - testpeach.xml - example.bin". The main area has two tabs: "Pit" and "Sample file". The "Pit" tab is selected, showing a hex dump of the generated payload:

00000000	54 65 73 74 69 6E 67 20 48 65 6C 6C 6F 57 6F 72	Testing HelloWor
00000010	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000020	50 65 61 63 68 46 75 7A 7A 65 72 21 FF AA BB CC	PeachFuzzer!ÿ»í

Below the hex dump is a tree view of the model structure:

Name	Position	Length	Value
TheModel	0	0	
DataElement_0	0	0	<Root> Element
DataElement_1	0	0	'test' Attribute
DataElement_2	0	0	true
DataElement_3	0	0	HelloWorld!&

Example 45. Load XML From File Example

This example uses the Xml analyzer on itself.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="TheModel">
        <String>
            <Analyzer class="Xml" />
        </String>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="TheModel" />
                <Data name="TheData" fileName="example.xml"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>
        <Publisher class="ConsoleHex" />
    </Test>
</Peach>

```

Output from the example that uses the Xml analyzer on itself.

```

>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 10150.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'TheModel' Bytes: 0/787, Bits: 0/6296
Peach.Core.Cracker.DataCracker getSize: -----> DataModel 'TheModel'
Peach.Core.Cracker.DataCracker scan: DataModel 'TheModel'
Peach.Core.Cracker.DataCracker scan: String 'TheModel.DataElement_0' -> Offset: 0,
Unsized element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: DataModel 'TheModel' Size: <null>, Bytes:
0/787, Bits: 0/6296
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker String 'TheModel.DataElement_0' Bytes: 0/787, Bit s:
0/6296

```

```

Peach.Core.Cracker.getSize: -----> String 'TheModel.DataElement_0'
Peach.Core.Cracker.scan: String 'TheModel.DataElement_0' -> Offset: 0,
Unsized element
Peach.Core.Cracker.lookahead: String 'TheModel.DataElement_0'
Peach.Core.Cracker.getSize: <----- Last Unsized: 6296
Peach.Core.Cracker.Crack: String 'TheModel.DataElement_0' Size: 6296 ,
Bytes: 0/787, Bits: 0/6296
Peach.Core.Dom.DataElement String 'TheModel.DataElement_0' value is: <?xml
version="1.0" encoding="utf-8"?> <Peach xmlns="http://pea.. (Len: 787 chars)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(669 bytes)
00000000 3C 50 65 61 63 68 20 78 6D 6C 6E 73 3D 22 68 74 <Peach xmlns="ht
00000010 74 70 3A 2F 2F 70 65 61 63 68 66 75 7A 7A 65 72 tp://peachfuzzer
00000020 2E 63 6F 6D 2F 32 30 31 32 2F 50 65 61 63 68 22 .com/2012/Peach"
00000030 20 78 6D 6C 6E 73 3A 78 73 69 3D 22 68 74 74 70 xmlns:xsi="http
00000040 3A 2F 2F 77 77 77 2E 77 33 2E 6F 72 67 2F 32 30 ://www.w3.org/20
00000050 30 31 2F 58 4D 4C 53 63 68 65 6D 61 2D 69 6E 73 01/XMLSchema-ins
00000060 74 61 6E 63 65 22 20 64 31 70 31 3A 73 63 68 65 tance" d1p1:sche
00000070 6D 61 4C 6F 63 61 74 69 6F 6E 3D 22 68 74 74 70 maLocation="http
00000080 3A 2F 2F 70 65 61 63 68 66 75 7A 7A 65 72 2E 63 ://peachfuzzer.c
00000090 6F 6D 2F 32 30 31 32 2F 50 65 61 63 68 20 2E 2E om/2012/Peach ..
000000A0 2F 70 65 61 63 68 2E 78 73 64 22 20 78 6D 6C 6E /peach.xsd" xmln
000000B0 73 3A 64 31 70 31 3D 22 68 74 74 70 3A 2F 2F 77 s:d1p1="http://w
000000C0 77 77 2E 77 33 2E 6F 72 67 2F 32 30 30 31 2F 58 ww.w3.org/2001/X
000000D0 4D 4C 53 63 68 65 6D 61 2D 69 6E 73 74 61 6E 63 MLSchema-instanc
000000E0 65 22 3E 3C 44 61 74 61 4D 6F 64 65 6C 20 6E 61 e"><DataModel na
000000F0 6D 65 3D 22 54 68 65 4D 6F 64 65 6C 22 3E 3C 53 me="TheModel"><S
00000100 74 72 69 6E 67 3E 3C 41 6E 61 6C 79 7A 65 72 20 tring><Analyzer
00000110 63 6C 61 73 73 3D 22 58 6D 6C 22 20 2F 3E 3C 2F class="Xml" /></
00000120 53 74 72 69 6E 67 3E 3C 2F 44 61 74 61 4D 6F 64 String></DataMod
00000130 65 6C 3E 3C 53 74 61 74 65 4D 6F 64 65 6C 20 6E el><StateModel n
00000140 61 6D 65 3D 22 54 68 65 53 74 61 74 65 22 20 69 ame="TheState" i
00000150 6E 69 74 69 61 6C 53 74 61 74 65 3D 22 69 6E 69 nitialState="ini
00000160 74 69 61 6C 22 3E 3C 53 74 61 74 65 20 6E 61 6D tial"><State nam
00000170 65 3D 22 69 6E 69 74 69 61 6C 22 3E 3C 41 63 74 e="initial"><Act
00000180 69 6F 6E 20 74 79 70 65 3D 22 6F 75 74 70 75 74 ion type="output
00000190 22 3E 3C 44 61 74 61 4D 6F 64 65 6C 20 72 65 66 "><DataModel ref
000001A0 3D 22 54 68 65 4D 6F 64 65 6C 22 20 2F 3E 3C 44 ="TheModel" /><D
000001B0 61 74 61 20 6E 61 6D 65 3D 22 45 78 61 6D 70 6C ata name="Exampl
000001C0 65 22 20 66 69 6C 65 4E 61 6D 65 3D 22 74 65 73 e" fileName="tes
000001D0 74 70 65 61 63 68 2E 78 6D 6C 22 20 2F 3E 3C 2F tpeach.xml" /></
000001E0 41 63 74 69 6F 6E 3E 3C 2F 53 74 61 74 65 3E 3C Action></State><
000001F0 2F 53 74 61 74 65 4D 6F 64 65 6C 3E 3C 54 65 73 /StateModel><Tes
00000200 74 20 6E 61 6D 65 3D 22 44 65 66 61 75 6C 74 22 t name="Default"
00000210 3E 3C 53 74 61 74 65 4D 6F 64 65 6C 20 72 65 66 ><StateModel ref

```

```
00000220 3D 22 54 68 65 53 74 61 74 65 22 20 2F 3E 3C 50 ="TheState" /><P  
00000230 75 62 6C 69 73 68 65 72 20 63 6C 61 73 73 3D 22 ublisher class="  
00000240 43 6F 6E 73 6F 6C 65 48 65 78 22 20 2F 3E 3C 4C onsoleHex" /><L  
00000250 6F 67 67 65 72 20 63 6C 61 73 73 3D 22 46 69 6C ogger class="Fil  
00000260 65 22 3E 3C 50 61 72 61 6D 20 6E 61 6D 65 3D 22 e"><Param name="Path" value="log  
00000270 50 61 74 68 22 20 76 61 6C 75 65 3D 22 6C 6F 67 s" /></Logger></  
00000280 73 22 20 2F 3E 3C 2F 4C 6F 67 67 65 72 3E 3C 2F Test></Peach>  
00000290 54 65 73 74 3E 3C 2F 50 65 61 63 68 3E  
  
Peach.Core.Publisher.ConsolePublisher close()  
Peach.Core.Engine runTest: context.config.singleIteration == true  
Peach.Core.Publisher.ConsolePublisher stop()  
  
[*] Test 'Default' finished.
```

Output in Peach Validator

Pit: TheModel | Sample file:

```

<?xml version="1.0" encoding="utf-8"?>..<Peach
    xmlns="http://peachfuzzer.com/2012/Peach"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ./peach.xsd">.... <DataModel name="TheModel">.. <String> .. <Analyzer class="XML1"/>.. </String>.. </DataModel>.... <StateModel name="TheState" initialState="initial">.. <State name=>

```

Name	Position	Length	Value
TheModel	0	787	
DataElement_0	0	0	<Peach> Element
DataElement_1	0	0	'xmlns' Attribute
DataElement_2	0	0	http://peachfuzzer.com/2012/Peach
DataElement_3	0	0	'xmlns:xsi' Attribute
DataElement_4	0	0	http://www.w3.org/2001/XMLSchema-instance
DataElement_5	0	0	'xsi:schemaLocation' Attribute
DataElement_6	0	0	http://peachfuzzer.com/2012/Peach ..//peach.xsd
DataElement_7	0	0	<DataModel> Element
DataElement_14	0	0	<StateModel> Element
DataElement_15	0	0	'name' Attribute
DataElement_16	0	0	TheState
DataElement_17	0	0	'initialState' Attribute
DataElement_18	0	0	initial
DataElement_19	0	0	<State> Element
DataElement_20	0	0	'name' Attribute
DataElement_21	0	0	initial
DataElement_22	0	0	<Action> Element
DataElement_23	0	0	'type' Attribute
DataElement_24	0	0	output
DataElement_25	0	0	<DataModel> Element
DataElement_26	0	0	'ref' Attribute
DataElement_27	0	0	TheModel
DataElement_28	0	0	<Data> Element
DataElement_29	0	0	'name' Attribute
DataElement_30	0	0	Example
DataElement_31	0	0	'fileName' Attribute
DataElement_32	0	0	testpeach.xml
DataElement_33	0	0	<Test> Element
DataElement_34	0	0	'name' Attribute
DataElement_36	0	0	<StateModel> Element
DataElement_37	0	0	'ref' Attribute
DataElement_38	0	0	TheState
DataElement_39	0	0	<Publisher> Element
DataElement_40	0	0	'class' Attribute
DataElement_41	0	0	ConsoleHex
DataElement_42	0	0	<Logger> Element
DataElement_43	0	0	'class' Attribute
DataElement_44	0	0	File
DataElement_45	0	0	<Param> Element
DataElement_46	0	0	'name' Attribute
DataElement_47	0	0	Path
DataElement_48	0	0	'value' Attribute
DataElement_49	0	0	logs

21.2.11. WebRecordProxy Analyzer

This analyzer starts an HTTP proxy and records all HTTP requests. The recorded requests are then converted into a Peach Pit and written to disk for future use. This analyzer produces pits using the [WebApi](#) publisher and [web](#) action.

After converting, some manual work is usually needed to have a fully working pit. This includes:

- Hooking up any authentication/authorization
- Adding slurps for resource identifiers
- Adding cleanup states to remove created resources
- Testing to verify it all works



This analyzer is intended to always be run from the command line.

Syntax

```
pittool analyzer WebRecordProxy 8080 web_api_pit.xml
```

Command Line Syntax

```
pittool analyzer WebRecordProxy <proxy port> <output file>
```

proxy port

TCP port used for incoming requests

output file

Filename of generated Peach Pit

Attributes

Required:

There are no required attributes.

Optional:

There are no optional attributes.

Examples

No examples.

21.2.12. Zip Analyzer

The Zip Analyzer consumes a zip compressed archive and converts it into a sequence of [Stream](#) elements.

The analyzer creates a [Stream](#) element for each file in the zip archive. The name of the stream element corresponds to the name of the file in the zip archive. The content of a stream element corresponds to the content of the file in the zip archive.

The Zip Analyzer allows the content of each file in the zip archive to map to a [DataModel](#). Using a file and the corresponding data model, Peach can further decompose the contents of a file in a zip archive instead of just treating the contents as a [Blob](#). The [Map](#) parameter of this element defines the mappings for the streams in the zip files and the associated data models.

You can attach this analyzer to a blob.

Syntax

```
<Blob>
  <Analyzer class="Zip" />
</Blob>
```

Attributes

Required:

None.

Optional:

None.

Parameters

Map

Controls mapping of stream names to the corresponding data models. This parameter consists of a comma-separated list of mappings. Each mapping consists of a tuple containing a regular expression and a corresponding DataModel name. The parts of a tuple are delineated with a forward slash character, similar to the substitution regular expression pattern.

When the regular expression matches the file name in the zip archive, Peach uses the corresponding DataModel on the file contents.

For example, to use the data model [TextModel](#) for all files ending in `.txt`, set the value of the parameter to `/.txt$/TextModel/`.

Examples

Example 46. Analyzing a .docx file

This example uses the zip analyzer to decompose a .docx file into its underlying streams.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="WordDoc">
        <Blob name="Data">
            <Analyzer class="Zip"/>
        </Blob>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="WordDoc"/>
                <Data fileName="example.docx"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="Zip">
            <Param name="FileName" value="fuzzed.docx" />
        </Publisher>
    </Test>
</Peach>
```

Output from the example that decomposes a .docx file into its underlying streams.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 22910.

[*] Test 'Default' starting with random seed 48751.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
```

```
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'WordDoc' Bytes: 0/11272, Bits: 0/90176
Peach.Core.Cracker.getSize: -----> DataModel 'WordDoc'
Peach.Core.Cracker.scan: DataModel 'WordDoc'
Peach.Core.Cracker.scan: Blob 'WordDoc.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.getSize: <----- Deterministic: ???
Peach.Core.Cracker.Crack: DataModel 'WordDoc' Size: <null>, Bytes:
0/11272, Bits: 0/90176
Peach.Core.Cracker -----
Peach.Core.Cracker.Blob 'WordDoc.Data' Bytes: 0/11272, Bits: 0/90176
Peach.Core.Cracker.getSize: -----> Blob 'WordDoc.Data'
Peach.Core.Cracker.scan: Blob 'WordDoc.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.lookahead: Blob 'WordDoc.Data'
Peach.Core.Cracker.getSize: <----- Last Unsized: 90176
Peach.Core.Cracker.Crack: Blob 'WordDoc.Data' Size: 90176, Bytes:
0/11272, Bits: 0/90176
Peach.Core.Dom.DataElement Blob 'WordDoc.Data' value is: 50 4b 03 04 14 00 06 00 08
00 00 00 21 00 df a4 d2 6c 5a 01 00 00 20 05 00 00 13 00 08 02 5b 43.. (Len: 11272
bytes)
Peach.Enterprise.Analyzers.ZipAnalyzer Attempting to parse: [Content_Types].xml
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.Block 'Content' Bytes: 0/1312, Bits: 0/10496
Peach.Core.Cracker.getSize: -----> Block 'Content'
Peach.Core.Cracker.scan: Block 'Content'
Peach.Core.Cracker.scan: Blob 'Content.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.getSize: <----- Deterministic: ???
Peach.Core.Cracker.Crack: Block 'Content' Size: <null>, Bytes: 0/1312,
Bits: 0/10496
Peach.Core.Cracker -----
Peach.Core.Cracker.Blob 'Content.Data' Bytes: 0/1312, Bits: 0/10496
Peach.Core.Cracker.getSize: -----> Blob 'Content.Data'
Peach.Core.Cracker.scan: Blob 'Content.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.lookahead: Blob 'Content.Data'
Peach.Core.Cracker.getSize: <----- Last Unsized: 10496
Peach.Core.Cracker.Crack: Blob 'Content.Data' Size: 10496, Bytes: 0/1312,
Bits: 0/10496
Peach.Core.Dom.DataElement Blob 'Content.Data' value is: 3c 3f 78 6d 6c 20 76 65 72
73 69 6f 6e 3d 22 31 2e 30 22 20 65 6e 63 6f 64 69 6e 67 3d 22 55 54.. (Len: 1312
bytes)
Peach.Enterprise.Analyzers.ZipAnalyzer Successfully parsed: [Content_Types].xml
Peach.Enterprise.Analyzers.ZipAnalyzer Attempting to parse: _rels/.rels
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.Block 'Content' Bytes: 0/590, Bits: 0/4720
Peach.Core.Cracker.getSize: -----> Block 'Content'
```

```
Peach.Core.Cracker.DataCracker scan: Block 'Content'
Peach.Core.Cracker.DataCracker scan: Blob 'Content.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: Block 'Content' Size: <null>, Bytes: 0/590,
Bits: 0/4720
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Blob 'Content.Data' Bytes: 0/590, Bits: 0/4720
Peach.Core.Cracker.DataCracker getSize: -----> Blob 'Content.Data'
Peach.Core.Cracker.DataCracker scan: Blob 'Content.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker lookahead: Blob 'Content.Data'
Peach.Core.Cracker.DataCracker getSize: <----- Last Unsized: 4720
Peach.Core.Cracker.DataCracker Crack: Blob 'Content.Data' Size: 4720, Bytes: 0/590,
Bits: 0/4720
Peach.Core.Dom.DataElement Blob 'Content.Data' value is: 3c 3f 78 6d 6c 20 76 65 72
73 69 6f 6e 3d 22 31 2e 30 22 20 65 6e 63 6f 64 69 6e 67 3d 22 55 54.. (Len: 590
bytes)
Peach.Enterprise.Analyzers.ZipAnalyzer Successfully parsed: _rels/.rels
Peach.Enterprise.Analyzers.ZipAnalyzer Attempting to parse:
word/_rels/document.xml.rels
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Block 'Content' Bytes: 0/817, Bits: 0/6536
Peach.Core.Cracker.DataCracker getSize: -----> Block 'Content'
Peach.Core.Cracker.DataCracker scan: Block 'Content'
Peach.Core.Cracker.DataCracker scan: Blob 'Content.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: Block 'Content' Size: <null>, Bytes: 0/817,
Bits: 0/6536
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Blob 'Content.Data' Bytes: 0/817, Bits: 0/6536
Peach.Core.Cracker.DataCracker getSize: -----> Blob 'Content.Data'
Peach.Core.Cracker.DataCracker scan: Blob 'Content.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker lookahead: Blob 'Content.Data'
Peach.Core.Cracker.DataCracker getSize: <----- Last Unsized: 6536
Peach.Core.Cracker.DataCracker Crack: Blob 'Content.Data' Size: 6536, Bytes: 0/817,
Bits: 0/6536
Peach.Core.Dom.DataElement Blob 'Content.Data' value is: 3c 3f 78 6d 6c 20 76 65 72
73 69 6f 6e 3d 22 31 2e 30 22 20 65 6e 63 6f 64 69 6e 67 3d 22 55 54.. (Len: 817
bytes)
Peach.Enterprise.Analyzers.ZipAnalyzer Successfully parsed:
word/_rels/document.xml.rels
Peach.Enterprise.Analyzers.ZipAnalyzer Attempting to parse: word/document.xml
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Block 'Content' Bytes: 0/1620, Bits: 0/12960
Peach.Core.Cracker.DataCracker getSize: -----> Block 'Content'
```

```
Peach.Core.Cracker.DataCracker scan: Block 'Content'
Peach.Core.Cracker.DataCracker scan: Blob 'Content.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: Block 'Content' Size: <null>, Bytes: 0/1620,
Bits: 0/12960
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Blob 'Content.Data' Bytes: 0/1620, Bits: 0/12960
Peach.Core.Cracker.DataCracker getSize: -----> Blob 'Content.Data'
Peach.Core.Cracker.DataCracker scan: Blob 'Content.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker lookahead: Blob 'Content.Data'
Peach.Core.Cracker.DataCracker getSize: <----- Last Unsized: 12960
Peach.Core.Cracker.DataCracker Crack: Blob 'Content.Data' Size: 12960, Bytes: 0/1620,
Bits: 0/12960
Peach.Core.Dom.DataElement Blob 'Content.Data' value is: 3c 3f 78 6d 6c 20 76 65 72
73 69 6f 6e 3d 22 31 2e 30 22 20 65 6e 63 6f 64 69 6e 67 3d 22 55 54.. (Len: 1620
bytes)
Peach.Enterprise.Analyzers.ZipAnalyzer Successfully parsed: word/document.xml
Peach.Enterprise.Analyzers.ZipAnalyzer Attempting to parse: word/theme/theme1.xml
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Block 'Content' Bytes: 0/6795, Bits: 0/54360
Peach.Core.Cracker.DataCracker getSize: -----> Block 'Content'
Peach.Core.Cracker.DataCracker scan: Block 'Content'
Peach.Core.Cracker.DataCracker scan: Blob 'Content.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: Block 'Content' Size: <null>, Bytes: 0/6795,
Bits: 0/54360
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Blob 'Content.Data' Bytes: 0/6795, Bits: 0/54360
Peach.Core.Cracker.DataCracker getSize: -----> Blob 'Content.Data'
Peach.Core.Cracker.DataCracker scan: Blob 'Content.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker lookahead: Blob 'Content.Data'
Peach.Core.Cracker.DataCracker getSize: <----- Last Unsized: 54360
Peach.Core.Cracker.DataCracker Crack: Blob 'Content.Data' Size: 54360, Bytes: 0/6795,
Bits: 0/54360
Peach.Core.Dom.DataElement Blob 'Content.Data' value is: 3c 3f 78 6d 6c 20 76 65 72
73 69 6f 6e 3d 22 31 2e 30 22 20 65 6e 63 6f 64 69 6e 67 3d 22 55 54.. (Len: 6795
bytes)
Peach.Enterprise.Analyzers.ZipAnalyzer Successfully parsed: word/theme/theme1.xml
Peach.Enterprise.Analyzers.ZipAnalyzer Attempting to parse: word/settings.xml
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Block 'Content' Bytes: 0/2477, Bits: 0/19816
Peach.Core.Cracker.DataCracker getSize: -----> Block 'Content'
Peach.Core.Cracker.DataCracker scan: Block 'Content'
Peach.Core.Cracker.DataCracker scan: Blob 'Content.Data' -> Offset: 0, Unsized
```

```
element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker Crack: Block 'Content' Size: <null>, Bytes: 0/2477,
Bits: 0/19816
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Blob 'Content.Data' Bytes: 0/2477, Bits: 0/19816
Peach.Core.Cracker.getSize: -----> Blob 'Content.Data'
Peach.Core.Cracker.DataCracker scan: Blob 'Content.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker lookahead: Blob 'Content.Data'
Peach.Core.Cracker.getSize: <----- Last Unsized: 19816
Peach.Core.Cracker.DataCracker Crack: Blob 'Content.Data' Size: 19816, Bytes: 0/2477,
Bits: 0/19816
Peach.Core.Dom.DataElement Blob 'Content.Data' value is: 3c 3f 78 6d 6c 20 76 65 72
73 69 6f 6e 3d 22 31 2e 30 22 20 65 6e 63 6f 64 69 6e 67 3d 22 55 54.. (Len: 2477
bytes)
Peach.Enterprise.Analyzers.ZipAnalyzer Successfully parsed: word/settings.xml
Peach.Enterprise.Analyzers.ZipAnalyzer Attempting to parse: word/fontTable.xml
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Block 'Content' Bytes: 0/1261, Bits: 0/10088
Peach.Core.Cracker.getSize: -----> Block 'Content'
Peach.Core.Cracker.DataCracker scan: Block 'Content'
Peach.Core.Cracker.DataCracker scan: Blob 'Content.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: Block 'Content' Size: <null>, Bytes: 0/1261,
Bits: 0/10088
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Blob 'Content.Data' Bytes: 0/1261, Bits: 0/10088
Peach.Core.Cracker.getSize: -----> Blob 'Content.Data'
Peach.Core.Cracker.DataCracker scan: Blob 'Content.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker lookahead: Blob 'Content.Data'
Peach.Core.Cracker.getSize: <----- Last Unsized: 10088
Peach.Core.Cracker.DataCracker Crack: Blob 'Content.Data' Size: 10088, Bytes: 0/1261,
Bits: 0/10088
Peach.Core.Dom.DataElement Blob 'Content.Data' value is: 3c 3f 78 6d 6c 20 76 65 72
73 69 6f 6e 3d 22 31 2e 30 22 20 65 6e 63 6f 64 69 6e 67 3d 22 55 54.. (Len: 1261
bytes)
Peach.Enterprise.Analyzers.ZipAnalyzer Successfully parsed: word/fontTable.xml
Peach.Enterprise.Analyzers.ZipAnalyzer Attempting to parse: word/webSettings.xml
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Block 'Content' Bytes: 0/497, Bits: 0/3976
Peach.Core.Cracker.DataCracker getSize: -----> Block 'Content'
Peach.Core.Cracker.DataCracker scan: Block 'Content'
Peach.Core.Cracker.DataCracker scan: Blob 'Content.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
```

```
Peach.Core.Cracker.DataCracker Crack: Block 'Content' Size: <null>, Bytes: 0/497,
Bits: 0/3976
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Blob 'Content.Data' Bytes: 0/497, Bits: 0/3976
Peach.Core.Cracker.DataCracker getSize: -----> Blob 'Content.Data'
Peach.Core.Cracker.DataCracker scan: Blob 'Content.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker lookahead: Blob 'Content.Data'
Peach.Core.Cracker.DataCracker getSize: <----- Last Unsized: 3976
Peach.Core.Cracker.DataCracker Crack: Blob 'Content.Data' Size: 3976, Bytes: 0/497,
Bits: 0/3976
Peach.Core.Dom.DataElement Blob 'Content.Data' value is: 3c 3f 78 6d 6c 20 76 65 72
73 69 6f 6e 3d 22 31 2e 30 22 20 65 6e 63 6f 64 69 6e 67 3d 22 55 54.. (Len: 497
bytes)
Peach.Enterprise.Analyzers.ZipAnalyzer Successfully parsed: word/webSettings.xml
Peach.Enterprise.Analyzers.ZipAnalyzer Attempting to parse: docProps/app.xml
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Block 'Content' Bytes: 0/711, Bits: 0/5688
Peach.Core.Cracker.DataCracker getSize: -----> Block 'Content'
Peach.Core.Cracker.DataCracker scan: Block 'Content'
Peach.Core.Cracker.DataCracker scan: Blob 'Content.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: Block 'Content' Size: <null>, Bytes: 0/711,
Bits: 0/5688
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Blob 'Content.Data' Bytes: 0/711, Bits: 0/5688
Peach.Core.Cracker.DataCracker getSize: -----> Blob 'Content.Data'
Peach.Core.Cracker.DataCracker scan: Blob 'Content.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker lookahead: Blob 'Content.Data'
Peach.Core.Cracker.DataCracker getSize: <----- Last Unsized: 5688
Peach.Core.Cracker.DataCracker Crack: Blob 'Content.Data' Size: 5688, Bytes: 0/711,
Bits: 0/5688
Peach.Core.Dom.DataElement Blob 'Content.Data' value is: 3c 3f 78 6d 6c 20 76 65 72
73 69 6f 6e 3d 22 31 2e 30 22 20 65 6e 63 6f 64 69 6e 67 3d 22 55 54.. (Len: 711
bytes)
Peach.Enterprise.Analyzers.ZipAnalyzer Successfully parsed: docProps/app.xml
Peach.Enterprise.Analyzers.ZipAnalyzer Attempting to parse: docProps/core.xml
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Block 'Content' Bytes: 0/747, Bits: 0/5976
Peach.Core.Cracker.DataCracker getSize: -----> Block 'Content'
Peach.Core.Cracker.DataCracker scan: Block 'Content'
Peach.Core.Cracker.DataCracker scan: Blob 'Content.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: Block 'Content' Size: <null>, Bytes: 0/747,
Bits: 0/5976
```

```
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Blob 'Content.Data' Bytes: 0/747, Bits: 0/5976
Peach.Core.Cracker.getSize: -----> Blob 'Content.Data'
Peach.Core.Cracker.scan: Blob 'Content.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker lookahead: Blob 'Content.Data'
Peach.Core.Cracker.getSize: <----- Last Unsized: 5976
Peach.Core.Cracker.Crack: Blob 'Content.Data' Size: 5976, Bytes: 0/747,
Bits: 0/5976
Peach.Core.Dom.DataElement Blob 'Content.Data' value is: 3c 3f 78 6d 6c 20 76 65 72
73 69 6f 6e 3d 22 31 2e 30 22 20 65 6e 63 6f 64 69 6e 67 3d 22 55 54.. (Len: 747
bytes)
Peach.Enterprise.Analyzers.ZipAnalyzer Successfully parsed: docProps/core.xml
Peach.Enterprise.Analyzers.ZipAnalyzer Attempting to parse: word/styles.xml
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Block 'Content' Bytes: 0/28676, Bits: 0/229408
Peach.Core.Cracker.getSize: -----> Block 'Content'
Peach.Core.Cracker.scan: Block 'Content'
Peach.Core.Cracker.scan: Blob 'Content.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.getSize: <----- Deterministic: ???
Peach.Core.Cracker.Crack: Block 'Content' Size: <null>, Bytes: 0/28676,
Bits: 0/229408
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Blob 'Content.Data' Bytes: 0/28676, Bits: 0/229408
Peach.Core.Cracker.getSize: -----> Blob 'Content.Data'
Peach.Core.Cracker.scan: Blob 'Content.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker lookahead: Blob 'Content.Data'
Peach.Core.Cracker.getSize: <----- Last Unsized: 229408
Peach.Core.Cracker.Crack: Blob 'Content.Data' Size: 229408, Bytes:
0/28676, Bits: 0/229408
Peach.Core.Dom.DataElement Blob 'Content.Data' value is: 3c 3f 78 6d 6c 20 76 65 72
73 69 6f 6e 3d 22 31 2e 30 22 20 65 6e 63 6f 64 69 6e 67 3d 22 55 54.. (Len: 28676
bytes)
Peach.Enterprise.Analyzers.ZipAnalyzer Successfully parsed: word/styles.xml
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Enterprise.Publishers.ZipPublisher start()
Peach.Enterprise.Publishers.ZipPublisher open()
Peach.Enterprise.Publishers.ZipPublisher Added 11 entries to zip file.
Peach.Enterprise.Publishers.ZipPublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Enterprise.Publishers.ZipPublisher stop()

[*] Test 'Default' finished.
```

View of example in Peach Validator

The screenshot shows the Peach Validator interface. The top pane displays a hex dump of the file's binary content, with bytes 00000000 through 000000D0 shown. The bottom pane shows a hierarchical tree of XML streams and their mappings:

Name	Position	Length	Value
WordDoc	0	11272	
Data	0	0	
Stream	0	0	[Content_Types].xml
Name	0	0	[Content_Types].xml
Attribute	0	0	0
Content	0	0	
Data	0	0	3c 3f 78 6d 6c 20 76 65 72 73 69 6f 6e 3d 22 31 2e 30 22
rels/_rels	0	0	_rels/.rels
word/_rels/document.xml.rels	0	0	word/_rels/document.xml.rels
word/document.xml	0	0	word/document.xml
word/theme/theme1.xml	0	0	word/theme/theme1.xml
word/settings.xml	0	0	word/settings.xml
word/fontTable.xml	0	0	word/fontTable.xml
word/webSettings.xml	0	0	word/webSettings.xml
docProps/app.xml	0	0	docProps/app.xml
docProps/core.xml	0	0	docProps/core.xml
word/styles.xml	0	0	word/styles.xml

Example 47. Analyzing a .docx file with mappings

This example uses the zip analyzer to decompose a .docx file into its underlying streams. This example also maps all .xml files to the data model *XmlModel* that further decomposes the data using the *Xml* analyzer.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="XmlModel">
        <String type="utf8" name="Xml">
            <Analyzer class="Xml"/>
        </String>
    </DataModel>

    <DataModel name="WordDoc">
        <Blob name="Data">
            <Analyzer class="Zip">
                <Param name="Map" value=".xml$/XmlModel/" />
            </Analyzer>
        </Blob>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="WordDoc"/>
                <Data fileName="example.docx"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="Zip">
            <Param name="FileName" value="fuzzed.docx" />
        </Publisher>
    </Test>
</Peach>
```

Output from the example that decomposes a .docx file into its underlying streams and maps all .xml files to the data model *XmlModel*.

```

[*] Test 'Default' starting with random seed 6071.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'WordDoc' Bytes: 0/11272, Bits: 0/90176
```

```
Peach.Core.Cracker.DataCracker getSize: -----> DataModel 'WordDoc'
Peach.Core.Cracker.scan: DataModel 'WordDoc'
Peach.Core.Cracker.scan: Blob 'WordDoc.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.getSize: <----- Deterministic: ???
Peach.Core.Cracker.Crack: DataModel 'WordDoc' Size: <null>, Bytes:
0/11272, Bits: 0/90176
Peach.Core.Cracker -----
Peach.Core.Cracker.Blob 'WordDoc.Data' Bytes: 0/11272, Bits: 0/90176
Peach.Core.Cracker.getSize: -----> Blob 'WordDoc.Data'
Peach.Core.Cracker.scan: Blob 'WordDoc.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.lookahead: Blob 'WordDoc.Data'
Peach.Core.Cracker.getSize: <----- Last Unsized: 90176
Peach.Core.Cracker.Crack: Blob 'WordDoc.Data' Size: 90176, Bytes:
0/11272, Bits: 0/90176
Peach.Core.Dom.DataElement Blob 'WordDoc.Data' value is: 50 4b 03 04 14 00 06 00 08
00 00 00 21 00 df a4 d2 6c 5a 01 00 00 20 05 00 00 13 00 08 02 5b 43.. (Len: 11272
bytes)
Peach.Enterprise.Analyzers.ZipAnalyzer Attempting to parse: [Content_Types].xml
Peach.Enterprise.Analyzers.ZipAnalyzer Resolved entry '[Content_Types].xml' to data
model 'XmlModel'.
Peach.Core.Cracker -----
Peach.Core.Cracker.DataModel 'Content' Bytes: 0/1312, Bits: 0/10496
Peach.Core.Cracker.getSize: -----> DataModel 'Content'
Peach.Core.Cracker.scan: DataModel 'Content'
Peach.Core.Cracker.scan: String 'Content.Xml' -> Offset: 0, Unsized
element
Peach.Core.Cracker.getSize: <----- Deterministic: ???
Peach.Core.Cracker.Crack: DataModel 'Content' Size: <null>, Bytes:
0/1312, Bits: 0/10496
Peach.Core.Cracker -----
Peach.Core.Cracker.String 'Content.Xml' Bytes: 0/1312, Bits: 0/10496
Peach.Core.Cracker.getSize: -----> String 'Content.Xml'
Peach.Core.Cracker.scan: String 'Content.Xml' -> Offset: 0, Unsized
element
Peach.Core.Cracker.lookahead: String 'Content.Xml'
Peach.Core.Cracker.getSize: <----- Last Unsized: 10496
Peach.Core.Cracker.Crack: String 'Content.Xml' Size: 10496, Bytes:
0/1312, Bits: 0/10496
Peach.Core.Dom.DataElement String 'Content.Xml' value is: <?xml version="1.0"
encoding="UTF-8" standalone="yes"?>
<Types .. (Len: 1312 chars)
Peach.Enterprise.Analyzers.ZipAnalyzer Successfully parsed: [Content_Types].xml
Peach.Enterprise.Analyzers.ZipAnalyzer Attempting to parse: _rels/.rels
Peach.Core.Cracker -----
Peach.Core.Cracker.Block 'Content' Bytes: 0/590, Bits: 0/4720
Peach.Core.Cracker.getSize: -----> Block 'Content'
```

```
Peach.Core.Cracker.DataCracker scan: Block 'Content'
Peach.Core.Cracker.DataCracker scan: Blob 'Content.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: Block 'Content' Size: <null>, Bytes: 0/590,
Bits: 0/4720
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Blob 'Content.Data' Bytes: 0/590, Bits: 0/4720
Peach.Core.Cracker.DataCracker getSize: -----> Blob 'Content.Data'
Peach.Core.Cracker.DataCracker scan: Blob 'Content.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker lookahead: Blob 'Content.Data'
Peach.Core.Cracker.DataCracker getSize: <----- Last Unsized: 4720
Peach.Core.Cracker.DataCracker Crack: Blob 'Content.Data' Size: 4720, Bytes: 0/590,
Bits: 0/4720
Peach.Core.Dom.DataElement Blob 'Content.Data' value is: 3c 3f 78 6d 6c 20 76 65 72
73 69 6f 6e 3d 22 31 2e 30 22 20 65 6e 63 6f 64 69 6e 67 3d 22 55 54.. (Len: 590
bytes)
Peach.Enterprise.Analyzers.ZipAnalyzer Successfully parsed: _rels/.rels
Peach.Enterprise.Analyzers.ZipAnalyzer Attempting to parse:
word/_rels/document.xml.rels
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Block 'Content' Bytes: 0/817, Bits: 0/6536
Peach.Core.Cracker.DataCracker getSize: -----> Block 'Content'
Peach.Core.Cracker.DataCracker scan: Block 'Content'
Peach.Core.Cracker.DataCracker scan: Blob 'Content.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: Block 'Content' Size: <null>, Bytes: 0/817,
Bits: 0/6536
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Blob 'Content.Data' Bytes: 0/817, Bits: 0/6536
Peach.Core.Cracker.DataCracker getSize: -----> Blob 'Content.Data'
Peach.Core.Cracker.DataCracker scan: Blob 'Content.Data' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker lookahead: Blob 'Content.Data'
Peach.Core.Cracker.DataCracker getSize: <----- Last Unsized: 6536
Peach.Core.Cracker.DataCracker Crack: Blob 'Content.Data' Size: 6536, Bytes: 0/817,
Bits: 0/6536
Peach.Core.Dom.DataElement Blob 'Content.Data' value is: 3c 3f 78 6d 6c 20 76 65 72
73 69 6f 6e 3d 22 31 2e 30 22 20 65 6e 63 6f 64 69 6e 67 3d 22 55 54.. (Len: 817
bytes)
Peach.Enterprise.Analyzers.ZipAnalyzer Successfully parsed:
word/_rels/document.xml.rels
Peach.Enterprise.Analyzers.ZipAnalyzer Attempting to parse: word/document.xml
Peach.Enterprise.Analyzers.ZipAnalyzer Resolved entry 'word/document.xml' to data
model 'XmlModel'.
Peach.Core.Cracker.DataCracker -----
```

```
Peach.Core.Cracker.DataCracker DataModel 'Content' Bytes: 0/1620, Bits: 0/12960
Peach.Core.Cracker.getSize: -----> DataModel 'Content'
Peach.Core.Cracker.scan: DataModel 'Content'
Peach.Core.Cracker.scan: String 'Content.Xml' -> Offset: 0, Unsized
element
Peach.Core.Cracker.getSize: <----- Deterministic: ???
Peach.Core.Cracker.Crack: DataModel 'Content' Size: <null>, Bytes:
0/1620, Bits: 0/12960
Peach.Core.Cracker -----
Peach.Core.Cracker.String 'Content.Xml' Bytes: 0/1620, Bits: 0/12960
Peach.Core.Cracker.getSize: -----> String 'Content.Xml'
Peach.Core.Cracker.scan: String 'Content.Xml' -> Offset: 0, Unsized
element
Peach.Core.Cracker.lookahead: String 'Content.Xml'
Peach.Core.Cracker.getSize: <----- Last Unsized: 12960
Peach.Core.Cracker.Crack: String 'Content.Xml' Size: 12960, Bytes:
0/1620, Bits: 0/12960
Peach.Core.Dom.DataElement String 'Content.Xml' value is: <?xml version="1.0"
encoding="UTF-8" standalone="yes"?>
<w:docu.. (Len: 1620 chars)
Peach.Enterprise.Analyzers.ZipAnalyzer Successfully parsed: word/document.xml
Peach.Enterprise.Analyzers.ZipAnalyzer Attempting to parse: word/theme/theme1.xml
Peach.Enterprise.Analyzers.ZipAnalyzer Resolved entry 'word/theme/theme1.xml' to data
model 'XmlModel'.
Peach.Core.Cracker -----
Peach.Core.Cracker.DataModel 'Content' Bytes: 0/6795, Bits: 0/54360
Peach.Core.Cracker.getSize: -----> DataModel 'Content'
Peach.Core.Cracker.scan: DataModel 'Content'
Peach.Core.Cracker.scan: String 'Content.Xml' -> Offset: 0, Unsized
element
Peach.Core.Cracker.getSize: <----- Deterministic: ???
Peach.Core.Cracker.Crack: DataModel 'Content' Size: <null>, Bytes:
0/6795, Bits: 0/54360
Peach.Core.Cracker -----
Peach.Core.Cracker.String 'Content.Xml' Bytes: 0/6795, Bits: 0/54360
Peach.Core.Cracker.getSize: -----> String 'Content.Xml'
Peach.Core.Cracker.scan: String 'Content.Xml' -> Offset: 0, Unsized
element
Peach.Core.Cracker.lookahead: String 'Content.Xml'
Peach.Core.Cracker.getSize: <----- Last Unsized: 54360
Peach.Core.Cracker.Crack: String 'Content.Xml' Size: 54360, Bytes:
0/6795, Bits: 0/54360
Peach.Core.Dom.DataElement String 'Content.Xml' value is: <?xml version="1.0"
encoding="UTF-8" standalone="yes"?>
<a:them.. (Len: 6735 chars)
Peach.Enterprise.Analyzers.ZipAnalyzer Successfully parsed: word/theme/theme1.xml
Peach.Enterprise.Analyzers.ZipAnalyzer Attempting to parse: word/settings.xml
Peach.Enterprise.Analyzers.ZipAnalyzer Resolved entry 'word/settings.xml' to data
```

```
model 'XmlModel'.
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'Content' Bytes: 0/2477, Bits: 0/19816
Peach.Core.Cracker getSize: -----> DataModel 'Content'
Peach.Core.Cracker.DataCracker scan: DataModel 'Content'
Peach.Core.Cracker.DataCracker scan: String 'Content.Xml' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: DataModel 'Content' Size: <null>, Bytes:
0/2477, Bits: 0/19816
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker String 'Content.Xml' Bytes: 0/2477, Bits: 0/19816
Peach.Core.Cracker.getSize: -----> String 'Content.Xml'
Peach.Core.Cracker.DataCracker scan: String 'Content.Xml' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker lookahead: String 'Content.Xml'
Peach.Core.Cracker.DataCracker getSize: <----- Last Unsized: 19816
Peach.Core.Cracker.DataCracker Crack: String 'Content.Xml' Size: 19816, Bytes:
0/2477, Bits: 0/19816
Peach.Core.Dom.DataElement String 'Content.Xml' value is: <?xml version="1.0"
encoding="UTF-8" standalone="yes"?>
<w:sett.. (Len: 2477 chars)
Peach.Enterprise.Analyzers.ZipAnalyzer Successfully parsed: word/settings.xml
Peach.Enterprise.Analyzers.ZipAnalyzer Attempting to parse: word/fontTable.xml
Peach.Enterprise.Analyzers.ZipAnalyzer Resolved entry 'word/fontTable.xml' to data
model 'XmlModel'.
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'Content' Bytes: 0/1261, Bits: 0/10088
Peach.Core.Cracker.getSize: -----> DataModel 'Content'
Peach.Core.Cracker.DataCracker scan: DataModel 'Content'
Peach.Core.Cracker.DataCracker scan: String 'Content.Xml' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: DataModel 'Content' Size: <null>, Bytes:
0/1261, Bits: 0/10088
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker String 'Content.Xml' Bytes: 0/1261, Bits: 0/10088
Peach.Core.Cracker.DataCracker getSize: -----> String 'Content.Xml'
Peach.Core.Cracker.DataCracker scan: String 'Content.Xml' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker lookahead: String 'Content.Xml'
Peach.Core.Cracker.DataCracker getSize: <----- Last Unsized: 10088
Peach.Core.Cracker.DataCracker Crack: String 'Content.Xml' Size: 10088, Bytes:
0/1261, Bits: 0/10088
Peach.Core.Dom.DataElement String 'Content.Xml' value is: <?xml version="1.0"
encoding="UTF-8" standalone="yes"?>
<w:font.. (Len: 1261 chars)
Peach.Enterprise.Analyzers.ZipAnalyzer Successfully parsed: word/fontTable.xml
```

```
Peach.Enterprise.Analyzers.ZipAnalyzer Attempting to parse: word/webSettings.xml
Peach.Enterprise.Analyzers.ZipAnalyzer Resolved entry 'word/webSettings.xml' to data
model 'XmlModel'.
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'Content' Bytes: 0/497, Bits: 0/3976
Peach.Core.Cracker.getSize: -----> DataModel 'Content'
Peach.Core.Cracker.DataCracker scan: DataModel 'Content'
Peach.Core.Cracker.DataCracker scan: String 'Content.Xml' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: DataModel 'Content' Size: <null>, Bytes: 0/497,
Bits: 0/3976
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker String 'Content.Xml' Bytes: 0/497, Bits: 0/3976
Peach.Core.Cracker.getSize: -----> String 'Content.Xml'
Peach.Core.Cracker.DataCracker scan: String 'Content.Xml' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker lookahead: String 'Content.Xml'
Peach.Core.Cracker.DataCracker getSize: <----- Last Unsized: 3976
Peach.Core.Cracker.DataCracker Crack: String 'Content.Xml' Size: 3976, Bytes: 0/497,
Bits: 0/3976
Peach.Core.Dom.DataElement String 'Content.Xml' value is: <?xml version="1.0"
encoding="UTF-8" standalone="yes"?>
<w:webS.. (Len: 497 chars)
Peach.Enterprise.Analyzers.ZipAnalyzer Successfully parsed: word/webSettings.xml
Peach.Enterprise.Analyzers.ZipAnalyzer Attempting to parse: docProps/app.xml
Peach.Enterprise.Analyzers.ZipAnalyzer Resolved entry 'docProps/app.xml' to data
model 'XmlModel'.
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'Content' Bytes: 0/711, Bits: 0/5688
Peach.Core.Cracker.DataCracker getSize: -----> DataModel 'Content'
Peach.Core.Cracker.DataCracker scan: DataModel 'Content'
Peach.Core.Cracker.DataCracker scan: String 'Content.Xml' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: DataModel 'Content' Size: <null>, Bytes: 0/711,
Bits: 0/5688
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker String 'Content.Xml' Bytes: 0/711, Bits: 0/5688
Peach.Core.Cracker.DataCracker getSize: -----> String 'Content.Xml'
Peach.Core.Cracker.DataCracker scan: String 'Content.Xml' -> Offset: 0, Unsized
element
Peach.Core.Cracker.DataCracker lookahead: String 'Content.Xml'
Peach.Core.Cracker.DataCracker getSize: <----- Last Unsized: 5688
Peach.Core.Cracker.DataCracker Crack: String 'Content.Xml' Size: 5688, Bytes: 0/711,
Bits: 0/5688
Peach.Core.Dom.DataElement String 'Content.Xml' value is: <?xml version="1.0"
encoding="UTF-8" standalone="yes"?>
```

```
<Proper.. (Len: 711 chars)
Peach.Enterprise.Analyzers.ZipAnalyzer Successfully parsed: docProps/app.xml
Peach.Enterprise.Analyzers.ZipAnalyzer Attempting to parse: docProps/core.xml
Peach.Enterprise.Analyzers.ZipAnalyzer Resolved entry 'docProps/core.xml' to data
model 'XmlModel'.
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'Content' Bytes: 0/747, Bits: 0/5976
Peach.Core.Cracker.getSize: -----> DataModel 'Content'
Peach.Core.Cracker.scan: DataModel 'Content'
Peach.Core.Cracker.scan: String 'Content.Xml' -> Offset: 0, Unsized
element
Peach.Core.Cracker.getSize: <----- Deterministic: ???
Peach.Core.Cracker.Crack: DataModel 'Content' Size: <null>, Bytes: 0/747,
Bits: 0/5976
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.String 'Content.Xml' Bytes: 0/747, Bits: 0/5976
Peach.Core.Cracker.getSize: -----> String 'Content.Xml'
Peach.Core.Cracker.scan: String 'Content.Xml' -> Offset: 0, Unsized
element
Peach.Core.Cracker.lookahead: String 'Content.Xml'
Peach.Core.Cracker.getSize: <----- Last Unsized: 5976
Peach.Core.Cracker.Crack: String 'Content.Xml' Size: 5976, Bytes: 0/747,
Bits: 0/5976
Peach.Core.Dom.DataElement String 'Content.Xml' value is: <?xml version="1.0"
encoding="UTF-8" standalone="yes"?>
<cp:cor.. (Len: 747 chars)
Peach.Enterprise.Analyzers.ZipAnalyzer Successfully parsed: docProps/core.xml
Peach.Enterprise.Analyzers.ZipAnalyzer Attempting to parse: word/styles.xml
Peach.Enterprise.Analyzers.ZipAnalyzer Resolved entry 'word/styles.xml' to data model
'XmlModel'.
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'Content' Bytes: 0/28676, Bits: 0/229408
Peach.Core.Cracker.getSize: -----> DataModel 'Content'
Peach.Core.Cracker.scan: DataModel 'Content'
Peach.Core.Cracker.scan: String 'Content.Xml' -> Offset: 0, Unsized
element
Peach.Core.Cracker.getSize: <----- Deterministic: ???
Peach.Core.Cracker.Crack: DataModel 'Content' Size: <null>, Bytes:
0/28676, Bits: 0/229408
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.String 'Content.Xml' Bytes: 0/28676, Bits: 0/229408
Peach.Core.Cracker.getSize: -----> String 'Content.Xml'
Peach.Core.Cracker.scan: String 'Content.Xml' -> Offset: 0, Unsized
element
Peach.Core.Cracker.lookahead: String 'Content.Xml'
Peach.Core.Cracker.getSize: <----- Last Unsized: 229408
Peach.Core.Cracker.Crack: String 'Content.Xml' Size: 229408, Bytes:
0/28676, Bits: 0/229408
```

```
Peach.Core.Dom.DataElement String 'Content.Xml' value is: <?xml version="1.0"
encoding="UTF-8" standalone="yes"?>
<w:styl.. (Len: 28676 chars)
Peach.Enterprise.Analyzers.ZipAnalyzer Successfully parsed: word/styles.xml
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Enterprise.Publishers.ZipPublisher start()
Peach.Enterprise.Publishers.ZipPublisher open()
Peach.Enterprise.Publishers.ZipPublisher Added 11 entries to zip file.
Peach.Enterprise.Publishers.ZipPublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Enterprise.Publishers.ZipPublisher stop()

[*] Test 'Default' finished.
```

View of example in Peach Validator

Peach Validator v3.0 - example.xml - example.docx

Pit: WordDoc | Sample file: WordDoc

00000000	50 4B 03 04 14 00 06 00 08 00 00 00 21 00 DF A4	PK.....!..Bx
00000010	D2 6C 5A 01 00 00 20 05 00 00 13 00 08 02 5B 43	Ö1z...[C
00000020	6F 6E 74 65 6E 74 5F 54 79 70 65 73 5D 2E 78 6D	ontent_Types].xm
00000030	6C 20 A2 04 02 28 A0 00 02 00 00 00 00 00 00 00	l e...(.
00000040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Name	Position	Length	Value
WordDoc	0	11272	
Data	0	0	
Stream	0	0	[Content_Types].xml
Name	0	0	[Content_Types].xml
Attribute	0	0	0
Content	0	0	
Xml	0	0	<Types> Element
xmlns	0	0	'xmlns' Attribute
Default	0	0	<Default> Element
Default_1	0	0	<Default> Element
Override	0	0	<Override> Element
Override_1	0	0	<Override> Element
Override_2	0	0	<Override> Element
Override_3	0	0	<Override> Element
Override_4	0	0	<Override> Element
Override_5	0	0	<Override> Element
Override_6	0	0	<Override> Element
Override_7	0	0	<Override> Element
Stream_1	0	0	_rels.rels
Stream_2	0	0	word/_rels/document.xml.rels
Stream_3	0	0	word/document.xml
Stream_4	0	0	word/theme/theme1.xml
Stream_5	0	0	word/settings.xml
Stream_6	0	0	word/fontTable.xml
Stream_7	0	0	word/webSettings.xml
Stream_8	0	0	docProps/app.xml
Stream_9	0	0	docProps/core.xml
Stream_10	0	0	word/styles.xml

21.3. Data Elements

The following items comprise the data elements in Peach.

Blob

Blob elements are containers for unstructured data (think byte arrays). A *Blob* is a term used in the relational database field for a column that holds raw binary data. Data contained in a *Blob* will be dumb fuzzed.

Block

The *Block* element is a container for other data elements. By itself, a *Block* does not have any size or contain other data. It simply groups other data elements and, by using the Block name, provides a mechanism to reference or operate on the group as a single unit.

For example, if two data elements are included in a checksum calculation, they can be placed in a *Block* element and then the fixup can perform its function by referencing the *Block*. Both items in the group will be included in the fixup operation.

Choice

The *Choice* element allows constructing switch-like statements in Peach data models. The *Choice* element is useful when modeling type-length-value (TLV) metaphors and in other situations where the data format changes.

DataModel

The *DataModel* element is a top-level element that defines a data model. Multiple *DataModel* elements can exist in a pit file. A *DataModel* element by itself has no size and is a container for other data elements.

Double

Double defines a floating-point number of 32 or 64 bits conforming to the IEEE 754 standard. Floating point values are packed into a byte representation with a byte order of little or big endian.

Double elements are always packed to bit/byte representation with byte order (endian-ness). For ASCII strings, use the *String* element instead of a *Double*.

Flags

Flags defines a set of flags and is a container for *Flag* elements. The *Flags* container can provide a nice shortcut when dealing with a flag set that has many unused positions.

Tip: The *Number* element supports unaligned sizes and can be used to define flags.

Flag

Flag defines a specific *Flag* in a flag set. *Flag* elements have a bit position and bit length.

Frag

Used to model protocol fragmentation

JsonArray

JsonArray is used to represent a JSON array. It's similar in operation to [Sequence](#).

JsonBool

The *JsonBool* element defines a Boolean value (values 0 or 1) that represents either true or false.

JsonBlob

A *JsonBlob* [1: Blob stands for "binary large object" a term used by databases to represent a column of binary data.] is used to represent binary JSON data (array of bytes). *JsonBlobs* are base64 encoded.

By definition, the internal contents of a *JsonBlob* is unknown. Consequently, Blobs are dumb fuzzed, as the list of applicable mutators below indicates.

JsonDouble

The *JsonDouble* element defines a JSON floating-point number of up to 64-bits.

JsonInteger

The *JsonInteger* element defines a binary number of any arbitrary bit size from 1 to 64. Binary numbers are packed into a byte representation with a byte order of little-endian or big-endian.

The *Number* element should not be used for character-based numbers, or ASCII numbers. Instead, use a [String](#) element.

JsonObject

The *JsonObject* element is used to model [JavaScript Object Notation \(JSON\)](#) objects.

JsonObject elements are containers that group JSON elements in a logical structure. For example, a *JsonString* element, a *JsonInteger* element, and a *JsonBlob* element can exist in the same *JsonObject*. A *JsonObject* can contain other *JsonObjects*, or nest, as deeply as needed.

JsonRaw

The *JsonRaw* element is used to embed raw string or binary data into a JSON document. This allows creation of json that does not adhere to the JSON specification.

This is a container element that is able to host other data elements.

JsonString

Represents a JSON string of Unicode characters.

All strings are encoded prior to output according to the JSON specification.

Strings can hold numbers that are stored in a string format. When strings contain a number, Peach additionally uses numerical mutators to mutate the strings.

Number

Number defines a binary integer type. A *Number* is always packed to bit/byte representation with byte order (endian-ness). For ASCII strings, use the *String* element instead of a *Number*.

Padding

Padding defines padding of one or more characters for a *Block* or a *DataModel*. The attributes of the *Padding* element specify things such as a reference to start the alignment calculation, a size, and the *Padding* element name. The most common use of *Padding* is to ensure a consistent length of a *Block* or *DataModel* that contains variable length data elements, such as padding a *DataModel* for a 64-byte packet.

Sequence

Sequence elements are used to construct arrays consisting of predefined data elements. A *Sequence* is able to contain multiple data types at once. Unlike regular arrays in Peach, the size of the Sequence is determined by the number of elements within the sequence.

Stream

Stream defines a *Stream* element with a name, attributes and content. *Streams* group one or more data elements into a logical structure.

String

A *string* defines a contiguous sequence of characters with encoding. Encodings include ASCII and many Unicode formats.

VarNumber

Allows modeling variable width *Numbers*. Size must be constrained using a size-of relationship or other method.

Xmlelement

The *Xmlelement* defines an XML attribute. This element is only valid as a child of [XmlElement](#).

Xmlelement

The *Xmlelement* defines an XML element. This element is only valid as a child of [XmlElement](#).

Xmlelement

Xmlelement defines an XML element.

21.3.1. Asn1Type

Asn1Type is used to model Abstract Syntax Notation One (ASN.1) structures.

The *Asn1Type* specifies the information of ASN.1 fields. ASN.1 data elements can be primitive (pre-defined) or constructed. ASN.1 structures can be very deep. It's recommended to use the ASN.1 analyzer to generate data models that can then be modified.



This element exists for manually coding ASN.1 data elements. An alternative is to use the ASN.1 analyzer that can generate an ASN.1 data model, and produces instances of this element in the resulting data model.

Syntax

```
<Asn1Type class="0" pc="1" tag="16" name="TerminationID">
  <Block name="Value">
    <Asn1Type class="2" pc="1" tag="0" name="wildcard">
      <Block name="Value" />
    </Asn1Type>
    <Asn1Type class="2" pc="0" tag="1" name="id">
      <Blob name="terminationIDValue" valueType="hex" value="00000000300015c1" />
    </Asn1Type>
  </Block>
</Asn1Type>
```

Attributes

Required:

tag

ASN.1 tag identifier (numeric)

Optional:

class

ASN.1 Class value, default is "0".

0

Universal class.

1

Application class.

2

Context-specific class.

Private class.

constraint

Scripting expression that evaluates to true or false, default is "".

forceMultiByteIdentifier

If "true", force long encoding of ASN.1 tags. The default is "false".

indefiniteLength

If "true", use the indefinite encoding form of the ASN.1 element. If "false", use definite encoding form. The default is "false".

longLength

If "true", always use the long encoding form (that is, multiple length octets) to express the length of the ASN.1 element. If "false", also allow the short encoding form where the number of content octets is expressed in 7 bits (127 or less). The default is "false".

minOccurs

The minimum number of times this blob must occur. Defaults to 1.

- Used to define arrays with variable size. Arrays defined by min/maxOccurs generally have a relation defined.
- Adding this attribute, even with a value of 1, converts the element to an array.

maxOccurs

The maximum number of times this blob can occur. Defaults to 1.

- Used to define arrays with static size.
- Adding this attribute, even with a value of 1, converts the element to an array.

mutable

Is this element mutable, default is "true".

name

Element name, default is "true".

pc

Integer that specifies whether the ASN.1 element is a primitive or a constructed element. The default is "0" which indicates a primitive element.

token

Is this element a token for parsing, default is "false".

Child Elements

Asn1Type

Asn1Type of a child ASN.1 data element.

Blob

Represents binary data (array of bytes) to create simple dumb fuzzers in Peach.

Block

Groups one or more data elements in a logical structure.

Choice

Indicates that all of the sub-elements are valid; but, only one sub-element should be selected.

Flags

Defines a set of bit-sized flags.

Number

Defines a binary number of arbitrary bit size.

Padding

Pads variably sized blocks or data models provide size uniformity or consistency.

Placement

Relocates an element after it has been cracked.

Stream

Groups one or more data elements in a logical structure.

String

Character sequence consisting of ASCII or Unicode characters.

XmleAttribute

Performs static transformations such as compression or encoding.

XmleElement

Defines an XML element, the basic building block of XML documents.

Fixup

Performs dynamic transformations such as checksums and CRCs.

Hint

Provides information to mutators.

Transformer

Performs static transformations such as compression or encoding.

Relation

Identifies a type of relationship with another data element, such as count.

Analyzer

Analyzes current element post cracking, can dynamically change model.

Mutators

This data element is a container element with several hidden internal elements of Number type, plus any value for the ASN.1 field. This data element will get fuzzed with all mutators from Block and Number.

Examples

Example 48. Example of ASN.1 Model

Example of modeling an ASN.1 specification fragment.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach>

    <DataModel name="TheDataModel">
        <Asn1Type class="2" pc="1" tag="0" name="terminationId">
            <Asn1Type class="0" pc="1" tag="16" name="TerminationID">
                <Block name="Value">
                    <Asn1Type class="2" pc="1" tag="0" name="wildcard">
                        <Block name="Value" />
                    </Asn1Type>
                    <Asn1Type class="2" pc="0" tag="1" name="id">
                        <Blob name="terminationIDValue" valueType="hex" value=
                            "00000000300015c1" />
                    </Asn1Type>
                </Block>
            </Asn1Type>
        </Asn1Type>
    </DataModel>

</Peach>
```

21.3.2. Blob

A Blob [2: Blob stands for "binary large object" a term used by databases to represent a column of binary data.] is used to represent binary data (array of bytes). Blobs are sized in bytes.

A Blob is always a child of a data element container such as [DataModel](#) or [Block](#). By definition, the internal contents of a Blob is unknown. Consequently, Blobs are dumb fuzzed, as the list of applicable mutators below indicates.

Blobs can be used to create simple dumb fuzzers in Peach, see [examples](#).

Syntax

```
<Blob valueType="hex" value="01 06 22 03" />
```

Attributes

Required:

There are no required attributes.

Optional:

[name](#)

Element name

[length](#)

The length of data in this element.

[lengthType](#)

Units of the length attribute. Defaults to "bytes". Can also be "chars", or "bits" where the number of bits is a multiple of 8.

[value](#)

Default value.

[valueType](#)

Format of value attribute.

[occurs](#)

Actual occurrences of element. Defaults to 1.

- Used to define arrays with static size.
- Adding this attribute, even with a value of 1, converts the element to an array.

minOccurs

The minimum number of times this blob must occur. Defaults to 1.

- Used to define arrays with variable size. Arrays defined by min/maxOccurs generally have a relation defined.
- Adding this attribute, even with a value of 1, converts the element to an array.

maxOccurs

The maximum number of times this blob can occur. Defaults to 1.

- Used to define arrays with static size.
- Adding this attribute, even with a value of 1, converts the element to an array.

constraint

A constraint in the form of an expression. Used during data cracking.

mutable

Is the blob changeable (should it be fuzzed). Defaults to true.

token

This element should be treated as a token when parsing. Defaults to false.

Child Elements

Analyzer

Analyzes current element post cracking, can dynamically change model.

Fixup

Performs dynamic transformations such as checksums and CRCs.

Hint

Provides information to mutators.

Placement

Relocates an element after it has been cracked.

Transformer

Performs static transformations such as compression or encoding.

Mutators

The following mutators operate on this element type.

Enabled when element is marked as an array

ArrayNumericalEdgeCasesMutator

This mutator grows and shrinks an array to counts based on numerical edge cases.

ArrayRandomizeOrderMutator

This mutator randomizes the order of items in an array.

ArrayReverseOrderMutator

This mutator reverses the order of items in an array.

ArrayVarianceMutator

This mutator grows and shrinks an array to a variance of counts based on the current size.

Used for all data elements

DataElementBitFlipper

This mutator produces test cases by flipping bits in the output value.

DataElementDuplicate

This mutator duplicates data elements.

DataElementRemove

This mutator removes data elements.

DataElementSwapNear

This mutator swaps data elements.

SampleNinjaMutator

This mutator combines data elements from different data sets.

Enabled when element is part of a size relation

SizedDataEdgeCase

This mutator causes the data portion of a relation to be sized as numerical edge cases.

SizedDataVariance

This mutator causes the data portion of a relation to be sized as numerical variances.

SizedEdgeCase

This mutator changes both sides of the relation (data and value) to match numerical edge cases.

SizedVariance

This mutator changes both sides of the relation (data and value) to match numerical variances of the current size.

Specific to this element type

BlobChangeFromNull

This mutator produces test cases in which null bytes in a `Blob` element are changed to a non-null value.

BlobChangeRandom

This mutator produces test cases by changing random selections of bytes to random value.

BlobChangeSpecial

This mutator produces test cases by changing random selections of bytes to one of 0x00, 0x01, 0xFE, 0xFF.

BlobChangeToNull

This mutator produces test cases by changing a random number of bytes to 0x00.

BlobExpandSingleIncrementing

This mutator produces test cases by expanding the size of the blob using incrementing values.

BlobExpandAllRandom

This mutator produces test cases by expanding the size of the blob using random values.

BlobExpandSingleRandom

This mutator produces test cases by expanding the size of the blob using a single random byte (repeated as needed).

BlobExpandZero

This mutator produces test cases by expanding the blob using null values.

BlobReduce

This mutator produces test cases by reducing the size of the blob by a random amount.

ExtraValues

This mutator allows providing extra test case values on a per-data element basis.

Examples

Example 49. Dumb file fuzzing

This is an example of dumb file fuzzing. Peach cracks the data from `sample.png` into a `Blob`. This is considered dumb fuzzing because the data model does not fully describe the structure of the data being fuzzed.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach>

    <DataModel name="TheDataModel">
```

```

<Blob />
</DataModel>

<!-- Define a simple state machine that will write the file and
    then launch a program using the FileWriter and DebuggerLaucher publishers -->
<StateModel name="State" initialState="Initial">
    <State name="Initial">

        <!-- Write out contents of file. -->
        <Action type="output">
            <DataModel ref="TestTemplate" />
            <Data fileName="sample.png" />
        </Action>

        <!-- Close file -->
        <Action type="close" />

        <!-- Launch the file consumer -->
        <Action type="call" method="ScoobySnacks" publisher="Peach.Agent"/>

    </State>
</StateModel>

<!-- Setup a local agent that will monitor for faults -->
<Agent name="LocalAgent">
    <Monitor class="WindowsDebugger">

        <!-- The command line to run. Notice the filename provided matched up
            to what is provided below in the Publisher configuration -->
        <Param name="Executable" value="c:\windows\system32\mspaint.exe" />
        <Param name="Arguments" value="fuzzfile.bin" />

        <!-- This parameter will cause the debugger to wait for an action-call in
            the state model with a method="ScoobySnacks" before running
            program.

            Note: You will also need to add a parameter to the publisher called
                  "debugger" and set it to "true"!
            -->
        <Param name="StartOnCall" value="ScoobySnacks" />

    </Monitor>
</Agent>

<Test name="Default">
    <Agent ref="LocalAgent" />
    <StateModel ref="State" />

```

```

<!-- Configure our publisher with correct filename to write too -->
<Publisher class="File">
    <Param name="FileName" value="fuzzfile.bin" />
</Publisher>

<!-- Configure a logger to store collected information --> </Test>
</Peach>

```

Example 50. Defining a Blob with a default value

A blob with a default value. Providing a default value does not set a fixed length unless the token="true" attribute is used.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

<DataModel name="Ex1">
    <Blob name="Unknown1" valueType="hex" value="AA BB CC DD" />
</DataModel>

<StateModel name="TheState" initialState="initial">
    <State name="initial">
        <Action type="output" publisher="ConsolePub">
            <DataModel ref="Ex1" />
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex" name="ConsolePub"/> </Test>
</Peach>

```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 46616.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(4 bytes)
00000000  AA BB CC DD                      *****
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

Using a Blob as part of a size relationship

A blob with size-of relationship:

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="Ex1">
        <Number name="LengthOfData" size="32">
            <Relation type="size" of="Data" />
        </Number>

        <Blob name="Data" valueType="hex" value="AA BB CC DD" />
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output" publisher="ConsolePub">
                <DataModel ref="Ex1" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex" name="ConsolePub"/> </Test>
    </Peach>

```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 18508.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(8 bytes)
00000000  04 00 00 00 AA BB CC DD          ????????
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

21.3.3. Block

Blocks are container elements that group data elements in a logical structure. For example, a *string* element, a *number* element, and a *blob* element can exist in the same *block*. *Blocks* can contain other blocks, or nest, as deeply as needed.

Blocks and [DataModels](#) are very similar; both can be used as the template for other Blocks or [DataModels](#). The only difference is the positioning of these elements.

- A *DataModel* is a top level element
- A *Block* is a child element of [DataModel](#)

Syntax

```
<Block name="HelloWorld">
  <String value="Hello world!" />
</Block>
```

Attributes

Required:

None.

Optional:

name

Name of the block.

ref

Reference to a [DataModel](#) to use as a template.

length

Data element length.

lengthType

The unit of measure attribute for the Length attribute. Default is bytes.

constraint

Scripting expression that evaluates to true or false. Default is null.

minOccurs

The minimum number of times this element must occur. Defaults to 1.

- Used to define arrays with variable size. Arrays defined by min/maxOccurs generally have a

relation defined.

- Adding this attribute, even with a value of 1, converts the element to an array.

maxOccurs

The maximum number of times this element can occur. Defaults to 1.

- Used to define arrays with static size.
- Adding this attribute, even with a value of 1, converts the element to an array.

occurs

Actual occurrences of element. Defaults to 1.

- Used to define arrays with static size.
- Adding this attribute, even with a value of 1, converts the element to an array.

mutable

Is data element changeable (should it be mutated), defaults to false.

Child Elements

Analyzer

Analyzes current element post cracking, can dynamically change model.

Blob

Represents binary data (array of bytes) to create simple dumb fuzzers in Peach.

Block

Groups one or more data elements in a logical structure.

Choice

Indicates that all of the sub-elements are valid; but, only one sub-element should be selected.

Fixup

Are dynamic transformations such as checksums and CRCs.

Flags

Defines a set of bit-sized flags.

Hint

Provides information to mutators.

Number

Defines a binary number of arbitrary bit size.

Padding

Pads variably sized blocks or data models provide size uniformity or consistency.

Placement

Relocates an element after it has been cracked.

Stream

Groups one or more data elements in a logical structure.

String

Character sequence consisting of ASCII or Unicode characters.

Transformer

Performs static transformations such as compression or encoding.

XmlElement

Defines an XML element, the basic building block of XML documents.

Mutators

The following mutators operate on this element type:

Enabled when element is marked as an array

ArrayNumericalEdgeCasesMutator

This mutator grows and shrinks an array to counts based on numerical edge cases.

ArrayRandomizeOrderMutator

This mutator randomizes the order of items in an array.

ArrayReverseOrderMutator

This mutator reverses the order of items in an array.

ArrayVarianceMutator

This mutator grows and shrinks an array to a variance of counts based on the current size.

Used for all data elements

DataElementBitFlipper

This mutator produces test cases by flipping bits in the output value.

DataElementDuplicate

This mutator duplicates data elements.

DataElementRemove

This mutator removes data elements.

DataElementSwapNear

This mutator swaps data elements.

SampleNinjaMutator

This mutator combines data elements from different data sets.

Enabled when element is part of a size relation

SizedDataEdgeCase

This mutator causes the data portion of a relation to be sized as numerical edge cases.

SizedDataVariance

This mutator causes the data portion of a relation to be sized as numerical variances.

SizedEdgeCase

This mutator changes both sides of the relation (data and value) to match numerical edge cases.

SizedVariance

This mutator changes both sides of the relation (data and value) to match numerical variances of the current size.

Examples

Example 51. Empty Block

The simplest block is an empty block. This definition produces no output.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="BlockExample1">
        <Block>
        </Block>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output" publisher="ConsolePub">
                <DataModel ref="BlockExample1" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex" name="ConsolePub"/>
    </Test>
</Peach>

```

Output from this example.

```

>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 59388.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(0 bytes)
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

Example 52. Nested Blocks

Blocks can be nested as deep as required. Blocks help create logical structure and do not change the data contained within.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

<DataModel name="BlockExample2">
  <Block>
    <Block>
      <Block>
        <String value="1" />
      </Block>

      <Block>
        <String value="2" />
      </Block>

        <String value="3" />
      </Block>
      <String value="4" />
    </Block>
  </DataModel>

  <StateModel name="TheState" initialState="initial">
    <State name="initial">
      <Action type="output" publisher="ConsolePub">
        <DataModel ref="BlockExample2" />
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheState"/>

      <Publisher class="ConsoleHex" name="ConsolePub"/>
  </Test>
</Peach>
```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 30169.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(4 bytes)
00000000 31 32 33 34                                1234
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

Example 53. Naming A Block

Assign blocks a friendly name to make them easier to understand and debug.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

  <DataModel name="BlockExample2">
    <Block name="HeaderDef">
      <String name="Header" />
      <String name="Colon" value=":"/>
      <String name="Val"/>
    </Block>

    <Block name="DataDef">
      <Number name="Type" size="8" value="4"/>
      <Number name="Data" size="8" value="32"/>
    </Block>
  </DataModel>

  <StateModel name="TheState" initialState="initial">
    <State name="initial">
      <Action type="output" publisher="ConsolePub">
        <DataModel ref="BlockExample2" />
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex" name="ConsolePub"/>
  </Test>
</Peach>
```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 58326.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(3 bytes)
00000000  3A 04 20                      :?
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

Example 54. Referencing A Block

A Block can use a reference definition from another container element to form its base definition. In this example, the Block *MyName* gets its base definition from the DataModel named *OtherDataModel*. All child elements declared in *MyName* become part of the block.



If a declared child element and a child element from the referenced contain have the same name, the definition of the declared child element is used. In other words, if a naming collision occurs, the declared child element definition overrides the definition from the referenced container element.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="OtherDataModel">
        <String value="Hello World"/>
    </DataModel>

    <DataModel name="ThisDataModel">
        <Block name="MyName" ref="OtherDataModel"/> ①
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output" publisher="ConsolePub">
                <DataModel ref="ThisDataModel" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex" name="ConsolePub"/>
    </Test>
</Peach>

```

The Block "MyName" is defined using the referenced block "OtherDataModel". When parsed, the resulting data structure will look like this. <1>

```

<DataModel name="ThisDataModel">
    <Block name="MyName">
        <String value="Hello World"/>
    </Block>
</DataModel>

```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 61348.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(11 bytes)
00000000  48 65 6C 6C 6F 20 57 6F  72 6C 64          Hello World
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

Referencing allows for powerful templates to be built. This is a template for a Key: Value\r\nn.

```
<DataModel name="Template">
  <String name="Key" />
  <String value=":" token="true" />
  <String name="Value" />
  <String value="\r\n" token="true" />
</DataModel>
```

To use this template as a reference.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

  <DataModel name="Template">
    <String name="Key" />
    <String value=":" token="true" />
    <String name="Value" />
    <String value="\r\n" token="true" />
  </DataModel>

  <DataModel name="OtherModel">
    <String value="Before Block\r\n" />

    <Block name="Customized" ref="Template"> ①
      <String name="Key" value="Content-Length" />
      <String name="Value" value="55"/>
    </Block>
  </DataModel>

  <StateModel name="TheState" initialState="initial">
    <State name="initial">
      <Action type="output" publisher="ConsolePub">
        <DataModel ref="OtherModel" />
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex" name="ConsolePub"/>
  </Test>
</Peach>
```

Output from this example.

```

>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 64782.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(34 bytes)
00000000  42 65 66 6F 72 65 20 42  6C 6F 63 6B 0D 0A 43 6F  Before Block??Co
00000010  6E 74 65 6E 74 2D 4C 65  6E 67 74 68 3A 20 35 35  ntent-Length: 55
00000020  0D 0A
???
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

Two key things happened here. When parsed, the Customized Block replaced its structure with the DataModel of Template, adding the string values of ":" and "\r\n".

At the same time, the "Customized" block overwrote the values of the String elements for Key and Value, replacing them with "Content-Length" and 55. The final DataModel would be parsed as so.

```

<DataModel name="OtherModel">
  <String value="BeforeBlock" />

  <Block name="Customized" ref="Template">
    <String name="Key" value="Content-Length" />
    <String value=":" token="true" />
    <String name="Value" value="55" />
    <String value="\r\n" token="true" />
  </Block>
</DataModel>

```

21.3.4. Bool

The *Bool* element defines a Boolean value (values 0 or 1) that represents either true or false. The intended use of *Bool* is in JSON strings, where the values `true` and `false` are assigned to variables upon parsing with the JSON analyzer.

When not used in JSON strings, *Bool* is a one-bit number with a value of 0 or 1.

The *Bool* element is a child element of [DataModel](#), [Block](#), [Sequence](#) or [Choice](#).

Syntax

```
<Bool name="Boolean" value="1"/>
```

Attributes

Required:

No required attributes.

Optional:

name

Name of the boolean value.

value

The default value to assign to the boolean. Valid options are integer values 0 or 1.

constraint

A constraint in the form of a python expression. Used during data cracking.

mutable

Is data element changeable (should it be mutated during fuzzing), defaults to true. Valid options true and false.

minOccurs

The minimum number of times this number must occur. Defaults to 1. Valid options are a positive integer value.

maxOccurs

The maximum number of times this number can occur. Defaults to 1. Valid options are a positive integer value.

occurs

The actual number of times this number occurs. Defaults to 1.

Child Elements

Hint

Provide information to mutators.

Placement

Relocate an element after it has been cracked.

Mutators

The following mutators will operate on this element type:

Enabled when element is marked as an array

ArrayNumericalEdgeCasesMutator

This mutator will grow and shrink an array to counts based on numerical edge cases.

ArrayRandomizeOrderMutator

This mutator will randomize the order of items in an array.

ArrayReverseOrderMutator

This mutator will reverse the order of items in an array.

ArrayVarianceMutator

This mutator will grow and shrink an array to a variance of counts based on the current size.

Used for all data elements

DataElementBitFlipper

This mutator will produce test cases by flipping bits in the output value.

DataElementDuplicate

This mutator will duplicate data elements.

DataElementRemove

This mutator will remove data elements.

DataElementSwapNear

This mutator will swap data elements.

SampleNinjaMutator

This mutator will combine data elements from different data sets.

Enabled when element is part of a size relation

SizedDataEdgeCase

This mutator will cause the data portion of a relation to be sized as numerical edge cases.

SizedDataVariance

This mutator will cause the data portion of a relation to be sized as numerical variances.

SizedEdgeCase

This mutator will change both sides of the relation (data and value) to match numerical edge cases.

SizedVariance

This mutator will change both sides of the relation (data and value) to match numerical variances of the current size.

Examples

Example 55. Size

Produce Json string with a boolean value of true.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">
  <DataModel name="BoolExample">
    <Json>
      <Bool name="bool" value="1"/>
    </Json>
  </DataModel>

  <StateModel name="TheState" initialState="Initial">
    <State name="Initial">
      <Action type="output">
        <DataModel ref="BoolExample"/>
      </Action>
    </State>
  </StateModel>

  <Agent name="TheAgent" />

  <Test name="Default">
    <Agent ref="TheAgent"/>

    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex"/>
  </Test>
</Peach>
```

Output from this example.

```
>peach -1 --debug BoolExample1.xml

[*] Test 'Default' starting with random seed 28925.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "Initial".
Peach.Core.Dom.Action Run(Action): Output
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(13 bytes)
00000000  7B 22 62 6F 6F 22 3A  74 72 75 65 7D          {"bool":true}
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

21.3.5. Choice

The Choice element acts like a switch statement in programming languages.

The Choice element is a child element of [DataModel](#) or [Block](#). Choice elements are expected to have valid sub-elements; Peach will select a sub-element to fuzz.

Syntax

```
<Choice name="ChoiceBlock">
  <Block name="Type1">
    <!-- ... -->
  </Block>
  <Block name="Type2">
    <!-- ... -->
  </Block>
  <Block name="Type3">
    <!-- ... -->
  </Block>
</Choice>
```

Attributes

Required:

None.

Optional:

name

Name of the choice section.

length

Data element length.

lengthType

Length attribute unit of measure. Defaults to bytes.

mutable

Element mutability. Defaults to false.

constraint

Scripting expression that evaluates to true or false. Defaults to null.

minOccurs

The minimum number of times this element must occur. Defaults to 1.

- Used to define arrays with variable size. Arrays defined by min/maxOccurs generally have a relation defined.
- Adding this attribute, even with a value of 1, converts the element to an array.

maxOccurs

The maximum number of times this element can occur. Defaults to 1.

- Used to define arrays with static size.
- Adding this attribute, even with a value of 1, converts the element to an array.

occurs

Actual occurrences of element. Defaults to 1.

- Used to define arrays with static size.
- Adding this attribute, even with a value of 1, converts the element to an array.

Child Elements

Analyzer

Analyze current element post cracking, can dynamically change model.

Blob

Represents binary data (array of bytes) to create simple dumb fuzzers in Peach.

Block

Group one or more data elements in a logical structure.

Choice

Indicate all of the sub-elements are valid; however, only one sub-element should be selected.

Fixup

Are dynamic transformations such as checksums and CRCs.

Flags

Defines a set of bit sized flags.

Hint

Provides information to mutators.

Number

Defines a binary number of lengths 8, 16, 24, 32, or 64 bits.

Padding

Pads variably sized blocks or data models to provide uniformity or consistency.

Placement

Relocates an element after it has been cracked.

Transformer

Performs static transformations such as compression or encoding.

XmlElement

Defines an XML element, the basic building block of XML documents.

Mutators

The following mutators operate on this element type:

Enabled when an element is marked as an array

ArrayNumericalEdgeCasesMutator

This mutator grows and shrinks an array to counts based on numerical edge cases.

ArrayRandomizeOrderMutator

This mutator randomizes the order of items in an array.

ArrayReverseOrderMutator

This mutator reverses the order of items in an array.

ArrayVarianceMutator

This mutator grows and shrinks an array to a variance of counts based on the current size.

Used for all data elements

DataElementBitFlipper

This mutator produces test cases by flipping bits in the output value.

DataElementDuplicate

This mutator duplicates data elements.

DataElementRemove

This mutator removes data elements.

DataElementSwapNear

This mutator swaps data elements.

SampleNinjaMutator

This mutator combines data elements from different data sets.

Enabled when element is part of a size relation

SizedDataEdgeCase

This mutator causes the data portion of a relation to be sized as numerical edge cases.

SizedDataVariance

This mutator causes the data portion of a relation to be sized as numerical variances.

SizedEdgeCase

This mutator changes both sides of the relation (data and value) to match numerical edge cases.

SizedVariance

This mutator changes both sides of the relation (data and value) to match numerical variances of the current size.

Specific to this element type

ChoiceSwitch

This mutator produces test cases by arbitrarily changing the selected sub-element of a Choice.

Examples

Example 56. Basic Example

A basic Choice block. This choice example cracks or consumes data of type 1, 2, and 3. Much like a regular switch statement, a decision needs to be made on a token.

If the first 8 bits are 1, the remaining data is treated as a 32 bit number. If the first 8 bits are 2, the remaining data is treated as a 255 bytes of binary data. If the first 8 bits are 3, the remaining data is treated as a 8 byte string.

When fuzzing, Peach chooses one of the three types and fuzzes the output as an 8 bit number followed by the corresponding type.

Peach attempts to fill all three types. You can use data sets to specify which choice to make at different stages in the state model.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

<DataModel name="ChoiceExample1">
  <Choice name="Choice1">
    <Block name="Type1">
      <Number name="Str1" size="8" value="1" token="true" />
      <Number size="32"/>
    </Block>
```

```

<Block name="Type2">
  <Number name="Str2" size="8" value="2" token="true" />
  <Blob length="255" />
</Block>

<Block name="Type3">
  <Number name="Str3" size="8" value="3" token="true" />
  <String length="8" />
</Block>
</Choice>
</DataModel>

<StateModel name="TheState" initialState="initial">
  <State name="initial">
    <Action type="output" publisher="ConsolePub">
      <DataModel ref="ChoiceExample1" />
      <Data>
        <Field name="Choice1.Type1" value="1"/>
      </Data>
    </Action>

    <Action type="output" publisher="ConsolePub">
      <DataModel ref="ChoiceExample1" />
      <Data>
        <Field name="Choice1.Type2" value="2"/>
      </Data>
    </Action>
  </State>
</StateModel>

<Test name="Default">
  <StateModel ref="TheState"/>

  <Publisher class="ConsoleHex" name="ConsolePub"/>
</Test>
</Peach>

```

Output from this example.

This example attempts to crack at least 3 different choices and no more than 6.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

<DataModel name="ChoiceExample1">
<Choice name="Choice1" minOccurs="3" maxOccurs="6">

    <Block name="Type1">
        <Number name="Str1" size="8" value="1" token="true" />
        <Number size="32"/>
    </Block>

    <Block name="Type2">
        <Number name="Str2" size="8" value="2" token="true" />
        <Blob length="255" />
    </Block>

    <Block name="Type3">
        <Number name="Str3" size="8" value="3" token="true" />
        <String length="8" />
    </Block>
</Choice>
</DataModel>

<StateModel name="TheState" initialState="initial">
<State name="initial">
    <Action type="output" publisher="ConsolePub">
        <DataModel ref="ChoiceExample1" />
        <Data>
            <Field name="Choice1[0].Type1" value="" />
            <Field name="Choice1[1].Type3" value="" />
            <Field name="Choice1[2].Type2" value="" />
        </Data>
    </Action>

    <Action type="output" publisher="ConsolePub">
        <DataModel ref="ChoiceExample1" />
        <Data>
            <Field name="Choice1[0].Type1" value="" />
            <Field name="Choice1[1].Type1" value="" />
            <Field name="Choice1[2].Type1" value="" />
            <Field name="Choice1[3].Type1" value="" />
            <Field name="Choice1[4].Type1" value="" />
        </Data>
    </Action>
</State>
</StateModel>
```

```
</Action>
</State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex" name="ConsolePub"/>
</Test>
</Peach>
```

Output from this example.

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 59860.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(270 bytes)
00000000 01 00 00 00 00 03 00 00 00 00 00 00 00 00 02 00 ??????????????????
00000010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ??????????????????
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ??????????????????
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ??????????????????
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ??????????????????
00000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ??????????????????
00000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ??????????????????
00000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ??????????????????
00000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ??????????????????
00000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ??????????????????
000000A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ??????????????????
000000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ??????????????????
000000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ??????????????????
000000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ??????????????????
000000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ??????????????????
000000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ??????????????????
00000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ??????????????????

Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(25 bytes)
00000000 01 00 00 00 00 01 00 00 00 00 01 00 00 00 00 01 ??????????????????
00000010 00 00 00 00 01 00 00 00 00 00 00 00 ??????????????????

Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

21.3.6. DataModel

The DataModel element is a child of the Peach root element. DataModels define the structure of data blocks by specifying additional child elements such as [Number](#), [Blob](#), or [String](#).

Syntax

```
<DataModel name="TheDataModel">
    <String value="Hello World" />
</DataModel>
```

Attributes

Required:

name

Friendly name of the DataModel. This attribute is useful when referencing a DataModel as a template or when debugging.

Optional:

ref

Reference to another DataModel to use as a template.

Child Elements

[Analyzer](#)

Analyzes current element post cracking, can dynamically change model.

[Blob](#)

Represents binary data (array of bytes) to create simple dumb fuzzers in Peach.

[Block](#)

Groups one or more data elements in a logical structure.

[Choice](#)

Indicates all of the sub-elements are valid; however, only one sub-element should be selected.

[Fixup](#)

Are dynamic transformations such as checksums and CRCs.

[Flags](#)

Defines a set of bit sized flags.

[Hint](#)

Provides information to mutators.

Number

Defines a binary number of arbitrary bit size.

Padding

Pads variably sized blocks or data models to provide uniformity or consistency.

Placement

Relocates an element after it has been cracked.

Stream

Groups one or more data elements in a logical structure.

Transformer

Performs static transformations such as compression or encoding.

XmlElement

Defines an XML element, the basic building block of XML documents.

Mutators

The following mutators operate on this element type:

Used for all data elements

DataElementBitFlipper

This mutator produces test cases by flipping bits in the output value.

DataElementDuplicate

This mutator duplicates data elements.

DataElementRemove

This mutator removes data elements.

DataElementSwapNear

This mutator swaps data elements.

SampleNinjaMutator

This mutator combines data elements from different data sets.

Enabled when element is part of a size relation

SizedDataEdgeCase

This mutator causes the data portion of a relation to be sized as numerical edge cases.

SizedDataVariance

This mutator causes the data portion of a relation to be sized as numerical variances.

SizedEdgeCase

This mutator changes both sides of the relation (data and value) to match numerical edge cases.

SizedVariance

This mutator changes both sides of the relation (data and value) to match numerical variances of the current size.

Examples

Example 58. DataModel Example

Any number of *DataModels* can be specified in a Peach Pit file. However, each *DataModel* must have a unique name. You can break down complex formats into smaller models by splitting them into logical parts. This makes DataModels easier to read, debug, and re-use.

An example DataModel named "HelloWorld" contains a single string and outputs "Hello world!"

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

  <DataModel name="HelloWorld">
    <String value="Hello world!" />
  </DataModel>

  <StateModel name="TheState" initialState="initial">
    <State name="initial">
      <Action type="output">
        <DataModel ref="HelloWorld" />
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex"/>
  </Test>
</Peach>
```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 63002.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(12 bytes)
00000000  48 65 6C 6C 6F 20 77 6F  72 6C 64 21          Hello world!
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

DataModels can reference other DataModels and inherit child elements with the ref attribute.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

  <DataModel name="ParentModel">
    <String value="Hello " />
  </DataModel>

  <DataModel name="HelloWorldModel" ref="ParentModel" >
    <String value=" world!" />
  </DataModel>

  <StateModel name="TheState" initialState="initial">
    <State name="initial">
      <Action type="output">
        <DataModel ref="HelloWorldModel" />
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheState"/>

      <Publisher class="ConsoleHex"/>
    </Test>
  </Peach>

```

Output from this example.

```
>peach -1 --debug DocSample.xml

[*] Test 'Default' starting with random seed 35043.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(13 bytes)
00000000  48 65 6C 6C 6F 20 20 77  6F 72 6C 64 21          Hello world!
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

Example 59. References (ref attribute)

When a reference (ref attribute) is supplied, the contents of the referenced DataModel are copied to create the base of the new DataModel. Any child elements in the new DataModel override elements from that base with the same name. In this example, the child DataModel Customized contains a String named Key. The value from the child DataModel overwrites the String "Key" of the parent DataModel, which has no value.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="Template">
        <String name="Key" />
        <String value=":" token="true" />
        <String name="Value" />
        <String value="\r\n" token="true" />
    </DataModel>

    <DataModel name="Customized" ref="Template">
        <String name="Key" value="Content-Length" />
        <String name="Value">
            <Relation type="size" of="HttpBody" />
        </String>
        <Blob name="HttpBody" />
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="Customized" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>
```

Output from this example.

```

>peach -1 --debug DocSample.xml

*] Test 'Default' starting with random seed 3945.

R1,-,-] Performing iteration
each.Core.Engine runTest: Performing recording iteration.
each.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
each.Core.Dom.Action ActionType.Output
each.Core.Publishers.ConsolePublisher start()
each.Core.Publishers.ConsolePublisher open()
each.Core.Publishers.ConsolePublisher output(4 bytes)
0000000 3A 20 0D 0A : ?? ①
each.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
each.Core.Dom.Action ActionType.Output
each.Core.Publishers.ConsolePublisher output(19 bytes)
0000000 43 6F 6E 74 65 6E 74 2D 4C 65 6E 67 74 68 3A 20 Content-Length: ②
0000010 30 0D 0A 0??
each.Core.Publishers.ConsolePublisher close()
each.Core.Engine runTest: context.config.singleIteration == true
each.Core.Publishers.ConsolePublisher stop()

*] Test 'Default' finished.

```

```

<1> The output of "Template"  is " : \r\n"
<2> The output of "Customized" is "Content-Length: 100\r\n"

```

When parsed into a DataModel, Customized looks like a combination of both data models.

```

<DataModel name="Customized" ref="Template">
  <String name="Key" value="Content-Length" />
  <String value=":" token="true" />
  <String name="Value">
    <Relation type="size" of="HttpBody" />
  </String>
  <String value="\r\n" token="true" />
  <Blob name="HttpBody"/>
</DataModel>

```

21.3.7. Double

The *Double* element defines a floating-point number of 32 or 64 bits. Floating point values are packed into a byte representation with a byte order of little or big endian.

The Double element is a child element of [DataModel](#), [Block](#), [Sequence](#) or [Choice](#).



The floating point representation used for the *Double* element is based on the IEEE 754 floating point specification. For floating point values not conforming to IEEE 754, consider using a [Blob](#) data type as an alternative.

Syntax

```
<Double name="Precision" size="64" endian="big"/>
```

Attributes

Required:

size

Size of floating point number in bits. Valid options are 32 or 64.

Optional:

name

Name of the floating point number.

value

The default value to assign to the floating point number.

valueType

The representation of the value. Valid options are string and hex.

token

This element is treated as a token when parsing, defaults to false. Valid options true and false.

endian

Byte order of the number, defaults to little. Valid options are big, little, and network. Network is the same as big.

constraint

A constraint in the form of a python expression. Used during data cracking.

mutable

Is data element changeable (should it be mutated during fuzzing), defaults to true. Valid options

true and false.

minOccurs

The minimum number of times this number must occur. Defaults to 1. Valid options are a positive integer value.

maxOccurs

The maximum number of times this number can occur. No default. Valid options are a positive integer value.

occurs

The actual number of times this number occurs. Defaults to 1.

Child Elements

Analyzer

Analyze current element post cracking, can dynamically change model.

Fixup

Dynamic transformations such as checksums and CRCs.

Hint

Provide information to mutators.

Placement

Relocate an element after it has been cracked.

Relation

Modeling of relationships in the data (such as comparisons)

Transformer

Static transformations such as compression or encoding.

Mutators

The following mutators will operate on this element type:

Enabled when element is marked as an array

ArrayNumericalEdgeCasesMutator

This mutator will grow and shrink an array to counts based on numerical edge cases.

ArrayRandomizeOrderMutator

This mutator will randomize the order of items in an array.

ArrayReverseOrderMutator

This mutator will reverse the order of items in an array.

ArrayVarianceMutator

This mutator will grow and shrink an array to a variance of counts based on the current size.

Used for all data elements

DataElementBitFlipper

This mutator will produce test cases by flipping bits in the output value.

DataElementDuplicate

This mutator will duplicate data elements.

DataElementRemove

This mutator will remove data elements.

DataElementSwapNear

This mutator will swap data elements.

SampleNinjaMutator

This mutator will combine data elements from different data sets.

Enabled when element is part of a size relation

SizedDataEdgeCase

This mutator will cause the data portion of a relation to be sized as numerical edge cases.

SizedDataVariance

This mutator will cause the data portion of a relation to be sized as numerical variances.

SizedEdgeCase

This mutator will change both sides of the relation (data and value) to match numerical edge cases.

SizedVariance

This mutator will change both sides of the relation (data and value) to match numerical variances of the current size.

Specific to this element type

DoubleRandom

This mutator will produce random values from the available numerical space.

DoubleVariance

This mutator will produce values near the current value of a number.

Examples

Example 60. Size

Produce 32 bit (4 byte) floating point number with a default value of 5.1:

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">
  <DataModel name="DoubleExample1">
    <Double name="Hi51" value="5.1" size="32"/>
  </DataModel>

  <StateModel name="TheState" initialState="Initial">
    <State name="Initial">
      <Action type="output">
        <DataModel ref="DoubleExample1"/>
      </Action>
    </State>
  </StateModel>

  <Agent name="TheAgent" />

  <Test name="Default">
    <Agent ref="TheAgent"/>

    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex"/>
  </Test>
</Peach>
```

Output from this example.

```
>peach -1 --debug DoubleExample1.xml

[*] Test 'Default' starting with random seed 12232.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "Initial".
Peach.Core.Dom.Action Run(Action): Output
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(4 bytes)
00000000  33 33 A3 40                                33.@ ①
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

① The 32 bit little endian value 5.1

To use 64 bits (four byte) change the size to 64.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

    <DataModel name="DoubleExample2">
        <Double name="Hi51" value="5.1" size="64"/>
    </DataModel>

    <StateModel name="TheState" initialState="Initial">
        <State name="Initial">
            <Action type="output">
                <DataModel ref="DoubleExample2"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="TheAgent" />

    <Test name="Default">
        <Agent ref="TheAgent"/>

        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>

```

Output from this example.

```

>peach -1 --debug DoubleExample2.xml

[*] Test 'Default' starting with random seed 51105.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "Initial".
Peach.Core.Dom.Action Run(Action): Output
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(8 bytes)
00000000  66 66 66 66 66 14 40                      ffffff.@ ①
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

① The 64 bit little endian value 5.1

NOTE: Double elements use the **size** attribute which is, by default, the size in **bits**. Double elements do not accept **length** attribute used by other elements.

Example 61. Endian

To change the endianness of the floating point number set the **endian** attribute. Endianness defines in which order the bytes are the least or most significant.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

    <DataModel name="DoubleExample3">
        <Double name="big" value="10.0" size="64" endian="big" />
    </DataModel>

    <DataModel name="DoubleExample4">
        <Double name="little" value="10.0" size="64" endian="little" />
    </DataModel>

    <StateModel name="TheState" initialState="Initial">
        <State name="Initial">
            <Action type="output">
                <DataModel ref="DoubleExample3"/>
            </Action>
            <Action type="output">
                <DataModel ref="DoubleExample4"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="TheAgent" />

    <Test name="Default">
        <Agent ref="TheAgent"/>

        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>
```

Output from this example.

```

>peach -1 --debug DoubleEndianExample.xml

[*] Test 'Default' starting with random seed 35381.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "Initial".
Peach.Core.Dom.Action Run(Action): Output
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(8 bytes)
00000000  40 24 00 00 00 00 00 00 @$..... ①
Peach.Core.Dom.Action Run(Action_1): Output
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Publishers.ConsolePublisher output(8 bytes)
00000000  00 00 00 00 00 00 24 40 .....$@ ②
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

① Big endian outputs the bytes in the order

② Little endian outputs the bytes in the order

Note, however, that endian-ness doesn't have any impact on output if the `valueType` is "hex":

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

    <DataModel name="DoubleExample5">
        <Double name="abcd" valueType="hex" value="ABCDEF01ABCDEF01" size="64"
endian="little" />
    </DataModel>

    <DataModel name="DoubleExample6">
        <Double name="abcd" valueType="hex" value="ABCDEF01ABCDEF01" size="64"
endian="big" />
    </DataModel>

    <StateModel name="TheState" initialState="Initial">
        <State name="Initial">
            <Action type="output">
                <DataModel ref="DoubleExample5"/>
            </Action>
            <Action type="output">
                <DataModel ref="DoubleExample6"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="TheAgent" />

    <Test name="Default">
        <Agent ref="TheAgent"/>

        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>
```

Output from this example.

```

>peach -1 --debug NumberEndianExample.xml

[*] Test 'Default' starting with random seed 53121.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "Initial".
Peach.Core.Dom.Action Run(Action): Output
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(8 bytes)
00000000 AB CD EF 01 AB CD EF 01 ..... ①
Peach.Core.Dom.Action Run(Action_1): Output
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Publishers.ConsolePublisher output(8 bytes)
00000000 AB CD EF 01 AB CD EF 01 ..... ②
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

① For little we see the expected output

② For big endian, nothing changes

When the attribute `valueType` is set to "hex" this indicates that the ordering is exactly as specified. The `endian` attribute still impacts mutation and input parsing.

21.3.8. Flag

The Flag element defines a specific bit field in a Flags container.

See also parent element [Flags](#).

Syntax

```
<Flags name="options" size="16">
  <Flag name="compression" position="0" size="1" />
  <Flag name="compressionType" position="1" size="3" />
  <Flag name="opcode" position="10" size="2" value="5" />
</Flags>
```

Attributes

Required:

size

Size in bits

position

Location of the flag. *Position* identifies the location of the first bit of a flag. The value is a zero-based index.

Optional:

name

Name of the element.

value

The default value contained within the element.

valueType

The format in which the default value is expressed. (i.e. hex, string, or literal).

token

Is data element a token? Default is false.

mutable

Is data element changeable (should it be mutated), defaults to true.

Child Elements

Analyzer

Analyzes the current element post cracking; can dynamically change model.

Fixup

Performs dynamic transformations such as checksums and CRCs.

Hint

Provides information to mutators.

Placement

Relocates an element after it has been cracked.

Relation

Identifies a type of relationship with another data element, such as count.

Transformer

Static transformations such as compression or encoding.

Mutators

The following mutators operate on this element type:

Enabled when element is marked as an array

ArrayNumericalEdgeCasesMutator

This mutator grows and shrinks an array to counts based on numerical edge cases.

ArrayRandomizeOrderMutator

This mutator randomizes the order of items in an array.

ArrayReverseOrderMutator

This mutator reverses the order of items in an array.

ArrayVarianceMutator

This mutator grows and shrinks an array to a variance of counts based on the current size.

Used for all data elements

DataElementBitFlipper

This mutator produces test cases by flipping bits in the output value.

DataElementDuplicate

This mutator duplicates data elements.

DataElementRemove

This mutator removes data elements.

DataElementSwapNear

This mutator swaps data elements.

SampleNinjaMutator

This mutator combines data elements from different data sets.

Enabled when element is part of a size relation

SizedDataEdgeCase

This mutator causes the data portion of a relation to be sized as numerical edge cases.

SizedDataVariance

This mutator causes the data portion of a relation to be sized as numerical variances.

SizedEdgeCase

This mutator changes both sides of the relation (data and value) to match numerical edge cases.

SizedVariance

This mutator changes both sides of the relation (data and value) to match numerical variances of the current size.

Specific to this element type

ExtraValues

This mutator provides extra test case values on a per-data element basis.

NumberEdgeCase

This mutator produces numerical edge cases for integer values.

NumberRandom

This mutator produces random values from the available numerical space.

NumberVariance

This mutator produces values near the current value of a number.

Examples

Example 62. Example of Flags

This example shows a real-world example of a flag set by modeling a TCP packet (without options). This example also shows using relations with the [Flag](#) element.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
```

```

xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="TcpPacket">
    <Block name="Header">
        <Number name="SrcPort" size="16" endian="big" value="1234"/>
        <Number name="DestPort" size="16" endian="big" value="1234"/>
        <Number name="SequenceNumber" size="32" endian="big" valueType="hex"
value="0043a577"/>
        <Number name="AcknowledgmentNumber" size="32" endian="big" value="0"/>

        <Flags name="ControlBits" size="16" endian="big">
            <Flag name="Offset" position="0" size="4" valueType="hex">
                <Relation type="size" of="Header" expressionGet="size * 4"
expressionSet="size / 4"/>
            </Flag>
            <Flag name="Reserved" position="4" size="3"/>
            <Flag name="NS" position="7" size="1"/>
            <Flag name="CWR" position="8" size="1"/>
            <Flag name="ECE" position="9" size="1"/>
            <Flag name="URG" position="10" size="1"/>
            <Flag name="ACK" position="11" size="1"/>
            <Flag name="PSH" position="12" size="1"/>
            <Flag name="RST" position="13" size="1"/>
            <Flag name="SYN" position="14" size="1"/>
            <Flag name="FIN" position="15" size="1"/>
        </Flags>

        <Number name="WindowSize" size="16" endian="big" valueType="hex" value=
"aaaa"/>
        <Number name="CheckSum" size="16" endian="big">
            <Fixup class="TCPChecksumFixup">
                <Param name="ref" value="TcpPacket" />
                <Param name="src" value="127.0.0.1" />
                <Param name="dst" value="127.0.0.1" />
            </Fixup>
        </Number>
        <Number name="UrgentPointer" size="16" endian="big"/>
    </Block>

    <Blob name="TcpPayload" value="this is a packet.\n"/>
</DataModel>

<StateModel name="TheStateModel" initialState="InitialState">
    <State name="InitialState">
        <Action type="output">
            <DataModel ref="TcpPacket" />
        </Action>
    </State>

```

```

</StateModel>

<Test name="Default">
    <StateModel ref="TheStateModel" />

    <Publisher class="ConsoleHex"/> </Test>
</Peach>
```

Produces the following output:

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 17543.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(38 bytes)
00000000  04 D2 04 D2 00 43 A5 77  00 00 00 00 50 00 AA AA  ?????C?w????P???
00000010  1D F6 00 00 74 68 69 73  20 69 73 20 61 20 70 61  ???this is a pa
00000020  63 6B 65 74 2E 0A                                         cket.?

Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

21.3.9. Flags

The Flags element defines a set of bit-sized flags.

See also [Flag](#).



The *Flags* element is largely provided for backwards compatibility with older versions of Peach that did not support arbitrarily-sized [Number](#) elements.

Syntax

```
<Flags name="options" size="16">
  <Flag name="compression" position="0" size="1" />
  <Flag name="compressionType" position="1" size="3" />
  <Flag name="opcode" position="10" size="2" value="5" />
</Flags>
```

Attributes

Required:

size

Size of the set of flags in bits. Only 8-bit aligned values are supported: 8, 16, 24, 32, 64.

Optional:

name

Name of the element.

endian

Byte order of number. Defaults to *little*.

token

Specifies whether this data element is a token. Defaults to *false*.

mutable

Specifies whether this data element should be mutated. Defaults to *true*.

constraint

Scripting expression that evaluates to true or false. Defaults to none.

minOccurs

The minimum number of times this element must occur. Defaults to 1.

- Used to define arrays with variable size. Arrays defined by min/maxOccurs generally have a

relation defined.

- Adding this attribute, even with a value of 1, converts the element to an array.

maxOccurs

The maximum number of times this element can occur. Defaults to 1.

- Used to define arrays with static size.
- Adding this attribute, even with a value of 1, converts the element to an array.

occurs

Actual occurrences of element. Defaults to 1.

- Used to define arrays with static size.
- Adding this attribute, even with a value of 1, converts the element to an array.

Child Elements

Required:

None.

Optional:

Analyzer

Analyzes current element post cracking, can dynamically change model.

Fixup

Performs dynamic transformations such as checksums and CRCs.

Flag

Defines a specific bit field in a Flags container.

Hint

Provides information to mutators.

Placement

Relocates an element after it has been cracked.

Transformer

Performs static transformations such as compression or encoding.

Mutators

The following mutators operate on this element type:

Enabled when element is marked as an array

ArrayNumericalEdgeCasesMutator

This mutator grows and shrinks an array to counts based on numerical edge cases.

ArrayRandomizeOrderMutator

This mutator randomizes the order of items in an array.

ArrayReverseOrderMutator

This mutator reverses the order of items in an array.

ArrayVarianceMutator

This mutator grows and shrinks an array to a variance of counts based on the current size.

Used for all data elements

DataElementBitFlipper

This mutator produces test cases by flipping bits in the output value.

DataElementDuplicate

This mutator duplicates data elements.

DataElementRemove

This mutator removes data elements.

DataElementSwapNear

This mutator swaps data elements.

SampleNinjaMutator

This mutator combines data elements from different data sets.

Enabled when element is part of a size relation

SizedDataEdgeCase

This mutator causes the data portion of a relation to be sized as numerical edge cases.

SizedDataVariance

This mutator causes the data portion of a relation to be sized as numerical variances.

SizedEdgeCase

This mutator changes both sides of the relation (data and value) to match numerical edge cases.

SizedVariance

This mutator changes both sides of the relation (data and value) to match numerical variances of the current size.

Examples

Example 63. Example of Flags

This example shows a real world example of a flag set by modeling a TCP packet (without options). This example also shows using relations with the `Flag` element.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="TcpPacket">
    <Block name="Header">
        <Number name="SrcPort" size="16" endian="big" value="1234"/>
        <Number name="DestPort" size="16" endian="big" value="1234"/>
        <Number name="SequenceNumber" size="32" endian="big" valueType="hex"
value="0043a577"/>
        <Number name="AcknowledgmentNumber" size="32" endian="big" value="0"/>

        <Flags name="ControlBits" size="16" endian="big">
            <Flag name="Offset" position="0" size="4" valueType="hex">
                <Relation type="size" of="Header" expressionGet="size * 4"
expressionSet="size / 4"/>
            </Flag>
            <Flag name="Reserved" position="4" size="3"/>
            <Flag name="NS" position="7" size="1"/>
            <Flag name="CWR" position="8" size="1"/>
            <Flag name="ECE" position="9" size="1"/>
            <Flag name="URG" position="10" size="1"/>
            <Flag name="ACK" position="11" size="1"/>
            <Flag name="PSH" position="12" size="1"/>
            <Flag name="RST" position="13" size="1"/>
            <Flag name="SYN" position="14" size="1"/>
            <Flag name="FIN" position="15" size="1"/>
        </Flags>

        <Number name="WindowSize" size="16" endian="big" valueType="hex" value=
"aaaa"/>
        <Number name="CheckSum" size="16" endian="big">
            <Fixup class="TCPChecksumFixup">
                <Param name="ref" value="TcpPacket" />
                <Param name="src" value="127.0.0.1" />
                <Param name="dst" value="127.0.0.1" />
            </Fixup>
        </Number>
        <Number name="UrgentPointer" size="16" endian="big"/>
    </Block>
</DataModel>
```

```

</Block>

<Blob name="TcpPayload" value="this is a packet.\n"/>
</DataModel>

<StateModel name="TheStateModel" initialState="InitialState">
    <State name="InitialState">
        <Action type="output">
            <DataModel ref="TcpPacket" />
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheStateModel" />

    <Publisher class="ConsoleHex"/>
</Test>
</Peach>

```

Produces the following output:

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 17543.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(38 bytes)
00000000  04 D2 04 D2 00 43 A5 77  00 00 00 00 50 00 AA AA  ?????C?w????P???
00000010  1D F6 00 00 74 68 69 73  20 69 73 20 61 20 70 61  ???this is a pa
00000020  63 6B 65 74 2E 0A                               cket.?

Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

21.3.10. Frag

This element is used model fragmentation layers in protocols. The fragmentation method can be simplistic such as length based or more complex requiring custom logic and inspection of payload to determine how to partition the data.

The Peach fragmentation support also has two corresponding Actions that are specific for supporting fragmented protocols: [outfrag](#) and [infrag](#). These actions are aware of fragmentation and will correctly output multiple fragments or receive and reconstruct multiple fragments.

The *Frag* element expects two child elements to exist: *Payload* and *Template*. The *Template* model defines what each fragment will look like while the *Payload* model contains the unfragmented data.

When outputting a *Frag* element via the [outfrag](#) action, the *Frag* element will generate a new child element called *Rendering* of type [Sequence](#) that contains each of the fragments. Each fragment is a copy of the *Template* element with the fragmented *Payload* data placed into its *FragData* element. During an *outfrag* action, each element in *Rendering* is sent via its own *output* action.

When receiving input into a *Frag* element via the [infrag](#) action, each fragment will be placed into a child element called *Rendering* of type [Sequence](#). Once all fragments have been received the unfragmented data is cracked into the *Payload* element.

All *Frag* elements are paired with a fragmentation algorithm which contains the logic for generating fragments and also reconstructing the fragments. Peach comes with a single fragmentation algorithm called [ByLength](#) which supports simple size based fragmentation. For more complicated fragmentation logic a custom fragmentation algorithm must be written. Currently the fragmentation algorithms must be written in a Microsoft.NET language such as C#.



The *Frag* element must be the first and only element in a [DataModel](#). Multiple layers of fragmentation is not supported.

See also: [action outfrag](#), [action infrag](#), [Sequence](#), [DataModel](#).

Syntax

```

<Frag fragLength="1024" totalLengthField="TotalLength" fragLengthField="FragLength"
fragIndexField="FragIndex">

    <Block name="Template">
        <Number name="FragLength" size="32"/>
        <Number name="FragIndex" size="32"/>
        <Number name="TotalLength" size="32"/>

        <Blob name="FragData" />
    </Block>

    <Block name="Payload">
        <!-- Payload contents -->
    </Block>

</Frag>

```

Attributes

Required:

No required attributes.

Optional:

name

Name of the boolean value.

fragLength

Fragment size in bytes. Used in combination with a fragmentation algorithm that supports this attribute such as the default *ByLength* algorithm.

class

Fragmentation algorithm to use, defaults to *ByLength*. The fragmentation algorithm provides the logic for generating fragments and also reconstructing fragments. Custom fragmentation algorithms are supported by extending from the [FragmentAlgorithm](#) base class.

constraint

A constraint in the form of a python expression. Used during data cracking.

payloadOptional

Protocol allows for null payload. Some fragmentation layers will send a single fragment with a null payload. This attribute enabled support for this mode of operation.

totalLengthField

Name of total length field in *Template* element. If the fragment template has a field for the total length of the payload it can be automatically set using this attribute.

fragmentLengthField

Name of fragment length field in *Template* element. If the fragment template has a field for the length of the current fragment it can be automatically set using this attribute.

fragmentOffsetField

Name of fragment offset field in *Template* element. If the fragment template has a field for the offset of the current fragment it can be automatically set using this attribute.

fragmentIndexField

Name of fragment index field in *Template* element. If the fragment template has a field for the offset of the current fragment it can be automatically set using this attribute.

reassembleDataSet

Controls how a file based [data set](#) is applied to a model containing a *Frag* element. If the value is *true*, the data set is expected to be fragmented and the *Frag* element will perform reassembly. If the value is *false*, the data set is not expected to be fragmented and is expected to only contain the value and will be applied directly to the *Payload* element. The default value is *false*.

Child Elements

No supported child elements.

Mutators

The mutations for the *Frag* element are:

- Fuzz each fragment. All elements specified in *Template* will be fuzzed.
- Fuzz *Payload*. All elements specified in *Payload* will be fuzzed.
- Fuzz the *Rendering* sequence using all Array mutators.
 - Out of order fragments
 - Missing fragments
 - Duplicated fragments

The following mutators will operate on this element type:

Specific to this element type

[**ArrayNumericalEdgeCasesMutator**](#)

This mutator will grow and shrink an array to counts based on numerical edge cases.

[**ArrayRandomizeOrderMutator**](#)

This mutator will randomize the order of items in an array.

ArrayReverseOrderMutator

This mutator will reverse the order of items in an array.

ArrayVarianceMutator

This mutator will grow and shrink an array to a variance of counts based on the current size.

Used for all data elements

DataElementDuplicate

This mutator will duplicate data elements.

DataElementRemove

This mutator will remove data elements.

DataElementSwapNear

This mutator will swap data elements.

SampleNinjaMutator

This mutator will combine data elements from different data sets.

Examples

Example 64. Simple Example

Produce three fragments with each fragment containing the current fragment length, fragment sequence and total length of data. The Payload is 30 bytes of 0x41.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

    <DataModel name="Fragmented">
        <Frag fragLength="10"
              totalLengthField="TotalLength"
              fragmentLengthField="FragLength"
              fragmentIndexField="FragIndex">

            <Block name="Template">
                <Number name="FragLength" size="32"/>
                <Number name="FragIndex" size="32"/>
                <Number name="TotalLength" size="32"/>

                <Blob name="FragData" />
            </Block>

            <Block name="Payload">
                <Blob valueType="hex" value="
                    41 41 41 41 41 41 41 41 41 41
                    41 41 41 41 41 41 41 41 41 41
                    41 41 41 41 41 41 41 41 41 41"/>
            </Block>
        </Frag>
    </DataModel>

    <StateModel name="TheState" initialState="Initial">
        <State name="Initial">
            <Action type="outfrag">
                <DataModel ref="Fragmented"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>
        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>

```

The example pit will produce three fragments with 10 bytes of payload per-fragment.

Output from this example:

```

>peach -1 --debug example.xml

[*] Web site running at: http://10.0.1.87:8888/

[*] Test 'Default' starting with random seed 7010.
2016-07-07 14:26:22.2979 Peach.Pro.Core.Loggers.JobLogger Writing debug.log to:
C:\peach-pro\output\win_x64_debug\bin\Logs\example.xml_20160707142621\debug.log

[R1,-,-] Performing iteration
2016-07-07 14:26:22.4288 Peach.Core.Engine runTest: Performing control recording
iteration.
2016-07-07 14:26:22.4690 Peach.Pro.Core.Dom.Frag Generating fragments:
2016-07-07 14:26:22.4870 Peach.Core.Dom.StateModel Run(): Changing to state
"Initial".
2016-07-07 14:26:22.4951 Peach.Core.Dom.Action Run(Action): Outfrag
2016-07-07 14:26:22.6139 Peach.Pro.Core.Publishers.ConsolePublisher start()
2016-07-07 14:26:22.6139 Peach.Pro.Core.Publishers.ConsolePublisher open()
2016-07-07 14:26:22.6188 Peach.Pro.Core.Publishers.ConsolePublisher output(22 bytes)
①
00000000 0A 00 00 00 01 00 00 00 1E 00 00 00 41 41 41 41 .....AAAA
00000010 41 41 41 41 41
2016-07-07 14:26:22.6188 Peach.Pro.Core.Publishers.ConsolePublisher output(22 bytes)
②
00000000 0A 00 00 00 02 00 00 00 1E 00 00 00 41 41 41 41 .....AAAA
00000010 41 41 41 41 41
2016-07-07 14:26:22.6188 Peach.Pro.Core.Publishers.ConsolePublisher output(22 bytes)
③
00000000 0A 00 00 00 03 00 00 00 1E 00 00 00 41 41 41 41 .....AAAA
00000010 41 41 41 41 41
2016-07-07 14:26:22.6188 Peach.Pro.Core.Publishers.ConsolePublisher close()
2016-07-07 14:26:22.6329 Peach.Core.Engine runTest: context.config.singleIteration ==
true
2016-07-07 14:26:22.6329 Peach.Pro.Core.Publishers.ConsolePublisher stop()
2016-07-07 14:26:22.6329 Peach.Core.Engine EndTest: Stopping all agents and monitors

[*] Test 'Default' finished.

```

① First fragment. Notice sequence number is 1.

② Second fragment. Notice sequence number is 2.

③ Third fragment. Notice sequence number is 3.

21.3.11. JsonArray

JsonArray is used to represent a JSON array. It's similar in operation to [Sequence](#).

A *JsonArray* element is used to construct an array with all its elements predefined. A *JsonArray* is able to contain multiple data types at once. Unlike regular arrays in Peach, the size of the array is determined by the number of elements within the array.

Syntax

```
<JsonObject>
    <JsonArray propertyName="items">
        <JsonString name="name" value="Peach Toy" />
        <JsonInteger name="stock" value="10" />
    </JsonArray>
</JsonObject>

<JsonArray>
    <JsonString name="name" value="Peach Toy" />
    <JsonInteger name="stock" value="10" />
</JsonArray>
```

Attributes

Required:

There are no required attributes.

Optional:

name

Name of the sequence.

isNull

Set default value of this element to null (defaults to false)

type

Type controls the output encoding allowing both traditional json and also binary json (bson). Supported values: *json* or *bson*. Defaults to *json*.

mutable

Is data element changeable (should it be mutated), defaults to false.

Child Elements

[Analyzer](#)

Analyze current element post cracking, can dynamically change model.

Fixup

Dynamic transformations such as checksums and CRCs.

Hint

Provide information to mutators.

JsonArray

JSON array

JsonBool

JSON boolean

JsonBlob

JSON BLOB

JsonDouble

JSON double/float

JsonInteger

JSON integer

JsonObject

JSON object

JsonString

JSON string

Placement

Relocate an element after it has been cracked.

Transformer

Static transformations such as compression or encoding.

Mutators

The following mutators will operate on this element type:

Specific to this element type

ArrayNumericalEdgeCasesMutator

This mutator will grow and shrink an array to counts based on numerical edge cases.

ArrayRandomizeOrderMutator

This mutator will randomize the order of items in an array.

ArrayReverseOrderMutator

This mutator will reverse the order of items in an array.

ArrayVarianceMutator

This mutator will grow and shrink an array to a variance of counts based on the current size.

Used for all data elements

DataElementDuplicate

This mutator will duplicate data elements.

DataElementRemove

This mutator will remove data elements.

DataElementSwapNear

This mutator will swap data elements.

SampleNinjaMutator

This mutator will combine data elements from different data sets.

Enabled when element is part of a size relation

SizedDataEdgeCase

This mutator will cause the data portion of a relation to be sized as numerical edge cases.

SizedDataVariance

This mutator will cause the data portion of a relation to be sized as numerical variances.

SizedEdgeCase

This mutator will change both sides of the relation (data and value) to match numerical edge cases.

SizedVariance

This mutator will change both sides of the relation (data and value) to match numerical variances of the current size.

Examples

Example 65. Simple Example

Simple example of a JSON array with two elements.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">
  <DataModel name="SequenceExample">
    <JsonObject>
      <JsonArray propertyName="items">
        <JsonString name="name" value="Peach Toy" />
        <JsonInteger name="stock" value="10" />
      </JsonArray>
    </JsonObject>
  </DataModel>

  <StateModel name="TheState" initialState="Initial">
    <State name="Initial">
      <Action type="output">
        <DataModel ref="SequenceExample"/>
      </Action>
    </State>
  </StateModel>

  <Agent name="TheAgent" />

  <Test name="Default">
    <Agent ref="TheAgent"/>

    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex"/>
  </Test>
</Peach>

```

Output from this example.

```

>peach -1 --debug example.xml

[*] Web site running at: http://10.0.1.57:8888/

[*] Test 'Default' starting with random seed 23959.
Peach.Pro.Core.Loggers.JobLogger Writing debug.log to:
c:\peach\Logs\example.xml_20160223175744\debug.log

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "Initial".
Peach.Core.Dom.Action Run(Action): Output
Peach.Pro.Core.Publishers.ConsolePublisher start()
Peach.Pro.Core.Publishers.ConsolePublisher open()
Peach.Pro.Core.Publishers.ConsolePublisher output(26 bytes)
00000000  7B 22 69 74 65 6D 73 22  3A 5B 22 50 65 61 63 68 {"items": ["Peach
00000010  20 54 6F 79 22 2C 31 30  5D 7D             Toy", 10]}
Peach.Pro.Core.Publishers.ConsolePublisher close()
Peach.Core.Agent.AgentManager DetectedFault: TheAgent
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Pro.Core.Publishers.ConsolePublisher stop()
Peach.Core.Engine EndTest: Stopping all agents and monitors

[*] Test 'Default' finished.

```

21.3.12. JsonBlob

A **JsonBlob** [3: Blob stands for "binary large object" a term used by databases to represent a column of binary data.] is used to represent binary JSON data (array of bytes). JsonBlobs are base64 encoded.

By definition, the internal contents of a JsonBlob is unknown. Consequently, Blobs are dumb fuzzed, as the list of applicable mutators below indicates.

Syntax

```

<JsonObject>
    <JsonBlob propertyName="image" valueType="hex" value="01 06 22 03" />
</JsonObject>

<JsonBlob valueType="hex" value="01 06 22 03" />

```

Attributes

Required:

There are no required attributes.

Optional:

name

Element name

isNull

Is this elements value null. JSON output will be `null` instead of value.

type

Type controls the output encoding allowing both traditional json and also binary json (bson).

Supported values: `json` or `bson`. Defaults to `json`.

value

Default value.

valueType

Format of value attribute.

mutable

Is the blob changeable (should it be fuzzed). Defaults to true.

token

This element should be treated as a token when parsing. Defaults to false.

Child Elements

Analyzer

Analyzes current element post cracking, can dynamically change model.

Fixup

Performs dynamic transformations such as checksums and CRCs.

Hint

Provides information to mutators.

Placement

Relocates an element after it has been cracked.

Transformer

Performs static transformations such as compression or encoding.

Mutators

The following mutators operate on this element type.

Used for all data elements

DataElementDuplicate

This mutator duplicates data elements.

DataElementRemove

This mutator removes data elements.

DataElementSwapNear

This mutator swaps data elements.

SampleNinjaMutator

This mutator combines data elements from different data sets.

Enabled when element is part of a size relation

SizedDataEdgeCase

This mutator causes the data portion of a relation to be sized as numerical edge cases.

SizedDataVariance

This mutator causes the data portion of a relation to be sized as numerical variances.

SizedEdgeCase

This mutator changes both sides of the relation (data and value) to match numerical edge cases.

SizedVariance

This mutator changes both sides of the relation (data and value) to match numerical variances of the current size.

Specific to this element type

BlobChangeFromNull

This mutator produces test cases in which null bytes in a [Blob](#) element are changed to a non-null value.

BlobChangeRandom

This mutator produces test cases by changing random selections of bytes to random value.

BlobChangeSpecial

This mutator produces test cases by changing random selections of bytes to one of 0x00, 0x01, 0xFE, 0xFF.

BlobChangeToNull

This mutator produces test cases by changing a random number of bytes to 0x00.

BlobExpandSingleIncrementing

This mutator produces test cases by expanding the size of the blob using incrementing values.

BlobExpandAllRandom

This mutator produces test cases by expanding the size of the blob using random values.

BlobExpandSingleRandom

This mutator produces test cases by expanding the size of the blob using a single random byte (repeated as needed).

BlobExpandZero

This mutator produces test cases by expanding the blob using null values.

BlobReduce

This mutator produces test cases by reducing the size of the blob by a random amount.

ExtraValues

This mutator allows providing extra test case values on a per-data element basis.

Examples

Example 66. Defining a JsonBlob with a default value

A JsonBlob with a default value. Providing a default value does not set a fixed length unless the token="true" attribute is used.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="Ex1">
        <JsonObject>
            <JsonBlob propertyName="rawData" valueType="hex" value="AA BB CC DD" />
        </JsonObject>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="Ex1" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>

```

Output from this example.

```
>peach -1 --debug example.xml

[*] Web site running at: http://10.0.1.57:8888/

[*] Test 'Default' starting with random seed 4555.
Peach.Pro.Core.Loggers.JobLogger Writing debug.log to:
c:\peach\Logs\example.xml_20160223173145\debug.log

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "initial".
Peach.Core.Dom.Action Run(Action): Output
Peach.Pro.Core.Publishers.ConsolePublisher start()
Peach.Pro.Core.Publishers.ConsolePublisher open()
Peach.Pro.Core.Publishers.ConsolePublisher output(22 bytes)
00000000  7B 22 72 61 77 44 61 74  61 22 3A 22 71 72 76 4D  {"rawData":"qrvm
00000010  33 51 3D 3D 22 7D                           3Q=="}
Peach.Pro.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Pro.Core.Publishers.ConsolePublisher stop()
Peach.Core.Engine EndTest: Stopping all agents and monitors

[*] Test 'Default' finished.
```

21.3.13. JsonBool

The *JsonBool* element defines a Boolean value (values 0 or 1) that represents either true or false.

Syntax

```
<JsonObject>
    <JsonBool propertyName="enabled" value="1"/>
</JsonObject>
```

Attributes

Required:

No required attributes.

Optional:

name

Name of the boolean value.

isNull

Set default value of this element to null (defaults to false)

type

Type controls the output encoding allowing both traditional json and also binary json (bson). Supported values: *json* or *bson*. Defaults to *json*.

value

The default value to assign to the boolean. Valid options are integer values 0 or 1.

mutable

Is data element changeable (should it be mutated during fuzzing), defaults to true. Valid options true and false.

Child Elements

Hint

Provide information to mutators.

Placement

Relocate an element after it has been cracked.

Mutators

The following mutators will operate on this element type:

Used for all data elements

DataElementDuplicate

This mutator will duplicate data elements.

DataElementRemove

This mutator will remove data elements.

DataElementSwapNear

This mutator will swap data elements.

SampleNinjaMutator

This mutator will combine data elements from different data sets.

Enabled when element is part of a size relation

SizedDataEdgeCase

This mutator will cause the data portion of a relation to be sized as numerical edge cases.

SizedDataVariance

This mutator will cause the data portion of a relation to be sized as numerical variances.

SizedEdgeCase

This mutator will change both sides of the relation (data and value) to match numerical edge cases.

SizedVariance

This mutator will change both sides of the relation (data and value) to match numerical variances of the current size.

Examples

Example 67. Simple Example

Produce Json string with a boolean value of true.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">
  <DataModel name="BoolExample">
    <JsonObject>
      <JsonBool propertyName="enabled" value="1"/>
    </JsonObject>
  </DataModel>

  <StateModel name="TheState" initialState="Initial">
    <State name="Initial">
      <Action type="output">
        <DataModel ref="BoolExample"/>
      </Action>
    </State>
  </StateModel>

  <Agent name="TheAgent" />

  <Test name="Default">
    <Agent ref="TheAgent"/>

    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex"/>
  </Test>
</Peach>

```

Output from this example.

```
>peach -1 --debug BoolExample1.xml

[*] Web site running at: http://10.0.1.57:8888/

[*] Test 'Default' starting with random seed 65435.
Peach.Pro.Core.Loggers.JobLogger Writing debug.log to:
c:\peach\Logs\example.xml_20160223180045\debug.log

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "Initial".
Peach.Core.Dom.Action Run(Action): Output
Peach.Pro.Core.Publishers.ConsolePublisher start()
Peach.Pro.Core.Publishers.ConsolePublisher open()
Peach.Pro.Core.Publishers.ConsolePublisher output(16 bytes)
00000000  7B 22 65 6E 61 62 6C 65  64 22 3A 74 72 75 65 7D  {"enabled":true}
Peach.Pro.Core.Publishers.ConsolePublisher close()
Peach.Core.Agent.AgentManager DetectedFault: TheAgent
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Pro.Core.Publishers.ConsolePublisher stop()
Peach.Core.Engine EndTest: Stopping all agents and monitors

[*] Test 'Default' finished.
```

21.3.14. JsonDouble

The *JsonDouble* element defines a JSON floating-point number of up to 64-bits.

The *JsonDouble* element is a child element of [DataModel](#), [Block](#), [Sequence](#), [JsonObject](#), [JsonArray](#), or [Choice](#).

Syntax

```
<JsonObject>
    <JsonDouble propertyName="Price" value="1.99" />
</JsonObject>

<JsonArray>
    <JsonDouble value="1.99" />
</JsonArray>
```

Attributes

Required:

None.

Optional:

name

Name of the floating point number.

propertyName

Property name for element when child of [JsonObject](#).

isNull

Is this elements value null. Json output will be **null** instead of value.

type

Type controls the output encoding allowing both traditional json and also binary json (bson). Supported values: *json* or *bson*. Defaults to *json*.

value

The default value to assign to the floating point number.

valueType

The representation of the value. Valid options are string and hex.

token

This element is treated as a token when parsing, defaults to false. Valid options true and false.

mutable

Is data element changeable (should it be mutated during fuzzing), defaults to true. Valid options true and false.

Child Elements

Analyzer

Analyze current element post cracking, can dynamically change model.

Fixup

Dynamic transformations such as checksums and CRCs.

Hint

Provide information to mutators.

Placement

Relocate an element after it has been cracked.

Relation

Modeling of relationships in the data (such as comparisons)

Transformer

Static transformations such as compression or encoding.

Mutators

The following mutators will operate on this element type:

Used for all data elements

DataElementDuplicate

This mutator will duplicate data elements.

DataElementRemove

This mutator will remove data elements.

DataElementSwapNear

This mutator will swap data elements.

SampleNinjaMutator

This mutator will combine data elements from different data sets.

Enabled when element is part of a size relation

SizedDataEdgeCase

This mutator will cause the data portion of a relation to be sized as numerical edge cases.

SizedDataVariance

This mutator will cause the data portion of a relation to be sized as numerical variances.

SizedEdgeCase

This mutator will change both sides of the relation (data and value) to match numerical edge cases.

SizedVariance

This mutator will change both sides of the relation (data and value) to match numerical variances of the current size.

Specific to this element type

DoubleRandom

This mutator will produce random values from the available numerical space.

DoubleVariance

This mutator will produce values near the current value of a number.

ExtraValues

This mutator provides extra test case values on a per-data element basis.

NumberEdgeCase

This mutator produces numerical edge cases for integer values.

NumberRandom

This mutator produces random values from the available numerical space.

NumberVariance

This mutator produces values near the current value of a number.

Examples

Example 68. Simple Double

This example outputs a double that is part of a [JsonObject](#).

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <JsonObject>
            <JsonDouble propertyName="price" value="1.99" />
        </JsonObject>
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState" >
        <State name="InitialState">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>

        <Publisher class="ConsoleHex" />
    </Test>
</Peach>

```

Produces the following output:

```
> peach -1 --debug example.xml

[*] Web site running at: http://10.0.1.57:8888/

[*] Test 'Default' starting with random seed 48084.
Peach.Pro.Core.Loggers.JobLogger Writing debug.log to:
c:\peach\Logs\example.xml_20160215194649\debug.log

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "InitialState".
Peach.Core.Dom.Action Run(Action): Output
Peach.Pro.Core.Publishers.ConsolePublisher start()
Peach.Pro.Core.Publishers.ConsolePublisher open()
Peach.Pro.Core.Publishers.ConsolePublisher output(14 bytes)
00000000  7B 22 70 72 69 63 65 22  3A 31 2E 39 39 7D      {"price":1.99}
Peach.Pro.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Pro.Core.Publishers.ConsolePublisher stop()
Peach.Core.Engine EndTest: Stopping all agents and monitors

[*] Test 'Default' finished.
```

Example 69. Null String

In this example our initial value for our JsonString element is null. During testing this field's value will be mutated to string values.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <JsonObject>
            <JsonString propertyName="phrase"isNull="true" />
        </JsonObject>
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState" >
        <State name="InitialState">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>

        <Publisher class="ConsoleHex" />
    </Test>
</Peach>

```

Produces the following output:

```
> peach -1 --debug example.xml

[*] Web site running at: http://10.0.1.57:8888/

[*] Test 'Default' starting with random seed 29586.
Peach.Pro.Core.Loggers.JobLogger Writing debug.log to:
c:\peach\Logs\example.xml_20160215192237\debug.log

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "InitialState".
Peach.Core.Dom.Action Run(Action): Output
Peach.Pro.Core.Publishers.ConsolePublisher start()
Peach.Pro.Core.Publishers.ConsolePublisher open()
Peach.Pro.Core.Publishers.ConsolePublisher output(15 bytes)
00000000  7B 22 70 68 72 61 73 65  22 3A 6E 75 6C 6C 7D      {"phrase":null}
Peach.Pro.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Pro.Core.Publishers.ConsolePublisher stop()
Peach.Core.Engine EndTest: Stopping all agents and monitors

[*] Test 'Default' finished.
```

Example 70. String in JSONArray

This example outputs a string that is part of a [JSONArray](#). Note that we do not specify `propertyName` in this case.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <JSONArray>
            <JsonString value="Hello World!" />
        </JSONArray>
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState" >
        <State name="InitialState">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>

        <Publisher class="ConsoleHex" />
    </Test>
</Peach>

```

Produces the following output:

```
> peach -1 --debug example.xml

[*] Web site running at: http://10.0.1.57:8888/

[*] Test 'Default' starting with random seed 4074.
Peach.Pro.Core.Loggers.JobLogger Writing debug.log to:
c:\peach\Logs\example.xml_20160215192532\debug.log

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "InitialState".
Peach.Core.Dom.Action Run(Action): Output
Peach.Pro.Core.Publishers.ConsolePublisher start()
Peach.Pro.Core.Publishers.ConsolePublisher open()
Peach.Pro.Core.Publishers.ConsolePublisher output(16 bytes)
00000000  5B 22 48 65 6C 6C 6F 20  57 6F 72 6C 64 21 22 5D  ["Hello World!"]
Peach.Pro.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Pro.Core.Publishers.ConsolePublisher stop()
Peach.Core.Engine EndTest: Stopping all agents and monitors

[*] Test 'Default' finished.
```

21.3.15. JsonInteger

The JsonInteger element defines a binary number of any arbitrary bit size from 1 to 64. Binary numbers are packed into a byte representation with a byte order of little-endian or big-endian.

The Number element should not be used for character-based numbers, or ASCII numbers. Instead, use a [String](#) element.

The Number element is a child element of [DataModel](#), [Block](#), or [Choice](#).



While Peach supports unaligned data structures, using unaligned data incurs a performance penalty. The penalty stems from bit slicing that occurs behind the scenes.



In Peach, two attributes are commonly used to indicate size or length of an element, *size* and *length*. The *size* attribute always refers to the number of bits in an element. In contrast, the *length* attribute refers to the number of bytes of an element.

Syntax

```
<JsonObject>
    <JsonInteger propertyName="Almonds" value="100" />
</JsonObject>

<JsonInteger value="100" />
```

Attributes

Required:

None.

Optional:

name

Name of the number.

isNull

Is this elements value null. JSON output will be **null** instead of value.

type

Type controls the output encoding allowing both traditional json and also binary json (bson). Supported values: *json* or *bson*. Defaults to *json*.

value

The default value to assign to the number.

valueType

The representation of the value. Valid options are string and hex.

token

This element is treated as a token when parsing. Valid values are true and false, defaults to false.

mutable

Is data element changeable (should it be mutated during fuzzing). Valid values are true and false, defaults to true.

Child Elements

Analyzer

Analyzes current element post cracking, can dynamically change model.

Fixup

Performs dynamic transformations such as checksums and CRCs.

Hint

Provides information to mutators.

Placement

Relocates an element after it has been cracked.

Relation

Identifies a type of relationship with another data element (such as count).

Transformer

Performs static transformations such as compression or encoding.

Mutators

The following mutators operate on this element type:

Used for all data elements

DataElementDuplicate

This mutator duplicates data elements.

DataElementRemove

This mutator removes data elements.

DataElementSwapNear

This mutator swaps data elements.

SampleNinjaMutator

This mutator combines data elements from different data sets.

Enabled when element is part of a size relation

SizedDataEdgeCase

This mutator causes the data portion of a relation to be sized as numerical edge cases.

SizedDataVariance

This mutator causes the data portion of a relation to be sized as numerical variances.

SizedEdgeCase

This mutator changes both sides of the relation (data and value) to match numerical edge cases.

SizedVariance

This mutator changes both sides of the relation (data and value) to match numerical variances of the current size.

Specific to this element type

ExtraValues

This mutator provides extra test case values on a per-data element basis.

NumberEdgeCase

This mutator produces numerical edge cases for integer values.

NumberRandom

This mutator produces random values from the available numerical space.

NumberVariance

This mutator produces values near the current value of a number.

Examples

Example 71. Use in a JsonObject

Produce a 32-bit (4-byte) number with a default value of 5.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

    <DataModel name="NumberExample1">
        <JsonObject>
            <JsonInteger propertyName="Almonds" value="100" />
        </JsonObject>
    </DataModel>

    <StateModel name="TheState" initialState="Initial">
        <State name="Initial">
            <Action type="output">
                <DataModel ref="NumberExample1"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>

```

Output from this example.

```
>peach -1 --debug NumberExample1.xml

[*] Web site running at: http://10.0.1.57:8888/

[*] Test 'Default' starting with random seed 50669.
Peach.Pro.Core.Loggers.JobLogger Writing debug.log to:
c:\peach\Logs\example.xml_20160223173806\debug.log

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "Initial".
Peach.Core.Dom.Action Run(Action): Output
Peach.Pro.Core.Publishers.ConsolePublisher start()
Peach.Pro.Core.Publishers.ConsolePublisher open()
Peach.Pro.Core.Publishers.ConsolePublisher output(15 bytes)
00000000 7B 22 41 6C 6D 6F 6E 64 73 22 3A 31 30 30 7D      {"Almonds":100}
Peach.Pro.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Pro.Core.Publishers.ConsolePublisher stop()
Peach.Core.Engine EndTest: Stopping all agents and monitors

[*] Test 'Default' finished.
```

21.3.16. JsonObject

The `JsonObject` element is used to model [JavaScript Object Notation \(JSON\)](#) objects.

`JsonObject` elements are containers that group JSON elements in a logical structure. For example, a `JsonString` element, a `JsonInteger` element, and a `JsonBlob` element can exist in the same `JsonObject`. A `JsonObject` can contain other `JsonObjects`, or nest, as deeply as needed.

Syntax

```
<JsonObject name="ServiceRequest">

    <JsonString propertyName="id" value="3fcbbc99-bb05-432d-8ce5-9df4fab91ae6" />

    <JsonArray propertyName="items">

        <JsonObject>
            <JsonString propertyName="id" value="5813200e-96db-4dd0-bc2a-fec6d19a7242" />
            <JsonDouble propertyName="price" value="1.99" />
            <JsonInteger propertyName="amount" value="10" />
        </JsonObject>

        <JsonObject>
            <JsonString propertyName="id" value="a921833c-5e3f-4199-a143-3ae60b6815ef" />
            <JsonDouble propertyName="price" value="2.99" />
            <JsonInteger propertyName="amount" value="1" />
        </JsonObject>

    </JsonArray>

</JsonObject>
```

Attributes

Required:

None.

Optional:

name

Name of the block.

isNull

Is this elements value null. Json output will be `null` instead of value.

type

Type controls the output encoding allowing both traditional json and also binary json (bson). Supported values: *json* or *bson*. Defaults to *json*.

ref

Reference to a [DataModel](#) to use as a template.

mutable

Is data element changeable (should it be mutated), defaults to false.

Child Elements

[Analyzer](#)

Analyzes current element post cracking, can dynamically change model.

[Choice](#)

Indicates that all of the sub-elements are valid; but, only one sub-element should be selected.

[Fixup](#)

Are dynamic transformations such as checksums and CRCs.

[Hint](#)

Provides information to mutators.

[JsonArray](#)

Json array

[JsonBool](#)

Json boolean

[JsonBlob](#)

Json blob

[JsonDouble](#)

Json double/float

[JsonInteger](#)

Json integer

[JsonObject](#)

Json object

[JsonString](#)

Json string

Placement

Relocates an element after it has been cracked.

Transformer

Performs static transformations such as compression or encoding.

Mutators

The following mutators operate on this element type:

Used for all data elements

DataElementDuplicate

This mutator duplicates data elements.

DataElementRemove

This mutator removes data elements.

DataElementSwapNear

This mutator swaps data elements.

SampleNinjaMutator

This mutator combines data elements from different data sets.

Enabled when element is part of a size relation

SizedDataEdgeCase

This mutator causes the data portion of a relation to be sized as numerical edge cases.

SizedDataVariance

This mutator causes the data portion of a relation to be sized as numerical variances.

SizedEdgeCase

This mutator changes both sides of the relation (data and value) to match numerical edge cases.

SizedVariance

This mutator changes both sides of the relation (data and value) to match numerical variances of the current size.

Examples

Example 72. Example JsonObject

Example of generating a json object with an array.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

<DataModel name="BlockExample1">
  <JsonObject name="ServiceRequest">

    <JsonString propertyName="id" value="3fcbbc99-bb05-432d-8ce5-9df4fab91ae6" />

    <JsonArray propertyName="items">

      <JsonObject>
        <JsonString propertyName="id" value="5813200e-96db-4dd0-bc2a-
fec6d19a7242" />
        <JsonDouble propertyName="price" value="1.99" />
        <JsonInteger propertyName="amount" value="10" />
      </JsonObject>

      <JsonObject>
        <JsonString propertyName="id" value="a921833c-5e3f-4199-a143-
3ae60b6815ef" />
        <JsonDouble propertyName="price" value="2.99" />
        <JsonInteger propertyName="amount" value="1" />
      </JsonObject>

    </JsonArray>
  </JsonObject>
</DataModel>

<StateModel name="TheState" initialState="initial">
  <State name="initial">
    <Action type="output" publisher="ConsolePub">
      <DataModel ref="BlockExample1" />
    </Action>
  </State>
</StateModel>

<Test name="Default">
  <StateModel ref="TheState"/>

  <Publisher class="ConsoleHex" name="ConsolePub"/>
</Test>
</Peach>

```

Output from this example.

```
>peach -1 --debug example.xml

[*] Web site running at: http://10.0.1.57:8888/

[*] Test 'Default' starting with random seed 29941.
Peach.Pro.Core.Loggers.JobLogger Writing debug.log to:
c:\peach\Logs\example.xml_20160223174712\debug.log

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "initial".
Peach.Core.Dom.Action Run(Action): Output
Peach.Pro.Core.Publishers.ConsolePublisher start()
Peach.Pro.Core.Publishers.ConsolePublisher open()
Peach.Pro.Core.Publishers.ConsolePublisher output(196 bytes)
00000000  7B 22 69 64 22 3A 22 33  66 63 62 62 63 39 39 2D  {"id":"3fcbbc99-
00000010  62 62 30 35 2D 34 33 32  64 2D 38 63 65 35 2D 39  bb05-432d-8ce5-9
00000020  64 66 34 66 61 62 39 31  61 65 36 22 2C 22 69 74  df4fab91ae6","it
00000030  65 6D 73 22 3A 5B 7B 22  69 64 22 3A 22 35 38 31  ems": [{"id":"581
00000040  33 32 30 30 65 2D 39 36  64 62 2D 34 64 64 30 2D  3200e-96db-4dd0-
00000050  62 63 32 61 2D 66 65 63  36 64 31 39 61 37 32 34  bc2a-fec6d19a724
00000060  32 22 2C 22 70 72 69 63  65 22 3A 31 2E 39 39 2C  2","price":1.99,
00000070  22 61 6D 6F 75 6E 74 22  3A 31 30 7D 2C 7B 22 69  "amount":10}, {"i
00000080  64 22 3A 22 61 39 32 31  38 33 33 63 2D 35 65 33  d":"a921833c-5e3
00000090  66 2D 34 31 39 39 2D 61  31 34 33 2D 33 61 65 36  f-4199-a143-3ae6
000000A0  30 62 36 38 31 35 65 66  22 2C 22 70 72 69 63 65  0b6815ef","price
000000B0  22 3A 32 2E 39 39 2C 22  61 6D 6F 75 6E 74 22 3A  ":"2.99,"amount":1}]]}
000000C0  31 7D 5D 7D

Peach.Pro.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Pro.Core.Publishers.ConsolePublisher stop()
Peach.Core.Engine EndTest: Stopping all agents and monitors

[*] Test 'Default' finished.
```

21.3.17. JsonRaw

The *JsonRaw* element is used to embed raw string or binary data into a JSON document. This allows creation of json that does not adhere to the JSON specification.

This is a container element that is able to host other data elements.

Syntax

```
<JsonObject name="ServiceRequest">

    <JsonString propertyName="id" value="3fcbbc99-bb05-432d-8ce5-9df4fab91ae6" />

    <JsonRaw propertyName="binary">
        <Blob value="01 02 03 04 05" valueType="hex"/>
    </JsonRaw>

</JsonObject>
```

Attributes

Required:

None.

Optional:

name

Name of the block.

isNull

Is this elements value null. Json output will be **null** instead of value.

ref

Reference to a [DataModel](#) to use as a template.

mutable

Is data element changeable (should it be mutated), defaults to false.

Child Elements

all data elements supported by Block are supported by JsonRaw

Analyzer

Analyzes current element post cracking, can dynamically change model.

Fixup

Are dynamic transformations such as checksums and CRCs.

Hint

Provides information to mutators.

Placement

Relocates an element after it has been cracked.

Mutators

The following mutators operate on this element type:

Used for all data elements

DataElementDuplicate

This mutator duplicates data elements.

DataElementRemove

This mutator removes data elements.

DataElementSwapNear

This mutator swaps data elements.

SampleNinjaMutator

This mutator combines data elements from different data sets.

Enabled when element is part of a size relation

SizedDataEdgeCase

This mutator causes the data portion of a relation to be sized as numerical edge cases.

SizedDataVariance

This mutator causes the data portion of a relation to be sized as numerical variances.

SizedEdgeCase

This mutator changes both sides of the relation (data and value) to match numerical edge cases.

SizedVariance

This mutator changes both sides of the relation (data and value) to match numerical variances of the current size.

Examples

Example 73. Example JsonRaw

Example of using JsonRaw to output non-standard JSON document, including binary values.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach>
  <DataModel name="TheDataModel">

    <JsonObject>
      <JsonArray propertyName="arrayWithRaw">
        <JsonRaw>
          <Blob value="rawvalue"/>
        </JsonRaw>
      </JsonArray>

      <JsonRaw propertyName="raw">
        <Blob value="01 02 03 04 05 06 07 08" valueType="hex"/>
      </JsonRaw>
    </JsonObject>

    <JsonObject>
      <JsonRaw propertyName="rawNull"isNull="true">
        <Blob value="rawvalue"/>
      </JsonRaw>
    </JsonObject>

  </DataModel>

  <StateModel name="TheState" initialState="initial">
    <State name="initial">
      <Action type="output">
        <DataModel ref="TheDataModel" />
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheState"/>
    <Publisher class="ConsoleHex"/>
  </Test>
</Peach>
```

Output from this example.

```
>peach -1 --debug example.xml

[[ Peach Pro v0.0.0.1
[[ Copyright (c) 2017 Peach Fuzzer, LLC

[*] Web site running at: http://10.0.1.113:8888/
2017-08-09 12:45:30.4826 Peach.Pro.Core.MutationStrategies.RandomStrategy Initialized
with seed 29202

[*] Test 'Default' starting with random seed 29202.
2017-08-09 12:45:30.5538 Peach.Pro.Core.Loggers.JobLogger Writing debug.log to:
c:\peach\Logs\example.xml_20170809124529\debug.log

[R1,-,-] Performing iteration
2017-08-09 12:45:30.6610 Peach.Core.Engine runTest: Performing control recording
iteration.
2017-08-09 12:45:30.7042 Peach.Core.Dom.StateModel Run(): Changing to state
"initial".
2017-08-09 12:45:30.7112 Peach.Core.Dom.Action Run(Action): Output
2017-08-09 12:45:30.8125 Peach.Pro.Core.Publishers.ConsolePublisher start()
2017-08-09 12:45:30.8125 Peach.Pro.Core.Publishers.ConsolePublisher open()
2017-08-09 12:45:30.8125 Peach.Pro.Core.Publishers.ConsolePublisher output(58 bytes)
00000000  7B 22 61 72 72 61 79 57  69 74 68 52 61 77 22 3A  {"arrayWithRaw"::
00000010  5B 72 61 77 76 61 6C 75  65 5D 2C 22 72 61 77 22  [rawvalue],"raw"
00000020  3A 01 02 03 04 05 06 07  08 7D 7B 22 72 61 77 4E  :.....}{"rawN
00000030  75 6C 6C 22 3A 6E 75 6C  6C 7D                      ull":null}
2017-08-09 12:45:30.8224 Peach.Pro.Core.Publishers.ConsolePublisher close()
2017-08-09 12:45:30.8224 Peach.Core.Engine runTest: context.config.singleIteration ==
true
2017-08-09 12:45:30.8224 Peach.Core.Engine All test cases executed, stopping engine.
2017-08-09 12:45:30.8224 Peach.Pro.Core.Publishers.ConsolePublisher stop()
2017-08-09 12:45:30.8224 Peach.Core.Engine EndTest: Stopping all agents and monitors

[*] Test 'Default' finished.
```

21.3.18. JsonString

Represents a JSON string of Unicode characters.

All strings are encoded prior to output according to the JSON specification.

Strings can hold numbers that are stored in a string format. When strings contain a number, Peach additionally uses numerical mutators to mutate the strings.

The `JsonString` element is a child element of [DataModel](#), [Block](#), [Sequence](#), [JsonObject](#), [JsonArray](#), or [Choice](#).

Syntax

```
<JsonString value="Hello World!" />

<JsonObject>
    <JsonString propertyName="name" value="Peach" />
</JsonObject>
```

Attributes

Required:

None.

Optional:

name

Name of the element.

propertyName

Property name for element when child of [JsonObject](#).

isNull

Is this elements value null. Json output will be `null` instead of `" "` or value.

type

Type controls the output encoding allowing both traditional json and also binary json (bson). Supported values: *json* or *bson*. Defaults to *json*.

value

The default value, defaults to `""`.

valueType

Format of the *value* attribute, defaults to string.

token

This element should be treated as a token when parsing. Valid values are true and false, defaults to false. + This attribute is primarily used to assist in cracking strings when consuming input in a model.

mutable

Should this data element be mutated (or, is it changeable)? Valid values are true and false, defaults to true.

Child Elements

Analyzer

Attaches an analyzer to this element

Fixup

Performs dynamic transformations such as checksums and CRCs.

Hint

Provides information to mutators.

Placement

Relocates an element after it has been cracked.

Relation

Identifies a type of relationship with another data element (such as count).

Transformer

Performs static transformations such as compression or encoding.

Mutators

The following mutators operate on this element type:

Used for all data elements

DataElementDuplicate

This mutator duplicates data elements.

DataElementRemove

This mutator removes data elements.

DataElementSwapNear

This mutator swaps data elements.

SampleNinjaMutator

This mutator combines data elements from different data sets.

Enabled when element is part of a size relation

SizedDataEdgeCase

This mutator causes the data portion of a relation to be sized as numerical edge cases.

SizedDataVariance

This mutator causes the data portion of a relation to be sized as numerical variances.

SizedEdgeCase

This mutator changes both sides of the relation (data and value) to match numerical edge cases.

SizedVariance

This mutator changes both sides of the relation (data and value) to match numerical variances of the current size.

Specific to this element type

ExtraValues

This mutator provides extra test case values on a per-data element basis.

StringAsciiRandom

This mutator generates strings with random ASCII characters.

StringCaseLower

This mutator generates a lower case version of the current value.

StringCaseRandom

This mutator generates a randomized case version of the current value.

StringCaseUpper

This mutator generates an upper case version of the current value.

StringLengthEdgeCase

This mutator generates strings with lengths based on numerical edge cases.

StringLengthVariance

This mutator generates strings with lengths based on a variance around the current string length.

StringList

This mutator allows providing a list of strings to use as test cases on an element by element basis.

StringStatic

This mutator generates test cases using a static set of strings.

StringUnicodeAbstractCharacters

This mutator generates Unicode strings using abstract characters.

StringUnicodeFormatCharacters

This mutator generates Unicode strings using format characters.

StringUnicodeInvalid

This mutator generates Unicode strings using invalid characters.

StringUnicodeNonCharacters

This mutator generates Unicode strings using non-characters.

StringUnicodePlane0

This mutator generates Unicode strings using Plane 0 characters.

StringUnicodePlane1

This mutator generates Unicode strings using Plane 1 characters.

StringUnicodePlane14

This mutator generates Unicode strings using Plan 14 characters.

StringUnicodePlane15And16

This mutator generates Unicode strings using Plane 15 and 16 characters.

StringUnicodePlane2

This mutator generates Unicode strings using Plane 2 characters.

StringUnicodePrivateUseArea

This mutator generates Unicode characters from the private use area.

StringXmlW3C

This mutator provides the W3C XML parser unit tests. Must be specifically enabled.

Examples

Example 74. Simple String

This example outputs a string that is part of a [JsonObject](#).

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <JsonObject>
            <JsonString propertyName="phrase" value="Hello World!" />
        </JsonObject>
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState" >
        <State name="InitialState">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>

        <Publisher class="ConsoleHex" />
    </Test>
</Peach>

```

Produces the following output:

```
> peach -1 --debug example.xml

[*] Web site running at: http://10.0.1.57:8888/

[*] Test 'Default' starting with random seed 51346.
Peach.Pro.Core.Loggers.JobLogger Writing debug.log to:
c:\peach\Logs\example.xml_20160215191651\debug.log

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "InitialState".
Peach.Core.Dom.Action Run(Action): Output
Peach.Pro.Core.Publishers.ConsolePublisher start()
Peach.Pro.Core.Publishers.ConsolePublisher open()
Peach.Pro.Core.Publishers.ConsolePublisher output(25 bytes)
00000000  7B 22 70 68 72 61 73 65  22 3A 22 48 65 6C 6C 6F {"phrase":"Hello
00000010  20 57 6F 72 6C 64 21 22  7D                     World!"}
Peach.Pro.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Pro.Core.Publishers.ConsolePublisher stop()
Peach.Core.Engine EndTest: Stopping all agents and monitors

[*] Test 'Default' finished.
```

Example 75. Null String

In this example our initial value for our jsonString element is null. During testing this field's value will be mutated to string values.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <JsonObject>
            <JsonString propertyName="phrase"isNull="true" />
        </JsonObject>
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState" >
        <State name="InitialState">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>

        <Publisher class="ConsoleHex" />
    </Test>
</Peach>

```

Produces the following output:

```
> peach -1 --debug example.xml

[*] Web site running at: http://10.0.1.57:8888/

[*] Test 'Default' starting with random seed 29586.
Peach.Pro.Core.Loggers.JobLogger Writing debug.log to:
c:\peach\Logs\example.xml_20160215192237\debug.log

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "InitialState".
Peach.Core.Dom.Action Run(Action): Output
Peach.Pro.Core.Publishers.ConsolePublisher start()
Peach.Pro.Core.Publishers.ConsolePublisher open()
Peach.Pro.Core.Publishers.ConsolePublisher output(15 bytes)
00000000  7B 22 70 68 72 61 73 65  22 3A 6E 75 6C 6C 7D      {"phrase":null}
Peach.Pro.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Pro.Core.Publishers.ConsolePublisher stop()
Peach.Core.Engine EndTest: Stopping all agents and monitors

[*] Test 'Default' finished.
```

Example 76. String in JSONArray

This example outputs a string that is part of a [JSONArray](#). Note that we do not specify `propertyName` in this case.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <JSONArray>
            <JsonString value="Hello World!" />
        </JSONArray>
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState" >
        <State name="InitialState">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>

        <Publisher class="ConsoleHex" />
    </Test>
</Peach>

```

Produces the following output:

```
> peach -1 --debug example.xml

[*] Web site running at: http://10.0.1.57:8888/

[*] Test 'Default' starting with random seed 4074.
Peach.Pro.Core.Loggers.JobLogger Writing debug.log to:
c:\peach\example.xml_20160215192532\debug.log

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "InitialState".
Peach.Core.Dom.Action Run(Action): Output
Peach.Pro.Core.Publishers.ConsolePublisher start()
Peach.Pro.Core.Publishers.ConsolePublisher open()
Peach.Pro.Core.Publishers.ConsolePublisher output(16 bytes)
00000000  5B 22 48 65 6C 6C 6F 20  57 6F 72 6C 64 21 22 5D  ["Hello World!"]
Peach.Pro.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Pro.Core.Publishers.ConsolePublisher stop()
Peach.Core.Engine EndTest: Stopping all agents and monitors

[*] Test 'Default' finished.
```

21.3.19. Null

The *Null* element defines an element which has no value.

The Null element is a child element of [DataModel](#), [Block](#), [Sequence](#) or [Choice](#).

Syntax

```
<Null name="null"/>
```

Attributes

Required:

No required attributes.

Optional:

name

Name of the Null value.

constraint

A constraint in the form of a python expression. Used during data cracking.

mutable

Is data element changeable (should it be mutated during fuzzing), defaults to true. Valid options true and false.

minOccurs

The minimum number of times this number must occur. Defaults to 1. Valid options are a positive integer value.

maxOccurs

The maximum number of times this number can occur. Defaults to 1. Valid options are a positive integer value.

occurs

The actual number of times this number occurs. Defaults to 1.

Child Elements

Placement

Relocate an element after it has been cracked.

Mutators

The following mutators will operate on this element type:

Enabled when element is marked as an array

ArrayNumericalEdgeCasesMutator

This mutator will grow and shrink an array to counts based on numerical edge cases.

ArrayRandomizeOrderMutator

This mutator will randomize the order of items in an array.

ArrayReverseOrderMutator

This mutator will reverse the order of items in an array.

ArrayVarianceMutator

This mutator will grow and shrink an array to a variance of counts based on the current size.

Used for all data elements

DataElementBitFlipper

This mutator will produce test cases by flipping bits in the output value.

DataElementDuplicate

This mutator will duplicate data elements.

DataElementRemove

This mutator will remove data elements.

DataElementSwapNear

This mutator will swap data elements.

SampleNinjaMutator

This mutator will combine data elements from different data sets.

Examples

Example 77. Json Null

Outputs a Json string with a single null value.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">
  <DataModel name="NullExample">
    <Json>
      <Null name="Null"/>
    </Json>
  </DataModel>

  <StateModel name="TheState" initialState="Initial">
    <State name="Initial">
      <Action type="output">
        <DataModel ref="NullExample"/>
      </Action>
    </State>
  </StateModel>

  <Agent name="TheAgent" />

  <Test name="Default">
    <Agent ref="TheAgent"/>

    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex"/>
  </Test>
</Peach>

```

Output from this example.

```
>peach -1 --debug NullExample1.xml

[*] Web site running at: http://localhost:8888/

[*] Test 'Default' starting with random seed 24442.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "Initial".
Peach.Core.Dom.Action Run(Action): Output
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(13 bytes)
00000000  7B 22 4E 75 6C 6C 22 3A  6E 75 6C 6C 7D          {"Null":null}
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

21.3.20. Number

The Number element defines a binary number of any arbitrary bit size from 1 to 64. Binary numbers are packed into a byte representation with a byte order of little-endian or big-endian.

The Number element should not be used for character-based numbers, or ASCII numbers. Instead, use a [String](#) element.

The Number element is a child element of [DataModel](#), [Block](#), or [Choice](#).

See also [VarNumber](#) and [String](#).



While Peach supports unaligned data structures, using unaligned data incurs a performance penalty. The penalty stems from bit slicing that occurs behind the scenes.



In Peach, two attributes are commonly used to indicate size or length of an element, *size* and *length*. The *size* attribute always refers to the number of bits in an element. In contrast, the *length* attribute refers to the number of bytes of an element.

Syntax

```
<Number name="Almonds" size="32" endian="big" signed="false" />
```

Attributes

Required:

size

Size of number in bits. Valid options are 1 through 64.

Optional:

name

Name of the number.

value

The default value to assign to the number.

valueType

The representation of the value. Valid options are string and hex.

token

This element is treated as a token when parsing. Valid values are true and false, defaults to false.

endian

Byte order of the number. Valid options are big, little, and network. Network is an alias for big. The default value is little.

signed

The number is signed or unsigned. Valid values are true and false, defaults to false.

constraint

A constraint in the form of a python expression. Used during data cracking.

mutable

Is data element changeable (should it be mutated during fuzzing). Valid values are true and false, defaults to true.

minOccurs

The minimum number of times this element must occur. Defaults to 1.

- Used to define arrays with variable size. Arrays defined by min/maxOccurs generally have a relation defined.
- Adding this attribute, even with a value of 1, converts the element to an array.

maxOccurs

The maximum number of times this element can occur.

- Used to define arrays with static size.
- Adding this attribute, even with a value of 1, converts the element to an array.

occurs

Actual occurrences of element. Defaults to 1.

- Used to define arrays with static size.
- Adding this attribute, even with a value of 1, converts the element to an array.

Child Elements

Analyzer

Analyzes current element post cracking, can dynamically change model.

Fixup

Performs dynamic transformations such as checksums and CRCs.

Hint

Provides information to mutators.

Placement

Relocates an element after it has been cracked.

Relation

Identifies a type of relationship with another data element (such as count).

Transformer

Performs static transformations such as compression or encoding.

Mutators

The following mutators operate on this element type:

Enabled when element is marked as an array

ArrayNumericalEdgeCasesMutator

This mutator grows and shrinks an array to counts based on numerical edge cases.

ArrayRandomizeOrderMutator

This mutator randomizes the order of items in an array.

ArrayReverseOrderMutator

This mutator reverses the order of items in an array.

ArrayVarianceMutator

This mutator grows and shrinks an array to a variance of counts based on the current size.

Used for all data elements

DataElementBitFlipper

This mutator produces test cases by flipping bits in the output value.

DataElementDuplicate

This mutator duplicates data elements.

DataElementRemove

This mutator removes data elements.

DataElementSwapNear

This mutator swaps data elements.

SampleNinjaMutator

This mutator combines data elements from different data sets.

Enabled when element is part of a size relation

SizedDataEdgeCase

This mutator causes the data portion of a relation to be sized as numerical edge cases.

SizedDataVariance

This mutator causes the data portion of a relation to be sized as numerical variances.

SizedEdgeCase

This mutator changes both sides of the relation (data and value) to match numerical edge cases.

SizedVariance

This mutator changes both sides of the relation (data and value) to match numerical variances of the current size.

Specific to this element type

ExtraValues

This mutator provides extra test case values on a per-data element basis.

NumberEdgeCase

This mutator produces numerical edge cases for integer values.

NumberRandom

This mutator produces random values from the available numerical space.

NumberVariance

This mutator produces values near the current value of a number.

Examples

Example 78. Size

Produce a 32-bit (4-byte) number with a default value of 5.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">
    <DataModel name="NumberExample1">
        <Number name="H15" value="5" size="32"/>
    </DataModel>

    <StateModel name="TheState" initialState="Initial">
        <State name="Initial">
            <Action type="output">
                <DataModel ref="NumberExample1"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="TheAgent" />

    <Test name="Default">
        <Agent ref="TheAgent"/>

        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>

```

Output from this example.

```
>peach -1 --debug NumberExample1.xml

[*] Test 'Default' starting with random seed 6226.
Peach.Core.MutationStrategies.RandomStrategy Iteration: Switch iteration, setting
controlIteration and controlRecordingIteration.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Updating action to original data model
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(4 bytes)
00000000  05 00 00 00                                ????
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

① The 32-bit, little-endian value is 5.

Change the previous example to use a 16-bit (two-byte) number by adjusting the size to 16.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

    <DataModel name="NumberExample2">
        <Number name="Hi5" value="5" size="16"/>
    </DataModel>

    <StateModel name="TheState" initialState="Initial">
        <State name="Initial">
            <Action type="output">
                <DataModel ref="NumberExample2"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="TheAgent" />

    <Test name="Default">
        <Agent ref="TheAgent"/>

        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>

```

Output from this example.

```

>peach -1 --debug NumberExample2.xml

[*] Test 'Default' starting with random seed 51118.
Peach.Core.MutationStrategies.RandomStrategy Iteration: Switch iteration, setting
controlIteration and controlRecordingIteration.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Updating action to original data model
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(2 bytes)
00000000  05 00                                ??          ①
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

① The 16 bit little endian value 5

NOTE: Numbers use the **size** attribute which specifies the number of **bits**. *Number* elements do not accept the **length** attribute used by other elements.

Example 79. Byte Alignment

While many data structures are byte aligned, some are not. This example arbitrarily defines sizes that do not fall on byte boundaries.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

    <DataModel name="ByteAlignmentExample1">
        <Number value="2" size="3" />
        <Number value="12" size="5" />
    </DataModel>

    <StateModel name="TheState" initialState="Initial">
        <State name="Initial">
            <Action type="output">
                <DataModel ref="ByteAlignmentExample1"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="TheAgent" />
    <Test name="Default">
        <Agent ref="TheAgent"/>

        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>

```

Output from this example.

```

>peach -1 --debug ByteAlignExample.xml

[*] Test 'Default' starting with random seed 41464.
Peach.Core.MutationStrategies.RandomStrategy Iteration: Switch iteration, setting
controlIteration and controlRecordingIteration.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Updating action to original data model
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(1 bytes)
00000000  4C                                L          ①
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

```

① Two numbers are compacted into one byte.

The first number is three bits with the value "2". This number becomes the first three bits of the byte. The remaining five bits are appended to the left.

In Python this could be written as the following:

```

>>> hex((2 << 5) + 12)
'0x4c'

```

Input parsing is simply the inverse.

```

>>> input_byte = 0x4C
>>> offset = 5
>>> (input_byte >> offset)
2                                              ①
>>> input_byte & (-1 + (2**offset))
12                                             ②

```

① The first byte is bit-shifted using the size of the second.

② By masking off the first number, we get the second.

Example 80. Endian

To change the endian-ness of the number, set the endian attribute. Endian-ness defines the order of the least-significant or most-significant bytes.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

  <DataModel name="NumberExample6">
    <Number name="abcd" value="52651" size="16" signed="false" endian="big" />
  </DataModel>

  <DataModel name="NumberExample7">
    <Number name="abcd" value="52651" size="16" signed="false" endian="little" />
  </DataModel>

  <StateModel name="TheState" initialState="Initial">
    <State name="Initial">
      <Action type="output">
        <DataModel ref="NumberExample6"/>
      </Action>
      <Action type="output">
        <DataModel ref="NumberExample7"/>
      </Action>
    </State>
  </StateModel>

  <Agent name="TheAgent" />

  <Test name="Default">
    <Agent ref="TheAgent"/>

    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex"/>
  </Test>
</Peach>
```

Output from this example.

```

>peach -1 --debug NumberEndianExample.xml

[*] Test 'Default' starting with random seed 16220.
Peach.Core.MutationStrategies.RandomStrategy Iteration: Switch iteration, setting
controlIteration and controlRecordingIteration.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Updating action to original data model
Peach.Core.Dom.Action Updating action to original data model
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(2 bytes)
00000000  CD AB                                ??          ①
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(2 bytes)
00000000  AB CD                                ??          ②
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

① Little endian outputs the bytes in the order CD AB

② Big endian outputs the bytes in the order AB CD

Note, however, that endian-ness doesn't have any impact on output if the **valueType** is "hex":

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

    <DataModel name="NumberExample6">
        <Number name="abcd" valueType="hex" value="ABCD" size="16" signed="false"
endian="little" />
    </DataModel>

    <DataModel name="NumberExample7">
        <Number name="abcd" valueType="hex" value="ABCD" size="16" signed="false"
endian="big" />
    </DataModel>

    <StateModel name="TheState" initialState="Initial">
        <State name="Initial">
            <Action type="output">
                <DataModel ref="NumberExample6"/>
            </Action>
            <Action type="output">
                <DataModel ref="NumberExample7"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="TheAgent" />

    <Test name="Default">
        <Agent ref="TheAgent"/>

        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>
```

Output from this example.

```

>peach -1 --debug NumberEndianExample.xml

[*] Test 'Default' starting with random seed 37516.
Peach.Core.MutationStrategies.RandomStrategy Iteration: Switch iteration, setting
controlIteration and controlRecordingIteration.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Updating action to original data model
Peach.Core.Dom.Action Updating action to original data model
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(2 bytes)
00000000 AB CD ?? ①
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(2 bytes)
00000000 AB CD ?? ②
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

① For little endian, the expected output displays.

② For big endian, nothing changes.

When the attribute `valueType` is set to "hex", the ordering is exactly as specified. The `Endian` attribute still impacts mutation and input parsing.

Example 81. Signed and Unsigned

To indicate value is signed, set the `signed` attribute equal to "true". The default is false.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

    <DataModel name="UnsignedExample">
        <Number name="UnsignedInt" value="4294967295" size="32"/>
    </DataModel>

    <DataModel name="SignedExample">
        <Number name="SignedInt" value="-2147483648" size="32" signed="true"/>
    </DataModel>

    <StateModel name="TheState" initialState="Initial">
        <State name="Initial">
            <Action type="output">
                <DataModel ref="UnsignedExample"/>
            </Action>
            <Action type="output">
                <DataModel ref="SignedExample"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="TheAgent" />

    <Test name="Default">
        <Agent ref="TheAgent"/>

        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>
```

Output from this example.

```

>peach -1 --debug NumberExample3.xml

[*] Test 'Default' starting with random seed 64304.
Peach.Core.MutationStrategies.RandomStrategy Iteration: Switch iteration, setting
controlIteration and controlRecordingIteration.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Updating action to original data model
Peach.Core.Dom.Action Updating action to original data model
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(4 bytes)
00000000  FF FF FF FF          ????? ①
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(4 bytes)
00000000  FF FF FF FF          ????? ②
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

① Output of unsigned 4294967295

② Output of signed -1

Example 82. Value Type

The valueType defines how to interpret the value attribute. Valid options are "string" and "hex". The default value is *string*.

To assign a value of 1000 to MyValue, use the default `valueType` of "string". The "string" type supports both decimal and hexadecimal values.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

    <DataModel name="NumberTypeExample1">
        <Number name="MyValue" value="1000" valueType="string" size="16" signed=
"false" />
    </DataModel>

    <DataModel name="NumberTypeExample2">
        <Number name="MyValue" value="0x03e8" valueType="string" size="16" signed=
"false" />
    </DataModel>

    <StateModel name="TheState" initialState="Initial">
        <State name="Initial">
            <Action type="output">
                <DataModel ref="NumberTypeExample1"/>
            </Action>
            <Action type="output">
                <DataModel ref="NumberTypeExample2"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="TheAgent" />

    <Test name="Default">
        <Agent ref="TheAgent"/>

        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>
```

Output from this example.

```

>peach -1 --debug NumberExample4.xml

[*] Test 'Default' starting with random seed 61690.
Peach.Core.MutationStrategies.RandomStrategy Iteration: Switch iteration, setting
controlIteration and controlRecordingIteration.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Updating action to original data model
Peach.Core.Dom.Action Updating action to original data model
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(2 bytes)
00000000 E8 03                                ??          ①
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(2 bytes)
00000000 E8 03                                ??          ②
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

① The value 1000

② The value 1000, having been defined in hex as 0x03e8

Notice that the `valueType` of "string" represents a number. This number may be changed by endian-ness. Observe that the numeric value was entered as 0x03e8, but Peach output the bytes 0xE8 and 0x03. The value was converted to little endian before being output.

To assign a value as if copied directly from a hex editor we can use the "hex" `valueType`. Values entered in "hex" are output exactly as input regardless of endian-ness, as shown in the previous example that exercises the `Endian` attribute.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

    <DataModel name="NumberExample5">
        <Number name="MyValue" value="AB CD" valueType="hex" size="16" signed="false"
    />
    </DataModel>

    <StateModel name="TheState" initialState="Initial">
        <State name="Initial">
            <Action type="output">
                <DataModel ref="NumberExample5"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="TheAgent" />

    <Test name="Default">
        <Agent ref="TheAgent"/>

        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>

```

Output from this example.

```
>peach -1 --debug NumberExample5.xml

[*] Test 'Default' starting with random seed 55408.
Peach.Core.MutationStrategies.RandomStrategy Iteration: Switch iteration, setting
controlIteration and controlRecordingIteration.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Updating action to original data model
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(2 bytes)
00000000 AB CD ?? ①
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

① The value is 43981.

21.3.21. Padding

The *Padding* element provides padding for variable-sized blocks or data models.

When padding the data, the value of the pad characters is always 0. This is not customizable.

When cracking data into a model that contains the padding element, the pad data is skipped.

Syntax

```
<DataModel name="NumberExample1">
  <String name="VariableSizeString" />

  <Padding />
</DataModel>
```

Attributes

Required:

None.

Optional:

name

Element name.

alignment

Align to the specified bit boundary: 8, 16, 32, 64, etc. The default value is 8

alignedTo

Name of reference element for alignment, defaults to parent element. The alignment starts at the beginning of the reference element

mutable

Is the data element changeable (should it be mutated during fuzzing), defaults to true. Valid options true and false.

Child Elements

Fixup

Performs dynamic transformations such as checksums and CRCs.

Hint

Provides information to mutators

Transformer

Performs static transformations such as compression or encoding.

Mutators

The following mutators can operate on this element type:

Enabled when the data element is marked as an array

ArrayNumericalEdgeCasesMutator

This mutator grows and shrinks an array to counts based on numerical edge cases.

ArrayRandomizeOrderMutator

This mutator randomizes the order of items in an array.

ArrayReverseOrderMutator

This mutator reverses the sequence of items in an array.

ArrayVarianceMutator

This mutator grows and shrinks an array to a variance of counts based on the current size.

Used for all data elements

DataElementBitFlipper

This mutator produces test cases by flipping bits in the output value.

DataElementDuplicate

This mutator duplicates data elements.

DataElementRemove

This mutator removes data elements.

DataElementSwapNear

This mutator swaps data elements.

SampleNinjaMutator

This mutator combines data elements from different data sets.

Enabled when the data element is part of a size relation

SizedDataEdgeCase

This mutator causes the data portion of a relation to be sized as numerical edge cases.

SizedDataVariance

This mutator causes the data portion of a relation to be sized as numerical variances.

SizedEdgeCase

This mutator changes both sides of the relation (data and value) to match numerical edge cases.

SizedVariance

This mutator changes both sides of the relation (data and value) to match numerical variances of the current size.

Examples

Example 83. Padding with Default Options

This example demonstrates the default case of padding. The AlignTo attribute aligns pad characters along 8-bit boundaries.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <Number size="1" value="0" />
        <Number size="8" value="0xff" />

        <Padding />
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState" >
        <State name="InitialState">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>

        <Publisher class="ConsoleHex" />
    </Test>
</Peach>
```

Produces the following output:

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 56742.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(2 bytes)
00000000  7F 80          .?
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

Example 84. Padding Aligned to 32 bit Boundary

This example aligns the pad data to a 32-bit boundary.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <Number size="1" value="0" />
        <Number size="8" value="0xff" />

        <Padding alignment="32" />
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState" >
        <State name="InitialState">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>

        <Publisher class="ConsoleHex" />
    </Test>
</Peach>

```

Produces the following output:

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 51106.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(4 bytes)
00000000  7F 80 00 00                                .???
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

Example 85. Padding Separated from Unaligned Value

This example demonstrates padding with unaligned data preceding and following the padding.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <Number name="ToBeAligned" size="1" value="0" />

        <Number size="16" value="42" />
        <Number size="2" value="1" />

        <Padding alignment="32" alignedTo="ToBeAligned" />

        <Number size="6" value="42" />
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState" >
        <State name="InitialState">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>

        <Publisher class="ConsoleHex" />
    </Test>
</Peach>

```

Produces the following output:

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 49630.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(7 bytes)
00000000  15 00 20 00 00 00 2A                      ?? ???
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

21.3.22. Sequence

A *Sequence* element is used to construct an array with all its elements predefined. A *Sequence* is able to contain multiple data types at once. Unlike regular arrays in Peach, the size of the *Sequence* is determined by the number of elements within the sequence.



A *Sequence* element that uses the Occurs, minOccurs, or maxOccurs attribute creates an array of *Sequences*.

Syntax

```
<Sequence>
  <String name="Foo" value="Hello world!" />
  <Number name="Num" value="101" />
</Sequence>
```

Attributes

Required:

There are no required attributes.

Optional:

name

Name of the sequence.

ref

Reference to a [DataModel](#) to use as a template.

length

Data element length.

lengthType

The unit measure of length attribute. Default is bytes.

constraint

Scripting expression that evaluates to true or false. Default is null.

minOccurs

The minimum number of times this sequence must occur.

maxOccurs

The maximum number of times this sequence can occur.

occurs

The actual number of times this sequence occurs.

mutable

Is data element changeable (should it be mutated), defaults to false.

Child Elements

Analyzer

Analyze current element post cracking, can dynamically change model.

Blob

Used to represent binary data (think array of bytes) to create simple dumb fuzzers in Peach.

Block

Group one or more data elements together into a logical structure.

Bool

Defines a boolean value.

Choice

Indicate any of the sub-elements are valid but only one should be selected.

Double

Defines a floating point number of 32 or 64 bits.

Fixup

Dynamic transformations such as checksums and CRCs.

Flags

Defines a set of bit sized flags.

Hint

Provide information to mutators.

Null

Defines a null value element.

Number

Defines a binary number of arbitrary bit size.

Padding

Pad out variably sized blocks or data models.

Placement

Relocate an element after it has been cracked.

Sequence

Groups one or more data elements together into a logical structure.

Stream

Group one or more data elements together into a logical structure.

Transformer

Static transformations such as compression or encoding.

XmlElement

Defines an XML element, the basic building block of XML documents.

Mutators

The following mutators will operate on this element type:

Enabled when element is marked as an array

ArrayNumericalEdgeCasesMutator

This mutator will grow and shrink an array to counts based on numerical edge cases.

ArrayRandomizeOrderMutator

This mutator will randomize the order of items in an array.

ArrayReverseOrderMutator

This mutator will reverse the order of items in an array.

ArrayVarianceMutator

This mutator will grow and shrink an array to a variance of counts based on the current size.

Used for all data elements

DataElementBitFlipper

This mutator will produce test cases by flipping bits in the output value.

DataElementDuplicate

This mutator will duplicate data elements.

DataElementRemove

This mutator will remove data elements.

DataElementSwapNear

This mutator will swap data elements.

SampleNinjaMutator

This mutator will combine data elements from different data sets.

Enabled when element is part of a size relation

SizedDataEdgeCase

This mutator will cause the data portion of a relation to be sized as numerical edge cases.

SizedDataVariance

This mutator will cause the data portion of a relation to be sized as numerical variances.

SizedEdgeCase

This mutator will change both sides of the relation (data and value) to match numerical edge cases.

SizedVariance

This mutator will change both sides of the relation (data and value) to match numerical variances of the current size.

Examples

Example 86. Empty Sequence

The simplest sequence has no children.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">
  <DataModel name="SequenceExample">
    <Sequence>

      </Sequence>
    </DataModel>

    <StateModel name="TheState" initialState="Initial">
      <State name="Initial">
        <Action type="output">
          <DataModel ref="SequenceExample"/>
        </Action>
      </State>
    </StateModel>

    <Agent name="TheAgent" />

    <Test name="Default">
      <Agent ref="TheAgent"/>

      <StateModel ref="TheState"/>

      <Publisher class="ConsoleHex"/>
    </Test>
  </Peach>

```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 3808.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "Initial".
Peach.Core.Dom.Action Run(Action): Output
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(2 bytes)
00000000  7B 7D                                {}
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

Example 87. Single Child

The output of a sequence with a single child

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">
  <DataModel name="SequenceExample">
    <Sequence>
      <Number name="num" size="8" value="101"/>
    </Sequence>
  </DataModel>

  <StateModel name="TheState" initialState="Initial">
    <State name="Initial">
      <Action type="output">
        <DataModel ref="SequenceExample"/>
      </Action>
    </State>
  </StateModel>

  <Agent name="TheAgent" />

  <Test name="Default">
    <Agent ref="TheAgent"/>

    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex"/>
  </Test>
</Peach>

```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 40441.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "Initial".
Peach.Core.Dom.Action Run(Action): Output
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(1 bytes)
00000000  65                                     e
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

Example 88. Naming A Sequence

Assign sequences a friendly name to make them easier to understand and debug.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">
  <DataModel name="SequenceExample">
    <Sequence name="MySeq">
      <Number name="num" size="8" value="101"/>
    </Sequence>
  </DataModel>

  <StateModel name="TheState" initialState="Initial">
    <State name="Initial">
      <Action type="output">
        <DataModel ref="SequenceExample"/>
      </Action>
    </State>
  </StateModel>

  <Agent name="TheAgent" />

  <Test name="Default">
    <Agent ref="TheAgent"/>

    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex"/>
  </Test>
</Peach>

```

Output from this example.

```
>peach -l --debug example.xml

[*] Test 'Default' starting with random seed 30169.

[*] Test 'Default' starting with random seed 37527.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "Initial".
Peach.Core.Dom.Action Run(Action): Output
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(1 bytes)
00000000 65                                     e
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

Example 89. Nested Sequences

Sequences can be nested as deep as required. Sequences help create logical structure and do not change the data contained within.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

  <DataModel name="SequenceExample">
    <Sequence>
      <Sequence>
        <Sequence>
          <String value="1" />
        </Sequence>

        <Sequence>
          <String value="2" />
        </Sequence>

        <String value="3" />
      </Sequence>
      <String value="4" />
    </Sequence>
  </DataModel>

  <StateModel name="TheState" initialState="initial">
    <State name="initial">
      <Action type="output" publisher="ConsolePub">
        <DataModel ref="SequenceExample" />
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex" name="ConsolePub"/>
  </Test>
</Peach>

```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 27277.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "initial".
Peach.Core.Dom.Action Run(Action): Output
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(4 bytes)
00000000  31 32 33 34                               1234
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

Example 90. Altering Sequence Values using Data Fields

Sequences values can be altered from the state model using data fields.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">
  <DataModel name="SequenceExample">
    <Sequence name="MySeq">
      <String name="val1" value="Old"/>
      <String name="val2" value="OLDER"/>
    </Sequence>
  </DataModel>

  <StateModel name="TheState" initialState="Initial">
    <State name="Initial">
      <Action type="output">
        <DataModel ref="SequenceExample"/>
        <Data>
          <Field name="MySeq[0]" value="Updated Value for val1"/>
        </Data>
      </Action>
    </State>
  </StateModel>

  <Agent name="TheAgent" />

  <Test name="Default">
    <Agent ref="TheAgent"/>

    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex"/>
  </Test>
</Peach>

```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 24392.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "Initial".
Peach.Core.Dom.Action Run(Action): Output
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(27 bytes)
00000000  55 70 64 61 74 65 64 20  56 61 6C 75 65 20 66 6F  Updated Value fo
00000010  72 20 76 61 6C 31 4F 4C  44 45 52                  r val10LDER
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

21.3.23. Stream

Streams are logical blocks of data that contain other data blocks (such as zip and audio/video [A/V] multimedia files). *Stream* elements are used to represent streams in a single model.

Like [Block](#) elements, *Stream* elements allow groupings of one or more data elements in a logical structure, in this case to represent the content of the stream.

Stream elements have two additional pieces of metadata that *Blocks* lack, a stream name and a stream attribute. Stream-aware publishers can use the metadata and content to combine all streams in a data model into a single file. A single data model representing a zip file would use the *Stream* element to represent each file that makes up the zip file.

Stream is a child element of either [DataModel](#) or [Block](#).

The [Zip](#) publisher looks for Stream elements and creates a zip archive of all the streams in a model.



Zip is the only Peach-supplied publisher aware of *Stream* elements. If you wish to fuzz an A/V file, you will need to create your own publisher.

Syntax

```
<Stream streamName="TheStream">
  <String value="Hello world!" />
</Block>
```

Attributes

Required:

streamName

Name of the underlying stream

Optional:

name

Name of the stream element.

ref

Reference to a [DataModel_Stream](#) to use as a template.

length

Data element length.

lengthType

The unit of measure for the length attribute. The default value is bytes.

constraint

Scripting expression that evaluates to true or false. Default is null.

minOccurs

The minimum number of times this element must occur. Defaults to 1.

- Used to define arrays with variable size. Arrays defined by min/maxOccurs generally have a relation defined.
- Adding this attribute, even with a value of 1, converts the element to an array.

maxOccurs

The maximum number of times this element can occur. Defaults to 1.

- Used to define arrays with static size.
- Adding this attribute, even with a value of 1, converts the element to an array.

occurs

Actual occurrences of element. Defaults to 1.

- Used to define arrays with static size.
- Adding this attribute, even with a value of 1, converts the element to an array.

mutable

Is this data element changeable (should it be mutated), defaults to false.

streamAttribute

Integer representing any underlying attribute of a stream, defaults to 0.

Child Elements

Analyzer

Analyzes current element post cracking, can dynamically change model.

Blob

Represents binary data (array of bytes) to create simple dumb fuzzers in Peach.

Block

Groups one or more data elements in a logical structure.

Choice

Indicates all sub-elements are valid; but only one sub-element should be selected.

Fixup

Perform dynamic transformations such as checksums and CRCs.

Flags

Defines a set of bit sized flags.

Hint

Provides information to mutators.

Number

Defines a binary number of arbitrary bit size.

Padding

Pads variably sized blocks or data models for uniformity and consistency.

Placement

Relocates an element after it has been cracked.

Transformer

Performs static transformations such as compression or encoding.

XmlElement

Defines an XML element, the basic building block of XML documents.

Mutators

The following mutators operate on this element type:

Enabled when element is marked as an array

ArrayNumericalEdgeCasesMutator

This mutator grows and shrinks an array to counts based on numerical edge cases.

ArrayRandomizeOrderMutator

This mutator randomizes the order of items in an array.

ArrayReverseOrderMutator

This mutator reverses the order of items in an array.

ArrayVarianceMutator

This mutator grows and shrinks an array to a variance of counts based on the current size.

Used for all data elements

DataElementBitFlipper

This mutator produces test cases by flipping bits in the output value.

DataElementDuplicate

This mutator duplicates data elements.

DataElementRemove

This mutator removes data elements.

DataElementSwapNear

This mutator swaps data elements.

SampleNinjaMutator

This mutator combines data elements from different data sets.

Enabled when element is part of a size relation

SizedDataEdgeCase

This mutator causes the data portion of a relation to be sized as numerical edge cases.

SizedDataVariance

This mutator causes the data portion of a relation to be sized as numerical variances.

SizedEdgeCase

This mutator changes both sides of the relation (data and value) to match numerical edge cases.

SizedVariance

This mutator changes both sides of the relation (data and value) to match numerical variances of the current size.

Background

The Stream element is very similar to the [Block](#) element, except the Stream element includes two pieces of additional data: the streamName and the streamAttribute.

Conceptually, the following Stream and Block examples are very similar to one another.

Stream example:

```
<Stream name="TheStream" streamName="file1.txt" streamAttribute="100">
  <String value="Hello World"/>
  <Transformer class="Base64Encode" />
</Stream>
```

Block example (compare this with the previous Stream definition):

```
<Block name="TheStream">
  <String name="Name" value="file1.txt"/>
  <Number name="Attribute" size="32" signed="false" value="100"/>
  <Block name="Content">
    <String value="Hello World"/>
    <Transformer class="Base64Encode"/>
  </Block>
</Block>
```

The stream name, attribute and children all support fuzzing.

Additionally, relations and fixups can reference children of different streams.

The Stream element is intended for use by publishers that are stream aware (such as [Zip](#)). If the publisher is not stream aware, the stream element is treated exactly like a [Block](#).

Examples

Example 91. Stream with Zip publisher

The following definition produces a zip file containing a single entry *file1.txt* containing the string *Hello World*.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="StreamExample1">
        <Stream streamName="file1.txt">
            <String value="Hello World"/>
        </Stream>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="StreamExample1" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="Zip">
            <Param name="FileName" value="fuzzed.zip" />
        </Publisher>
    </Test>
</Peach>

```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 59388.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Enterprise.Publishers.ZipPublisher start()
Peach.Enterprise.Publishers.ZipPublisher open()
Peach.Enterprise.Publishers.ZipPublisher Added 1 entries to zip file.
Peach.Enterprise.Publishers.ZipPublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Enterprise.Publishers.ZipPublisher stop()

[*] Test 'Default' finished.
```

Example 92. Stream with [ConsoleHex publisher](#)

Streams are treated like blocks when used with publishers that are not stream aware.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="StreamExample2">
        <Stream streamName="file1.txt">
            <String value="Hello World"/>
        </Stream>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output" publisher="ConsolePub">
                <DataModel ref="StreamExample2" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex" name="ConsolePub"/>
    </Test>
</Peach>

```

Output from this example.

```

>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 30169.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(11 bytes)
00000000  48 65 6C 6C 6F 20 57 6F  72 6C 64          Hello World
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

Example 93. Multiple streams

Produce a zip file containing multiple files.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

<DataModel name="StreamExample3">
  <Stream streamName="file1.txt">
    <String value="Root file one"/>
  </Stream>
  <Stream streamName="dir/file1.txt">
    <String value="File one in subdirectory"/>
  </Stream>
  <Stream streamName="dir/file2.txt">
    <String value="File two in subdirectory"/>
  </Stream>
</DataModel>

<StateModel name="TheState" initialState="initial">
  <State name="initial">
    <Action type="output">
      <DataModel ref="StreamExample3" />
    </Action>
  </State>
</StateModel>

<Test name="Default">
  <StateModel ref="TheState"/>

  <Publisher class="Zip">
    <Param name="FileName" value="fuzzed.zip" />
  </Publisher>
</Test>
</Peach>
```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 58326.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Enterprise.Publishers.ZipPublisher start()
Peach.Enterprise.Publishers.ZipPublisher open()
Peach.Enterprise.Publishers.ZipPublisher Added 3 entries to zip file.
Peach.Enterprise.Publishers.ZipPublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Enterprise.Publishers.ZipPublisher stop()

[*] Test 'Default' finished.
```

Contents of produced **fuzzed.zip**

```
> unzip -l fuzzed.zip
Archive: fuzzed.zip
      Length      Date      Time    Name
----- -----
      13  04-09-2014 18:14  file1.txt
      24  04-09-2014 18:14  dir/file1.txt
      24  04-09-2014 18:14  dir/file2.txt
-----
                           -----
      61                           3 files
```

21.3.24. String

A string containing ASCII or Unicode characters. Strings can be null terminated, padded, with or without a fixed length.

All strings are encoded prior to output with the default 7-bit ASCII encoding (aka Latin-1). A number of different Unicode encodings are also supported (see the *type* attribute).

Strings can hold numbers that are stored in a string format. When strings contain a number, Peach additionally uses numerical mutators to mutate the strings.

Syntax

```
<String value="Hello World!" />  
<String value="Null terminated string" nullTerminated="true" />
```

Attributes

Required:

None.

Optional:

name

Name of the string.

length

Length of the string, typically expressed as bytes. The default length is unspecified. The units of measure for length are specified in the *lengthType* attribute.

lengthType

Units of measure of the length attribute. Valid values are bits-in multiples of 8, bytes, and chars. The default value is bytes.

type

Character encoding type, defaults to *utf8*. Valid options are: ascii, utf7, utf8, utf16, utf16be, utf32. + Before strings are output, they are first encoded. The default encoding is utf-8. Most common ASCII and Unicode encodings are supported.

value

The default value, defaults to "".

valueType

Format of the *value* attribute, defaults to string.

nullTerminated

Specifies whether this string is null terminated. Valid values are true and false, defaults to false.

The *nullTerminated* attribute indicates the produced string should be null terminated.



Only the final value has the null terminator applied. If the value is accessed, the internal value will not have the null attached.



When the *nullTerminated* attribute is provided, the [String](#) must not specify a *length*; these two attributes are mutually exclusive.

padCharacter

Character to use as padding at the end of the string, defaults to null (0x00).

If the *length* attribute is provided and the value contained in the string is shorter than the length, the string is padded to fill the length. This attribute allows controlling the character used to pad the string.

token

This element should be treated as a token when parsing. Valid values are true and false, defaults to false. + This attribute is primarily used to assist in cracking strings when consuming input in a model.

constraint

Scripting expression that evaluates to true or false. The default value is "", meaning that a constraint is not defined.

This attribute is used exclusively during data cracking and has two main uses:

- Provides switch-like processing when combined with the [Choice](#) element;
- Controls array expansion.



This expression does not control or affect mutated values in anyway.

mutable

Should this data element be mutated (or, is it changeable)? Valid values are true and false, defaults to true.

minOccurs

The minimum number of times this element must occur. Defaults to 1.

- Used to define arrays with variable size. Arrays defined by min/maxOccurs generally have a relation defined.
- Adding this attribute, even with a value of 1, converts the element to an array.

maxOccurs

The maximum number of times this element can occur. Defaults to 1.

- Used to define arrays with static size.
- Adding this attribute, even with a value of 1, converts the element to an array.

occurs

Actual occurrences of element. Defaults to 1.

- Used to define arrays with static size.
- Adding this attribute, even with a value of 1, converts the element to an array.

Child Elements

Analyzer

Attaches an analyzer to this element

Fixup

Performs dynamic transformations such as checksums and CRCs.

Hint

Provides information to mutators.

Placement

Relocates an element after it has been cracked.

Relation

Identifies a type of relationship with another data element (such as count).

Transformer

Performs static transformations such as compression or encoding.

Mutators

The following mutators operate on this element type:

Enabled when element is marked as an array

ArrayNumericalEdgeCasesMutator

This mutator grows and shrinks an array to counts based on numerical edge cases.

ArrayRandomizeOrderMutator

This mutator randomizes the order of items in an array.

ArrayReverseOrderMutator

This mutator reverses the order of items in an array.

ArrayVarianceMutator

This mutator grows and shrinks an array to a variance of counts based on the current size.

Used for all data elements

DataElementBitFlipper

This mutator produces test cases by flipping bits in the output value.

DataElementDuplicate

This mutator duplicates data elements.

DataElementRemove

This mutator removes data elements.

DataElementSwapNear

This mutator swaps data elements.

SampleNinjaMutator

This mutator combines data elements from different data sets.

Enabled when element is part of a size relation

SizedDataEdgeCase

This mutator causes the data portion of a relation to be sized as numerical edge cases.

SizedDataVariance

This mutator causes the data portion of a relation to be sized as numerical variances.

SizedEdgeCase

This mutator changes both sides of the relation (data and value) to match numerical edge cases.

SizedVariance

This mutator changes both sides of the relation (data and value) to match numerical variances of the current size.

Specific to this element type

ExtraValues

This mutator provides extra test case values on a per-data element basis.

StringAsciiRandom

This mutator generates strings with random ASCII characters.

StringCaseLower

This mutator generates a lower case version of the current value.

StringCaseRandom

This mutator generates a randomized case version of the current value.

StringCaseUpper

This mutator generates an upper case version of the current value.

StringLengthEdgeCase

This mutator generates strings with lengths based on numerical edge cases.

StringLengthVariance

This mutator generates strings with lengths based on a variance around the current string length.

StringList

This mutator allows providing a list of strings to use as test cases on an element by element basis.

StringStatic

This mutator generates test cases using a static set of strings.

StringUnicodeAbstractCharacters

This mutator generates Unicode strings using abstract characters.

StringUnicodeFormatCharacters

This mutator generates Unicode strings using format characters.

StringUnicodeInvalid

This mutator generates Unicode strings using invalid characters.

StringUnicodeNonCharacters

This mutator generates Unicode strings using non-characters.

StringUnicodePlane0

This mutator generates Unicode strings using Plane 0 characters.

StringUnicodePlane1

This mutator generates Unicode strings using Plane 1 characters.

StringUnicodePlane14

This mutator generates Unicode strings using Plan 14 characters.

StringUnicodePlane15And16

This mutator generates Unicode strings using Plane 15 and 16 characters.

[StringUnicodePlane2](#)

This mutator generates Unicode strings using Plane 2 characters.

[StringUnicodePrivateUseArea](#)

This mutator generates Unicode characters from the private use area.

[StringXmlW3C](#)

This mutator provides the W3C XML parser unit tests. Must be specifically enabled.

Examples

Example 94. Simple ASCII String

This example outputs a string with minimal attribute declarations. Other than the *value* attribute, the string uses default values for its attributes. The string consists of 7-bit ASCII characters.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String value="Hello World!" />
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState" >
        <State name="InitialState">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>

        <Publisher class="ConsoleHex" />
    </Test>
</Peach>
```

Produces the following output:

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 25723.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(12 bytes)
00000000  48 65 6C 6C 6F 20 57 6F  72 6C 64 21          Hello World!
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

Example 95. String with UTF-16 Encoding

This example outputs a string consisting of characters represented with Unicode UTF-16 encoding. UTF-16 is a two-byte character encoding that supports Latin and non-Latin character sets. Also, UTF-16 is the WCHAR type on the Windows operating systems.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String type="utf16" value="Hello World!" />
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState" >
        <State name="InitialState">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>

        <Publisher class="ConsoleHex" />
    </Test>
</Peach>

```

Produces the following output:

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 57920.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(24 bytes)
00000000  48 00 65 00 6C 00 6C 00  6F 00 20 00 57 00 6F 00  H?e?l?l?o? ?W?o?
00000010  72 00 6C 00 64 00 21 00
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

Example 96. Null Terminated String

This example outputs a null-terminated string.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String nullTerminated="true" value="Hello World!" />
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState" >
        <State name="InitialState">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>

        <Publisher class="ConsoleHex" />
    </Test>
</Peach>
```

Produces the following output:

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 53517.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(13 bytes)
00000000  48 65 6C 6C 6F 20 57 6F  72 6C 64 21 00          Hello World!?
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

Example 97. String Padded with Default Character

This example outputs a string that is shorter than the required length. The string receives one or more pad characters to reach its required length. The default pad character is null (0x00).

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String length="20" value="Hello World!" />
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState" >
        <State name="InitialState">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>

        <Publisher class="ConsoleHex" />
    </Test>
</Peach>

```

Produces the following output:

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 43832.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(20 bytes)
00000000  48 65 6C 6C 6F 20 57 6F  72 6C 64 21 00 00 00 00  Hello World!?????
00000010  00 00 00 00                                     ****
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

Example 98. String Padded with Specified Character

This example output a string that is shorter than the required length. The string receives one or more pad characters to reach its required length. Unlike the earlier example, we define the pad character as _, thus overriding the default pad character.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="TheDataModel">
    <String length="20" padCharacter="_" value="Hello World!" />
</DataModel>

<StateModel name="TheStateModel" initialState="InitialState" >
    <State name="InitialState">
        <Action type="output">
            <DataModel ref="TheDataModel"/>
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheStateModel"/>

        <Publisher class="ConsoleHex" />
    </Test>
</Peach>
```

Produces the following output:

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 62597.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(20 bytes)
00000000  48 65 6C 6C 6F 20 57 6F  72 6C 64 21 5F 5F 5F 5F  Hello World!_____
00000010  5F 5F 5F 5F
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

Example 99. String with Backslash Characters

This example outputs a string that contains carriage return and line feed characters using the `\r` and `\n` notation. Also, the output string includes a backslash character (`\`).

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String value="This is the first line\nAnd this is the second line\n\rThis is
backslash \\." />
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState" >
        <State name="InitialState">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>

        <Publisher class="Console" />
    </Test>
</Peach>

```

Produces the following output:

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 29966.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(72 bytes)
This is the first line
And this is the second line
This is backslash \.Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

Example 100. String with Size-Of Relation

In this example, a the DataModel declares a String element that contains the ASCII length of some data. A size-of relation is used to allow the size to dynamically update during fuzzing.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="TheDataModel">
    <String name="Length">
        <Relation type="size" of="Data" />
    </String>

    <String value="\n" />

    <Block name="Data">
        <String value="This is some data!" />
        <String value=" And this is even more data!" />
    </Block>
</DataModel>

<StateModel name="TheStateModel" initialState="InitialState" >
    <State name="InitialState">
        <Action type="output">
            <DataModel ref="TheDataModel"/>
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheStateModel"/>

    <Publisher class="Console" />
</Test>
</Peach>
```

Produces the following output:

```
> peach -l --debug example.xml

[*] Test 'Default' starting with random seed 2887.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(49 bytes)
46
This is some data! And this is even more data!Peach.Core.Publishers.ConsolePubli
sher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

Example 101. Using token Attribute to Crack Strings

In this example, Peach parses a simple text string using the *token* attribute. This example uses two files, a file containing sample data called `string.txt` and the pit file `example.xml`. The sample string to parse consists of three parts: 1)the key, 2)the token separator, and 3)a value.

Potentially, the key and value can be any arbitrary size, so cracking this sample string requires knowledge about the token separator. In the pit file, the string that is used as a token includes the *token* attribute to indicate that the token must be present in the incoming data stream. This allows the data cracker to figure out the length of both the key and the value.



Peach fuzzes elements marked as *token*.

10. `string.txt`

```
Content-length: 10
```

11. example.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

  <DataModel name="TheDataModel">
    <String name="Key"/>

    <String value=":" token="true" />

    <String name="Value"/>
  </DataModel>

  <StateModel name="TheStateModel" initialState="InitialState" >
    <State name="InitialState">
      <Action type="output">
        <DataModel ref="TheDataModel"/>
        <Data fileName="string.txt" />
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheStateModel"/>

    <Publisher class="Console" />
  </Test>
</Peach>
```

Produces the following output:

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 18622.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'TheDataModel' Bytes: 0/21, Bits: 0/168

Peach.Core.Cracker.DataCracker getSize: -----> DataModel 'TheDataModel'
Peach.Core.Cracker.DataCracker scan: DataModel 'TheDataModel'
Peach.Core.Cracker.DataCracker scan: String 'TheDataModel.Key' -> Offset: 0, Unsized
element
```

```
Peach.Core.Cracker.getSize: <----- Deterministic: ???  
Peach.Core.Cracker Crack: DataModel 'TheDataModel' Size: <null>, Bytes:  
0/21, Bits: 0/168  
Peach.Core.Cracker -----  
Peach.Core.Cracker.String 'TheDataModel.Key' Bytes: 0/21, Bits: 0/168  
Peach.Core.Cracker.getSize: -----> String 'TheDataModel.Key'  
Peach.Core.Cracker.scan: String 'TheDataModel.Key' -> Offset: 0, Unsized  
element  
Peach.Core.Cracker.lookahead: String 'TheDataModel.Key'  
Peach.Core.Cracker.scan: String 'TheDataModel.DataElement_0' -> Pos: 0,  
Saving Token  
Peach.Core.Cracker.scan: String 'TheDataModel.DataElement_0' -> Pos: 8,  
Length: 8  
Peach.Core.Cracker.scan: String 'TheDataModel.Value' -> Offset: 8,  
Unsized element  
Peach.Core.Cracker.getSize: <----- Required Token: 112  
Peach.Core.Cracker.Crack: String 'TheDataModel.Key' Size: 112, Bytes:  
0/21, Bits: 0/168  
Peach.Core.Dom.DataElement String 'TheDataModel.Key' value is: Content-length ①  
Peach.Core.Cracker -----  
Peach.Core.Cracker.String 'TheDataModel.DataElement_0' Bytes: 14/21,  
Bits: 112/168  
Peach.Core.Cracker.getSize: -----> String 'TheDataModel.DataElement_0'  
Peach.Core.Cracker.scan: String 'TheDataModel.DataElement_0' -> Pos: 0,  
Saving Token  
Peach.Core.Cracker.scan: String 'TheDataModel.DataElement_0' -> Pos: 8,  
Length: 8  
Peach.Core.Cracker.getSize: <----- Size: 8  
Peach.Core.Cracker.Crack: String 'TheDataModel.DataElement_0' Size: 8,  
Bytes: 14/21, Bits: 112/168  
Peach.Core.Dom.DataElement String 'TheDataModel.DataElement_0' value is: : ②  
Peach.Core.Cracker -----  
Peach.Core.Cracker.String 'TheDataModel.Value' Bytes: 15/21, Bits:  
120/168  
Peach.Core.Cracker.getSize: -----> String 'TheDataModel.Value'  
Peach.Core.Cracker.scan: String 'TheDataModel.Value' -> Offset: 0,  
Unsized element  
Peach.Core.Cracker.lookahead: String 'TheDataModel.Value'  
Peach.Core.Cracker.getSize: <----- Last Unsized: 48  
Peach.Core.Cracker.Crack: String 'TheDataModel.Value' Size: 48, Bytes:  
15/21, Bits: 120/168  
Peach.Core.Dom.DataElement String 'TheDataModel.Value' value is: 10 ③  
  
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted  
Peach.Core.Dom.Action ActionType.Output  
Peach.Core.Publishers.ConsolePublisher start()  
Peach.Core.Publishers.ConsolePublisher open()  
Peach.Core.Publishers.ConsolePublisher output(21 bytes)
```

```
Content-length: 10
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

- ① Cracking "Content-length" into *Key*
- ② Cracking ":" into token string
- ③ Cracking "10\n" into *Value*

21.3.25. VarNumber

The *VarNumber* element defines a variable width binary number of any arbitrary bit size from 1 to 64. Binary numbers are packed into a byte representation with a byte order of little-endian or big-endian.

VarNumber is similar to [Number](#) except the size is not hard coded and instead is driven by a method such as a size-of relation or limit of available data.

The *VarNumber* element should not be used for character-based numbers, or ASCII numbers. Instead, use a [String](#) element.

See also [Number](#).



While Peach supports unaligned data structures, using unaligned data incurs a performance penalty. The penalty stems from bit slicing that occurs behind the scenes.

Syntax

```
<Number name="SizeOfAlmondsNumber" size="4" endian="big" signed="false">
    <Relation type="size" of="Almonds" />
</Number>

<VarNumber name="Almonds" endian="big" signed="false" />
```

Attributes

Required:

No required attributes.

Optional:

[name](#)

Name of the number.

[value](#)

The default value to assign to the number.

[valueType](#)

The representation of the value. Valid options are string and hex.

[token](#)

This element is treated as a token when parsing. Valid values are true and false, defaults to false.

endian

Byte order of the number. Valid options are big, little, and network. Network is an alias for big. The default value is little.

signed

The number is signed or unsigned. Valid values are true and false, defaults to false.

constraint

A constraint in the form of a python expression. Used during data cracking.

mutable

Is data element changeable (should it be mutated during fuzzing). Valid values are true and false, defaults to true.

minOccurs

The minimum number of times this element must occur. Defaults to 1.

- Used to define arrays with variable size. Arrays defined by min/maxOccurs generally have a relation defined.
- Adding this attribute, even with a value of 1, converts the element to an array.

maxOccurs

The maximum number of times this element can occur.

- Used to define arrays with static size.
- Adding this attribute, even with a value of 1, converts the element to an array.

occurs

Actual occurrences of element. Defaults to 1.

- Used to define arrays with static size.
- Adding this attribute, even with a value of 1, converts the element to an array.

Child Elements

Analyzer

Analyzes current element post cracking, can dynamically change model.

Fixup

Performs dynamic transformations such as checksums and CRCs.

Hint

Provides information to mutators.

Placement

Relocates an element after it has been cracked.

Relation

Identifies a type of relationship with another data element (such as count).

Transformer

Performs static transformations such as compression or encoding.

Mutators

The following mutators operate on this element type:

Enabled when element is marked as an array

ArrayNumericalEdgeCasesMutator

This mutator grows and shrinks an array to counts based on numerical edge cases.

ArrayRandomizeOrderMutator

This mutator randomizes the order of items in an array.

ArrayReverseOrderMutator

This mutator reverses the order of items in an array.

ArrayVarianceMutator

This mutator grows and shrinks an array to a variance of counts based on the current size.

Used for all data elements

DataElementBitFlipper

This mutator produces test cases by flipping bits in the output value.

DataElementDuplicate

This mutator duplicates data elements.

DataElementRemove

This mutator removes data elements.

DataElementSwapNear

This mutator swaps data elements.

SampleNinjaMutator

This mutator combines data elements from different data sets.

Enabled when element is part of a size relation

SizedDataEdgeCase

This mutator causes the data portion of a relation to be sized as numerical edge cases.

SizedDataVariance

This mutator causes the data portion of a relation to be sized as numerical variances.

SizedEdgeCase

This mutator changes both sides of the relation (data and value) to match numerical edge cases.

SizedVariance

This mutator changes both sides of the relation (data and value) to match numerical variances of the current size.

Specific to this element type

ExtraValues

This mutator provides extra test case values on a per-data element basis.

NumberEdgeCase

This mutator produces numerical edge cases for integer values.

NumberRandom

This mutator produces random values from the available numerical space.

NumberVariance

This mutator produces values near the current value of a number.

Examples

Example 102. Simple VarNumber Example

Output with different values in a *VarNumber* element to show how the encoded expands as needed.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

    <DataModel name="TheDataModel">

        <Number name="NumLength" size="8">
            <Relation type="size" of="Almonds"/>
        </Number>

        <VarNumber name="Almonds" value="32000" />

    </DataModel>

    <StateModel name="TheState" initialState="Initial">
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
                <Data>
                    <Field name="Almonds" value="1"/> ①
                </Data>
            </Action>

            <Action type="output">
                <DataModel ref="TheDataModel"/>
                <Data>
                    <Field name="Almonds" value="32000"/> ②
                </Data>
            </Action>

            <Action type="output">
                <DataModel ref="TheDataModel"/>
                <Data>
                    <Field name="Almonds" value="3200000"/> ③
                </Data>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>

```

- ① First output should produce 1 byte for the size and 1 byte for the number
- ② Second output should produce 1 byte for the size and 2 bytes for the number
- ③ Third output should produce 1 byte for the size and 3 bytes for the number

Output from this example:

```
>peach -1 --debug NumberExample1.xml

[*] Web site running at: http://10.0.1.87:8888/

[*] Test 'Default' starting with random seed 56906.
2016-07-08 16:38:47.6099 Peach.Pro.Core.Loggers.JobLogger Writing debug.log to:
c:\peach-pro\output\win_x64_debug\bin\Logs\example.xml_20160708163846\debug.log

[R1,-,-] Performing iteration
2016-07-08 16:38:47.7699 Peach.Core.Engine runTest: Performing control recording
iteration.
2016-07-08 16:38:47.8159 Peach.Core.Dom.StateModel Run(): Changing to state
"Initial".
2016-07-08 16:38:47.8259 Peach.Core.Dom.Action Run(Action): Output
2016-07-08 16:38:47.9450 Peach.Pro.Core.Publishers.ConsolePublisher start()
2016-07-08 16:38:47.9450 Peach.Pro.Core.Publishers.ConsolePublisher open()
2016-07-08 16:38:47.9450 Peach.Pro.Core.Publishers.ConsolePublisher output(2 bytes)
①
00000000 01 01 .. .
2016-07-08 16:38:47.9550 Peach.Core.Dom.Action Run(Action_1): Output
2016-07-08 16:38:47.9550 Peach.Pro.Core.Publishers.ConsolePublisher output(3 bytes)
②
00000000 02 7D 00 .}.
2016-07-08 16:38:47.9550 Peach.Core.Dom.Action Run(Action_2): Output
2016-07-08 16:38:47.9550 Peach.Pro.Core.Publishers.ConsolePublisher output(4 bytes)
③
00000000 03 30 D4 00 .0..
2016-07-08 16:38:47.9550 Peach.Pro.Core.Publishers.ConsolePublisher close()
2016-07-08 16:38:47.9550 Peach.Core.Engine runTest: context.config.singleIteration ==
true
2016-07-08 16:38:47.9677 Peach.Pro.Core.Publishers.ConsolePublisher stop()
2016-07-08 16:38:47.9677 Peach.Core.Engine EndTest: Stopping all agents and monitors

[*] Test 'Default' finished.
```

- ① Value of 1 encoded as a single byte
- ② Value of 32000 using two bytes
- ③ Value of 3200000 using three bytes

21.3.26. XmlAttribute

Defines an attribute for an XML element. The parent element must be an [XmlElement](#).



The *XmlElement* and the *XmlAttribute* elements do not support cracking of data. If you need to crack XML content into *XmlElement* and *XmlAttribute* elements, use an [XmlAnalyzer](#) attached to a String element.



The resulting value produced by any child element must be a string. In the case of elements that produce binary output, a [Transformer](#) must be used to convert the binary output to a string. An example of such a transformer is the [Base64Encode](#) transformer.

See also [XML analyzer](#) and [XmlElement](#) element.

Syntax

```
<XmlElement name="example" elementName="Foo">
  <XmlAttribute attributeName="bar">
    <String value="My Attribute!" />
  </XmlAttribute>
</XmlElement>
```

```
<Foo bar="My Attribute!" />
```

Attributes

Required:

attributeName

Name of XML element

Optional:

name

Name of the data model

constraint

Scripting expression that evaluates to true or false. Defaults to null meaning that no constraint is declared.

minOccurs

The minimum number of times this element must occur. Defaults to 1.

- Used to define arrays with variable size. Arrays defined by min/maxOccurs generally have a relation defined.
- Adding this attribute, even with a value of 1, converts the element to an array.

maxOccurs

The maximum number of times this element can occur. Defaults to 1.

- Used to define arrays with static size.
- Adding this attribute, even with a value of 1, converts the element to an array.

mutable

Should this data element be mutated (Or, is it changeable)? The default value is true.

ns

XML namespace.

Child Elements

Blob

Used to represent binary data (array of bytes) to create simple dumb fuzzers in Peach.

Block

Groups one or more data elements in a logical structure.

Choice

Indicates all sub-elements are valid; however, one sub-element should be selected.

Fixup

Performs dynamic transformations such as checksums and CRCs.

Flags

Defines a set of bit sized flags.

Hint

Provides information to mutators.

Number

Defines a binary number of lengths 8, 16, 24, 32, or 64 bits.

String

Defines a single or double byte string.

Mutators

The following mutators operate on this element type:

Enabled when element is marked as an array

ArrayNumericalEdgeCasesMutator

This mutator grows and shrinks an array to counts based on numerical edge cases.

ArrayRandomizeOrderMutator

This mutator randomizes the order of items in an array.

ArrayReverseOrderMutator

This mutator reverses the order of items in an array.

ArrayVarianceMutator

This mutator grows and shrinks an array to a variance of counts based on the current size.

Used for all data elements

DataElementBitFlipper

This mutator produces test cases by flipping bits in the output value.

DataElementDuplicate

This mutator duplicates data elements.

DataElementRemove

This mutator removes data elements.

DataElementSwapNear

This mutator swaps data elements.

SampleNinjaMutator

This mutator combines data elements from different data sets.

Examples

Example 103. Cracking XML File using Analyzer

The following example loads the `example.xml` file into a `String` element, then uses the XML analyzer to convert it to `XmlElement` and `XmlAttribute` elements.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String>
            <Analyzer class="Xml" />
        </String>
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState" >
        <State name="InitialState">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
                <Data fileName="example.xml" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>

        <Publisher class="Console" />
    </Test>
</Peach>

```

Produces the following output:

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 1238.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'TheDataModel' Bytes: 0/684, Bits: 0/5472
Peach.Core.Cracker.DataCracker getSize: -----> DataModel 'TheDataModel'
Peach.Core.Cracker.DataCracker scan: DataModel 'TheDataModel'
Peach.Core.Cracker.DataCracker scan: String 'TheDataModel.DataElement_0' -> Offset:
0, Unsized element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: DataModel 'TheDataModel' Size: <null>, Bytes:
0/684, Bits: 0/5472
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker String 'TheDataModel.DataElement_0' Bytes: 0/684,

```

```

Bits: 0/5472
Peach.Core.Cracker.getSize: -----> String 'TheDataModel.DataElement_0'
Peach.Core.Cracker.scan: String 'TheDataModel.DataElement_0' -> Offset:
0, Unsized element
Peach.Core.Cracker.lookahead: String 'TheDataModel.DataElement_0'
Peach.Core.Cracker.getSize: <----- Last Unsized: 5472
Peach.Core.Cracker.Crack: String 'TheDataModel.DataElement_0' Size:5472,
Bytes: 0/684, Bits: 0/5472
Peach.Core.Dom.DataElement String 'TheDataModel.DataElement_0' value is: <?xml
version="1.0" encoding="utf-8"?>
<Peach xmlns="http://pea.. (Len: 684 chars)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(618 bytes) ①
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
d1p1:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd"
xmlns:d1p1="http://www.w3.org/2001/XMLSchema-instance"><DataModel
name="TheDataModel"><String><Analyzer class="Xml"
/></String></DataModel><StateModelname="TheStateModel"
initialState="InitialState"><State name="InitialState"><Action
type="output"><DataModel ref="TheDataModel" /><Data fileName="c:\temp\example.xml"
/></Action></State></StateModel><Test name="Default"><StateModel ref="TheStateModel"
/><Publisher class="Console" /></Test></Peach>Peach.Core.Publishers.
ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

C:\peach\output\win_x64_debug\bin>
```

① Generated XML output

Example 104. Converting Binary Data with Transformer

The following example converts binary data to a string format using a [Transformer](#) with an [XmlAttribute](#) element.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <XmlElement elementType="Value">
            <XmlAttribute attributeName="data">
                <Block>
                    <Number size="32" value="42" />
                    <Number size="32" value="42" />
                    <Number size="32" value="42" />

                    <Transformer class="Base64Encode" />
                </Block>
            </XmlAttribute>
        </XmlElement>
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState" >
        <State name="InitialState">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>

        <Publisher class="Console" />
    </Test>
</Peach>

```

Produces the following output:

```
> peach -l --debug example.xml

[*] Test 'Default' starting with random seed 59320.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(33 bytes) ①
<Value data="KgAAACoAAAAqAAAA" />Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

① Generated output is Base64 encoded, converting the binary data to a string

21.3.27. XmlCharacterData

Defines an XML CDATA section. This element must a child of [XmlElement](#) and is used to fuzz the content of an XML CDATA section. All the output produced from *XmlCharacterData* and it's parent [XmlElement](#) are well formed.



The *XmlCharacterData* and the *XmlElement* elements do not support cracking of data. If you need to crack XML content into *XmlCharacterDataElement* and *XmlElement* components, use an [XmlAnalyzer](#) attached to a String element.



The resulting value produced by any child element(s) must be a string. In the case of elements that produce binary output, a [Transformer](#) must be used to convert the binary output to a string. An example of such a transformer is the [Base64Encode](#) transformer.

See also [XML analyzer](#), [XmlElement](#) and [XmlAttribute](#) elements.

Syntax

```
<XmlElement elementType="Foo">
  <XmlCharacterData>
    <String value="Hello World!" />
  </XmlCharacterData>
</XmlElement>
```

```
<Foo><![CDATA[Hello World!]]></Foo>
```

Attributes

Required:

No required attributes.

Optional:

[name](#)

Name of the data model

[length](#)

length of the data element. Default is null

[lengthType](#)

Units of the length attribute. Default is bytes

constraint

Scripting expression that evaluates to true or false. Default is null

token

This element should be treated as a token when parsing, defaults to false.

mutable

Is data element changeable (should it be mutated), defaults to true.

Child Elements

Blob

Used to represent binary data (array of bytes) to create simple dumb fuzzers in Peach.

Block

Group one or more data elements in a logical structure.

Choice

Indicate all sub-elements are valid; however, only one sub-element should be selected.

Fixup

Dynamic transformations such as checksums and CRCs.

Flags

Defines a set of bit sized flags.

Hint

Provide information to mutators.

Number

Defines a binary number of lengths 8, 16, 24, 32, or 64 bits.

String

Defines a single or double byte string.

Xmlelement

Defines an XML element, the basic building block of XML documents.

Mutators

The following mutators operate on this element type:

Enabled when element is marked as an array

ArrayNumericalEdgeCasesMutator

This mutator grows and shrinks an array to counts based on numerical edge cases.

ArrayRandomizeOrderMutator

This mutator randomizes the order of items in an array.

ArrayReverseOrderMutator

This mutator reverses the order of items in an array.

ArrayVarianceMutator

This mutator grows and shrinks an array to a variance of counts based on the current size.

Used for all data elements

DataElementBitFlipper

This mutator produces test cases by flipping bits in the output value.

DataElementDuplicate

This mutator duplicates data elements.

DataElementRemove

This mutator removes data elements.

DataElementSwapNear

This mutator swaps data elements.

SampleNinjaMutator

This mutator combines data elements from different data sets.

Examples

Example 105. XML Namespace

The following example models an XML snippet with a CDATA section.

12. XML to model

```
<Foo><! [CDATA[Hello World!] ]></Foo>
```

13. example.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="TheDataModel">
    <XmlElement elementName="Foo">
        <XmlCharacterData>
            <String value="Hello World!" />
        </XmlCharacterData>
    </XmlElement>
</DataModel>

<StateModel name="TheStateModel" initialState="InitialState" >
    <State name="InitialState">
        <Action type="output">
            <DataModel ref="TheDataModel"/>
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheStateModel"/>
    <Publisher class="Console" />
</Test>
</Peach>
```

Produces the following output:

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 50415.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(90 bytes) ①
<Foo><![CDATA[Hello World!]]></Foo>Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

① Produced XML

21.3.28. XmlElement

Defines an XML element, the basic building block of XML documents. The *XmlElement* is used to fuzz the content of an XML document, but not the XML parser. All the output produced from *XmlElement* and *XmlAttribute* are well formed.



The *XmlElement* and the *XmlAttribute* elements do not support cracking of data. If you need to crack XML content into *XmlElement* and *XmlAttribute* components, use an [XmlAnalyzer](#) attached to a String element.



The resulting value produced by any child element must be a string. In the case of elements that produce binary output, a [Transformer](#) must be used to convert the binary output to a string. An example of such a transformer is the [Base64Encode](#) transformer.

See also [XML analyzer](#) and [XmlAttribute](#) element.

Syntax

```
<XmlElement elementType="Foo">
    <XmlElement elementType="Bar">
        <String value="Hello World!" />
    </XmlElement/>
</XmlElement>
```

```
<Foo><Bar>Hello World!</Bar></Foo>
```

Attributes

Required:

elementType

Name of the XML element

Optional:

name

Name of the data model

length

length of the data element. Default is null

lengthType

Units of the length attribute. Default is bytes

constraint

Scripting expression that evaluates to true or false. Default is null

minOccurs

The minimum number of times this element must occur. Defaults to 1.

- Used to define arrays with variable size. Arrays defined by min/maxOccurs generally have a relation defined.
- Adding this attribute, even with a value of 1, converts the element to an array.

maxOccurs

The maximum number of times this element can occur. Defaults to 1.

- Used to define arrays with static size.
- Adding this attribute, even with a value of 1, converts the element to an array.

occurs

Actual occurrences of element. Defaults to 1.

- Used to define arrays with static size.
- Adding this attribute, even with a value of 1, converts the element to an array.

ns

XML namespace

token

This element should be treated as a token when parsing, defaults to false.

mutable

Is data element changeable (should it be mutated), defaults to true.

Child Elements

Blob

Used to represent binary data (array of bytes) to create simple dumb fuzzers in Peach.

Block

Group one or more data elements in a logical structure.

Choice

Indicate all sub-elements are valid; however, only one sub-element should be selected.

Fixup

Dynamic transformations such as checksums and CRCs.

Flags

Defines a set of bit sized flags.

Hint

Provide information to mutators.

Number

Defines a binary number of lengths 8, 16, 24, 32, or 64 bits.

String

Defines a single or double byte string.

XmlAttribute

Defines an attribute for an XML element.

XmlElement

Defines an XML element, the basic building block of XML documents.

Mutators

The following mutators operate on this element type:

Enabled when element is marked as an array

ArrayNumericalEdgeCasesMutator

This mutator grows and shrinks an array to counts based on numerical edge cases.

ArrayRandomizeOrderMutator

This mutator randomizes the order of items in an array.

ArrayReverseOrderMutator

This mutator reverses the order of items in an array.

ArrayVarianceMutator

This mutator grows and shrinks an array to a variance of counts based on the current size.

Used for all data elements

DataElementBitFlipper

This mutator produces test cases by flipping bits in the output value.

DataElementDuplicate

This mutator duplicates data elements.

DataElementRemove

This mutator removes data elements.

DataElementSwapNear

This mutator swaps data elements.

SampleNinjaMutator

This mutator combines data elements from different data sets.

Examples

Example 106. XML Namespace

The following example models an XML snippet with namespaces.

14. XML to model

```
<Peach xmlns="http://peachfuzzer.com/2012/Peach">
    <DataModel name="TheDataModel"/>
</Peach>
```

15. example.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="TheDataModel">
    <XmlElement elementName="Peach" ns="http://peachfuzzer.com/2012/Peach">
        <XmlElement elementName="DataModel" ns="
http://peachfuzzer.com/2012/Peach">
            <XmlAttribute attributeName="name">
                <String value="TheDataModel" />
            </XmlAttribute>
        </XmlElement>
    </XmlElement>
</DataModel>

<StateModel name="TheStateModel" initialState="InitialState" >
    <State name="InitialState">
        <Action type="output">
            <DataModel ref="TheDataModel"/>
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheStateModel"/>

    <Publisher class="Console" />
</Test>
</Peach>
```

Produces the following output:

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 50415.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(90 bytes) ①
<Peach xmlns="http://peachfuzzer.com/2012/Peach"><DataModel name="TheDataModel"
/></Peach>Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

① Produced XML

Example 107. Cracking XML File using Analyzer

The following example loads the `example.xml` file into a `String` element, then use the XML analyzer to convert it to `XmlElement` and `XmlAttribute` elements.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String>
            <Analyzer class="Xml" />
        </String>
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState" >
        <State name="InitialState">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
                <Data fileName="example.xml" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>

        <Publisher class="Console" />
    </Test>
</Peach>

```

Produces the following output:

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 1238.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'TheDataModel' Bytes: 0/684, Bits: 0/5472
Peach.Core.Cracker.DataCracker getSize: -----> DataModel 'TheDataModel'
Peach.Core.Cracker.DataCracker scan: DataModel 'TheDataModel'
Peach.Core.Cracker.DataCracker scan: String 'TheDataModel.DataElement_0' -> Offset:
0, Unsized element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: DataModel 'TheDataModel' Size: <null>, Bytes:
0/684, Bits: 0/5472
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker String 'TheDataModel.DataElement_0' Bytes: 0/684,

```

```

Bits: 0/5472
Peach.Core.Cracker.getSize: -----> String 'TheDataModel.DataElement_0'
Peach.Core.Cracker.scan: String 'TheDataModel.DataElement_0' -> Offset:
0, Unsized element
Peach.Core.Cracker.lookahead: String 'TheDataModel.DataElement_0'
Peach.Core.Cracker.getSize: <----- Last Unsized: 5472
Peach.Core.Cracker.Crack: String 'TheDataModel.DataElement_0' Size:5472,
Bytes: 0/684, Bits: 0/5472
Peach.Core.Dom.DataElement String 'TheDataModel.DataElement_0' value is: <?xml
version="1.0" encoding="utf-8"?>
<Peach xmlns="http://pea.. (Len: 684 chars)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(618 bytes) ①
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
d1p1:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd"
xmlns:d1p1="http://www.w3.org/2001/XMLSchema-instance"><DataModel
name="TheDataModel"><String><Analyzer class="Xml"
/></String></DataModel><StateModelname="TheStateModel"
initialState="InitialState"><State name="InitialState"><Action
type="output"><DataModel ref="TheDataModel" /><Data fileName="c:\temp\example.xml"
/></Action></State></StateModel><Test name="Default"><StateModel ref="TheStateModel"
/><Publisher class="Console" /></Test></Peach>Peach.Core.Publishers.
ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

C:\peach\output\win_x64_debug\bin>
```

① Generated XML output

Example 108. Converting Binary Data with Transformer

The following example converts binary data to a string format using a [Transformer](#) with an [XmlElement](#) element.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <XmlElement elementType="Value">
            <Block>
                <Number size="32" value="42" />
                <Number size="32" value="42" />
                <Number size="32" value="42" />

                <Transformer class="Base64Encode" />
            </Block>
        </XmlElement>
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState" >
        <State name="InitialState">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>

        <Publisher class="Console" />
    </Test>
</Peach>

```

Produces the following output:

```
> peach -l --debug example.xml

[*] Test 'Default' starting with random seed 1238.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(31 bytes) ①
<Value>KgAACoAAAAqAAAA</Value>Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

① Generated output is Base64 encoded, converting the binary data to a string

21.4. Data Element Children

This section contains information about data elements specified as child elements of another element. Child elements modify or clarify the use and scope of the data element. See also [Fixups](#) and [Transformers](#).

21.4.1. Relation

Peach allows data relationship modeling and provides three types of Relations: Size, Count and Offset.

Each Relation element has two required attributes:

type

Declares the relationship type to model (Size, Count, Offset).

of

Names the first data element of the relationship in the data model.

Size-of Relation

The Size relationship type specifies that "X is the size of Y" where "X" is the parent element of the relation element; and "Y" is the first data element of the relationship, specified as the value of the *of* attribute.

The Number element, the parent element of "Relation", is the byte count of the relation element. Unicode characters may be multi-byte and the Number value reflects the additional size.

In this example, the value of the number element indicates the size of the string element named *TheValue* in bytes.

```
<Number size="32" signed="false">
  <Relation type="size" of="theValue" />
</Number>
<String name="theValue" />
```

With expressionGet/expressionSet

Gets are applied during the cracking process and Sets are applied during the publishing process.

For the Size relation, expressionGet and expressionSet use the two variables, `self` and `size`. Self refers to the parent element and size is an integer.

The expressionGet and expressionSet operations should be mathematically inverse operations. The input value for expressionGet and the output value of expressionSet should be identical.

In this example, we provide two python expressions that allow us to modify the size when it is get or

set:

```
<Number size="32" signed="false">
  <Relation type="size" of="Value" expressionGet="size/2" expressionSet="size*2" />
</Number>
<String name="TheValue" />
```

expressionGet

The result of this python expression is used internally to determine how many bytes the [String](#) *TheValue* reads. If Peach picks up 10, it internally stores a 5 and in turn Peach will read 5 bytes into the string.

expressionSet

Produces a value for the publisher. In the prior example, [size](#) stored for *TheValue* is "5" (length of *TheValue*), so the value which Peach outputs via a Publisher will be "5*2" or 10.

Count-of Relation

The Count relationship type specifies that "X is the count of Y" where "X" is the parent element of the relation. "Y" is the relationship element; "Y" is specified as the value of *of*; and, "Y" is an array. The Count relationship only applies to arrays.

The Number element is the byte count of the relationship array. Because Count applies to arrays, specify either [minOccurs](#) or [maxOccurs](#) after the String name.

In this example the number indicates the count of the array *Strings*.

```
<Number size="32" signed="false">
  <Relation type="count" of="Strings" />
</Number>
<String name="Strings" nullTerminated="true" maxOccurs="1024" />
```

With expressionGet/expressionSet

Gets are applied during the cracking process and Sets are applied during the publishing process.

For the Count-of relation, expressionGet and expressionSet use the two variables, [self](#) and [count](#). Self refers to the parent element and count is an integer.

The expressionGet and expressionSet operations should be mathematically inverse operations. The input value for expressionGet and the output value of expressionSet should be identical.

In this example, we provide two python expressions that allow us to modify the count when it is set or get.

```
<Number name="CountIndicator" size="32" signed="false">
  <Relation type="count" of="TheValue" expressionGet="count/2" expressionSet="count*2" />
</Number>
<String name="TheValue" nullTerminated="true" maxOccurs="1024" />
```

expressionGet

This value is used internally and ends up determining how many items `String` expands to. Because of the `maxOccurs=1024` restriction on recurring strings, the max value that Peach should encounter while trying to crack in the `CountIndicator` element is 2048.

expressionSet

Sets the value that is produced. In the prior example, `count` is based on how many `String` elements are read.

Offset-of Relation

The Offset relationship type specifies "X is the offset (in bytes) of Y."

Offset relations allow modeling formats that require changing the offset and also outputting the offset of various elements.

If there are no Relative attributes, Offset is measured from the beginning of the data model. If there is a Relative attribute, Offset is measured from the specified element.

Here we have a series of elements which are ASCII representations of numeric values of the offset sizes to various string elements below.

```

<DataModel name="TheDataModel">
    <String length="4" padCharacter=" " >
        <Relation type="offset" of="Offset0" />
    </String>
    <String length="4" padCharacter=" " >
        <Relation type="offset" of="Offset1" />
    </String>
    <String length="4" padCharacter=" " >
        <Relation type="offset" of="Offset2" />
    </String>
    <String length="4" padCharacter=" " >
        <Relation type="offset" of="Offset3" />
    </String>
    <String length="4" padCharacter=" " >
        <Relation type="offset" of="Offset4" />
    </String>

    <String length="4" padCharacter=" " >
        <Relation type="offset" of="Offset5" />
    </String>

    <String length="4" padCharacter=" " >
        <Relation type="offset" of="Offset6" />
    </String>

    <Block>
        <Block name="Offset0">
            <Block>
                <String name="Offset1" value="CRAZY STRING!" />
                <String value="aslkjalskdjas" />
                <String value="aslkdjalskdjasdkjasdlkjasd" />
            </Block>
            <String name="Offset2" value="ALSKJDALKSJJD" />
            <Block>
                <String name="Offset3" value="1" />
                <String name="Offset4" value="" />
                <String name="Offset5" value="1293812093" />
            </Block>
        </Block>
    </Block>
</DataModel>

```

Relative Offset

A relative offset is the offset from the data element the relation is attached to.

If there is data that represents the distance (in bytes) to somewhere in target element, use the relative offset when you model your data. Peach automatically calculates the location so you know exact where it is.

In this example, when determining the offset of *StringData*, Peach adds or subtracts the position of *OffsetToString* to its value as needed to determine the correct offset.

```
<!-- Other data elements precede -->

<Number name="OffsetToString">
    <Relation type="offset" of="StringData" relative="true" />
</Number>

<String name="StringData" nullTerminated="true"/>
```

relativeTo Offset

Offsets can also relate to another element. This is used when an element contains the offset to another element from the start of a structure.

If there is data that represents the distance (in bytes) to another target element, use relativeTo Offset when you model your data. Peach automatically calculates the location so you know exact where it is. This keeps the relationship intact when fuzzing.

In the following example the offset of *StringData* is calculated by adding the value of *OffsetToString* to the position of *Structure*.

```
<Block name="Structure">
    <!-- Other data elements precede -->

    <Number name="OffsetToString">
        <Relation type="offset" of="StringData" relative="true" relativeTo="Structure" />
    </Number>

    <String name="StringData" nullTerminated="true"/>
</Structure>
```

With expressionGet/expressionSet

Gets are applied during the cracking process and Sets are applied during the publishing process.

For the Offset relations, expressionGet and expressionSet use the two variables, **self** and **offset**. Self refers to the Parent element and offset is an integer.

Expression gets and sets should be each other's mathematical inverse. The Get input and Set output should be the same.

In this example we provide two python expressions that allow us to modify the offset when it is set or get.

```
<DataModel name="TheDataModel">
    <Number name="num" size="32">
        <Relation type="offset" of="Offset0" expressionGet="offset / 2" expressionSet=
"offset * 2"/>
    </Number>

    <Blob/>

    <String name="Target" value="CRAZY STRING!" />
</DataModel>
```

expressionGet

This value is used internally and determines the starting point of **Target** when cracking. data. In the preceding example, if the value of the number **num** is 20, the string **Target** will begin at 10 bytes from the beginning of the data model.

expressionSet

Sets the value that is produced. In the preceding example, **offset** is based on the distance in bytes from the start of the data model to the beginning of the string **Target**.

Offset Relation with Placement

In this model we use a typical pattern where an array of offsets gives us the location of another element. We use the **Placement** element to move the created *Data* strings to after our block called *Chunks*.

NOTE: Placement only works when parsing data into a DataModel. See [Placement](#) for more information.

```
<DataModel name="TheDataModel">
  <Block name="Chunks">
    <Block name="ArrayOfChunks" maxOccurs="4">
      <Number size="8" signed="false">
        <Relation type="offset" of="Data"/>
      </Number>
      <String name="Data" length="6">
        <Placement after="Chunks"/>
      </String>
    </Block>
  </Block>
</DataModel>
```

21.4.2. Placement

The placement element tells the data cracker to move specific elements after parsing the input stream. This, combined with [offset-of relation](#), are the ways Peach supports handling files that contain references to elements by offset.

See also [Arrays of Offsets to Data](#) chapter.



Placement only works while parsing data into the DataModel by an input Action or a Data statement pointing to a file.

Syntax

```
<DataModel name="TheDataModel">
  <Block name="Chunks">
    <Block name="ArrayOfChunks" maxOccurs="4">
      <Number size="8" signed="false">
        <Relation type="offset" of="Data"/>
      </Number>
      <String name="Data" length="6">
        <Placement after="Chunks"/>
      </String>
    </Block>
  </Block>
</DataModel>
```

Attributes

One of the following is required:

after

Name of the data element used in relocation. The relocated data is placed after the specified data element.

before

Name of the data element used in relocation. The relocated data is placed before the specified data element.

Child Elements

None.

Examples

21.5. Fixups

A data that has been fuzzed might fail internal checks because integrity fields, such as checksums, are no longer valid. Fixups provide mechanisms to update the integrity fields and facilitate downstream processing. Peach provides "Fixups" to recalculate integrity-related items such as checksums, CRCs, and hash values.



Fixups should be run after fuzzing and before processing fuzzed data.

Table 1. Default Fixups

Fixup	Checksum	Hashing	Utility	Sequencing
CiscoCdp Checksum	X			
CopyValue			X	
Crc	X			
CrcDual	X			
Expression			X	
FillValue			X	
FragSeqIncrement				X
FromFile			X	
Hmac		X		
Icmp Checksum	X			
IcmpV6 Checksum	X			
IsoFletcher16 Checksum	X			
Lrc	X			
Md5		X		
Script			X	
Sequence Increment				X
Sequence Random				X
Sha		X		
Sha224		X		
Sha256		X		
Sha384		X		
Sha512		X		

Fixup	Checksum	Hashing	Utility	Sequencing
Sspi			Y	
TCP Checksum	X			
UDP Checksum	X			
Unix Time			X	

21.5.1. CiscoCdpChecksum

CiscoCdpChecksum is a Custom Peach algorithm to fix Cisco's one-off error in their CRC implementation (known feature). It is defined in RFC1071. It can be found in the Open source CRCTool Library. This fixup is used when fuzzing protocols perform the Cisco implementation of CRC.

Parent Elements

Number String

Parameters

ref

Reference to the input data element used in the CRC calculation.

Examples

Example 109. Basic Usage Example

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="NoCrcPacket">
        <Number name="SrcPort" size="16" endian="big" value="1234"/>
        <Number name="DestPort" size="16" endian="big" value="1235"/>
        <Number name="Length" size="16" endian="big">
            <Relation type="size" of="NoCrcPacket"/>
        </Number>
        <Number name="checksum" size="16">
        </Number>
    </DataModel>

    <DataModel name="Packet">
        <Number name="SrcPort" size="16" endian="big" value="1234"/>
        <Number name="DestPort" size="16" endian="big" value="1235"/>
        <Number name="Length" size="16" endian="big">
            <Relation type="size" of="Packet"/>
        </Number>
        <Number name="checksum" size="16">
            <Fixup class="CiscoFixup">
                <Param name="ref" value="Packet" />
            </Fixup>
        </Number>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="NoCrcPacket" />
            </Action>

            <Action type="output">
                <DataModel ref="Packet" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex" />
    </Test>
</Peach>

```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 26396.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Binding Error, unable to resolve binding 'Packet' attached to
'NoCrcPacket.Length'.
Peach.Core.Dom.SizeTypeRelation Error, Of returned null
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(8 bytes)
00000000  04 D2 04 D3 00 08 00 00          ??????? ①
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(8 bytes)
00000000  04 D2 04 D3 00 08 52 F6          ??????R? ②
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

① Output without CiscoFixup. Last two bytes are zero.

② Output with CiscoFixup. Last two bytes are a valid CRC for the packet.

21.5.2. CopyValue

The *CopyValue* copies the reference element (ref) value into ours. This is useful for situations where two elements must be identical for redundancy checks.

Parent Elements

Number String Blob Block

Parameters

ref

Reference to the source data element, containing the value to copy.

Examples

Example 110. Basic Usage Example

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="Packet">
        <String name="Magic1" value="ABCD" />
        <String name="Magic2">
            <Fixup class="CopyValue">
                <Param name="ref" value="Magic1" />
            </Fixup>
        </String>
        <Number name="Length" size="16" endian="big">
            <Relation type="size" of="Packet"/>
        </Number>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="Packet" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>
```

Example 111. Example with Block copy

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

  <DataModel name="Packet">
    <Block name="M1">
      <Number size="32" value="1234" />
      <String value="ASDF" />
    </Block>
    <Block name="M2">
      <Number size="32"/>
      <String />
      <Fixup class="CopyValue">
        <Param name="ref" value="M1" />
      </Fixup>
    </Block>
  </DataModel>

  <StateModel name="TheState" initialState="initial">
    <State name="initial">
      <Action type="output">
        <DataModel ref="Packet" />
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheState"/>

      <Publisher class="ConsoleHex" />
    </Test>
  </Peach>

```

Output from example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 3042.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(16 bytes)
00000000 D2 04 00 00 41 53 44 46 D2 04 00 00 41 53 44 46    ???ASDF????ASDF ①
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

① The CopyValue fixup copied the first block into the second block.

21.5.3. CrcDual

The *CrcDual* produces a CRC using data from two elements. It is defined in ISO 3309 and can be found in the Open source CRCTool Library. This is used when fuzzing protocols that perform the CRC checksum on two elements.

Parent Elements

Number String

Parameters

ref1

Reference to first input element used in the CRC calculation.

ref2

Reference to second element used in the CRC calculation.

type

Type of CRC to run [CRC32, CRC16, CRC_CCITT]. Defaults to CRC32.

Examples

Example 112. Basic Usage Example

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

  <DataModel name="Packet">
    <Number name="SrcPort" size="16" endian="big" value="1234"/>
    <Number name="DestPort" size="16" endian="big" value="1235"/>
    <Number name="Length" size="16" endian="big">
      <Relation type="size" of="Packet"/>
    </Number>
    <Number name="checksum" size="16">
      <Fixup class="CrcDualFixup">
        <Param name="ref1" value="SrcPort" />
        <Param name="ref2" value="DestPort" />
        <Param name="type" value="CRC16" />
      </Fixup>
    </Number>
  </DataModel>

  <StateModel name="TheState" initialState="initial">
    <State name="initial">
      <Action type="output">
        <DataModel ref="Packet" />
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex" />
  </Test>
</Peach>
```

Output from this example

```

>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 65340.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(8 bytes)
00000000  04 D2 04 D3 00 08 E2 54          ??????T
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

21.5.4. Crc

The *Crc* produces a CRC using data from the reference element. It is defined in ISO 3309. This is used when fuzzing CRC checksum protocols.

Parent Elements

[Number](#) [String](#)

Parameters

ref

Reference to the input data element used in the CRC calculation.

type

Type of CRC to run [CRC32, CRC16, CRC_CCITT]. Defaults to CRC32.

Examples

Example 113. Basic Usage Example

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="Packet">
        <Number name="SrcPort" size="16" endian="big" value="1234"/>
        <Number name="DestPort" size="16" endian="big" value="1235"/>
        <Number name="Length" size="16" endian="big">
            <Relation type="size" of="Packet"/>
        </Number>
        <Number name="checksum" size="16">
            <Fixup class="Crc">
                <Param name="ref" value="Packet" />
                <Param name="type" value="CRC16" />
            </Fixup>
        </Number>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="Packet" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex" />
    </Test>
</Peach>

```

Output from this example

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 52848.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(8 bytes)
00000000  04 D2 04 D3 00 08 F6 6A          ??????j
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

21.5.5. Expression

The *Expression* evaluates a scripting expression that produces the new value for our element.

Parameters

ref

Reference to the input data element used in the given expression.

expression

Scripting expression to evaluate. Must return a string or integer value.

Scripting Variables

self

Fixup instance

ref

Referenced element

data

Referenced elements value as a byte array

Examples

Example 114. Basic Usage Example

16. example.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

  <Import import="ExpressionFixup" />

  <DataModel name="NumericExpressionData">
    <Number name="NumericData" size="32" value="14"/>
    <Number size="32">
      <Fixup class="ExpressionFixup">
        <Param name="ref" value="NumericData" />
        <Param name="expression" value="ExpressionFixup.multiply_by_three(ref)" />
      </Fixup>
    </Number>
  </DataModel>

  <DataModel name="ByteArrayExpressionData">
```

```

<Blob>
  <Fixup class="ExpressionFixup">
    <Param name="ref" value="StringData" />
    <Param name="expression" value="ExpressionFixup.rot13(ref)" />
  </Fixup>
</Blob>
<String name="StringData" value="uryyb jbeyq"/>
</DataModel>

<DataModel name="InlineExpression">
  <String name="StringData" value="hello there \n"/>
  <String>
    <Fixup class="ExpressionFixup">
      <Param name="ref" value="StringData" />
      <Param name="expression" value="str(str(ref.DefaultValue).upper())" />
    </Fixup>
  </String>
</DataModel>

<StateModel name="TheState" initialState="DumpNumeric">
  <State name="DumpNumeric">
    <Action type="output">
      <DataModel ref="NumericExpressionData"/>
    </Action>
    <Action type="changeState" ref="DumpByteArray"/>
  </State>

  <State name="DumpByteArray">
    <Action type="output">
      <DataModel ref="ByteArrayExpressionData"/>
    </Action>
    <Action type="changeState" ref="DumpInline"/>
  </State>

  <State name="DumpInline">
    <Action type="output">
      <DataModel ref="InlineExpression"/>
    </Action>
  </State>

</StateModel>

<Test name="Default">
  <StateModel ref="TheState"/>

  <Publisher class="ConsoleHex"/>
</Test>
</Peach>

```

17. ExpressionFixup.py

```
from code import InteractiveConsole

import clr
clr.AddReferenceByPartialName('Peach.Core')
import Peach.Core

def multiply_by_three(ref):
    return int(ref.DefaultValue) * 3

def rot13(ref):
    return str(ref.DefaultValue).encode('rot13')

def debug(ctx, ref, data):
    """Useful for basic debugging.

    <Param name="expression" value="debug(self, ref, data)" />
    """
    console = InteractiveConsole(locals=dict(globals().items()+locals().items()))
    console.interact()
```

Output from this example.

```

>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 25461.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(8 bytes)
00000000  0E 00 00 00 2A 00 00 00           ?????*???? ①
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.ChangeState
Peach.Core.Dom.Action Changing to state: DumpByteArray
Peach.Core.Dom.StateModel Run(): Changing to state "DumpByteArray".
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(22 bytes)
00000000  68 65 6C 6C 6F 20 77 6F  72 6C 64 75 72 79 79 62  hello worlduryyb ②
00000010  20 6A 62 65 79 71           jbeyq
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.ChangeState
Peach.Core.Dom.Action Changing to state: DumpInline
Peach.Core.Dom.StateModel Run(): Changing to state "DumpInline".
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(26 bytes)
00000000  68 65 6C 6C 6F 20 74 68  65 72 65 20 0A 48 45 4C  hello there ?HEL ③
00000010  4C 4F 20 54 48 45 52 45  20 0A           LO THERE ?
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

① Output using multiply_by_three expression

② Output using rot13 expression

③ Output using an inline express

21.5.6. FillValue

The *FillValue* is used to fill a data element with sequential numbers. The first number of the sequence is the value of the start parameter, and the end of the sequence is the value of the stop parameter.

If the range specified by the start and stop parameters is less than the size of the referenced element, the sequence repeats after reaching the last value in the range.

This fixup is used when fuzzing the IPsec protocol, as the required padding in the encrypted portion of the data requires padding that is sequential numbers.

Parent Elements

[Number Blob Padding](#)

Parameters

ref

Reference to the data element that receives the sequence of values.

start

Inclusive start fill value.

stop

Inclusive stop fill value.

Examples

Example 115. Basic Usage Example

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="Data">
        <String name="Start" value="Start"/>
        <Blob name="Data" length="17">
            <Fixup class="FillValue">
                <Param name="ref" value="Data"/>
                <Param name="start" value="0"/>
                <Param name="stop" value="10"/>
            </Fixup>
        </Blob>
        <String name="Stop" value="Stop"/>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="Data" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex"/> </Test>
    </Peach>

```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 57241.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(26 bytes)
00000000  53 74 61 72 74 00 01 02  03 04 05 06 07 08 09 0A  Start????????????? ①
00000010  00 01 02 03 04 05 53 74  6F 70                      ?????Stop
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

- ① The FillValue fixup adds sequential numbers from 0 to 10 then repeats until the size of the element is filled.

21.5.7. FragSeqIncrement

FragSeqIncrement is a specialized version of [SequenceIncrement](#) fixup for use in [Frag](#) element *Templates*. This fixup should not be used for any other purpose.

FragSeqIncrement supplies a value that increments with each iteration in a fuzzing session. The value supplied for the first iteration starts with 1. This fixup is useful when a field must be a unique value, or a sequenced value every iteration.

This checksum is useful for fuzzing a protocol that contains an increasing numerical sequence. The Fixup produces valid numbers for the data element it modifies. The maximum value supplied by *FragSeqIncrement* is constrained to the size of the data element.

See also [SequenceIncrement](#), [Frag](#).



This fixup must be used in a [Frag](#) element *Template* model.

Parent Elements

[Number String](#)

Parameters

Group

Designates a group name that bundles a single fixup to multiple data elements. The result coordinates the incrementing process with the members of the group. The default value is "".

InitialValue

Sets the initial value for the first iteration. The default value is 1, and the sequence using the default value starts with 1, 2, 3. If InitialValue is set to the value 57, the sequence starts with 57, 58, 59.

Offset

Sets the initial value each iteration to Offset * (Iteration - 1). The default value is null.

Once

Increment the value once per iteration. The default value is false.

Examples

Example 116. Simple Example

Produce three fragments with each fragment containing the current fragment length, fragment sequence and total length of data. The Payload is 30 bytes of 0x41.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

    <DataModel name="Fragmented">
        <Frag fragLength="10" >

            <Block name="Template">
                <Number name="FragSequence" size="32">
                    <Fixup class="FragSeqIncrement" />
                </Number>

                <Blob name="FragData" />
            </Block>

            <Block name="Payload">
                <Blob valueType="hex" value="
                    41 41 41 41 41 41 41 41 41 41
                    41 41 41 41 41 41 41 41 41 41
                    41 41 41 41 41 41 41 41 41 41"/>
            </Block>
        </Frag>
    </DataModel>

    <StateModel name="TheState" initialState="Initial">
        <State name="Initial">
            <Action type="outfrag">
                <DataModel ref="Fragmented"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>
        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>

```

The example pit will produce three fragments with 10 bytes of payload per-fragment.

Output from this example:

```
>peach -1 --debug example.xml

[*] Web site running at: http://10.0.1.87:8888/

[*] Test 'Default' starting with random seed 8670.
2016-07-12 13:40:06.2441 Peach.Pro.Core.Loggers.JobLogger Writing debug.log to:
c:\peach-pro\output\win_x64_debug\bin\Logs\example.xml_20160712134004\debug.log

[R1,-,-] Performing iteration
2016-07-12 13:40:06.4156 Peach.Core.Engine runTest: Performing control recording
iteration.
2016-07-12 13:40:06.4457 Peach.Core.Dom.Frag Generating fragments:
2016-07-12 13:40:06.4617 Peach.Core.Dom.StateModel Run(): Changing to state
"Initial".
2016-07-12 13:40:06.4617 Peach.Core.Dom.Action Run(Action): Outfrag
2016-07-12 13:40:06.5968 Peach.Pro.Core.Publishers.ConsolePublisher start()
2016-07-12 13:40:06.5968 Peach.Pro.Core.Publishers.ConsolePublisher open()
2016-07-12 13:40:06.6008 Peach.Pro.Core.Publishers.ConsolePublisher output(14 bytes)
00000000 01 00 00 00 41 41 41 41 41 41 41 41 41 41 41 ....AAAAAAA①
2016-07-12 13:40:06.6008 Peach.Pro.Core.Publishers.ConsolePublisher output(14 bytes)
00000000 02 00 00 00 41 41 41 41 41 41 41 41 41 41 41 ....AAAAAAA②
2016-07-12 13:40:06.6008 Peach.Pro.Core.Publishers.ConsolePublisher output(14 bytes)
00000000 03 00 00 00 41 41 41 41 41 41 41 41 41 41 41 ....AAAAAAA③
2016-07-12 13:40:06.6008 Peach.Pro.Core.Publishers.ConsolePublisher close()
2016-07-12 13:40:06.6008 Peach.Core.Engine runTest: context.config.singleIteration ==
true
2016-07-12 13:40:06.6008 Peach.Pro.Core.Publishers.ConsolePublisher stop()
2016-07-12 13:40:06.6008 Peach.Core.Engine EndTest: Stopping all agents and monitors

[*] Test 'Default' finished.
```

① First fragment. Notice sequence number is 1.

② Second fragment. Notice sequence number is 2.

③ Third fragment. Notice sequence number is 3.

21.5.8. FromFile

The *FromFile* is used to fill a BLOB data element with the contents of the specified file. The entire file is transferred, replacing the content of the BLOB. The fixup supports reading of raw or PEM-encoded data files.

The main use case is to access cryptographic keys stored in files instead of exposing the key values in Pits or configuration files. In this use case, Peach pulls keys from .pem files in fuzzing cryptographic protocols.

Parent Elements

[Blob](#)

Parameters

Filename

Filename that contains the data to load into the BLOB.

Encoding

Encoding scheme used on the data in the file. Values are

- Raw - This is the default value.
- Pem - Base-64 X.509 encoding.

Required

Does the file have to exist? Defaults to true. When false, the blob will be empty with length of zero.

Examples

21.5.9. IcmpChecksum

The *IcmpChecksum* produces a checksum using the ICMP algorithm used commonly in ICMP and IPv4. It is defined and documented in RFC 1071 and RFC 2460 and can be found in the Custom Peach implementation.

Parent Elements

Number String

Parameters

ref

The input data element used in the checksum calculation.

Examples

Example 117. Basic Usage Example

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

  <DataModel name="Data">
    <Block name="Content">
      <Number name="Type" size="8"/>
      <Number name="Code" size="8"/>
      <Number name="Checksum" endian="big" size="16">
        <Fixup class="IcmpChecksumFixup">
          <Param name="ref" value="Content" />
        </Fixup>
      </Number>
      <Blob name="Payload" value="hello" />
    </Block>
  </DataModel>

  <StateModel name="TheState" initialState="initial">
    <State name="initial">
      <Action type="output">
        <DataModel ref="Data" />
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="RawV4">
      <Param name="Host" value="127.0.0.1"/>
      <Param name="Interface" value="127.0.0.1"/>
      <Param name="Protocol" value="1"/>
    </Publisher>
  </Test>
</Peach>
```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 37625.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.RawV4Publisher start()
Peach.Core.Publishers.RawV4Publisher open()
Peach.Core.Publishers.RawV4Publisher output(9 bytes)
Peach.Core.Publishers.RawV4Publisher

00000000  00 00 BC 2D 68 65 6C 6C 6F          ..%hello

Peach.Core.Publishers.RawV4Publisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.RawV4Publisher stop()

[*] Test 'Default' finished.
```

21.5.10. IcmpV6Checksum

The *IcmpV6Checksum* fixup produces a checksum using the ICMPV6 algorithm used commonly in ICMP and IPv6. It is defined and documented in RFC 1071 and RFC 2460 and can be found in the Custom Peach implementation.

Parent Elements

- [Number](#)
- [String](#)

Parameters

ref

Reference to the input data element used in the checksum calculation.

src

Source IPv6 address of the local machine.

dst

Destination IPv6 address of the remote machine.

Examples

Example 118. Basic Usage Example

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="Data">
        <Block name="Content">
            <Number name="Type" size="8"/>
            <Number name="Code" size="8"/>
            <Number name="Checksum" endian="big" size="16">
                <Fixup class="IcmpV6ChecksumFixup">
                    <Param name="ref" value="Content" />
                    <Param name="src" value="::1" />
                    <Param name="dst" value="::1" />
                </Fixup>
            </Number>
            <Blob name="Payload" value="hello" />
        </Block>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="Data" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="RawV6">
            <Param name="Host" value="::1"/>
            <Param name="Interface" value="::1"/>
            <Param name="Protocol" value="1"/>
        </Publisher>  </Test>
    </Peach>

```

Output from this example.

```
>peach -1 --debug DocSample.xml

[*] Test 'Default' starting with random seed 43286.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.RawV6Publisher start()
Peach.Core.Publishers.RawV6Publisher open()
Peach.Core.Publishers.RawV6Publisher output(9 bytes)
Peach.Core.Publishers.RawV6Publisher

00000000  00 00 BB E8 68 65 6C 6F          ...>èhello

Peach.Core.Publishers.RawV6Publisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.RawV6Publisher stop()

[*] Test 'Default' finished.
```

21.5.11. IsoFletcher16Checksum

The *IsoFletcher16Checksum* fixup produces a checksum defined in RFC 1008 section 7. OSPF version 2 is one notable protocol which uses this checksum algorithm, as defined in RFC 2328 section 12.1.7.

Parent Elements

Number String

Parameters

ref

The input data element used in the checksum calculation.

Examples

Example 119. Basic Usage Example

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="Data">
        <Block name='Message'>
            <Number name='Header' size='32' endian='big' value='0' />
            <Number name='Checksum' size='16' endian='big' value='0'>
                <Fixup class='IsoFletcher16Checksum'>
                    <Param name='ref' value='Message' />
                </Fixup>
            </Number>
            <Number name='Payload' size='32' endian='big' value='3133731337' />
        </Block>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="Data" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>
        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>
```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 32810.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "initial".
Peach.Core.Dom.Action Run(Action): Output
Peach.Pro.Core.Publishers.ConsolePublisher start()
Peach.Pro.Core.Publishers.ConsolePublisher open()
Peach.Pro.Core.Publishers.ConsolePublisher output(10 bytes)
00000000  00 00 00 00 49 37 BA C8 F2 09      ....I7....
Peach.Pro.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Pro.Core.Publishers.ConsolePublisher stop()
Peach.Core.Engine EndTest: Stopping all agents and monitors

[*] Test 'Default' finished.
```

21.5.12. Lrc

The *Lrc* fixup produces a longitudinal redundancy check (LRC) using data from a single element. The LRC is defined in ISO 1150:1978 and the algorithm is part of the custom Peach implementation.

Parent Elements

- [Number](#)
- [String](#)
- [Blob](#)

Parameters

ref

Reference to the input data element used in the LRC calculation.

Examples

Example 120. Basic Usage Example

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

  <DataModel name="Data">
    <String name="Start" value="Start"/>
    <Blob name="Data" valueType="hex" value="BEEFEA7E41">
      <Fixup class="LRCFixup">
        <Param name="ref" value="Data"/>
      </Fixup>
    </Blob>
    <String name="Stop" value="Stop"/>
  </DataModel>

  <StateModel name="TheState" initialState="initial">
    <State name="initial">
      <Action type="output">
        <DataModel ref="Data" />
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex"/>
  </Test>
</Peach>

```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 13931.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(10 bytes)
00000000  53 74 61 72 74 AA 53 74  6F 70          Start?Stop
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

21.5.13. Hmac

The *Hmac* fixup hashes the *ref* element's value using the one of the defined HMAC algorithms. This is used in fuzzing IPsec protocol implementations.

Parent Elements

[String Blob](#)

Parameters

ref

Reference to the input data element used in the HMAC calculation.

Key

Key to use for HMAC.

HMAC

Hash algorithm to use (HMACSHA1, HMACMD5, HMACRIPEMD160, HMACSHA256, HMACSHA384, HMACSHA512, MACTripleDES).

Length

Length in bytes to return from HMAC, the default value is 0 and returns all bytes. Allows all or a portion of the calculated hash to be returned to the element.

Examples

Example 121. SHA-1 HMAC returning all bytes

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

  <DataModel name="Data">
    <String name="Start" value="Start"/>
    <Blob name="Data" valueType="hex" value="BEEFEA7E41">
      <Fixup class="HMAC">
        <Param name="ref" value="Data"/>
        <Param name="Key" value="aeaeaeaeaeaeaeaeaeaeaeaeaeae"/>
        <Param name="Hash" value="HMACSHA1"/>
        <Param name="Length" value="0"/>
      </Fixup>
    </Blob>
    <String name="Stop" value="Stop"/>
  </DataModel>

  <StateModel name="TheState" initialState="initial">
    <State name="initial">
      <Action type="output">
        <DataModel ref="Data" />
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex"/>
  </Test>
</Peach>

```

Output from this example

```
>peach -1 --debug DocSample.xml

[*] Test 'Default' starting with random seed 40206.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(29 bytes)
00000000  53 74 61 72 74 43 A0 3C  5D 3C 68 39 53 B4 55 B5  StartC?<]<h9S?U?
00000010  07 E1 74 60 37 E9 72 D4  52 53 74 6F 70          ??t`7?r?RStop
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

Example 122. SHA-1 HMAC returning 10 bytes of output

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

  <DataModel name="Data">
    <String name="Start" value="Start"/>
    <Blob name="Data" valueType="hex" value="BEEFEA7E41">
      <Fixup class="HMAC">
        <Param name="ref" value="Data"/>
        <Param name="Key" value="aeaeaeaeaeaeaeaeaeaeaeaeaeae"/>
        <Param name="Hash" value="HMACSHA1"/>
        <Param name="Length" value="10"/>
      </Fixup>
    </Blob>
    <String name="Stop" value="Stop"/>
  </DataModel>

  <StateModel name="TheState" initialState="initial">
    <State name="initial">
      <Action type="output">
        <DataModel ref="Data" />
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheState"/>

      <Publisher class="ConsoleHex"/>
    </Test>
  </Peach>

```

Output from this example

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 27945.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(19 bytes)
00000000  53 74 61 72 74 43 A0 3C  5D 3C 68 39 53 B4 55 53  StartC?<]<h9S?US ①
00000010  74 6F 70                                     top
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

① The output of the HMAC is truncated to 10 bytes

21.5.14. Md5

The *Md5* fixup hashes the value of the reference element using the MD5 algorithm. The MD5 algorithm is defined in RFC 1321 and can be found in the .Net Framework Implementation.

Parent Elements

- [String](#)
- [Blob](#)

Parameters

ref

Reference to the input data element used in the hash calculation.

Examples

Example 123. Basic Usage Example

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

  <DataModel name="Data">
    <String name="Start" value="Start"/>
    <Blob name="Data" valueType="hex" value="BEEFEA7E41">
      <Fixup class="MD5Fixup">
        <Param name="ref" value="Data"/>
      </Fixup>
    </Blob>
    <String name="Stop" value="Stop"/>
  </DataModel>

  <StateModel name="TheState" initialState="initial">
    <State name="initial">
      <Action type="output">
        <DataModel ref="Data" />
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheState"/>

      <Publisher class="ConsoleHex"/> </Test>
  </Peach>

```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 60443.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(25 bytes)
00000000  53 74 61 72 74 10 09 04  EA 69 04 2A 0E 15 00 72  Start?????i?*???r
00000010  FD 70 D0 25 52 53 74 6F  70                      ?p%RStop
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

21.5.15. Script

Script is a proxy that allow you to write fixups in a scripting language such as Python or Ruby.

The class parameter is required for python fixups. It has two arguments:

self

a python construct inherent in the language.

element

the data element that the script "fixed up".

The class parameter's returned argument must match the Fixup parent's DataElement type.

Parent Elements

Parent type must match the DataElement type returned by the Fixup.

Parameters

ref

Reference to the input data element used in the script.

class

Reference to the Python class to call.

Examples

Example 124. Basic Usage Example

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

<Import import="MathWrapperFixup" />

<DataModel name="TheDataModel">
    <Number name="FirstNumber" size="16" endian="big" value="10"/>
    <Number name="Log10Number" size="16" endian="big">
        <Fixup class="ScriptFixup">
            <Param name="class" value="MathWrapperFixup.MathWrapper" />
            <Param name="ref" value="FirstNumber" />
        </Fixup>
    </Number>
</DataModel>

<StateModel name="TheState" initialState="initial">
    <State name="initial">
        <Action type="output">
            <DataModel ref="TheDataModel" />
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex" /> </Test>
</Peach>

```

```

# MathWrapperFixup.py
import math

class MathWrapper:
    def __init__(self, parent):
        self._parent = parent

    def fixup(self, element):
        return int(math.log10(element.DefaultValue))

```

Output from this example.

```
>peach -1 --debug DocSample.xml

Peach.Core.Fixups.ScriptFixup ScriptFixup(): _pythonFixup != null

[*] Test 'Default' starting with random seed 22619.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Fixups.ScriptFixup fixupImpl(): ref: 27449293
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(4 bytes)
00000000  00 0A 00 01                                ****
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

21.5.16. SequenceIncrement

SequenceIncrement supplies a value that increments with each iteration in a fuzzing session. The value supplied for the first iteration starts with 1. This fixup is useful when a field must be a unique value, or a sequenced value every iteration.

This checksum is useful for fuzzing a protocol that contains an increasing numerical sequence. The Fixup produces valid numbers for the data element it modifies. The maximum value supplied by *SequenceIncrement* is constrained to the size of the data element.

Parent Elements

Number String

Parameters

Group

Designates a group name that bundles a single fixup to multiple data elements. The result coordinates the incrementing process with the members of the group. The default value is "".

InitialValue

Sets the initial value for the first iteration. The default value is 1, and the sequence using the default value starts with 1, 2, 3. If InitialValue is set to the value 57, the sequence starts with 57, 58, 59.

Offset

Sets the initial value each iteration to Offset * (Iteration - 1). The default value is null.

Once

Increment the value once per iteration. The default value is false.

Examples

Example 125. Basic SequencialIncrement Example

Three outputs with Once set to false. The number is incremented before each action.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

  <DataModel name="Packet">
    <Number name="SrcPort" size="16" endian="big" value="1234"/>
    <Number name="DestPort" size="16" endian="big" value="1235"/>
    <Number name="Length" size="16" endian="big">
      <Relation type="size" of="Packet"/>
    </Number>
    <Number name="IncrementingNumber" size="16" endian="big">
      <Fixup class="SequenceIncrementFixup"/>
    </Number>
  </DataModel>

  <StateModel name="TheState" initialState="initial">
    <State name="initial">
      <Action type="output">
        <DataModel ref="Packet" />
      </Action>
      <Action type="output">
        <DataModel ref="Packet" />
      </Action>
      <Action type="output">
        <DataModel ref="Packet" />
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex" />
  </Test>
</Peach>
```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 64358.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(8 bytes)
00000000 04 D2 04 D3 00 08 00 01           ????????
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(8 bytes)
00000000 04 D2 04 D3 00 08 00 02           ????????
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(8 bytes)
00000000 04 D2 04 D3 00 08 00 03           ????????
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

Example 126. Non Default Offset

Three outputs with Once set to false and Offset to 5. The number is incremented by 5 each iteration.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

  <DataModel name="Packet">
    <Number name="SrcPort" size="16" endian="big" value="1234"/>
    <Number name="DestPort" size="16" endian="big" value="1235"/>
    <Number name="Length" size="16" endian="big">
      <Relation type="size" of="Packet"/>
    </Number>
    <Number name="IncrementingNumber" size="16" endian="big">
      <Fixup class="SequenceIncrementFixup">
        <Param name="Offset" value="5" />
      </Fixup>
    </Number>
  </DataModel>

  <StateModel name="TheState" initialState="initial">
    <State name="initial">
      <Action type="output">
        <DataModel ref="Packet" />
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheState"/>

      <Publisher class="ConsoleHex" />
    </Test>
  </Peach>

```

Output of this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 26794.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(8 bytes)
00000000  04 D2 04 D3 00 08 00 01           ????????
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

Example 127. Non Default Once

Three outputs with Once set to true. Each action outputs the same incremented number.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="Packet">
        <Number name="SrcPort" size="16" endian="big" value="1234"/>
        <Number name="DestPort" size="16" endian="big" value="1235"/>
        <Number name="Length" size="16" endian="big">
            <Relation type="size" of="Packet"/>
        </Number>
        <Number name="IncrementingNumber" size="16" endian="big">
            <Fixup class="SequenceIncrementFixup">
                <Param name="Once" value="true" />
            </Fixup>
        </Number>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="Packet" />
            </Action>
            <Action type="output">
                <DataModel ref="Packet" />
            </Action>
            <Action type="output">
                <DataModel ref="Packet" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex" />
    </Test>
</Peach>

```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 2157.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(8 bytes)
00000000 04 D2 04 D3 00 08 00 01           ????????
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(8 bytes)
00000000 04 D2 04 D3 00 08 00 01           ????????
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(8 bytes)
00000000 04 D2 04 D3 00 08 00 01           ????????
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()
```

Example 128. Group

The group parameter coordinates the incrementing process among the data elements that specify a fixup with a common group name, as in the following DataModel.

```

<?xml version="1.0" encoding="utf-8"?>

<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name='DM'>
        <Number name='num' size='16'>
            <Fixup class='SequenceIncrementFixup'>
                <Param name='Group' value='mygroup' />
            </Fixup>
        </Number>
        <Number name='num2' size='16'>
            <Fixup class='SequenceIncrementFixup'>
                <Param name='Group' value='mygroup' />
            </Fixup>
        </Number>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="Packet" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex" />
    </Test>
</Peach>
```

In this example, data elements "num" and "num3" each have a fixup defined with the Group parameter set to "mygroup". This means that both data elements use the same fixup. When fuzzing occurs, the fixup initially assigns the value 1 to num. The next data item, "num2", uses this fixup as well and receives the value 2.

On the second fuzzing iteration, num and num2 receive the values 3 and 4 from the fixup, respectively.

Now, to contrast this behavior, if the Group parameter is not specified, num uses one fixup that has a starting value of 1; num2 also uses a fixup, albeit a different fixup, that has a starting value of 1. When fuzzing occurs, the fixup associated to num assigns the value 1 to num. The other fixup, associated to num2, assigns the value 1 to num2.

On the second fuzzing iteration, num and num2 receive the next values from their fixups. For num, this value is 2. For num2, the value from the second fixup is 2 as well.

21.5.17. SequenceRandom

The *SequenceRandom* fixup produces a random integer each iteration. This is useful when a field must be a unique value every iteration.

This checksum can be used when fuzzing protocols that contain a nonce or pseudo-random sequence number. The RNG uses the same seed as Peach, so values produced are reproducible by setting the seed via the command line.



SequenceRandom does not protect duplicate values from occurring.

Parent Elements

[Number](#) [String](#)

Parameters

This fixup does not support any parameters.

Examples

Example 129. Example of SequenceRandom Fixup

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="Packet">
        <Number name="SrcPort" size="16" endian="big" value="1234"/>
        <Number name="DestPort" size="16" endian="big" value="1235"/>
        <Number name="Length" size="16" endian="big">
            <Relation type="size" of="Packet"/>
        </Number>
        <Number name="Nonce" size="16" endian="big">
            <Fixup class="SequenceRandomFixup" />
        </Number>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="Packet" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex" />
    </Test>
</Peach>
```

Output of this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 24885.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(8 bytes)
00000000  04 D2 04 D3 00 08 93 F9          ????????
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

21.5.18. Sha1

Performs an SHA-1 hash of the value of the reference element and places the result in the parent element. This is used when fuzzing protocols that hash data with SHA-1.

The SHA1 algorithm is defined in US Federal Information Processing Standard (FIPS) PUB 180-4 and is implemented in .Net Framework Implementation - System.Security.Cryptography.

Parent Elements

[String](#) [Blob](#)

Parameters

ref

Reference to the input data element used in the hash calculation.

Examples

Example 130. Basic Sha1 Fixup Example

```

<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="Packet">
        <Block name="Header">
            <Number name="SrcPort" size="16" endian="big" value="1234"/>
            <Number name="DestPort" size="16" endian="big" value="1235"/>
            <Number name="Length" size="16" endian="big">
                <Relation type="size" of="Packet"/>
            </Number>
        </Block>
        <Blob name="Checksum">
            <Fixup class="SHA1Fixup">
                <Param name="ref" value="Header" />
            </Fixup>
        </Blob>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="Packet" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex" />
    </Test>
</Peach>
```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 7957.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(26 bytes)
00000000  04 D2 04 D3 00 1A 56 02  66 86 1E 9C 67 29 55 B9  ??????V?f???g)U?
00000010  E4 16 DE 0F 81 F0 10 19  B8 42                      ??????????B
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

21.5.19. Sha224

Performs an SHA-224 hash of value of the reference element value and places the result in the parent element. This is used when fuzzing protocols that hash data with SHA-224.

The SHA224 algorithm is defined in US Federal Information Processing Standard (FIPS) PUB 180-4 and is implemented in .Net Framework Implementation - System.Security.Cryptography.

Parent Elements

[String Blob](#)

Parameters

ref

Reference to the input data element used in the hash calculation.

Examples

Example 131. Basic Sha224 Example

```

<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="Packet">
        <Block name="Header">
            <Number name="SrcPort" size="16" endian="big" value="1234"/>
            <Number name="DestPort" size="16" endian="big" value="1235"/>
            <Number name="Length" size="16" endian="big">
                <Relation type="size" of="Packet"/>
            </Number>
        </Block>
        <Blob name="Checksum">
            <Fixup class="SHA224Fixup">
                <Param name="ref" value="Header" />
            </Fixup>
        </Blob>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="Packet" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex" />
    </Test>
</Peach>
```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 54740.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(34 bytes)
00000000  04 D2 04 D3 00 22 C3 B5  EB 17 A9 4E DE EE 74 69  ???????"?????N??ti
00000010  DD 51 F3 E0 83 0B BD 39  BA 98 EB E7 A7 DC ED F2  ?Q?????9?????????
00000020  1A A3                                         ???

Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

21.5.20. Sha256

Performs an SHA-256 hash of the value of the reference element and places the result in the parent element. This is used when fuzzing protocols that hash data with SHA-256.

The SHA256 algorithm is defined in US Federal Information Processing Standard (FIPS) PUB 180-4 and is implemented in .Net Framework Implementation - System.Security.Cryptography.

Parent Elements

[String Blob](#)

Parameters

ref

Reference to the input data element used in the hash calculation.

Examples

Example 132. Basic Sha256 Example

```

<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="Packet">
        <Block name="Header">
            <Number name="SrcPort" size="16" endian="big" value="1234"/>
            <Number name="DestPort" size="16" endian="big" value="1235"/>
            <Number name="Length" size="16" endian="big">
                <Relation type="size" of="Packet"/>
            </Number>
        </Block>
        <Blob name="Checksum">
            <Fixup class="SHA256Fixup">
                <Param name="ref" value="Header" />
            </Fixup>
        </Blob>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="Packet" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>
```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 16510.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(38 bytes)
00000000  04 D2 04 D3 00 26 77 C1  D9 9C D4 EC 59 14 A1 92  ??????&w?????Y???
00000010  09 5D 9A A0 45 66 7C A4  14 80 DB A5 66 EC 53 3C  ?]??Ef|?????f?S<
00000020  95 E4 34 CF 63 B9                      ??4?c?

Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

21.5.21. Sha384

Performs an SHA-384 hash of value of the reference element and places the result in the parent element. This is used when fuzzing protocols that hash data with SHA-384.

The SHA384 algorithm is defined in US Federal Information Processing Standard (FIPS) PUB 180-4 and is implemented in .Net Framework Implementation - System.Security.Cryptography.

Parent Elements

[String Blob](#)

Parameters

ref

Reference to the input data element used in the hash calculation.

Examples

Example 133. Basic Sha384 Example

```

<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

  <DataModel name="Packet">
    <Block name="Header">
      <Number name="SrcPort" size="16" endian="big" value="1234"/>
      <Number name="DestPort" size="16" endian="big" value="1235"/>
      <Number name="Length" size="16" endian="big">
        <Relation type="size" of="Packet"/>
      </Number>
    </Block>
    <Blob name="Checksum">
      <Fixup class="SHA384Fixup">
        <Param name="ref" value="Header"/>
      </Fixup>
    </Blob>
  </DataModel>

  <StateModel name="TheState" initialState="initial">
    <State name="initial">
      <Action type="output">
        <DataModel ref="Packet"/>
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex"/>
  </Test>
</Peach>
```

Output of this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 22335.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(54 bytes)
00000000  04 D2 04 D3 00 36 20 74  77 0A 0F B0 5A F8 B0 6B  ?????6 tw???Z??k
00000010  BE FC 4C CC 83 1A 25 C5  E9 5B 02 F5 F7 E0 41 05  ??L???%??[????A?
00000020  77 FB 86 6F 02 40 CE 32  E5 46 85 74 7B A3 0C 70  w??o?@?2?F?t{??p
00000030  E6 67 7D 0B 74 5A                      ?g}?tZ

Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

21.5.22. Sha512

Performs an SHA-512 hash of the value of the reference element value and places the result in the parent element. This is used when fuzzing protocols that hash data with SHA-512.

The SHA512 algorithm is defined in US Federal Information Processing Standard (FIPS) PUB 180-4 and is implemented in .Net Framework Implementation - System.Security.Cryptography.

Parent Elements

[String Blob](#)

Parameters

ref

Reference to the input data element used in the hash calculation.

Examples

Example 134. Example of SHA512Fixup Usage

```

<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="Packet">
        <Block name="Header">
            <Number name="SrcPort" size="16" endian="big" value="1234"/>
            <Number name="DestPort" size="16" endian="big" value="1235"/>
            <Number name="Length" size="16" endian="big">
                <Relation type="size" of="Packet"/>
            </Number>
        </Block>
        <Blob name="Checksum">
            <Fixup class="SHA512Fixup">
                <Param name="ref" value="Header" />
            </Fixup>
        </Blob>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="Packet" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex" />
    </Test>
</Peach>
```

Output of this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 4183.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(70 bytes)
00000000  04 D2 04 D3 00 46 ED 87  D8 F8 8A B6 42 62 F9 51  ?????F??????Bb?Q
00000010  4F D8 A5 C7 6C 19 7C 14  8C 03 E3 09 EA 5D 28 78  0???l?|??????](x
00000020  E1 98 6D A7 1A 96 BA 9A  E1 F3 F0 B2 B3 EA 05 5C  ??m?????????????\?
00000030  9F A1 94 90 0A 68 80 2B  DB 9B F0 B6 05 2D 4D E6  ???h?+?????-M?
00000040  DF 36 BB 42 F8 31                               ?6?B?1

Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

21.5.23. Sspi

The `Sspi` fixup enables Peach to perform the Microsoft SSPI challenge-response authentication process for the NTLMv2 mechanism and attempts to negotiate it. The Microsoft SSPI authentication process is common across Microsoft protocols and APIs, including Windows file-sharing protocols CIFS/SMB.

The challenge-response nature of this protocol necessitates using the fixup multiple times to complete an authentication process. For NTLMv2, the challenge-response process consists of at least three messages between the client and server. This fixup uses and stores data in the iteration state bag to consume the last message from the server. Also, the SSPI fixup provides a Boolean indicator identifying whether another message is needed or authentication is complete.

Since the number of messages needed to complete the authentication process is not known initially (due to the mechanism being negotiated), the protocol or API typically has a loop that continues until authentication completes. A loop is the preferred way to model this in the state model of the fuzzing definition. A specific state will be performed over and over until the fixup indicates that authentication has completed. See the examples section of this fixup for an example of how this might be modeled.

Example 135. NOTES

The CIFS_Server Pit is a working example of this process. By default, this fixup uses two iteration state store keys during its operation:

- `Peach.SspiSecurityBuffer` - Stores messages received from the server. In response to this message, the fixup consumes this data and produces the next message in the series.
- `Peach.SspiContinueNeeded` - Indicates whether an additional authentication message from the service is needed. False indicates the authentication is complete. True indicates another message is needed from the server, and is the default value.

Parent Elements

[Blob](#)

Parameters

Required:

User

User name of the authorizing account.

Password

Password associated with the authorizing account.

Optional:

Domain

Domain in which the authorizing account resides. Default is "".

ContinueNeeded

Reference to a key for connection-oriented use in NTLM. The value is the name of the field in the state bag that indicates whether authentication continues (as in a connection).

Examples

Sspi Fixup Example

This example consists of two code fragments:

- A few lines of Python code takes the received message from the server and places it into the Peach state store. The fixup then processes the message text next time the script runs.
- A portion of the State model that performs the challenge response authentication.



The challenge-response authentication requires multiple exchanges between the client and the server to complete the authentication process. The example shows the portion of the state model that achieves the authentication.

The following Python fragment sets up the storage for the challenge/response value in the state bag:

```
import clr  
clr.AddReferenceByPartialName('Peach.Core')
```

```
import Peach.Core  
import System
```

```
def ContinueUpdate(ctx, action):  
    securityBuffer = action.dataModel.find('SecurityBuffer')  
    if securityBuffer != None:  
        ctx.iterationStateStore['Peach.SspiSecurityBuffer'] = securityBuffer.Value
```

The following pit fragment defines the action to achieve authentication in the state model.

```
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=  
"http://www.w3.org/2001/XMLSchema-instance"  
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">
```

```

<DataModel name="Packet">
    <String name="SspiResponse">
        <Fixup class="SspiFixup">
            <String name="User" value="PeachUser"/>
            <String name="Password" value="Calhoun30701"/>
            <String name="Domain" value="CAITLIN" />
            <String name="SecurityContinues" value="" />
        </Fixup>
    </String>

    <BLOB name="UserData" />
</DataModel>

<StateModel>
    <State name="Initial">
        <!-- Fill in initial state details -->
    </State>

    <!-- The next state will loop several times to perform
        the challenge-response authentication -->
    <State name="Authenticate">
        <Action type="output" name="SessionSetupRequest">
            <DataModel ref="CIFS:TcpRequest"/>
        </Action>

        <Action type="input" name="SessionSetupResponse"
            onComplete="cifs.ContinueUpdate(Context, Action)">
            <DataModel ref="CIFS:TcpResponse"/>
        </Action>

        <!-- Next action does the loop until authenticate is done -->
        <Action type="changeState" ref="Authenticate"
            when="Context.iterationStateStore['Peach.SspiContinueNeeded']" />

        <!-- Authentication complete, goto next state -->
        <Action type="changeState" ref="ConnectIpc" />
    </State>
</StateModel>

<!-- Place the remaining pit details here. -->

</Peach>

```

21.5.24. TCPChecksum

The *TCPChecksum* fixup produces an Internet checksum as defined in RFC 1071. This checksum is primarily used when fuzzing the TCPv4 and TCPv6 Protocols.

A discussion on how to apply TCPChecksum can be found in RFC 793 for IPv4 and RFC 2460 for IPv6.

Parent Elements

[Number String](#)

Parameters

ref

Reference to the input data element used in the checksum calculation.

src

Source IP Address. IP address of the local machine.

dst

Destination IP Address. IP address of the remote machine.

Examples

Example 136. Basic Usage Example

This example shows the basic usage of TCPChecksum.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="Packet">
        <Block name="Header">
            <Number name="SrcPort" size="16" endian="big" value="1234"/>
            <Number name="DestPort" size="16" endian="big" value="1235"/>
            <Number name="SequenceNumber" size="32" endian="big" valueType="hex"
value="0043a577"/>
            <Number name="AcknowledgmentNumber" size="32" endian="big" value="0"/>
            <Number name="DataOffset" size="4" endian="big" />
            <Number name="ControlBits" size="12" endian="big" />
            <Number name="WindowSize" size="16" endian="big" valueType="hex" value=
"aaaa"/>
            <Number name="CheckSum" size="16" endian="big">
                <Fixup class="TCPChecksum">
                    <Param name="ref" value="Packet" />
                    <Param name="src" value="127.0.0.1" />
                    <Param name="dst" value="127.0.0.1" />
                </Fixup>
            </Number>
            <Number name="UrgentPointer" size="16" endian="big"/>
        </Block>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="Packet" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex" />      </Test>
    </Peach>

```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 20133.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(20 bytes)
00000000  04 D2 04 D3 00 43 A5 77  00 00 00 00 00 AA AA  ?????C?w?????????
00000010  A7 D8 00 00                           ****
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

21.5.25. UDPChecksum

The *UDPChecksum* fixup produces an checksum as defined in RFC 1071. This checksum is primarily used when fuzzing the UDPv4 and UDPv6 Protocols.

A discussion on how to apply UDPChecksum can be found in RFC 793 for IPv4 and RFC 2460 for IPv6.

Parent Elements

Number String

Parameters

ref

Reference to the input data element used in the checksum calculation.

src

Source IP Address. IP address of the local machine.

dst

Destination IP Address. IP address of the remote machine.

Examples

Example 137. UDP Packet Example

This example models a UDP packet which includes use of the UDPChecksum.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="Packet">
        <Number name="SrcPort" size="16" endian="big" value="1234"/>
        <Number name="DestPort" size="16" endian="big" value="1235"/>
        <Number name="Length" size="16" endian="big">
            <Relation type="size" of="Packet"/>
        </Number>
        <Number name="checksum" size="16">
            <Fixup class="UDPChecksum">
                <Param name="ref" value="Packet" />
                <Param name="src" value="192.168.1.10" />
                <Param name="dst" value="192.168.1.11" />
            </Fixup>
        </Number>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="Packet" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex" />      </Test>
    </Peach>

```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 60502.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(8 bytes)
00000000  04 D2 04 D3 00 08 D3 72          ???????r
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

21.5.26. UnixTime

The *FromFile* is used to fill a BLOB data element with the contents of the specified file. The entire file is transferred, replacing the content of the BLOB. The fixup supports reading of raw or PEM-encoded data files.

The main use case is to access cryptographic keys stored in files instead of exposing the key values in Pits or configuration files. In this use case, Peach pulls keys from .pem files in fuzzing cryptographic protocols.

Parent Elements

- [Number](#) (Not using Format parameter)
- [String](#) (When using Format parameter)

Parameters

Gmt

Is time in GMT? Defaults to true.

Format

Format string to encode time with. Uses the Microsoft.NET format strings for Date and Time. When specified the parent element must be a String. If no format is provided the parent must be a Number.

- [Standard Date and Time Format Strings](#)
- [Custom Date and Time Format Strings](#)

Examples

Example 138. Simple Example w/o Format Parameter

Simple example of using the *UnixTime* fixup with out the Format parameter.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

  <DataModel name="TheDataModel">
    <Number name="DateAndTime" size="32">
      <Fixup class="UnixTime" />
    </Number>
  </DataModel>

  <StateModel name="TheState" initialState="initial">
    <State name="initial">
      <Action type="output">
        <DataModel ref="TheDataModel" />
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheState"/>
    <Publisher class="ConsoleHex"/>
  </Test>
</Peach>

```

Output from this example:

```
>peach -1 --debug example.xml

[*] Web site running at: http://10.0.1.87:8888/

[*] Test 'Default' starting with random seed 27871.
2016-07-08 18:08:51.9607 Peach.Pro.Core.Loggers.JobLogger Writing debug.log to:
c:\peach-pro\output\win_x64_debug\bin\Logs\example.xml_20160708180850\debug.log

[R1,-,-] Performing iteration
2016-07-08 18:08:52.1052 Peach.Core.Engine runTest: Performing control recording
iteration.
2016-07-08 18:08:52.1443 Peach.Core.Dom.StateModel Run(): Changing to state
"initial".
2016-07-08 18:08:52.1443 Peach.Core.Dom.Action Run(Action): Output
2016-07-08 18:08:52.2744 Peach.Pro.Core.Publishers.ConsolePublisher start()
2016-07-08 18:08:52.2744 Peach.Pro.Core.Publishers.ConsolePublisher open()
2016-07-08 18:08:52.2784 Peach.Pro.Core.Publishers.ConsolePublisher output(4 bytes)
00000000 A4 4E 80 57 .N.W
2016-07-08 18:08:52.2784 Peach.Pro.Core.Publishers.ConsolePublisher close()
2016-07-08 18:08:52.2784 Peach.Core.Engine runTest: context.config.singleIteration ==
true
2016-07-08 18:08:52.2784 Peach.Pro.Core.Publishers.ConsolePublisher stop()
2016-07-08 18:08:52.2944 Peach.Core.Engine EndTest: Stopping all agents and monitors

[*] Test 'Default' finished.
```

Example 139. Simple Example with Format Parameter

Simple example of using the *UnixTime* fixup with the Format parameter.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String name="RFC1123Format" >
            <Fixup class="UnixTime">
                <Param name="Format" value="R" />
            </Fixup>
        </String>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="TheDataModel" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>
        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>

```

Output from this example:

```
>peach -1 --debug example.xml

[*] Web site running at: http://10.0.1.87:8888/

[*] Test 'Default' starting with random seed 40931.
2016-07-08 18:10:55.0103 Peach.Pro.Core.Loggers.JobLogger Writing debug.log to:
c:\peach-pro\output\win_x64_debug\bin\Logs\example.xml_20160708181053\debug.log

[R1,-,-] Performing iteration
2016-07-08 18:10:55.1412 Peach.Core.Engine runTest: Performing control recording
iteration.
2016-07-08 18:10:55.1692 Peach.Core.Dom.StateModel Run(): Changing to state
"initial".
2016-07-08 18:10:55.1692 Peach.Core.Dom.Action Run(Action): Output
2016-07-08 18:10:55.2913 Peach.Pro.Core.Publishers.ConsolePublisher start()
2016-07-08 18:10:55.2913 Peach.Pro.Core.Publishers.ConsolePublisher open()
2016-07-08 18:10:55.2953 Peach.Pro.Core.Publishers.ConsolePublisher output(29 bytes)
00000000 53 61 74 2C 20 30 39 20 4A 75 6C 20 32 30 31 36 Sat, 09 Jul 2016
00000010 20 30 31 3A 31 30 3A 35 35 20 47 4D 54 01:10:55 GMT
2016-07-08 18:10:55.2953 Peach.Pro.Core.Publishers.ConsolePublisher close()
2016-07-08 18:10:55.3103 Peach.Core.Engine runTest: context.config.singleIteration ==
true
2016-07-08 18:10:55.3103 Peach.Pro.Core.Publishers.ConsolePublisher stop()
2016-07-08 18:10:55.3103 Peach.Core.Engine EndTest: Stopping all agents and monitors

[*] Test 'Default' finished.
```

21.6. Transformers

Transformers perform static transforms or encoding on the parent element.

Transformers most often occur in pairs, as encoding and decoding functions. The decoding function is the inverse of the encoding function. In this sense, they are bi-directional. Examples include ZIP compression, Base64 encoding, and HTML encoding.

Some transformers lack a decoding function; they are unidirectional. For example, the MD5 transformer is a unidirectional transformer that creates a hash.

Transformers are different than Fixups:

- Transformers operate on the parent data element.
- Fixups generate their outputs by using values from other data elements.

21.6.1. Example

```
<DataModel name="Base64TLV">
  <Number name="Type" size="8" signed="false" value="1" token="true" />
  <Number name="Length" size="16" signed="false">
    <Relation type="size" of="base64Block" />
  </Number>

  <Block name="base64Block">
    <Transformer class="Base64Encode" />
    <Blob name="Data" />
  </Block>
</DataModel>
```

The output of the above data model is `0x01<len(b64(Data))><b64(Data)>`.

Table 2. Peach Transformers

Transformer	Compress	Crypto	Encode
Aes128		X	
Base64Decode			X
Base64Encode			X
Bz2Compress	X		
Bz2Decompress	X		
Des		X	

Transformer	Compress	Crypto	Encode
GzipCompress	X		
GzipDecompress	X		
Hex			X
HMAC		X	
HtmlDecode			X
HtmlEncode			X
IntToHex			X
Ipv4StringToOctet			X
Ipv6StringToOctet			X
JsEncode			X
MD5Crypt		X	
NetBiosEncode			X
Sha1		X	
Sha256		X	
SidStringToBytes			X
TripleDes		X	
Truncate			
UrlEncode			X

21.6.2. Aes128

Type

Cryptography

Aes128 transforms the parent [DataModel](#) by encrypting the data using AES (Rijndael) with the provided Key and Initialization Vector.

The [Transformer](#) can bidirectionally encrypt and decrypt data.

Parameters

Required:

Key

User provided symmetric key that is used to encrypt the value. Must be a hex string representation of a 16 byte key.

IV

User provided initialization vector that is used as the first block for the AES128 operation. Must be a hex string representation of an 16 byte value.

Optional:

None.

Actions Supported

input

On input, this transformer decrypts the incoming data.

output

On output, this transformer encrypts the outgoing data.

Examples

Example 140. Simple Encrypt and Decrypt Example

This example uses Aes128 on an inline value, writes the transformed value to a file, and reads the value from the file to the console.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

<DataModel name="Ex1">
```

```

<String name="TransformMe" value="supersupersecret" >
    <Transformer class="Aes128">
        <Param name="Key" value="ae1234567890aeaffeda214354647586"/>
        <Param name="IV" value="aaeaeaeaeaeaeaeaeaeaeaeaeae"/>
    </Transformer>
</String>
</DataModel>

<StateModel name="TheState" initialState="initial">
    <State name="initial">
        <!-- Encrypted Output -->
        <Action type="output" publisher="ConsolePub">
            <DataModel ref="Ex1" />
        </Action>

        <!-- Write Encrypted Output to File -->
        <Action type="output" publisher="FilePubWrite">
            <DataModel ref="Ex1" />
        </Action>

        <Action type="close" publisher="FilePubWrite" />

        <!-- Read and decrypt encrypted file and slurp output to console -->
        <Action type="input" publisher="FilePubRead" >
            <DataModel name="InputModel" ref="Ex1" />
        </Action>

        <Action type="slurp" valueXpath="//InputModel//TransformMe" setXpath=
        "//OutputModel//StringValue" />

        <Action type="output" publisher="ConsolePub">
            <DataModel name="OutputModel">
                <String name="StringValue" />
            </DataModel>
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex" name="ConsolePub"/>

    <Publisher class="File" name="FilePubWrite">
        <Param name="FileName" value="encrypted.bin" />
    </Publisher>

    <Publisher class="File" name="FilePubRead">

```

```

        <Param name="FileName" value="encrypted.bin" />
        <Param name="Overwrite" value="false" />
    </Publisher>
</Test>
</Peach>

```

Output from this example.

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 7617.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(16 bytes)
00000000  22 8D 4A 8B 30 1F 4D 6B  1A 31 24 3D B7 ED 97 E5  "?J?0?Mk?1$=????
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(16 bytes)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Input
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher input()
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataModel 'InputModel' Bytes: 0/16, Bits: 0/128
Peach.Core.Cracker.getSize: -----> DataModel 'InputModel'
Peach.Core.Cracker.scan: DataModel 'InputModel'
Peach.Core.Cracker.scan: String 'InputModel.TransformMe' -> Offset: 0,
Unsized element
Peach.Core.Cracker.getSize: <----- Deterministic: ???
Peach.Core.Cracker.Crack: DataModel 'InputModel' Size: <null>, Bytes :
0/16, Bits: 0/128
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker String 'InputModel.TransformMe' Bytes: 0/16, Bits :
0/128
Peach.Core.Cracker.getSize: -----> String 'InputModel.TransformMe'
Peach.Core.Cracker.scan: String 'InputModel.TransformMe' -> Offset: 0,

```

Unsized element

```
Peach.Core.Cracker.lookahead: String 'InputModel.TransformMe'  
Peach.Core.Cracker.getSize: <----- Last Unsized: 128  
Peach.Core.Cracker.Crack: String 'InputModel.TransformMe' Size: 128,  
Bytes: 0/16, Bits: 0/128  
Peach.Core.Dom.DataElement String 'InputModel.TransformMe' value is: supersupersecret  
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted  
Peach.Core.Dom.Action ActionType.Slurp  
Peach.Core.Dom.Action Slurp, setting OutputModel.StringValue from  
InputModel.TransformMe  
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted  
Peach.Core.Dom.Action ActionType.Output  
Peach.Core.Publishers.ConsolePublisher output(16 bytes)  
00000000 73 75 70 65 72 73 75 70 65 72 73 65 63 72 65 74 supersupersecret  
Peach.Core.Publishers.ConsolePublisher close()  
Peach.Core.Publishers.FilePublisher close()  
Peach.Core.Engine runTest: context.config.singleIteration == true  
Peach.Core.Publishers.ConsolePublisher stop()  
Peach.Core.Publishers.FilePublisher stop()  
Peach.Core.Publishers.FilePublisher stop()
```

[*] Test 'Default' finished.

21.6.3. Base64Decode

Type

Encoder/Decoder

Base64Decode decodes the value in the parent [DataModel](#) using a Base64 decoding algorithm.

The [Transformer](#) can bidirectionally encode data as well as decode data.

Parameters

None.

Attributes

None.

Actions Supported

input

On input, this transformer encodes the data into binary values using a Base64 encoding algorithm.

output

On output, this transformer decodes the data into characters using a Base64 decoding algorithm.

Examples

Example 141. Decoding Value To and From File Example

This example uses Base64Encode on an inline value, writes the transformed value to a file, and reads the value from the file to the console.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="Ex1">
        <String value="Plaintext: " token="true"/>
        <String name="TransformMe" value="QUJDREVGR0g=>
            <Transformer class="Base64Decode" />
        </String>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <!-- Plaintext Output -->
```

```

<Action type="output" publisher="ConsolePub">
    <DataModel ref="Ex1" />
</Action>

<!-- Write Plaintext Output to File -->
<Action type="output" publisher="FilePubWrite">
    <DataModel ref="Ex1" />
    <Data>
        <Field name="TransformMe" value="QUJDREVGR0g=" />
    </Data>
</Action>

<Action type="close" publisher="FilePubWrite" />

<!-- Read and encode Plaintext file and slurp Base64 string to output in
console -->
<Action type="input" publisher="FilePubRead" >
    <DataModel name="InputModel" ref="Ex1" />
</Action>

<Action type="slurp" valueXpath="//InputModel//TransformMe" setXpath=
"//OutputModel//StringValue" />

<Action type="output" publisher="ConsolePub">
    <DataModel name="OutputModel">
        <String name="StringValue" />
    </DataModel>
</Action>
</State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex" name="ConsolePub"/>

    <Publisher class="File" name="FilePubWrite">
        <Param name="FileName" value="encoded.bin" />
    </Publisher>

    <Publisher class="File" name="FilePubRead">
        <Param name="FileName" value="encoded.bin" />
        <Param name="Overwrite" value="false" />
    </Publisher>
</Test>
</Peach>

```

Output from this example.

```
> peach -l --debug example.xml

[*] Test 'Default' starting with random seed 12661.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(19 bytes)
00000000  50 6C 61 69 6E 74 65 78  74 3A 20 41 42 43 44 45  Plaintext: ABCDE
00000010  46 47 48                                     FGH
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(19 bytes)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Input
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher input()
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'InputModel' Bytes: 0/19, Bits: 0/152
Peach.Core.Cracker.DataCracker getSize: -----> DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: String 'InputModel.DataElement_0' -> Pos: 0,
Saving Token
Peach.Core.Cracker.DataCracker scan: String 'InputModel.DataElement_0' -> Pos: 88,
Length: 88
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: DataModel 'InputModel' Size: <null>, Bytes:
0/19, Bits: 0/152
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.String 'InputModel.DataElement_0' Bytes: 0/19, Bits:
0/152
Peach.Core.Cracker.DataCracker getSize: -----> String 'InputModel.DataElement_0'

Peach.Core.Cracker.DataCracker scan: String 'InputModel.DataElement_0' -> Pos: 0,
Saving Token
Peach.Core.Cracker.DataCracker scan: String 'InputModel.DataElement_0' -> Pos: 88,
```

```
Length: 88
Peach.Core.Cracker.getSize: <----- Size: 88
Peach.Core.Cracker Crack: String 'InputModel.DataElement_0' Size: 88,
Bytes: 0/19, Bits: 0/152
Peach.Core.Dom.DataElement String 'InputModel.DataElement_0' value is: Plaintext :
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.String 'InputModel.TransformMe' Bytes: 11/19, Bits:
88/152
Peach.Core.Cracker.getSize: -----> String 'InputModel.TransformMe'
Peach.Core.Cracker.scan: String 'InputModel.TransformMe' -> Offset: 0,
Unsized element
Peach.Core.Cracker.lookahead: String 'InputModel.TransformMe'
Peach.Core.Cracker.getSize: <----- Last Unsized: 64
Peach.Core.Cracker.Crack: String 'InputModel.TransformMe' Size: 96,
Bytes: 0/12, Bits: 0/96
Peach.Core.Dom.DataElement String 'InputModel.TransformMe' value is: QUJDREVGR0g=
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Slurp
Peach.Core.Dom.Action Slurp, setting OutputModel.StringValue from
InputModel.TransformMe
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(12 bytes)
00000000  51 55 4A 44 52 45 56 47  52 30 67 3D          QUJDREVGR0g=
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()
Peach.Core.Publishers.FilePublisher stop()
Peach.Core.Publishers.FilePublisher stop()

[*] Test 'Default' finished.
```

21.6.4. Base64Encode

Type

Encoder/Decoder

Base64Encode encodes the value in the parent [DataModel](#) using a Base64 encoding algorithm.

The [Transformer](#) can bidirectionally encode data as well as decode data.

Parameters

None.

Attributes

None.

Actions Supported

input

On input, this transformer decodes the data into characters using a Base64 decoding algorithm.

output

On output, this transformer encodes the data into binary values using a Base64 encoding algorithm.

Examples

Example 142. Encoding Value To and From File Example

This example uses *Base64Encode* on an inline value, writes the transformed value to a file, and reads the value from the file to the console.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="Ex1">
        <String value="Base64: " token="true"/>
        <String name="TransformMe" value="ABCDEFGH">
            <Transformer class="Base64Encode" />
        </String>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <!-- Encoded Output -->
```

```

<Action type="output" publisher="ConsolePub">
    <DataModel ref="Ex1" />
</Action>

<!-- Write Encoded Output to File -->
<Action type="output" publisher="FilePubWrite">
    <DataModel ref="Ex1" />
    <Data>
        <Field name="TransformMe" value="ABCDEFGH" />
    </Data>
</Action>

<Action type="close" publisher="FilePubWrite" />

<!-- Read and decode encoded file and slurp Base64 decoded string to
output in console -->
<Action type="input" publisher="FilePubRead" >
    <DataModel name="InputModel" ref="Ex1" />
</Action>

<Action type="slurp" valueXpath="//InputModel//TransformMe" setXpath=
"//OutputModel//StringValue" />

<Action type="output" publisher="ConsolePub">
    <DataModel name="OutputModel">
        <String name="StringValue" />
    </DataModel>
</Action>
</State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>
    <Publisher class="ConsoleHex" name="ConsolePub"/>

    <Publisher class="File" name="FilePubWrite">
        <Param name="FileName" value="encoded.bin" />
    </Publisher>

    <Publisher class="File" name="FilePubRead">
        <Param name="FileName" value="encoded.bin" />
        <Param name="Overwrite" value="false" />
    </Publisher>
</Test>
</Peach>

```

Output from this example.

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 11558.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(20 bytes)
00000000  42 61 73 65 36 34 3A 20  51 55 4A 44 52 45 56 47  Base64: QUJDREVG
00000010  52 30 67 3D                           R0g=
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(20 bytes)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Input
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher input()
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'InputModel' Bytes: 0/20, Bits: 0/160
Peach.Core.Cracker.DataCracker getSize: -----> DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: String 'InputModel.DataElement_0' -> Pos: 0,
Saving Token
Peach.Core.Cracker.DataCracker scan: String 'InputModel.DataElement_0' -> Pos: 64,
Length: 64
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: DataModel 'InputModel' Size: <null>, Bytes:
0/20, Bits: 0/160
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker String 'InputModel.DataElement_0' Bytes: 0/20, Bits:
0/160
Peach.Core.Cracker.DataCracker getSize: -----> String 'InputModel.DataElement_0'

Peach.Core.Cracker.DataCracker scan: String 'InputModel.DataElement_0' -> Pos: 0,
Saving Token
Peach.Core.Cracker.DataCracker scan: String 'InputModel.DataElement_0' -> Pos: 64,
Length: 64
Peach.Core.Cracker.DataCracker getSize: <----- Size: 64
```

```
Peach.Core.Cracker.DataCracker Crack: String 'InputModel.DataElement_0' Size: 64 ,  
Bytes: 0/20, Bits: 0/160  
Peach.Core.Dom.DataElement String 'InputModel.DataElement_0' value is: Base64:  
Peach.Core.Cracker.DataCracker -----  
Peach.Core.Cracker.DataCracker String 'InputModel.TransformMe' Bytes: 8/20, Bits :  
64/160  
Peach.Core.Cracker.DataCracker getSize: -----> String 'InputModel.TransformMe'  
Peach.Core.Cracker.DataCracker scan: String 'InputModel.TransformMe' -> Offset: 0,  
Unsized element  
Peach.Core.Cracker.DataCracker lookahead: String 'InputModel.TransformMe'  
Peach.Core.Cracker.DataCracker getSize: <----- Last Unsized: 96  
Peach.Core.Cracker.DataCracker Crack: String 'InputModel.TransformMe' Size: 64,  
Bytes: 0/8, Bits: 0/64  
Peach.Core.Dom.DataElement String 'InputModel.TransformMe' value is: ABCDEFGH  
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted  
Peach.Core.Dom.Action ActionType.Slurp  
Peach.Core.Dom.Action Slurp, setting OutputModel.StringValue from  
InputModel.TransformMe  
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted  
Peach.Core.Dom.Action ActionType.Output  
Peach.Core.Publishers.ConsolePublisher output(8 bytes)  
00000000  41 42 43 44 45 46 47 48                      ABCDEFGH  
Peach.Core.Publishers.ConsolePublisher close()  
Peach.Core.Publishers.FilePublisher close()  
Peach.Core.Engine runTest: context.config.singleIteration == true  
Peach.Core.Publishers.ConsolePublisher stop()  
Peach.Core.Publishers.FilePublisher stop()  
Peach.Core.Publishers.FilePublisher stop()  
  
[*] Test 'Default' finished.
```

21.6.5. Bz2Compress

Type

Compress

Bz2Compress performs a bzip2 compression on the value in the parent [DataModel](#).

The [Transformer](#) can bidirectionally compress data as well as decompress data.

Parameters

None.

Attributes

Required:

None.

Optional:

None.

Actions Supported

[input](#)

On input, this transformer decompresses the incoming data.

[output](#)

On output, this transformer compresses the outgoing data.

Examples

Example 143. Compression To and From File Example

This example uses Bz2Compress to compress a value, write the value to the console, write to a file, and then read it from the file.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

  <DataModel name="Ex1">
    <String name="TransformMe" value="shrinkmepleaseshrinkmeplease">
      <Transformer class="Bz2Compress" />
    </String>
  </DataModel>

  <StateModel name="TheState" initialState="initial">
    <State name="initial">
      <!-- Compressed Output -->
      <Action type="output" publisher="ConsolePub">
        <DataModel ref="Ex1" />
      </Action>

      <!-- Write Compressed Output to File -->
      <Action type="output" publisher="FilePubWrite">
        <DataModel ref="Ex1" />
        <Data>
          <Field name="TransformMe" value="shrinkmepleaseshrinkmeplease" />
        </Data>
      </Action>

      <Action type="close" publisher="FilePubWrite" />
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex" name="ConsolePub"/>

    <Publisher class="File" name="FilePubWrite">
      <Param name="FileName" value="compressed.bin" />
    </Publisher>
  </Test>
</Peach>

```

Output from this example.

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 47050.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(53 bytes)
00000000  42 5A 68 39 31 41 59 26  53 59 A7 83 53 78 00 00  BZh91AY&SY??Sx??
00000010  0D 81 80 22 6F 58 00 20  00 31 00 D3 4D 01 55 03  ???"oX? ?1??M?U?
00000020  41 EA 5D 4D A1 1E 44 47  51 18 7C 5D C9 14 E1 42  A?]M??DGQ?|]???B
00000030  42 9E 0D 4D E0                      B??M?

Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(53 bytes)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()
Peach.Core.Publishers.FilePublisher stop()

[*] Test 'Default' finished.
```

21.6.6. Bz2Decompress

Type

Compress

Bz2Decompress performs a bzip2 decompression on the value in the parent [DataModel](#).

The [Transformer](#) can bidirectionally compress data as well as decompress data.

Parameters

None.

Attributes

Required:

None.

Optional:

None.

Actions Supported

[input](#)

On input, this transformer compresses the incoming data.

[output](#)

On output, this transformer decompresses the outgoing data.

Examples

Example 144. Decompress From a Value To a File

This example uses Bz2Decompress to decompress a value, write the value to the console, write to a file, and then reads it from the file.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

  <DataModel name="Ex1">
    <Blob name="TransformMe" value="42 5A 68 39 31 41 59 26 53 59 A7 83 53 78 00
00 0D 81 80 22 6F 58 00 20 00 31 00 D3 4D 01 55 03 41 EA 5D 4D A1 1E 44 47 51 18 7C
5D C9 14 E1 42 42 9E 0D 4D E0" valueType="hex" >
```

```

        <Transformer class="Bz2Decompress" />
    </Blob>
</DataModel>

<StateModel name="TheState" initialState="initial">
    <State name="initial">
        <!-- Decompressed Output -->
        <Action type="output" publisher="ConsolePub">
            <DataModel ref="Ex1" />
        </Action>

        <!-- Write Decompressed Output to File -->
        <Action type="output" publisher="FilePubWrite">
            <DataModel ref="Ex1" />
        </Action>

        <Action type="close" publisher="FilePubWrite" />

        <!-- Read and decompress from file and slurp output to console -->
        <Action type="input" publisher="FilePubRead" >
            <DataModel name="InputModel" ref="Ex1" />
        </Action>

        <Action type="slurp" valueXpath="//InputModel//TransformMe" setXpath=
"//OutputModel//BlobValue" />

        <Action type="output" publisher="ConsolePub">
            <DataModel name="OutputModel">
                <Blob name="BlobValue" />
            </DataModel>
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex" name="ConsolePub"/>

    <Publisher class="File" name="FilePubWrite">
        <Param name="FileName" value="decompressed.bin" />
    </Publisher>

    <Publisher class="File" name="FilePubRead">
        <Param name="FileName" value="decompressed.bin" />
        <Param name="Overwrite" value="false" />
    </Publisher>
</Test>

```

Output from this example.

```
> peach -l --debug example.xml

[*] Test 'Default' starting with random seed 19925.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(28 bytes)
00000000 73 68 72 69 6E 6B 6D 65 70 6C 65 61 73 65 73 68 shrinkmepleasesh
00000010 72 69 6E 6B 6D 65 70 6C 65 61 73 65 rinkmeplease
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(28 bytes)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Input
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher input()
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataModel 'InputModel' Bytes: 0/28, Bits: 0/224
Peach.Core.Cracker.getSize: -----> DataModel 'InputModel'
Peach.Core.Cracker.scan: DataModel 'InputModel'
Peach.Core.Cracker.scan: Blob 'InputModel.TransformMe' -> Offset: 0,
Unsized element
Peach.Core.Cracker.getSize: <----- Deterministic: ???
Peach.Core.Cracker.Crack: DataModel 'InputModel' Size: <null>, Bytes
: 0/28, Bits: 0/224
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.Blob 'InputModel.TransformMe' Bytes: 0/28, Bits:
0/224
Peach.Core.Cracker.getSize: -----> Blob 'InputModel.TransformMe'
Peach.Core.Cracker.scan: Blob 'InputModel.TransformMe' -> Offset: 0,
Unsized element
Peach.Core.Cracker.lookahead: Blob 'InputModel.TransformMe'
Peach.Core.Cracker.getSize: <----- Last Unsized: 224
```

```
Peach.Core.Cracker.DataCracker Crack: Blob 'InputModel.TransformMe' Size: 424, Bytes: 0/53, Bits: 0/424
Peach.Core.Dom.DataElement Blob 'InputModel.TransformMe' value is: 42 5a 68 39 3
1 41 59 26 53 59 a7 83 53 78 00 00 0d 81 80 22 6f 58 00 20 00 31 00 d3 4d 01 55
03.. (Len: 53 bytes)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Slurp
Peach.Core.Dom.Action Slurp, setting OutputModel.BlobValue from InputModel.TransformMe
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(53 bytes)
00000000 42 5A 68 39 31 41 59 26 53 59 A7 83 53 78 00 00 BZh91AY&SY??Sx??
00000010 0D 81 80 22 6F 58 00 20 00 31 00 D3 4D 01 55 03 ???"oX? ?1??M?U?
00000020 41 EA 5D 4D A1 1E 44 47 51 18 7C 5D C9 14 E1 42 A?]M??DGQ?|]???B
00000030 42 9E 0D 4D E0 B??M?

Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()
Peach.Core.Publishers.FilePublisher stop()
Peach.Core.Publishers.FilePublisher stop()

[*] Test 'Default' finished.
```

21.6.7. Des

Type

Cryptography

Des transforms the parent data element by encrypting the data using DES with the provided Key and Initialization Vector.

The [Transformer](#) can bidirectionally encrypt and decrypt data.

Parameters

Required:

Key

User provided symmetric key that is used to encrypt the value. Must be a hex string representation of a 16 byte key.

IV

User provided initialization vector that is used as the first block for the AES128 operation. Must be a hex string representation of an 16 byte value.

Optional:

CipherMode

Cipher mode to use during encryption. Defaults to CBC.

- CBC
- ECB
- CFB
- CTS
- OFB

PaddingMode

Padding mode. Defaults to Zeros.

- Zeros
- None
- PKCS7
- ANSIX923
- ISO101026

Actions Supported

input

On input, this transformer decrypts the incoming data.

output

On output, this transformer encrypts the outgoing data.

Examples

Example 145. Simple Encrypt and Decrypt Example

This example uses DES on an inline value, writes the transformed value to a file, and reads the value from the file to the console.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="Ex1">
        <String name="TransformMe" value="supersupersecret" >
            <Transformer class="Des">
                <Param name="Key" value="ae 12 34 56 78 90 ae af"/>
                <Param name="IV"  value="ae ae ae ae ae ae ae"/>
            </Transformer>
        </String>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <!-- Encrypted Output -->
            <Action type="output" publisher="ConsolePub">
                <DataModel ref="Ex1" />
            </Action>

            <!-- Write Encrypted Output to File -->
            <Action type="output" publisher="FilePubWrite">
                <DataModel ref="Ex1" />
            </Action>

            <Action type="close" publisher="FilePubWrite" />
        <!-- Read and decrypt encrypted file and slurp output to console -->
        <Action type="input" publisher="FilePubRead" >
            <DataModel name="InputModel" ref="Ex1" />
        </Action>
    </StateModel>
</Peach>
```

```

<Action type="slurp" valueXpath="//InputModel//TransformMe" setXpath=
"/OutputModel/StringValue" />

<Action type="output" publisher="ConsolePub">
    <DataModel name="OutputModel">
        <String name="StringValue" />
    </DataModel>
</Action>
</State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex" name="ConsolePub"/>

    <Publisher class="File" name="FilePubWrite">
        <Param name="FileName" value="encrypted.bin" />
    </Publisher>

    <Publisher class="File" name="FilePubRead">
        <Param name="FileName" value="encrypted.bin" />
        <Param name="Overwrite" value="false" />
    </Publisher>
</Test>
</Peach>

```

Output from this example.

```

> peach -1 --debug example.xml

[*] Web site running at: http://10.0.1.87:8888/

[*] Test 'Default' starting with random seed 56253.
2016-07-08 17:27:30.8520 Peach.Core.Loggers.JobLogger Writing debug.log to:
c:\peach-pro\output\win_x64_debug\bin\Logs\example.xml_20160708172729\debug.log

[R1,-,-] Performing iteration
2016-07-08 17:27:30.9893 Peach.Core.Engine runTest: Performing control recording
iteration.
2016-07-08 17:27:31.0193 Peach.Core.Dom.StateModel Run(): Changing to state
"initial".
2016-07-08 17:27:31.0193 Peach.Core.Dom.Action Run(Action): Output
2016-07-08 17:27:31.1414 Peach.Core.Publisher start()
2016-07-08 17:27:31.1414 Peach.Core.Publisher open()
2016-07-08 17:27:31.1414 Peach.Core.Publisher output(16 bytes)

```

```

00000000 FF 84 0E BF DC 49 70 56 86 80 8A C3 16 61 4F E5 ....IpV.....a0. ①
2016-07-08 17:27:31.1414 Peach.Core.Dom.Action Run(Action_1): Output
2016-07-08 17:27:31.1414 Peach.Pro.Core.Publishers.FilePublisher start()
2016-07-08 17:27:31.1414 Peach.Pro.Core.Publishers.FilePublisher open()
2016-07-08 17:27:31.1544 Peach.Pro.Core.Publishers.FilePublisher output(16 bytes)
2016-07-08 17:27:31.1544 Peach.Core.Dom.Action Run(Action_2): Close
2016-07-08 17:27:31.1544 Peach.Pro.Core.Publishers.FilePublisher close()
2016-07-08 17:27:31.1544 Peach.Core.Dom.Action Run(Action_3): Input
2016-07-08 17:27:31.1544 Peach.Pro.Core.Publishers.FilePublisher start()
2016-07-08 17:27:31.1544 Peach.Pro.Core.Publishers.FilePublisher open()
2016-07-08 17:27:31.1544 Peach.Pro.Core.Publishers.FilePublisher input()
2016-07-08 17:27:31.1704 DataCracker -+ DataModel 'InputModel', Bytes: 0/16, Bits: 0/128
2016-07-08 17:27:31.1704 DataCracker | Size: ??? (Deterministic)
2016-07-08 17:27:31.1704 DataCracker |-- String 'TransformMe', Bytes: 0/16, Bits: 0/128
2016-07-08 17:27:31.1854 DataCracker | Size: 16 bytes | 128 bits (Last Unsized)
2016-07-08 17:27:31.1854 DataCracker | Value: supersupersecret
2016-07-08 17:27:31.1854 DataCracker /
2016-07-08 17:27:31.1854 Peach.Core.Dom.Actions.Input Final pos: 16 length: 16 crack consumed: 16 bytes
2016-07-08 17:27:31.1854 Peach.Core.Dom.Action Run(Action_4): Slurp
2016-07-08 17:27:31.2064 Peach.Core.Dom.Actions.Slurp Slurp, setting OutputModel.StringValue from InputModel.TransformMe
2016-07-08 17:27:31.2064 Peach.Core.Dom.Action Run(Action_5): Output
2016-07-08 17:27:31.2064 Peach.Pro.Core.Publishers.ConsolePublisher output(16 bytes)
00000000 73 75 70 65 72 73 75 70 65 72 73 65 63 72 65 74 supersupersecret ②
2016-07-08 17:27:31.2064 Peach.Pro.Core.Publishers.ConsolePublisher close()
2016-07-08 17:27:31.2064 Peach.Pro.Core.Publishers.FilePublisher close()
2016-07-08 17:27:31.2174 Peach.Core.Engine runTest: context.config.singleIteration == true
2016-07-08 17:27:31.2174 Peach.Pro.Core.Publishers.ConsolePublisher stop()
2016-07-08 17:27:31.2174 Peach.Pro.Core.Publishers.FilePublisher stop()
2016-07-08 17:27:31.2174 Peach.Pro.Core.Publishers.FilePublisher stop()
2016-07-08 17:27:31.2174 Peach.Core.Engine EndTest: Stopping all agents and monitors

[*] Test 'Default' finished.

```

① Encrypted output

② Decrypted output

21.6.8. GzipCompress

Type

Compress

GzipCompress performs a Gzip compression on the value in the parent [DataModel](#).

The [Transformer](#) can bidirectionally compress data as well as decompress data.

Parameters

None.

Attributes

Required:

None.

Optional:

None.

Actions Supported

[input](#)

On input, *GzipCompress* decompresses incoming data.

[output](#)

On output, *GzipCompress* compresses all outgoing data.

Examples

Example 146. Compression To and From File Example

This example uses *GzipCompress* to compress a value, write the value to the console, write to a file, and then reads it from the file to the console.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

<DataModel name="Ex1">
    <String name="TransformMe" value="shrinkmepleaseshrinkmeplease">
        <Transformer class="GzipCompress" />
    </String>
```

```

</DataModel>

<StateModel name="TheState" initialState="initial">
    <State name="initial">
        <!-- Compressed Output -->
        <Action type="output" publisher="ConsolePub">
            <DataModel ref="Ex1" />
        </Action>

        <!-- Write Compressed Output to File -->
        <Action type="output" publisher="FilePubWrite">
            <DataModel ref="Ex1" />
            <Data>
                <Field name="TransformMe" value="shrinkmepleaseshrinkmeplease" />
            </Data>
        </Action>

        <Action type="close" publisher="FilePubWrite" />

        <!-- Read and compress file and slurp output to console -->
        <Action type="input" publisher="FilePubRead" >
            <DataModel name="InputModel" ref="Ex1" />
        </Action>

        <Action type="slurp" valueXpath="//InputModel//TransformMe" setXpath=
        "//OutputModel//StringValue" />

        <Action type="output" publisher="ConsolePub">
            <DataModel name="OutputModel">
                <String name="StringValue" />
            </DataModel>
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex" name="ConsolePub"/>

    <Publisher class="File" name="FilePubWrite">
        <Param name="FileName" value="encrypted.bin" />
    </Publisher>

    <Publisher class="File" name="FilePubRead">
        <Param name="FileName" value="encrypted.bin" />
        <Param name="Overwrite" value="false" />

```

```
</Publisher>
</Test>
</Peach>
```

Output from this example.

```
> peach -l --debug example.xml

[*] Test 'Default' starting with random seed 62862.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(37 bytes)
00000000  1F 8B 08 00 00 00 00 04 00 2B CE 28 CA CC CB  ??????????+?(???
00000010  CE 4D 2D C8 49 4D 2C 4E  2D 46 E1 01 00 32 B6 7E  ?M-?IM,N-F????2?~
00000020  40 1C 00 00 00                                     @?????

Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(37 bytes)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Input
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher input()
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataModel 'InputModel' Bytes: 0/37, Bits: 0/296
Peach.Core.Cracker.DataCracker getSize: -----> DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: String 'InputModel.TransformMe' -> Offset: 0,
Unsized element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: DataModel 'InputModel' Size: <null>, Bytes :
0/37, Bits: 0/296
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker String 'InputModel.TransformMe' Bytes: 0/37, Bits :
0/296
Peach.Core.Cracker.DataCracker getSize: -----> String 'InputModel.TransformMe'
Peach.Core.Cracker.DataCracker scan: String 'InputModel.TransformMe' -> Offset: 0,
```

Unsized element

```
Peach.Core.Cracker.lookahead: String 'InputModel.TransformMe'  
Peach.Core.Cracker.getSize: <----- Last Unsized: 296  
Peach.Core.Cracker.Crack: String 'InputModel.TransformMe' Size: 224,  
Bytes: 0/28, Bits: 0/224  
Peach.Core.Dom.DataElement String 'InputModel.TransformMe' value is:  
shrinkmepleaseshrinkmeplease  
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted  
Peach.Core.Dom.Action ActionType.Slurp  
Peach.Core.Dom.Action Slurp, setting OutputModel.StringValue from  
InputModel.TransformMe  
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted  
Peach.Core.Dom.Action ActionType.Output  
Peach.Core.Publishers.ConsolePublisher output(28 bytes)  
00000000 73 68 72 69 6E 6B 6D 65 70 6C 65 61 73 65 73 68 shrinkmepleasesh  
00000010 72 69 6E 6B 6D 65 70 6C 65 61 73 65 rinkmeplease  
Peach.Core.Publishers.ConsolePublisher close()  
Peach.Core.Publishers.FilePublisher close()  
Peach.Core.Engine runTest: context.config.singleIteration == true  
Peach.Core.Publishers.ConsolePublisher stop()  
Peach.Core.Publishers.FilePublisher stop()  
Peach.Core.Publishers.FilePublisher stop()
```

[*] Test 'Default' finished.

21.6.9. GzipDecompress

Type

Compress

GzipDecompress performs a gzip decompression on the value in the parent [DataModel](#).

The [Transformer](#) can bidirectionally compress data as well as decompress data.

Parameters

None.

Attributes

Required:

None.

Optional:

None.

Actions Supported

[input](#)

On input, *GzipDecompress* compresses incoming data.

[output](#)

On output, *GzipDecompress* decompresses all outgoing data.

Examples

Example 147. Compression To and From File Example

This example uses *GzipCompress* to compress a value, write the value to the console, write to a file, and then reads it from the file to the console.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

  <DataModel name="Ex1">
    <Blob name="TransformMe" value="1F 8B 08 00 00 00 00 00 04 00 EC BD 07 60 1C
49 96 25 26 2F 6D CA 7B 7F 4A F5 4A D7 E0 74 A1 08 80 60 13 24 D8 90 40 10 EC C1 88
CD E6 92 EC 1D 69 47 23 29 AB 2A 81 CA 65 56 65 5D 66 16 40 CC ED 9D BC F7 DE 7B EF"
```

```

BD F7 DE 7B EF BD F7 BA 3B 9D 4E 27 F7 DF FF 3F 5C 66 64 01 6C F6 CE 4A DA C9 9E 21
80 AA C8 1F 3F 7E 7C 1F 3F 22 9A 79 5D 2C DF 2E F2 55 99 67 4D 1E FE F5 FF 04 00 00
FF FF 32 B6 7E 40 1C 00 00 00" valueType="hex">
    <Transformer class="GzipDecompress" />
</Blob>
</DataModel>

<StateModel name="TheState" initialState="initial">
    <State name="initial">
        <!-- Decompressed Output -->
        <Action type="output" publisher="ConsolePub">
            <DataModel ref="Ex1" />
        </Action>

        <!-- Write Decompressed Output to File -->
        <Action type="output" publisher="FilePubWrite">
            <DataModel ref="Ex1" />
        </Action>

        <Action type="close" publisher="FilePubWrite" />

        <!-- Read and decompress file and slurp output to console -->
        <Action type="input" publisher="FilePubRead" >
            <DataModel name="InputModel" ref="Ex1" />
        </Action>

        <Action type="slurp" valueXpath="//InputModel//TransformMe" setXpath=
"//OutputModel//BlobValue" />

        <Action type="output" publisher="ConsolePub">
            <DataModel name="OutputModel">
                <Blob name="BlobValue" />
            </DataModel>
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex" name="ConsolePub"/>

    <Publisher class="File" name="FilePubWrite">
        <Param name="FileName" value="decompressed.bin" />
    </Publisher>

    <Publisher class="File" name="FilePubRead">
        <Param name="FileName" value="decompressed.bin" />

```

```

    <Param name="Overwrite" value="false" />
  </Publisher>
</Test>
</Peach>

```

Output from this example.

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 52916.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(28 bytes)
00000000  73 68 72 69 6E 6B 6D 65  70 6C 65 61 73 65 73 68  shrinkmepleasesh
00000010  72 69 6E 6B 6D 65 70 6C  65 61 73 65           rinkmeplease
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(28 bytes)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Input
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher input()
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataModel 'InputModel' Bytes: 0/28, Bits: 0/224
Peach.Core.Cracker.getSize: -----> DataModel 'InputModel'
Peach.Core.Cracker.scan: DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: Blob 'InputModel.TransformMe' -> Offset: 0
  Unsized element
Peach.Core.Cracker.getSize: -----> Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: DataModel 'InputModel' Size: <null>, Byte
: 0/28, Bits: 0/224
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker Blob 'InputModel.TransformMe' Bytes: 0/28, Bits:
0/224
Peach.Core.Cracker.getSize: -----> Blob 'InputModel.TransformMe'
Peach.Core.Cracker.DataCracker scan: Blob 'InputModel.TransformMe' -> Offset: 0

```

```
Unsized element
Peach.Core.Cracker.DataCracker lookahead: Blob 'InputModel.TransformMe'
Peach.Core.Cracker.getSize: <----- Last Unsized: 224
Peach.Core.Cracker Crack: Blob 'InputModel.TransformMe' Size: 296,
bytes: 0/37, Bits: 0/296
Peach.Core.Dom.DataElement Blob 'InputModel.TransformMe' value is: 1f 8b 08 00
0 00 00 00 04 00 2b ce 28 ca cc cb ce 4d 2d c8 49 4d 2c 4e 2d 46 e1 01 00 32 b6
7e.. (Len: 37 bytes)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Slurp
Peach.Core.Dom.Action Slurp, setting OutputModel.BlobValue from InputModel.TransformMe
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(37 bytes)
00000000  1F 8B 08 00 00 00 00 00 04 00 2B CE 28 CA CC CB  ??????????+?(???
00000010  CE 4D 2D C8 49 4D 2C 4E 2D 46 E1 01 00 32 B6 7E  ?M-?IM,N-F???2?~
00000020  40 1C 00 00 00                                     @?????

Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()
Peach.Core.Publishers.FilePublisher stop()
Peach.Core.Publishers.FilePublisher stop()

[*] Test 'Default' finished.
```

21.6.10. Hex

Type

Encoder/Decoder

HexTransformer takes the provided value and produces an ASCII string of hex characters that represent the data.

The [Transformer](#) can bidirectionally transform this data from hex to string.

Parameters

None.

Attributes

None.

Actions Supported

input

On input, HexTransformer converts a hex string into ASCII for the incoming data.

output

On output, HexTransformer converts ASCII into a hex string for the outgoing data.

Examples

Example 148. Simple Console Example

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

<DataModel name="Ex1">
    <String name="TransformMe" value="ABCDEF">
        <Transformer class="Hex" />
    </String>
</DataModel>

<StateModel name="TheState" initialState="initial">
    <State name="initial">
        <!-- Encoded Output -->
        <Action type="output" publisher="ConsolePub">
            <DataModel ref="Ex1" />
        </Action>
    </State>
</StateModel>
```

```

<!-- Write Encoded Output to File -->
<Action type="output" publisher="FilePubWrite">
    <DataModel ref="Ex1" />
    <Data>
        <Field name="TransformMe" value="ABCDEF" />
    </Data>
</Action>

<Action type="close" publisher="FilePubWrite" />

<!-- Read and decode encoded file and slurp decoded string to output in
console -->
<Action type="input" publisher="FilePubRead" >
    <DataModel name="InputModel" ref="Ex1" />
</Action>

<Action type="slurp" valueXpath="//InputModel//TransformMe" setXpath=
"/OutputModel/StringValue" />

<Action type="output" publisher="ConsolePub">
    <DataModel name="OutputModel">
        <String name="StringValue" />
    </DataModel>
</Action>
</State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex" name="ConsolePub"/>

    <Publisher class="File" name="FilePubWrite">
        <Param name="FileName" value="encoded.bin" />
    </Publisher>

    <Publisher class="File" name="FilePubRead">
        <Param name="FileName" value="encoded.bin" />
        <Param name="Overwrite" value="false" />
    </Publisher>
</Test>
</Peach>

```

Output from this example.

```
> peach -1 --debug example.xml
```

```

[*] Test 'Default' starting with random seed 47130.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(12 bytes)
00000000 34 31 34 32 34 33 34 34 34 35 34 36           414243444546
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(12 bytes)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Input
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher input()
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'InputModel' Bytes: 0/12, Bits: 0/96
Peach.Core.Cracker.DataCracker getSize: -----> DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: String 'InputModel.TransformMe' -> Offset: 0,
Unsized element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: DataModel 'InputModel' Size: <null>, Bytes :
0/12, Bits: 0/96
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker String 'InputModel.TransformMe' Bytes: 0/12, Bits :
0/96
Peach.Core.Cracker.DataCracker getSize: -----> String 'InputModel.TransformMe'
Peach.Core.Cracker.DataCracker scan: String 'InputModel.TransformMe' -> Offset: 0,
Unsized element
Peach.Core.Cracker.DataCracker lookahead: String 'InputModel.TransformMe'
Peach.Core.Cracker.DataCracker getSize: <----- Last Unsized: 96
Peach.Core.Cracker.DataCracker Crack: String 'InputModel.TransformMe' Size: 48,
Bytes: 0/6, Bits: 0/48
Peach.Core.Dom.DataElement String 'InputModel.TransformMe' value is: ABCDEF
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Slurp
Peach.Core.Dom.Action Slurp, setting OutputModel.StringValue from

```

```
InputModel.TransformMe
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(6 bytes)
00000000  41 42 43 44 45 46                      ABCDEF
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()
Peach.Core.Publishers.FilePublisher stop()
Peach.Core.Publishers.FilePublisher stop()

[*] Test 'Default' finished.
```

21.6.11. Hmac

Type

Cryptography

HMAC produces an HMAC hash on the value in the parent [DataModel](#) as described in RFC 2104.

This [Transformer](#) can only be applied to outgoing data because hashes are one-way operations.

Parameters

None.

Attributes

None.

Actions Supported

output

On output, this transformer hashes the outgoing data.

Examples

Example 149. Simple Encode Example

This example uses HMAC on an inline value, writes the transformed value to a file, and reads the value from the file to the console.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="Ex1">
        <Block name="Main">
            <String name="TransformMe" value="superdoopersecret">
                <Transformer class="Hmac" />
            </String>
        </Block>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <!-- Encrypted Output -->
            <Action type="output" publisher="ConsolePub">
                <DataModel ref="Ex1" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>
        <Publisher class="ConsoleHex" name="ConsolePub"/>
    </Test>
</Peach>

```

Output from this example.

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 42451.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(20 bytes)
00000000  DF 1B FE 8F 8A 44 D5 87  33 AE DD 18 2A 90 D2 34  ?????D??3???*??4
00000010  7F 7E FD B9                               ~??
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

21.6.12. HtmlDecode

Type

Encoder/Decoder

HtmlDecode decodes any HTML/XML-encoded characters in the string in the parent [DataModel](#). The following characters are decoded: less than (<), ampersand (&), apostrophe ('), and quotation mark (").

The [Transformer](#) can bidirectionally encode data as well as decode data.

Parameters

None.

Attributes

None.

Actions Supported

input

On input, this transformer encodes characters to their HTML-encoded character sequences for incoming data.

output

On output, this transformer decodes HTML-encoded character sequences to single-character representations for outgoing data.

Examples

Example 150. Simple Encode Example

This example uses *HtmlDecode* on an inline value, writes the transformed value to a file, and reads the value from the file to the console.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="Ex1">
        <String name="TransformMe" value="&lt;XML&gt;">
            <Transformer class="HtmlDecode" />
        </String>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
```

```

<State name="initial">
    <!-- Decoded HTML Output -->
    <Action type="output" publisher="ConsolePub">
        <DataModel ref="Ex1" />
    </Action>

    <!-- Write Decoded Output to File -->
    <Action type="output" publisher="FilePubWrite">
        <DataModel ref="Ex1" />
        <Data>
            <Field name="TransformMe" value="&lt;XML&gt;" />
        </Data>
    </Action>

    <Action type="close" publisher="FilePubWrite" />

    <!-- Read and Encode file and slurp the HTML to output in console -->
    <!-- Only the special character '<' will be encoded -->
    <Action type="input" publisher="FilePubRead" >
        <DataModel name="InputModel" ref="Ex1" />
    </Action>

    <Action type="slurp" valueXpath="//InputModel//TransformMe" setXpath=
    "//OutputModel//StringValue" />

    <Action type="output" publisher="ConsolePub">
        <DataModel name="OutputModel">
            <String name="StringValue" />
        </DataModel>
    </Action>
</State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex" name="ConsolePub"/>

    <Publisher class="File" name="FilePubWrite">
        <Param name="FileName" value="decoded.bin" />
    </Publisher>

    <Publisher class="File" name="FilePubRead">
        <Param name="FileName" value="decoded.bin" />
        <Param name="Overwrite" value="false" />
    </Publisher>
</Test>
</Peach>

```

Output from this example.

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 63821.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(5 bytes)
00000000  3C 58 4D 4C 3E                                <XML>
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(5 bytes)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Input
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher input()
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'InputModel' Bytes: 0/5, Bits: 0/40
Peach.Core.Cracker.DataCracker getSize: -----> DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: String 'InputModel.TransformMe' -> Offset: 0,
Unsized element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: DataModel 'InputModel' Size: <null>, Bytes :
0/5, Bits: 0/40
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker String 'InputModel.TransformMe' Bytes: 0/5, Bits: 0/40
Peach.Core.Cracker.DataCracker getSize: -----> String 'InputModel.TransformMe'
Peach.Core.Cracker.DataCracker scan: String 'InputModel.TransformMe' -> Offset: 0,
Unsized element
Peach.Core.Cracker.DataCracker lookahead: String 'InputModel.TransformMe'
Peach.Core.Cracker.DataCracker getSize: <----- Last Unsized: 40
Peach.Core.Cracker.DataCracker Crack: String 'InputModel.TransformMe' Size: 64,
Bytes: 0/8, Bits: 0/64
Peach.Core.Dom.DataElement String 'InputModel.TransformMe' value is: &lt;XML>
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
```

```
Peach.Core.Dom.Action ActionType.Slurp
Peach.Core.Dom.Action Slurp, setting OutputModel.StringValue from
InputModel.TransformMe
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(8 bytes)
00000000  26 6C 74 3B 58 4D 4C 3E          &lt;XML>
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()
Peach.Core.Publishers.FilePublisher stop()
Peach.Core.Publishers.FilePublisher stop()
```

[*] Test 'Default' finished.

21.6.13. HtmlEncode

Type

Encoder/Decoder

HtmlEncode produce an HTML/XML-encoded string of the provided value in the parent [DataModel](#). The following characters are encoded: less than (<), ampersand (&), apostrophe ('), and quotation mark (").

The [Transformer](#) can bidirectionally encode data as well as decode data.

Parameters

None.

Attributes

None.

Actions Supported

input

On input, *HtmlEncode* decodes HTML-encoded character sequences to single-character representations for incoming data.

output

On output, *HtmlEncode* encodes characters to their HTML-encoded character sequences for outgoing data.

Examples

Example 151. Simple Encode Example

This example uses *HtmlEncode* on an inline value, writes the transformed value to a file, and reads the value from the file to the console.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="Ex1">
        <String value="&lt;Body&gt; " token="true"/>
        <String name="TransformMe" value="These are encoded: &lt; && quot; ' ' These are not: &gt; !$2/\\"c*">
            <Transformer class="HtmlEncode" />
        </String>
        <String value=" &lt;/Body&gt;" token="true"/>
```

```

</DataModel>

<StateModel name="TheState" initialState="initial">
    <State name="initial">
        <!-- Encoded Output -->
        <Action type="output" publisher="ConsolePub">
            <DataModel ref="Ex1" />
        </Action>

        <!-- Write Encoded Output to File -->
        <Action type="output" publisher="FilePubWrite">
            <DataModel ref="Ex1" />
            <Data>
                <Field name="TransformMe" value="These are encoded: &lt;, &gt;,
&amp;, &quot;\nThese are not: '!$2/\\"c*' />
            </Data>
        </Action>

        <Action type="close" publisher="FilePubWrite" />

        <!-- Read and decode encoded file and slurp the value to output in
console -->
        <Action type="input" publisher="FilePubRead" >
            <DataModel name="InputModel" ref="Ex1" />
        </Action>

        <Action type="slurp" valueXpath="//InputModel//TransformMe" setXpath=
"//OutputModel//StringValue" />

        <Action type="output" publisher="ConsolePub">
            <DataModel name="OutputModel">
                <String name="StringValue" />
            </DataModel>
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex" name="ConsolePub"/>

    <Publisher class="File" name="FilePubWrite">
        <Param name="FileName" value="encoded.bin" />
    </Publisher>

    <Publisher class="File" name="FilePubRead">

```

```

        <Param name="FileName" value="encoded.bin" />
        <Param name="Overwrite" value="false" />
    </Publisher>
</Test>
</Peach>

```

Output from this example.

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 52076.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(83 bytes)
00000000  3C 42 6F 64 79 3E 20 54  68 65 73 65 20 61 72 65  <Body> These are
00000010  20 65 6E 63 6F 64 65 64  3A 20 26 6C 74 3B 20 26  encoded: &lt; &
00000020  61 6D 70 3B 20 26 71 75  6F 74 3B 20 26 23 33 39  amp; &quot; &#39
00000030  3B 20 20 20 54 68 65 73  65 20 61 72 65 20 6E 6F  ; These are no
00000040  74 3A 20 3E 21 24 32 2F  5C 63 2A 20 3C 2F 42 6F  t: >!$2/\c* </Bo
00000050  64 79 3E                               dy>

Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(84 bytes)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Input
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher input()
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataModel 'InputModel' Bytes: 0/84, Bits: 0/672
Peach.Core.Cracker.DataCracker getSize: -----> DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: String 'InputModel.DataElement_0' -> Pos: 0,
Saving Token
Peach.Core.Cracker.DataCracker scan: String 'InputModel.DataElement_0' -> Pos: 56,
Length: 56
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
```

```
Peach.Core.Cracker.DataCracker Crack: DataModel 'InputModel' Size: <null>, Bytes :  
0/84, Bits: 0/672  
Peach.Core.Cracker.DataCracker -----  
Peach.Core.Cracker.String 'InputModel.DataElement_0' Bytes: 0/84, Bits:  
0/672  
Peach.Core.Cracker.DataCracker getSize: -----> String 'InputModel.DataElement_0'  
  
Peach.Core.Cracker.DataCracker scan: String 'InputModel.DataElement_0' -> Pos: 0,  
Saving Token  
Peach.Core.Cracker.DataCracker scan: String 'InputModel.DataElement_0' -> Pos: 56,  
Length: 56  
Peach.Core.Cracker.DataCracker getSize: <----- Size: 56  
Peach.Core.Cracker.DataCracker Crack: String 'InputModel.DataElement_0' Size: 56,  
Bytes: 0/84, Bits: 0/672  
Peach.Core.Dom.DataElement String 'InputModel.DataElement_0' value is: <Body>  
Peach.Core.Cracker.DataCracker -----  
Peach.Core.Cracker.DataCracker String 'InputModel.TransformMe' Bytes: 7/84, Bits :  
56/672  
Peach.Core.Cracker.DataCracker getSize: -----> String 'InputModel.TransformMe'  
Peach.Core.Cracker.DataCracker scan: String 'InputModel.TransformMe' -> Offset:  
0, Unsized element  
Peach.Core.Cracker.DataCracker lookahead: String 'InputModel.TransformMe'  
Peach.Core.Cracker.DataCracker scan: String 'InputModel.DataElement_1' -> Pos: 0,  
Saving Token  
Peach.Core.Cracker.DataCracker scan: String 'InputModel.DataElement_1' -> Pos: 64,  
Length: 64  
Peach.Core.Cracker.DataCracker getSize: <----- Required Token: 552  
Peach.Core.Cracker.DataCracker Crack: String 'InputModel.TransformMe' Size: 424,  
Bytes: 0/53, Bits: 0/424  
Peach.Core.Dom.DataElement String 'InputModel.TransformMe' value is: These are  
encoded: <, >, &, "  
These are not: '!$2/\c*'  
Peach.Core.Cracker.DataCracker -----  
Peach.Core.Cracker.DataCracker String 'InputModel.DataElement_1' Bytes: 76/84, Bits:  
608/672  
Peach.Core.Cracker.DataCracker getSize: -----> String 'InputModel.DataElement_1'  
  
Peach.Core.Cracker.DataCracker scan: String 'InputModel.DataElement_1' -> Pos: 0,  
Saving Token  
Peach.Core.Cracker.DataCracker scan: String 'InputModel.DataElement_1' -> Pos: 64,  
Length: 64  
Peach.Core.Cracker.DataCracker getSize: <----- Size: 64  
Peach.Core.Cracker.DataCracker Crack: String 'InputModel.DataElement_1' Size: 64,  
Bytes: 76/84, Bits: 608/672  
Peach.Core.Dom.DataElement String 'InputModel.DataElement_1' value is: </Body>  
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted  
Peach.Core.Dom.Action ActionType.Slurp  
Peach.Core.Dom.Action Slurp, setting OutputModel.StringValue from
```

```
InputModel.TransformMe
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(53 bytes)
00000000  54 68 65 73 65 20 61 72  65 20 65 6E 63 6F 64 65 These are encode
00000010  64 3A 20 3C 2C 20 3E 2C  20 26 2C 20 22 0A 54 68 d: <, >, &, "?Th
00000020  65 73 65 20 61 72 65 20  6E 6F 74 3A 20 27 21 24 ese are not: '!$2/\c*
00000030  32 2F 5C 63 2A

Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()
Peach.Core.Publishers.FilePublisher stop()
Peach.Core.Publishers.FilePublisher stop()

[*] Test 'Default' finished.
```

21.6.14. IntToHex

Type

Encoder/Decoder

IntToHex takes a string representation of an integer and produces an ASCII string of hex characters that represent that integer.

The [Transformer](#) can bidirectionally transform this data from hex to integer.

Parameters

Required:

None.

Optional:

FormatString

A format string in the Microsoft.NET format. Changing the format string allows controlling how the number is converted to hex. For example a value of `X2` would produce a two digit hex number (00, 01, .., FF). Defaults to `X`.

For additional information see [Standard Numeric Format Strings](#)

Attributes

None.

Actions Supported

input

On input, *IntToHex* converts a hex string representation of a number into ASCII for the incoming data.

output

On output, *IntToHex* converts an ASCII number into a hex string for the outgoing data.

Examples

Example 152. Basic Usage Example

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">
```

```

<DataModel name="Ex1">
    <String name="TransformMe" value="1234">
        <Transformer class="IntToHex" />
    </String>
</DataModel>

<StateModel name="TheState" initialState="initial">
    <State name="initial">
        <!-- Encoded Output -->
        <Action type="output" publisher="ConsolePub">
            <DataModel ref="Ex1" />
        </Action>

        <!-- Write Encoded Output to File -->
        <Action type="output" publisher="FilePubWrite">
            <DataModel ref="Ex1" />
            <Data>
                <Field name="TransformMe" value="1234" />
            </Data>
        </Action>

        <Action type="close" publisher="FilePubWrite" />

        <!-- Read and decode encoded file and slurp decoded string to output in
console -->
        <Action type="input" publisher="FilePubRead" >
            <DataModel name="InputModel" ref="Ex1" />
        </Action>

        <Action type="slurp" valueXpath="//InputModel//TransformMe" setXpath=
"//OutputModel//StringValue" />

        <Action type="output" publisher="ConsolePub">
            <DataModel name="OutputModel">
                <String name="StringValue" />
            </DataModel>
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex" name="ConsolePub"/>

    <Publisher class="File" name="FilePubWrite">
        <Param name="FileName" value="encrypted.bin" />
    </Publisher>

```

```

<Publisher class="File" name="FilePubRead">
    <Param name="FileName" value="encrypted.bin" />
    <Param name="Overwrite" value="false" />
</Publisher>
</Test>
</Peach>

```

Output from this example.

```

> peach -l --debug example.xml

[*] Test 'Default' starting with random seed 49496.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(3 bytes)
00000000  34 44 32                                4D2
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(3 bytes)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Input
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher input()
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'InputModel' Bytes: 0/3, Bits: 0/24
Peach.Core.Cracker.DataCracker getSize: -----> DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: String 'InputModel.TransformMe' -> Offset: 0,
Unsized element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: DataModel 'InputModel' Size: <null>, Bytes :
0/3, Bits: 0/24
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker String 'InputModel.TransformMe' Bytes: 0/3, Bits: 0/24
Peach.Core.Cracker.DataCracker getSize: -----> String 'InputModel.TransformMe'

```

```
Peach.Core.Cracker.DataCracker scan: String 'InputModel.TransformMe' -> Offset: 0,
Unsized element
Peach.Core.Cracker.DataCracker lookahead: String 'InputModel.TransformMe'
Peach.Core.Cracker.getSize: <----- Last Unsized: 24
Peach.Core.Cracker.Crack: String 'InputModel.TransformMe' Size: 32,
Bytes: 0/4, Bits: 0/32
Peach.Core.Dom.DataElement String 'InputModel.TransformMe' value is: 1234
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Slurp
Peach.Core.Dom.Action Slurp, setting OutputModel.StringValue from
InputModel.TransformMe
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(4 bytes)
00000000  31 32 33 34                                1234
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()
Peach.Core.Publishers.FilePublisher stop()
Peach.Core.Publishers.FilePublisher stop()

[*] Test 'Default' finished.
```

21.6.15. Ipv4StringToOctet

Type

Encoder/Decoder

Transformer type: Encoder/Decoder

Ipv4StringToOctet produces the bytes from the string representation of an Internet Protocol version 4 address. An example would be transforming the string 127.0.0.1 to 0x7F000001.

The [Transformer](#) can bidirectionally encode data as well as decode data.

Parameters

None.

Attributes

None.

Actions Supported

input

On input, *Ipv4StringToOctet* converts bytes into an IP address string representation.

output

On output, *Ipv4StringToOctet* converts an IP address string representation into bytes.

Examples

Example 153. Simple Encode Example

This example uses *Ipv4StringToOctet* on an inline value, writes the transformed value to a file, and reads the value from the file to the console.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="Ex1">
        <String name="TransformMe" value="192.168.1.1">
            <Transformer class="Ipv4StringToOctet" />
        </String>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
```

```

<State name="initial">
    <!-- Encoded Octet Output -->
    <Action type="output" publisher="ConsolePub">
        <DataModel ref="Ex1" />
    </Action>

    <!-- Write Encoded Octet Output to File -->
    <Action type="output" publisher="FilePubWrite">
        <DataModel ref="Ex1" />
        <Data>
            <Field name="TransformMe" value="192.168.1.1" />
        </Data>
    </Action>

    <Action type="close" publisher="FilePubWrite" />

    <!-- Read and Decode file and slurp the IP String to output in console
-->
    <Action type="input" publisher="FilePubRead" >
        <DataModel name="InputModel" ref="Ex1" />
    </Action>

    <Action type="slurp" valueXpath="//InputModel//TransformMe" setXpath=
"//OutputModel//StringValue" />

    <Action type="output" publisher="ConsolePub">
        <DataModel name="OutputModel">
            <String name="StringValue" />
        </DataModel>
    </Action>
</State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex" name="ConsolePub"/>

    <Publisher class="File" name="FilePubWrite">
        <Param name="FileName" value="encoded.bin" />
    </Publisher>

    <Publisher class="File" name="FilePubRead">
        <Param name="FileName" value="encoded.bin" />
        <Param name="Overwrite" value="false" />
    </Publisher>
</Test>
</Peach>

```

Output from this example.

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 2053.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(4 bytes)
00000000  C0 A8 01 01          *****
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(4 bytes)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Input
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher input()
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'InputModel' Bytes: 0/4, Bits: 0/32
Peach.Core.Cracker.DataCracker getSize: -----> DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: String 'InputModel.TransformMe' -> Offset: 0,
Unsized element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: DataModel 'InputModel' Size: <null>, Bytes :
0/4, Bits: 0/32
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker String 'InputModel.TransformMe' Bytes: 0/4, Bits: 0/32
Peach.Core.Cracker.DataCracker getSize: -----> String 'InputModel.TransformMe'
Peach.Core.Cracker.DataCracker scan: String 'InputModel.TransformMe' -> Offset: 0,
Unsized element
Peach.Core.Cracker.DataCracker lookahead: String 'InputModel.TransformMe'
Peach.Core.Cracker.DataCracker getSize: <----- Last Unsized: 32
Peach.Core.Cracker.DataCracker Crack: String 'InputModel.TransformMe' Size: 88,
Bytes: 0/11, Bits: 0/88
Peach.Core.Dom.DataElement String 'InputModel.TransformMe' value is: 192.168.1.1
```

```
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Slurp
Peach.Core.Dom.Action Slurp, setting OutputModel.StringValue from
InputModel.TransformMe
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(11 bytes)
00000000  31 39 32 2E 31 36 38 2E  31 2E 31          192.168.1.1
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()
Peach.Core.Publishers.FilePublisher stop()
Peach.Core.Publishers.FilePublisher stop()

[*] Test 'Default' finished.
```

21.6.16. Ipv6StringToOctet

Type

Encoder/Decoder

Ipv6StringToOctet produces the octet bytes from the string representation of an Internet Protocol version 6 address. An example would be transforming the string ::1 to 0x00000000000000000000000000000001.

The [Transformer](#) can bidirectionally encode data as well as decode data.

Parameters

None.

Attributes

None.

Actions Supported

input

On input, `Ipv6StringToOctet` converts bytes into an IP address string representation.

output

On output, `Ipv6StringToOctet` converts an IP address string representation into bytes.

Examples

Example 154. Simple Encode Example

This example uses `Ipv6StringToOctet` on an inline value, writes the transformed value to a file, and reads the value from the file to the console.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="Ex1">
        <String name="TransformMe" value="fe80::4090:3894:4a88:a6f">
            <Transformer class="IPv6StringToOctet" />
        </String>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
```

```

<!-- Encoded Octet Output -->
<Action type="output" publisher="ConsolePub">
    <DataModel ref="Ex1" />
</Action>

<!-- Write Encoded Octet Output to File -->
<Action type="output" publisher="FilePubWrite">
    <DataModel ref="Ex1" />
    <Data>
        <Field name="TransformMe" value="fe80::4090:3894:4a88:a6f" />
    </Data>
</Action>

<Action type="close" publisher="FilePubWrite" />

<!-- Read and Decode file and slurp the IP String to output in console
-->

<Action type="input" publisher="FilePubRead" >
    <DataModel name="InputModel" ref="Ex1" />
</Action>

<Action type="slurp" valueXpath="//InputModel//TransformMe" setXpath=
"/OutputModel//StringValue" />

<Action type="output" publisher="ConsolePub">
    <DataModel name="OutputModel">
        <String name="StringValue" />
    </DataModel>
</Action>
</State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex" name="ConsolePub"/>

    <Publisher class="File" name="FilePubWrite">
        <Param name="FileName" value="encoded.bin" />
    </Publisher>

    <Publisher class="File" name="FilePubRead">
        <Param name="FileName" value="encoded.bin" />
        <Param name="Overwrite" value="false" />
    </Publisher>
</Test>
</Peach>

```

Output from this example.

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 10701.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(16 bytes)
00000000  FE 80 00 00 00 00 00 40 90 38 94 4A 88 0A 6F  ???????@?8?J??o
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(16 bytes)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Input
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher input()
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataModel 'InputModel' Bytes: 0/16, Bits: 0/128
Peach.Core.Cracker.getSize: -----> DataModel 'InputModel'
Peach.Core.Cracker.scan: DataModel 'InputModel'
Peach.Core.Cracker.scan: String 'InputModel.TransformMe' -> Offset: 0,
Unsized element
Peach.Core.Cracker.getSize: <----- Deterministic: ???
Peach.Core.Cracker.Crack: DataModel 'InputModel' Size: <null>, Bytes :
0/16, Bits: 0/128
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.String 'InputModel.TransformMe' Bytes: 0/16, Bits :
0/128
Peach.Core.Cracker.getSize: -----> String 'InputModel.TransformMe'
Peach.Core.Cracker.scan: String 'InputModel.TransformMe' -> Offset: 0,
Unsized element
Peach.Core.Cracker.DataCracker lookahead: String 'InputModel.TransformMe'
Peach.Core.Cracker.getSize: <----- Last Unsized: 128
Peach.Core.Cracker.Crack: String 'InputModel.TransformMe' Size: 192,
Bytes: 0/24, Bits: 0/192
Peach.Core.Dom.DataElement String 'InputModel.TransformMe' value is:
```

```
fe80::4090:3894:4a88:a6f
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Slurp
Peach.Core.Dom.Action Slurp, setting OutputModel.StringValue from
InputModel.TransformMe
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(24 bytes)
00000000  66 65 38 30 3A 3A 34 30  39 30 3A 33 38 39 34 3A  fe80::4090:3894:
00000010  34 61 38 38 3A 61 36 66                               4a88:a6f
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()
Peach.Core.Publishers.FilePublisher stop()
Peach.Core.Publishers.FilePublisher stop()

[*] Test 'Default' finished.
```

21.6.17. JsEncode

Type

Encoder/Decoder

JsEncode produces a JavaScript-encoded result of the value in the parent [DataModel](#). All characters are encoded.

This [Transformer](#) only encodes the output. *JsEncode* does not decode any input.

The *JsEncode* algorithm filters alphabetic characters (A-Z, a-z), numerals (0-9), spaces, commas, and periods to pass through the transformer unaltered. Other characters are filtered and have formatting applied to them.

Parameters

None.

Attributes

None.

Actions Supported

[output](#)

On output, this transformer encodes the value using the *JsEncode* algorithm to the outgoing data.

Examples

Example 155. Simple Encode Example

This example uses *JsEncode* on an inline value and writes the transformed value to the console.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="Ex1">
        <String value="JSEncoded: " token="true"/>
        <String name="TransformMe" value="~!@#$%^&*()_+=`?.,">
            <Transformer class="JsEncode" />
        </String>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <!-- Encoded XML Output -->
            <Action type="output" publisher="ConsolePub">
                <DataModel ref="Ex1" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex" name="ConsolePub"/>
    </Test>
</Peach>

```

Output from this example.

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 33041.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(81 bytes)
00000000  4A 53 45 6E 63 6F 64 65  64 3A 20 5C 78 37 45 5C  JSEncoded: \x7E\
00000010  78 32 31 5C 78 34 30 5C  78 32 33 5C 78 32 34 5C  x21\x40\x23\x24\
00000020  78 32 35 5C 78 35 45 5C  78 32 36 5C 78 32 41 5C  x25\x5E\x26\x2A\
00000030  78 32 38 5C 78 32 39 5C  78 35 46 5C 78 32 42 5C  x28\x29\x5F\x2B\
00000040  78 32 44 5C 78 33 44 5C  78 36 30 5C 78 33 46 2E  x2D\x3D\x60\x3F.
00000050  2C ,  

Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

21.6.18. Md5

Type

Cryptography

MD5 creates an MD5 hash of the value in the parent [DataModel](#).

This [Transformer](#) can only be applied to outgoing data because hashes are one-way operations.

Parameters

None.

Attributes

None.

Actions Supported

[output](#)

On output, this transformer hashes the outgoing data.

Examples

Example 156. Simple Console Example

This example uses MD5 on an inline value and writes the transformed value to the console.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="Ex1">
        <Block name="Main">
            <String name="TransformMe" value="superdoopersecret">
                <Transformer class="Md5" />
            </String>
        </Block>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <!-- Encrypted Output -->
            <Action type="output" publisher="ConsolePub">
                <DataModel ref="Ex1" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex" name="ConsolePub"/>
    </Test>
</Peach>

```

Output from this example.

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 58687.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(16 bytes)
00000000  04 9B F0 86 41 46 4B 3F  3D 49 95 6D D0 81 12 BA  ???AFK?=I?m???
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

21.6.19. NetBiosDecode

Type

Encoder/Decoder

NetBiosDecode decodes NetBios-encoded strings.

The [Transformer](#) can bidirectionally encode data as well as decode data.

Parameters

Required:

None.

Optional:

pad

Boolean value that determines whether the NetBios name should be padded/trimmed to 32 bytes.

Attributes

None.

Actions Supported

input

On input, this transformer encodes incoming data using the NetBios encoding algorithm.

output

On output, this transformer decodes outgoing data using the NetBios decoding algorithm.

Examples

Example 157. Decoding Value To and From File Example

This example uses NetBiosDecode on an inline value, writes the transformed value to a file, and reads the value from the file to the console.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

  <DataModel name="Ex1">
    <String value="NetBiosDecoded: " token="true"/>
    <String name="TransformMe" value="45 4D 45 42 45 4F 45 4E 45 42 45 4F">
```

```

    valueType="hex">
        <Transformer class="NetBiosDecode" />
    </String>
</DataModel>

<StateModel name="TheState" initialState="initial">
    <State name="initial">
        <!-- Decoded Output -->
        <Action type="output" publisher="ConsolePub">
            <DataModel ref="Ex1" />
        </Action>

        <!-- Write Encoded Output to File -->
        <Action type="output" publisher="FilePubWrite">
            <DataModel ref="Ex1" />
            <Data>
                <Field name="TransformMe" value="45 4D 45 42 45 4F 45 4E 45 42 45
4F" valueType="hex"/>
            </Data>
        </Action>
        <Action type="close" publisher="FilePubWrite" />

        <!-- Read and decode encoded file and slurp the XML body to output in
console -->
        <Action type="input" publisher="FilePubRead" >
            <DataModel name="InputModel" ref="Ex1" />
        </Action>
        <Action type="slurp" valueXpath="//InputModel//TransformMe" setXpath=
"/OutputModel//StringValue" />
        <Action type="output" publisher="ConsolePub">
            <DataModel name="OutputModel">
                <String name="StringValue" />
            </DataModel>
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex" name="ConsolePub"/>

    <Publisher class="File" name="FilePubWrite">
        <Param name="FileName" value="encrypted.bin" />
    </Publisher>

    <Publisher class="File" name="FilePubRead">

```

```

    <Param name="FileName" value="encrypted.bin" />
    <Param name="Overwrite" value="false" />
  </Publisher>
</Test>
</Peach>
```

Output from this example.

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 54340.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(22 bytes)
00000000  4E 65 74 42 69 6F 73 44  65 63 6F 64 65 64 3A 20  NetBiosDecoded:
00000010  4C 41 4E 4D 41 4E                               LANMAN
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(22 bytes)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Input
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher input()
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'InputModel' Bytes: 0/22, Bits: 0/176
Peach.Core.Cracker.DataCracker getSize: -----> DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: String 'InputModel.DataElement_0' -> Pos: 0,
Saving Token
Peach.Core.Cracker.DataCracker scan: String 'InputModel.DataElement_0' -> Pos: 128,
Length: 128
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: DataModel 'InputModel' Size: <null>, Bytes
: 0/22, Bits: 0/176
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker String 'InputModel.DataElement_0' Bytes: 0/22, Bits:
```

0/176

```
Peach.Core.Cracker.getSize: -----> String 'InputModel.DataElement_0'

Peach.Core.Cracker.scan: String 'InputModel.DataElement_0' -> Pos: 0,
Saving Token
Peach.Core.Cracker.scan: String 'InputModel.DataElement_0' -> Pos: 128,
Length: 128
Peach.Core.Cracker.getSize: <----- Size: 128
Peach.Core.Cracker.Crack: String 'InputModel.DataElement_0' Size: 128,
Bytes: 0/22, Bits: 0/176
Peach.Core.Dom.DataElement String 'InputModel.DataElement_0' value is:
NetBiosDecoded:
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.String 'InputModel.TransformMe' Bytes: 16/22, Bit s:
128/176
Peach.Core.Cracker.getSize: -----> String 'InputModel.TransformMe'
Peach.Core.Cracker.scan: String 'InputModel.TransformMe' -> Offset: 0,
Unsized element
Peach.Core.Cracker.lookahead: String 'InputModel.TransformMe'
Peach.Core.Cracker.getSize: <----- Last Unsized: 48
Peach.Core.Cracker.Crack: String 'InputModel.TransformMe' Size: 96,
Bytes: 0/12, Bits: 0/96
Peach.Core.Dom.DataElement String 'InputModel.TransformMe' value is: EMEBEOENEBE0
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Slurp
Peach.Core.Dom.Action Slurp, setting OutputModel.StringValue from
InputModel.TransformMe
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(12 bytes)
00000000  45 4D 45 42 45 4F 45 4E  45 42 45 4F          EMEBEOENEBE0
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()
Peach.Core.Publishers.FilePublisher stop()
Peach.Core.Publishers.FilePublisher stop()
```

[*] Test 'Default' finished.

21.6.20. NetBiosEncode

Type

Encoder/Decoder

NetBiosEncode performs a NetBios encoding of the value in the parent [DataModel](#).

The [Transformer](#) can bidirectionally encode data as well as decode data.

Parameters

Required:

None.

Optional:

pad

Boolean value that determines whether the NetBios name should be padded/trimmed to 32 bytes.

Attributes

None.

Actions Supported

input

On input, this transformer decodes incoming data using the NetBios decoding algorithm.

output

On output, this transformer encodes outgoing data using the NetBios encoding algorithm.

Examples

Example 158. Encoding Value To and From File Example

This example uses *NetBiosEncode* on an inline value, writes the transformed value to a file, and reads the value from the file to the console.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

  <DataModel name="Ex1">
    <String value="NetBiosEncoded: " token="true"/>
    <String name="TransformMe" value="LANMAN" >
```

```

        <Transformer class="NetBiosEncode" />
    </String>
</DataModel>

<StateModel name="TheState" initialState="initial">
    <State name="initial">
        <!-- Encoded NetBios Output -->
        <Action type="output" publisher="ConsolePub">
            <DataModel ref="Ex1" />
        </Action>

        <!-- Write Encoded Output to File -->
        <Action type="output" publisher="FilePubWrite">
            <DataModel ref="Ex1" />
            <Data>
                <Field name="TransformMe" value="LANMAN" />
            </Data>
        </Action>

        <Action type="close" publisher="FilePubWrite" />

        <!-- Read and decode encoded file and slurp the XML body to output in
console -->
        <Action type="input" publisher="FilePubRead" >
            <DataModel name="InputModel" ref="Ex1" />
        </Action>

        <Action type="slurp" valueXpath="//InputModel//TransformMe" setXpath=
"//OutputModel//StringValue" />

        <Action type="output" publisher="ConsolePub">
            <DataModel name="OutputModel">
                <String name="StringValue" />
            </DataModel>
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex" name="ConsolePub"/>

    <Publisher class="File" name="FilePubWrite">
        <Param name="FileName" value="encoded.bin" />
    </Publisher>

```

```

<Publisher class="File" name="FilePubRead">
    <Param name="FileName" value="encoded.bin" />
    <Param name="Overwrite" value="false" />
</Publisher>
</Test>
</Peach>

```

Output from this example.

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 14812.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(28 bytes)
00000000  4E 65 74 42 69 6F 73 45  6E 63 6F 64 65 64 3A 20  NetBiosEncoded:
00000010  45 4D 45 42 45 4F 45 4E  45 42 45 4F                           EMEBEOENEBEO
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(28 bytes)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Input
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher input()
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'InputModel' Bytes: 0/28, Bits: 0/224
Peach.Core.Cracker.DataCracker getSize: -----> DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: String 'InputModel.DataElement_0' -> Pos: 0 ,
Saving Token
Peach.Core.Cracker.DataCracker scan: String 'InputModel.DataElement_0' -> Pos: 1 28,
Length: 128
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: DataModel 'InputModel' Size: <null>, Bytes
: 0/28, Bits: 0/224
Peach.Core.Cracker.DataCracker -----

```

```
Peach.Core.Cracker.String 'InputModel.DataElement_0' Bytes: 0/28, Bits: 0/224
Peach.Core.Cracker.getSize: -----> String 'InputModel.DataElement_0'

Peach.Core.Cracker.scan: String 'InputModel.DataElement_0' -> Pos: 0, Saving Token
Peach.Core.Cracker.scan: String 'InputModel.DataElement_0' -> Pos: 1 28, Length: 128
Peach.Core.Cracker.getSize: <----- Size: 128
Peach.Core.Cracker.Crack: String 'InputModel.DataElement_0' Size: 12 8, Bytes: 0/28, Bits: 0/224
Peach.Core.Dom.DataElement String 'InputModel.DataElement_0' value is: NetBiosEncoded:
Peach.Core.Cracker -----
Peach.Core.Cracker.String 'InputModel.TransformMe' Bytes: 16/28, Bits: 128/224
Peach.Core.Cracker.getSize: -----> String 'InputModel.TransformMe'
Peach.Core.Cracker.scan: String 'InputModel.TransformMe' -> Offset: 0, Unsized element
Peach.Core.Cracker.lookahead: String 'InputModel.TransformMe'
Peach.Core.Cracker.getSize: <----- Last Unsized: 96
Peach.Core.Cracker.Crack: String 'InputModel.TransformMe' Size: 48, Bytes: 0/6, Bits: 0/48
Peach.Core.Dom.DataElement String 'InputModel.TransformMe' value is: LANMAN
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Slurp
Peach.Core.Dom.Action Slurp, setting OutputModel.StringValue from InputModel.TransformMe
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(6 bytes)
00000000  4C 41 4E 4D 41 4E          LANMAN
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()
Peach.Core.Publishers.FilePublisher stop()
Peach.Core.Publishers.FilePublisher stop()

[*] Test 'Default' finished.
```

21.6.21. Sha1

Type

Cryptography

Sha1 produces an SHA1 hash of the value in the parent [DataModel](#).

This [Transformer](#) can only be applied to outgoing data because hashes are one-way operations.

Parameters

None.

Attributes

None.

Actions Supported

output

On output, this transformer hashes the outgoing data.

Examples

Example 159. Simple Console Example

This example uses Sha256 on an inline value and writes the transformed value to the console.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="Ex1">
        <Block name="Main">
            <String name="TransformMe" value="superdoopersecret">
                <Transformer class="Sha1" />
            </String>
        </Block>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <!-- Hashed Output -->
            <Action type="output" publisher="ConsolePub">
                <DataModel ref="Ex1" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex" name="ConsolePub"/>
    </Test>
</Peach>

```

Output from this example.

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 30940.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(20 bytes)
00000000  00 93 99 11 AB 4B D1 9C  8E FC 36 41 36 32 E3 AE  ?????K????6A62??
00000010  35 10 19 70                               5??p
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

21.6.22. Sha256

Type

Cryptography

Sha256 produces an SHA256 hash of the value in the parent [DataModel](#).

This [Transformer](#) can only be applied to outgoing data because hashes are one-way operations.

Parameters

None.

Actions Supported

[output](#)

On output, this transformer hashes the outgoing data.

Examples

[Example 160. Simple Console Example](#)

This example uses Sha256 on an inline value and writes the transformed value to the console.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="Ex1">
        <Block name="Main">
            <String name="TransformMe" value="superdoopersecret">
                <Transformer class="Sha256" />
            </String>
        </Block>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <!-- Hash Output -->
            <Action type="output" publisher="ConsolePub">
                <DataModel ref="Ex1" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex" name="ConsolePub"/>
    </Test>
</Peach>

```

Output from this example.

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 46784.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(32 bytes)
00000000  02 44 19 39 45 FF 4A CD A8 1B 28 89 15 08 2A 04  ?D?9E?J???(??*?
00000010  9B 07 9A 31 E8 B4 79 B7 AE D0 49 66 0D 92 81 5B  ???1??y???If???
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

21.6.23. SidStringToBytes

Type

Encoder/Decoder

SidStringToBytes produces the byte representation from a security identifier (sid) string.

This [Transformer](#) can bidirectionally encode data as well as decode data.

Parameters

None.

Attributes

None.

Actions Supported

input

On input, this transformer converts bytes to a sid string.

output

On output, this transformer converts a sid string to bytes.

Examples

Example 161. Simple Console Example

This example uses the *SidStringtoBytes* transformer on an inline value and writes the transformed value to a file, reads the value from a file, and outputs it to the console.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="Ex1">
        <String name="TransformMe" value="S-1-5-21-2127521184-1604012920-1887927527-
1712781">
            <Transformer class="SidStringToBytes" />
        </String>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <!-- Byte Output -->
```

```

<Action type="output" publisher="ConsolePub">
    <DataModel ref="Ex1" />
</Action>

<!-- Write Byte Output to File -->
<Action type="output" publisher="FilePubWrite">
    <DataModel ref="Ex1" />
    <Data>
        <Field name="TransformMe" value="S-1-5-21-2127521184-1604012920-
1887927527-1712781" />
    </Data>
</Action>

<Action type="close" publisher="FilePubWrite" />

<!-- Read and encode binary file and slurp SID encoded string to output
in console -->
<Action type="input" publisher="FilePubRead" >
    <DataModel name="InputModel" ref="Ex1" />
</Action>

<Action type="slurp" valueXpath="//InputModel//TransformMe" setXpath=
"/OutputModel/StringValue" />

<Action type="output" publisher="ConsolePub">
    <DataModel name="OutputModel">
        <String name="StringValue" />
    </DataModel>
</Action>
</State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex" name="ConsolePub"/>

    <Publisher class="File" name="FilePubWrite">
        <Param name="FileName" value="encoded.bin" />
    </Publisher>

    <Publisher class="File" name="FilePubRead">
        <Param name="FileName" value="encoded.bin" />
        <Param name="Overwrite" value="false" />
    </Publisher>
</Test>
</Peach>

```

Output from this example.

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 32239.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(28 bytes)
00000000  01 05 00 00 00 00 05 15 00 00 00 A0 65 CF 7E  ?????????????e?~
00000010  78 4B 9B 5F E7 7C 87 70  8D 22 1A 00           xK?-_|?p?"??
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(28 bytes)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Input
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher input()
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'InputModel' Bytes: 0/28, Bits: 0/224
Peach.Core.Cracker.DataCracker getSize: -----> DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: String 'InputModel.TransformMe' -> Offset: 0,
Unsized element
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: DataModel 'InputModel' Size: <null>, Bytes :
0/28, Bits: 0/224
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker String 'InputModel.TransformMe' Bytes: 0/28, Bits :
0/224
Peach.Core.Cracker.DataCracker getSize: -----> String 'InputModel.TransformMe'
Peach.Core.Cracker.DataCracker scan: String 'InputModel.TransformMe' -> Offset: 0,
Unsized element
Peach.Core.Cracker.DataCracker lookahead: String 'InputModel.TransformMe'
Peach.Core.Cracker.DataCracker getSize: <----- Last Unsized: 224
Peach.Core.Cracker.DataCracker Crack: String 'InputModel.TransformMe' Size: 392,
Bytes: 0/49, Bits: 0/392
```

```
Peach.Core.Dom.DataElement String 'InputModel.TransformMe' value is: S-1-5-21-
2127521184-1604012920-1887927527-1712781
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Slurp
Peach.Core.Dom.Action Slurp, setting OutputModel.StringValue from
InputModel.TransformMe
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(49 bytes)
00000000 53 2D 31 2D 35 2D 32 31 2D 32 31 32 37 35 32 31      S-1-5-21-2127521
00000010 31 38 34 2D 31 36 30 34 30 31 32 39 32 30 2D 31      184-1604012920-1
00000020 38 38 37 39 32 37 35 32 37 2D 31 37 31 32 37 38      887927527-171278
00000030 31                                         1

Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()
Peach.Core.Publishers.FilePublisher stop()
Peach.Core.Publishers.FilePublisher stop()
```

[*] Test 'Default' finished.

21.6.24. TripleDes

Type

Cryptography

The *TripleDesTransformer* produces the bytes from a Triple DES encryption of the value in the parent [DataModel](#).

The [Transformer](#) can bi-directionally encrypt data as well as decrypt data.

Parameters

None.

Attributes

Required:

Key

User provided symmetric key used to encrypt the value. Must be a hex string representation of a 24 byte key.

IV

User provided initialization vector used as the first block for the TripleDES operation. Must be a hex string representation of an 8 byte value.

Optional:

None.

Actions Supported

input

On input, this transformer decrypts the incoming data.

output

On output, this transformer encrypts the outgoing data.

Examples

Example 162. Simple Console Example

This example uses the TripleDes transformer on an inline value and writes the transformed value to a file, reads the value from a file, and outputs it to the console.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
```

```

"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

<DataModel name="Ex1">
    <String name="TransformMe" value="superdoopersecret">
        <Transformer class="TripleDes">
            <Param name="Key" value=
"ae1234567890aeaffeda214354647586fefdfaddefeeaf12"/>
            <Param name="IV" value="aeaeaeaeaeaeae"/>
        </Transformer>
    </String>
</DataModel>

<StateModel name="TheState" initialState="initial">
    <State name="initial">
        <!-- Encrypted Output -->
        <Action type="output" publisher="ConsolePub">
            <DataModel ref="Ex1" />
        </Action>

        <!-- Write Encrypted Output to File -->
        <Action type="output" publisher="FilePubWrite">
            <DataModel ref="Ex1" />
            <Data>
                <Field name="TransformMe" value="superdoopersecret" />
            </Data>
        </Action>

        <Action type="close" publisher="FilePubWrite" />

        <!-- Read and decrypt encrypted file and slurp output to console -->
        <Action type="input" publisher="FilePubRead" >
            <DataModel name="InputModel" ref="Ex1" />
        </Action>

        <Action type="slurp" valueXpath="//InputModel//TransformMe" setXpath=
"//OutputModel//StringValue" />

        <Action type="output" publisher="ConsolePub">
            <DataModel name="OutputModel">
                <String name="StringValue" />
            </DataModel>
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>

```

```

<Publisher class="ConsoleHex" name="ConsolePub"/>

<Publisher class="File" name="FilePubWrite">
    <Param name="FileName" value="encrypted.bin" />
</Publisher>

<Publisher class="File" name="FilePubRead">
    <Param name="FileName" value="encrypted.bin" />
    <Param name="Overwrite" value="false" />
</Publisher>
</Test>
</Peach>

```

Output from this example.

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 54300.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(24 bytes)
00000000  27 C4 63 A3 AA 09 C1 6D  08 CF DC C8 F5 CD E2 DB  '?c????m?????????
00000010  19 31 30 F1 6A C8 28 10                      ?10?j?(?
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(24 bytes)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Input
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher input()
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'InputModel' Bytes: 0/24, Bits: 0/192
Peach.Core.Cracker.DataCracker getSize: -----> DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: String 'InputModel.TransformMe' -> Offset: 0,

```

```
Unsized element
Peach.Core.Cracker.getSize: <----- Deterministic: ???
Peach.Core.Cracker Crack: DataModel 'InputModel' Size: <null>, Bytes :
0/24, Bits: 0/192
Peach.Core.Cracker -----
Peach.Core.Cracker String 'InputModel.TransformMe' Bytes: 0/24, Bits :
0/192
Peach.Core.Cracker getSize: -----> String 'InputModel.TransformMe'
Peach.Core.Cracker scan: String 'InputModel.TransformMe' -> Offset: 0,
Unsized element
Peach.Core.Cracker lookahead: String 'InputModel.TransformMe'
Peach.Core.Cracker getSize: <----- Last Unsized: 192
Peach.Core.Cracker Crack: String 'InputModel.TransformMe' Size: 192,
Bytes: 0/24, Bits: 0/192
Peach.Core.Dom.DataElement String 'InputModel.TransformMe' value is:
superdoopersecret
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Slurp
Peach.Core.Dom.Action Slurp, setting OutputModel.StringValue from
InputModel.TransformMe
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(24 bytes)
00000000 73 75 70 65 72 64 6F 6F 70 65 72 73 65 63 72 65  superdoopersecre
00000010 74 00 00 00 00 00 00 00 t???????
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()
Peach.Core.Publishers.FilePublisher stop()
Peach.Core.Publishers.FilePublisher stop()

[*] Test 'Default' finished.
```

21.6.25. Truncate

Type

Encoder

Truncate will shrink the value in the parent [DataModel](#) to *Length* bytes.

This [Transformer](#) can only be applied to outgoing data because shrinking is a one-way operation.

Parameters

Length

Length of the result value.

Optional:

Offset

Optional starting index for the resulting substring.

Attributes

No Attributes are supported by this transformer.

Actions Supported

output

On output this transformer will truncate the outgoing data.

Examples

Example 163. Simple Console Example

This example uses Truncate on an inline value and writes the transformed value to the console.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="Ex1">
        <Block name="Main">
            <String name="TransformMe" value="superdoopersecret">
                <Transformer class="Truncate">
                    <Param name="Length" value="5" />
                </Transformer>
            </String>
        </Block>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <!-- Truncated Output -->
            <Action type="output" publisher="ConsolePub">
                <DataModel ref="Ex1" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="ConsoleHex" name="ConsolePub"/>    </Test>
    </Peach>

```

Output from this example.

```
> peach -1 --debug example.xml

[*] Web site running at: http://localhost:8888/

[*] Test 'Default' starting with random seed 3082.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "initial".
Peach.Core.Dom.Action Run(Action): Output
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(5 bytes)
00000000  73 75 70 65 72                      super
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

21.6.26. UrlEncode

Type

Encoder/Decoder

UrlEncode produces a URL-encoded string of the provided value in the parent [DataModel](#).

The [Transformer](#) can bidirectionally encode data as well as decode data.

Parameters

None.

Attributes

None.

Actions Supported

input

On input, this transformer decodes a URL-encoded string.

output

On output, this transformer encodes a URL-encoded string.

Examples

Example 164. Simple Encode Example

This example uses *UrlEncode* on an inline value, writes the transformed value to a file, and reads the value from the file to the console.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="Ex1">
        <String value="http://peachfuzzer.com/?q=" token="true"/>
        <String name="TransformMe" value="'abcd=1234'">
            <Transformer class="UrlEncode" />
        </String>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <!-- Encoded URL Output -->
```

```

<Action type="output" publisher="ConsolePub">
    <DataModel ref="Ex1" />
</Action>

<!-- Write Encoded Output to File -->
<Action type="output" publisher="FilePubWrite">
    <DataModel ref="Ex1" />
    <Data>
        <Field name="TransformMe" value="'abcd=1234'" />
    </Data>
</Action>

<Action type="close" publisher="FilePubWrite" />

<!-- Read and decode encoded file and slurp URL parameter to output in
console -->
<Action type="input" publisher="FilePubRead" >
    <DataModel name="InputModel" ref="Ex1" />
</Action>

<Action type="slurp" valueXpath="//InputModel//TransformMe" setXpath=
"//OutputModel//StringValue" />

<Action type="output" publisher="ConsolePub">
    <DataModel name="OutputModel">
        <String name="StringValue" />
    </DataModel>
</Action>
</State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="ConsoleHex" name="ConsolePub"/>

    <Publisher class="File" name="FilePubWrite">
        <Param name="FileName" value="encoded.bin" />
    </Publisher>

    <Publisher class="File" name="FilePubRead">
        <Param name="FileName" value="encoded.bin" />
        <Param name="Overwrite" value="false" />
    </Publisher>
</Test>
</Peach>

```

Output from this example.

```
> peach -l --debug example.xml

[*] Test 'Default' starting with random seed 29519.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(43 bytes)
00000000  68 74 74 70 3A 2F 2F 70  65 61 63 68 66 75 7A 7A   http://peachfuzz
00000010  65 72 2E 63 6F 6D 2F 3F  71 3D 25 32 37 61 62 63   er.com/?q=%27abc
00000020  64 25 33 64 31 32 33 34  25 32 37                   d%3d1234%27
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(43 bytes)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Input
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher input()
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'InputModel' Bytes: 0/43, Bits: 0/344
Peach.Core.Cracker.DataCracker getSize: -----> DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: DataModel 'InputModel'
Peach.Core.Cracker.DataCracker scan: String 'InputModel.DataElement_0' -> Pos: 0,
Saving Token
Peach.Core.Cracker.DataCracker scan: String 'InputModel.DataElement_0' -> Pos: 208,
Length: 208
Peach.Core.Cracker.DataCracker getSize: <----- Deterministic: ???
Peach.Core.Cracker.DataCracker Crack: DataModel 'InputModel' Size: <null>, Bytes:
0/43, Bits: 0/344
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker String 'InputModel.DataElement_0' Bytes: 0/43, Bits:
0/344
Peach.Core.Cracker.DataCracker getSize: -----> String 'InputModel.DataElement_0'
Peach.Core.Cracker.DataCracker scan: String 'InputModel.DataElement_0' -> Pos: 0,
Saving Token
```

```
Peach.Core.Cracker.DataCracker scan: String 'InputModel.DataElement_0' -> Pos: 208,
Length: 208
Peach.Core.Cracker.DataCracker getSize: <----- Size: 208
Peach.Core.Cracker.DataCracker Crack: String 'InputModel.DataElement_0' Size: 208,
Bytes: 0/43, Bits: 0/344
Peach.Core.Dom.DataElement String 'InputModel.DataElement_0' value is:
http://peachfuzzer.com/?q=
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker String 'InputModel.TransformMe' Bytes: 26/43, Bit s:
208/344
Peach.Core.Cracker.DataCracker getSize: -----> String 'InputModel.TransformMe'
Peach.Core.Cracker.DataCracker scan: String 'InputModel.TransformMe' -> Offset: 0,
Unsized element
Peach.Core.Cracker.DataCracker lookahead: String 'InputModel.TransformMe'
Peach.Core.Cracker.DataCracker getSize: <----- Last Unsized: 136
Peach.Core.Cracker.DataCracker Crack: String 'InputModel.TransformMe' Size: 88,
Bytes: 0/11, Bits: 0/88
Peach.Core.Dom.DataElement String 'InputModel.TransformMe' value is: 'abcd=1234'
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Slurp
Peach.Core.Dom.Action Slurp, setting OutputModel.StringValue from
InputModel.TransformMe
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(11 bytes)
00000000  27 61 62 63 64 3D 31 32  33 34 27          'abcd=1234'
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()
Peach.Core.Publishers.FilePublisher stop()
Peach.Core.Publishers.FilePublisher stop()

[*] Test 'Default' finished.
```

21.7. State Modeling

This section provides reference information for state-modeling-related elements.

21.7.1. StateModel

Two Peach models can create a fuzzer, the [DataModel](#) and the [StateModel](#).

The *StateModel* consists of at least one [State](#) element. *StateModel* defines how to send data to and receive data from the fuzzing target as well as creating the basic state machine logic needed to test a protocol.



Only one *StateModel* can be used for a given Test.

StateModel can range from very simple to extremely complex. When starting out, keep the *StateModel* simple and expand as needed.

Multiple [State](#) elements are required for non-deterministic interactions (such as a target sending an unpredictable command and expecting a certain response from the fuzzer). In these situations, couple the [changeState](#) action with a [when](#) attribute to change to the correct state depending on Peach's state and data.

StateModel can contain various States to model full interactions with a target. Many interactions have deterministic I/O and do not need multiple States. Since actions inside a [State](#) occur sequentially, top to bottom, that is usually enough control.

Recursion into States is allowed in Peach. When a state is re-entered, all data previously set through cracking input or slurping is reset to its default values.



Peach is not responsible for guaranteeing an exit condition in recursive state changes; if you need such an exit condition, you will need to create it.

Each [Action](#) supports various parameters that execute scripts in certain times (such as once an [Action](#) has completed or before it starts). This can be used by complicated StateModels that need counters or other advanced operations.

Advanced StateModels can use the Peach Statebag dictionaries from within these scripts to store data. The Statebag dictionaries are accessible via class RunContext.

StateModel variables can persist throughout a single iteration with *iterationStateStore* (or the entire fuzzing session with *stateStore*) and can be referred to through the scripting environment.

Attributes

Required:

name

The name of the StateModel

initialState

Name of the State to start executing first

Child Elements

State

One or more State elements are required.

Examples

Example 165. File Fuzzing

When file fuzzing, Peach writes data to a file, then calls the target process to open the file. Peach can uses a single state and three actions for a simple file fuzzer.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TestTemplate">
        <String value="Hello World!" />
    </DataModel>

    <StateModel name="State" initialState="Initial">
        <State name="Initial">

            <!-- Output content of file -->
            <Action type="output">
                <DataModel ref="TestTemplate" />
            </Action>

            <!-- Close file -->
            <Action type="close" />

            <!-- Launch the file consumer -->
            <Action type="call" method="ScoobySnacks" publisher="Peach.Agent"/>

        </State>
    </StateModel>

    <Agent name="LocalAgent">
        <Monitor class="WindowsDebugger">
            <Param name="Executable" value="c:\windows\system32\notepad.exe" />
            <Param name="Arguments" value="fuzzfile.bin" />
            <Param name="StartOnCall" value="ScoobySnacks" />
        </Monitor>
        <Monitor class="PageHeap">
            <Param name="Executable" value="notepad.exe"/>
        </Monitor>
    </Agent>

    <Test name="Default">
        <Agent ref="LocalAgent" />
        <StateModel ref="State"/>

        <Publisher class="File">
            <Param name="FileName" value="fuzzfile.bin" />
        </Publisher>
    </Test>

</Peach>

```

Example 166. Simple Network StateModel

In this state model, Peach sends and receives a set of packets from a TCP port.

This example requires Windows XP or newer with Windows Debugging Tools installed. The sample executable *CrashableServer.exe* is included with the Peach distribution.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TestTemplate">
        <String name="TheString" value="Hello World!" />
    </DataModel>

    <StateModel name="State" initialState="Initial">
        <State name="Initial">

            <Action type="output">
                <DataModel ref="TestTemplate" />
            </Action>

            <Action type="output">
                <DataModel ref="TestTemplate" />
            </Action>

        </State>
    </StateModel>

    <Agent name="LocalAgent">
        <Monitor class="WindowsDebugger">
            <Param name="Executable" value="CrashableServer.exe" />
            <Param name="Arguments" value="127.0.0.1 4244" />
            <!--<Param name="WinDbgPath" value="C:\Program Files (x86)\Debugging
Tools for Windows (x86)" /-->
        </Monitor>
        <Monitor class="PageHeap">
            <Param name="Executable" value="CrashableServer.exe"/>
            <!--<Param name="WinDbgPath" value="C:\Program Files (x86)\Debugging
Tools for Windows (x86)" /-->
        </Monitor>
        <Monitor class="NetworkCapture">
            <Param name="Device" value="Local Area Connection"/>
        </Monitor>
    </Agent>
```

```

<Test name="Default">
    <Agent ref="LocalAgent" />

    <StateModel ref="State"/>
    <Publisher class="Tcp">
        <Param name="Host" value="127.0.0.1" />
        <Param name="Port" value="4244" />
    </Publisher>
</Test>
</Peach>

```

Example 167. Multiple State StateModel

The following StateModel utilizes multiple states to interact with the fuzzing target.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

<DataModel name="Question">
    <String value="Select A,B, or C:" />
</DataModel>

<DataModel name="CResponse">
    <String value="C is for Correct" />
    <String value=" === " />
    <String value="YOU WIN" />
</DataModel>

<DataModel name="BResponse">
    <String value="B is for Bananas" />
    <String value=":" />
    <String value=" B - A - N - A - N - A - S" />
</DataModel>

<DataModel name="AResponse">
    <String value="A is for Apples" />
    <String value=". " />
    <String value="Play Again" />
    <String value="\n" />
</DataModel>

<DataModel name="Selection">
    <String name="Letter" length="1"/>

```

```

<Blob length="1" valueType="hex" value="0A" token="true" />
</DataModel>

<StateModel name="TheStateModel" initialState="InitialState">
    <State name="InitialState">
        <Action type="accept" />
        <Action type="changeState" ref="AskState" />
    </State>

    <State name="AskState">
        <Action type="output">
            <DataModel ref="Question" />
        </Action>

        <Action type="input">
            <DataModel name="TheirSelection" ref="Selection"/>
        </Action>

        <Action type="changeState" ref="SelectedA" when=
"str(state.actions[1].dataModel.find('Letter').DefaultValue) == 'A'" />
        <Action type="changeState" ref="SelectedB" when=
"str(state.actions[1].dataModel.find('Letter').DefaultValue) == 'B'" />
        <Action type="changeState" ref="SelectedC" when=
"str(state.actions[1].dataModel.find('Letter').DefaultValue) == 'C'" />
    </State>

    <State name="SelectedA">
        <Action type="output">
            <DataModel ref="AResponse" />
        </Action>
        <Action type="changeState" ref="AskState" />
    </State>

    <State name="SelectedB">
        <Action type="output">
            <DataModel ref="BResponse" />
        </Action>
    </State>

    <State name="SelectedC">
        <Action type="output">
            <DataModel ref="CResponse" />
        </Action>
    </State>
</StateModel>
```

```
<Test name="Default">
    <StateModel ref="TheStateModel"/>
    <Publisher class="TcpListener">
        <Param name="Interface" value="0.0.0.0" />
        <Param name="Port" value="31337" />
        <Param name="AcceptTimeout" value="10000" />
        <Param name="Timeout" value="10000" />
    </Publisher>

    <Logger class="File" >
        <Param name="Path" value="logs"/>
    </Logger>
</Test>
</Peach>
```

Example 168. Loop Using Iteration State Bag

This example uses the iteration state bag to simulate a "do while" or "do until" loop.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String value="Looping!\n" />
    </DataModel>

    <StateModel name="State" initialState="Initial">
        <State name="Initial">

            <!-- Initialize our counter -->
            <Action type="changeState" ref="Loop"
onStart="context.iterationStateStore['count'] = 0" />

        </State>

        <State name="Loop">

            <!-- onStart will increment counter -->
            <Action type="output" onStart="context.iterationStateStore['count'] =
context.iterationStateStore['count'] + 1">
                <DataModel ref="TheDataModel" />
            </Action>

            <!-- Loop until our counter is greater than 3 -->
            <Action type="changeState" ref="Loop"
when="context.iterationStateStore['count'] < 3" />

        </State>

    </StateModel>

    <Test name="Default">
        <StateModel ref="State"/>

        <Publisher class="Console"/>
    </Test>

</Peach>

```

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 28742.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.ChangeState
Peach.Core.Dom.Action Changing to state: Loop
Peach.Core.Dom.StateModel Run(): Changing to state "Loop".
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(9 bytes)
Looping! ①
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.ChangeState
Peach.Core.Dom.Action Changing to state: Loop
Peach.Core.Dom.StateModel Run(): Changing to state "Loop".
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(9 bytes)
Looping! ②
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.ChangeState
Peach.Core.Dom.Action Changing to state: Loop
Peach.Core.Dom.StateModel Run(): Changing to state "Loop".
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(9 bytes)
Looping! ③
Peach.Core.Dom.Action Run: action 'Action_1' when returned false ④
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

① Output from iteration 1

② Output from iteration 2

③ Output from iteration 3

④ *when* expression returning false causing exit from loop

21.7.2. State

State elements reside in [StateModels](#). Each State element contains a logical Peach work unit.

Each [StateModel](#) must contain at least one State element. If you wish to model complex protocols or systems, use multiple State elements within your [StateModel](#).

Syntax

```
<StateModel name="StateModel" initialState="InitialState">
    <State name="InitialState">

        <Action name="SendData1" type="output">
            <DataModel ref="MyDataModel1" />
        </Action>

        <Action name="SendData2" type="output">
            <DataModel ref="MyDataModel2" />
        </Action>

    </State>
</StateModel>
```

State elements contain one or more Action elements. Each Action element can perform tasks related to how the individual State element consolidates logic.

Within a State element, actions execute in sequence from top to bottom, in the same order they appear in the State definition.

State elements can occur linearly by chaining from one state to another. Yet, Peach supports state changes that permit branching and looping among the State elements. Branches and loops in the StateModel are common in fuzzing scenarios where the input drives a response or a choice of responses.

- The *initialState* attribute in the [StateModel](#) declaration determines the first State element of the [StateModel](#) to run.
- Peach transitions into a new State after a *changeState* action.



If a set of Action elements always executes in the same order, place the set of actions in a single State element.

Attributes

Required:

name

The name of the State element.

Optional:

onStart

Scripting statement run prior to any actions.

onComplete

Scripting statement run after explicit actions.

Child Elements

Action

One or more Actions are required.

Examples

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

  <DataModel name="Hello">
    <String value="Hello\n" token="true"/>
  </DataModel>

  <DataModel name="Question">
    <String value="Pick a number between 1 and 5!" />
  </DataModel>

  <DataModel name="Wrong">
    <String value="Wrong!" />
  </DataModel>

  <DataModel name="Close">
    <String value="Very Close! Try Again!\n" />
  </DataModel>

  <DataModel name="Correct">
    <String value="Correct!" />
  </DataModel>

  <DataModel name="Selection">
    <Choice name="TheSelection">
      <String name="one" value="1" length="1" token="true" />
```

```

        <String name="two" value="2" length="1" token="true" />
        <String name="three" value="3" length="1" token="true" />
        <String name="rest" length="1" />
    </Choice>
    <String value="\n" token="true"/>
</DataModel>

<StateModel name="TheStateModel" initialState="InitialState">
    <State name="InitialState">
        <Action name="AcceptConnection" type="accept" />

        <Action name="HelloIn" type="input">
            <DataModel ref="Hello"/>
        </Action>

        <Action type="changeState" ref="AskState" />
    </State>

    <State name="AskState">
        <Action name="AskTheQuestion" type="output">
            <DataModel ref="Question" />
        </Action>

        <Action name="TheirAnswer" type="input">
            <DataModel name="TheirSelection" ref="Selection"/>
        </Action>

        <Action type="changeState" ref="TryAgain" when=
"state.actions[1].dataModel.find('one') is not None or
State.actions[1].dataModel.find('three') is not None"/>
        <Action type="changeState" ref="Win" when=
"state.actions[1].dataModel.find('two') is not None"/>
        <Action type="changeState" ref="Lose" />
    </State>

    <State name="TryAgain">
        <Action type="output">
            <DataModel ref="Close" />
        </Action>
        <Action type="changeState" ref="AskState" />
    </State>

    <State name="Win">
        <Action type="output">
            <DataModel ref="Correct" />
        </Action>
    </State>

```

```
<State name="Lose">
    <Action type="output">
        <DataModel ref="Wrong" />
    </Action>
</State>

</StateModel>

<Test name="Default">
    <StateModel ref="TheStateManager"/>
    <Publisher class="TcpListener">
        <Param name="Interface" value="0.0.0.0" />
        <Param name="Port" value="31337" />
        <Param name="AcceptTimeout" value="10000" />
        <Param name="Timeout" value="10000" />
    </Publisher>

    <Strategy class="Random"/>

    <Logger class="File" >
        <Param name="Path" value="logs"/>
    </Logger>
</Test>
</Peach>
```

21.7.3. Action

Action elements are part of the [StateModel](#), and send commands to a [Publisher](#). Within the StateModel, Action elements are child elements of [State](#).

Action elements set up and manage state in a fuzzing session. They issue commands to set up and control the Peach environment. For example, the *open* action initializes a listener by initiating a network connection or opening a file handle. And, the *close* action releases resources at the end of a fuzzing iteration.

Action elements provide control over state changes, movement of data between data models, and calling custom methods defined by agents. For example, the *changeState* action initiates a transition to a new state in the *StateModel*.

Additionally, Action elements issue directives to Publishers to do things such as read input, send output, and access a property.

Four Actions elements are implicit: *start*, *stop*, *open*, and *close*. They execute automatically and do not need to be declared in the StateModel.

- The *start* and *stop* actions occur at the start and end of a fuzzing session.
- The *open* and *close* actions occur at the start and end of each iteration in a fuzzing session.

Other actions require explicit declarations in order to execute.

Default Order of Actions

The following is the default order in which Actions are performed when fuzzing:

1. *start* - Implicit, once per session
2. *open* - Implicit, once per iteration
3. Explicit actions (*accept*, *input*, *output*, *web*, etc.)
4. *close* - Implicit, once per iteration
5. *stop* - Implicit, once per session

Syntax

```
<StateModel name="StateModel" initialState="InitialState">
  <State name="InitialState">

    <Action name="SendData" type="output">
      <DataModel ref="MyDataModel" />

      <!-- Optional data element -->
      <Data name="load defaults" fileName="template.bin" />
    </Action>

  </State>
</StateModel>
```

Complicated StateModels that have counters (or other advanced operations) use Action parameters that execute scripts at certain times (such as before the Action starts or once it has completed).

Since Pits can contain multiple Publishers within a Test, many Actions have a parameter to declare the specific Publisher that carries out the command.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="Ping">
        <String value="PING" token="true"/>
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState">
        <State name="InitialState">

            <Action name="PingPacket" type="output" publisher="TheTCPPub">
                <DataModel ref="Ping"/>
            </Action>

            <Action name="PingUDPPacket" type="output" publisher="TheUDPPub">
                <DataModel ref="Ping"/>
            </Action>

        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>
        <Publisher name="TheTCPPub" class="Tcp">
            <Param name="Host" value="127.0.0.1" />
            <Param name="Port" value="31337" />
        </Publisher>
        <Publisher name="TheUDPPub" class="Udp">
            <Param name="Host" value="127.0.0.1" />
            <Param name="Port" value="1337" />
        </Publisher>

        <Strategy class="Random"/>

        <Logger class="File" >
            <Param name="Path" value="logs"/>
        </Logger>
    </Test>
</Peach>

```

Attributes

Required:

name

Name of the action.

type

Action type. The following enumeration summarizes the available action types. Detailed descriptions of the actions follow the child elements.

- [accept](#) - temporarily blocks execution.
- [call](#) - provides a method-calling metaphor.
- [changeState](#) - transitions from one state to another within the [StateModel](#).
- [close](#) - causes the associated Publisher to close.
- [getProperty](#) - provides the retrieval function of a property metaphor (get function).
- [input](#) - reads input data using the Publisher.
- [infrag](#) - perform one or more input actions to support protocol fragmentation.
- [open \(or connect\)](#) - causes the associated Publisher to open its resources.
- [output](#) - causes the associated Publisher to write data.
- [outfrag](#) - perform one or more output actions to support protocol fragmentation.
- [setProperty](#) - provides the output function of a property metaphor (set function) that modifies the value of the property.
- [slurp](#) - copies a value from a data element in one DataModel to a data element in another DataModel.
- [start](#) - causes a Publisher to perform its initialization tasks.
- [stop](#) - causes a Publisher to perform final cleanup.
- [web](#) - sent web api/http requests via WebApi publisher.

Required Based on Type:

valueXpath

Path to the source element defined using XPath notation when the Action type is [slurp](#).

setXpath

Path to the destination element defined using XPath notation when Action type is [slurp](#).

ref

Reference of the state to change to when the Action type is [changeState](#).

property

Name of the property to get or set in the publisher when the Action type is [setProperty](#) or [getProperty](#).

Optional:

publisher

Name of the Publisher, or **Peach.Agent**, to perform the action.

onComplete

Expression to evaluate when an action completes.

onStart

Expression to evaluate at the start of an action.

when

Perform the action if the provided expression evaluates to true.

Child Elements

DataModel

DataModel to fuzz.

Data

Set of initial data to crack into the DataModel before fuzzing.

Param

Argument passed with the call. The argument will be fuzzed.

Result

Output of the call Action. The result will be cracked into a DataModel.

start

The *start* action causes a [Publisher](#) to perform the initialization that usually occurs at the start of a fuzzing session. *start* normally occurs once per fuzzing session, at the start of session. Another Action element, [stop](#), also normally occurs once per fuzzing session, at the end of the session.

start is an implicit Peach action that should not be declared in the [StateModel](#) unless a change is needed in the default behavior.

NOTE: Including the *start* or *stop* action in the StateModel causes the action to run every iteration instead of once per session. This declaration should be avoided unless specifically required.

Not all [Publishers](#) make use of *start* and *stop*.

Default Order of Actions

The following is the default Actions order when fuzzing:

1. *start* - Implicit, once per session
2. *open* - Implicit, once per iteration
3. Explicit actions (such as *accept*, *input* and *output*)
4. *close* - Implicit, once per iteration
5. *stop* - Implicit, once per session

Syntax

```
<StateModel name="TheStateModel" initialState="InitialState">
    <State name="InitialState">
        <Action type="start" />
    </State>
</StateModel>
```

Attributes

Required:

type

Action type, must be set to "start"

Optional:

name

Name used to identify the action

publisher

Name of the publisher to perform this action

when

Only perform the action if the provided expression evaluates to true

onStart

Expression to run at the start of an action.

onComplete

Expression to run at the completion of an action

Child Elements

None.

Examples

Example 169. Default behavior

The Example Pit shows normal (default) behavior. Declarations for *start* and *stop* are unnecessary and not included. The output follows the Pit definition.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="Ping">
        <String value="PING" token="true"/>
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState">
        <State name="InitialState">

            <!-- Start action is implicitly called once per session by Peach.
                It will cause the Publisher to perform any initialization. -->

            <!-- Open action is implicitly called on every iteration by Peach.
                It will cause the Publisher to open a connection to the remote host. -->

            <!-- This action is called on every iteration. It will output data from
                the data model "Ping" using our Publisher. -->
            <Action type="output">
                <DataModel ref="Ping"/>
            </Action>

            <!-- Close action is implicitly called on every iteration by Peach.
                It will cause the Publisher to close its connection to the remote
                host if it's still open. -->

            <!-- This action is implicitly called once per fuzzing session. It will perform
                any
                final clean up actions to fully stop the Publisher. -->

        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>
        <Publisher class="Tcp">
            <Param name="Host" value="127.0.0.1" />
            <Param name="Port" value="31337" />
        </Publisher>

        <Logger class="File" >
            <Param name="Path" value="logs"/>
        </Logger>
    </Test>
</Peach>

```

Output showing the default behavior of the sample pit where the *start* action is called once at the beginning of the fuzzing session and the *stop* action is called once at the end of the fuzzing session.

```
> peach --range 1,3 --seed 51405 example.xml

[[ Peach Professional v3.0.0
[[ Copyright (c) Peach Fuzzer LLC
Peach.Core.Engine runTest: context.config.range == true, start: 1, stop: 3

[*] Test 'Default' starting with random seed 51405.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpClientPublisher start()                                     ①
Peach.Core.Publishers.TcpClientPublisher open()
Peach.Core.Publishers.TcpClientPublisher output(4 bytes)
Peach.Core.Publishers.TcpClientPublisher

00000000  50 49 4E 47                           PING

Peach.Core.Publishers.TcpClientPublisher close()
Peach.Core.Publishers.TcpClientPublisher Shutting down connection to 127.0.0.1:31337
Peach.Core.Publishers.TcpClientPublisher Read 0 bytes from 127.0.0.1:31337, closing
client connection.
Peach.Core.Publishers.TcpClientPublisher Closing connection to 127.0.0.1:31337

[1,3,0:00:01.614] Performing iteration
[*] Fuzzing: Ping.DataElement_0
[*] Mutator: DataElementSwapNearNodesMutator
Peach.Core.MutationStrategies.RandomStrategy Action_Starting: Fuzzing:
Ping.DataElement_0
Peach.Core.MutationStrategies.RandomStrategy Action_Starting: Mutator:
DataElementSwapNearNodesMutator
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpClientPublisher open()
Peach.Core.Publishers.TcpClientPublisher output(4 bytes)
Peach.Core.Publishers.TcpClientPublisher

00000000  50 49 4E 47                           PING

Peach.Core.Publishers.TcpClientPublisher close()
Peach.Core.Publishers.TcpClientPublisher Shutting down connection to 127.0.0.1:31337
Peach.Core.Publishers.TcpClientPublisher Read 0 bytes from 127.0.0.1:31337, closing
client connection.
```

Peach.Core.Publisher.TcpClientPublisher Closing connection to 127.0.0.1:31337

```
[2,3,0:00:03.438] Performing iteration
[*] Fuzzing: Ping.DataElement_0
[*] Mutator: UnicodeUtf8ThreeCharMutator
Peach.Core.MutationStrategies.RandomStrategy Action_Starting: Fuzzing:
Ping.DataElement_0
Peach.Core.MutationStrategies.RandomStrategy Action_Starting: Mutator:
UnicodeUtf8ThreeCharMutator
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpClientPublisher open()
Peach.Core.Publishers.TcpClientPublisher output(522 bytes)
Peach.Core.Publishers.TcpClientPublisher
```

```

Peach.Core.Publisher.TcpClientPublisher close()
Peach.Core.Publisher.TcpClientPublisher Shutting down connection to 127.0.0.1:31337
Peach.Core.Publisher.TcpClientPublisher Read 0 bytes from 127.0.0.1:31337, closing
client connection.
Peach.Core.Publisher.TcpClientPublisher Closing connection to 127.0.0.1:31337

[3,3,0:00:01.454] Performing iteration
[*] Fuzzing: Ping.DataElement_0
[*] Mutator: UnicodeUtf8ThreeCharMutator
Peach.Core.MutationStrategies.RandomStrategy Action_Start: Fuzzing:
Ping.DataElement_0
Peach.Core.MutationStrategies.RandomStrategy Action_Start: Mutator:
UnicodeUtf8ThreeCharMutator
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publisher.TcpClientPublisher open()
Peach.Core.Publisher.TcpClientPublisher output(1968 bytes)
Peach.Core.Publisher.TcpClientPublisher

00000000 EF 83 B0 EF 83 B0 EF 83 B0 EF 83 B0 EF ..... .
00000010 83 B0 EF 83 B0 EF 83 B0 EF 83 B0 EF 83 ..... .
00000020 B0 EF 83 B0 EF 83 B0 EF 83 B0 EF 83 B0 EF ..... .
00000030 EF 83 B0 EF 83 B0 EF 83 B0 EF 83 B0 EF ..... .
00000040 83 B0 EF 83 B0 EF 83 B0 EF 83 B0 EF 83 ..... .
00000050 B0 EF 83 B0 EF 83 B0 EF 83 B0 EF 83 B0 EF ..... .
00000060 EF 83 B0 EF 83 B0 EF 83 B0 EF 83 B0 EF ..... .
00000070 83 B0 EF ..... .
00000080 B0 EF 83 B0 EF 83 B0 EF 83 B0 EF 83 B0 EF ..... .
.....
Peach.Core.Publisher.TcpClientPublisher close()
Peach.Core.Publisher.TcpClientPublisher Shutting down connection to 127.0.0.1:31337
Peach.Core.Publisher.TcpClientPublisher Read 0 bytes from 127.0.0.1:31337, closing
client connection.
Peach.Core.Publisher.TcpClientPublisher Closing connection to 127.0.0.1:31337
Peach.Core.Publisher.TcpClientPublisher stop() ②

[*] Test 'Default' finished.

```

① Start action at beginning of fuzzing session

② Stop action at end of fuzzing session

Example 170. Explicit calling of start

The Example Pit shows explicit declaration of the *start* and *stop* actions. The *start* and *stop* declarations cause them to be called every iteration, as seen in the output listing.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="Ping">
        <String value="PING" token="true"/>
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState">
        <State name="InitialState">

            <!-- This action is called on every iteration. It will cause
                the Tcp Publisher to perform any initialization. -->
            <Action type="start" />

            <!-- This action is called on every iteration. It will cause the
                Tcp Publisher to open a connection to the remote host. -->
            <Action type="open" />

            <!-- This action is called on every iteration. It will output data from
                the data model "Ping" using our Publisher. -->
            <Action type="output">
                <DataModel ref="Ping"/>
            </Action>

            <!-- This action is called on every iteration. It will cause the
                Tcp Publisher to close its connection to the remote host if it's
                still open. -->
            <Action type="close" />

            <!-- This action is called on every iteration. It will perform any
                final clean up actions to fully stop the Publisher. -->
            <Action type="stop" />

        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>
        <Publisher class="Tcp">
            <Param name="Host" value="127.0.0.1" />
            <Param name="Port" value="31337" />
        </Publisher>

        <Logger class="File" >
            <Param name="Path" value="logs"/>
        </Logger>
    
```

```
</Test>
</Peach>
```

In the following output, the *start* action executes at the beginning of every iteration and the *stop* action follows at the end of every iteration.

```
> peach --range 1,3 --seed 51405 example.xml

[[ Peach Professional v3.0.0
[[ Copyright (c) Peach Fuzzer LLC
Peach.Core.Engine runTest: context.config.range == true, start: 1, stop: 3

[*] Test 'Default' starting with random seed 51405.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Start
Peach.Core.Publishers.TcpClientPublisher start() ①
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Open
Peach.Core.Publishers.TcpClientPublisher open()
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpClientPublisher output(4 bytes)
Peach.Core.Publishers.TcpClientPublisher

00000000  50 49 4E 47                      PING

Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.TcpClientPublisher close()
Peach.Core.Publishers.TcpClientPublisher Shutting down connection to 127.0.0.1:31337
Peach.Core.Publishers.TcpClientPublisher Read 0 bytes from 127.0.0.1:31337, closing
client connection.
Peach.Core.Publishers.TcpClientPublisher Closing connection to 127.0.0.1:31337
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Stop
Peach.Core.Publishers.TcpClientPublisher stop() ②

[1,3,0:00:04.512] Performing iteration
Peach.Core.Dom.Action ActionType.Start
Peach.Core.Publishers.TcpClientPublisher start() ③
Peach.Core.Dom.Action ActionType.Open
Peach.Core.Publishers.TcpClientPublisher open()
[*] Fuzzing: Ping.DataElement_0
[*] Mutator: DataElementSwapNearNodesMutator
```

```

Peach.Core.MutationStrategies.RandomStrategy Action_Starting: Fuzzing:
Ping.DataElement_0
Peach.Core.MutationStrategies.RandomStrategy Action_Starting: Mutator:
DataElementSwapNearNodesMutator
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpClientPublisher output(4 bytes)
Peach.Core.Publishers.TcpClientPublisher

00000000  50 49 4E 47                                PING

Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.TcpClientPublisher close()
Peach.Core.Publishers.TcpClientPublisher Shutting down connection to 127.0.0.1:31337
Peach.Core.Publishers.TcpClientPublisher Read 0 bytes from 127.0.0.1:31337, closing
client connection.
Peach.Core.Publishers.TcpClientPublisher Closing connection to 127.0.0.1:31337
Peach.Core.Dom.Action ActionType.Stop
Peach.Core.Publishers.TcpClientPublisher stop()          ④

[2,3,0:00:05.246] Performing iteration
Peach.Core.Dom.Action ActionType.Start
Peach.Core.Publishers.TcpClientPublisher start()          ⑤
Peach.Core.Dom.Action ActionType.Open
Peach.Core.Publishers.TcpClientPublisher open()
[*] Fuzzing: Ping.DataElement_0
[*] Mutator: UnicodeUtf8ThreeCharMutator
Peach.Core.MutationStrategies.RandomStrategy Action_Starting: Fuzzing:
Ping.DataElement_0
Peach.Core.MutationStrategies.RandomStrategy Action_Starting: Mutator:
UnicodeUtf8ThreeCharMutator
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpClientPublisher output(522 bytes)
Peach.Core.Publishers.TcpClientPublisher

00000000  EF 83 B0 EF 83 B0 EF 83 B0 EF 83 B0 EF ..... .
00000010  83 B0 EF ..... .
00000020  B0 EF 83 B0 EF ..... .
00000030  EF 83 B0 EF ..... .
00000040  83 B0 EF ..... .
00000050  B0 EF 83 B0 EF ..... .
00000060  EF 83 B0 EF ..... .
00000070  83 B0 EF ..... .
00000080  B0 EF 83 B0 EF ..... .
00000090  EF 83 B0 EF ..... .
000000A0  83 B0 EF ..... .
000000B0  B0 EF 83 B0 EF ..... .
000000C0  EF 83 B0 EF ..... .
000000D0  83 B0 EF ..... .

```

000000E0	B0 EF 83 B0
000000F0	EF 83 B0 EF
00000100	83 B0 EF 83
00000110	B0 EF 83 B0
00000120	EF 83 B0 EF
00000130	83 B0 EF 83
00000140	B0 EF 83 B0
00000150	EF 83 B0 EF
00000160	83 B0 EF 83
00000170	B0 EF 83 B0
00000180	EF 83 B0 EF
00000190	83 B0 EF 83
000001A0	B0 EF 83 B0
000001B0	EF 83 B0 EF
000001C0	83 B0 EF 83
000001D0	B0 EF 83 B0
000001E0	EF 83 B0 EF
000001F0	83 B0 EF 83
00000200	B0 EF 83 B0 EF 83 B0 EF 83 B0 EF 83 B0

```
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.TcpClientPublisher close()
Peach.Core.Publishers.TcpClientPublisher Shutting down connection to 127.0.0.1:31337
Peach.Core.Publishers.TcpClientPublisher Read 0 bytes from 127.0.0.1:31337, closing
client connection.
Peach.Core.Publishers.TcpClientPublisher Closing connection to 127.0.0.1:31337
Peach.Core.Dom.Action ActionType.Stop
Peach.Core.Publishers.TcpClientPublisher stop() ⑥
```

```
[3,3,0:00:01.705] Performing iteration
Peach.Core.Dom.Action ActionType.Start
Peach.Core.Publishers.TcpClientPublisher start()
Peach.Core.Dom.Action ActionType.Open
Peach.Core.Publishers.TcpClientPublisher open()
[*] Fuzzing: Ping.DataElement_0
[*] Mutator: UnicodeUtf8ThreeCharMutator
Peach.Core.MutationStrategies.RandomStrategy Action_Starting: Fuzzing:
Ping.DataElement_0
Peach.Core.MutationStrategies.RandomStrategy Action_Starting: Mutator:
UnicodeUtf8ThreeCharMutator
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpClientPublisher output(1968 bytes)
Peach.Core.Publishers.TcpClientPublisher
```

00000000	EF 83 B0 EF
00000010	83 B0 EF 83
00000020	B0 FF 83 B0

■ ■ ■ ■

```
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.TcpClientPublisher close()
Peach.Core.Publishers.TcpClientPublisher Shutting down connection to 127.0.0.1:31337
Peach.Core.Publishers.TcpClientPublisher Read 0 bytes from 127.0.0.1:31337, closing
client connection.
Peach.Core.Publishers.TcpClientPublisher Closing connection to 127.0.0.1:31337
Peach.Core.Dom.Action ActionType.Stop
Peach.Core.Publishers.TcpClientPublisher stop() ⑧
```

8

- ① *Start* action at beginning of iteration
 - ② *Stop* action at end of iteration
 - ③ *Start* action at beginning of iteration
 - ④ *Stop* action at end of iteration
 - ⑤ *Start* action at beginning of iteration
 - ⑥ *Stop* action at end of iteration
 - ⑦ *Start* action at beginning of iteration
 - ⑧ *Stop* action at end of iteration

stop

The *stop* action causes a [Publisher](#) to perform the final cleanup that usually occurs once, at the end of a fuzzing session. Another Action element, [*start*](#), also normally occurs once per fuzzing session, at the start of the session.

stop is an implicit Peach action that should not be declared in the [StateModel](#) unless a change is needed in the default behavior.

NOTE: Including the *stop* or *start* action in the StateModel causes the action to run every iteration instead of once per session. This declaration should be avoided unless specifically required.

Not all [Publishers](#) make use of stop and start.

Default Order of Actions

The following is the default Actions order when fuzzing:

1. start - Implicit, once per session
2. open - Implicit, once per iteration
3. Explicit actions (such as accept, input and output)
4. close - Implicit, once per iteration
5. stop - Implicit, once per session

Syntax

```
<StateModel name="TheStateModel" initialState="InitialState">
  <State name="InitialState">
    <Action type="stop" />
  </State>
</StateModel>
```

Attributes

Required:

type

Action type, must be set to "stop"

Optional:

name

Name used to identify the action

publisher

Name of the publisher that this action should be called on

when

Only perform action if the expression provided evaluates to true

onStart

Expression to run on start of an action.

onComplete

Expression to run on completion of an action

Child Elements

None.

Examples**Example 171. Default behavior**

Example Pit shows normal (default) behavior. Note that the *stop* and *start* actions are not declared. The output follows the example pit.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="Ping">
        <String value="PING" token="true"/>
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState">
        <State name="InitialState">

            <!-- Start action is implicitly called once per session by Peach.
                It will cause the Publisher to perform any initialization. -->

            <!-- Open action is implicitly called on every iteration by Peach.
                It will cause the Publisher to open a connection to the remote host. -->

            <!-- This action is called on every iteration. It will output data from
                the data model "Ping" using our Publisher. -->
            <Action type="output">
                <DataModel ref="Ping"/>
            </Action>

            <!-- Close action is implicitly called on every iteration by Peach.
                It will cause the Publisher to close its connection to the remote
                host if it's still open. -->

            <!-- This action is implicitly called once per fuzzing session. It will perform
                any
                final clean up actions to fully stop the Publisher. -->

        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>
        <Publisher class="Tcp">
            <Param name="Host" value="127.0.0.1" />
            <Param name="Port" value="31337" />
        </Publisher>

        <Logger class="File" >
            <Param name="Path" value="logs"/>
        </Logger>
    </Test>
</Peach>

```

Output showing the default behavior of the sample pit with implicit usage for *start* and *stop*. The *start* action is called once at the beginning of the fuzzing session and the *stop* action is called once at the end of the fuzzing session.

```
> peach --range 1,3 --seed 51405 example.xml

[[ Peach Professional v3.0.0
[[ Copyright (c) Peach Fuzzer LLC
Peach.Core.Engine runTest: context.config.range == true, start: 1, stop: 3

[*] Test 'Default' starting with random seed 51405.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpClientPublisher start() ①
Peach.Core.Publishers.TcpClientPublisher open()
Peach.Core.Publishers.TcpClientPublisher output(4 bytes)
Peach.Core.Publishers.TcpClientPublisher

00000000  50 49 4E 47                      PING

Peach.Core.Publishers.TcpClientPublisher close()
Peach.Core.Publishers.TcpClientPublisher Shutting down connection to 127.0.0.1:31337
Peach.Core.Publishers.TcpClientPublisher Read 0 bytes from 127.0.0.1:31337, closing
client connection.
Peach.Core.Publishers.TcpClientPublisher Closing connection to 127.0.0.1:31337

[1,3,0:00:01.614] Performing iteration
[*] Fuzzing: Ping.DataElement_0
[*] Mutator: DataElementSwapNearNodesMutator
Peach.Core.MutationStrategies.RandomStrategy Action_Starting: Fuzzing:
Ping.DataElement_0
Peach.Core.MutationStrategies.RandomStrategy Action_Starting: Mutator:
DataElementSwapNearNodesMutator
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpClientPublisher open()
Peach.Core.Publishers.TcpClientPublisher output(4 bytes)
Peach.Core.Publishers.TcpClientPublisher

00000000  50 49 4E 47                      PING

Peach.Core.Publishers.TcpClientPublisher close()
Peach.Core.Publishers.TcpClientPublisher Shutting down connection to 127.0.0.1:31337
Peach.Core.Publishers.TcpClientPublisher Read 0 bytes from 127.0.0.1:31337, closing
client connection.
```

Peach.Core.Publisher.TcpClientPublisher Closing connection to 127.0.0.1:31337

```
[2,3,0:00:03.438] Performing iteration
[*] Fuzzing: Ping.DataElement_0
[*] Mutator: UnicodeUtf8ThreeCharMutator
Peach.Core.MutationStrategies.RandomStrategy Action_Starting: Fuzzing:
Ping.DataElement_0
Peach.Core.MutationStrategies.RandomStrategy Action_Starting: Mutator:
UnicodeUtf8ThreeCharMutator
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpClientPublisher open()
Peach.Core.Publishers.TcpClientPublisher output(522 bytes)
Peach.Core.Publishers.TcpClientPublisher
```

```

Peach.Core.Publisher.TcpClientPublisher close()
Peach.Core.Publisher.TcpClientPublisher Shutting down connection to 127.0.0.1:31337
Peach.Core.Publisher.TcpClientPublisher Read 0 bytes from 127.0.0.1:31337, closing
client connection.
Peach.Core.Publisher.TcpClientPublisher Closing connection to 127.0.0.1:31337

[3,3,0:00:01.454] Performing iteration
[*] Fuzzing: Ping.DataElement_0
[*] Mutator: UnicodeUtf8ThreeCharMutator
Peach.Core.MutationStrategies.RandomStrategy Action_Start: Fuzzing:
Ping.DataElement_0
Peach.Core.MutationStrategies.RandomStrategy Action_Start: Mutator:
UnicodeUtf8ThreeCharMutator
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publisher.TcpClientPublisher open()
Peach.Core.Publisher.TcpClientPublisher output(1968 bytes)
Peach.Core.Publisher.TcpClientPublisher

00000000 EF 83 B0 EF 83 B0 EF 83 B0 EF 83 B0 EF ..... .
00000010 83 B0 EF 83 B0 EF 83 B0 EF 83 B0 EF 83 ..... .
00000020 B0 EF 83 B0 EF 83 B0 EF 83 B0 EF 83 B0 EF ..... .
00000030 EF 83 B0 EF 83 B0 EF 83 B0 EF 83 B0 EF ..... .
00000040 83 B0 EF 83 B0 EF 83 B0 EF 83 B0 EF 83 ..... .
00000050 B0 EF 83 B0 EF 83 B0 EF 83 B0 EF 83 B0 EF ..... .
00000060 EF 83 B0 EF 83 B0 EF 83 B0 EF 83 B0 EF ..... .
00000070 83 B0 EF ..... .
00000080 B0 EF 83 B0 EF 83 B0 EF 83 B0 EF 83 B0 EF ..... .
.....
Peach.Core.Publisher.TcpClientPublisher close()
Peach.Core.Publisher.TcpClientPublisher Shutting down connection to 127.0.0.1:31337
Peach.Core.Publisher.TcpClientPublisher Read 0 bytes from 127.0.0.1:31337, closing
client connection.
Peach.Core.Publisher.TcpClientPublisher Closing connection to 127.0.0.1:31337
Peach.Core.Publisher.TcpClientPublisher stop() ②

[*] Test 'Default' finished.

```

① Start action at beginning of fuzzing session

② Stop action at end of fuzzing session

Example 172. Explicit calling of stop

Example Pit that declares the *start* and *stop* actions. The declaration causes *start* and *stop* to execute on every iteration, as indicated in the output listing.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="Ping">
        <String value="PING" token="true"/>
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState">
        <State name="InitialState">

            <!-- This action is called on every iteration. It will cause
                the Tcp Publisher to perform any initialization. -->
            <Action type="start" />

            <!-- This action is called on every iteration. It will cause the
                Tcp Publisher to open a connection to the remote host. -->
            <Action type="open" />

            <!-- This action is called on every iteration. It will output data from
                the data model "Ping" using our Publisher. -->
            <Action type="output">
                <DataModel ref="Ping"/>
            </Action>

            <!-- This action is called on every iteration. It will cause the
                Tcp Publisher to close its connection to the remote host if it's
                still open. -->
            <Action type="close" />

            <!-- This action is called on every iteration. It will perform any
                final clean up actions to fully stop the Publisher. -->
            <Action type="stop" />

        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>
        <Publisher class="Tcp">
            <Param name="Host" value="127.0.0.1" />
            <Param name="Port" value="31337" />
        </Publisher>

        <Logger class="File" >
            <Param name="Path" value="logs"/>
        </Logger>
    
```

```
</Test>
</Peach>
```

In the following output, the *start* action executes at the beginning of every iteration and is followed by a *stop* action at the end of every iteration.

```
> peach --range 1,3 --seed 51405 example.xml

[[ Peach Professional v3.0.0
[[ Copyright (c) Peach Fuzzer LLC
Peach.Core.Engine runTest: context.config.range == true, start: 1, stop: 3

[*] Test 'Default' starting with random seed 51405.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Start
Peach.Core.Publishers.TcpClientPublisher start() ①
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Open
Peach.Core.Publishers.TcpClientPublisher open()
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpClientPublisher output(4 bytes)
Peach.Core.Publishers.TcpClientPublisher

00000000  50 49 4E 47                      PING

Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.TcpClientPublisher close()
Peach.Core.Publishers.TcpClientPublisher Shutting down connection to 127.0.0.1:31337
Peach.Core.Publishers.TcpClientPublisher Read 0 bytes from 127.0.0.1:31337, closing
client connection.
Peach.Core.Publishers.TcpClientPublisher Closing connection to 127.0.0.1:31337
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Stop
Peach.Core.Publishers.TcpClientPublisher stop() ②

[1,3,0:00:04.512] Performing iteration
Peach.Core.Dom.Action ActionType.Start
Peach.Core.Publishers.TcpClientPublisher start() ③
Peach.Core.Dom.Action ActionType.Open
Peach.Core.Publishers.TcpClientPublisher open()
[*] Fuzzing: Ping.DataElement_0
[*] Mutator: DataElementSwapNearNodesMutator
```

```

Peach.Core.MutationStrategies.RandomStrategy Action_Starting: Fuzzing:
Ping.DataElement_0
Peach.Core.MutationStrategies.RandomStrategy Action_Starting: Mutator:
DataElementSwapNearNodesMutator
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpClientPublisher output(4 bytes)
Peach.Core.Publishers.TcpClientPublisher

00000000  50 49 4E 47                                PING

Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.TcpClientPublisher close()
Peach.Core.Publishers.TcpClientPublisher Shutting down connection to 127.0.0.1:31337
Peach.Core.Publishers.TcpClientPublisher Read 0 bytes from 127.0.0.1:31337, closing
client connection.
Peach.Core.Publishers.TcpClientPublisher Closing connection to 127.0.0.1:31337
Peach.Core.Dom.Action ActionType.Stop
Peach.Core.Publishers.TcpClientPublisher stop()          ④

[2,3,0:00:05.246] Performing iteration
Peach.Core.Dom.Action ActionType.Start
Peach.Core.Publishers.TcpClientPublisher start()          ⑤
Peach.Core.Dom.Action ActionType.Open
Peach.Core.Publishers.TcpClientPublisher open()
[*] Fuzzing: Ping.DataElement_0
[*] Mutator: UnicodeUtf8ThreeCharMutator
Peach.Core.MutationStrategies.RandomStrategy Action_Starting: Fuzzing:
Ping.DataElement_0
Peach.Core.MutationStrategies.RandomStrategy Action_Starting: Mutator:
UnicodeUtf8ThreeCharMutator
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpClientPublisher output(522 bytes)
Peach.Core.Publishers.TcpClientPublisher

00000000  EF 83 B0 EF 83 B0 EF 83 B0 EF 83 B0 EF ..... .
00000010  83 B0 EF ..... .
00000020  B0 EF 83 B0 EF ..... .
00000030  EF 83 B0 EF ..... .
00000040  83 B0 EF ..... .
00000050  B0 EF 83 B0 EF ..... .
00000060  EF 83 B0 EF ..... .
00000070  83 B0 EF ..... .
00000080  B0 EF 83 B0 EF ..... .
00000090  EF 83 B0 EF ..... .
000000A0  83 B0 EF ..... .
000000B0  B0 EF 83 B0 EF ..... .
000000C0  EF 83 B0 EF ..... .
000000D0  83 B0 EF ..... .

```

000000E0	B0 EF 83 B0
000000F0	EF 83 B0 EF
00000100	83 B0 EF 83
00000110	B0 EF 83 B0
00000120	EF 83 B0 EF
00000130	83 B0 EF 83
00000140	B0 EF 83 B0
00000150	EF 83 B0 EF
00000160	83 B0 EF 83
00000170	B0 EF 83 B0
00000180	EF 83 B0 EF
00000190	83 B0 EF 83
000001A0	B0 EF 83 B0
000001B0	EF 83 B0 EF
000001C0	83 B0 EF 83
000001D0	B0 EF 83 B0
000001E0	EF 83 B0 EF
000001F0	83 B0 EF 83
00000200	B0 EF 83 B0 EF 83 B0 EF 83 B0 EF 83 B0

```
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.TcpClientPublisher close()
Peach.Core.Publishers.TcpClientPublisher Shutting down connection to 127.0.0.1:31337
Peach.Core.Publishers.TcpClientPublisher Read 0 bytes from 127.0.0.1:31337, closing
client connection.
Peach.Core.Publishers.TcpClientPublisher Closing connection to 127.0.0.1:31337
Peach.Core.Dom.Action ActionType.Stop
Peach.Core.Publishers.TcpClientPublisher stop() ⑥
```

```
[3,3,0:00:01.705] Performing iteration
Peach.Core.Dom.Action ActionType.Start
Peach.Core.Publishers.TcpClientPublisher start()
Peach.Core.Dom.Action ActionType.Open
Peach.Core.Publishers.TcpClientPublisher open()
[*] Fuzzing: Ping.DataElement_0
[*] Mutator: UnicodeUtf8ThreeCharMutator
Peach.Core.MutationStrategies.RandomStrategy Action_Starting: Fuzzing:
Ping.DataElement_0
Peach.Core.MutationStrategies.RandomStrategy Action_Starting: Mutator:
UnicodeUtf8ThreeCharMutator
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpClientPublisher output(1968 bytes)
Peach.Core.Publishers.TcpClientPublisher
```

00000000	EF 83 B0 EF
00000010	83 B0 EF 83
00000020	B0 FF 83 B0

00000030	EF 83 B0 EF
00000040	83 B0 EF 83
00000050	B0 EF 83 B0
00000060	EF 83 B0 EF 83
00000070	83 B0 EF 83
00000080	B0 EF 83 B0
....

```
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.TcpClientPublisher close()
Peach.Core.Publishers.TcpClientPublisher Shutting down connection to 127.0.0.1:31337
Peach.Core.Publishers.TcpClientPublisher Read 0 bytes from 127.0.0.1:31337, closing
client connection.
Peach.Core.Publishers.TcpClientPublisher Closing connection to 127.0.0.1:31337
Peach.Core.Dom.Action ActionType.Stop
Peach.Core.Publishers.TcpClientPublisher stop() ⑧
```

- ① Start action at beginning of iteration
- ② Stop action at end of iteration
- ③ Start action at beginning of iteration
- ④ Stop action at end of iteration
- ⑤ Start action at beginning of iteration
- ⑥ Stop action at end of iteration
- ⑦ Start action at beginning of iteration
- ⑧ Stop action at end of iteration

open, connect

The *open* (or *connect*) action is an implicit action that causes the associated publisher to open its resource. The *open* Action executes by default at the start of each iteration.

Each Publisher is an I/O adapter and performs the *open* Action according to its resource needs, as in the following examples:

- The [File publisher](#) opens a file handle.
- The [TCP publisher](#) initiates a network connection.

The only time to declare the *open* action in the StateModel is when the default behavior needs to be modified.



connect is an alias for *open*. Both names can be used interchangeably.

Default Order of Actions

The following is the default order in which Actions are performed when fuzzing:

1. *start* - Implicit, once per session
2. *open* - Implicit, once per iteration
3. Explicit actions (*accept*, *input*, *output*, etc.)
4. *close* - Implicit, once per iteration
5. *stop* - Implicit, once per session

Syntax

```
<StateModel name="TheStateModel" initialState="InitialState">
    <State name="InitialState">

        <Action type="open" />

        <Action type="output">
            <DataModel ref="DataModelToWrite"/>
        </Action>

    </State>
</StateModel>
```

```
<StateModel name="TheStateModel" initialState="InitialState">
  <State name="InitialState">

    <Action type="connect" />

    <Action type="output">
      <DataModel ref="DataModelToSend"/>
    </Action>

  </State>
</StateModel>
```

Attributes

Required:

type

Action type, must be set to "open" or "connect"

Optional:

name

Name used to identify the action

publisher

Name of the publisher to perform this action

when

Perform action if the provided expression evaluates to true

onStart

Evaluate the expression at the start of an action

onComplete

Evaluate the expression upon completion of an action

Child Elements

None.

Examples

Example 173. Implicit Use of *open* Action

This is an example of the default implicit behavior of *open*. Note the absence of any declaration for *open*.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TestTemplate">
        <String value="Hello World!" />
    </DataModel>

    <StateModel name="State" initialState="Initial">
        <State name="Initial">

            <Action type="output">
                <DataModel ref="TestTemplate" />
            </Action>

        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="State"/>

        <Publisher class="File">
            <Param name="FileName" value="fuzzed.txt" />
        </Publisher>
    </Test>

</Peach>
<!-- end -->

```

Output from executing the previous pit.

```
> peach -1 --debug C:\temp\example.xml

[*] Test 'Default' starting with random seed 18872.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()                                     ①
Peach.Core.Publishers.FilePublisher output(12 bytes)
Peach.Core.Publishers.FilePublisher close()                                    ②
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.FilePublisher stop()

[*] Test 'Default' finished.
```

① Implicit *open* Action occurring at start of iteration

② Implicit *close* Action occurring prior to end of iteration

Example 174. Explicit Use of *open* Action

In this example, the *open* and *close* actions are used explicitly to perform two connections using the same publisher.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TestTemplate">
        <String name="Value" />
    </DataModel>

    <StateModel name="State" initialState="Initial">
        <State name="Initial">

            <Action type="output">
                <DataModel ref="TestTemplate" />
                <Data>
                    <Field name="Value" value="Connection #1\n" />
                </Data>
            </Action>

            <Action type="close" />

            <Action type="open" />

            <Action type="output">
                <DataModel ref="TestTemplate" />
                <Data>
                    <Field name="Value" value="Connection #2\n" />
                </Data>
            </Action>

        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="State"/>

        <Publisher class="Tcp">
            <Param name="Host" value="127.0.0.1" />
            <Param name="Port" value="31337" />
        </Publisher>
    </Test>

</Peach>
```

The netcat program can be used to simulate a listener. Netcat runs twice, once per connection.

```
> nc -l 31337 ; echo "====" ; nc -l 31337
Connection #1
=====
Connection #2
```

Output from this example.

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 61010.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpClientPublisher start()
Peach.Core.Publishers.TcpClientPublisher open() (1)
Peach.Core.Publishers.TcpClientPublisher output(14 bytes)
Peach.Core.Publishers.TcpClientPublisher

00000000  43 6F 6E 6E 65 63 74 69  6F 6E 20 23 31 0A          Connection #1.

Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.TcpClientPublisher close() (2)
Peach.Core.Publishers.TcpClientPublisher Shutting down connection to 127.0.0.1:31337
Peach.Core.Publishers.TcpClientPublisher Read 0 bytes from 127.0.0.1:31337, closing
client connection.
Peach.Core.Publishers.TcpClientPublisher Closing connection to 127.0.0.1:31337
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Open
Peach.Core.Publishers.TcpClientPublisher open() (3)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpClientPublisher output(14 bytes)
Peach.Core.Publishers.TcpClientPublisher

00000000  43 6F 6E 6E 65 63 74 69  6F 6E 20 23 32 0A          Connection #2.

Peach.Core.Publishers.TcpClientPublisher close() (4)
Peach.Core.Publishers.TcpClientPublisher Shutting down connection to 127.0.0.1:31337
Peach.Core.Publishers.TcpClientPublisher Read 0 bytes from 127.0.0.1:31337, closing
client connection.
Peach.Core.Publishers.TcpClientPublisher Closing connection to 127.0.0.1:31337
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.TcpClientPublisher stop()

[*] Test 'Default' finished.

```

① Implicit *open*

② Explicit *close*

③ Explicit *open*

④ Implicit *close*

close

The *close* Action causes the associated publisher to close its resource.

- For the [File Publisher](#), this action closes the file handle.
- For the [TCP Publisher](#), this action closes the network connection.

Close is an implicit action that occurs by default at the end of each iteration.

The only time you need to explicitly call the *close* action is when the default behavior needs to be changed. For example, during file fuzzing, the target application is launched after closing:

1. the *output* Action initiates writing the data
2. the *close* Action closes the target
3. the target application subsequently launches using the *call* Action

Default Order of Actions

The following is the default order in which Actions are performed when fuzzing:

1. start - Implicit, once per session
2. open - Implicit, once per iteration
3. Explicit actions (such as accept, input, and output)
4. close - Implicit, once per iteration
5. stop - Implicit, once per session

Syntax

```
<StateModel name="TheStateModel" initialState="InitialState">
  <State name="InitialState">

    <Action type="output">
      <DataModel ref="FileHeader"/>
    </Action>

    <Action type="close" />

    <Action type="call" method="LaunchTarget" publisher="Peach.Agent" />

  </State>
</StateModel>
```

Attributes

Required:

type

Action type, must be set to "close"

Optional:

name

Name used to identify the action

publisher

Name of the publisher that this action should be called on

when

Only perform action if the expression provided evaluates to true

onStart

Expression to run on start of an action.

onComplete

Expression to run on completion of an action

Child Elements

No child elements are supported by this element.

Examples

Example 175. Implicit Use of *close* Action

This is an example of the default implicit behavior of *close*.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TestTemplate">
        <String value="Hello World!" />
    </DataModel>

    <StateModel name="State" initialState="Initial">
        <State name="Initial">

            <Action type="output">
                <DataModel ref="TestTemplate" />
            </Action>

        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="State"/>

        <Publisher class="File">
            <Param name="FileName" value="fuzzed.txt" />
        </Publisher>
    </Test>

</Peach>
<!-- end -->

```

When run the following output is generated.

```

> peach -1 --debug C:\temp\example.xml

[*] Test 'Default' starting with random seed 18872.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()                                     ①
Peach.Core.Publishers.FilePublisher output(12 bytes)
Peach.Core.Publishers.FilePublisher close()                                    ②
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.FilePublisher stop()

[*] Test 'Default' finished.

```

① Implicit *open* Action occurring at start of iteration

② Implicit *close* Action occurring prior to end of iteration

Example 176. Explicit Use of *close* Action

This is an example of needed to call *close* in a different order than the default order. When file fuzzing, the file data is written using an *output*, then the file is closed using a *close* action. And finally the target is launched using a *call* action with the special Publisher name of *Peach.Agent*.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TestTemplate">
        <String value="Hello World!" />
    </DataModel>

    <StateModel name="State" initialState="Initial">
        <State name="Initial">

            <Action type="output">
                <DataModel ref="TestTemplate" />
            </Action>

            <!-- Close file -->
            <Action type="close" />

            <!-- Launch the file consumer -->
            <Action type="call" method="ScoobySnacks" publisher="Peach.Agent"/>

        </State>
    </StateModel>

    <Agent name="LocalAgent">
        <Monitor class="WindowsDebugger">
            <Param name="Executable" value="c:\windows\system32\notepad.exe" />
            <Param name="Arguments" value="fuzzfile.bin" />
            <Param name="StartOnCall" value="ScoobySnacks" />
        </Monitor>
        <Monitor class="PageHeap">
            <Param name="Executable" value="notepad.exe" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <Agent ref="LocalAgent" />
        <StateModel ref="State"/>

        <Publisher class="File">
            <Param name="FileName" value="fuzzfile.bin" />
        </Publisher>
    </Test>

</Peach>
<!-- end -->

```

When run the following output is generated.

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 14756.
Peach.Core.Agent.Agent StartMonitor: Monitor WindowsDebugger
Peach.Core.Agent.Agent StartMonitor: Monitor_1 PageHeap
Peach.Core.Agent.Agent SessionStarting: Monitor
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid SessionStarting
Peach.Core.Agent.Agent SessionStarting: Monitor_1

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()                                     ①
Peach.Core.Publishers.FilePublisher output(12 bytes)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Close                                         ②
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Call
Peach.Core.Agent.AgentManager Message: Action.Call => ScoobySnacks
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _StopDebugger
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid Cpu is idle, stopping process.
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _StopDebugger
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid DetectedFault()
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid DetectedFault() - No fault detected
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.FilePublisher stop()
Peach.Core.Agent.Agent SessionFinished: Monitor_1
Peach.Core.Agent.Agent SessionFinished: Monitor
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid SessionFinished
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _StopDebugger
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _FinishDebugger
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _StopDebugger
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _StopDebugger
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _FinishDebugger
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _StopDebugger

[*] Test 'Default' finished.
```

① Implicit *open* Action

② Explicit *close* Action occurring prior to call Action

accept

The *accept* Action blocks execution until an incoming connection is available, made, and accepted; or a timeout occurs. This action is most common in situations where a target application initiates the communication and Peach acts as a server.

Not all [Publishers](#) support the *accept* action type. [TcpListener](#) is an example of a Publisher that supports *accept*.

Peach is single-threaded so it can't execute any other task (including scripts) while waiting for a target connection.

Accept Timeout

Some Publishers use an *AcceptTimeout* parameter to limit the wait for a connection. If the timeout occurs prior to accepting an incoming connection, Peach behaves as follows:

Timeout occurs during a record iteration

Peach throws an exception and stops. The results of a record iteration are used as a standard for comparing with results of control iterations. When things do not work correctly during a record iteration, the environment is not working correctly or the pit is not correct. In either case, the fuzzing session cannot continue.

Timeout occurs during a control iteration

Peach triggers a fault. Control iterations are used as checkpoints to verify that the target is operating correctly. The pit must enable Control iterations to reach this state. During control iterations, Peach assumes any error results from the target entering an unwanted state. Such transitions are considered faults and logged accordingly.

Timeout occurs during a fuzzing iteration

Peach continues to the next iteration. During normal fuzzing iterations, Peach ignores odd behavior from the target unless it occurs during a control iteration.

Default Order of Actions

When fuzzing occurs, actions are performed in the following (default) order:

1. *start* - Implicit, once per session
2. *open* - Implicit, once per iteration
3. Explicit actions (like *accept*, *input*, and *output*)
4. *close* - Implicit, once per iteration
5. *stop* - Implicit, once per session

Syntax

```
<StateModel name="TheStateModel" initialState="InitialState">
  <State name="InitialState">
    <Action type="accept" />
    <Action type="output">
      <DataModel ref="PacketModel"/>
    </Action>
  </State>
</StateModel>
```

Attributes

Required:

type

Action type, must be set to "accept"

Optional:

name

Name used to identify the action

publisher

Name of the publisher to perform this action

when

Perform this action if the provided expression evaluates to true

onStart

Evaluate the expression at the start of an action

onComplete

Evaluate the expression upon completion of an action

Child Elements

None.

Examples

Example 177. Ping-Pong Accept Example

This is a simple example of using the *accept* action. Netcat (nc) is used in this example as the

client.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

  <DataModel name="Ping">
    <String value="PING" token="true"/>
  </DataModel>

  <DataModel name="Pong">
    <String value="PONG" />
  </DataModel>

  <StateModel name="TheStateModel" initialState="InitialState">
    <State name="InitialState">
      <Action type="accept" />

      <Action type="input">
        <DataModel ref="Ping"/>
      </Action>

      <Action type="output">
        <DataModel ref="Pong"/>
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheStateModel"/>

    <Publisher class="TcpListener">
      <Param name="Interface" value="0.0.0.0" />
      <Param name="Port" value="31337" />
      <Param name="AcceptTimeout" value="10000" />
      <Param name="Timeout" value="10000" />
    </Publisher>

    <Logger class="File" >
      <Param name="Path" value="logs"/>
    </Logger>
  </Test>
</Peach>
```

Output from this example that the server produces.

Once Peach starts, type the following command line and press RETURN to recreate the output. Again, nc is netcat.

```
nc -vv 127.0.0.1 31337
```

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 32331.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Accept
Peach.Core.Publishers.TcpListenerPublisher start()
Peach.Core.Publishers.TcpListenerPublisher open()
Peach.Core.Publishers.TcpListenerPublisher accept() ①
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Input
Peach.Core.Publishers.TcpListenerPublisher input()
Peach.Core.Publishers.TcpListenerPublisher Read 5 bytes from 127.0.0.1:62407
Peach.Core.Publishers.TcpListenerPublisher

00000000  50 49 4E 47 0A                      PING.

Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataCracker DataModel 'Ping' Bytes: 0/5, Bits: 0/40
Peach.Core.Cracker.getSize: -----> DataModel 'Ping'
Peach.Core.Cracker.scan: DataModel 'Ping'
Peach.Core.Cracker.scan: String 'Ping.DataElement_0' -> Pos: 0, Saving Token
Peach.Core.Cracker.scan: String 'Ping.DataElement_0' -> Pos: 32, Length: 32
Peach.Core.Cracker.getSize: <----- Deterministic: ???
Peach.Core.Cracker.Crack: DataModel 'Ping' Size: <null>, Bytes: 0/5, Bits: 0/40
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.String 'Ping.DataElement_0' Bytes: 0/5, Bits: 0/40
Peach.Core.Cracker.getSize: -----> String 'Ping.DataElement_0'
Peach.Core.Cracker.scan: String 'Ping.DataElement_0' -> Pos: 0, Saving Token
Peach.Core.Cracker.scan: String 'Ping.DataElement_0' -> Pos: 32, Length: 32
Peach.Core.Cracker.getSize: <----- Size: 32
Peach.Core.Cracker.Crack: String 'Ping.DataElement_0' Size: 32, Bytes: 0/5, Bits: 0/40
Peach.Core.Dom.DataElement String 'Ping.DataElement_0' value is: PING
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
```

```
Peach.Core.Dom.ActionType.Output
Peach.Core.Publishers.TcpListenerPublisher output(4 bytes)
Peach.Core.Publishers.TcpListenerPublisher

00000000  50 4F 4E 47                                PONG

Peach.Core.Publishers.TcpListenerPublisher close()
Peach.Core.Publishers.TcpListenerPublisher Shutting down connection to
127.0.0.1:62407
Peach.Core.Publishers.TcpListenerPublisher Read 0 bytes from 127.0.0.1:62407, closing
client connection.
Peach.Core.Publishers.TcpListenerPublisher Closing connection to 127.0.0.1:62407

Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.TcpListenerPublisher stop()

[*] Test 'Default' finished.
```

① Peach waits here for the incoming connection.

Interaction from the client, including the Netcat command line.

Once Netcat is running, type "PING" in upper case and press RETURN. "PONG" will be sent back by Peach.

```
> nc -vv 127.0.0.1 31337
Connection to 127.0.0.1 31337 port [tcp/*] succeeded!
PING
PONG
```

input

The *input* Action receives or reads in data using a [Publisher](#).

The most common use for *input* actions is to receive data that will be fuzzed. Once received and [cracked](#) into a data model, the data is available for other uses, such as:

- copying in [output](#) actions
- using in conditional expressions, such as with the [when](#) attribute.

input Actions are associated with a [DataModel](#) (provided as a child element) that drives the amount of requested data. For stream-based publishers that can read forever (such as TCP), the DataModel must prevent the potential of infinite data read operations, such as setting a timeout value.

Table 3. Stream Publisher versus Datagram Publisher

Stream	Datagram
TCP	UDP
Number of packets unimportant	Number of packets important
Implicit data length not included	Includes implicit data length

With a stream based publisher, the number of underlying packets received is not important; the data is accessed as a continual stream of incoming data. This does not apply to non-stream based publishers such as the datagram protocol [UDP](#). Non-stream based publishers need to know the number of underlying packets received. In UDP, each input and output action receives or sends a single datagram. When cracking the UDP data into a DataModel only the data in the single packet can be used. There is an implicit length to the data that is not present in [TCP](#).

Input Timeout

If the input time out is reached prior to all incoming data being received, Peach behaves as follows:

Timeout during a record iteration

When a record iteration generates errors, Peach assumes the environment is not working correctly or the pit is incorrect. In both cases, fuzzing cannot continue. Peach throws an exception and stops.

Timeout during a control iteration

Control iterations must be enabled to reach for this state. Control iterations assume that any error is the result of the target entering an unwanted state. Peach triggers a fault.

Timeout during a fuzzing iteration

Peach continues fuzzing with the next iteration. During normal fuzzing iterations, Peach assumes fuzzing causes odd behaviors from the target that are ignored, unless they occur during a control iteration.

Default Order of Actions

The following is the default order in which Actions are performed when fuzzing:

1. *start* - Implicit, once per session
2. *open* - Implicit, once per iteration
3. Explicit actions (such as *accept*, *input*, and *output*)
4. *close* - Implicit, once per iteration
5. *stop* - Implicit, once per session

Syntax

```
<StateModel name="TheStateModel" initialState="InitialState">
  <State name="InitialState">
    <Action type="input">
      <DataModel ref="InputModel" />
    </Action>
  </State>
</StateModel>
```

Attributes

Required:

type

Action type, must be set to "input"

Optional:

name

Name used to identify the action

publisher

Name of the publisher to perform this action

when

Perform action if the provided expression evaluates to true

onStart

Evaluate expression at the start of an action.

onComplete

Evaluate expression upon completion of an action

Child Elements

DataModel

Reference to a DataModel that will receive cracked input data

Examples

Example 178. Receiving Input from TCP Publisher

This is a simple example of using the *input* action to receive data. Netcat (nc) is used in this example as the client.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <!-- Marking the string as a token will imply a length of 4 characters -->
    <DataModel name="Ping">
        <String value="PING" token="true"/>
    </DataModel>

    <DataModel name="Pong">
        <String value="PONG" />
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState">
        <State name="InitialState">
            <Action type="accept" />

            <Action type="input">
                <DataModel ref="Ping"/>
            </Action>

            <Action type="output">
                <DataModel ref="Pong"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>

        <Publisher class="TcpListener">
            <Param name="Interface" value="0.0.0.0" />
            <Param name="Port" value="31337" />
            <Param name="AcceptTimeout" value="10000" />
            <Param name="Timeout" value="10000" />
        </Publisher>

        <Logger class="File" >
            <Param name="Path" value="logs"/>
        </Logger>
    </Test>
</Peach>

```

Output from this example. Once Peach is started, use the netcat command (nc) to recreate output. Type the following command line and press RETURN to continue.

```
nc -vv 127.0.0.1 31337
```

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 32331.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Accept
Peach.Core.Publishers.TcpListenerPublisher start()
Peach.Core.Publishers.TcpListenerPublisher open()
Peach.Core.Publishers.TcpListenerPublisher accept()
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Input
Peach.Core.Publishers.TcpListenerPublisher input() ①
Peach.Core.Publishers.TcpListenerPublisher Read 5 bytes from 127.0.0.1:62407
Peach.Core.Publishers.TcpListenerPublisher

00000000  50 49 4E 47 0A                      PING.

Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataModel 'Ping' Bytes: 0/5, Bits: 0/40 ②
Peach.Core.Cracker.getSize: -----> DataModel 'Ping'
Peach.Core.Cracker.scan: DataModel 'Ping'
Peach.Core.Cracker.scan: String 'Ping.DataElement_0' -> Pos: 0, Saving Token
Peach.Core.Cracker.scan: String 'Ping.DataElement_0' -> Pos: 32, Length: 32
Peach.Core.Cracker.getSize: <----- Deterministic: ???
Peach.Core.Cracker.Crack: DataModel 'Ping' Size: <null>, Bytes: 0/5, Bits: 0/40
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.String 'Ping.DataElement_0' Bytes: 0/5, Bits: 0/40
Peach.Core.Cracker.getSize: -----> String 'Ping.DataElement_0'
Peach.Core.Cracker.scan: String 'Ping.DataElement_0' -> Pos: 0, Saving Token
Peach.Core.Cracker.scan: String 'Ping.DataElement_0' -> Pos: 32, Length: 32
Peach.Core.Cracker.getSize: <----- Size: 32
Peach.Core.Cracker.Crack: String 'Ping.DataElement_0' Size: 32, Bytes: 0/5, Bits: 0/40
Peach.Core.Dom.DataElement String 'Ping.DataElement_0' value is: PING
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpListenerPublisher output(4 bytes)
```

```
Peach.Core.Publisher.TcpListenerPublisher
```

```
00000000 50 4F 4E 47
```

```
PONG
```

```
Peach.Core.Publisher.TcpListenerPublisher close()
Peach.Core.Publisher.TcpListenerPublisher Shutting down connection to
127.0.0.1:62407
Peach.Core.Publisher.TcpListenerPublisher Read 0 bytes from 127.0.0.1:62407, closing
client connection.
Peach.Core.Publisher.TcpListenerPublisher Closing connection to 127.0.0.1:62407

Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publisher.TcpListenerPublisher stop()

[*] Test 'Default' finished.
```

① Data received by TCP publisher

② Debugging output from the data cracker

Netcat command line. Once Netcat is running, type "PING" in upper case letters and press RETURN. Peach responds with "PONG".

```
> nc -vv 127.0.0.1 31337
Connection to 127.0.0.1 31337 port [tcp/*] succeeded!
PING
PONG
```

infrag

The *infrag* action is a special action to perform fragmented input. It must be used in conjunction with a data model that contains the [Frag](#) element as it's first and only element. The *infrag* action will utilize the fragmentation algorithm from the *Frag* element to identify when to stop receiving fragments. One or more [input](#) actions will be performed until the fragmentation algorithm is satisfied or a timeout occurs. Each *input* action that occurs will crack incoming data into a copy of the *Frag*'s *Template* element and added to the *Rendering* sequence element.

Once all of the fragments have been received they will be reconstructed and cracked into the *Frag* elements *Payload* element. After the *infrag* action has successfully been run values can be slurped from the *Payload* element or the *Rendering* sequence element.

See also: [Frag](#), [outfrag](#), [input](#).

Input Timeout

If the input time out is reached prior to all incoming data being received, Peach behaves as follows:

Timeout during a record iteration

When a record iteration generates errors, Peach assumes the environment is not working correctly or the pit is incorrect. In both cases, fuzzing cannot continue. Peach throws an exception and stops.

Timeout during a control iteration

Control iterations must be enabled to reach for this state. Control iterations assume that any error is the result of the target entering an unwanted state. Peach triggers a fault.

Timeout during a fuzzing iteration

Peach continues fuzzing with the next iteration. During normal fuzzing iterations, Peach assumes fuzzing causes odd behaviors from the target that are ignored, unless they occur during a control iteration.

Default Order of Actions

The following is the default order in which Actions are performed when fuzzing:

1. *start* - Implicit, once per session
2. *open* - Implicit, once per iteration
3. Explicit actions (such as *accept*, *input*, *infrag*, *output* and *outfrag*)
4. *close* - Implicit, once per iteration
5. *stop* - Implicit, once per session

Syntax

```
<StateModel name="TheStateModel" initialState="InitialState">
  <State name="InitialState">
    <Action type="infrag">
      <DataModel ref="ModelWithFragElement" />
    </Action>
  </State>
</StateModel>
```

Attributes

Required:

type

Action type, must be set to "infrag"

Optional:

name

Name used to identify the action

publisher

Name of the publisher to perform this action

when

Perform action if the provided expression evaluates to true

onStart

Evaluate expression at the start of an action.

onComplete

Evaluate expression upon completion of an action

Child Elements

DataModel

Reference to a DataModel that will receive cracked input data

Examples

Example 179. Receiving Input from TCP Publisher

A simple fragmented protocol in which each fragment has a single byte of payload data prefixed with the total length of data expected. The provided payload `example.txt` contains 4 bytes of data resulting in 4 fragments.

18. example.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

  <DataModel name="Fragmented">
    <Frag fragLength="1" totalLengthField="TotalLength">

      <Block name="Template">
        <String name="TotalLength" length="1"/>
        <Blob name="FragData" length="1" />
      </Block>

      <Block name="Payload">
        <String name="Value" />
      </Block>
    </Frag>
  </DataModel>

  <StateModel name="TheState" initialState="Initial">
    <State name="Initial">
      <Action type="infrag">
        <DataModel ref="Fragmented"/>
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheState"/>
    <Publisher class="Tcp">
      <Param name="Host" value="127.0.0.1"/>
      <Param name="Port" value="6666"/>
    </Publisher>
  </Test>
</Peach>
```

19. example.txt

```
4T4e4s4t
```

Prior to running Peach start a netcat listener using the following command line. This will provide the fragments to Peach.

```
nc -vv 127.0.0.1 6666 < example.txt
```

Run peach:

```
> peach -1 --debug example.xml

[*] Web site running at: http://10.0.1.87:8888/

[*] Test 'Default' starting with random seed 61424.
2016-07-07 16:59:13.0764 Peach.Pro.Core.Loggers.JobLogger Writing debug.log to:
C:\peach-pro\output\win_x64_debug\bin\Logs\example.xml_20160707165911\debug.log

[R1,-,-] Performing iteration
2016-07-07 16:59:13.2186 Peach.Core.Engine runTest: Performing control recording
iteration.
2016-07-07 16:59:13.2344 Peach.Pro.Core.Dom.Frag Generating fragments:
2016-07-07 16:59:13.2344 Peach.Core.Dom.StateModel Run(): Changing to state
"Initial".
2016-07-07 16:59:13.2554 Peach.Core.Dom.Action Run(Action): Infrag
2016-07-07 16:59:13.2554 Peach.Pro.Core.Publishers.TcpClientPublisher start()
2016-07-07 16:59:13.2554 Peach.Pro.Core.Publishers.TcpClientPublisher open()
2016-07-07 16:59:13.2684 Peach.Pro.Core.Publishers.TcpClientPublisher input()
2016-07-07 16:59:13.2684 Peach.Pro.Core.Publishers.TcpClientPublisher Read 8 bytes
from 127.0.0.1:6666
2016-07-07 16:59:13.2684 Peach.Pro.Core.Publishers.TcpClientPublisher

00000000 34 54 34 65 34 73 34 74 4T4e4s4t

2016-07-07 16:59:13.2684 Peach.Pro.Core.Publishers.TcpClientPublisher Read 0 bytes
from 127.0.0.1:6666, closing client connection.
2016-07-07 16:59:13.2684 Peach.Pro.Core.Publishers.TcpClientPublisher Closing
connection to 127.0.0.1:6666
2016-07-07 16:59:13.2854 DataCracker -+ Block 'Frag_1', Bytes: 0/8, Bits: 0/64
2016-07-07 16:59:13.2964 DataCracker | Size: ??? (Deterministic)
2016-07-07 16:59:13.2964 DataCracker |-- String 'TotalLength', Bytes: 0/8, Bits:
0/64
2016-07-07 16:59:13.2964 DataCracker | Size: 1 bytes | 8 bits (Has Length)
2016-07-07 16:59:13.2964 DataCracker | Value: 4
2016-07-07 16:59:13.3135 DataCracker |-- Blob 'FragData', Bytes: 1/8, Bits: 8/64
2016-07-07 16:59:13.3135 DataCracker | Size: 1 bytes | 8 bits (Has Length)
2016-07-07 16:59:13.3135 DataCracker | Value: 54
2016-07-07 16:59:13.3135 DataCracker /
2016-07-07 16:59:13.3135 Peach.Pro.Core.Dom.Actions.Infrag Fragment 1: pos: 2 length:
8 crack consumed: 2 bytes ①
2016-07-07 16:59:13.3135 Peach.Pro.Core.Publishers.TcpClientPublisher input()
2016-07-07 16:59:13.3135 DataCracker -+ Block 'Frag_2', Bytes: 2/8, Bits: 16/64
```

```

2016-07-07 16:59:13.3135 DataCracker | Size: ??? (Deterministic)
2016-07-07 16:59:13.3135 DataCracker |-- String 'TotalLength', Bytes: 2/8, Bits:
16/64
2016-07-07 16:59:13.3135 DataCracker | Size: 1 bytes | 8 bits (Has Length)
2016-07-07 16:59:13.3135 DataCracker | Value: 4
2016-07-07 16:59:13.3135 DataCracker |-- Blob 'FragData', Bytes: 3/8, Bits: 24/64
2016-07-07 16:59:13.3135 DataCracker | Size: 1 bytes | 8 bits (Has Length)
2016-07-07 16:59:13.3135 DataCracker | Value: 65
2016-07-07 16:59:13.3135 DataCracker /
2016-07-07 16:59:13.3135 Peach.Pro.Core.Dom.Actions.Infrag Fragment 2: pos: 4 length:
8 crack consumed: 2 bytes ②
2016-07-07 16:59:13.3135 Peach.Pro.Core.Publishers.TcpClientPublisher input()
2016-07-07 16:59:13.3135 DataCracker -+ Block 'Frag_3', Bytes: 4/8, Bits: 32/64
2016-07-07 16:59:13.3135 DataCracker | Size: ??? (Deterministic)
2016-07-07 16:59:13.3135 DataCracker |-- String 'TotalLength', Bytes: 4/8, Bits:
32/64
2016-07-07 16:59:13.3135 DataCracker | Size: 1 bytes | 8 bits (Has Length)
2016-07-07 16:59:13.3135 DataCracker | Value: 4
2016-07-07 16:59:13.3135 DataCracker |-- Blob 'FragData', Bytes: 5/8, Bits: 40/64
2016-07-07 16:59:13.3135 DataCracker | Size: 1 bytes | 8 bits (Has Length)
2016-07-07 16:59:13.3135 DataCracker | Value: 73
2016-07-07 16:59:13.3135 DataCracker /
2016-07-07 16:59:13.3135 Peach.Pro.Core.Dom.Actions.Infrag Fragment 3: pos: 6 length:
8 crack consumed: 2 bytes ③
2016-07-07 16:59:13.3135 Peach.Pro.Core.Publishers.TcpClientPublisher input()
2016-07-07 16:59:13.3135 DataCracker -+ Block 'Frag_4', Bytes: 6/8, Bits: 48/64
2016-07-07 16:59:13.3135 DataCracker | Size: ??? (Deterministic)
2016-07-07 16:59:13.3135 DataCracker |-- String 'TotalLength', Bytes: 6/8, Bits:
48/64
2016-07-07 16:59:13.3135 DataCracker | Size: 1 bytes | 8 bits (Has Length)
2016-07-07 16:59:13.3135 DataCracker | Value: 4
2016-07-07 16:59:13.3135 DataCracker |-- Blob 'FragData', Bytes: 7/8, Bits: 56/64
2016-07-07 16:59:13.3135 DataCracker | Size: 1 bytes | 8 bits (Has Length)
2016-07-07 16:59:13.3135 DataCracker | Value: 74
2016-07-07 16:59:13.3135 DataCracker /
2016-07-07 16:59:13.3135 Peach.Pro.Core.Dom.Actions.Infrag Fragment 4: pos: 8 length:
8 crack consumed: 2 bytes ④
2016-07-07 16:59:13.3265 Peach.Pro.Core.Dom.Actions.Infrag Reassembled fragment is 4
bytes ⑤
2016-07-07 16:59:13.3265 DataCracker -+ DataModel 'Fragmented', Bytes: 8/8, Bits:
64/64
2016-07-07 16:59:13.3265 DataCracker | Size: ??? (Deterministic)
2016-07-07 16:59:13.3265 DataCracker |-+ Frag 'DataElement_0', Bytes: 8/8, Bits:
64/64
2016-07-07 16:59:13.3265 DataCracker | | Size: ??? (Deterministic)
2016-07-07 16:59:13.3265 DataCracker | | Cracking Payload
2016-07-07 16:59:13.3265 DataCracker | |-+ Block 'Payload', Bytes: 0/4, Bits: 0/32
2016-07-07 16:59:13.3265 DataCracker | | | Size: ??? (Deterministic)

```

```
2016-07-07 16:59:13.3265 DataCracker | | |-- String 'Value', Bytes: 0/4, Bits: 0/32
2016-07-07 16:59:13.3265 DataCracker | | |   Size: 4 bytes | 32 bits (Last Unsized)
2016-07-07 16:59:13.3265 DataCracker | | |   Value: Test
2016-07-07 16:59:13.3265 DataCracker | | /
2016-07-07 16:59:13.3265 DataCracker | /
2016-07-07 16:59:13.3265 DataCracker /
2016-07-07 16:59:13.3265 Peach.Pro.Core.Dom.Actions.Infrag Final pos: 8 length: 8
crack consumed: 8 bytes
2016-07-07 16:59:13.3265 Peach.Pro.Core.Publishers.TcpClientPublisher close()
2016-07-07 16:59:13.3495 Peach.Core.Engine runTest: context.config.singleIteration ==
true
2016-07-07 16:59:13.3495 Peach.Pro.Core.Publishers.TcpClientPublisher stop()
2016-07-07 16:59:13.3495 Peach.Core.Engine EndTest: Stopping all agents and monitors

[*] Test 'Default' finished.
```

- ① First fragment cracked
- ② Second fragment cracked
- ③ Third fragment cracked
- ④ Forth and final fragment cracked
- ⑤ Data reassembled and starting crack into Payload element.

output

The *output* action writes or sends data using a [Publisher](#). *output* Actions are associated with a [DataModel](#) (specified as a child element) that produces the data to be sent.

Along with a DataModel, data sets can be provided using the [Data](#) element. Multiple Data elements can be provided that are switched between during fuzzing depending on the fuzzing strategy used. *Output* actions are fuzzed by Peach.

When output is called, the fuzzer implicitly calls *start* then *open*, if they have not yet been called. If a Pit defines multiple publishers, the *publisher* attribute specifies the publisher that performs the write/send request. The exact behavior of an *output* action is Publisher dependent. Review the documentation for a specific Publisher for details on its implementation actions. For example, calling output multiple times in succession on a stream publisher such as [TCP](#) appears in the logs as one output, whereas issuing multiple outputs in succession for the [UDP](#) publisher sends the data as separate UDP Packets and lists in the logs as separate calls.

Default Order of Actions

The following is the default order in which Actions are performed when fuzzing:

1. *start* - Implicit, once per session
2. *open* - Implicit, once per iteration
3. Explicit actions (such as *accept*, *input*, and *output*)
4. *close* - Implicit, once per iteration
5. *stop* - Implicit, once per session

Syntax

```
<StateModel name="TheStateModel" initialState="InitialState">
  <State name="InitialState">
    <Action type="output">
      <DataModel ref="SomeDataModel" />
    </Action>

    <Action type="output">
      <DataModel ref="SomeDataModel" />
      <Data name="SomeSampleData" fileName="sample.bin" />
    </Action>
  </State>
</StateModel>
```

Attributes

Required:

type

Action type, must be set to "output"

Optional:

name

Name used to identify the action

publisher

Name of the publisher to perform this action

when

Perform the action if the provided expression evaluates to true

onStart

Evaluate expression at the start of an action.

onComplete

Evaluate expression upon completion of an action

Child Elements

DataModel

Reference to a DataModel supplying data to fuzz

Data

Set of initial data to crack into the above DataModel before fuzzing

Examples

Example 180. File fuzzing with *output* action

Example of using the *output* action with multiple data sets to perform file fuzzing. This is a typical configuration for dumb file fuzzing.

Notice the special publisher *Peach.Agent*. This publisher sends the *call* action to all Agents, who in turn, forward the *call* to each associated Monitor. The method call is handled by the WindowsDebugger monitor, causing it to launch *notepad.exe*. For file fuzzing, note that the target is launched after creating the new fuzzed file.

This example runs on the Windows® Operating System, version XP or newer, with Windows Debugging Tools installed.

1. Save example Pit as "example.xml"
2. Run "Peach.exe --range 1,10 --debug example.xml"
3. You should see *mspaint.exe* open and close several times.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="DumbModel">
        <Blob />
    </DataModel>

    <StateModel name="State" initialState="Initial">
        <State name="Initial">

            <Action type="output">
                <DataModel ref="DumbModel" />

                <!-- provide a folder of files to fuzz -->
                <Data fileName="#{Peach.Pwd##\samples_png}" />
            </Action>

            <!-- Close file -->
            <Action type="close" />

            <!-- Launch the file consumer -->
            <Action type="call" method="ScoobySnacks" publisher="Peach.Agent"/>

        </State>
    </StateModel>

    <Agent name="LocalAgent">
        <Monitor class="WindowsDebugger">
            <Param name="Executable" value="c:\windows\system32\mspaint.exe" />
            <Param name="Arguments" value="fuzzed.png" />
            <Param name="StartOnCall" value="ScoobySnacks" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <Agent ref="LocalAgent" />
        <StateModel ref="State"/>

        <Publisher class="File">
            <Param name="FileName" value="fuzzed.png" />
        </Publisher>
    </Test>

</Peach>

```

Example 181. Sending Output with TCP Publisher

This example uses the *output* action to send data. Netcat (nc) is used in this example as the client.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <!-- Marking the string as a token will imply a length of 4 characters -->
    <DataModel name="Ping">
        <String value="PING" token="true"/>
    </DataModel>

    <DataModel name="Pong">
        <String value="PONG" />
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState">
        <State name="InitialState">
            <Action type="accept" />

            <Action type="input">
                <DataModel ref="Ping"/>
            </Action>

            <Action type="output">
                <DataModel ref="Pong"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>

        <Publisher class="TcpListener">
            <Param name="Interface" value="0.0.0.0" />
            <Param name="Port" value="31337" />
            <Param name="AcceptTimeout" value="10000" />
            <Param name="Timeout" value="10000" />
        </Publisher>

        <Logger class="File" >
            <Param name="Path" value="logs"/>
        </Logger>
    </Test>
</Peach>

```

Output from this example. Once Peach is started, use the netcat command line to recreate output. Type the following command line and press RETURN to start netcat.

```
nc -vv 127.0.0.1 31337
```

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 32331.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Accept
Peach.Core.Publishers.TcpListenerPublisher start()
Peach.Core.Publishers.TcpListenerPublisher open()
Peach.Core.Publishers.TcpListenerPublisher accept()
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Input
Peach.Core.Publishers.TcpListenerPublisher input() ①
Peach.Core.Publishers.TcpListenerPublisher Read 5 bytes from 127.0.0.1:62407
Peach.Core.Publishers.TcpListenerPublisher

00000000  50 49 4E 47 0A          PING.

Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.DataModel 'Ping' Bytes: 0/5, Bits: 0/40
Peach.Core.Cracker.getSize: -----> DataModel 'Ping'
Peach.Core.Cracker.scan: DataModel 'Ping'
Peach.Core.Cracker.scan: String 'Ping.DataElement_0' -> Pos: 0, Saving Token
Peach.Core.Cracker.scan: String 'Ping.DataElement_0' -> Pos: 32, Length: 32
Peach.Core.Cracker.getSize: <----- Deterministic: ???
Peach.Core.Cracker.Crack: DataModel 'Ping' Size: <null>, Bytes: 0/5, Bits: 0/40
Peach.Core.Cracker.DataCracker -----
Peach.Core.Cracker.String 'Ping.DataElement_0' Bytes: 0/5, Bits: 0/40
Peach.Core.Cracker.getSize: -----> String 'Ping.DataElement_0'
Peach.Core.Cracker.scan: String 'Ping.DataElement_0' -> Pos: 0, Saving Token
Peach.Core.Cracker.scan: String 'Ping.DataElement_0' -> Pos: 32, Length: 32
Peach.Core.Cracker.getSize: <----- Size: 32
Peach.Core.Cracker.Crack: String 'Ping.DataElement_0' Size: 32, Bytes: 0/5, Bits: 0/40
Peach.Core.Dom.DataElement String 'Ping.DataElement_0' value is: PING
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpListenerPublisher output(4 bytes)
Peach.Core.Publishers.TcpListenerPublisher
```

```

00000000  50 4F 4E 47                                PONG

Peach.Core.Publisher close()
Peach.Core.Publisher Shutting down connection to
127.0.0.1:62407
Peach.Core.Publisher Read 0 bytes from 127.0.0.1:62407, closing
client connection.
Peach.Core.Publisher Closing connection to 127.0.0.1:62407

Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publisher stop()

[*] Test 'Default' finished.

```

- ① Data received by TCP publisher
- ② Debugging output from the data cracker

Netcat command line. Once Netcat is running, type "PING" in upper case letters and press RETURN. Peach responds with "PONG".

```

> nc -vv 127.0.0.1 31337
Connection to 127.0.0.1 31337 port [tcp/*] succeeded!
PING
PONG

```

Example 182. Multiple Publishers with *output* Action

This example uses the *output* action with multiple publishers to write two files out on every iteration.

1. Save example Pit as "example.xml"
2. Run "Peach.exe -1 --debug example.xml"
3. You should see two files, fuzzed1.txt and fuzzed2.txt, created.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="File1Model">
        <String value="I'm file #1" />
    </DataModel>

    <DataModel name="File2Model">
        <String value="I'm file #2" />
    </DataModel>

    <StateModel name="State" initialState="Initial">
        <State name="Initial">

            <Action type="output" publisher="File1">
                <DataModel ref="File1Model" />
            </Action>

            <Action type="output" publisher="File2">
                <DataModel ref="File2Model" />
            </Action>

        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="State"/>

        <Publisher name="File1" class="File">
            <Param name="FileName" value="fuzzed1.txt" />
        </Publisher>

        <Publisher name="File2" class="File">
            <Param name="FileName" value="fuzzed2.txt" />
        </Publisher>
    </Test>

</Peach>

```

outfrag

The *outfrag* action is a special action to perform fragmented output. It must be used in conjunction with a data model that contains the [Frag](#) element as it's first and only element. The *outfrag* action will perform multiple [output](#) actions to send all the fragments in the *Rendering* sequence collection. These fragments are generated by the fragmentation algorithm associated with the *Frag* element.

See also: [Frag](#), [infrag](#), [output](#).

Default Order of Actions

The following is the default order in which Actions are performed when fuzzing:

1. *start* - Implicit, once per session
2. *open* - Implicit, once per iteration
3. Explicit actions (such as *accept*, *input*, *infrag*, *output* and *outfrag*)
4. *close* - Implicit, once per iteration
5. *stop* - Implicit, once per session

Syntax

```
<StateModel name="TheStateModel" initialState="InitialState">
  <State name="InitialState">

    <Action type="outfrag">
      <DataModel ref="ModelWithFragElement" />
    </Action>

  </State>
</StateModel>
```

Attributes

Required:

type

Action type, must be set to "outfrag"

Optional:

name

Name used to identify the action

publisher

Name of the publisher to perform this action

when

Perform action if the provided expression evaluates to true

onStart

Evaluate expression at the start of an action.

onComplete

Evaluate expression upon completion of an action

Child Elements

DataModel

Reference to a DataModel supplying data to fuzz

Data

Set of initial data to crack into the above DataModel before fuzzing

Examples

Example 183. Simple Example

Produce three fragments with each fragment containing the current fragment length, fragment sequence and total length of data. The Payload is 30 bytes of 0x41.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

    <DataModel name="Fragmented">
        <Frag fragLength="10"
              totalLengthField="TotalLength"
              fragmentLengthField="FragLength"
              fragmentIndexField="FragIndex">

            <Block name="Template">
                <Number name="FragLength" size="32"/>
                <Number name="FragIndex" size="32"/>
                <Number name="TotalLength" size="32"/>

                <Blob name="FragData" />
            </Block>

            <Block name="Payload">
                <Blob valueType="hex" value="
                    41 41 41 41 41 41 41 41 41 41
                    41 41 41 41 41 41 41 41 41 41
                    41 41 41 41 41 41 41 41 41 41"/>
            </Block>
        </Frag>
    </DataModel>

    <StateModel name="TheState" initialState="Initial">
        <State name="Initial">
            <Action type="outfrag">
                <DataModel ref="Fragmented"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>
        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>

```

The example pit will produce three fragments with 10 bytes of payload per-fragment.

Output from this example:

```

>peach -1 --debug BoolExample1.xml

[*] Web site running at: http://10.0.1.87:8888/

[*] Test 'Default' starting with random seed 7010.
2016-07-07 14:26:22.2979 Peach.Pro.Core.Loggers.JobLogger Writing debug.log to:
C:\peach-pro\output\win_x64_debug\bin\Logs\example.xml_20160707142621\debug.log

[R1,-,-] Performing iteration
2016-07-07 14:26:22.4288 Peach.Core.Engine runTest: Performing control recording
iteration.
2016-07-07 14:26:22.4690 Peach.Pro.Core.Dom.Frag Generating fragments:
2016-07-07 14:26:22.4870 Peach.Core.Dom.StateModel Run(): Changing to state
"Initial".
2016-07-07 14:26:22.4951 Peach.Core.Dom.Action Run(Action): Outfrag
2016-07-07 14:26:22.6139 Peach.Pro.Core.Publishers.ConsolePublisher start()
2016-07-07 14:26:22.6139 Peach.Pro.Core.Publishers.ConsolePublisher open()
2016-07-07 14:26:22.6188 Peach.Pro.Core.Publishers.ConsolePublisher output(22 bytes)
①
00000000 0A 00 00 00 01 00 00 00 1E 00 00 00 41 41 41 41 .....AAAA
00000010 41 41 41 41 41
2016-07-07 14:26:22.6188 Peach.Pro.Core.Publishers.ConsolePublisher output(22 bytes)
②
00000000 0A 00 00 00 02 00 00 00 1E 00 00 00 41 41 41 41 .....AAAA
00000010 41 41 41 41 41
2016-07-07 14:26:22.6188 Peach.Pro.Core.Publishers.ConsolePublisher output(22 bytes)
③
00000000 0A 00 00 00 03 00 00 00 1E 00 00 00 41 41 41 41 .....AAAA
00000010 41 41 41 41 41
2016-07-07 14:26:22.6188 Peach.Pro.Core.Publishers.ConsolePublisher close()
2016-07-07 14:26:22.6329 Peach.Core.Engine runTest: context.config.singleIteration ==
true
2016-07-07 14:26:22.6329 Peach.Pro.Core.Publishers.ConsolePublisher stop()
2016-07-07 14:26:22.6329 Peach.Core.Engine EndTest: Stopping all agents and monitors

[*] Test 'Default' finished.

```

① First fragment. Notice sequence number is 1.

② Second fragment. Notice sequence number is 2.

③ Third fragment. Notice sequence number is 3.

call

The *call* action provides a method-calling metaphor in Peach state models. The *call* action is useful when fuzzing targets interact with the method metaphor. In these instances, the interesting fuzzing data is typically in the calling parameters. Examples of interactions that use a method-call metaphor include Microsoft COM/DCOM, Web Services, and RPC.

Call actions support zero or more (user-defined and supplied) parameters and a resulting return value.

- If the method has no parameters, the *call* action performs the call.
- If the method has user-supplied parameters, the *call* action includes the parameters as items to fuzz.
- If the method has a return value, the *call* action can capture the return value and cracks the data into the provided *DataModel*.

As with the *output* action, a single data model can represent a single parameter along with zero or more *Data* sets. If the resulting return value is captured, Peach considers it input and cracks the return value in the provided *DataModel*.

Interacting with Agents and Monitors

Peach supports a specialized use of the *call* action to interact with [Agents](#) and [Monitors](#). A special Publisher called *Peach.Agent* sends the *call* action to all configured Agents. In turn, each Agent forwards the *call* action to its associated monitors. If a monitor supports interaction via the *call* action, documentation for that monitor will describe the interaction associated with *call*.

An instance of using the *call* action to interact with [Agents](#) and [Monitors](#) is provided in the examples section.



Pits in the Peach Pit Library have two pre-defined methods for the *call* action that some monitors use for iteration-based interaction:

- * *StartIterationEvent* performs actions at the start of each iteration
- * *ExitIterationEvent* performs actions at the end of each iteration

Param and the *call* action

xref:Param[Param] can be used as a `_call_` action argument. It is used for configuration settings and provides a key-value pair to the parent element.

Default Order of Actions

The following is the default order in which Actions are performed when fuzzing:

1. start - Implicit, once per session
2. open - Implicit, once per iteration
3. Explicit actions (such as accept, input, and output)
4. close - Implicit, once per iteration
5. stop - Implicit, once per session

Syntax

```
<StateModel name="TheStateModel" initialState="InitialState">
  <State name="InitialState">
    <Action type="call" method="openUrl">

      <Param name="p1">
        <DataModel ref="Param1DataModel" />
      </Param>

      <Param name="p2">
        <DataModel ref="Param2DataModel" />
        <Data name="p2data">
          <Field name="value" value="http://foo.com" />
        </Data>
      </Param>

      <Result>
        <DataModel ref="ResultDataModel" />
      </Result>

    </Action>
  </State>
</StateModel>
```

Attributes

Required:

type

Action type, must be set to "call"

Optional:

name

Name used to identify the action

method

String describing the method to execute

publisher

Name of the publisher that this action should be called on or Peach.Agent

when

Only perform action if the expression provided evaluates to true

onStart

Expression to run on start of an action.

onComplete

Expression to run on completion of an action

Child Elements

Param

Argument to be passed with the call. Zero or more Param elements can be provided. Param is considered an action that outputs data and as such, Peach fuzzes output data by default. Each parameter contains a single [DataModel](#) as a child element and zero or more [Data sets](#).

Result

Captures result of the call Action. Zero or one Result element can be provided. The result data is cracked into the specified [DataModel](#). Result elements are treated as input and can be used with [slurp](#) actions.

Examples

Example 184. Call action using Com Publisher

This example fuzzes the name of a video file, not the contents of the video file. Also, this example requires a QuickTime movie file to run.

1. Start QuickTime
2. Verify that a QuickTime movie with the filename "video.mov" exists in current folder.
3. Save example to "example.xml".
4. Run "Peach.Core.ComContainer.exe".
5. Run "Peach.exe -1 --debug example.xml".
6. You should see the QuickTime movie start to play.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <!-- Import python module so we can sleep after saying play -->
    <Import import="time"/>

    <DataModel name="TheDataModel">
        <String name="Value" />
    </DataModel>

    <StateModel name="TheState" initialState="Initial">

        <State name="Initial">

            <Action type="call" method="Players[1].OpenURL">
                <!-- This parameter will be fuzzed -->
                <Param name="P1">
                    <DataModel ref="TheDataModel" />

                    <Data>
                        <Field name="Value" value=
"https://archive.org/download/AppleComputersQuicktimeSample/sample.mp4"/>
                    </Data>
                </Param>
            </Action>

            <!-- The onComplete expression will pause the fuzzer to let
                 the video play for 6 seconds. -->
            <Action type="call" method="Players[1].QTControl.Movie.Play" onComplete=
"time.sleep(6)"/>

        </State>

    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="Com">
            <Param name="clsid" value="QuickTimePlayerLib.QuickTimePlayerApp"/>
        </Publisher>
    </Test>

</Peach>

```

Example 185. Interacting with Agents and Monitors

This example controls when the WindowsDebugger monitor launches a target executable (*notepad.exe*) under a debugger. This configuration is common with file fuzzing.

Note the special *Peach.Agent* publisher name. This Publisher causes the *call* action to be sent to all Agents. In turn, each Agent forwards the *call* action to its associated Monitors. The method call will be handled by the WindowsDebugger monitor, causing it to launch *notepad.exe*. For file fuzzing, ensure the target is launched **after** writing out the new fuzzed file.

This example requires a machine running the Windows® Operating system, version XP or newer, with the Windows Debugging Tools installed.

1. Save the example Pit as "example.xml"
2. Run "Peach.exe --range 1,10 --debug example.xml"
3. You should see *notepad.exe* open and close several times.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TestTemplate">
        <String value="Hello World!" />
    </DataModel>

    <StateModel name="State" initialState="Initial">
        <State name="Initial">

            <Action type="output">
                <DataModel ref="TestTemplate" />
            </Action>

            <!-- Close file -->
            <Action type="close" />

            <!-- Launch the file consumer -->
            <Action type="call" method="ScoobySnacks" publisher="Peach.Agent"/>

        </State>
    </StateModel>

    <Agent name="LocalAgent">
        <Monitor class="WindowsDebugger">
            <Param name="Executable" value="c:\windows\system32\notepad.exe" />
            <Param name="Arguments" value="fuzzfile.bin" />
            <Param name="StartOnCall" value="ScoobySnacks" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <Agent ref="LocalAgent" />
        <StateModel ref="State"/>

        <Publisher class="File">
            <Param name="FileName" value="fuzzfile.bin" />
        </Publisher>
    </Test>

</Peach>

```

getProperty

The *getProperty* Action retrieves the value of a property from an object that uses the property metaphor and stores the value in a DataModel. *getProperty* Actions specify in a child element the [DataModel](#) that receives the retrieved data.

getProperty provides one half of a property metaphor in Peach state models. *getProperty* actions are considered input actions. [setProperty](#) is the other half (output) of the property metaphor.

The *getProperty* Action has two main uses:

- when fuzzing targets such as Microsoft COM/DCOM objects that use the property metaphor. The *getProperty* Action instructs a Publisher to retrieve the value of a specified property.
- when accessing properties in a Publisher that supports the property metaphor. Some Publishers support using properties to get or set values, such as the UDP, RawV4, RawIPv4, and RawV6 Publishers). An instance of this type of Publisher is in the examples section.



Properties exposed by publishers are described in the publisher documentation.

Default Order of Actions

The following is the default order in which Actions are performed when fuzzing:

1. *start* - Implicit, once per session
2. *open* - Implicit, once per iteration
3. Explicit actions (*accept*, *input*, *output*, etc.)
4. *close* - Implicit, once per iteration
5. *stop* - Implicit, once per session

Syntax

```
<StateModel name="TheStateModel" initialState="InitialState">
  <State name="InitialState">
    <Action type="getProperty" property="Name">
      <DataModel ref="NameModel"/>
    </Action>
  </State>
</StateModel>
```

Attributes

Required:

type

Action type, must be set to "getProperty"

Optional:

name

Name used to identify the action

property

Name of the property to retrieve

publisher

Name of the publisher to perform this action, or the Name publisher with the property to retrieve

when

Perform this action if the provided expression evaluates to true

onStart

Evaluate expression at the start of an action

onComplete

Evaluate expression upon completion of an action

Child Elements**DataModel**

Reference to a DataModel store the retrieved data

Examples**Example 186. Accessing a Property using Microsoft COM Publisher**

This example shows interacting with the QuickTime COM object.

getProperty retrieves the height of the player window. Since *getProperty* is considered a data input, it is not fuzzed. To fuzz the height property, *setProperty* would be used.

This example requires a Windows XP or newer machine with Apple QuickTime installed.

1. Start QuickTime
2. Save example to "example.xml"
3. Run "Peach.Core.ComContainer.exe"
4. Run "Peach.exe -1 --debug example.xml"
5. You should see the QuickTime movie start to play

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <!-- Import python module so we can sleep after saying play -->
    <Import import="time"/>

    <DataModel name="TheDataModel">
        <String name="Value" />
    </DataModel>

    <StateModel name="TheState" initialState="Initial">

        <State name="Initial">

            <Action type="call" method="Players[1].OpenURL">
                <!-- This parameter will be fuzzed -->
                <Param name="P1">
                    <DataModel ref="TheDataModel" />

                    <Data>
                        <Field name="Value" value=
"https://archive.org/download/AppleComputersQuicktimeSample/sample.mp4"/>
                    </Data>
                </Param>
            </Action>

            <Action type="getProperty" property="Players[1].Height">
                <DataModel ref="TheDataModel" />
            </Action>

            <!-- The onComplete expression will pause the fuzzer to let
                 the video play for 6 seconds. -->
            <Action type="call" method="Players[1].QTControl.Movie.Play" onComplete=
"time.sleep(6)"/>

        </State>

    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>

        <Publisher class="Com">
            <Param name="clsid" value="QuickTimePlayerLib.QuickTimePlayerApp"/>
        </Publisher>
    </Test>

```

```
</Peach>
```

Example 187. Accessing Publisher Properties using *getProperty* Action

This example retrieves a property from the [UDP Publisher](#).

This example uses netcat as the client connecting to Peach.

1. Save the example Pit to "example.xml"
2. Run "peach -1 --debug example.xml"
3. Run "echo -n "WHATSMYIP" | nc -4u -w1 localhost 1234"

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="Ping">
        <String name="PingStr" value="WHATSMYIP" token="true"/>
    </DataModel>

    <DataModel name="IpAddress">
        <Number size="32" name="IP" value="9999" />
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState">
        <State name="InitialState">
            <Action type="open" />

            <Action type="input">
                <DataModel ref="Ping"/>
            </Action>

            <Action name="GetIpAddress" type="getProperty" property="LastRecvAddr">
                <DataModel ref="IpAddress" />
            </Action>

            <Action type="slurp" valueXpath="//GetIpAddress//IP" setXpath=
//IpResponse//IP" />

            <Action name="IpResponse" type="output">
                <DataModel ref="IpAddress"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>
        <Publisher class="Udp">
            <Param name="Host" value="127.0.0.1" />
            <Param name="SrcPort" value="1234" />
            <Param name="Timeout" value="10000" />
        </Publisher>

        <Logger class="File" >
            <Param name="Path" value="logs"/>
        </Logger>
    </Test>
</Peach>

```

setProperty

The *setProperty* Action allows Peach to modify values of properties in the target or to modify properties of a Publisher while fuzzing. *setProperty* Actions specify the [DataModel](#) property in the target that receives fuzzing.

setProperty provides one half of a property metaphor in Peach state models. *setProperty* actions are considered output actions. [*getProperty*](#) is the other half (input) of the property metaphor.

Properties that are modified receive fuzzing in the same way an output action would be. For example, a test target, such as a COM object, can have a property set and mutated by Peach, as in the following example.

The second use, setting and mutating a property in a publisher, is fuzzing the MTU attribute of the [RawEther](#) Publisher. Some Publishers support the use of properties.



If you do not want to fuzz a property that is set using *setProperty*, set the mutable attribute of the data item to false. The item declaration is located in the DataModel.

Default Order of Actions

The following is the default order in which Actions are performed when fuzzing:

1. *start* - Implicit, once per session
2. *open* - Implicit, once per iteration
3. Explicit actions (*accept*, *input*, *output*, etc.)
4. *close* - Implicit, once per iteration
5. *stop* - Implicit, once per session

Syntax

```
<StateModel name="TheStateModel" initialState="InitialState">
    <State name="InitialState">
        <Action type="setProperty" property="Name">
            <DataModel ref="NameModel"/>
        </Action>
    </State>
</StateModel>
```

Attributes

Required:

type

Action type, must be set to "setProperty"

Optional:

name

Name used to identify the action

property

Name of property to set in the fuzzing target, or in the publisher

publisher

Name of the publisher to perform this action, or the Name publisher with the property to modify

when

Perform action if the provided expression evaluates to true

onStart

Evaluate expression at the start of an action

onComplete

Evaluate expression upon completion of an action

Child Elements**DataModel**

Reference to a DataModel containing data to fuzz

Data

Set of initial data crack into the above DataModel before fuzzing

Examples**Example 188. Accessing a Property using Microsoft COM Publisher**

This example fuzzes property values of a QuickTime COM object. *setProperty* sets the height of the player window. Since *setProperty* is considered data output, it is fuzzed.

This example requires a machine with the Windows Operating System, version XP or newer, with Apple QuickTime installed.

1. Start QuickTime
2. Save example to "example.xml"
3. Run "Peach.Core.ComContainer.exe"
4. Run "Peach.exe -1 --debug example.xml"

5. You should see the QuickTime movie start to play

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

  <!-- Import python module so we can sleep after saying play -->
  <Import import="time"/>

  <DataModel name="TheDataModel">
    <String name="Value" />
  </DataModel>

  <StateModel name="TheState" initialState="Initial">

    <State name="Initial">

      <Action type="call" method="Players[1].OpenURL">
        <!-- This parameter will be fuzzed -->
        <Param name="P1">
          <DataModel ref="TheDataModel" />

          <Data>
            <Field name="Value" value=
"https://archive.org/download/AppleComputersQuicktimeSample/sample.mp4"/>
          </Data>
        </Param>
      </Action>

      <!-- This property will be fuzzed -->
      <Action type="setProperty" property="Players[1].Height">
        <DataModel ref="TheDataModel" />
        <Data>
          <Field name="Value" value="100" />
        </Data>
      </Action>

      <!-- The onComplete expression will pause the fuzzer to let
          the video play for 6 seconds. -->
      <Action type="call" method="Players[1].QTControl.Movie.Play" onComplete=
"time.sleep(6)"/>

    </State>

  </StateModel>
```

```
<Test name="Default">
  <StateModel ref="TheState"/>

  <Publisher class="Com">
    <Param name="clsid" value="QuickTimePlayerLib.QuickTimePlayerApp"/>
  </Publisher>
</Test>

</Peach>
```

web

Model web api/http requests. Provides full control over the request being modeled. Must be used in conjunction with the [WebApi](#) publisher.

This action gives full control over:

- Path
- Query string
- Headers
- Form data
- Multipart requests
- Body
 - Binary
 - Json
 - Text
 - XML

Several analyzers are useful when building Pits using this action type:

Json

Converts JSON documents or strings into Peach data models. Can be used both inside of DataModels with the String element or also via the command line.

Postman

Converts Postman Catalogs to Peach Pits.

Swagger

Converts Swagger JSON to Peach Pits

WebRecordProxy

Recording proxy captures web requests and generates full pit. This makes creating the base pit easy, simply use as your HTTP proxy.

Xml

Converts XML documents or string into Peach data models. Can be used both inside of DataModels with the String element or also via the command line.



SSL/TLS is supported, just use <https> as the protocol in the URL.



Must be used with [WebApi](#) publisher.

Syntax

```
<StateModel name="TheStateModel" initialState="InitialState">
  <State name="InitialState">
    <Action type="web" method="GET" url="http://localhost/product/{id}">
      <Path key="id" value="100" />
      <Query key="first" value="Mike" />
      <Query key="last" value="Smith" />
      <Header key="Content-Type" value="application/json"/>
      <Body name="Json">
        <DataModel ref="JsonBody" />
      </Body>
      <Response/>
    </Action>
  </State>
</StateModel>
```

Attributes

Required:

type

Action type, must be set to "web"

method

HTTP method verb (GET, DELETE, POST, PUT, etc.)

url

Fully qualified URL including path. Excludes query string.

URL can include path substitution identifiers in the form `{id}`, where `id` matches the `id` attribute of a child **Path** element (see examples below). Multiple path substitutions can be performed.

Optional:

name

Name used to identify the action

publisher

Name of the publisher that this action should be called on or Peach.Agent

when

Only perform action if the expression provided evaluates to true

onStart

Expression to run on start of an action.

onComplete

Expression to run on completion of an action

Child Elements

Path

Each substitution identifier defined in the `url` attribute of the web action must have a corresponding Path child element. This provides a default value and data model for the path variable.

Query

Defines a querystring key/value pair. The query portion of the URL will be built using these child elements.

Header

Define an HTTP header.

FormData

Define a key/value pair of form data. These values are transmitted via the request body. It is not possible to combine FormData with a Body or Part child.

Body

Define the request body. Only one Body child element is allowed. Body elements cannot be mixed with FormData or Part elements. Only one type of body is allowed.

Part

Define a part of a multipart request. Each part can contain `Header`, `FormData` and `Body` child elements. Part elements cannot be used with FormData or Body elements.

Response

Capture the response with an optional custom data model. Exposes status code, headers and body to scripting and *slurp* actions. If omitted, a default Response element is generated to capture the request response.

Examples

Example 189. Calling WebApi Services with Result

The following example provides three fragments using the GET and POST methods. For the GET request, the Result element is used to capture any returned data.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">
```

```

<DataModel name="postData">
    <JsonObject>
        <JsonString propertyName="Name" value="Widget" />
        <JsonDouble propertyName="Price" value="1.99" />
        <JsonInteger propertyName="Quantify" value="1" />
    </JsonObject>
</DataModel>

<DataModel name="WebApiResult">
    <Choice name="ResultOrEmpty">
        <String name="Result">
            <Analyzer class="Json" />
        </String>
        <Block name="Empty" />
    </Choice>
</DataModel>

<StateModel name="Default" initialState="FirstState">
    <State name="FirstState">
        <Action type="web" method="GET" url="http://www.example.com/product/{id}">
            <Path name="Id" key="id" value="1"/>

            <Response />
            <DataModel ref="WebApiResult" />
        </Response>
    </Action>

        <Action type="web" method="GET" url="http://www.example.com/invoices">
            <Query name="StartDate" key="start_date" value="11-21-2011" />
            <Query name="EndDate" key="end_date" value="11-21-2015" />

            <Response>
                <DataModel ref="WebApiResult" />
            </Response>
        </Action>

        <Action type="call" method="POST" url="http://www.example.com/product/{id}">
            <Path name="Id" key="id" value="100" />
            <Body name="postData">
                <DataModel ref="postData" />
            </Body>
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="Default" />
    <Publisher class="WebApi" />

```

</Test>

</Peach>

Example 190. Posting XML

The following example provides three fragments using the GET and POST methods.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="PostData">
    <XmlElement elementName="Product">
        <XmlAttribute attributeName="Name">
            <String value="Widget" />
        </XmlAttribute>
        <XmlAttribute attributeName="Price">
            <String value="1.99" />
        </XmlAttribute>
        <XmlAttribute attributeName="Quantity">
            <String value="1" />
        </XmlAttribute>
    </XmlElement>
</DataModel>

<StateModel name="Default" initialState="FirstState">
    <State name="FirstState">

        <Action type="web" method="POST" url="http://www.example.com/product/{id}">
            <Path key="id" value="1"/>

            <Body name="postData">
                <DataModel ref="PostData" />
            </Body>
        </Action>

    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="Default" />
    <Publisher class="WebApi" />
</Test>

</Peach>
```

Example 191. Posting Binary

The following example provides three fragments using the GET and POST methods.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="PostData">
    <Blob />
</DataModel>

<StateModel name="Default" initialState="FirstState">
    <State name="FirstState">

        <Action type="web" method="POST" url=
"http://www.example.com/product/{id}/image">
            <Path key="id" value="1"/>
            <Body name="postData">
                <DataModel ref="PostData" />
                <Data fileName="image.png" />
            </Body>
        </Action>

    </State>
</StateModel>

<Test name="Default">

    <StateModel ref="Default"/>
    <Publisher class="WebApi"/>

</Test>
</Peach>
```

Example 192. Setting Custom Header via Pit

The following example shows how to set a custom header via the Pit XML. The custom header is named "X-CustomHeader" with a value of "Hello World!".

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<StateModel name="Default" initialState="FirstState">
    <State name="FirstState">

        <Action type="web" method="GET" url="http://www.example.com/product/{id}">
            <Path key="id" value="1"/>
            <Header name="x-custom" key="X-CustomHeader" value="Hello World!" />
            <Response/>
        </Action>

    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="Default" />
    <Publisher class="WebApi" />
</Test>

</Peach>
```

Example 193. Setting Custom Authentication Header via Python

The following example shows how to add custom authentication via a python script. In this example we will configure a pit for fuzzing an Amazon AWS S3 service endpoint. This is only an example and should not actually be used to fuzz AWS.

```

import base64
import hmac
from hashlib import sha1
from email.Utils import formatdate

AWS_ACCESS_KEY_ID = "44CF9590006BF252F707"
AWS_SECRET_KEY = "0txrzxIsfpFjA7SwPzILwy8Bw21TLhquhboDYROV"

def AwsAuthGen(context, action):

    # Get the Publisher (WebApiPublisher)
    if action.publisher:
        publisher = context.test.publishers[action.publisher]
    else:
        publisher = context.test.publishers[0]

    XAmzDate = formatdate()

    h = hmac.new(AWS_SECRET_KEY, "PUT\n\napplication/json\n\nnx-amz-date:%s\n\n/?policy"
    % XAmzDate, sha1)
    authToken = base64.encodestring(h.digest()).strip()

    publisher.Headers.Add("x-amz-date", XAmzDate)
    publisher.Headers.Add("Authorization", "AWS %s:%s" % (AWS_ACCESS_KEY_ID,
    authToken))

# end

```

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<Import import="aws_s3_example"/>

<!--
{
"Version": "2008-10-17",
"Id": "aaaa-bbbb-cccc-dddd",
"Statement" : [
    {
        "Effect": "Allow",
        "Sid": "1",
        "Principal" : {
            "AWS": ["111122223333", "444455556666"]
        },

```

```

        "Action":["s3:*"],
        "Resource":"arn:aws:s3:::bucket/*"
    }
]
}
-->
<!-- Generated using the JSON analyzer -->
<DataModel name="Policy">
    <JsonObject>
        <JsonString propertyName="Version" name="Version" value="2008-10-17" />
        <JsonString propertyName="Id" name="Id" value="aaaa-bbbb-cccc-dddd" />
        <JsonArray propertyName="Statement" name="Statement">
            <JsonObject propertyName="Statement" name="Statement">
                <JsonString propertyName="Effect" name="Effect" value="Allow" />
                <JsonString propertyName="Sid" name="Sid" value="1" />
                <JsonObject propertyName="Principal" name="Principal">
                    <JsonArray propertyName="AWS" name="AWS">
                        <JsonString propertyName="AWS" name="AWS" value="111122223333" />
                        <JsonString value="444455556666" />
                    </JsonArray>
                </JsonObject>
            </JsonArray>
            <JsonArray propertyName="Action" name="Action">
                <JsonString propertyName="Action" name="Action" value="s3:*" />
            </JsonArray>
            <JsonString propertyName="Resource" name="Resource" value=
"arn:aws:s3:::bucket/*" />
        </JsonObject>
    </JsonArray>
</JsonObject>
</DataModel>

<StateModel name="TheStateModel" initialState="Initial">
    <State name="Initial">
        <Action type="web" method="PUT" url="http://XXXXX.s3.amazonaws.com/?policy"
               onStart="aws_s3_example.AwsAuthGen(context, action)">
            <Body name="Body">
                <DataModel ref="Policy" />
            </Body>
        </Action>
    </State>
</StateModel>

<Test name="Default" maxOutputSize="20000000">
    <StateModel ref="TheStateModel"/>
    <Publisher class="WebApi">

```

```
<Param name="FaultOnStatusCodes" value="500,501,502,503,504,505" />
</Publisher>
</Test>
</Peach>
```

slurp

Slurp moves data between two DataModels. The Slurp Action copies a data value from one non-container element in one DataModel to another non-container element in a separate DataModel.

A non-container element is an element that does not contain other elements. Number, string, and blob are types of non-container elements. In contrast, array, [Block](#), [Choice](#), and [DataModel](#)) are types of container elements.

- The [valueXpath](#) attribute defines the input element. The data model and the element in valueXpath were previously referenced by a prior input action.
- The [setXpath](#) attribute defines the output element. The data model and the element in setXpath will be referenced by a future action.

The XML Path (XPath) language is used by valueXpath and setXpath to specify the data to move. XPath targets a specific element for the [valueXpath](#) and [setXpath](#) parameters. This allows a variety of ways to access these elements in the Document Object Model (DOM).

A common use for slurp is during a protocol sequence where a sequence id or a challenge id needs to round-trip from the server to the client and back to the server (see the Challenge Response example below). The *slurp* Action copies the data received from the server (valueXpath attribute) into the response packet (setXpath attribute). When the response packet is populated, it is sent back to the server using another Action.

Potential Slurp errors include the following:

- Copying from a container element or copying to a container element (Copying is restricted to simple types, such as number, string, or blob.)
- The valueXpath parameter does not return a single element (One element is required; no more; no less. If you need to move more than one set of data, issue multiple slurp commands.)
- The valueXpath and setXpath parameters return the same element
- The setXpath parameter does not return any elements (The element has no content or the path is non-existent.)

Assigning unique names to elements simplifies targeting those elements with the valueXpath and setXpath attributes. Using XPath wildcards with unique element names allows an element to be targeted using its name instead of using the full path to the element. See the following examples for slurp actions with and without unique element names.

XPath Further Reading

The following links provide additional information regarding XPath.

- [XPath Tutorial](#)
- [XPath Syntax](#)
- [XPath Specification](#)

Syntax

```
<DataModel name="ReceiveChallenge">
  <String name="Challenge" />
</DataModel>

<DataModel name="SendChallenge">
  <String name="Challenge" />
</DataModel>

<StateModel name="TheStateModel" initialState="InitialState">
  <State name="InitialState">
    <Action name="ReceiveChallenge" type="input">
      <DataModel name="TheReceiveChallenge" ref="ReceiveChallenge"/>
    </Action>

    <Action type="slurp" valueXpath="//TheReceiveChallenge/Challenge" setXpath=
"//TheSendChallenge/Challenge" />

    <Action name="SendChallenge" type="output">
      <DataModel name="TheSendChallenge" ref="SendChallenge"/>
    </Action>
  </State>
</StateModel>
```

Attributes

Required:

type

Action type, must be set to "slurp"

Optional:

name

Name used to identify the action

valueXpath

Path to the source element. Must evaluate to match a single element.

setXpath

Path to the destination element. Can match multiple elements on which to set the value.

when

Perform action if the provided expression evaluates to true

onStart

Evaluate expression to run at the start of an action

onComplete

Evaluate expression upon completion of an action

Child Elements

None.

Examples

Example 194. Non-unique Names

This example uses fully specified names in the xpath query to identify elements.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="Ping">
        <String name="PingStr" value="Stop Copying Me" token="true"/>
    </DataModel>

    <DataModel name="Pong">
        <String name="Resp" value="I am not" />
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState">
        <State name="InitialState">
            <Action name="PingPacket" type="input">
                <DataModel ref="Ping"/>
            </Action>

            <Action type="slurp" valueXpath="//PingPacket/Ping/PingStr" setXpath=
"//PongPacket/CopyResponse/Resp" />

            <Action name="PongPacket" type="output">
                <DataModel name="CopyResponse" ref="Pong"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>
        <Publisher class="Udp">
            <Param name="Host" value="127.0.0.1" />
            <Param name="SrcPort" value="1234" />
            <Param name="Timeout" value="10000" />
        </Publisher>

        <!-- Test with following command -->
        <!-- echo -n "Stop Copying Me" | nc -4u -w1 localhost 1234 -->

        <Strategy class="Random"/>

        <Logger class="File" >
            <Param name="Path" value="logs"/>
        </Logger>
    </Test>
</Peach>
```

Example 195. Unique Names

This example uses unique names for the source element (input) and for the destination element (output) involved in the slurp. Having unique names allows the simpler xpath notation that uses wildcards (//) instead of using a full name.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="Ping">
        <String name="PingStr" value="Stop Copying Me" token="true"/>
    </DataModel>

    <DataModel name="Pong">
        <String name="Resp" value="I am not" />
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState">
        <State name="InitialState">
            <Action name="PingPacket" type="input">
                <DataModel ref="Ping"/>
            </Action>

            <Action type="slurp" valueXpath="//PingStr" setXpath("//Resp") />

            <Action name="PongPacket" type="output">
                <DataModel name="CopyResponse" ref="Pong"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>
        <Publisher class="Udp">
            <Param name="Host" value="127.0.0.1" />
            <Param name="SrcPort" value="1234" />
            <Param name="Timeout" value="10000" />
        </Publisher>

        <!-- Test with following command -->
        <!-- echo -n "Stop Copying Me" | nc -4u -w1 localhost 1234 -->

        <Strategy class="Random"/>

        <Logger class="File" >
            <Param name="Path" value="logs"/>
        </Logger>
    </Test>
</Peach>

```

Example 196. Setting Multiple Elements

This example copies a value to multiple elements at once.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

  <DataModel name="Ping">
    <String name="PingStr" value="Stop Copying Me" token="true"/>
  </DataModel>

  <DataModel name="Pong">
    <String name="Resp" value="I am not" />
  </DataModel>

  <StateModel name="TheStateModel" initialState="InitialState">
    <State name="InitialState">
      <Action type="input">
        <DataModel ref="Ping"/>
      </Action>

      <!-- Will copy value from our input action to all of our output actions. -->
      <Action type="slurp" valueXpath="//PingStr" setXpath("//Resp" />

      <Action type="output">
        <DataModel ref="Pong"/>
      </Action>

      <Action type="output">
        <DataModel ref="Pong"/>
      </Action>

      <Action type="output">
        <DataModel ref="Pong"/>
      </Action>

      <Action type="output">
        <DataModel ref="Pong"/>
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheStateModel"/>
    <Publisher class="Udp">
```

```
<Param name="Host" value="127.0.0.1" />
<Param name="SrcPort" value="1234" />
<Param name="Timeout" value="10000" />
</Publisher>

<!-- Test with following command -->
<!-- echo -n "Stop Copying Me" | nc -4u -w1 localhost 1234 -->

<Strategy class="Random"/>

<Logger class="File" >
  <Param name="Path" value="logs"/>
</Logger>
</Test>
</Peach>
```

Example 197. Challenge Response

This example shows how to use slurp for round-tripping a challenge id just issued by a server in the next response packet to the server.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="ReceiveChallenge">
        <String name="Challenge" />
    </DataModel>

    <DataModel name="SendChallenge">
        <String name="Challenge" />
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState">
        <State name="InitialState">
            <Action name="ReceiveChallenge" type="input">
                <DataModel name="TheReceiveChallenge" ref="ReceiveChallenge"/>
            </Action>

            <Action type="slurp" valueXpath="//TheReceiveChallenge/Challenge" setXpath=
"//TheSendChallenge/Challenge" />

            <Action name="SendChallenge" type="output">
                <DataModel name="TheSendChallenge" ref="SendChallenge"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheStateModel"/>
        <Publisher class="Udp">
            <Param name="Host" value="127.0.0.1" />
            <Param name="SrcPort" value="1234" />
            <Param name="Timeout" value="10000" />
        </Publisher>

        <!-- Test with following command -->
        <!-- echo -n "Stop Copying Me" | nc -4u -w1 localhost 1234 -->

        <Strategy class="Random"/>

        <Logger class="File" >
            <Param name="Path" value="logs"/>
        </Logger>
    </Test>
</Peach>
```

changeState

The *changeState* Action transitions to a different state within the [StateModel](#). The *changeState* Action typically includes the [when](#) attribute to identify when to switch states based on received data.

changeState is useful for network protocols that select response data based on an input code. The test target usually progresses sequentially through the StateModel, often removing the need to change states if the path of events are predefined.

Syntax

```
<StateModel name="TheStateModel" initialState="InitialState">
  <State name="InitialState">

    <Action type="input">
      <DataModel ref="InputModel" />
    </Action>

    <Action type="changeState" ref="State2"
      when="self.dataModel.find('Value').InternalValue == 'FOO' />

  </State>

  <State name="State2">
    <Action type="output">
      <DataModel ref="OutputModel" />
    </Action>
  </State>
</StateModel>
```

Attributes

Required:

type

Action type, must be set to "changeState"

Optional:

name

Name used to identify the action

ref

Name of the State to transition to in StateModel

when

Perform action if the provided expression evaluates to true

onStart

Evaluate expression at start of an action

onComplete

Evaluate expression upon completion of an action

Child Elements

None.

Examples

Example 198. Action changeState Example

The following example changes behavior based on input received from the target client.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

  <DataModel name="Ping">
    <Choice>
      <String name="PingPingStr" value="PINGPING" token="true" />
      <String name="PingStr" value="PING" token="true"/>
    </Choice>
  </DataModel>

  <DataModel name="Pong">
    <String value="PONG" />
  </DataModel>

  <DataModel name="PongPong">
    <String value="PONGPONG" />
  </DataModel>

  <StateModel name="TheStateModel" initialState="InitialState">
    <State name="InitialState">

      <Action type="accept" />

      <Action type="input">
        <DataModel ref="Ping"/>
      </Action>

    </State>
  </StateModel>
</Peach>
```

```

<!-- Switch states only when input was PINGPING -->
<Action type="changeState" ref="PongPongBack"
    when="state.actions[1].dataModel.find('PingPingStr') != None" />

<Action type="output">
    <DataModel ref="Pong"/>
</Action>

</State>

<!-- This state is only reached when input was PINGPING -->
<State name="PongPongBack">

<Action type="output">
    <DataModel ref="PongPong"/>
</Action>

</State>

</StateModel>

<Test name="Default">
    <StateModel ref="TheStateModel"/>
    <Publisher class="TcplListener">
        <Param name="Interface" value="0.0.0.0" />
        <Param name="Port" value="31337" />
        <Param name="AcceptTimeout" value="10000" />
        <Param name="Timeout" value="10000" />
    </Publisher>

    <Logger class="File" >
        <Param name="Path" value="logs"/>
    </Logger>
</Test>
</Peach>

```

Example 199. Looping Based on Input

The following example loops through received input until the string "PING" is found.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="Ping">

```

```

<Choice>
  <String name="PingPingStr" value="PINGPING" token="true" />
  <String name="PingStr" value="PING" token="true"/>
</Choice>
</DataModel>

<DataModel name="Pong">
  <String value="PONG" />
</DataModel>

<StateModel name="TheStateModel" initialState="InitialState">
  <State name="InitialState">
    <Action type="accept" />
    <Action type="checkState" ref="ReceiveInput" />
  </State>
  <State name="ReceiveInput">
    <Action type="input">
      <DataModel ref="Ping"/>
    </Action>
    <!-- Switch states only when input was PINGPING -->
    <Action type="changeState" ref="PongPongBack"
      when="state.actions[0].dataModel.find('PingPingStr') != None" />
    <!-- Run this state again -->
    <Action type="changeState" ref="ReceiveInput" />
  </State>
  <!-- This state is only reached when input was PINGPING -->
  <State name="SendOutput">
    <Action type="output">
      <DataModel ref="Pong"/>
    </Action>
  </State>
</StateModel>

<Test name="Default">
  <StateModel ref="TheStateModel"/>
  <Publisher class="TcpListener">

```

```
<Param name="Interface" value="0.0.0.0" />
<Param name="Port" value="31337" />
<Param name="AcceptTimeout" value="10000" />
<Param name="Timeout" value="10000" />
</Publisher>

<Logger class="File" >
  <Param name="Path" value="logs"/>
</Logger>
</Test>
</Peach>
```

Result

Result is a child of the [call](#) Action.

Methods run with a call Action often return useful data. Result allows cracking this returned data into a DataModel. This data can then be slurped into other DataModels and re-used for various purposes.

Syntax

```
<Action type="call" method="Players[1].GetUrl">
  <Result>
    <DataModel ref="TheDataModel" />
  </Result>
</Action>
```

Attributes:

[name](#)

Name of parameter [optional].

Valid Child Elements:

[DataModel](#)

Reference to a DataModel to contain the cracked data.

Godel

Godel elements allow scripting expressions to be evaluated when the Peach StateModel executes. If any of the Godel scripting expressions evaluates to false, Peach generates a fault.

A single Godel element can be placed as a child of [StateModel](#), [State](#) or [Action](#):

- When a Godel element is a child of [StateModel](#), the expression evaluates when the StateModel starts and when the StateModel finishes.
- When a Godel element is a child of [State](#), the expression evaluates when the State starts and when the State finishes.
- When a Godel element is a child of [Action](#), the expression evaluate when the Action starts and finishes.

Godel elements support the following scripting expressions:

- *inv* - An invariant expression that evaluates both at the start and at the finish of the appropriate level of the StateModel (StateModel, State, or Action)
- *pre* - An expression evaluates at start of the appropriate level of the StateModel
- *post* - A expression that is evaluated upon completion of the appropriate level of the StateModel

Syntax

```
<Godel inv="self != None"/>
```

Attributes

Required:

None.

Optional:

[name](#)

Name of the data model

[ref](#)

Reference to a [Godel] to use as a template.

[inv](#)

Scripting expression that evaluates to true or false. Default is null.

[pre](#)

Scripting expression that evaluates to true or false. Default is null.

post

Scripting expression that evaluates to true or false. Default is null.

Scripting Expressions

When evaluating the *inv* and *pre* scripting expressions two variables are provided: *self* and *context*:

- *self* is a reference to the parent element.

For example, when a Godel element is a child of State, *self* is the state.

- *context* is a reference to the Peach RunContext that contains all state information regarding the current fuzzing run.

When evaluating the *post* scripting expressions, an additional variable *pre* is provided.

- The *pre* variable is a reference to the previous self from when the parent element started.

For example, when a *post* expression is placed on an Action, this expression can access the action as it was prior to completing with the *pre* variable.

Examples

The following example uses a simple Godel pre-condition on StateModel, State and Action

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="DM">
        <String name="str" value="Hello"/>
    </DataModel>

    <StateModel name="SM" initialState="Initial">
        <Godel pre="str(self.states[0].actions[0].dataModel.find('str').InternalValue) ==
'Hello'" />

        <State name="Initial">
            <Godel pre="str(self.actions[0].dataModel.find('str').InternalValue) ==
'Hello'" />

            <Action type="output">
                <Godel pre="str(self.dataModel.find('str').InternalValue) == 'Hello'" />

                <DataModel ref="DM"/>
            </Action>

        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="SM"/>
        <Publisher class="Console"/>
    </Test>
</Peach>

```

Produces the following output:

```
>peach -l example.xml --debug

Godel.Core.GodelPitParser Attached godel node to Action 'SM.Initial.Action'.
Godel.Core.GodelPitParser Attached godel node to State 'SM.Initial'.
Godel.Core.GodelPitParser Attached godel node to StateModel 'SM'.

[*] Test 'Default' starting with random seed 56278.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Godel.Core.GodelContext Godel pre: Passed. (StateModel 'SM')
Godel.Core.GodelContext Godel pre: Passed. (State 'SM.Initial')
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Godel.Core.GodelContext Godel pre: Passed. (Action 'SM.Initial.Action')
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(5 bytes)
HelloPeach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

The following example shows using a reference to a Godel template

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="DM">
        <String name="str" value="Hello"/>
    </DataModel>

    <Godel name="check" pre="str(self.dataModel.find('str').InternalValue) == 'Hello'" />

    <StateModel name="SM" initialState="Initial">
        <State name="Initial">
            <Action type="output">
                <Godel ref="check"/>
                <DataModel ref="DM"/>
            </Action>

            <Action type="output">
                <Godel ref="check"/>
                <DataModel ref="DM"/>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="SM"/>
        <Publisher class="Console"/>
    </Test>
</Peach>

```

Produces the following output:

```
>peach -1 example.xml --debug

Godel.Core.GodelPitParser Attached godel node to Action 'SM.Initial.Action'.
Godel.Core.GodelPitParser Attached godel node to Action 'SM.Initial.Action_1'.

[*] Test 'Default' starting with random seed 54985.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Godel.Core.GodelContext Godel pre: Passed. (Action 'SM.Initial.Action')
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(5 bytes)
HelloPeach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Godel.Core.GodelContext Godel pre: Passed. (Action 'SM.Initial.Action_1')
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(5 bytes)
HelloPeach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.
```

21.8. Agents

A Peach agent is a light weight process that can host multiple monitors. These monitors perform tasks such as fault detection, data collection and automation.

The Peach agent communicates over a network channel to the main Peach process where all captured information is reported. This allows the main Peach process to perform monitoring locally or on the device under test.

Monitors that perform fault detection provide methods to identify when a problem with the target under test during testing. For example, you might use a debugger monitor to detect when a target crashes and collect information about the crash. Peach finds more issues when robust fault detection is configured.

Data collection monitors are used to gather additional information about a fault that has occurred. This can include taking a network capture of the test traffic, collecting log files and running scripts to collect information about the target state. The goal is to collect any information that will be useful in tracking down the root cause of the faulting condition.

Automation monitors are used to automate the target and target environment. This can include startup automation such as configuring an environment and starting the target. Restarting the environment when a fault occurs so testing can continue. And finally shutting down the target/environment when testing has completed. Some configurations may also require triggering the target to connect/consume data.

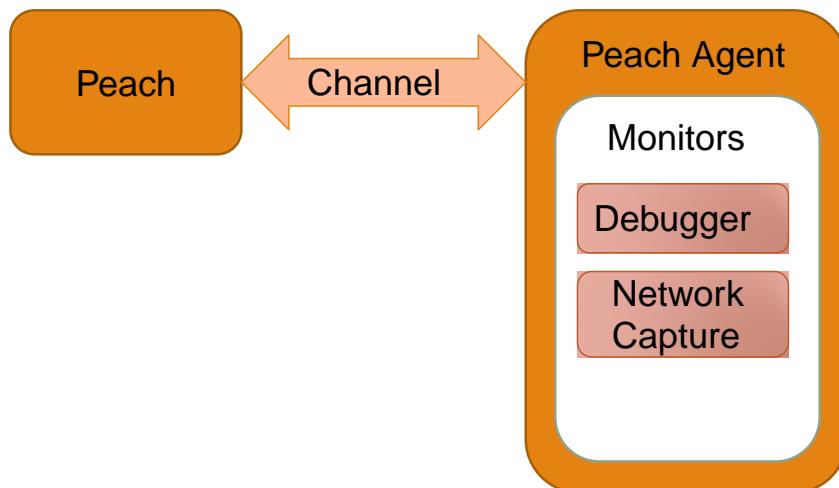


Figure 2. Peach Agent Block Diagram

Agent Privileges

Some monitors or publishers hosted by an agent process sometimes require heightened privileges. If you receive an error regarding permissions, try running the agent with root or administrative privileges.

OS X/Linux/Unix

On OS X, Linux and other Unix systems, a process can be launched as root using the *sudo* utility.

For local agents:

```
sudo ./peach
```

For remote agents:

```
sudo ./peachagent
```

Windows

On Windows a process can be started with administrator rights by right-clicking and selecting "Run as Adminsitritor". Optionally the user can be given additional needed priviledges by the system administrator.

Privileged Monitors

Which monitors require additional privileges depends on the specific OS and privileges of the user, but typically the following require special rights:

- PageHeap (Windows)
- NetworkCapture (OS X/Linux/Unix)

Network Firewall

Many modern operating systems such as Windows and Linux come with a firewall that prevents incoming or outgoing network traffic. This can prevent a remote agent from communicating with Peach. On Windows the user is asked when program first runs if a firewall rule should be added.

On Linux, specifically Ubuntu distributions, this command allows packets through the firewall using port 9001 (default the port for remote agents).

```
sudo ufw allow 9001
```

21.8.1. Agent Channels

Local Agent (local)

The agent is hosted in the current Peach process. This is the default channel. An example agent URL is `local://`

Remote Agent (tcp)

The external Peach agent communicates using network calls over TCP. Peach can run as a remote agent using the [PeachAgent](#) program. Remote agents do not require a valid license and are only used to host monitors or remote publishers. An example agent URL is `tcp://192.168.1.2:9002`

Local Agent

The peach runtime supports a local agent that runs in process. LocalAgent is the default agent unless another agent type is specified.

Agent URL:

```
local://
```

20. Configuring a local agent

```
<Agent name="local">
    <!-- Monitors -->
</Agent>
```

Remote Agent

Peach includes a remote [agent server](#) that can be used to host Monitors and Publishers on remote machines. Usage of a remote agent requires a location URL of the following format:

Agent URL:

```
tcp://HOST:PORT
```

HOST

Remote host the agent is running on

PORT

Remote port the agent is bound to (defaults to 9001)

Example:

```
tcp://192.168.1.100:9001
```

21. Configuring a local agent

```
<Agent name="local" location="tcp://192.168.1.100:9001">
  <!-- Monitors -->
</Agent>
```

REST JSON Agent

This agent is intended to communicate with custom remote agents written in other languages.

22. Example configuration with remote publisher

```
<Agent name="TheAgent" location="http://127.0.0.1:9980">
    <Monitor class="WindowsDebugger">
        <Param name="Executable" value="c:\windows\system32\mspaint.exe" />
        <Param name="Arguments" value="fuzzed.png" />
        <Param name="WinDbgPath" value="C:\Program Files (x86)\Debugging Tools for Windows (x86)" />
        <Param name="StartOnCall" value="ScoobySnacks"/>
    </Monitor>
    <Monitor class="PageHeap">
        <Param name="Executable" value="c:\windows\system32\mspaint.exe" />
        <Param name="WinDbgPath" value="C:\Program Files (x86)\Debugging Tools for Windows (x86)" />
    </Monitor>
</Agent>

<Test name="Default">
    <Agent ref="TheAgent"/>
    <StateModel ref="TheState"/>

    <Publisher class="Remote">
        <Param name="Agent" value="TheAgent"/>
        <Param name="Class" value="File"/>
        <Param name="FileName" value="fuzzed.png"/>
    </Publisher>
</Test>
```

GET /Agent/AgentConnect (Required)

Peach instance is connecting to a remote agent. When called, remote agent should reset it's current state, closing any open monitors or publishers. This method is called once per fuzzing session, unless the connection was lost due to the target system crashing or being reset by post-fault automation.

Responses

HTTP Code	Description	Schema
200	OK	No Content

GET /Agent/AgentDisconnect (Required)

Peach instance is disconnecting from the remote agent. When called, the remote agent should release any resources created, including monitors and publishers. This method is called once at the end of a fuzzing session. This method should never return an error.

Responses

HTTP Code	Description	Schema
200	OK	No Content

POST /Agent/StartMonitor

Start a monitor. This is called after [AgentConnect](#) to start an instance of a monitor. The parameters contain the information configured in the [Monitor](#) element of the Pit file.

Parameters

Type	Name	Description	Required	Schema
Query	name	Name of monitor	True	
Query	cls	Monitor class. Value from class attribute.	True	
Body		Arguments for monitor.	True	Args

Responses

HTTP Code	Description	Schema
200	OK	No Content

GET /Agent/StopAllMonitors (Required)

Stop all active monitors. Typically called prior to [AgentDisconnect](#).

Responses

HTTP Code	Description	Schema
200	OK	No Content

GET /Agent/SessionStarting (Required)

Session starting. Called once to indicate a fuzzing job is starting.

Responses

HTTP Code	Description	Schema
200	OK	No Content

GET /Agent/SessionFinished (Required)

Session finished. Called once to indicate a fuzzing job has finished. Typically called prior to [StopAllMonitors](#) and [AgentDisconnect](#).

Responses

HTTP Code	Description	Schema
200	OK	No Content

GET /Agent/IterationStarting (Required)

Iteration starting. Called at the start of each test case.

Parameters

Type	Name	Description	Required	Schema
Query	isReproduction	Is current test case part of fault reproduction?	true	Boolean
Query	lastWasFault	Was last iteration a fault?	true	Boolean
Query	iterationCount	DEPRECATED, always 0	true	Integer

Responses

HTTP Code	Description	Schema
200	OK	No Content

GET /Agent/IterationFinished (Required)

Iteration finished. Called at the end of each test case.

Responses

HTTP Code	Description	Schema
200	OK	No Content

GET /Agent/DetectedFault (Required)

Was a fault detected? Called after [IterationFinished](#) for each test case.

Responses

HTTP Code	Description	Schema
200	OK	DetectedFault Response

GET /Agent/GetMonitorData (Required)

Return monitor data. Called when a fault has been detected on the current test case and the engine is collecting data from the agents/monitors. Data for each monitor that has been started is returned in the results of this call.



GetMonitorData can be called even if [DetectedFault](#) returns false if another peach component (agent, publisher, etc.) indicates a fault has occurred.

Responses

HTTP Code	Description	Schema
200	OK	GetMonitorData Response

GET /Agent/Message (Required)

Message (Event) from state model. Called when an event is broadcast from the StateModel using an action type *call* with a publisher of *Peach.Agent*.

Parameters

Type	Name	Description	Required	Schema
Query	msg	Message/event	True	String

Responses

HTTP Code	Description	Schema
200	OK	No Content

POST /Publisher/CreatePublisher

Create a publisher hosted in the remote agent. Only a single Publisher can be created per remote agent.

Parameters

Type	Name	Description	Required	Schema
Body			True	CreatePublisher Request

Responses

HTTP Code	Description	Schema
200	OK	PublisherResponse

GET /Publisher/start

Action of type *start* called on publisher.

Responses

HTTP Code	Description	Schema
200	OK	PublisherResponse

GET /Publisher/stop

Action of type *stop* called on publisher.

Responses

HTTP Code	Description	Schema
200	OK	PublisherResponse

GET /Publisher/open

Action of type *open* called on publisher. Prior to *open* being called both [Set_Iteration](#) and [Set_IsControlIteration](#) are called.

Responses

HTTP Code	Description	Schema
200	OK	PublisherResponse

GET /Publisher/Set_Iteration

Provide the current iteration number to the publisher. This is called prior to the first [open](#) or [call](#).

Parameters

Type	Name	Description	Required	Schema
Body			true	Iteration

Responses

HTTP Code	Description	Schema
200	OK	PublisherResponse

GET /Publisher/Set_IsControlIteration

Provide the current iteration number to the publisher. This is called prior to the first [open](#) or [call](#).

Parameters

Type	Name	Description	Required	Schema
Body			true	IsControlIteration

Responses

HTTP Code	Description	Schema
200	OK	PublisherResponse

GET /Publisher/close

Action of type [open](#) called on publisher.

Responses

HTTP Code	Description	Schema
200	OK	PublisherResponse

GET /Publisher/accept

Action of type *accept* called on publisher. Call should block until completion.

Responses

HTTP Code	Description	Schema
200	OK	PublisherResponse

GET /Publisher/call

Action of type *call* called on publisher.

Parameters

Type	Name	Description	Required	Schema
Body			true	Call

Responses

HTTP Code	Description	Schema
200	OK	Result

POST /Publisher/setProperty

Action of type *setProperty* called on publisher.

Parameters

Type	Name	Description	Required	Schema
Body			true	SetProperty

Responses

HTTP Code	Description	Schema
200	OK	PublisherResponse

GET /Publisher/getProperty

Action of type *getProperty* called on publisher.

Parameters

Type	Name	Description	Required	Schema
Body	property	The property to retrieve the value of.	true	Json String

Responses

HTTP Code	Description	Schema
200	OK	Result

POST /Publisher/output

Action of type *output* called on publisher.

Parameters

Type	Name	Description	Required	Schema
Body	data	Data to output	true	Output

Responses

HTTP Code	Description	Schema
200	OK	PublisherResponse

GET /Publisher/input

Action of type *input* called on publisher. Calls to [WantBytes](#) will be made to read the input data as needed by the data cracker following an *input* call.

Responses

HTTP Code	Description	Schema
200	OK	PublisherResponse

GET /Publisher/WantBytes

Called to read data during an *input* action. Always follows a call to [input](#).

Parameters

Type	Name	Description	Required	Schema
Body	count	Number of bytes to read	true	Count

Responses

HTTP Code	Description	Schema
200	OK	DataResponse

Schemas

The following are sent/received in the JSON format.

Args

Variable set of arguments passed into our out of a call. Used by StartMonitor to pass monitor arguments.

23. Example monitor definition

```
<Monitor class="WindowsDebugger">
    <Param name="Executable" value="c:\windows\system32\mspaint.exe" />
    <Param name="Arguments" value="fuzzed.png" />
    <Param name="WinDbgPath" value="C:\Program Files (x86)\Debugging Tools for Windows (x86)" />
    <Param name="StartOnCall" value="ScoobySnacks"/>
</Monitor>
```

24. Resulting arguments object

```
{
  "args" : {
    "Executable" : "c:\\windows\\\\system32\\\\mspaint.exe",
    "Arguments" : "fuzzed.png",
    "WinDbgPath" : "C:\\\\Program Files (x86)\\\\Debugging Tools for Windows (x86)",
    "StartOnCall" : "ScoobySnacks"
  }
}
```

DetectedFault Response

Generic status response object.

```
{
    "Status" : "true",
}
```

Table 4. Parameters

Name	Description	Required	Schema
Status	Was a fault detected?	true	Bool

GetMonitorData Response

Collection of monitor data.

```
{
    "Results": [
        {
            "detectionSource":"",
            "monitorName":"",
            "collectedData":[
                {"": "data1", "Value": "AA=="}
            ]
        }
    ]
}
```

Table 5. Parameters

Name	Description	Required	Schema
Results	Array of monitor data, one entry per monitor	true	MonitorData

MonitorData Response

Data collected by a monitor.

```
{
    "detectionSource": "RunCommand",
    "monitorName": "CheckPid",
    "collectedData": [
        {"Key": "stdout.txt", "Value": "AA=="}
    ]
}
```

Table 6. Parameters

Name	Description	Required	Schema
detectionSource	Monitors class name	true	String
monitorName	Name attribute from monitor definition	true	String
collectedData	Array of assets collected/created by monitor	true	Data

Data Response

Named binary data.

```
{
  "Key": "stdout.txt",
  "Value": "AA=="
}
```

Table 7. Parameters

Name	Description	Required	Schema
Key	Filename for data	true	String
Value	Data (base64 encoded)	true	Bytes

PublisherResponse Response

Response for Publisher API

25. Example of non-error result

```
{
  "error": false,
}
```

26. Example of error result

```
{
  "error": true,
  "errorString": "Error creating publisher XYZ"
}
```

Table 8. Parameters

Name	Description	Required	Schema
error	Has an error occurred?	true	Boolean
errorString	Error message	false	String

CreatePublisher Request

Create publisher request object.

```
{
  "iteration": 1,
  "isControlIteration": false,
  "Cls": "Ioctl",
  "args":{
    "arg1": "xyz",
    "arg2": "xyz",
    "arg3": "xyz",
  }
}
```

Table 9. Parameters

Name	Description	Required	Schema
iteration	Iteration/testcase number	true	Integer
isControlIteration	Is this a control iteration	true	Boolean
Cls	Publisher class attribute	true	String
args	Publisher arguments	true	Object

Iteration

Create publisher request object.

```
{
  "iteration": 1
}
```

Table 10. Parameters

Name	Description	Required	Schema
iteration	Iteration/testcase number	true	Integer

IsControlIteration

Create publisher request object.

```
{
  "isControlIteration": false
}
```

Table 11. Parameters

Name	Description	Required	Schema
isControlIteration	Is this a control iteration	true	Boolean

Call

Information needed to complete a *call* action.

```
{
  "method": "PerformWork",
  "args" : [
    {"name":"firstName", "data":"AA==", "type":"in" }
  ]
}
```

Table 12. Parameters

Name	Description	Required	Schema
method	Method to call, maps to the <i>method</i> attribute of a <i>call</i> action.	true	String
args	Arugments for call. Zero or more.	true	Array of CallArg

CallArg

Argument for a call action.

```
{
  "name" : "firstName",
  "data" : "AA==",
  "type" : "in"
}
```

Table 13. Parameters

Name	Description	Required	Schema
name	Argument/parameter name	true	String
data	Binary data base64 encoded	true	Bytes
type	DEPRECATED, always <i>in</i>	true	String

Result

The result of a *call* or *getProperty* action.

27. Example of successful result

```
{
  "value": "AA==",
  "error": false,
}
```

28. Example of error result

```
{
  "value": null,
  "error": true,
  "errorString": "Error call method"
}
```

Table 14. Parameters

Name	Description	Required	Schema
value	Resulting data, can be null. Base64 encoded.	true	Bytes
error	Has an error occurred?	true	Boolean
errorString	Error message	false	String

DataResponse

The result of a *call* or *getProperty* action.

29. Example of successful result

```
{  
    "data": "AA==",  
    "error": false,  
}
```

30. Example of error result

```
{  
    "data": null,  
    "error": true,  
    "errorString": "Error call method"  
}
```

Table 15. Parameters

Name	Description	Required	Schema
data	Resulting data, can be null. Base64 encoded.	true	Bytes
error	Has an error occurred?	true	Boolean
errorString	Error message	false	String

Output

Data to output

```
{  
    "data": "AA=="  
}
```

Table 16. Parameters

Name	Description	Required	Schema
data	Property value. Base64 encoded.	true	Bytes

SetProperty

Contains information required to make a SetProperty call.

```
{
  "property": "FirstName",
  "data"      : "AA=="
}
```

Table 17. Parameters

Name	Description	Required	Schema
property	Property to set data on	true	String
data	Property value. Base64 encoded.	true	Bytes

Count

Count of bytes requested.

```
{
  "count": 1000,
}
```

Table 18. Parameters

Name	Description	Required	Schema
count	Property to set data on	true	Integer

21.9. Monitors

Agents are special Peach processes that can be run locally or remotely. These processes host one or more Monitors that can perform such actions as attaching debuggers, watching memory consumption, or detecting faults.

The following table lists each monitor by name, function type (Fault detection, Data collection, Automation), and by the operating systems that support the monitor. If the operating systems column is blank for an entry, that monitor is available in all supported operating systems.

Table 19. Monitors

Monitor	Fault Detection	Data Collection	Automation	Operating Systems [4: When an operating system is listed, the monitor is available only for the listed operating system. A blank entry indicates the monitor is available for Windows, Linux, and OS X operating systems.]
Android	X		X	
Android Emulator	X		X	
APC Power			X	
ButtonClicker			X	Windows
CanaKit Relay			X	
CAN Capture		X		
CAN Error Frame	X			
CAN Send Frame			X	
CAN Timing	X			
CAN Threshold	X			
Cleanup Folder			X	
Cleanup Registry			X	Windows
Crash Reporter	X	X		OS X
Crash Wrangler	X	X	X	OS X
Gdb	X	X	X	Linux, OS X
GdbServer	X	X	X	Linux, OS X

Monitor	Fault Detection	Data Collection	Automation	Operating Systems [4: When an operating system is listed, the monitor is available only for the listed operating system. A blank entry indicates the monitor is available for Windows, Linux, and OS X operating systems.]
IpPower9258			X	
LinuxCoreFile	X	X		Linux
Memory	X			
NetworkCapture	X	X		
Page Heap			X	Windows
Ping	X			
Popup Watcher	X		X	Windows
Process Launcher	X		X	
Process Killer			X	
Run Command	X	X	X	
Save File		X		
Serial Port	X	X	X	
SNMP Power			X	
Socket Listener	X	X		
SSH Command	X	X	X	
SSH Downloader		X		
Syslog	X	X	X	
TcpPort	X	X	X	
Vmware Control			X	
Windows Debugger	X	X	X	Windows
Windows Service	X		X	Windows

21.9.1. Android Monitor

Monitor Categories: Automation, Fault detection

The *Android* monitor examines both targeted Android applications and the state of the Android OS. Peach supports Android OS versions 4.0 to 5.1, inclusively.

Two expected use cases are:

- Maintaining state while fuzzing native code
- Launching and monitoring Android Java applications.

The *Android* monitor can start or restart the device at the following times:

- The start of a fuzzing run
- The start of each test iteration
- The start of an iteration that immediately follows a fault
- When called from the state model

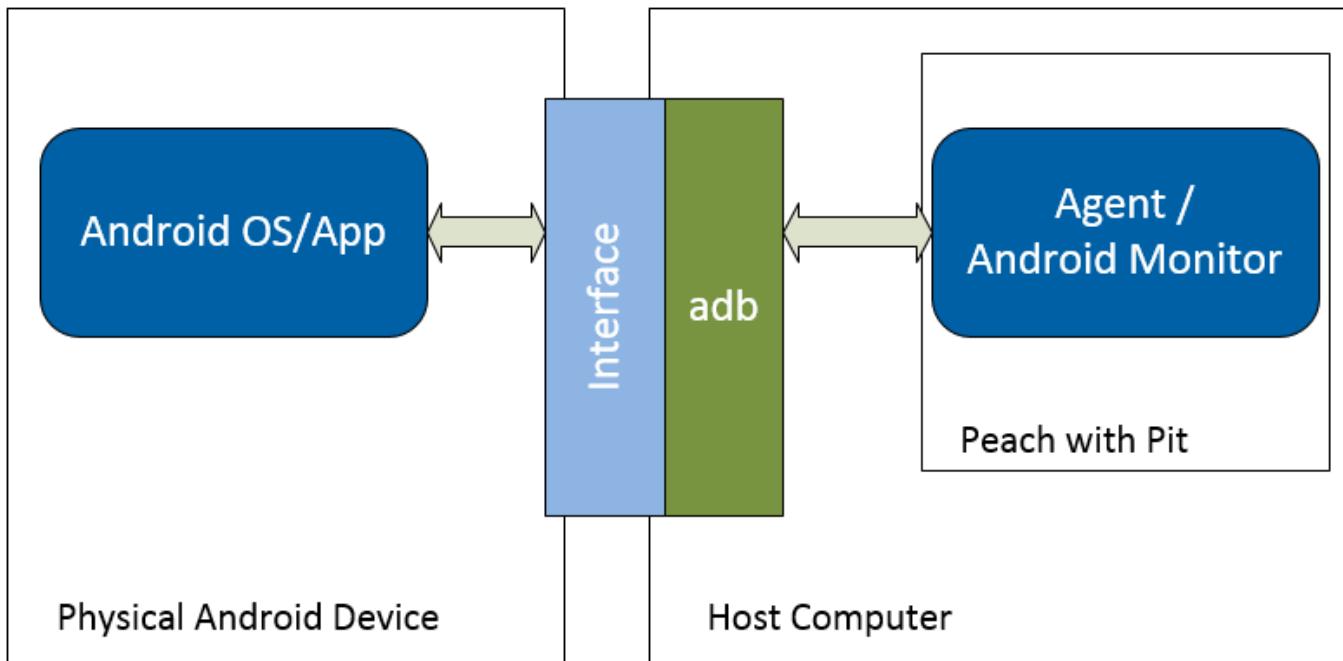
The Android monitor watches the message logs and the device, and can generate faults for the following conditions:

- A logging message matches the fault search criteria, Peach logs a fault.
- A logging message matches the fault search criteria, Peach logs a fault and stops fuzzing.
- A physical device becomes non-responsive.
- A virtual device becomes non-responsive or lost.

After detecting a fault, the monitor collects data from the device log files and crash dumps.

Additionally, the monitor logs exceptions, and updates fault bucket information. For bucketing, Peach uses the text from the fault to determine the major bucket level. The minor bucket level is not used. The risk evaluation looks for error, fatal error, or unknown.

The *Android* monitor uses the Android Debugging Bridge (adb) to communicate with a device. This monitor can target both emulated and physical devices. The *Android* monitor requires [Android Platform Tools](#) and either an emulator or a physical Android device. The configuration for a physical device follows. For a configuration using a virtual device, see the [Android Emulator Monitor](#).



Connecting to a physical device requires the device serial number. You can obtain this from a connected device by using the following adb command: "adb devices". The result is a list of devices that adb found. The information for each device consists of two parts: the device number and the connection status between adb and the device. The list includes physical and virtual devices.



For more information about debugging Android devices, see the following:

- [Android Debug Bridge](#).
- [How To Install and Use ADB, the Android Debug Bridge Utility](#)

Parameters

Required:

ApplicationName

Name of the Android application.

Optional:

ActivityName

Name of the application activity, defaults to "".

AdbPath

Directory path to adb, defaults to "".

ClearAppData

Removes the application data and cache every iteration, defaults to false.

ClearAppDataOnFault

Removes the application data and cache on faulting iterations, defaults to false.

CommandTimeout

Sets the maximum number of seconds to wait for the adb command to complete, defaults to 10 seconds.

ConnectTimeout

Sets the maximum number of seconds to wait to establish an adb connection, defaults to 5 seconds.

DeviceMonitor

Identifies the Android monitor that supplies the device serial number, defaults to "". Used when monitoring a virtual device.

DeviceSerial

The serial number of the device to monitor, defaults to "". Used when monitoring a physical device.

FaultRegex

Specifies a regular expression; when matched from a log entry, triggers a fault. The default pattern is (^E/ActivityMonitor)|(^E/AndroidRuntime)|(^F/.*)

FaultWaitTime

Sets the time period, in milliseconds, to wait when checking for a fault, defaults to 0 ms.

IgnoreRegex

Specifies a regular expression; when matched, the monitor ignores potential false positive faults, defaults to "".

MustStopRegex

Specifies a regular expression; when a match occurs, the monitor triggers a fault and stops fuzzing, defaults to "".

ReadyTimeout

Sets the maximum number of seconds to wait for the device to reach readiness—able to respond to inputs, defaults to 600 seconds.

RebootEveryN

Specifies the number of iterations between successive device reboots, defaults to 0.

RebootOnFault

Reboots the device when a fault occurs, defaults to false.

RestartEveryIteration

Restarts the application every iteration, defaults to false.

StartOnCall

Starts the application when notified by the state machine. The string value used here must match the Call Action statement of the state model. The default string is "".

WaitForReadyOnCall

Waits for the device to be ready when notified by the state machine. The string used here must match the corresponding Call Action statement of the state model. the default string is "".



The DeviceMonitor and the DeviceSerial parameters are mutually exclusive. Use DeviceSerial to provide the serial number of a physical device. Use DeviceMonitor when using the Android Emulator, as the Emulator will provide the serial number of the virtual device.

Examples

Example 200. Basic Usage Example

This example runs the BadBehaviorActivity, sending random taps to generate different types of exceptions and crashes.

To run the Android emulator, set your AdbPath to the directory containing the adb (Android Debug Bridge) platform-tools directory and point the EmulatorPath to the adb tools directory.

The Avd parameter must also be the name of a valid AVD (Android Virtual Device). To create a new AVD:

1. Open the *android.bat* file located in the adb SDK tools directory.
2. From the GUI that opens, click on *Tools* in the menu bar, then *Manage AVDs*....
3. From the window that opens, click *New...* and create a new AVD.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="TheDataModel">
    <Number size='32' signed="false" value="31337" />
</DataModel>

<DataModel name="X">
    <Number size='32' signed="false" value="100" />
</DataModel>

<DataModel name="Y">
    <Number size='32' signed="false" value="0" />
```

```

</DataModel>

<StateModel name="State" initialState="Initial" >
    <State name="Initial" >
        <Action type="call" method="tap">
            <Param>
                <DataModel ref="X"/>
            </Param>
            <Param>
                <DataModel ref="Y"/>
            </Param>
        </Action>
    </State>
</StateModel>

<Agent name="TheAgent">
    <Monitor name="Emu" class="AndroidEmulator">
        <Param name="Avd" value="Nexus4" />
        <Param name="EmulatorPath" value="C:\adt-bundle-windows-x86_64-
20131030\sdk\tools"/>
    </Monitor>

    <Monitor name="App" class="Android">
        <Param name="ApplicationName" value="com.android.development" />
        <Param name="ActivityName" value=".BadBehaviorActivity" />
        <Param name="AdbPath" value="C:\adt-bundle-windows-x86_64-
20131030\sdk\platform-tools"/>
        <Param name="DeviceMonitor" value="Emu" />
    </Monitor>
</Agent>

<Test name="Default">
    <StateModel ref="State"/>
    <Agent ref="TheAgent" />

    <Publisher class="AndroidMonkey">
        <Param name="DeviceMonitor" value="App"/>
    </Publisher>
</Test>
</Peach>

```

Output for this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 3054.
Peach.Core.Agent.Agent StartMonitor: Emu AndroidEmulator
Peach.Core.Agent.Agent StartMonitor: App Android
Peach.Core.Agent.SessionStarting: Emu
Peach.Enterprise.Agent.Monitors.AndroidEmulator Starting android emulator
Peach.Enterprise.Agent.Monitors.AndroidEmulator Resolved emulator instance to android
device 'emulator-5554'
Peach.Enterprise.Agent.Monitors.AndroidEmulator Android emulator 'emulator-5554'
successfully started
Peach.Core.Agent.SessionStarting: App
Peach.Enterprise.AndroidBridge Initializing android debug bridge.
Peach.Enterprise.AndroidBridge Android debug bridge initialized.
Peach.Enterprise.Agent.Monitors.AndroidMonitor Resolved device 'emulator-5554' from
monitor 'Emu'.
Peach.Enterprise.AndroidDevice Waiting for device 'emulator-5554' to become ready
Peach.Enterprise.AndroidDevice Device 'emulator-5554' is now ready
Peach.Enterprise.AndroidDevice Executing command on 'emulator-5554': am start -W -S
-n com.android.development/.BadBehaviorActivity

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Call
Peach.Enterprise.Publishers.AndroidMonkeyPublisher start()
Peach.Enterprise.Publishers.AndroidMonkeyPublisher call(tap,
System.Collections.Generic.List`1[Peach.Core.Dom.ActionParameter])
Peach.Core.Agent.AgentManager Message: App => DeviceSerial
Peach.Enterprise.Publishers.AndroidMonkeyPublisher Resolved device 'emulator-5554'
from monitor 'App'.
Peach.Enterprise.AndroidDevice Executing command on 'emulator-5554': input tap 100 0
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Enterprise.Publishers.AndroidMonkeyPublisher stop()
Peach.Core.Agent.SessionFinished: App
Peach.Enterprise.AndroidBridge Terminating android debug bridge.
Peach.Core.Agent.SessionFinished: Emu
Peach.Enterprise.Agent.Monitors.AndroidEmulator Sending stop command to emulator
'emulator-5554'
Peach.Enterprise.Agent.Monitors.AndroidEmulator Waiting for emulator 'emulator-5554'
to exit
Peach.Enterprise.Agent.Monitors.AndroidEmulator Emulator 'emulator-5554' exited with
code: 0
Peach.Enterprise.Agent.Monitors.AndroidEmulator Emulator 'emulator-5554' exited

[*] Test 'Default' finished.
```

21.9.2. AndroidEmulator Monitor

Monitor Categories: Automation, Fault detection

The *AndroidEmulator* monitor handles setup and teardown for Android Virtual Devices (AVDs) running within the Android Emulator. This monitor detects faults related to the emulator operation, not fuzzing results.

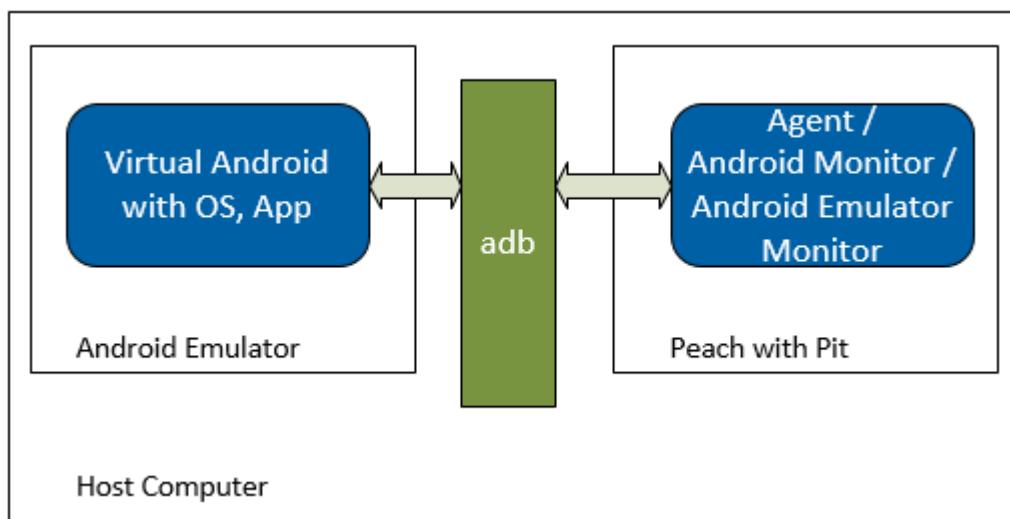
The monitor provides the following functionality:

- Start a virtual device at the start of a fuzzing run.
- Start a virtual device at the start of each test iteration.
- Start a virtual device when called from the state model.
- Start a virtual device at the beginning of an iteration that immediately follows a fault.
- Shuts down at the end of a fuzzing session.
- Logs timeout messages when querying the emulator for the device serial number.
- Logs timeout messages when shutting down the emulator.
- logs messages sent to StdErr.
- Logs messages sent to StdOut.

This monitor requires the following items to run:

- The Android monitor that watches the OS and application being fuzzed.
- The [Android Emulator](#)
- The [Android Platform Tools](#).

The fuzzing configuration for a virtual or emulated device follows. For a configuration using a physical device, see the [Android Monitor](#).



Parameters

Required:

Avd

Android virtual device.

Optional:

EmulatorPath

Directory containing the Android emulator. If not provided, Peach searches the directories in the PATH variable for the installed emulator.

Headless

If true, runs the emulator **without a display**. Defaults to false.

RestartAfterFault

If `true`, restarts the emulator when any monitor detects a fault. If `false`, restarts the emulator only if the emulator exits or crashes. This argument defaults to `true`.

RestartEveryIteration

Restart emulator on every iteration. Defaults to false.

StartOnCall

Start the emulator when notified by the state machine.

StartTimeout

How many seconds to wait for emulator to start running. Defaults to 30.

StopTimeout

How many seconds to wait for emulator to exit. Defaults to 30.

Examples

Example 201. Basic Usage Example

Runs the BadBehaviorActivity, sending random taps to generate different types of exceptions and crashes.

To run the Android emulator, set your AdbPath to the directory containing the adb (Android Debug Bridge) platform-tools directory and point the EmulatorPath to the adb tools directory.



Peach executes the monitors in the order that they are listed in the fuzzing definition. Position the Android Emulator Monitor before (above) the Android monitor in your Pit, so that, at run time, the virtual device exists when adb tries to connect to it.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <Number size='32' signed="false" value="31337" />
    </DataModel>

    <DataModel name="X">
        <Number size='32' signed="false" value="100" />
    </DataModel>

    <DataModel name="Y">
        <Number size='32' signed="false" value="0" />
    </DataModel>

    <StateModel name="State" initialState="Initial" >
        <State name="Initial" >
            <Action type="call" method="tap">
                <Param>
                    <DataModel ref="X"/>
                </Param>
                <Param>
                    <DataModel ref="Y"/>
                </Param>
            </Action>
        </State>
    </StateModel>

    <Agent name="TheAgent">
        <Monitor name="Emu" class="AndroidEmulator">
            <Param name="Avd" value="Nexus4" />
            <Param name="EmulatorPath" value="C:\adt-bundle-windows-x86_64-
20130717\sdk\tools"/>
        </Monitor>

        <Monitor name="App" class="Android">
            <Param name="ApplicationName" value="com.android.development" />
            <Param name="ActivityName" value=".BadBehaviorActivity" />
            <Param name="AdbPath" value="C:\adt-bundle-windows-x86_64-
20130717\sdk\platform-tools"/>
            <Param name="DeviceMonitor" value="Emu" />
        </Monitor>
    </Agent>

    <Test name="Default">

```

```
<StateModel ref="State"/>
<Agent ref="TheAgent" />

<Publisher class="AndroidMonkey">
    <Param name="DeviceMonitor" value="App"/>
</Publisher>
</Test>
</Peach>
```

Output for this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 3054.
Peach.Core.Agent.Agent StartMonitor: Emu AndroidEmulator
Peach.Core.Agent.Agent StartMonitor: App Android
Peach.Core.Agent.SessionStarting: Emu
Peach.Enterprise.Agent.Monitors.AndroidEmulator Starting android emulator
Peach.Enterprise.Agent.Monitors.AndroidEmulator Resolved emulator instance to android
device 'emulator-5554'
Peach.Enterprise.Agent.Monitors.AndroidEmulator Android emulator 'emulator-5554'
successfully started
Peach.Core.Agent.SessionStarting: App
Peach.Enterprise.AndroidBridge Initializing android debug bridge.
Peach.Enterprise.AndroidBridge Android debug bridge initialized.
Peach.Enterprise.Agent.Monitors.AndroidMonitor Resolved device 'emulator-5554' from
monitor 'Emu'.
Peach.Enterprise.AndroidDevice Waiting for device 'emulator-5554' to become ready
Peach.Enterprise.AndroidDevice Device 'emulator-5554' is now ready
Peach.Enterprise.AndroidDevice Executing command on 'emulator-5554': am start -W -S
-n com.android.development/.BadBehaviorActivity

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Call
Peach.Enterprise.Publishers.AndroidMonkeyPublisher start()
Peach.Enterprise.Publishers.AndroidMonkeyPublisher call(tap,
System.Collections.Generic.List`1[Peach.Core.Dom.ActionParameter])
Peach.Core.Agent.AgentManager Message: App => DeviceSerial
Peach.Enterprise.Publishers.AndroidMonkeyPublisher Resolved device 'emulator-5554'
from monitor 'App'.
Peach.Enterprise.AndroidDevice Executing command on 'emulator-5554': input tap 100 0
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Enterprise.Publishers.AndroidMonkeyPublisher stop()
Peach.Core.Agent.SessionFinished: App
Peach.Enterprise.AndroidBridge Terminating android debug bridge.
Peach.Core.Agent.SessionFinished: Emu
Peach.Enterprise.Agent.Monitors.AndroidEmulator Sending stop command to emulator
'emulator-5554'
Peach.Enterprise.Agent.Monitors.AndroidEmulator Waiting for emulator 'emulator-5554'
to exit
Peach.Enterprise.Agent.Monitors.AndroidEmulator Emulator 'emulator-5554' exited with
code: 0
Peach.Enterprise.Agent.Monitors.AndroidEmulator Emulator 'emulator-5554' exited

[*] Test 'Default' finished.
```

21.9.3. APC Power Monitor

Monitor Categories: Automation

The *APC Power* monitor switches outlets on an APC power distribution unit (PDU) on and off via SNMPv1. This monitor is useful for automatically power cycling devices during a fuzzing session. APC's Switched Rack Power Distribution Unit (AC7900) is known to work with this monitor.

Each *APC Power* monitor switches one or more of a PDU's outlets, according to the configuration. All affected outlets are given the same commands, so turning some outlets on and others off would require another monitor. The monitor can reset the power outlets at the following points in time:

- At the start or end of a fuzzing run
- At the start or end of each test iteration
- After detecting a fault
- At the start of an iteration that immediately follows a fault
- When a specified call is received from the state model



The [IpPower9258 Monitor](#) provides similar features, specific to the IP Power 9258 devices. The [SnmpPower Monitor](#) is designed to work with non-APC PDUs that can be controlled via SNMPv1. For controlling power to a device by wiring through a relay, Peach provides a monitor for the [CanaKit 4-Port USB Relay Controller](#).

Parameters

Required:

Host

IP address of the switched power distribution unit.

OutletGrouping

Whether outlets on the PDU are identified individually ([Outlet](#)) or in groups ([OutletGroup](#)). Default is [Outlet](#).

Outlets

Comma-separated list of numeric identifiers for outlets or outlet groups to control.

Optional:

Port

SNMP port on the switched power distribution unit. Default is [161](#).

ReadCommunity

SNMP community string to use when reading the state of the outlets. Default is [public](#).

WriteCommunity

SNMP community string to use when modifying the state of the outlets. Default is **private**.

RequestTimeout

Maximum duration in milliseconds to block when sending an SNMP request to the PDU. Default is **1000**.

SanityCheckOnStart

On startup, ensure switch state changes persist. Default is **true**.

SanityCheckWaitTimeout

Maximum duration to wait for state change to take effect during startup sanity check. Default is **3000**.

ResetOnCall

Reset power when the specified call is received from the state model. This value is used only when the *When* parameter is set to **OnCall**.

PowerOffOnEnd

Power off when the fuzzing session completes, default is **false**.

PowerOnOffPause

Pause in milliseconds between power off/power on, default is **500**.

When

When to reset power on the specified outlets or outlet groups. Default is **OnFault**.

"When" Setting	Description
DetectFault	Reset power when checking for a fault. This occurs after OnIterationEnd.
OnStart	Reset power when the fuzzing session starts. This occurs once per session.
OnEnd	Reset power when the fuzzing session stops. This occurs once per session.
OnIterationStart	Reset power at the start of each iteration.
OnIterationEnd	Reset power at the end of each iteration.
OnFault	Reset power when any monitor detects a fault. This is the default setting.
OnIterationStartAfterFault	Reset power at the start of an iteration that immediately follows a fault detection.

"When" Setting	Description
OnCall	Reset power when the call specified by the <i>ResetOnCall</i> parameter is received from the state model.

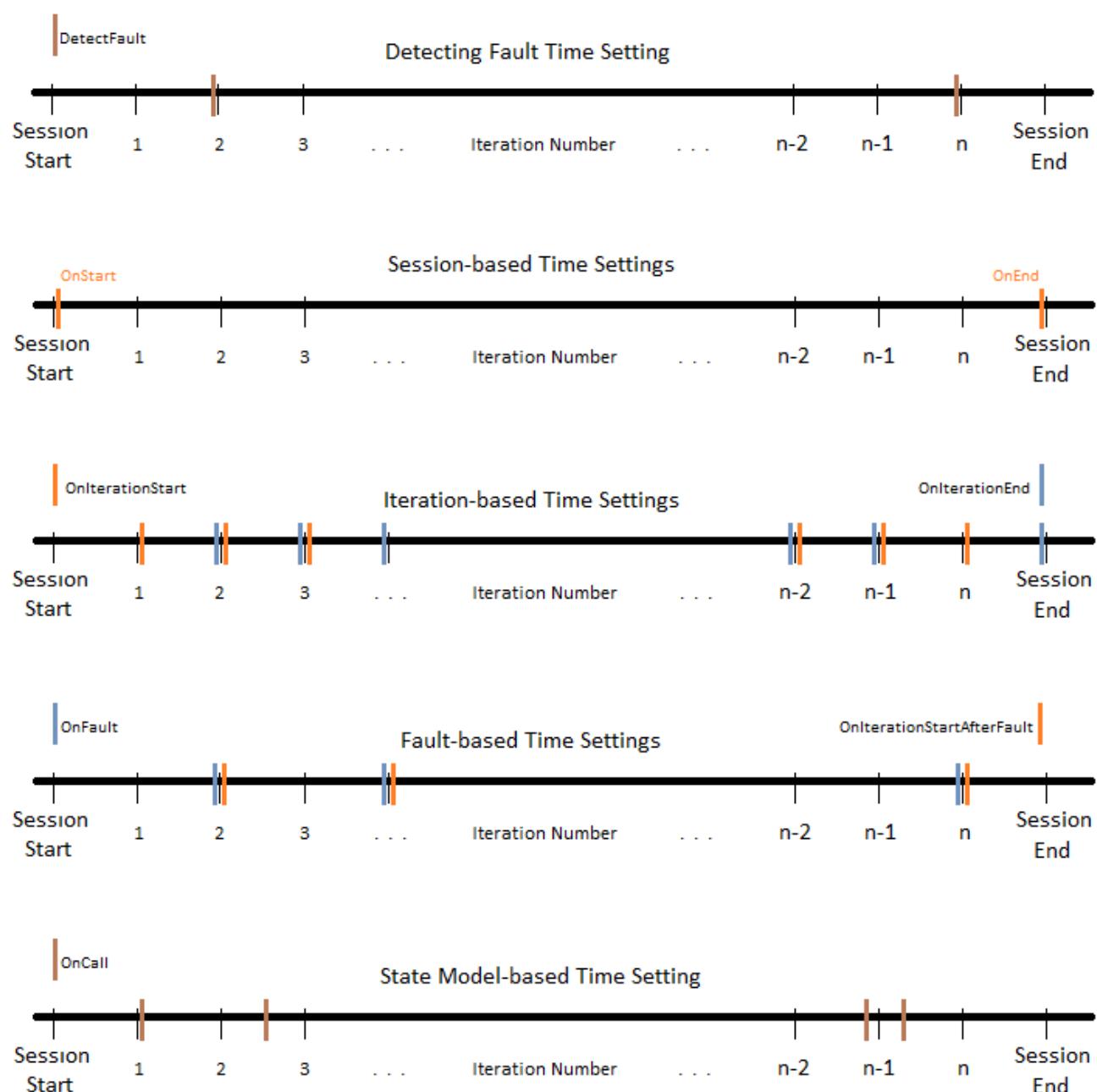


Figure 3. When Choices for Performing an Action

Examples

Example 202. Reset power on ports 1 and 2 of an APC PDU

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="TheDataModel">
    <Number size="32" signed="false" value="31337" />
</DataModel>

<StateModel name="State" initialState="Initial" >
    <State name="Initial">
        <Action type="output">
            <DataModel ref="TheDataModel"/>
        </Action>
    </State>
</StateModel>

<Agent name="Local">
    <Monitor class="ApcPower">
        <Param name="Host" value="10.0.1.101" />
        <Param name="Outlets" value="1,2" />

        <!-- For APC devices that manage groups of outlets, be sure to
            set OutletGrouping. -->
        <!-- <Param name="OutletGrouping" value="OutletGroup" /> -->
    </Monitor>
</Agent>

<Test name="Default">
    <StateModel ref="State"/>
    <Agent ref="Local" />
    <Publisher class="ConsoleHex"/>
</Test>
</Peach>
```

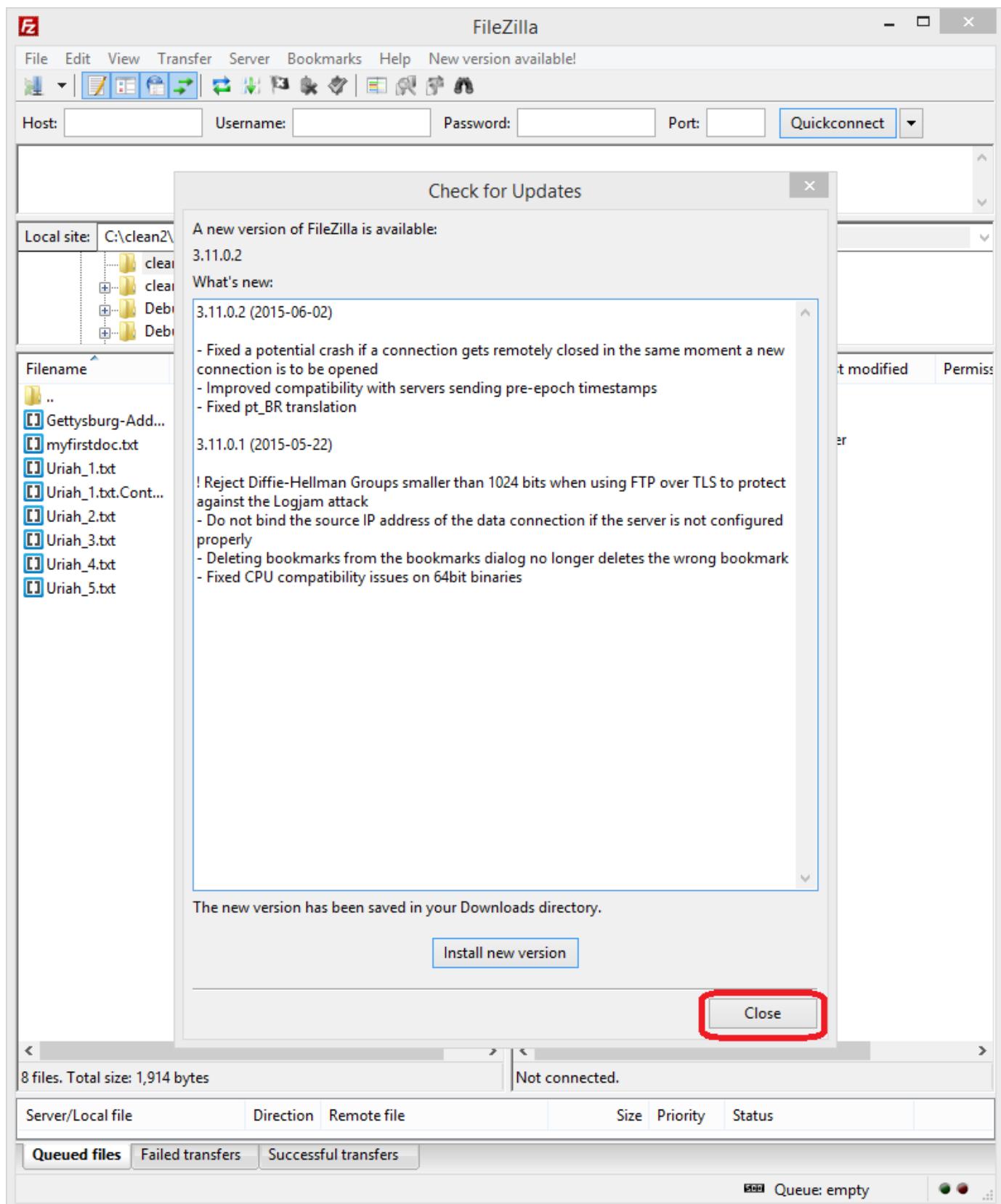
21.9.4. ButtonClicker Monitor

Monitor Category: Automation

The *ButtonClicker* monitor provides automation functionality by clicking buttons in the Windows GUI. This monitor runs in the Windows environment. *ButtonClicker* watches and clicks the appropriate button within the specified window. You can use *ButtonClicker* to click a button in the window, such as "OK" or "Close".

The intent of this monitor is to initiate an action or to close a window to keep a fuzzing session active. *Buttonclicker* runs from the beginning of a fuzzing session to the end of the session.

The following example uses an FTP client, FileZilla, that sometimes opens a popup window asking whether to install an update. Here, *ButtonClicker* monitor provides a mouse click to the "Close" button to close the popup window. With the popup window out of the way, fuzzing continues without delay.



Another monitor to consider for watching popup windows is [PopupWatcher](#), that can monitor several popup windows.

Parameters

Required:

WindowText

Text from the window title that identifies the window to receive the button click. The text string can be part or all of the window title.

ButtonName

Text label of the button to click. The label is displayed to the user, and is on or near the button that will receive the click.

Optional:

None.

If Peach has trouble clicking a button, the button might have a link to a shortcut key.

The shortcut key is displayed in the label using an underlined character. Windows does this by inserting an ampersand "&" immediately before the shortcut key within the button label. Further, not all underlining shows in the initial display of the window.

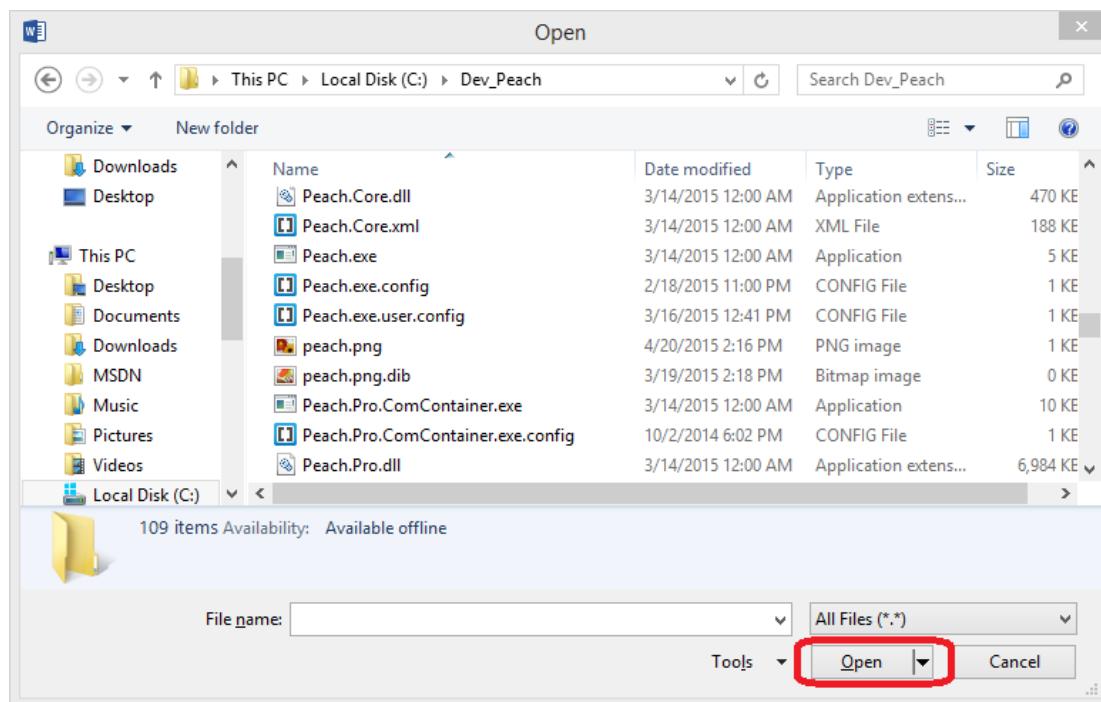
In an application, you can manually force underlining to display in window buttons by pressing the <CTRL> or <ALT> key. Once you find the underlining in the application, you can adjust the value of the **ButtonName** parameter for *ButtonClick* by inserting an ampersand (&) immediately before any underlined character. Then, Peach will find the button to click.

For example, in Microsoft Word, the Browsing dialog box used to open a document does not immediately display underlining in its command buttons. You can see this by following the sequence that opens a file:

1. Click **File** on the ribbon.
2. Click **Open** on the vertical menu.
3. Click **Computer** in the Open column.
4. Click the **Browse** button.



The dialog opens and the "Open" button in the lower right corner is not underlined. Press the <ALT> or <CTRL> key to see the underlining as in the following illustration.



Examples

Basic Usage Example

This example uses the ButtonClicker monitor to respond to a question from a pop-up window that displays when restarting an application after a crash.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <Number size="32" signed="false" value="31337" />
    </DataModel>

    <StateModel name="State" initialState="Initial" >
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="Local">
        <Monitor class="ButtonClicker">
            <Param name="WindowText" value="Do you want to start in safe mode?" />
            <Param name="ButtonName" value="No" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <StateModel ref="State"/>
        <Agent ref="Local" />
        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>
```

21.9.5. CanaKitRelay Monitor

Monitor category: Automation

The *CanaKitRelay* monitor provides automation to a number of test configurations:

- Control power to an external device that is the fuzzing target. You can turn it on at the start of the test, then toggle power after a fault occurs to return the device to a known, stable state.
- Control a supporting device in a fuzzing test configuration. Supporting devices can include recording devices, lighting, heating devices, sound or motion generators. An example is turning on an espresso machine after every 10,000th test iteration.
- Emulate a button push by inserting the device into a circuit containing the button. The idea is to automate the press of a button, such as **NumLock**, or the play button on a surveillance device.
- For simulating the attachment or removal of a cable, such as USB, by routing the power line (VCC) through the relay.

The CanaKit is an external product that you can add as part of your test configuration. The kit consists of 4 relays. A relay is an electrically operated switch that uses an electromagnet to operate a switching mechanism. Each relay in the kit is capable of controlling a 5-amp, 110V AC or a 24V DC circuit. Communication with the kit occurs over USB.

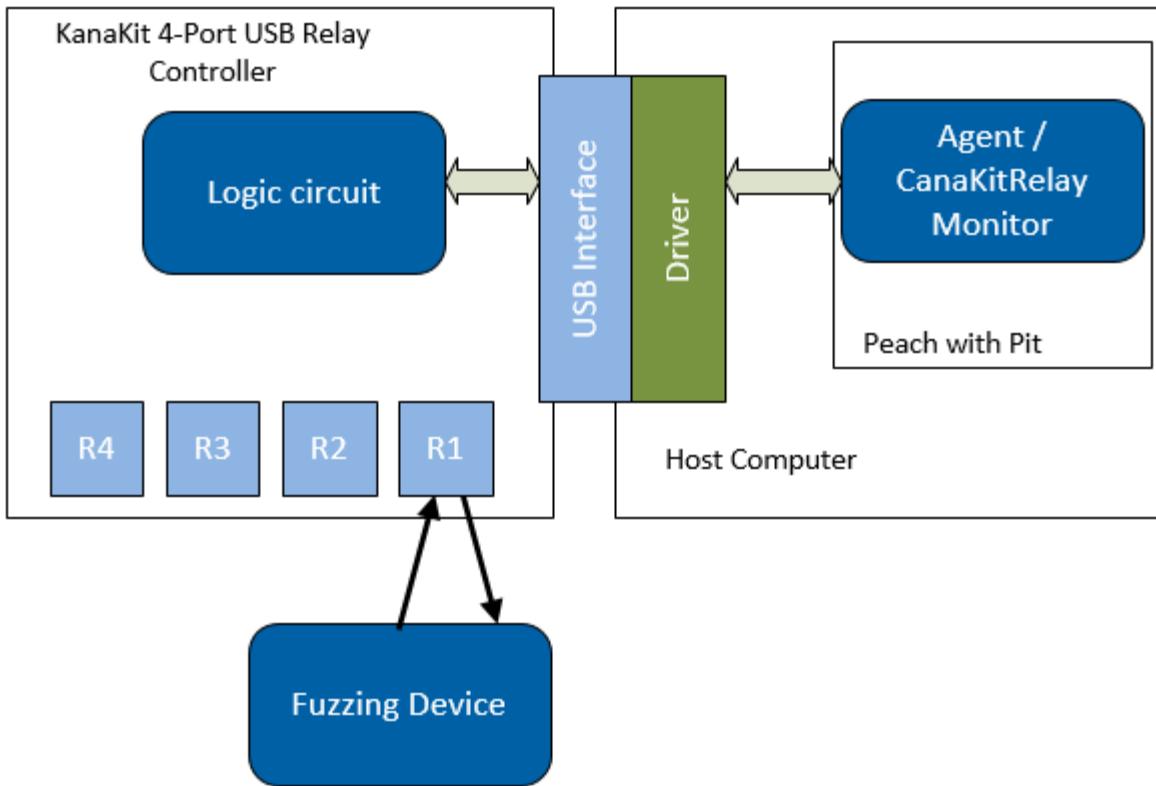
For more information on the kit, including configurations, installation, and the relay command set, see [CanaKit 4-Port USB Relay Controller](#).

You can purchase the CanaKit from the manufacturer at the previous web site or from Amazon.com. In each case, the kit price is about \$60.00 U.S. plus shipping costs.

The *CanaKitRelay* monitor controls one relay in a kit. Use one monitor per relay to control multiple relays concurrently. Within a fuzzing session, the monitor can trigger the relay at the following times:

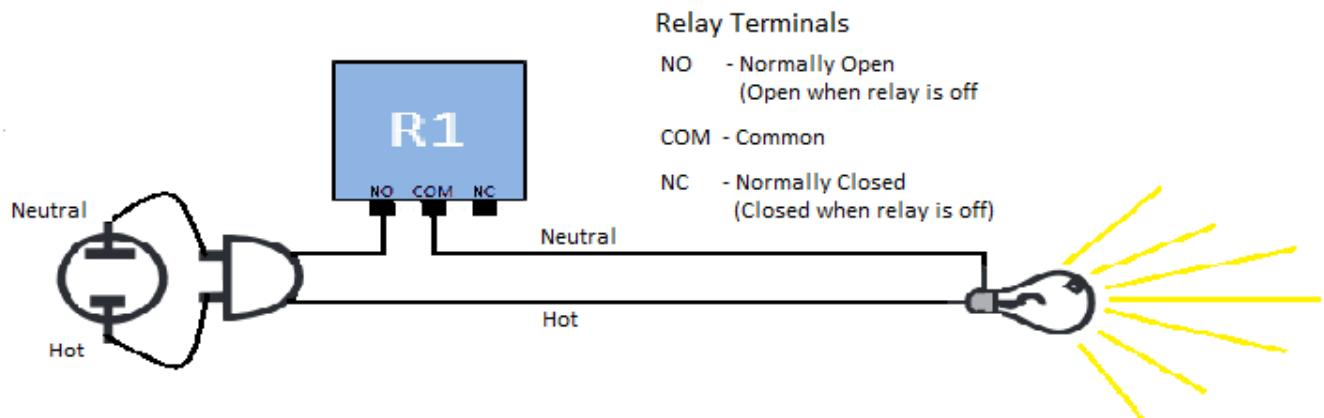
- At the start or end of a fuzzing run
- At the start or end of each test iteration
- During the detection of a fault
- After detecting a fault
- At the start of an iteration that immediately follows a fault
- When a specified call is received from the state model

The following diagram shows a sample configuration that controls power through Relay 1.



Each relay supplies three terminals: Normally Open (NO), Common (COM), and Normally Closed (NC). Basic configurations will connect the hot wire to Common and the other wire to either NO or NC. For DC connections, attach the anode (+) to the Common terminal.

NO provides an open circuit when the relay is off. NC provides a closed circuit when the relay is off, giving opposite on/off states from NO. The following diagram shows the terminal layout and imagines that Relay 1 is On.



 The CanaKit Relay requires a driver that is available at the supplier's website. At the time of this writing, the driver is unsigned, which forces you to turn off Driver Signing Enforcement when installing the driver in 64-bit Windows 8. For instructions on how to install unsigned drivers in this environment, see [How to Disable Driver Signature Verification on 64-Bit Windows 8.1](#).

After installing the CanaKit device driver, connect the unit to your PC. Windows dynamically assigns a serial port to the USB channel for the connection. You can see the port assignment by looking at the [Ports\(COM and LPT\)](#) entry in the Device Manager. The Device Manager is available from the System applet in the Control Panel.



For controlling power to devices using 3-prong outlets, Peach provides the [IpPower9258](#), [ApcPower](#), and [SnmpPower](#) monitors.

Parameters

Required:

SerialPort

Serial port for the board (such as COM2).

RelayNumber

Relay to trigger (1, 2, 3, or 4). Each relay number corresponds to a single relay in the kit, as shown in the first diagram.

Optional:

Action

Perform an action on the specified relay, defaults to ToggleOff. Valid actions include the following:

Action	Description
ToggleOff	Sets the relay to the OFF position, then sets the relay to the ON position.
ToggleOn	Sets the relay to the ON position, then sets the relay to the OFF position.
SetOn	Sets the relay to the ON position.
SetOff	Sets the relay to the OFF position.

StartOnCall

Toggle power when the specified call is received from the state model. This value is used only when the *When* parameter is set to [OnCall](#).

ToggleDelay

Pause in milliseconds between off/on, defaults to **500**. Formerly named OnOffPause.

When

Specify one of the following values to determine when a relay should be toggled:

"When" Setting	Description
DetectFault	Toggle power when checking for a fault. This occurs after OnIterationEnd.
OnStart	Toggle power when the fuzzing session starts. This occurs once per session.
OnEnd	Toggle power when the fuzzing session stops. This occurs once per session.
OnIterationStart	Toggle power at the start of each iteration.
OnIterationEnd	Toggle power at the end of each iteration.
OnFault	Toggle power when any monitor detects a fault. This is the default setting.
OnIterationStartAfterFault	Toggle power at the start of an iteration that immediately follows a fault detection.
OnCall	Toggle power when the call specified by the <i>StartOnCall</i> parameter is received from the state model.

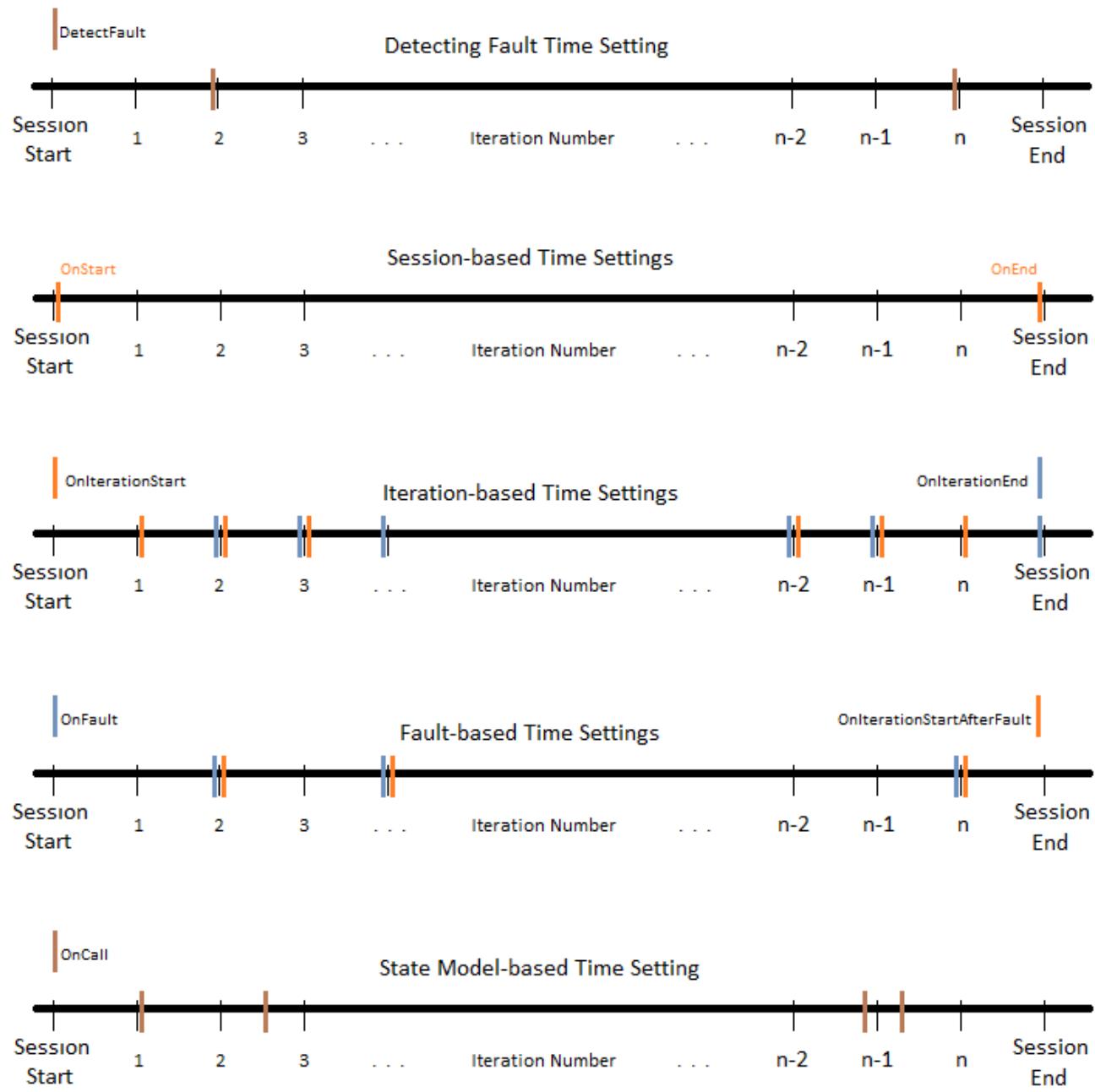


Figure 4. When Choices for Performing an Action

Examples

Reset power on Relay 1

This uses the CanaKitRelay monitor to reset relay 1, which toggles the power off, then back on. A device attached to this relay will restart when the relay resets. The default setting for the *When* parameter is **OnFault**, so the relay will be toggled after a fault is detected.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <Number size="32" signed="false" value="31337" />
    </DataModel>

    <StateModel name="State" initialState="Initial" >
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="TheAgent">
        <Monitor class="CanaKitRelay">
            <Param name="SerialPort" value="COM5" />
            <Param name="RelayNumber" value="1" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <StateModel ref="State"/>
        <Agent ref="TheAgent" />
        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>

```

21.9.6. CAN Capture Monitor

Monitor Category: Data Collection

The *CAN Capture* monitor collects all frames received (but not transmitted) on the specified interface. If a fault occurs, the capture is saved in the *PCAP* format, loadable into Wireshark for analysis.



Not all information from the CAN frame is visible in Wireshark. Basic flags and data are available, but not the full frame. This is a limitation of the format and not Peach Fuzzer.

Parameters

Required:

CanDriver

Driver to use. Defaults to *Vector XL*.

CanChannel

Channel number

CanBitrate

Set the bitrate for CAN packet transmission (default 500,000)

Optional:

None

21.9.7. CAN Error Frame Monitor

Monitor Category: Fault detection

The *CAN Error Frame* monitor is will trigger a fault when an error frame is received on the specified device/channel. Multiple *CAN Error Frame* monitors can be configured to monitor different devices/channel combinations.

Parameters

Required:

CanDriver

Driver to use. Defaults to *Vector XL*.

CanChannel

Channel number

CanBitrate

Set the bitrate for CAN packet reception (default 500,000)

Optional:

No optional parameters supported on this monitor.

Examples

21.9.8. CAN Send Frame Monitor

Monitor Category: Automation

The *CAN Send Frame* monitor is used to send CAN frames that are not fuzzed. Frames can be sent at specific points during a test case, every N milliseconds, or both. This can be useful to simulate messages required by a target.

The CAN frames being sent can share the same driver and channel as the fuzzing, or use a different driver/channel.



The DLC is auto set based on the data size provided.

Parameters

Required:

CanDriver

Driver to use. Defaults to *Vector XL*.

CanChannel

Channel number

CanBitrate

Set the bitrate for CAN packet transmission (default 500,000)

Id

CAN Frame ID field in hex.

Data

CAN data to transmit in hex.

Optional:

SendEvery

Send frame every N milliseconds. Disable by setting to zero. Defaults to 0.

When

Specify one of the following values to determine when a CAN frame should be sent (defaults to **OnStart**):

"When" Setting	Description
DetectFault	Send CAN frame when checking for a fault. This occurs after OnIterationEnd.

"When" Setting	Description
OnStart	Send CAN frame when the fuzzing session starts. This occurs once per session.
OnEnd	Send CAN frame when the fuzzing session stops. This occurs once per session.
OnIterationStart	Send CAN frame at the start of each iteration.
OnIterationEnd	Send CAN frame at the end of each iteration.
OnFault	Send CAN frame when any monitor detects a fault. This is the default setting.
OnIterationStartAfterFault	Send CAN frame at the start of an iteration that immediately follows a fault detection.
OnCall	Send CAN frame when the call specified by the <i>StartOnCall</i> parameter is received from the state model.

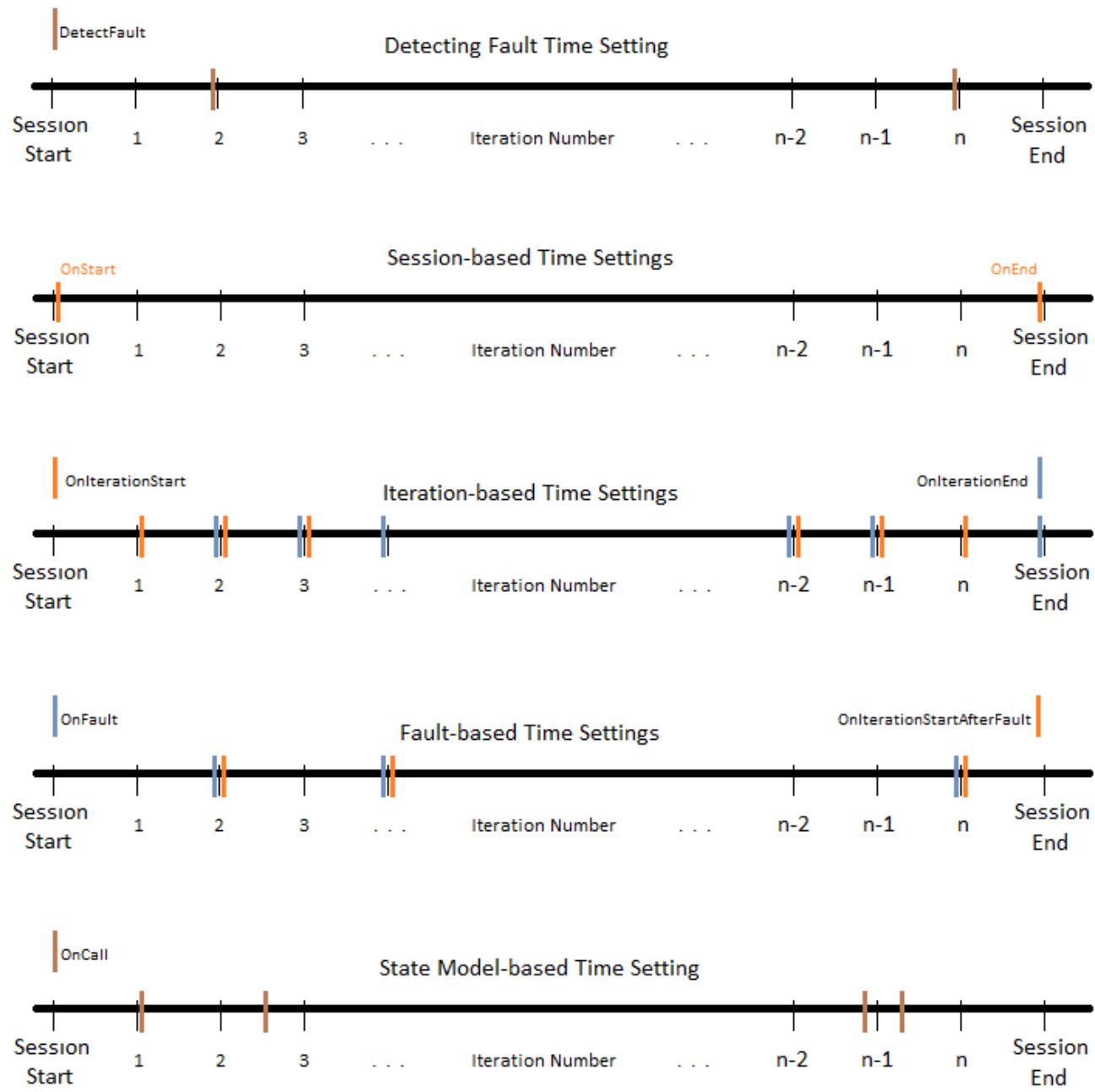


Figure 5. When Choices for Performing an Action

StartOnCall

Send frame when the specified event is received from the state model. This value is used only when the *When* parameter is set to **OnCall**.

Examples

Example 203. Send CAN Frame Every 500 ms

This parameter example is from a setup that uses the CAN Send Frame monitor to send a CAN frame every 500 ms.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach>

    <DataModel name="CanId" mutable="false">
        <Number name="Value" size="29" signed="false"
            endian="big" mutable="false" />
    </DataModel>

    <DataModel name="Standard_Message_1_3">
        <Number name="Signal_8_VtSig" size="8" signed="false" value="170" />
    </DataModel>

    <StateModel name="TheStateManager" initialState="SendCanFrames">
        <State name="SendCanFrames">
            <Action name="SendCanFrame_3" type="call" method="Send">
                <Param name="Id">
                    <DataModel ref="CanId" />
                    <Data>
                        <Field name="Value" value="0x1"/>
                    </Data>
                </Param>
                <Param name="Data">
                    <DataModel ref="Standard_Message_1_3" />
                </Param>
            </Action>
        </State>
    </StateModel>

    <Agent name="Local">
        <Monitor class="CanSendFrame">
            <Param name="CanDriver" value="Vector XL"/>
            <Param name="CanChannel" value="1"/>
            <Param name="CanBitrate" value="500000"/>
            <Param name="Id" value="0x07DC"/>
            <Param name="Data" value="AA BB CC DD EE FF"/>
            <Param name="SendEvery" value="500"/>
        </Monitor>
    </Agent>

    <Test name="Default" maxOutputSize="64">
        <Agent ref="Local"/>
        <StateModel ref="TheStateManager" />
        <Publisher class="Can">
            <Param name="Driver" value="Vector XL" />
            <Param name="Channel" value="1" />
            <Param name="Bitrate" value="500000" />
        </Publisher>
    </Test>

```

```
</Peach>
```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 61495.
Peach.Pro.Core.Loggers.JobLogger Writing debug.log to: c:\peach-
pro\output\win_x64_debug\bin\Logs\CanSendFrame_Example.xml_20180521162530\debug.log
Peach.Core.Agent.Channels.AgentLocal StartMonitor: Monitor CanSendFrame
Peach.Pro.Core.Publishers.Can.VectorCanDriver VectorCanDriver.Initialize: Vector XL
-Receive loop started
Peach.Core.Agent.Channels.AgentLocal SessionStarting
Peach.Pro.Core.Agent.Monitors.CanSendFrame Starting timer to send CAN frame every 500
ms
Peach.Pro.Core.Agent.Monitors.CanSendFrame Sending frame id '0x07DC' of 'AA BB CC DD
EE FF'
Peach.Core.Engine runTest: Iteration Starting: 1, =====

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing control recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "SendCanFrames".
Peach.Core.Dom.Action Run(SendCanFrame_3): Call
Peach.Pro.Core.Publishers.CanPublisher start()
Peach.Pro.Core.Publishers.CanPublisher call(Send) ActionParameter Count: 2
Peach.Pro.Core.Publishers.CanPublisher call(Send) BitwiseStream Count: 2
Peach.Pro.Core.Publishers.CanPublisher open()
Peach.Pro.Core.Publishers.CanPublisher close()
Peach.Core.Agent.AgentManager DetectedFault: checking for fault in agent Local
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Engine All test cases executed, stopping engine.
Peach.Pro.Core.Publishers.CanPublisher stop()
Peach.Core.Engine EndTest: Stopping all agents and monitors

[*] Test 'Default' finished.
```

21.9.9. CAN Timing Monitor

Monitor Category: Fault detection

The *CAN Timing* monitor is will trigger a fault when a frame is not received within a specified time window. Many CAN targets are designed to send one or more CAN frames every N milliseconds. When transmission of these frames stops or is out of spec, this can indicate a failure in the target device.

The CAN frames being sent can share the same driver and channel as the fuzzing, or use a different driver/channel.

Parameters

Required:

CanDriver

Driver to use. Defaults to *Vector XL*.

CanChannel

Channel number

CanBitrate

Set the bitrate for CAN packet reception (default 500,000)

Id

CAN Frame ID field in hex to expect

Window

Reception window (how often frame should be received) in milliseconds. If frame with **Id** is not received in this window of time, a fault will be raised.

Optional:

Examples

Example 204. Expect frame every half second

This parameter example is from a setup that uses the CAN Timing Monitor to fault if a specific CAN frame is not received every half second.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach>

<DataModel name="CanId" mutable="false">
    <Number name="Value" size="29" signed="false"
        endian="big" mutable="false" />
```

```

</DataModel>

<DataModel name="Standard_Message_1_3">
    <Number name="Signal_8_VtSig" size="8" signed="false" value="170" />
</DataModel>

<StateModel name="TheStateModel" initialState="SendCanFrames">
    <State name="SendCanFrames">
        <Action name="SendCanFrame_3" type="call" method="Send">
            <Param name="Id">
                <DataModel ref="CanId" />
            <Data>
                <Field name="Value" value="0x1"/>
            </Data>
        </Param>
        <Param name="Data">
            <DataModel ref="Standard_Message_1_3" />
        </Param>
        </Action>
    </State>
</StateModel>

<Agent name="Local">
    <Monitor class="CanTiming">
        <Param name="CanDriver" value="Vector XL"/>
        <Param name="CanChannel" value="1"/>
        <Param name="CanBitrate" value="500000"/>
        <Param name="Id" value="0x07DC"/>
        <Param name="Window" value="500"/>
    </Monitor>
</Agent>

<Test name="Default" maxOutputSize="64">
    <Agent ref="Local"/>
    <StateModel ref="TheStateModel" />
    <Publisher class="Can">
        <Param name="Driver" value="Vector XL" />
        <Param name="Channel" value="1" />
        <Param name="Bitrate" value="500000" />
    </Publisher>
</Test>
</Peach>

```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 25814.
Peach.Pro.Core.Loggers.JobLogger Writing debug.log to: peach-
pro\Logs\CanTiming_Example.xml_20180522150641\debug.log
Peach.Core.Agent.Channels.AgentLocal StartMonitor: Monitor CanTiming
Peach.Pro.Core.Publishers.Can.VectorCanDriver VectorCanDriver.Initialize: Vector XL
-Receive loop started
Peach.Core.Agent.Channels.AgentLocal SessionStarting
Peach.Core.Engine runTest: Iteration Starting: 1, =====

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing control recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "SendCanFrames".
Peach.Core.Dom.Action Run(SendCanFrame_3): Call
Peach.Pro.Core.Publishers.CanPublisher start()
Peach.Pro.Core.Publishers.CanPublisher call(Send) ActionParameter Count: 2
Peach.Pro.Core.Publishers.CanPublisher call(Send) BitwiseStream Count: 2
Peach.Pro.Core.Publishers.CanPublisher open()
Peach.Pro.Core.Publishers.CanPublisher close()
Peach.Core.Agent.AgentManager DetectedFault: checking for fault in agent Local
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Engine All test cases executed, stopping engine.
Peach.Pro.Core.Publishers.CanPublisher stop()
Peach.Core.Engine EndTest: Stopping all agents and monitors

[*] Test 'Default' finished.
```

21.9.10. CAN Threshold Monitor

Monitor Category: Fault detection

The *CAN Threshold* monitor triggers a fault when a CAN signal is outside a specified threshold. The threshold is provided as a python expression that must evaluate to bool true or false.

This is one of several ways to monitor a CAN target during fuzzing to determine if testing has adversely affected the target.

The CAN frames being sent can share the same driver and channel as the fuzzing, or use a different driver/channel.

Parameters

Required:

CanDriver

Driver to use. Defaults to *Vector XL*.

CanChannel

Channel number

CanBitrate

Set the bitrate for CAN packet reception (default 500,000)

Id

CAN Frame ID field in hex to expect

SignalEndian

Endianness of signal (if needed). Defaults to **little**. Options are **little** or **big**.

SignalOffset

Bit offset to signal start

SignalSize

Length of signal field in bits

SignalType

Data type of signal. This is used to convert signal data into a usable type in the python threshold expression.

Signal Types	
Int	SignalSize can be any value 32 or lower
Float	SignalSize must be 32 or 64

Signal Types	
Long	SignalSize between 33 and 64
String	Interpreted as UTF-8
Binary	SignalSize must be factor of 8

Expression

Stateless threshold expression used to determin if the signal value is within spec. Expression results are cached for speed, and must not be statefull. The threshold expression is specified as a Python 2 expression. The provided expression must evaluate to a bool true/false. Python expressions are single line code statements. The following is an example of a code snippet that verifies a signal value is between 10 and 20 inclusive: `signal >= 10 and signal <= 20`

The Python expressions have access to the following local variables:

Local Variables	Description
id	CAN Frame ID as a python int
logger	Logging interface output stored in debug.log in run folder and test output. logger.Debug(msg)
signal	Signal value decoded based on provided type

Optional:

No optional parameters.

Examples

Example 205. Expect frame every half second

This parameter example is from a setup that uses the CAN Threshold Monitor to fault if a signal in the CAN frame with id 0x7DE is not between 10 and 100.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach>

    <DataModel name="CanId" mutable="false">
        <Number name="Value" size="29" signed="false"
            endian="big" mutable="false" />
    </DataModel>

    <DataModel name="Standard_Message_1_3">
        <Number name="Signal_8_VtSig" size="8" signed="false" value="170" />
    </DataModel>
```

```

<StateModel name="TheStateModel" initialState="SendCanFrames">
    <State name="SendCanFrames">
        <Action name="SendCanFrame_3" type="call" method="Send">
            <Param name="Id">
                <DataModel ref="CanId" />
                <Data>
                    <Field name="Value" value="0x1"/>
                </Data>
            </Param>
            <Param name="Data">
                <DataModel ref="Standard_Message_1_3" />
            </Param>
        </Action>
    </State>
</StateModel>

<Agent name="Local">
    <Monitor class="CanTiming">
        <Param name="CanDriver" value="Vector XL"/>
        <Param name="CanChannel" value="1"/>
        <Param name="CanBitrate" value="500000"/>
        <Param name="Id" value="0x07DC"/>
        <Param name="Window" value="500"/>
    </Monitor>
</Agent>

<Test name="Default" maxOutputSize="64">
    <Agent ref="Local"/>
    <StateModel ref="TheStateModel" />
    <Publisher class="Can">
        <Param name="Driver" value="Vector XL" />
        <Param name="Channel" value="1" />
        <Param name="Bitrate" value="500000" />
    </Publisher>
</Test>
</Peach>

```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 25814.
Peach.Pro.Core.Loggers.JobLogger Writing debug.log to: peach-
pro\Logs\CanTiming_Example.xml_20180522150641\debug.log
Peach.Core.Agent.Channels.AgentLocal StartMonitor: Monitor CanTiming
Peach.Pro.Core.Publishers.Can.VectorCanDriver VectorCanDriver.Initialize: Vector XL
-Receive loop started
Peach.Core.Agent.Channels.AgentLocal SessionStarting
Peach.Core.Engine runTest: Iteration Starting: 1, =====

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing control recording iteration.
Peach.Core.Dom.StateModel Run(): Changing to state "SendCanFrames".
Peach.Core.Dom.Action Run(SendCanFrame_3): Call
Peach.Pro.Core.Publishers.CanPublisher start()
Peach.Pro.Core.Publishers.CanPublisher call(Send) ActionParameter Count: 2
Peach.Pro.Core.Publishers.CanPublisher call(Send) BitwiseStream Count: 2
Peach.Pro.Core.Publishers.CanPublisher open()
Peach.Pro.Core.Publishers.CanPublisher close()
Peach.Core.Agent.AgentManager DetectedFault: checking for fault in agent Local
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Engine All test cases executed, stopping engine.
Peach.Pro.Core.Publishers.CanPublisher stop()
Peach.Core.Engine EndTest: Stopping all agents and monitors

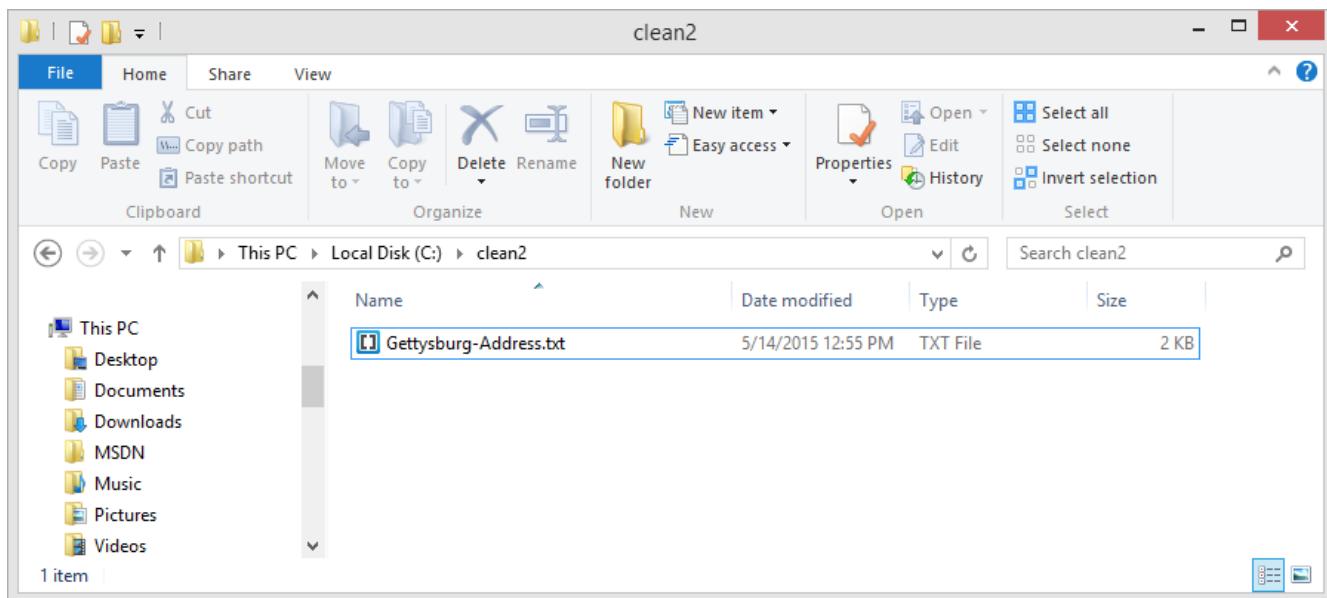
[*] Test 'Default' finished.
```

21.9.11. CleanupFolder Monitor

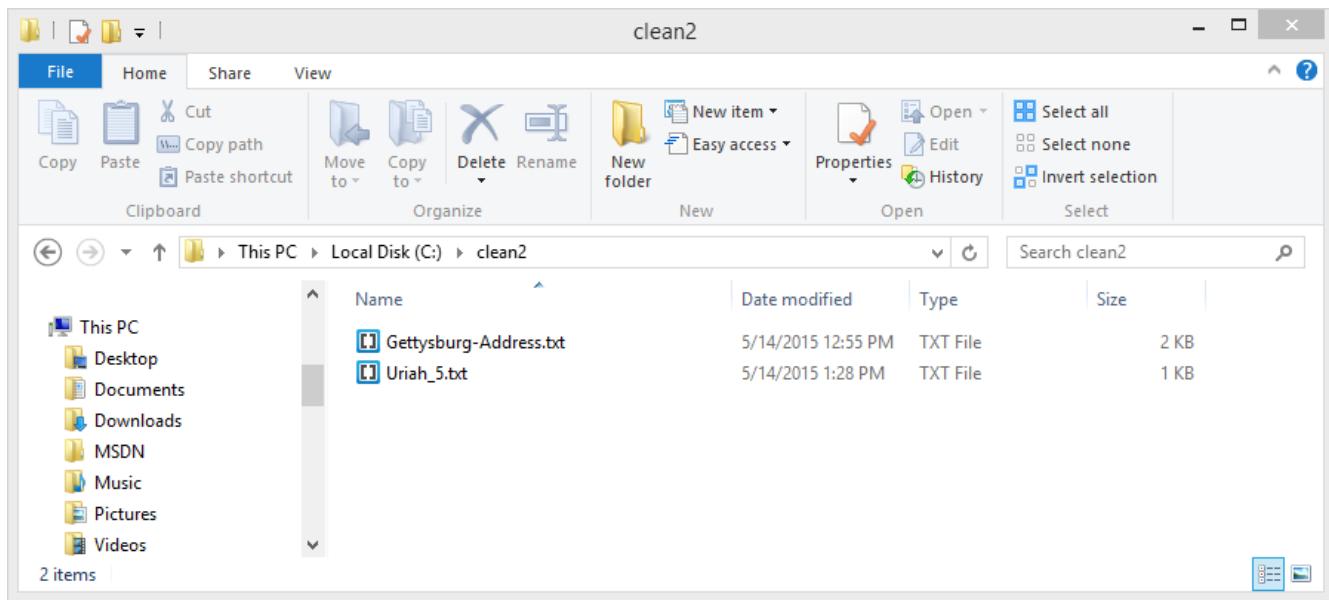
Monitor Category: Automation

The *CleanupFolder* monitor provides automated cleanup of files created during a fuzzing session. This monitor purges files produced during an iteration before the start of the following iteration. *CleanupFolder* acts upon a specified directory, and does not affect any files or directories that predate the fuzzing session.

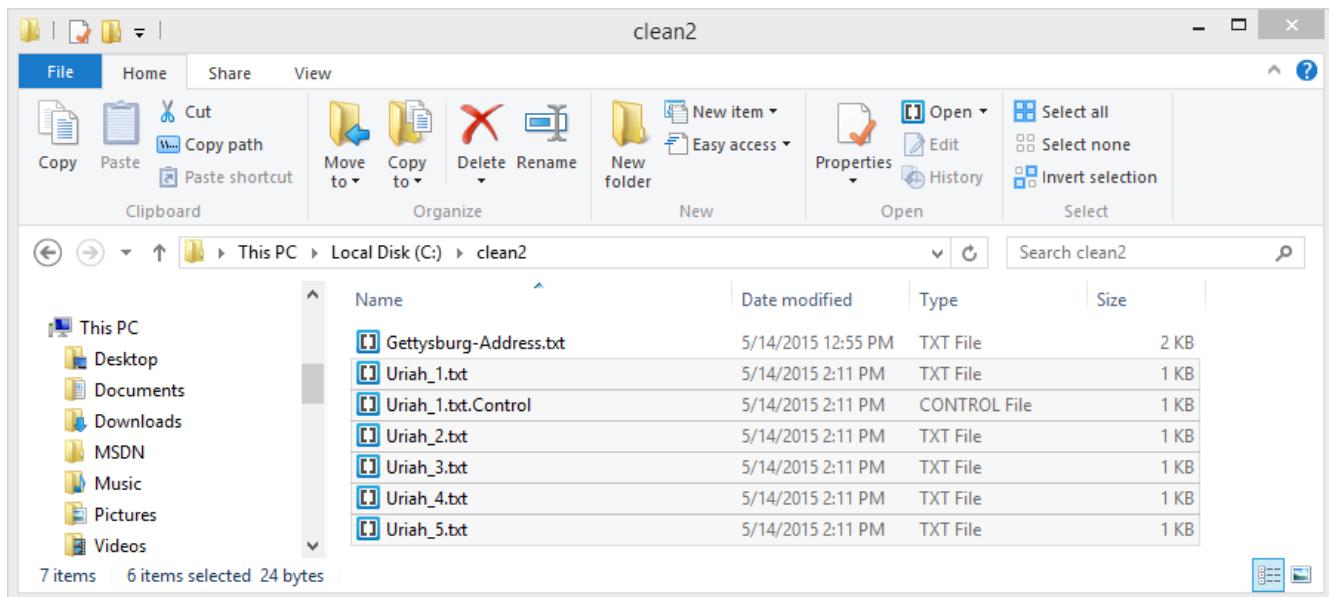
1. The directory, `clean2`, initially contains a single file.



2. Before each iteration, the *CleanupFolder* monitor deletes from the specified folder all files that were generated in the previous fuzzing iteration.
3. During each iteration, the File publisher creates a fuzzed file.
4. The following image shows the output from a file fuzzing session that uses *CleanupFolder* on the `clean2` directory. The session duration is 5 iterations.



5. The following image shows the `clean2` directory with output from a file fuzzing session that does *not* use `CleanupFolder`, but is otherwise identical to the pit used for the previous diagram. Files generated in the fuzzing session are highlighted.



The `FilePerIteration` publisher, that generates a new filename for each fuzzed file, produced the fuzzed output files for both of the previous images.

Parameters

Required:

Folder

Folder to clean.

Optional:

None.

Examples

Example 206. Remove contents of a folder

Run this example from the folder specified using the *folder* parameter.

During each iteration, the file `fuzzed.txt` is created by the File publisher. At the end of each iteration, `fuzzed.txt` is deleted by the CleanupFolder monitor.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <Number size="32" signed="false" value="31337" />
    </DataModel>

    <StateModel name="State" initialState="Initial" >
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>

            <Action type="output" publisher="FilePub">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="TheAgent">
        <Monitor class="CleanupFolder">
            <Param name="Folder" value="C:\\\\cleanme" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <StateModel ref="State"/>
        <Agent ref="TheAgent" />
        <Publisher class="ConsoleHex"/>
        <Publisher name="FilePub" class="File">
            <Param name="FileName" value="fuzzed.txt" />
        </Publisher>
    </Test>
</Peach>

```

Output from this example.

```
>peach -l --debug example.xml

[*] Test 'Default' starting with random seed 43073.
Peach.Core.Agent.Agent StartMonitor: Monitor CleanupFolder
Peach.Core.Agent.SessionStarting: Monitor

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(4 bytes)
00000000  69 7A 00 00                                iz??
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()
Peach.Core.Agent.Agent SessionFinished: Monitor

[*] Test 'Default' finished.
```

21.9.12. CleanupRegistry Monitor (Windows)

Monitor Category: Automation

The *CleanupRegistry* monitor provides automated cleanup of Windows registry entries created during a fuzzing session. Cleanup occurs before every iteration.

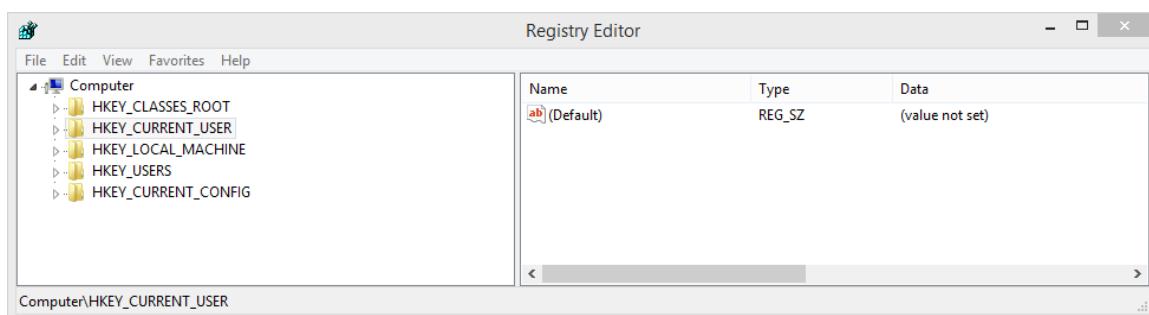
This monitor uses a specified registry key (a parent key) as a reference point and deletes all child keys or descendants of the parent key. All child keys at all levels beneath the parent key are removed. Values attached to child keys are removed as well.

Optional, *CleanupRegistry* can include or exclude the parent key and values attached to the parent key in its purge.

The Windows Registry is a hierarchical database that stores information about the computer system, configuration, applications, current user settings, and performance data. The contents and use of the registry has evolved since its inception. Originally, the registry stored installation and initialization settings for applications. On current versions of Windows, the registry contains information for the current user, and much more, such as a list of external serial ports that the machine has allocated.

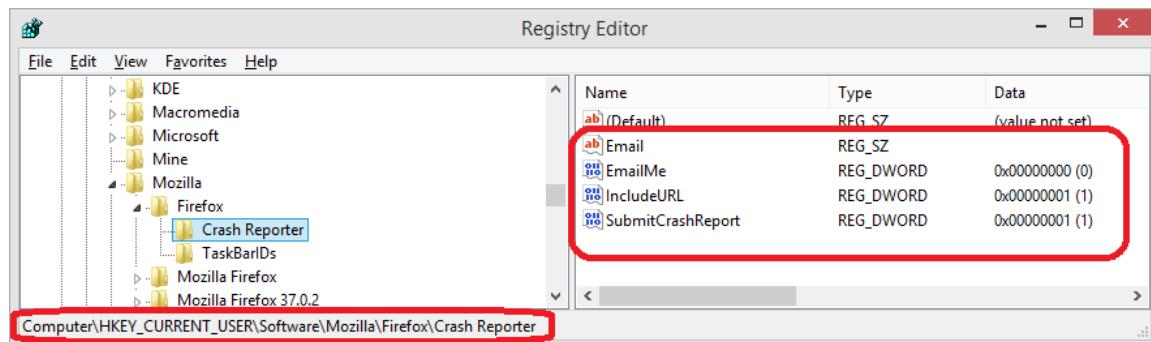
One common use of the registry is to store a list of the most recently used files used in an application. This list is typically displayed to a user to improve the experience of selecting a file for the application to open.

The Windows registry consists of five hierarchies called hives, as shown in the following illustration. Each hive consists of keys and values.



- A key is a container that can contain other keys and values. A key has a name, a type, and optionally, a data value. The following illustration shows a selected key. The complete name for this key is reported in the lower left corner.
- A value is an item associated with a key that has a type, a data value, and optionally, a name. In the following illustration, the named values defined for the selected key are listed in the right side of the window.





Each hive has one root key that is the entry point to the hive. These root keys are the prefixes described for the *CleanupRegistry Key* parameter. The hives used most often are **HKCU** and **HKLM**, but keys from other hives are accessible as well.

Locating a key in the registry is similar to locating a folder on disk. Start with the root key and specify the sequence of keys you need to reach the desired key. Separate keys with the back slash character "\", just like the folders in a path.

A root key, such as **HKCU**, has child keys; that is, the keys that it contains. A non-root key, such as **HKCU\Software**, has a parent key (**HKCU**) and, optionally, child keys. Note that **HKCU** is the parent key of **Software**. The key **HKCU\Software\Mozilla\FireFox** also has two child keys or children (**Crash Reporter** and **TaskBarIDs**).

For more information, see [Windows Registry](#). Note the rich bibliography, which can serve as a foundation for additional reading. Also, the Microsoft Developer Network (MSDN) has several articles on the registry.

In *CleanupRegistry*, you do not have control over individual values. You simply identify the parent key, and specify whether you need to zap the parent key; then let the monitor clean up around that registry key after each iteration.



Sometimes, applications keep lists of recently used files in the registry, typically in the **HKCU** or **HKLM** hives. If you receive an error about not being able to create a key in the registry, you might need to clear the children of a specified key.

Parameters

Required:

Key

Registry key to remove. The following **key prefixes** are used:

HKCU - Current user

HKCC - Current configuration

HKLM - Local machine

HKPD - Performance data

HKU - Users

Optional:

ChildrenOnly

If true, omits the parent key from the purge, thereby deleting all descendants (sub-keys) of the specified key.

If false, the parent key and all descendants are deleted. Defaults to false.

Examples

Cleanup for office

This uses the CleanupRegistry monitor to remove a specified key from the Windows Registry at the beginning of a fuzzing session as part of the control iteration.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <Number size="32" signed="false" value="31337" />
    </DataModel>

    <StateModel name="State" initialState="Initial" >
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="TheAgent">
        <Monitor class="CleanupRegistry">
            <Param name="Key" value="HKLM\SOFTWARE\Office13\Recovery" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <StateModel ref="State"/>
        <Agent ref="TheAgent" />
        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>
```

Output for this example.

```
>peach -l --debug example.xml

[*] Test 'Default' starting with random seed 28078.
Peach.Core.Agent.Agent StartMonitor: Monitor CleanupRegistry
Peach.Core.Agent.SessionStarting: Monitor

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.OS.Windows.Agent.Monitors.CleanupRegistry Removing key:
SOFTWARE\Office13\Recovery ①
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(4 bytes)
00000000  69 7A 00 00                                iz??
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()
Peach.Core.Agent.Agent SessionFinished: Monitor

[*] Test 'Default' finished.
```

① Deleting the registry key

21.9.13. CrashReporter Monitor (OS X)

Monitor categories: Data collection, Fault detection

The *CrashReporter* monitor collects and logs core dump information from crashes detected by the OS X System Crash Reporter. Use *CrashReporter* when crashes can occur and you cannot use [CrashWrangler](#).

The monitoring scope can focus on a single executable, specified by setting the **ProcessName** parameter. Or, the monitoring scope can include all processes. By default, all processes are monitored.

Before fuzzing, make sure to disable the Apple crash report dialog window. When the fuzzing session ends, reenable the crash report dialog window. Commands to disable and enable the crash report dialog window follow:

- Disable the crash report dialog:

```
defaults write com.apple.CrashReporter DialogType none
```

- Enable the crash report dialog after fuzzing:

```
defaults write com.apple.CrashReporter DialogType crashreport
```

Parameters

Required:

None.

Optional:

ProcessName

Name of the process to watch (optional, defaults to all)

Examples

Example 207. Catch all crashes

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="TheDataModel">
    <Number size="32" signed="false" value="31337" />
</DataModel>

<StateModel name="State" initialState="Initial" >
    <State name="Initial">
        <Action type="output">
            <DataModel ref="TheDataModel"/>
        </Action>
    </State>
</StateModel>

<Agent name="TheAgent">
    <Monitor class="CrashReporter"/>
</Agent>

<Test name="Default">
    <StateModel ref="State"/>
    <Agent ref="TheAgent" />
    <Publisher class="ConsoleHex"/>
</Test>
</Peach>
```

21.9.14. CrashWrangler Monitor (OS X)

Monitor Categories: Automation, Data collection, Fault detection



The gdb debugger and [gdb](#) monitor are the preferred tools to detect crashes and to collect core files on OS X.

The *CrashWrangler* monitor launches a process attached to the CrashWrangler debugger and monitors the process for crashes and other faults. This monitor runs only on OS X systems. Use this monitor when gdb is not an option, such as when anti-debugging mechanisms are used on the test target.

CrashWrangler monitor detects crashes, exceptions, access violations, and faults. This monitor can generate faults for an application that exits early or that fails to exit.

After detecting a fault, the *CrashWrangler* monitor collects a stack trace, data from the device log files, and crash dumps. Additionally, the monitor logs exceptions, and updates fault bucket information. For bucketing, Peach uses the text from the fault to determine the major bucket level. The minor bucket level is not used. The risk evaluation provides the following levels: exploitable, not exploitable, or unknown.

This monitor uses Apple's Crash Wrangler tool that can be downloaded from the developer website. Crash Wrangler must be compiled on each machine it is used.



When using more than one instance of CrashWrangler in the same fuzzing session, assign unique names for the CwLockFile, CwLogFile, and CwPidFile files used with each instance of CrashWrangler. This practice will avoid contention issues involving these files.

Parameters

Required:

Executable

Command or application to launch. This parameter name is preferred over Command.

Command

Command to execute. Alias with Executable.



The Command parameter is supported, but is being deprecated. Instead, use the Executable parameter.

Optional:

Arguments

Command line arguments for the application that CrashWrangler launches, defaults to none.

CwLockFile

CrashWrangler Lock file, defaults to `cw.lock`.

CwLogFile

CrashWrangler Log file, defaults to `cw.log`.

CsPidFile

CrashWrangler PID file, defaults to `cw.pid`.

ExecHandler

Crash Wrangler execution handler program, defaults to `exc_handler`.

ExploitableReads

Are read a/v's considered exploitable? Defaults to false.

FaultOnEarlyExit

Trigger a fault if the process exits prematurely, defaults to false.

NoCpuKill

Disable process killing by CPU usage, Defaults to false.

RestartAfterFault

If "true", restarts the target when any monitor detects a fault. If "false", restarts the target only if the process exits or crashes.

This argument defaults to true.

RestartOnEachTest

Restarts the process for each iteration, defaults to false.

StartOnCall

Start the executable on a state model call, defaults to none.

UseDebugMalloc

Use the OS X Debug Malloc (slower), defaults to false.

WaitForExitOnCall

Wait for the process to exit on a state model call and fault if a timeout occurs, defaults to none.

WaitForExitTimeout

WaitForExitOnCall timeout value, expressed in milliseconds. -1 is infinite, defaults to 10000.

Examples

Example 208. Fuzzing Safari

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <Number size="32" signed="false" value="31337" />
    </DataModel>

    <StateModel name="State" initialState="Initial" >
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
            <Action type="close"/>
            <Action type="call" method="ScoobySnacks" publisher="Peach.Agent"/>
        </State>
    </StateModel>

    <Agent name="Local">
        <Monitor class="CrashWrangler">
            <Param name="Executable" value=
"/Applications/Safari.app/Contents/MacOS/Safari" />
            <Param name="Arguments" value="fuzzed.bin" />

            <Param name="UseDebugMalloc" value="true" />
            <Param name="ExploitableReads" value="true" />
            <Param name="ExecHandler" value=".exc_handler" />

            <Param name="StartOnCall" value="ScoobySnacks" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <StateModel ref="State"/>
        <Agent ref="TheAgent" />

        <Publisher class="File">
            <Param name="FileName" value="fuzzed.bin"/>
        </Publisher>
    </Test>
</Peach>
```

21.9.15. Gdb Monitor (Linux, OS X)

Monitor Categories: Automation, Data collection, Fault detection

The *Gdb* monitor uses GDB to launch an executable. It then monitors the executing process for specific signals/exceptions. *Gdb* is the main debugger for Linux and OS X operating systems.

When a configured signal/exception is handled, *Gdb* collects and logs information about the crash including frame information, registers and backtrace.

Gdb supports bucketing of crashes and basic risk ranking that is based on the exploitability of the fault. Bucketing uses categories for major and minor issues. The exploitability evaluation is similar to the !exploitable debugging extension for .NET.

For GDB Server support see the [GdbServer](#) monitor.



The *Gdb* monitor requires a recent version of GDB.

Parameters

Required:

Executable

Executable to launch. This should be just the executable. Command line arguments can be provided using the optional *Arguments* parameter.

Optional:

Arguments

Command line arguments for the executable.

FaultOnEarlyExit

Trigger a fault if the process exits prematurely, defaults to `false`.

GdbPath

Path to `gdb`, defaults to `/usr/bin/gdb`.

HandleSignals

Signals to consider faults. Space separated list of signals/exceptions to handle as faults. Defaults to: SIGSEGV SIGFPE SIGABRT SIGILL SIGPIPE SIGBUS SIGSYS SIGXCPU SIGXFSZ EXC_BAD_ACCESS EXC_BAD_INSTRUCTION EXC_ARITHMETIC

NoCpuKill

Disable process killing when the CPU usage nears zero, defaults to `false`.

RestartAfterFault

If `true`, restarts the target when any monitor detects a fault. If `false`, restarts the target only if the process exits or crashes. This argument defaults to `true`.

RestartOnEachTest

Restarts the process for each iteration, defaults to `false`.

Script

Script file used to drive GDB and perform crash analysis. This script sets up GDB to run the target application, report back information such as child pid, and handle any signals. This script also performs crash analysis and bucketing. An example script is provided in the examples section.

The script is a Mushtache style template with the following available parameters:

executable

Maps to the Executable parameter

arguments

Maps to the Arguments parameter

exploitableScript

Default crash analysis and bucketing script

faultOnEarlyExit

Maps to the FaultOnEarlyExit parameter

gdbLog

Log file used to record information that will be logged with any fault

gdbCmd

This script file after being processed by Mushtache

gdbPath

Maps to the GdbPath parameter

gdbPid

Pid file generated by script at start of executable

gdbTempDir

Temporary directory to hold generated data

handleSignals

Maps to the HandleSignals parameter

noCpuKill

Maps to the NoCpuKill parameter

restartOnEachTest

Maps to the RestartOnEachTest parameter

restartAfterFault

Maps to the RestartAfterFault parameter

startOnCall

Maps to the StartOnCall parameter

waitForExitOnCall

Maps to the WaitForExitOnCall parameter

waitForExitTimeout

Maps to the WaitForExitTimeout parameter

StartOnCall

Start the executable on a state model call.

WaitForExitOnCall

Wait for the process to exit on a state model call and fault if a timeout is reached.

WaitForExitTimeout

WaitForExitOnCall timeout value, expressed in milliseconds. **-1** is infinite, defaults to **10000**.

Gdb Installation

Installing **gdb** on Linux consists of using the package manager to download and install the **gdb** debugger and the needed support files and libraries.

- For Ubuntu systems, run the following command install **gdb**:

```
sudo apt-get install gdb
```

- For CentOS and RedHat Enterprise Linux installations, run the following command to install **gdb**:

```
sudo yum install gdb
```

- For SUSE Enterprise Linux installations, run the following command to install **gdb**:

```
sudo zypper install gdb
```

You can also install **gdb** on OS X using a package manager. See the [Installing GDB on OS X Mavericks](#)

article for more information.

Examples

Example 209. Script Example

This is an example *Script* for driving GDB.

The *Gdb* monitor expects specific output that the included exploit analysis to generate the fault title, risk and bucket hashes.

The following regular expressions are used to extract information from log. If replacing the crash analysis portion of the script make sure to output to allow the first three regexes to pass.

```
"^Hash: (\w+)\.(\w+)$" -- Major and Minor bucket
"^Exploitability Classification: (.*)$" -- Risk
"^Short description: (.*)$" -- Title
"^Other tags: (.*)$" -- If found, added to Title
```

Script:

```

define log_if_crash
if ($_thread != 0x00)
printf "Crash detected, running exploitable.\n"
set logging overwrite on
set logging redirect on
set logging on {{gdbLog}}          ②
exploitable -v                   ③
printf "\n--- Info Frame ---\n\n"
info frame
printf "\n--- Info Registers ---\n\n"
info registers
printf "\n--- Backtrace ---\n\n"
thread apply all bt full
set logging off
end
end

handle all nostop noprint
handle {{handleSignals}} stop print

file {{executable}}
set args {{arguments}}
source {{exploitableScript}}


python
def on_start(evt):
    import tempfile, os
    h,tmp = tempfile.mkstemp()
    os.close(h)
    with open(tmp, 'w') as f:
        f.write(str(gdb.inferiors()[0].pid))
    os.rename(tmp, '{{gdbPid}}')      ①
    gdb.events.cont.disconnect(on_start)
gdb.events.cont.connect(on_start)
end

printf "starting inferior: '{{executable}} {{arguments}}'\n"

run
log_if_crash
quit

```

① Must generate *gdbPid* when target has been started

② Must generate *gdbLog* when a fault has been handled

③ Generates output compatible with regexes

Example 210. Basic Usage Example

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="TheDataModel">
    <Number size="32" signed="false" value="31337" />
</DataModel>

<StateModel name="State" initialState="Initial" >
    <State name="Initial">
        <Action type="call" method="ScoobySnacks" publisher="Peach.Agent"/>

        <Action type="accept"/>
        <Action type="input">
            <DataModel ref="TheDataModel"/>
        </Action>

        <Action type="output">
            <DataModel ref="TheDataModel"/>
        </Action>
    </State>
</StateModel>

<Agent name="Local">
    <Monitor class="Gdb">
        <Param name="Executable" value="/usr/bin/curl"/>
        <Param name="Arguments" value="http://localhost"/>
        <Param name="StartOnCall" value="ScoobySnacks"/>
    </Monitor>
</Agent>

<Test name="Default">
    <StateModel ref="State"/>
    <Agent ref="Local" />

    <Publisher class="TcpListener">
        <Param name="Interface" value="127.0.0.1" />
        <Param name="Port" value="80"/>
    </Publisher></Test>
</Peach>
```

21.9.16. GdbServer Monitor (Linux, OS X)

Monitor Categories: Automation, Data collection, Fault detection

The *GdbServer* monitor uses GDB to access a remote debugger that implements the gdb server protocol. This can be `gdbserver` or any other implementation. This monitor also supports the multi mode of the extended `gdbserver` protocol via the `RemoteExecutable` parameter.

When a configured signal/exception is handled, *GdbServer* collects and logs information about the crash including frame information, registers and backtrace.

GdbServer supports bucketing of crashes and basic risk ranking that is based on the exploitability of the fault. Bucketing uses categories for major and minor issues. The exploitability evaluation is similar to the !exploitable debugging extension for .NET.

For normal GDB support see the [Gdb](#) monitor.



The *GdbServer* monitor requires a recent version of GDB.

Parameters

Required:

Target

GDB target command arguments. Example: `remote 192.168.1.2:6000`.

When possible use the extended-remote version: `extended-remote IP:PORT`.

LocalExecutable

Local copy of executable remote target is running.

Optional:

RemoteExecutable

Enables multi-process capabilities in the extended gdb server protocol. Provides the remote executable name to load and run. This is the remote exec file-name.

FaultOnEarlyExit

Trigger a fault if the process exits prematurely, defaults to `false`.

GdbPath

Path to `gdb`, defaults to `/usr/bin/gdb`.

HandleSignals

Signals to consider faults. Space separated list of signals/exceptions to handle as faults. Defaults to: SIGSEGV SIGFPE SIGABRT SIGILL SIGPIPE SIGBUS SIGSYS SIGXCPU SIGXFSZ EXC_BAD_ACCESS

EXC_BAD_INSTRUCTION EXC_ARITHMETIC

NoCpuKill

Disable process killing when the CPU usage nears zero, defaults to `false`.

RestartAfterFault

If `true`, restarts the debugger when any monitor detects a fault. If `false`, restarts the debugger only if the process exits or crashes. This argument defaults to `true`.

RestartOnEachTest

Restarts the process for each iteration, defaults to `false`.

Script

Script file used to drive GDB and perform crash analysis. This script sets up GDB to run the target application, report back information such as child pid, and handle any signals. This script also performs crash analysis and bucketing. An example script is provided in the examples section.

The script is a Mushtache style template with the following available parameters:

exploitableScript

Default crash analysis and bucketing script

faultOnEarlyExit

Maps to the FaultOnEarlyExit parameter

gdbLog

Log file used to record information that will be logged with any fault

gdbCmd

This script file after being processed by Mushtache

gdbPath

Maps to the GdbPath parameter

gdbPid

Pid file generated by script at start of executable

gdbTempDir

Temporary directory to hold generated data

handleSignals

Maps to the HandleSignals parameter

localExecutable

Maps to the LocalExecutableParameter

noCpuKill

Maps to the NoCpuKill parameter

remoteExecutable

Maps to the RemoteExecutable parameter

restartOnEachTest

Maps to the RestartOnEachTest parameter

restartAfterFault

Maps to the RestartAfterFault parameter

startOnCall

Maps to the StartOnCall parameter

target

Maps to the Target parameter

waitForExitOnCall

Maps to the WaitForExitOnCall parameter

waitForExitTimeout

Maps to the WaitForExitTimeout parameter

StartOnCall

Start the executable on a state model call.

WaitForExitOnCall

Wait for the process to exit on a state model call and fault if a timeout is reached.

WaitForExitTimeout

WaitForExitOnCall timeout value, expressed in milliseconds. -1 is infinite, defaults to 10000.

Gdb Installation

Installing `gdb` on Linux consists of using the package manager to download and install the `gdb` debugger and the needed support files and libraries.

- For Ubuntu systems, run the following command install `gdb`:

```
sudo apt-get install gdb
```

- For CentOS and RedHat Enterprise Linux installations, run the following command to install `gdb`:

```
sudo yum install gdb
```

- For SUSE Enterprise Linux installations, run the following command to install `gdb`:

```
sudo zypper install gdb
```

You can also install `gdb` on OS X using a package manager. See the [Installing GDB on OS X Mavericks](#) article for more information.

Examples

Example 211. Script Example

This is an example `Script` for driving GDB.

The `GdbServer` monitor expects specific output that the included exploit analysis to generate the fault title, risk and bucket hashes.

The following regular expressions are used to extract information from log. If replacing the crash analysis portion of the script make sure to output to allow the first three regexes to pass.

```
"^Hash: (\w+)\.(\w+)$" -- Major and Minor bucket
"^Exploitability Classification: (.*)$" -- Risk
"^Short description: (.*)$" -- Title
"^Other tags: (.*)$" -- If found, added to Title
```

Script:

```
define log_if_crash
if ($_thread != 0x00)
printf "Crash detected, running exploitable.\n"
set logging overwrite on
set logging redirect on
set logging on {{gdbLog}}          ②
exploitable -v                    ④
printf "\n--- Info Frame ---\n\n"
info frame
printf "\n--- Info Registers ---\n\n"
info registers
printf "\n--- Backtrace ---\n\n"
thread apply all bt full
set logging off
end
```

```

end

handle all nostop noprint
handle {{handleSignals}} stop print

file {{executable}}
source {{exploitableScript}}


python

import sys

def on_start(evt):
    import tempfile, os
    h,tmp = tempfile.mkstemp()
    os.close(h)
    with open(tmp, 'w') as f:
        f.write(str(gdb.inferiors()[0].pid))
    os.renames(tmp, '{{gdbPid}}')          ①
    gdb.events.cont.disconnect(on_start)

gdb.events.cont.connect(on_start)

print("starting inferior: '{{target}} {{remoteExecutable}}'")

try:
    if len('{{remoteExecutable}}') > 1:
        print('starting in extended-remote, multi-process mode')
        gdb.execute('set remote exec-file {{remoteExecutable}}')
        gdb.execute('target {{target}}')
        gdb.execute('run')
    else:
        gdb.execute('target {{target}}')
        gdb.execute('continue')

except:
    e = sys.exc_info()[1]
    print('Exception starting target: ' + str(e))
    import tempfile, os
    h,tmp = tempfile.mkstemp()
    os.close(h)
    with open(tmp, 'w') as f:
        f.write(str(e))
    os.renames(tmp, '{{gdbConnectError}}')  ③
    gdb.execute('quit')

end

```

```
log_if_crash  
quit
```

- ① Must generate *gdbPid* when target has been started
- ② Must generate *gdbLog* when a fault has been handled
- ③ On remote connection error, output the error message to *gdbConnectError*
- ④ Generates output compatible with regexes

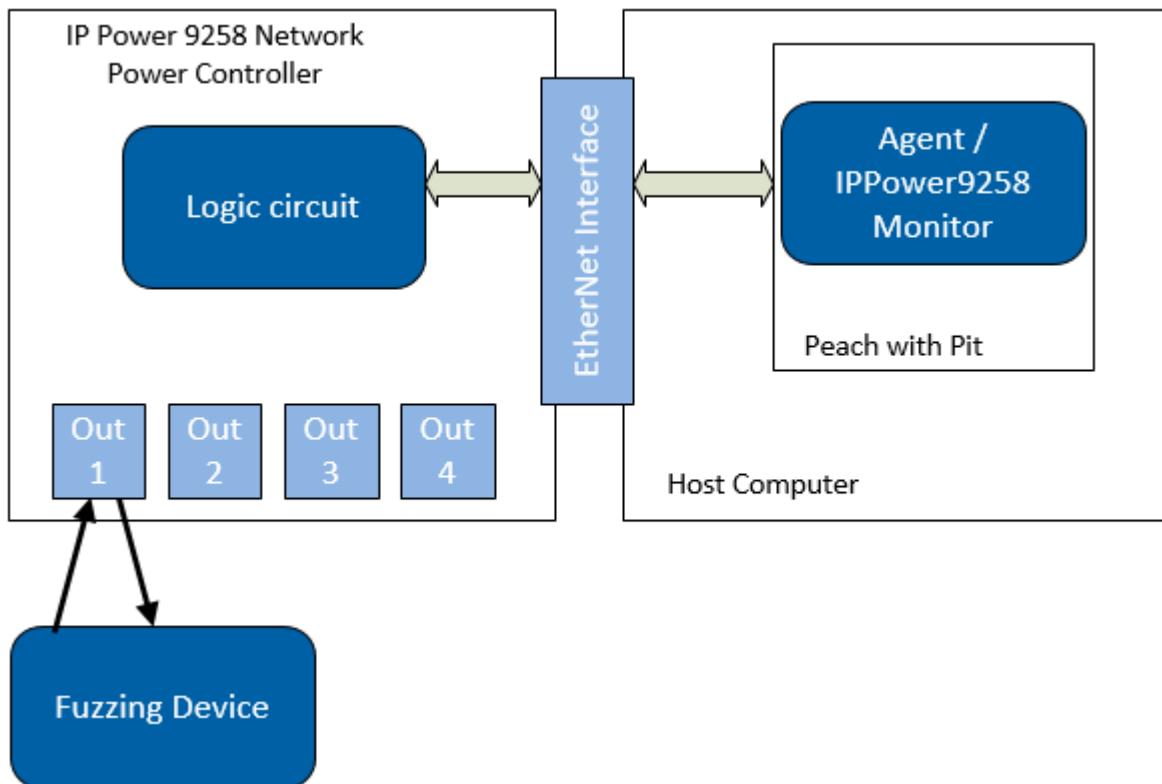
21.9.17. IpPower9258 Monitor

Monitor Category: Automation

The *IpPower9258* monitor controls an IP Power 9258 Network Power Controller (IP9258). The IP9258 consists of four three-prong outlets that support electrical loads of 6 amps at 110 or 240 VAC. Communications between Peach and the IP9258 uses Ethernet cabling. This monitor allows devices plugged into the IP Power 9258 switch to be powered on/off during fuzzing.

Each *IpPower9258* monitor switches an individual outlet. Use one monitor per outlet to control multiple outlets concurrently. The monitor can toggle power on an outlet at the following points in time:

- At the start or end of a fuzzing run
- At the start or end of each test iteration
- During the detection of a fault
- After detecting a fault
- At the start of an iteration that immediately follows a fault
- When a specified call is received from the state model





Peach also supports power distribution units that can be controlled using SNMPv1. For APC PDUs use the [APC Power Monitor](#), and for others try the [SNMP Power Monitor](#). To control power to a device by wiring through a relay, Peach provides the [CanaKit 4-Port USB Relay Monitor](#).

Parameters

Required:

Host

Host or IP address (can include HTTP interface port e.g. :8080)

Port

Port/Outlet to reset (1, 2, 3, 4)

User

Username to be used when connecting to the IP Power 9258 device.

Password

Password to be used when connecting to the IP Power 9258 device.

Optional:

StartOnCall

Toggle power when the specified call is received from the state model. This value is used only when the *When* parameter is set to [OnCall](#).

PowerOnOffPause

Pause in milliseconds between power off/power on, default is [500](#).

PowerOffPEnd

Power off when the fuzzing session completes, default is [false](#).

When

Specify one of the following values to determine when a port should be toggled:

"When" Setting	Description
DetectFault	Toggle power when checking for a fault. This occurs after OnIterationEnd.
OnStart	Toggle power when the fuzzing session starts. This occurs once per session.
OnEnd	Toggle power when the fuzzing session stops. This occurs once per session.

"When" Setting	Description
OnIterationStart	Toggle power at the start of each iteration.
OnIterationEnd	Toggle power at the end of each iteration.
OnFault	Toggle power when any monitor detects a fault. This is the default setting.
OnIterationStartAfterFault	Toggle power at the start of an iteration that immediately follows a fault detection.
OnCall	Toggle power when the call specified by the <i>StartOnCall</i> parameter is received from the state model.

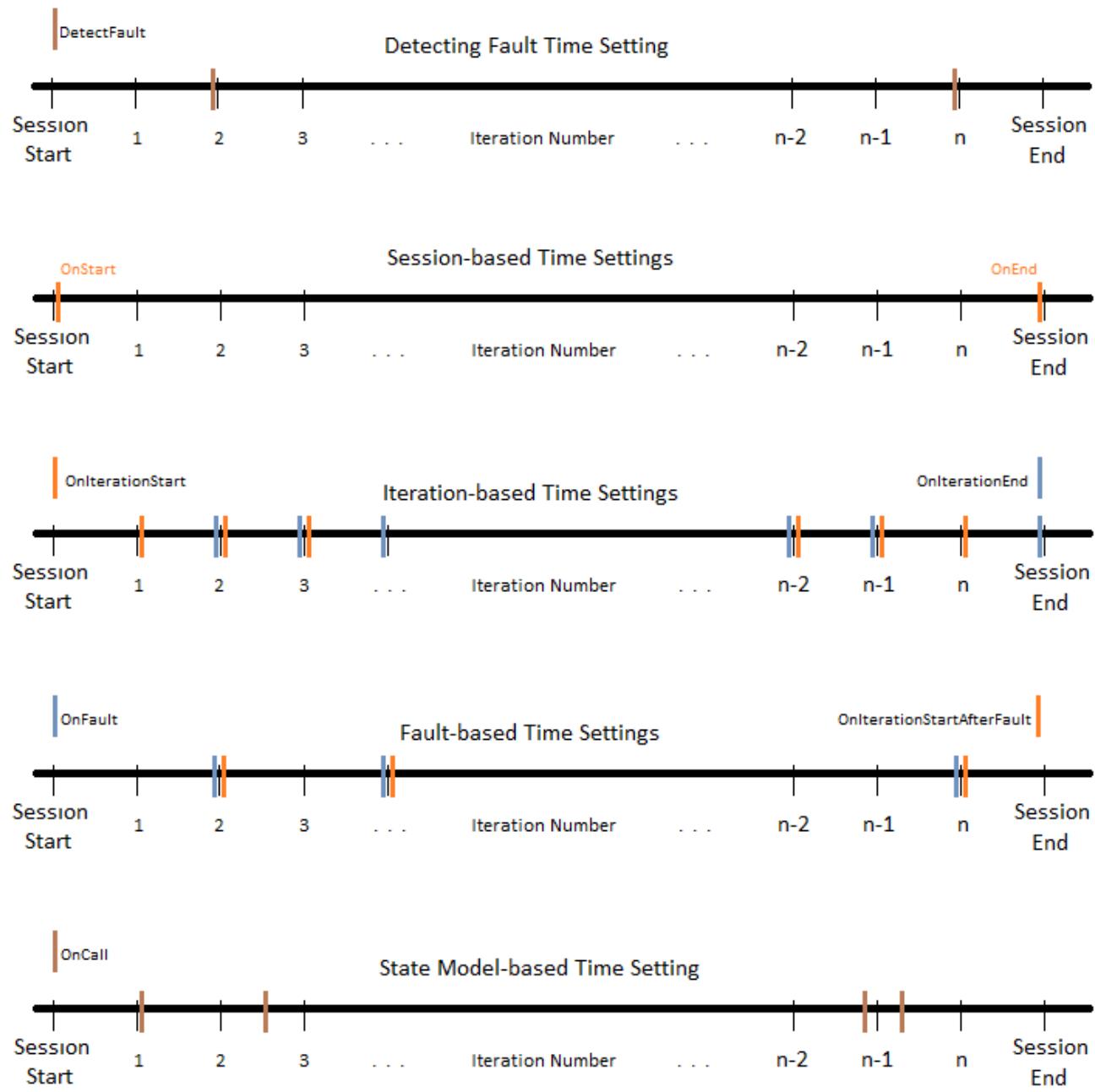


Figure 6. When Choices for Performing an Action

Examples

Example 212. Reset power on port 1

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="TheDataModel">
    <Number size="32" signed="false" value="31337" />
</DataModel>

<StateModel name="State" initialState="Initial" >
    <State name="Initial">
        <Action type="output">
            <DataModel ref="TheDataModel"/>
        </Action>
    </State>
</StateModel>

<Agent name="Local">
    <Monitor class="IpPower9258">
        <Param name="Host" value="192.168.1.1:8080" />
        <Param name="Port" value="1" />
        <Param name="User" value="peach" />
        <Param name="Password" value="power" />
    </Monitor>
</Agent>

<Test name="Default">
    <StateModel ref="State"/>
    <Agent ref="Local" />
    <Publisher class="ConsoleHex"/>
</Test>
</Peach>
```

21.9.18. LinuxCoreFile Monitor (Linux)

Monitor Categories: Data collection, Fault detection

The `gdb` debugger and [Gdb Monitor](#) are the preferred tools to detect crashes and to collect core files on Linux.



This monitor runs only on Linux systems. Use this monitor when `gdb` is not an option, such as when anti-debugging mechanisms are used on the test target.

The *LinuxCoreFile* monitor detects when a process crashes and collects the resulting core file. The monitoring scope includes all processes by default, but can focus on a single executable.

This monitor runs for the entire fuzzing session and uses the Linux crash recording facility. When a crash occurs, the *LinuxCoreFile* monitor pulls the logging information and available core files, and sets the bucketing information. The major bucket is based on the name of the crashed executable, and the minor bucket is a constant value based on the string "CORE".

At the end of the session, *LinuxCoreFile* restores the saved state and removes the logging folder.

Because the *LinuxCoreFile* monitor registers a custom core file handler with the Linux kernel, only one instance of the monitor is allowed to run on a host at any given time.

Setup Requirements

This monitor has the following setup requirements:

- *LinuxCoreFile* requires root or equivalent privileges to run. The *LinuxCoreFile* monitor registers a script with the kernel to catch core dumps of faulting processes.
- Core files must be enabled. If the maximum core file size in the system is zero, no core files are created. You can enable core file generation using the following process:

1. Find the current core file hard and soft limits in use by the operating system. The hard limit is an absolute maximum that, once set, cannot be increased during a session. The soft limit is the current maximum file size that you can adjust up to the value of the hard limit. If the hard limit or the soft limit is set to zero, core files are disabled.

- You can display the core file hard limit using the following command:

```
ulimit -Hc
```

- You can display the current core file soft limit using the following command:

```
ulimit -Sc
```

2. Set the core file hard limit using the following command. You can specify `unlimited` or a

numeric value that represents the number of 512-byte blocks to allow in a core file:

```
ulimit -Hc unlimited
```

3. Set the core file soft limit using the following command. You can specify any value less than or equal to the hard limit.

```
ulimit -Sc unlimited
```

The hard and soft limits can be added to [/etc/sysctl.conf](#). Then, whenever this file loads, appropriate core file limits are specified.

- [gdb](#) must be installed to analyze the resulting core files. For information on installing [gdb](#), see the [Gdb Monitor](#).

Parameters

Required:

None.

Optional:

Executable

Target executable process, used to filter crashes, defaults to all processes.

LogFolder

Folder with log files, defaults to [/var/peachcrash](#).

Examples

Example 213. Basic Usage

This example produces a fault on the first iteration to show how the LinuxCoreFile monitor works.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String name="TheString" value="Hello World!" />
    </DataModel>

    <StateModel name="State" initialState="Initial" >
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
            <Action type="close"/>
            <Action type="call" method="ScoobySnacks" publisher="Peach.Agent"/>
        </State>
    </StateModel>

    <Agent name="Local">
        <Monitor class="LinuxCoreFile">
            <Param name="Executable" value="CrashableServer" />
        </Monitor>

        <Monitor class="Process">
            <Param name="Executable" value="CrashableServer" />
            <Param name="Arguments" value="127.0.0.1 4244" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <StateModel ref="State" />
        <Agent ref="Local" />

        <Publisher class="Tcp">
            <Param name="Host" value="127.0.0.1" />
            <Param name="Port" value="4244" />
        </Publisher>
    </Test>
</Peach>

```

Output from this example.

```

$ peach --debug --seed=1 --range=1,1 example.xml

Peach.Core.Engine runTest: context.config.range == true, start: 1, stop: 1

```

```

[*] Test 'Default' starting with random seed 1.
Peach.Core.Agent.Agent StartMonitor: Monitor LinuxCoreF
Peach.Core.Agent.Agent StartMonitor: Monitor_1 Process
Peach.Core.Agent.Agent SessionStarting: Monitor
Peach.Core.Agent.Agent SessionStarting: Monitor_1
Peach.Core.Agent.Monitors.Process _Start(): Starting process
Establishing the listener...

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Waiting for a connection...
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpClientPublisher start()
Peach.Core.Publishers.TcpClientPublisher open()
Accepted connection from 127.0.0.1:35321.
Peach.Core.Publishers.TcpClientPublisher output(12 bytes)
Peach.Core.Publishers.TcpClientPublisher

00000000  48 65 6C 6C 6F 20 57 6F  72 6C 64 21          Hello World!

Received 12 bytes from client.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.TcpClientPublisher close()
Peach.Core.Publishers.TcpClientPublisher Shutting down connection to 127.0.0.1:4244
Connection closed by peer.
Shutting connection down...
Connection is down.
Waiting for a connection...
Peach.Core.Publishers.TcpClientPublisher Read 0 bytes from 127.0.0.1:4244, closing
client connection.
Peach.Core.Publishers.TcpClientPublisher Closing connection to 127.0.0.1:4244
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Call
Peach.Core.Agent.AgentManager Message: Action.Call => ScoobySnacks

[1,1,0:00:00.347] Performing iteration
[*] Fuzzing: TheDataModel.TheString
[*] Mutator: UnicodeBomMutator
Peach.Core.MutationStrategies.RandomStrategy Action_Start: Fuzzing:
TheDataModel.TheString
Peach.Core.MutationStrategies.RandomStrategy Action_Start: Mutator:
UnicodeBomMutator
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpClientPublisher open()
Accepted connection from 127.0.0.1:48111.

```

```

Peach.Core.Publishers.TcpClientPublisher output(1354 bytes)
Peach.Core.Publishers.TcpClientPublisher

00000000  FE FF FF FE FE FF FE BB  BF FE FF FF FE FE FF FE  ??????????????????
00000010  BB BF FF FE FF FE FF FE  FF FE FF FE FF FE FF FE  ??????????????????
00000020  FF FE FF FE FE FF FE FF  FE FF FE BB BF FE FF FE  ??????????????????
00000030  BB BF FF FE FF FE FF  FE BB BF FE FF FE FF FE  ??????????????????

...
Received 1024 bytes from client.

In CrashMe()
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.TcpClientPublisher close()
Peach.Core.Publishers.TcpClientPublisher Shutting down connection to 127.0.0.1:4244
Peach.Core.Publishers.TcpClientPublisher Unable to complete reading data from
127.0.0.1:4244. Connection reset by peer
Peach.Core.Publishers.TcpClientPublisher Closing connection to 127.0.0.1:4244
Peach.Core.Dom.Action ActionType.Call
Peach.Core.Agent.AgentManager Message: Action.Call => ScoobySnacks
Peach.Core.Agent.AgentManager Fault detected. Collecting monitor data.
Peach.Core.Engine runTest: detected fault on iteration 1

-- Caught fault at iteration 1, trying to reproduce --

Peach.Core.Loggers.FileLogger Found core fault [] ①
Peach.Core.Loggers.FileLogger Saving action: 1.Initial.Action.bin
Peach.Core.Loggers.FileLogger Saving fault: ②
Peach.Core.Engine runTest: Attempting to reproduce fault.
Peach.Core.Engine runTest: replaying iteration 1

[1,1,0:00:02.673] Performing iteration
Peach.Core.Agent.Monitors.Process _Start(): Starting process
[*] Fuzzing: TheDataModel.TheString
[*] Mutator: UnicodeBomMutator
Peach.Core.MutationStrategies.RandomStrategy Action_Starting: Fuzzing:
TheDataModel.TheString
Peach.Core.MutationStrategies.RandomStrategy Action_Starting: Mutator:
UnicodeBomMutator
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpClientPublisher open()
Establishing the listener...
Peach.Core.Publishers.TcpClientPublisher open: Warn, Unable to connect to remote host
127.0.0.1 on port 4244. Trying again in 1ms...
Waiting for a connection...
Peach.Core.Publishers.TcpClientPublisher output(1354 bytes)
Peach.Core.Publishers.TcpClientPublisher

```

```
00000000 FE FF FF FE FE FF BB BF FE FF FF FE FE FF FE ??????????????????  
00000010 BB BF FF FE FF FE FF FE FF FE FF FE FF FE ??????????????????  
00000020 FF FE FF FE FE FF FE FF FE FF BB BF FE FF FE ??????????????????  
00000030 BB BF FF FE FF FE FF FE BB BF FE FF FE FF FE ??????????????????  
...
```

Accepted connection from 127.0.0.1:40387.

Received 1024 bytes from client.

In CrashMe()

```
Peach.Core.Dom.Action ActionType.Close  
Peach.Core.Publishers.TcpClientPublisher close()  
Peach.Core.Publishers.TcpClientPublisher Shutting down connection to 127.0.0.1:4244  
Peach.Core.Publishers.TcpClientPublisher Unable to complete reading data from  
127.0.0.1:4244. Connection reset by peer  
Peach.Core.Publishers.TcpClientPublisher Closing connection to 127.0.0.1:4244  
Peach.Core.Dom.Action ActionType.Call  
Peach.Core.Agent.AgentManager Message: Action.Call => ScoobySnacks  
Peach.Core.Agent.AgentManager Fault detected. Collecting monitor data.  
Peach.Core.Engine runTest: detected fault on iteration 1  
Peach.Core.Engine >> OnFault
```

-- Reproduced fault at iteration 1 --

```
Peach.Core.Loggers.FileLogger Found core fault []  
Peach.Core.Loggers.FileLogger Saving action: 1.Initial.Action.bin  
Peach.Core.Loggers.FileLogger Saving fault:  
Peach.Core.Engine << OnFault  
Peach.Core.Engine runTest: Reproduced fault, continuing fuzzing at iteration 1  
Peach.Core.Publishers.TcpClientPublisher stop()  
Peach.Core.Agent.SessionFinished: Monitor_1  
Peach.Core.Agent.Monitors.Process _Stop(): Closing process handle  
Peach.Core.Agent.SessionFinished: Monitor
```

[*] Test 'Default' finished.

① When the program crashes, a core dump is produced and captured by the LinuxCoreFile monitor.

② The core dump is saved in the fault record and then removed from its original location.

21.9.19. Memory Monitor

Monitor Category: Data collection, Fault detection

The *Memory* monitor provides two modes of operation:

1. Data collection

When the *MemoryLimit* is **0**, the monitor will collect memory metrics at the end of each iteration. If a fault is triggered by another monitor, the collected memory metrics will be stored as **Usage.txt** in the fault's data bundle.

2. Fault detection

When the *MemoryLimit* is **> 0**, the monitor will initiate a fault when the memory usage for the process being monitored exceeds the specified limit.

The reported metrics for this monitor include the following items:

- **Private memory size** - Number of bytes allocated for the process. An approximate number of bytes a process is using.
- **Working set memory** - Total physical memory used by the process, consisting of in-memory private bytes plus memory-mapped files, such as DLLs.
- **Peak working set** - Largest working set used by the process.
- **Virtual memory size** - Total address space occupied by the entire process, including the working set plus paged private bytes and the standby list.



This monitor requires that you set a memory limit and identify a process to monitor, specifying the process by name or by process id.

Parameters

Required:

Either one of *Pid* or *ProcessName* is required. It is an error to specify both.

Pid

Process ID to monitor.

ProcessName

Name of the process to monitor.

Optional:

MemoryLimit

A value specified in bytes.

When `0` is specified, enable data collection mode, which causes memory metrics to be collected at the end of every iteration.

When a value greater than `0` is specified, enable fault detection mode, which causes a fault to occur if the memory usage of the monitored process exceeds the specified limit.

Defaults to `0`.

StopOnFault

Stop fuzzing if a fault is triggered, defaults to `false`.

Examples

Example 214. Monitor memory via PID

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

    <DataModel name="TheDataModel">
        <Number size="32" signed="false" value="31337" />
    </DataModel>

    <StateModel name="State" initialState="Initial" >
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="Local">
        <Monitor class="Memory">
            <Param name="ProcessName" value="Notepad"/>
            <Param name="MemoryLimit" value="1000000" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <StateModel ref="State" />
        <Agent ref="Local" />
        <Publisher class="ConsoleHex" />
    </Test>
</Peach>

```

Output from example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 54974.
Peach.Core.Agent.Agent StartMonitor: Monitor Memory
Peach.Core.Agent.SessionStarting: Monitor

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(4 bytes)
00000000  69 7A 00 00                                iz??
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()
Peach.Core.Agent.Agent SessionFinished: Monitor

[*] Test 'Default' finished.
```

21.9.20. NetworkCapture Monitor

Monitor Categories: Data collection, Fault detection

The *NetworkCapture* monitor performs network captures during the fuzzing iteration. When a packet arrives, this monitor writes the content into a file, increments the received packet count, and waits for the next packet to arrive. If a filter is used, the captured packets and associated packet count are for packets that pass the filtering criteria.

The captured data begins afresh for each iteration. If a fault occurs, the captured data is logged as a **.pcap** file and returned with the fault data. The **.pcap** file is compatible with Wireshark and **tcpdump**. A sample capture follows:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.146.128	10.71.0.126	QUAKEWC	59	Client to Server Connectionless [Malformed Packet]
2	0.004981	10.71.0.126	192.168.146.128	QUAKEWC	60	Server to Client Connectionless [Malformed Packet]
3	0.025181	192.168.146.128	10.71.0.126	QUAKEWC	139	Client to Server Connectionless [Malformed Packet]
4	0.027660	10.71.0.126	192.168.146.128	QUAKEWC	60	Server to Client Connectionless [Malformed Packet]
5	0.058870	192.168.146.128	10.71.0.126	QUAKEWC	69	Client to Server Game
6	0.085727	192.168.146.128	10.71.0.126	QUAKEWC	64	Client to Server Game
7	0.087455	10.71.0.126	192.168.146.128	QUAKEWC	60	Server to Client Game
8	0.118592	192.168.146.128	10.71.0.126	QUAKEWC	66	Client to Server Game
9	0.120024	10.71.0.126	192.168.146.128	QUAKEWC	60	Server to Client Game
10	0.149570	192.168.146.128	10.71.0.126	QUAKEWC	71	Client to Server Game
11	0.151014	10.71.0.126	192.168.146.128	QUAKEWC	323	Server to Client Game
12	0.188515	192.168.146.128	10.71.0.126	QUAKEWC	78	Client to Server Game
13	0.215186	192.168.146.128	10.71.0.126	QUAKEWC	63	Client to Server Game
14	0.217020	10.71.0.126	192.168.146.128	QUAKEWC	787	Server to Client Game
15	0.247285	192.168.146.128	10.71.0.126	QUAKEWC	80	Client to Server Game
16	0.277527	192.168.146.128	10.71.0.126	QUAKEWC	65	Client to Server Game
17	0.309523	192.168.146.128	10.71.0.126	QUAKEWC	66	Client to Server Game

Frame 1: 59 bytes on wire (472 bits), 59 bytes captured (472 bits)
Ethernet II, Src: VMware_33:39:7d (00:0c:29:33:39:7d), Dst: VMware_eb:aa:b1 (00:50:56:eb:aa:b1)
Internet Protocol Version 4, Src: 192.168.146.128 (192.168.146.128), Dst: 10.71.0.126 (10.71.0.126)
User Datagram Protocol, Src Port: 27001 (27001), Dst Port: 27500 (27500)
Quakeworld Network Protocol
[Malformed Packet: QUAKEWORLD]

```
0000  00 50 56 eb aa b1 00 0c 29 33 39 7d 08 00 45 00 .PV..... )39}..E.  
0010  00 2d 26 2b 00 00 80 11 b6 a7 c0 a8 92 80 0a 47 .-&.... ....G  
0020  00 7e 69 79 6b 6c 00 19 45 7e ff ff ff ff 67 65 .~jykl. E~....ge  
0030  74 63 68 61 6c 6c 65 6e 67 65 0a tchallen ge.
```

Peach supports multiple *NetworkCapture* monitors in the same pit, as well as simple and compound packet filters in a single monitor. A compound packet filter consists of more than one packet filter joined by AND or OR.

A packet filter is a boolean value applied to a packet. If the result of the operation is true, the packet is accepted. If the result is false, the packet is ignored.

The main benefit of using filters is performance. Applying a filter to a packet produces a smaller dataset, resulting in faster processing time for each iteration.

Two strategies for developing effective filters:

1. Develop a filter that pinpoints the packets to process.
2. Develop a filter that prunes unwanted packets using negative logic.



Generally, using one monitor with a complex filter is better than using two monitors with simpler filters. Using one monitor places all the packets in a single file with an order of arrival. Using multiple monitors makes correlating arrival more difficult because each monitor has its own file to keep the processed packets.

For information on filter strings, see the following:

- [Berkely Packet Filters - The Basics](#)
- [tcpdump man page](#)

Parameters

Required:

Device

Device name or port where the packet capture takes place.

The Peach command line option `--showdevices` causes Peach to generate a list of all available network interfaces.



- On Windows platforms, the `ipconfig` utility lists the network devices.
- On Unix platforms, the `ifconfig` utility lists the network devices.

Optional:

Filter

PCAP style filter string. If present, the filter restricts capture to packets that match the filter string.

Examples

Example 215. Capture output to CrashableServer on port 4244

This example runs `CrashableServer.exe` and capture all network traffic using the NetworkCapture monitor when a fault occurs.

To run this example, point to the `CrashableServer` location (normally the Peach directory).

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String value="Hello World!" />
    </DataModel>

    <StateModel name="State" initialState="Initial" >
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="Local">
        <Monitor class="NetworkCapture">
            <Param name="Device" value="Local Area Connection" />
            <Param name="Filter" value="port 4244" />
        </Monitor>

        <Monitor class="WindowsDebugger">
            <Param name="Executable" value="CrashableServer.exe" />
            <Param name="Arguments" value="127.0.0.1 4244" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <StateModel ref="State"/>
        <Agent ref="Local" />

        <Publisher class="TcpClient">
            <Param name="Host" value="127.0.0.1"/>
            <Param name="Port" value="4244"/>
        </Publisher>
    </Test>
</Peach>
```

Output from this example.

```
>peach -l --debug example.xml
```

```
[*] Test 'Default' starting with random seed 10470.  
Peach.Core.Agent.Agent StartMonitor: Monitor NetworkCapture  
Peach.Core.Agent.Agent StartMonitor: Monitor_1 WindowsDebugger  
Peach.Core.Agent.Agent SessionStarting: Monitor  
Peach.Core.Agent.Agent SessionStarting: Monitor_1  
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid SessionStarting  
Establishing the listener...  
  
[R1,-,-] Performing iteration  
Waiting for a connection...  
Peach.Core.Engine runTest: Performing recording iteration.  
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted  
Peach.Core.Dom.Action ActionType.Output  
Peach.Core.Publishers.TcpClientPublisher start()  
Peach.Core.Publishers.TcpClientPublisher open()  
Accepted connection from 127.0.0.1:51784.  
Peach.Core.Publishers.TcpClientPublisher output(12 bytes)  
Peach.Core.Publishers.TcpClientPublisher  
  
00000000  48 65 6C 6C 6F 20 57 6F  72 6C 64 21          Hello World!  
  
Received 12 bytes from client.  
Peach.Core.Publishers.TcpClientPublisher close()  
Peach.Core.Publishers.TcpClientPublisher Shutting down connection to 127.0.0.1:4244  
Connection closed by peer.  
Shutting connection down...  
Connection is down.  
Waiting for a connection...  
Peach.Core.Publishers.TcpClientPublisher Read 0 bytes from 127.0.0.1:4244, closing  
client connection.  
Peach.Core.Publishers.TcpClientPublisher Closing connection to 127.0.0.1:4244  
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid DetectedFault()  
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid DetectedFault() - No fault detected  
Peach.Core.Engine runTest: context.config.singleIteration == true  
Peach.Core.Publishers.TcpClientPublisher stop()  
Peach.Core.Agent.Agent SessionFinished: Monitor_1  
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid SessionFinished  
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _StopDebugger  
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _FinishDebugger  
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _StopDebugger  
Peach.Core.Agent.Agent SessionFinished: Monitor  
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _StopDebugger  
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _FinishDebugger  
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _StopDebugger  
  
[*] Test 'Default' finished.
```

Running this example for a few iterations will produce a crash. When Peach is logging the fault, a **.pcap** file is created inside the fault record.

21.9.21. PageHeap Monitor (Windows)

Monitor Category: Automation

The *PageHeap* monitor enables heap allocation monitoring for an executable through the Windows debugger. Peach sets and clears the parameters used for monitoring heap allocation at the beginning and end of the fuzzing session.



The *PageHeap* monitor requires heightened or administrative permissions to run.

Parameters

Required:

Executable

Executable name (no path)

Optional:

WinDbgPath

Path to the Windows Debugger installation. If not provided, Peach attempts to locate the directory.

Examples

Example 216. Enable for Notepad

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String length="32" value="31337" />
    </DataModel>

    <StateModel name="State" initialState="Initial" >
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
            <Action type="close"/>
            <Action type="call" method="launchProgram" publisher="Peach.Agent"/>
        </State>
    </StateModel>

    <Agent name="Local">
        <Monitor class="WindowsDebugger">
            <Param name="Executable" value="notepad.exe" />
            <Param name="Arguments" value="fuzzed.txt" />
            <Param name="StartOnCall" value="launchProgram" />
        </Monitor>

        <Monitor class="PageHeap">
            <Param name="Executable" value="notepad.exe" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <StateModel ref="State" />
        <Agent ref="Local" />
        <Publisher class="File">
            <Param name="FileName" value="fuzzed.txt" />
        </Publisher>
    </Test>
</Peach>

```

Output from this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 5090.
Peach.Core.Agent.Agent StartMonitor: Monitor WindowsDebugger
Peach.Core.Agent.Agent StartMonitor: Monitor_1 PageHeap
Peach.Core.Agent.Agent SessionStarting: Monitor
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid SessionStarting
Peach.Core.Agent.Agent SessionStarting: Monitor_1

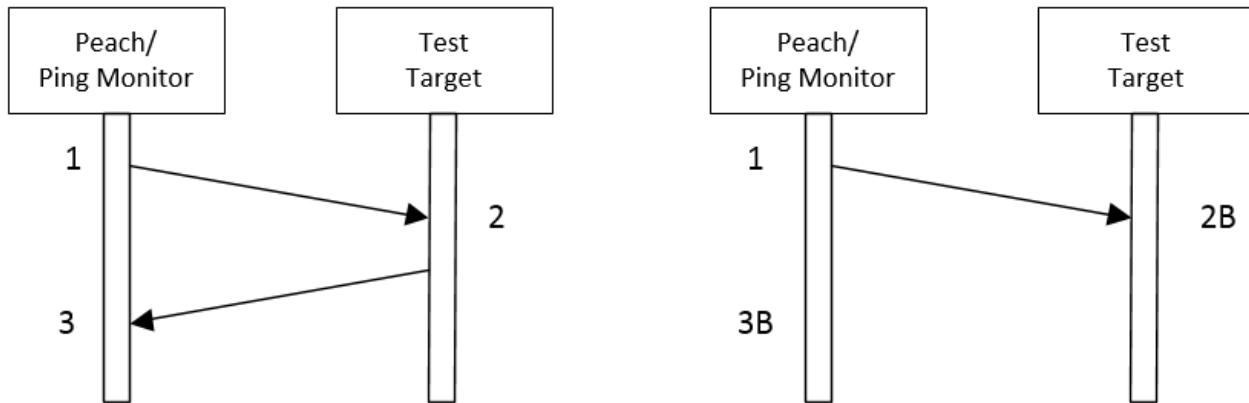
[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(32 bytes)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Call
Peach.Core.Agent.AgentManager Message: Action.Call => launchProgram
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _StopDebugger
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid Cpu is idle, stopping process.
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _StopDebugger
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid DetectedFault()
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid DetectedFault() - No fault detected
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.FilePublisher stop()
Peach.Core.Agent.Agent SessionFinished: Monitor_1
Peach.Core.Agent.Agent SessionFinished: Monitor
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid SessionFinished
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _StopDebugger
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _FinishDebugger
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _StopDebugger
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _StopDebugger
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _FinishDebugger
Peach.Core.Agent.Monitors.WindowsDebuggerHybrid _StopDebugger

[*] Test 'Default' finished.
```

21.9.22. Ping Monitor

Monitor Category: Fault detection

The *Ping* monitor verifies whether a device is functioning by sending a packet to a target location, and waiting for a response from the device. *Ping* continues to monitor until either a *Timeout* occurs or a response from the target device reaches the *Ping* monitor. *Ping* runs at the end of each iteration.



In the first scenario (left), *Ping* successfully interacts with the test target.

1. At the end of each iteration, the *Ping* monitor sends a message to the test target.
2. In this case, the target receives the message and sends a response.
3. The monitor waits a specified time period for a response. In this case, the response within the time period, and gives the "OK" response. The *Ping* Monitor resets for the sequence to repeat at the end of the next iteration.

In the second scenario (right), the test target is non-responsive.

1. At the end of the iteration, the *Ping* monitor sends a message to the test target.
2. (2B) In this case, the target is non-responsive or no longer exists. No response is sent.
3. (3B) The monitor waits a specified time period for a response. In this case, a timeout occurs. The *Ping* monitor issues a fault and collects data around the fault. *Ping* resets for the next iteration.

Ping can validate that a target is still up or is waiting to restart. This is useful when fuzzing embedded devices that crash surreptitiously.

Additionally, by using the *FaultOnSuccess* parameter, *Ping* can help you to verify whether or not a new device at a specific address becomes available, or that a non-functioning device starts to function. For example, if you are fuzzing a device that programmatically turns other computers on, you can use *Ping* with *FaultOnSuccess* set to `true` to determine whether the computers are off (an uninteresting result) or on (an interesting result).

When running under Unix, the following restrictions apply:



- Root privileges are required
- *Data* parameter is limited to 72 bytes

Parameters

Required:

Host

Host name or IP address to ping.

Optional:

Data

Data to send in the ping packet payload. If this value is left blank, the OS will use default data to send.

FaultOnSuccess

Fault if ping is successful, defaults to `false`.

Timeout

Timeout value expressed in milliseconds, defaults to `1000`.

Examples

Example 217. Ping Host

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <Number size="32" signed="false" value="31337" />
    </DataModel>

    <StateModel name="State" initialState="Initial" >
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="Local">
        <Monitor class="Ping">
            <Param name="Host" value="www.peachfuzzer.com" />
            <Param name="Timeout" value="10000"/>
        </Monitor>
    </Agent>

    <Test name="Default">
        <StateModel ref="State" />

        <Agent ref="Local" />

        <Publisher class="TcpClient">
            <Param name="Host" value="www.peachfuzzer.com"/>
            <Param name="Port" value="80"/>
        </Publisher>
    </Test>
</Peach>

```

Output from this example.

```
>peach -l --debug example.xml

[*] Test 'Default' starting with random seed 26777.
Peach.Core.Agent.Agent StartMonitor: Monitor Ping
Peach.Core.Agent.SessionStarting: Monitor

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpClientPublisher start()
Peach.Core.Publishers.TcpClientPublisher open()
Peach.Core.Publishers.TcpClientPublisher output(4 bytes)
Peach.Core.Publishers.TcpClientPublisher

00000000  69 7A 00 00                                iz..

Peach.Core.Publishers.TcpClientPublisher close()
Peach.Core.Publishers.TcpClientPublisher Shutting down connection to
198.185.159.135:80
Peach.Core.Publishers.TcpClientPublisher Read 0 bytes from 198.185.159.135:80,
closing client connection.
Peach.Core.Publishers.TcpClientPublisher Closing connection to 198.185.159.135:80
Peach.Core.Agent.Monitors.PingMonitor DetectedFault(): www.peachfuzzer.com replied
after 10000ms
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.TcpClientPublisher stop()
Peach.Core.Agent.SessionFinished: Monitor

[*] Test 'Default' finished.
```

21.9.23. PopupWatcher Monitor (Windows)

Monitor Categories: Automation, Fault detection

The *PopupWatcher* monitor closes pop-up windows based on a window title. *PopupWatcher* monitors the test target for a list of windows. When a window opens whose name is in the list, *PopupWatcher* closes the pop-up window, and if specified in the configuration, initiates a fault.

This monitor starts at the beginning of the session and runs to the session end.



Some applications re-use a pop-up window for many purposes. The window has one title, but the main area of the window can display several different messages depending on the context of the application. If you're interested in monitoring a pop-up window based on content rather than the window title, consider using the [ButtonClicker Monitor](#).

Parameters

Required:

WindowNames

One or more Window names separated by commas.



The comma-delimited list should not contain any white-space characters.

Optional:

Fault

Trigger a fault when a pop-up window is found, default `false`.

Examples

Example 218. Close Notepad

For this example, you must first launch `notepad.exe`, then start Peach.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <Number size="32" signed="false" value="31337" />
    </DataModel>

    <StateModel name="State" initialState="Initial" >
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="Local">
        <Monitor class="PopupWatcher">
            <Param name="WindowNames" value="Notepad" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <StateModel ref="State"/>
        <Agent ref="Local" />
        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>

```

Output for this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 18897.
Peach.Core.Agent.Agent StartMonitor: Monitor PopupWatcher
Peach.Core.Agent.SessionStarting: Monitor

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(4 bytes)
00000000  69 7A 00 00                                iz??
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()
Peach.Core.Agent.Agent SessionFinished: Monitor

[*] Test 'Default' finished.
```

Example 219. Fault on Assert

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <Number size="32" signed="false" value="31337" />
    </DataModel>

    <StateModel name="State" initialState="Initial" >
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="Local">
        <Monitor class="PopupWatcher">
            <Param name="WindowNames" value="Assert" />
            <Param name="Fault" value="True" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <StateModel ref="State" />
        <Agent ref="Local" />
        <Publisher class="ConsoleHex" />
    </Test>
</Peach>

```

Output for this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 55395.
Peach.Core.Agent.Agent StartMonitor: Monitor PopupWatcher
Peach.Core.Agent.SessionStarting: Monitor

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(4 bytes)
00000000  69 7A 00 00                                iz??
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()
Peach.Core.Agent.Agent SessionFinished: Monitor

[*] Test 'Default' finished.
```

21.9.24. Process Monitor

Monitor Categories: Automation, Fault detection

The *Process* monitor controls a process during a fuzzing run. This monitor provides automation by controlling when the process starts, restarts and ends, and whether the process should be killed. This monitor also provides fault detection for early exit and failure to exit. Finally, the *Process* monitor provides data collection by copying messages from standard out and standard error.

The *Process* monitor provides the following functionality:

- Start a process at the session start.
- Start or restart a process on every iteration.
- Start a process in response to a call from the state model.
- Wait for a process to exit in response to a call from the state model.
- Restart a process when it exits.
- Terminates a process if the CPU usage is low.
- Logs a fault if a process exits early.
- Logs a fault if a process fails to exit.

The *Process* monitor initiates a fault when a process being monitored experiences the following conditions:

- Early exit
- Timeout while waiting for exit
- [Address Sanitizer](#) detection

Parameters

Required:

Executable

Executable to launch

Optional:

Arguments

Command line arguments

FaultOnEarlyExit

Trigger fault if process exits, defaults to `false`.

NoCpuKill

Disable process killing when the CPU usage nears zero, defaults to `false`.

RestartOnEachTest

Restart process on every iteration.

RestartAfterFault

If `true`, restarts the target when any monitor detects a fault. If `false`, restarts the target only if the process exits or crashes. This argument defaults to `true`.

StartOnCall

Start process when the specified call is received from the state model.

WaitForExitOnCall

Wait for process to exit when the specified call is received from the state model.

WaitForExitTimeout

Wait timeout value, expressed in milliseconds. Triggers a fault when the timeout period expires. Defaults to `10000`. Use `-1` for infinite, no timeout.

Examples

Example 220. Start Process

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String length="32" value="31337"/>
    </DataModel>

    <StateModel name="State" initialState="Initial" >
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>

            <Action type="close"/>
        </State>
    </StateModel>

    <Agent name="Local">
        <Monitor class="Process">
            <Param name="Executable" value="notepad.exe" />
            <Param name="Arguments" value="fuzzed.txt" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <StateModel ref="State"/>
        <Agent ref="Local" />

        <Publisher class="File">
            <Param name="FileName" value="fuzzed.txt"/>
        </Publisher>
    </Test>
</Peach>
```

Output for this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 20172.
Peach.Core.Agent.Agent StartMonitor: Monitor Process
Peach.Core.Agent.SessionStarting: Monitor
Peach.Core.Agent.Monitors.Process _Start(): Starting process

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(32 bytes)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Call
Peach.Core.Agent.AgentManager Message: Action.Call => launchProgram
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.FilePublisher stop()
Peach.Core.Agent.SessionFinished: Monitor
Peach.Core.Agent.Monitors.Process _Stop(): Killing process

[*] Test 'Default' finished.
```

When running this example, notepad opens when the session starts and closes when the session finishes.

Example 221. Restart Process on Each Test

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String length="32" value="31337" />
    </DataModel>

    <StateModel name="State" initialState="Initial" >
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
            <Action type="close"/>
        </State>
    </StateModel>

    <Agent name="Local">
        <Monitor class="Process">
            <Param name="Executable" value="notepad.exe" />
            <Param name="Arguments" value="fuzzed.txt" />
            <Param name="RestartOnEachTest" value="true" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <StateModel ref="State"/>
        <Agent ref="Local" />

        <Publisher class="File">
            <Param name="FileName" value="fuzzed.txt"/>
        </Publisher>
    </Test>
</Peach>
```

Output for this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 40308.
Peach.Core.Agent.Agent StartMonitor: Monitor Process
Peach.Core.Agent.SessionStarting: Monitor

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Agent.Monitors.Process _Start(): Starting process
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(32 bytes)
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Close
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Call
Peach.Core.Agent.AgentManager Message: Action.Call => launchProgram
Peach.Core.Agent.Monitors.Process _Stop(): Killing process
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.FilePublisher stop()
Peach.Core.Agent.SessionFinished: Monitor

[*] Test 'Default' finished.
```

When running this example, notepad repeatedly opens and closes.

Example 222. Start Process From State Model

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String length="32" value="31337" />
    </DataModel>

    <StateModel name="State" initialState="Initial" >
        <State name="Initial">
            <Action type="call" method="ScoobySnacks" publisher="Peach.Agent" />
        </State>
    </StateModel>

    <Agent name="Local">
        <Monitor class="Process">
            <Param name="Executable" value="notepad.exe" />
            <Param name="Arguments" value="fuzzed.txt" />
            <Param name="StartOnCall" value="ScoobySnacks" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <StateModel ref="State"/>
        <Agent ref="Local" />

        <Publisher class="File">
            <Param name="FileName" value="fuzzed.txt"/>
        </Publisher>
    </Test>
</Peach>
```

Output for this example.

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 63117.
Peach.Core.Agent.Agent StartMonitor: Monitor Process
Peach.Core.Agent.SessionStarting: Monitor

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Call
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Agent.AgentManager Message: Action.Call => ScoobySnacks
Peach.Core.Agent.Monitors.Process _Start(): Starting process
Peach.Core.Agent.Monitors.Process Cpu is idle, stopping process.
Peach.Core.Agent.Monitors.Process _Stop(): Killing process
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.FilePublisher stop()
Peach.Core.Agent.SessionFinished: Monitor

[*] Test 'Default' finished.
```

When running this example, notepad repeatedly opens and closes.

Example 223. Wait for process to exit in state model

For this example to complete, you must close notepad when it opens.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String length="32" value="31337" />
    </DataModel>

    <StateModel name="State" initialState="Initial">
        <State name="Initial">
            <!-- This action will block until process exits -->
            <Action type="call" method="ScoobySnacks" publisher="Peach.Agent" />
        </State>
    </StateModel>

    <Agent name="Local">
        <Monitor class="Process">
            <Param name="Executable" value="notepad.exe" />
            <Param name="Arguments" value="fuzzed.txt" />
            <Param name="WaitForExitOnCall" value="ScoobySnacks" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <StateModel ref="State"/>
        <Agent ref="Local" />

        <Publisher class="File">
            <Param name="FileName" value="fuzzed.txt"/>
        </Publisher>
    </Test>
</Peach>
```

Output from this example

```
>peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 6946.
Peach.Core.Agent.Agent StartMonitor: Monitor Process
Peach.Core.Agent.SessionStarting: Monitor
Peach.Core.Agent.Monitors.Process _Start(): Starting process

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Call
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Agent.AgentManager Message: Action.Call => ScoobySnacks
Peach.Core.Agent.Monitors.Process WaitForExit(10000)
Peach.Core.Agent.Monitors.Process _Stop(): Closing process handle
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.FilePublisher stop()
Peach.Core.Agent.SessionFinished: Monitor

[*] Test 'Default' finished.
```

Since notepad doesn't close automatically, remember to close notepad after each iteration.

Use [WaitForExitOnCall](#) when you want to halt fuzzing until the process closes.

21.9.25. ProcessKiller Monitor

Monitor Category: Automation

The *ProcessKiller* monitor kills (terminates) specified processes after each iteration.

Parameters

Required:

ProcessNames

Comma separated list of the processes to kill.



The process name is usually the executable filename without the extension (`.exe`). For example, `notepad.exe` will be `Notepad` or `notepad`. For Windows operating systems, the process name can be found by using the `tasklist.exe` command.

The comma-delimited list should not contain any white-space characters.

Optional:

None.

Examples

Example 224. Terminate two processes

Before running this example, open `notepad.exe`.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <Number size="32" signed="false" value="31337" />
    </DataModel>

    <StateModel name="State" initialState="Initial" >
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="Local">
        <Monitor class="ProcessKiller">
            <Param name="ProcessNames" value="Notepad" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <StateModel ref="State"/>
        <Agent ref="Local" />
        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>

```

Output for this example.

```
>peach -l --debug example.xml

[*] Test 'Default' starting with random seed 41446.
Peach.Core.Agent.Agent StartMonitor: Monitor ProcessKiller
Peach.Core.Agent.SessionStarting: Monitor

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(4 bytes)
00000000  69 7A 00 00                                iz??
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()
Peach.Core.Agent.Agent SessionFinished: Monitor

[*] Test 'Default' finished.
```

In this example, peach kills the **notepad** process whenever it runs.

21.9.26. Run Command

Monitor Categories: Automation, Data collection, Fault detection

The *RunCommand* monitor can be used to launch a command at various points in time during a fuzzing session:

- At the start or end of a fuzzing run
- At the start or end of each test iteration
- After detecting a fault
- At the start of an iteration that immediately follows a fault
- When a specified call is received from the state model

If a fault occurs, the monitor captures and logs the console output from `stdout` and `stderr`. Additionally, this monitor can initiate a fault under the following conditions (in-order of evaluation):

1. An [Address Sanitizer](#) message appears in `stderr`.
2. The specified regular expression matches messages in `stdout` or `stderr`.
3. The command takes longer to finish than the specified timeout duration.
4. The command exits with a specified exit code.
5. The command exits with a nonzero exit code.

Parameters

Required:

Command

The command or application to launch.

Optional:

Arguments

Command line arguments

FaultOnNonZeroExit

When this value is set to `true`, generate a fault if the exit code is non-zero. The default value is `false`.

FaultOnExitCode

When this value is set to `true`, generate a fault if the exit code matches the specified *FaultExitCode* parameter. The default value is `false`.

FaultExitCode

When *FaultOnExitCode* is set to `true`, generate a fault if the specified exit code occurs. The default

value is **1**.

StartOnCall

Launch the command when the monitor receives the specified call from the state machine. This value is used only when the *When* parameter is set to **OnCall**.

FaultOnRegex

If this value is specified, generate a fault if the specified regular expression matches the command output. The default value is unspecified.

Timeout

Maximum time period, in milliseconds, for the process to run. Generate a fault if the command runs longer than the specified value. This feature is disabled by specifying **-1** for the time period. The default value is **-1**.

WorkingDirectory

Set the current working directory for the command launched by this monitor. The default value is the Peach current working directory. The current working directory for the command is valid until the command changes the directory or ends.

When

Specify one of the following values to determine when to launch the command:

"When" Setting	Description
OnStart	Run command when the fuzzing session starts. This occurs once per session.
OnEnd	Run command when the fuzzing session stops. This occurs once per session.
OnIterationStart	Run command at the start of each iteration.
OnIterationEnd	Run command at the end of each iteration.
OnFault	Run command when any monitor detects a fault.
OnIterationStartAfterFault	Run command at the start of an iteration that immediately follows a fault detection.
OnCall	Run command when the call specified by the <i>StartOnCall</i> parameter is received from the state model. This is the default setting.

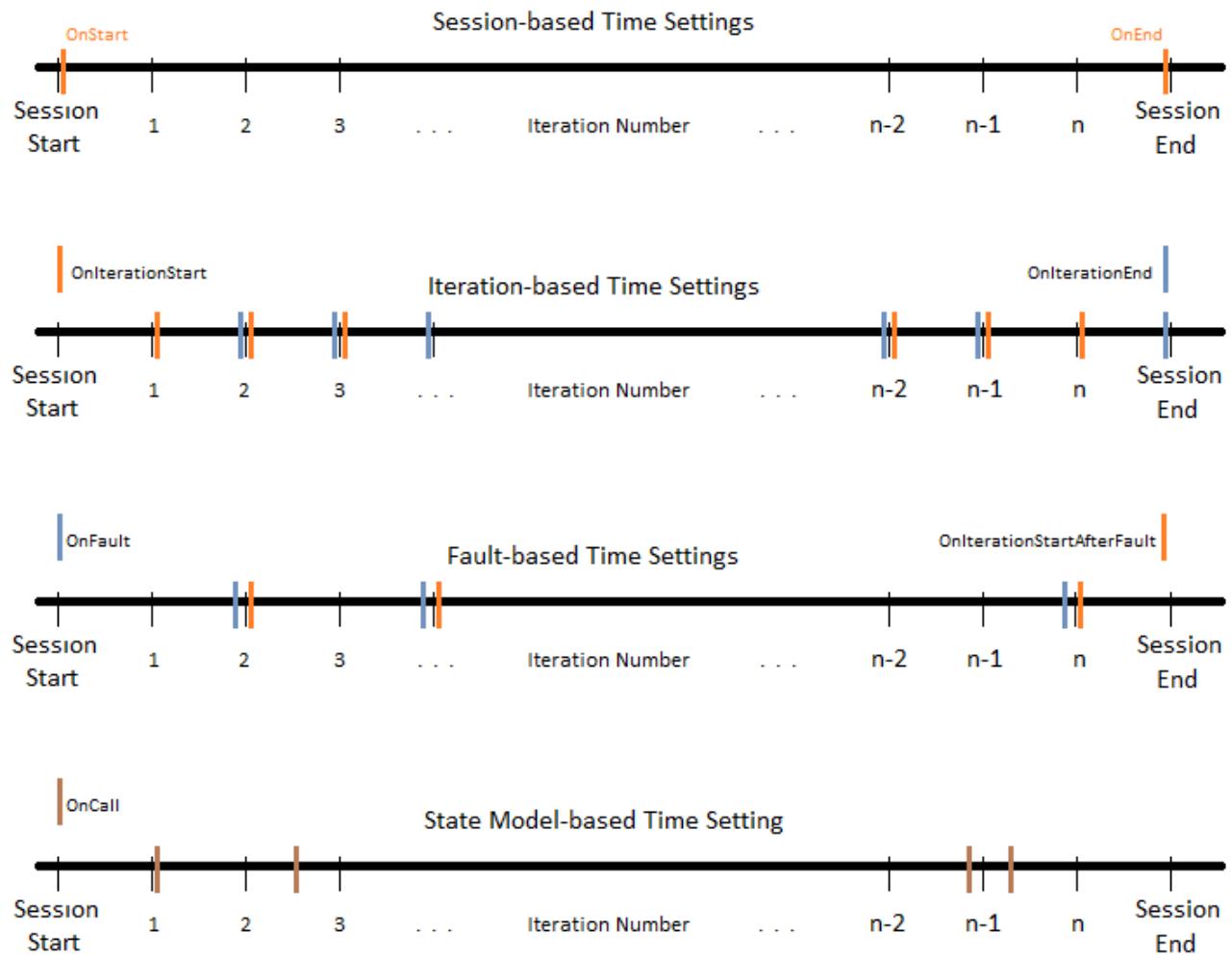


Figure 7. When Choices

Examples

Example 225. Using *RunCommand* for Fault Detection

This example detects a fault by checking for any lines that begin with `ERROR_`.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String value="Hello World!" />
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState">
        <State name="InitialState">
            <Action type="output">
                <DataModel ref="TheDataModel" />
            </Action>
        </State>
    </StateModel>

    <Agent name="TheAgent">
        <Monitor class="RunCommand">
            <Param name="Command" value="python"/>
            <Param name="Arguments" value="check_for_fault.py" />
            <Param name="WorkingDirectory" value="C:\MyScripts" />
            <Param name="FaultOnRegex" value="^ERROR_" />
            <Param name="When" value="OnIterationEnd" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <Agent ref="TheAgent"/>
        <StateModel ref="TheStateModel" />
        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>
```

Example 226. Using *RunCommand* for Data Collection

This example captures `stderr` and `stdout` for data collection when another monitor detects a fault.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String value="Hello World!" />
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState">
        <State name="InitialState">
            <Action type="output">
                <DataModel ref="TheDataModel" />
            </Action>
        </State>
    </StateModel>

    <Agent name="TheAgent">
        <Monitor class="RunCommand">
            <Param name="Command" value="python"/>
            <Param name="Arguments" value="collect_log.py" />
            <Param name="WorkingDirectory" value="C:\MyScripts" />
            <Param name="When" value="OnFault" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <Agent ref="TheAgent"/>
        <StateModel ref="TheStateModel" />
        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>
```

Example 227. Using *RunCommand* for Automation

This example runs the `clear_state.py` python script on the next iteration start after a fault is detected. This can be used to get the target back into a working state so that fuzzing can continue.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String value="Hello World!" />
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState">
        <State name="InitialState">
            <Action type="output">
                <DataModel ref="TheDataModel" />
            </Action>
        </State>
    </StateModel>

    <Agent name="TheAgent">
        <Monitor class="RunCommand">
            <Param name="Command" value="python"/>
            <Param name="Arguments" value="clear_state.py" />
            <Param name="WorkingDirectory" value="C:\MyScripts" />
            <Param name="When" value="OnIterationStartAfterFault" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <Agent ref="TheAgent"/>
        <StateModel ref="TheStateModel" />
        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>
```

21.9.27. SaveFile Monitor

Monitor Categories: Data collection

The *SaveFile* monitor saves a specified file as part of the logged data when a fault occurs. A copy of the file is placed in the log folder.

Parameters

Required:

Filename

File to save when a fault is detected by another monitor.

Optional:

None.

Examples

Collect logs when a fault occurs

In this example, the [Process Monitor](#) is used to launch `nginx` as the target. When a fault is detected by this monitor, the *SaveFile* monitor is configured to collect logs from `nginx`. These logs will be available as part of the data collected for the fault.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <Number size="32" signed="false" value="31337" />
    </DataModel>

    <StateModel name="State" initialState="Initial" >
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="Local">
        <Monitor class="Process">
            <Param name="Executable" value="/usr/sbin/nginx" />
            <Param name="FaultOnEarlyExit" value="true" />
        </Monitor>
        <Monitor name="Save access.log" class="SaveFile">
            <Param name="Filename" value="/var/log/nginx/access.log" />
        </Monitor>
        <Monitor name="Save error.log" class="SaveFile">
            <Param name="Filename" value="/var/log/nginx/error.log" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <StateModel ref="State"/>
        <Agent ref="Local" />
        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>

```

21.9.28. Serial Port Monitor

Monitor Categories: Automation, Data collection, Fault detection

The *Serial Port* monitor can be used to perform data collection, fault detection, or automation, based on specified parameters.

The default usage of the *Serial Port* monitor is data collection. The data received via the serial port is logged when a fault occurs.

To perform fault detection, specify a regular expression using the *FaultRegex* parameter. When the regular expression matches, Peach generates a fault.

For automation tasks, use the *WaitForRegex* and *WaitWhen* parameters. These automation parameters cause Peach to wait for matching input before continuing. The *Serial Port* monitor can wait at various points in time during a fuzzing session:

- At the start or end of a fuzzing run
- At the start or end of each test iteration
- After detecting a fault
- At the start of an iteration that immediately follows a fault
- When a specified call is received from the state model

Additionally, Peach supports multiple *Serial Port* monitors in a pit, allowing for more complex configurations. This can be used to monitor multiple serial ports. Multiple monitors may also be configured to use the same port, allowing for fault detection, automation, and/or data collection to occur on a single port.

Parameters

Required:

Port

The port to use (for example, `COM1` or `/dev/ttyS0`)

Optional:

BaudRate

The baud rate (only standard values are allowed). Defaults to `115200`.

DataBits

The data bits value. Defaults to `8`.

Parity

Specifies the parity bit. Defaults to `None`. Available options for this parameter are:

Even

Mark

None

Odd

Space

StopBits

Specifies the number of stop bits used. Defaults to **One**. Available options for this parameter are:

One

OnePointFive

Two

Handshake

Specifies the control protocol used in establishing a serial port communication. Defaults to **None**.

Available options for this parameter are:

None

RequestToSend

RequestToSendXOnXOff

XOnXOff

DtrEnable

Enables the Data Terminal Ready (DTR) signal during serial communication. Defaults to **false**.

RtsEnable

Enables the Request To Transmit (RTS) signal during serial communication. Defaults to **false**.

MaxBufferSize

Maximum amount of serial data to store in bytes. Defaults to **1048576**.

FaultRegex

Generate a fault when the specified regular expression matches received data. This causes the *Serial Port* monitor to be used for fault detection.

WaitRegex

Wait until the specified regular expression matches received data. This causes the *Serial Port* monitor to be used for automation.

WaitOnCall

Begin waiting for the regular expression specified in the *WaitRegex* parameter after the monitor receives the specified call from the state machine. This value is used only when the *WaitWhen* parameter is set to **OnCall**.

WaitWhen

Specify one of the following values to determine when to begin waiting for the regular expression specified in the *WaitRegex* parameter to match received data:

"WaitWhen" Setting	Description
OnStart	Waits when the fuzzing session starts. This occurs once per session. This is the default setting.
OnEnd	Waits when the fuzzing session stops. This occurs once per session.
OnIterationStart	Waits at the start of each iteration.
OnIterationEnd	Waits at the end of each iteration.
OnFault	Waits when any monitor detects a fault.
OnIterationStartAfterFault	Waits at the start of the iteration that immediately follows a fault detection.
OnCall	Waits upon receipt of the call specified by the <i>WaitOnCall</i> parameter from the state model.

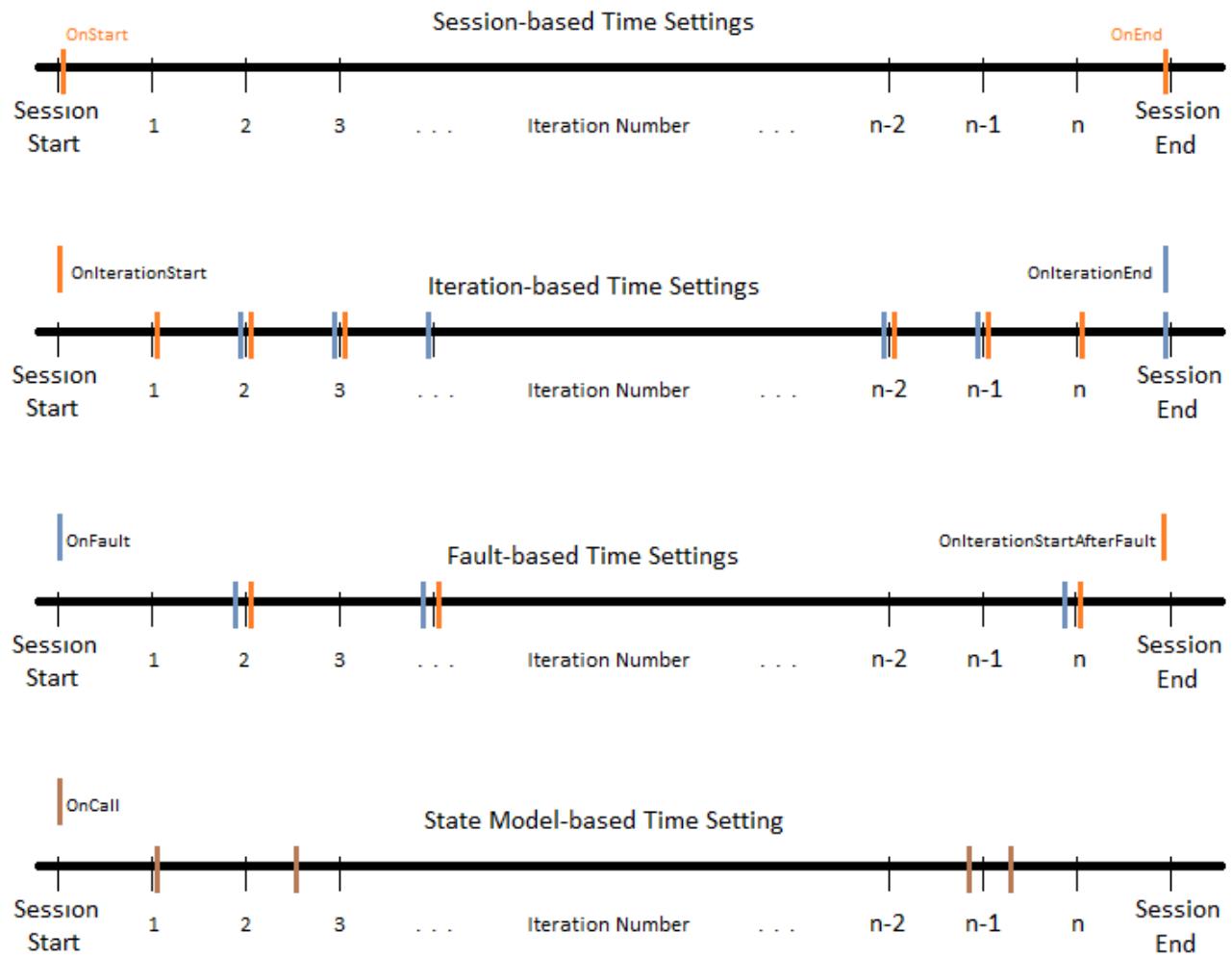


Figure 8. WaitWhen Choices

Examples

Example 228. Data Collection example

This example shows the *Serial Port* monitor configured to log data received from COM1.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="TheDataModel">
    <Number size="32" signed="false" value="31337" />
</DataModel>

<StateModel name="State" initialState="Initial" >
    <State name="Initial">
        <Action type="output">
            <DataModel ref="TheDataModel"/>
        </Action>
    </State>
</StateModel>

<Agent name="Local">
    <Monitor class="SerialPort">
        <Param name="Port" value="COM1" />
    </Monitor>
</Agent>

<Test name="Default">
    <StateModel ref="State"/>
    <Agent ref="Local" />
    <Publisher class="ConsoleHex"/>
</Test>
</Peach>
```

Example 229. Collect Serial Data on Fault

This example shows the *Serial Port* monitor configured to log received data and to generate a fault when the text **ERROR** is received.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="TheDataModel">
    <Number size="32" signed="false" value="31337" />
</DataModel>

<StateModel name="State" initialState="Initial" >
    <State name="Initial">
        <Action type="output">
            <DataModel ref="TheDataModel"/>
        </Action>
    </State>
</StateModel>

<Agent name="Local">
    <Monitor class="SerialPort">
        <Param name="Port" value="COM1" />
        <Param name="FaultRegex" value="ERROR" />
    </Monitor>
</Agent>

<Test name="Default">
    <StateModel ref="State"/>
    <Agent ref="Local" />
    <Publisher class="ConsoleHex"/>
</Test>
</Peach>
```

Example 230. Combined Automation and Fault Detection example

This example might be used when fuzzing a network device such as a router. One *Serial Port* monitor is configured to wait until the router has booted before starting the fuzzing session. Another *Serial Port* monitor is configured to detect faults and also to wait for the router to finish rebooting after a fault is detected. The **IpPower9258 Monitor** is configured to reboot the router after a fault is detected.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <Number size="32" signed="false" value="31337" />
    </DataModel>

    <StateModel name="State" initialState="Initial" >
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="Local">
        <!-- Waits at the start of fuzzing for message -->
        <Monitor class="SerialPort">
            <Param name="Port" value="COM1" />
            <Param name="WaitForRegex" value="Boot up completed" />
        </Monitor>

        <!-- Fault when "ERROR" is found, and also wait for boot message after fault. -->
        <Monitor class="SerialPort">
            <Param name="Port" value="COM1" />
            <Param name="FaultRegex" value="ERROR" />
            <Param name="WaitRegex" value="Boot up completed" />
            <Param name="WaitWhen" value="OnIterationAfterFault" />
        </Monitor>

        <!-- Restart device on fault -->
        <Monitor class="IpPower9258">
            <Param name="Host" value="10.1.1.1" />
            <Param name="Port" value="1" />
            <Param name="User" value="guest" />
            <Param name="Password" value="guest123" />
            <Param name="When" value="OnFault" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <StateModel ref="State"/>
        <Agent ref="Local" />
        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>

```

Example 231. Multiple Serial Port monitors for different ports

This example connects Peach to the console port and also the debug port of a target device. The monitor on the console port is set up for fault detection, data collection, and automation. The monitor on the debug port is set up for data collection.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="TheDataModel">
    <Number size="32" signed="false" value="31337" />
</DataModel>

<StateModel name="State" initialState="Initial" >
    <State name="Initial">
        <Action type="output">
            <DataModel ref="TheDataModel"/>
        </Action>
    </State>
</StateModel>

<Agent name="Local">
    <!-- Console Port -->
    <Monitor class="SerialPort">
        <Param name="Port" value="COM1" />
        <Param name="FaultRegex" value="ERROR" />
        <Param name="WaitRegex" value="Boot up completed" />
        <Param name="WaitWhen" value="OnIterationAfterFault" />
    </Monitor>

    <!-- Debug port -->
    <Monitor class="SerialPort">
        <Param name="Port" value="COM2" />
    </Monitor>
</Agent>

<Test name="Default">
    <StateModel ref="State"/>
    <Agent ref="Local" />
    <Publisher class="ConsoleHex"/>
</Test>
</Peach>
```

21.9.29. SNMP Power Monitor

Monitor Categories: Automation

The *SNMP Power* monitor switches outlets on a power distribution unit (PDU) on and off via SNMPv1. This monitor is useful for automatically power cycling devices during a fuzzing session.

Each *SNMP Power* monitor switches one or more of a PDU's outlets, according to the configuration. All affected outlets are given the same commands, so turning some outlets on and others off would require another monitor. The monitor can reset the power outlets at the following points in time:

- At the start or end of a fuzzing run
- At the start or end of each test iteration
- After detecting a fault
- At the start of an iteration that immediately follows a fault
- When a specified call is received from the state model



The [IpPower9258 Monitor](#) and [ApcPower Monitor](#) provide similar features, for IP Power 9258 and APC devices, respectively. For controlling power to a device by wiring through a relay, Peach provides a monitor for the [CanaKit 4-Port USB Relay Controller](#).

Parameters

Required:

Host

IP address of the switched power distribution unit.

OIDs

Comma-separated list of OIDs for controlling the power outlets. To determine the OIDs, start by installing the SNMP MIB provided by the device manufacturer. Use a utility like `snmptranslate` to lookup numeric OID associate with the OID name of an outlet. For example, `snmptranslate -On PowerNet-MIB::sPDUOutletCtl.1` indicates `.1.3.6.1.4.1.318.1.1.4.4.2.1.3.1` is the OID to use for outlet 1 on an APC Switched Power Distribution Unit (AC7900).

Optional:

OnCode

On indicator code used by outlet OIDs. Default is `1`.

OffCode

Off indicator code used by outlet OIDs. Default is `2`.

Port

SNMP port on the switched power distribution unit. Default is **161**.

ReadCommunity

SNMP community string to use when reading the state of the outlets. Default is **public**.

WriteCommunity

SNMP community string to use when modifying the state of the outlets. Default is **private**.

RequestTimeout

Maximum duration in milliseconds to block when sending an SNMP request to the PDU. Default is **1000**.

SanityCheckOnStart

On startup, ensure switch state changes persist. Default is **true**.

SanityCheckWaitTimeout

Maximum duration to wait for state change to take effect during startup sanity check. Default is **3000**.

ResetOnCall

Reset power when the specified call is received from the state model. This value is used only when the *When* parameter is set to **OnCall**.

PowerOffOnEnd

Power off when the fuzzing session completes, default is **false**.

PowerOnOffPause

Pause in milliseconds between power off/power on, default is **500**.

When

When to reset power on the specified outlets. Default is **OnFault**.

"When" Setting	Description
DetectFault	Reset power when checking for a fault. This occurs after OnIterationEnd.
OnStart	Reset power when the fuzzing session starts. This occurs once per session.
OnEnd	Reset power when the fuzzing session stops. This occurs once per session.
OnIterationStart	Reset power at the start of each iteration.
OnIterationEnd	Reset power at the end of each iteration.

"When" Setting	Description
OnFault	Reset power when any monitor detects a fault. This is the default setting.
OnIterationStartAfterFault	Reset power at the start of an iteration that immediately follows a fault detection.
OnCall	Reset power when the call specified by the <i>ResetOnCall</i> parameter is received from the state model.

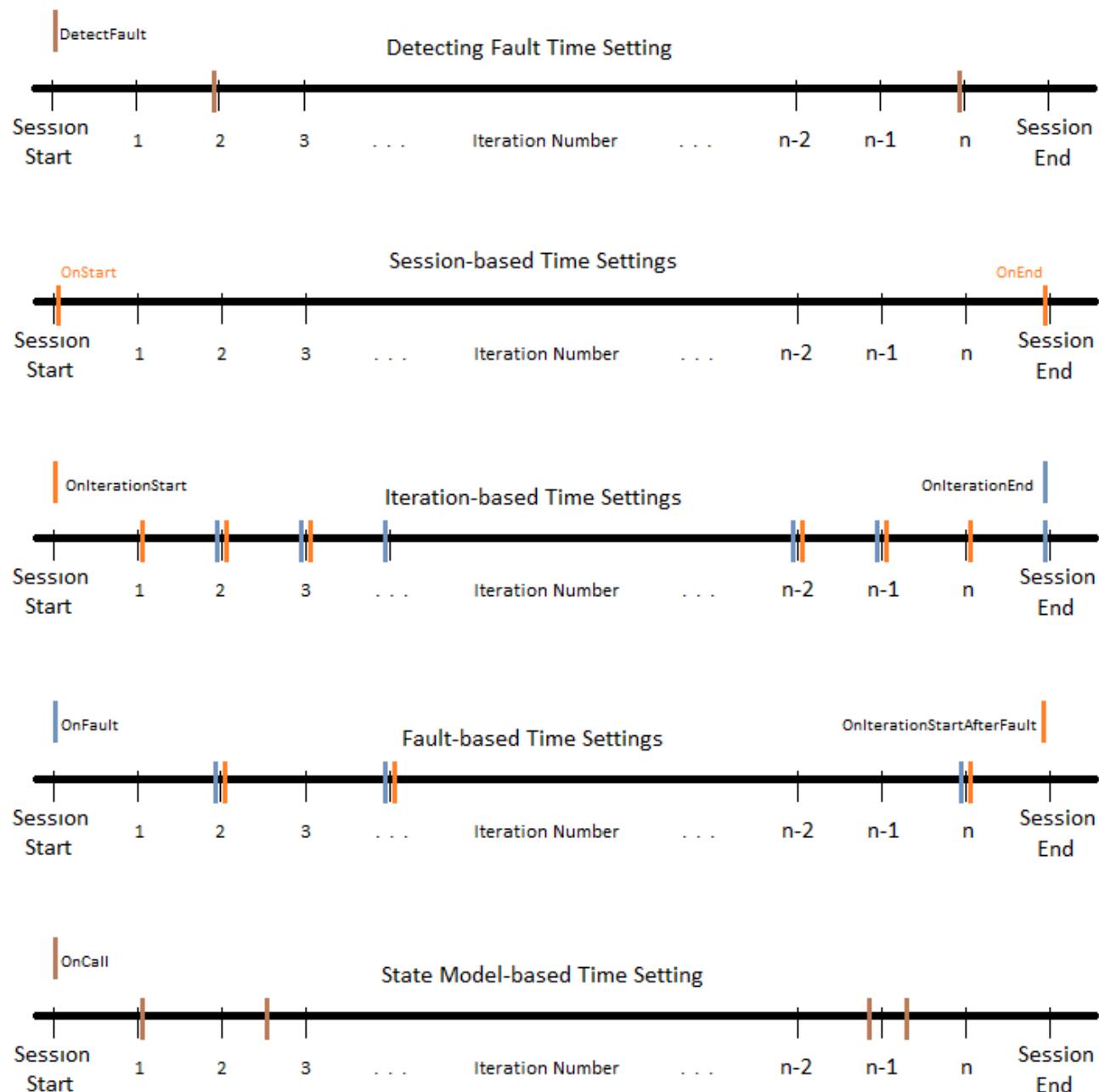


Figure 9. When Choices for Performing an Action

Examples

Example 232. Reset power on ports 1 and 2 of a PDU

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="TheDataModel">
    <Number size="32" signed="false" value="31337" />
</DataModel>

<StateModel name="State" initialState="Initial" >
    <State name="Initial">
        <Action type="output">
            <DataModel ref="TheDataModel"/>
        </Action>
    </State>
</StateModel>

<Agent name="Local">
    <Monitor class="SnmpPower">
        <Param name="Host" value="10.0.1.101" />
        <Param name="OIDs" value=
".1.3.6.1.4.1.318.1.1.4.4.2.1.3.1,.1.3.6.1.4.1.318.1.1.4.4.2.1.3.2" />
    </Monitor>
</Agent>

<Test name="Default">
    <StateModel ref="State"/>
    <Agent ref="Local" />
    <Publisher class="ConsoleHex"/>
</Test>
</Peach>
```

21.9.30. Socket Monitor

Monitor Category: Data collection, Fault detection

The *Socket* monitor waits for an incoming TCP or UDP connection at the end of a test iteration. This monitor accepts a point-to-point connection to a single host or a multicast connection where the host broadcasts to one or more clients. Multicast connections are not supported when using the TCP protocol.

The monitor can be configured to be used for data collection or fault detection depending on the *FaultOnSuccess* parameter value and whether or not data is received within the specified timeout. The following table provides the available options:

FaultOnSuccess	Data Received	Behavior
true	yes	Data collection
true	no	Fault detection
false	yes	Fault detection
false	no	Data collection

Parameters

Required:

None.

Optional:

Host

IP address of the remote host to receive data from. Defaults to "", which means accept data from any host.

Interface

IP address of the interface to listen on. Defaults to **0.0.0.0**, which means listen to all interfaces on the host.

Port

Port to listen on. Defaults to **8080**.

Protocol

Protocol type to listen for. Defaults to **tcp**. Available options for this parameter are **tcp** and **udp**.

Timeout

Length of time in milliseconds to wait for an incoming connection. Defaults to **1000**.

FaultOnSuccess

Generate a fault if no data is received. Defaults to `false`.

Examples

Example 233. Fault Detection example

This example generates a fault if data from a tcp connection on port `53` is received at the end of a test iteration.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="TheDataModel">
    <Number size="32" signed="false" value="31337" />
</DataModel>

<StateModel name="State" initialState="Initial" >
    <State name="Initial">
        <Action type="output">
            <DataModel ref="TheDataModel"/>
        </Action>
    </State>
</StateModel>

<Agent name="Local">
    <Monitor class="Socket">
        <Param name="Port" value="53" />
    </Monitor>
</Agent>

<Test name="Default">
    <StateModel ref="State"/>
    <Agent ref="Local" />
    <Publisher class="ConsoleHex"/>
</Test>
</Peach>
```

21.9.31. SshCommand Monitor

Monitor Categories: Automation, Data collection, Fault detection

The *SshCommand* monitor connects to a remote host over SSH (Secure Shell), runs a command, and waits for the command to complete. The output from the process is logged when a fault is detected. This monitor can operate as a fault detector, data collector, and automation module depending on configuration.

SshCommand supports password, keyboard, and private key authentication methods.

To increase the speed of operation, the monitor holds open the SSH connection to the remote machine across test iterations. This removes the cost of authenticating every time the command is executed. If multiple *SshCommand* monitors are configured against the same remote host, multiple SSH connections are created and held open.

Fault Detection

This monitor can perform fault detection depending on the configuration of the *FaultOnMatch* and *CheckValue* parameters. The following table describes the behavior of these parameters:

FaultOnMatch	CheckValue match	Behavior
true	yes	Fault detection
true	no	Data collection
false	yes	Data collection
false	no	Fault detection

This monitor will also automatically detect [AddressSanitizer](#) crash information and generate a fault if found.

Data Collection

The monitor always collects the output from the executed command and reports it for logging when a fault is detected.

Automation

The *SshCommand* monitor can run the specified command at various points in time during a fuzzing session:

- At the start or end of a fuzzing run
- At the start or end of each test iteration
- While detecting a fault
- After detecting a fault

- At the start of an iteration that immediately follows a fault
- When a specified call is received from the state model

Parameters

Required:

Host

Remote hostname or IP address for the SSH connection.

Username

Username for authentication with the remote host.

Command

The command to execute on the remote host.

Optional:

Password

Password for authentication with the remote host. Defaults to `""`. Either the *Password* or the *KeyPath* parameter must be set.

KeyPath

A local path to the private part of an SSH key-pair to be used for authentication with the remote host. Defaults to `""`. Either the *Password* or the *KeyPath* parameter must be set.

CheckValue

A regular expression to match the command output. Defaults to `""`.

FaultOnMatch

Trigger a fault if *FaultOnMatch* is `true` and the *CheckValue* regular expression matches, or *FaultOnMatch* is `false` and the *CheckValue* regular expression does not match. Defaults to `false`.

StartOnCall

Run the specified command after the monitor receives the specified call from the state machine. This value is used only when the *When* parameter is set to `OnCall`.

When

Specify one of the following values to determine when to run the specified command:

When Value	Description
<code>DetectFault</code>	Run the command to perform fault detection. Requires a regular expression to be specified in the <i>CheckValue</i> parameter. This is the default setting.

When Value	Description
OnStart	Run the command when fuzzing session starts. This occurs once per session.
OnEnd	Run the command when fuzzing session stops. This occurs once per session.
OnIterationStart	Run the command at the start of each iteration.
OnIterationEnd	Run the command at the end of each iteration.
OnFault	Run the command when any monitor detects a fault.
OnIterationStartAfterFault	Run the command at the start of an iteration that immediately follows a fault detection.
OnCall	Run the command upon receipt of the call specified by the <i>WaitOnCall</i> parameter from the state model.

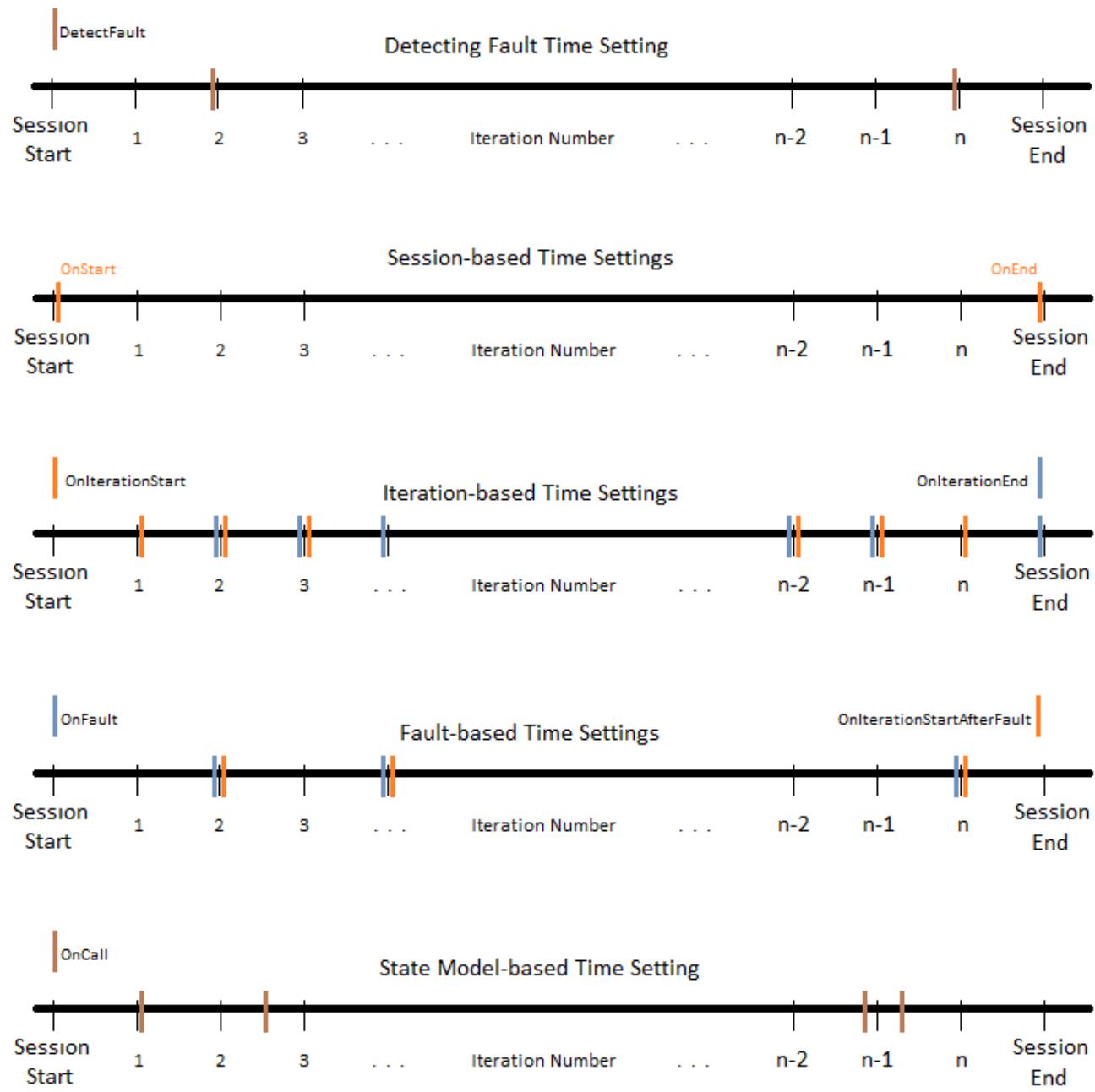


Figure 10. When Choices

Examples

Example 234. Checking for core dump files

This example connects to the target machine using SSH during the fault detection phase of a test iteration. A fault occurs if any core files exist.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

  <DataModel name="TheDataModel">
    <Number size="32" signed="false" value="31337" />
  </DataModel>

  <StateModel name="State" initialState="Initial" >
    <State name="Initial">
      <Action type="output">
        <DataModel ref="TheDataModel"/>
      </Action>
    </State>
  </StateModel>

  <Agent name="Local">
    <Monitor class="Ssh">
      <Param name="Host" value="my.target.com" />
      <Param name="Username" value="tester" />
      <Param name="Password" value="Password!" />
      <Param name="Command" value="ls /var/cores/*.core" />
      <Param name="CheckValue" value="target.*?.core" />
      <Param name="FaultOnMatch" value="true" />
    </Monitor>
  </Agent>

  <Test name="Default">
    <StateModel ref="State"/>
    <Agent ref="Local" />
    <Publisher class="ConsoleHex"/>
  </Test>
</Peach>
```

21.9.32. SshDownloader Monitor

Monitor Categories: Data collection

The *SshDownloader* monitor downloads a file or folder from a remote host over SSH SFTP (Secure Shell File Transfer Protocol) after any other monitor detects a fault.

SshDownloader supports password, keyboard, and private key authentication methods.

SshDownloader can be configured to delete files from the source after they have been downloaded.

Parameters

Required:

Host

Remote hostname or IP address for the SSH connection.

Username

Username for authentication with the remote host.

Optional:

Password

Password for authentication with the remote host. Defaults to `""`. Either the *Password* or the *KeyPath* parameter must be set.

KeyPath

A local path to the private part of an SSH key-pair to be used for authentication with the remote host. Defaults to `""`. Either the *Password* or the *KeyPath* parameter must be set.

File

Path of the remote file to download. Defaults to `""`. Either the *File* or the *Folder* parameter must be set.

Folder

Path of the remote folder to download. Defaults to `""`. Either the *File* or the *Folder* parameter must be set.

Remove

When this value is set to `true`, remove the remote file after the download completes. Defaults to `true`.

Examples

Example 235. Download a log file

This example downloads a log file when a fault is detected by any other monitor.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <Number size="32" signed="false" value="31337" />
    </DataModel>

    <StateModel name="State" initialState="Initial" >
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="Local">
        <Monitor class="SshDownloader">
            <Param name="Host" value="my.target.com" />
            <Param name="Username" value="tester" />
            <Param name="Password" value="Password!" />
            <Param name="File" value="/var/log/syslog" />
            <Param name="Remove" value="false" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <StateModel ref="State"/>
        <Agent ref="Local" />
        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>
```

21.9.33. Syslog Monitor

Monitor Categories: Automation, Data collection, Fault detection

The *Syslog* monitor listens on a specified port for incoming syslog messages. This monitor is capable of performing automation, data collection, and fault detection.

The default usage of the *Syslog* monitor is data collection. The syslog messages received are logged when a fault occurs.

To perform fault detection, specify a regular expression using the *FaultRegex* parameter. When the regular expression matches an incoming syslog message, Peach generates a fault.

For automation tasks, use the *WaitForRegex* and *WaitWhen* parameters. These automation parameters cause Peach to wait for matching input before continuing. The *Syslog* monitor can wait at various points in time during a fuzzing session:

- At the start or end of a fuzzing run
- At the start or end of each test iteration
- After detecting a fault
- At the start of an iteration that immediately follows a fault
- When a specified call is received from the state model

Parameters

Required:

None.

Optional:

Port

Port number to listen on. The default value is [514](#).

Interface

IP address of the interface to listen on. The default value is [0.0.0.0](#), which listens on all interfaces.

FaultRegex

Generate a fault when the specified regular expression matches received data. This causes the *Syslog* monitor to be used for fault detection.

WaitRegex

Wait until the specified regular expression matches received data. This causes the *Syslog* monitor to be used for automation.

WaitOnCall

Begin waiting for the regular expression specified in the *WaitRegex* parameter after the monitor receives the specified call from the state machine. This value is used only when the *WaitWhen* parameter is set to **OnCall**.

WaitWhen

Specify one of the following values to determine when to begin waiting for the regular expression specified in the *WaitRegex* parameter to match received data:

"WaitWhen" Setting	Description
OnStart	Waits when the fuzzing session starts. This occurs once per session. This is the default setting.
OnEnd	Waits when the fuzzing session stops. This occurs once per session.
OnIterationStart	Waits at the start of each iteration.
OnIterationEnd	Waits at the end of each iteration.
OnFault	Waits when any monitor detects a fault.
OnIterationStartAfterFault	Waits at the start of the iteration that immediately follows a fault detection.
OnCall	Waits upon receipt of the call specified by the <i>WaitOnCall</i> parameter from the state model.

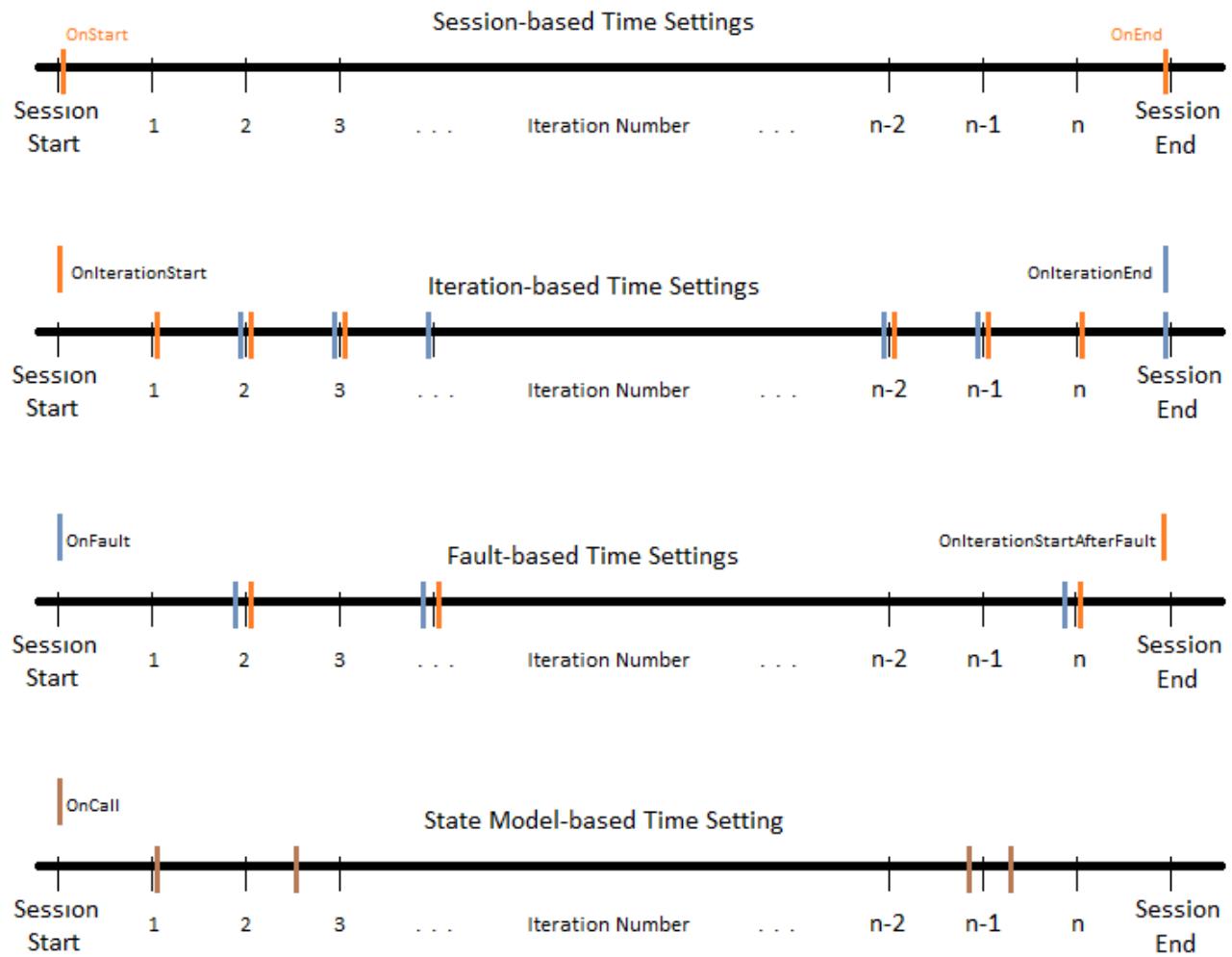


Figure 11. WaitWhen Choices

Examples

Example 236. Data Collection example

This example shows the *Syslog* monitor listening on the default port for incoming messages. When a fault occurs, all messages are saved.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <Number size="32" signed="false" value="31337" />
    </DataModel>

    <StateModel name="State" initialState="Initial" >
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="Local">
        <Monitor class="Syslog"/>
    </Agent>

    <Test name="Default">
        <StateModel ref="State"/>
        <Agent ref="Local" />
        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>
```

Example 237. Fault Detection example

This example shows the *Syslog* monitor listening on the default port for incoming messages. *Syslog* compares the *FaultRegex* regular expression with each incoming message; when a match occurs, the monitor generates a fault and saves the syslog data.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

  <DataModel name="TheDataModel">
    <Number size="32" signed="false" value="31337" />
  </DataModel>

  <StateModel name="State" initialState="Initial" >
    <State name="Initial">
      <Action type="output">
        <DataModel ref="TheDataModel"/>
      </Action>
    </State>
  </StateModel>

  <Agent name="Local">
    <Monitor class="Syslog">
      <Param name="FaultRegex" value="ERROR" />
    </Monitor>
  </Agent>

  <Test name="Default">
    <StateModel ref="State"/>
    <Agent ref="Local" />
    <Publisher class="ConsoleHex"/>
  </Test>
</Peach>
```

Example 238. Combined Automation and Fault Detection example

This example might be used when fuzzing a network device such as a router. One *Syslog* monitor is configured to wait until the router has booted before starting the fuzzing session. Another *Syslog* monitor is configured to detect faults and also to wait for the router to finish rebooting after a fault is detected. The [IpPower9258 Monitor](#) is configured to reboot the router after a fault is detected.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <Number size="32" signed="false" value="31337" />
    </DataModel>

    <StateModel name="State" initialState="Initial" >
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="Local">
        <!-- Waits at the start of fuzzing for message -->
        <Monitor class="Syslog">
            <Param name="WaitForRegex" value="Boot up completed" />
        </Monitor>

        <!-- Fault when "ERROR" is found, and also wait for boot message after fault. -->
        <Monitor class="Syslog">
            <Param name="FaultRegex" value="ERROR" />
            <Param name="WaitRegex" value="Boot up completed" />
            <Param name="WaitWhen" value="OnIterationAfterFault" />
        </Monitor>

        <!-- Restart device on fault -->
        <Monitor class="IpPower9258">
            <Param name="Host" value="10.1.1.1" />
            <Param name="Port" value="1" />
            <Param name="User" value="guest" />
            <Param name="Password" value="guest123" />
            <Param name="When" value="OnFault" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <StateModel ref="State"/>
        <Agent ref="Local" />
        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>

```

21.9.34. TcpPort Monitor

Monitor Categories: Automation, Data collection, Fault detection

The *TcpPort* monitor detects state changes on TCP ports. A state change is a transition in port status from *Open* to *Closed* or from *Closed* to *Open*. The *TcpPort* monitor can be configured in the following ways:

- as an automation task (wait until a specified state occurs)
- fault detection (fault on a specific state)
- data collection (report the current state)

For automation tasks, the *When* parameter can be configured to detect the state of a TCP port at various points in time during a fuzzing session:

- At the start or end of a fuzzing run
- At the start or end of each test iteration
- After detecting a fault
- At the start of an iteration that immediately follows a fault
- When a specified call is received from the state model

Parameters

Required:

Host

Hostname or IP address of the remote host

Port

Port number to monitor

Optional:

Action

Action to take (Automation, Data, Fault). Defaults to [Automation](#).

Automation

Wait for the port to reach a specified state. The *When* parameter determines when the monitor should detect the state of the specified port.

Data

Report the state of the port immediately following a fault that is detected by any other monitor.

Fault

Generate a fault if the port is in a specified state at the end of an iteration.

State

Port state to monitor. The default value is **Open**.

Open

The port is available for use.

Closed

The port is not available.

Timeout

The amount of time in milliseconds to wait for the specified TCP state to occur. Specify **-1** to indicate an infinite timeout.

WaitOnCall

Detect port state upon receipt of the specified call from the state model. This value is used only when the *When* parameter is set to **OnCall**.

When

Specify one of the following values to control when port state detection should occur during a fuzzing session:

"When" Setting	Description
OnStart	Detect port state when the fuzzing session starts. This occurs once per session.
OnEnd	Detect port state when the fuzzing session stops. This occurs once per session.
OnIterationStart	Detect port state at the start of each iteration.
OnIterationEnd	Detect port state at the end of each iteration.
OnFault	Detect port state when any monitor detects a fault.
OnIterationStartAfterFault	Detect port state at the start of the iteration that immediately follows a fault detection.
OnCall	Detect port state upon receipt of the call specified by the <i>WaitOnCall</i> parameter from the state model. This is the default setting.

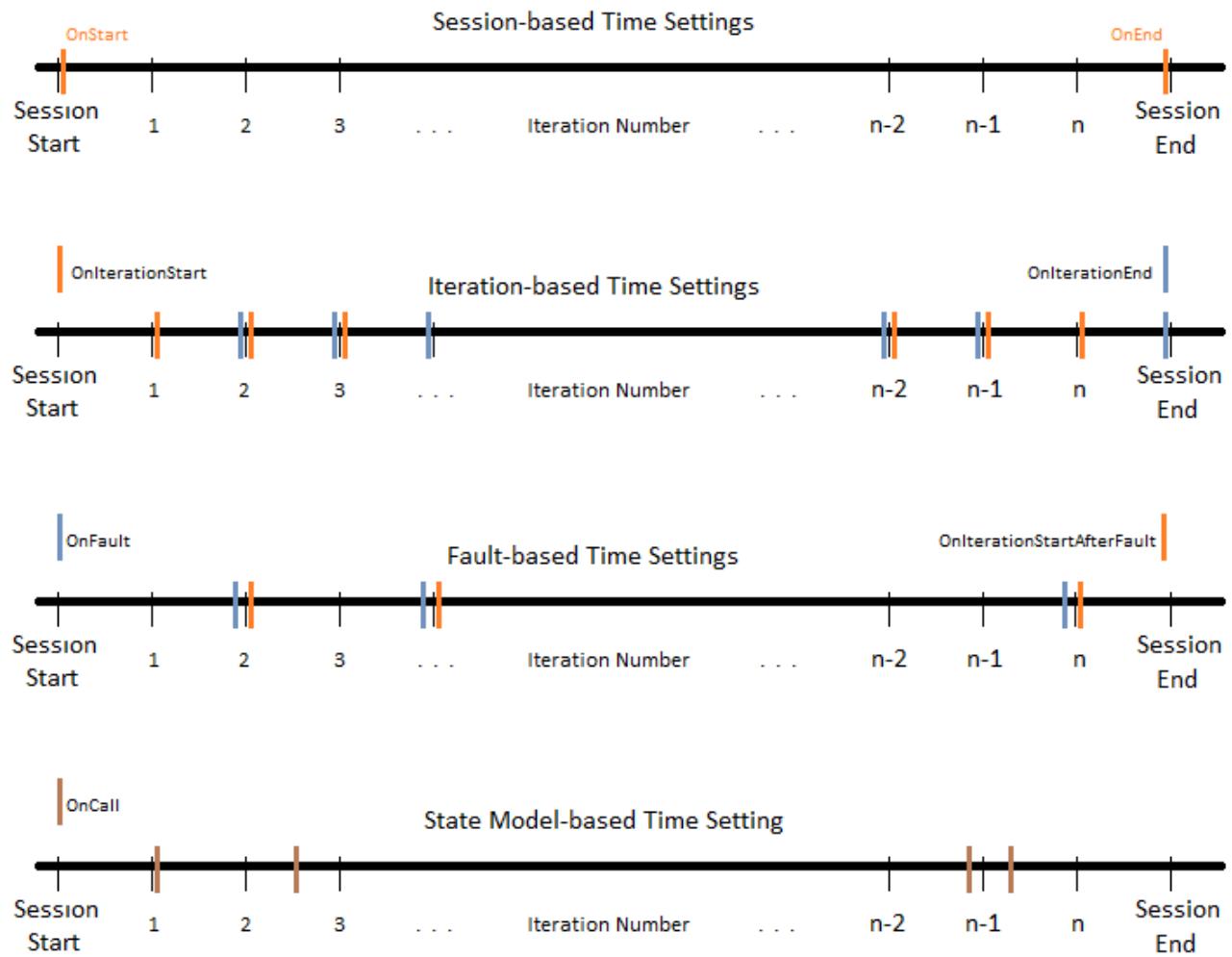


Figure 12. When Choices

Examples

Example 239. Automation example: Open

This example causes Peach to wait until the remote port is in an open state after the `WaitForPort` call is received from the state model. No timeout interval is provided, so Peach will wait forever.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <Number size="32" signed="false" value="31337" />
    </DataModel>

    <StateModel name="State" initialState="Initial" >
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>

            <Action type="call" method="WaitForPort" publisher="Peach.Agent" />
        </State>
    </StateModel>

    <Agent name="Local">
        <Monitor class="TcpPort">
            <Param name="Host" value="192.168.133.4" />
            <Param name="Port" value="502" />
            <Param name="WaitOnCall" value="WaitForPort" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <StateModel ref="State"/>
        <Agent ref="Local" />
        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>
```

Example 240. Automation example: Closed

This example causes Peach to wait until the remote port is in a closed state after the `WaitForPort` call is received from the state model. No timeout interval is provided, so Peach will wait forever.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <Number size="32" signed="false" value="31337" />
    </DataModel>

    <StateModel name="State" initialState="Initial" >
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>

            <Action type="call" method="WaitForPort" publisher="Peach.Agent" />
        </State>
    </StateModel>

    <Agent name="Local">
        <Monitor class="TcpPort">
            <Param name="Host" value="192.168.133.4" />
            <Param name="Port" value="502" />
            <Param name="State" value="Closed" />
            <Param name="WaitOnCall" value="WaitForPort" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <StateModel ref="State"/>
        <Agent ref="Local" />
        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>
```

Example 241. Fault Detection example

This example inspects the state of the remote port at the end of an iteration. A fault is generated if the port is closed at the end of an iteration.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <Number size="32" signed="false" value="31337" />
    </DataModel>

    <StateModel name="State" initialState="Initial" >
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="Local">
        <Monitor class="TcpPort">
            <Param name="Host" value="192.168.133.4" />
            <Param name="Port" value="502" />
            <Param name="Action" value="Fault" />
            <Param name="State" value="Closed" />
        </Monitor>
    </Agent>

    <Test name="Default">
        <StateModel ref="State"/>
        <Agent ref="Local" />
        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>
```

Example 242. Data Collection example

This example causes the TcpPort monitor to report the state of a port after a fault is detected by any other monitor.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <Number size="32" signed="false" value="31337" />
    </DataModel>

    <StateModel name="State" initialState="Initial" >
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="Local">
        <!-- Fault detection -->
        <Monitor class="Gdb">
            <Param name="Executable" value="/usr/bin/curl"/>
            <Param name="Arguments" value="http://localhost"/>
            <Param name="StartOnCall" value="ScoobySnacks"/>
        </Monitor>

        <!-- Data collection -->
        <Monitor class="TcpPort">
            <Param name="Host" value="192.168.133.4" />
            <Param name="Port" value="502" />
            <Param name="Action" value="Data" />
        </Monitor>
    </Agent>

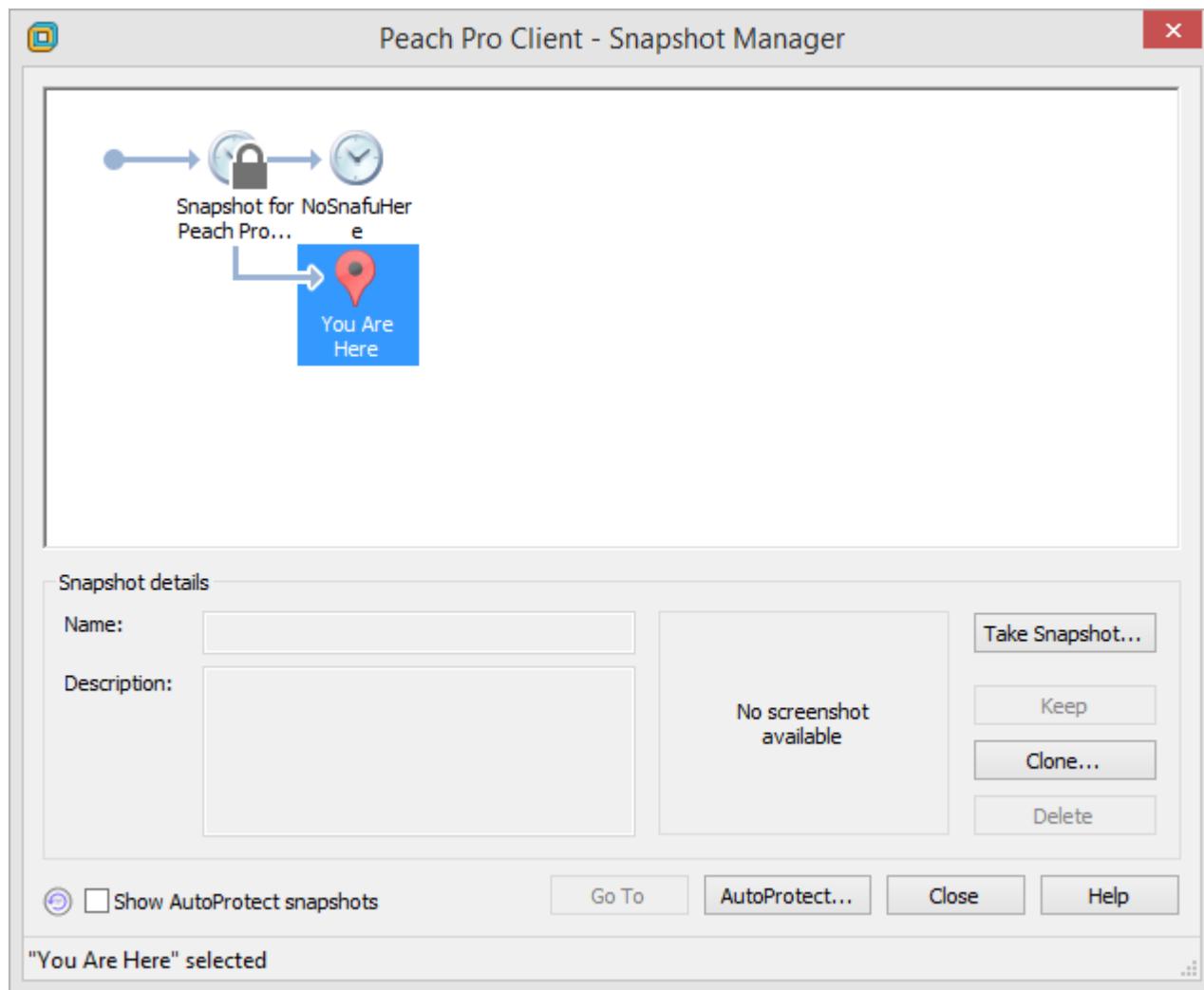
    <Test name="Default">
        <StateModel ref="State"/>
        <Agent ref="Local" />
        <Publisher class="ConsoleHex"/>
    </Test>
</Peach>
```

21.9.35. Vmware Monitor

Monitor Categories: Automation

The *Vmware* monitor can control a VMware virtual machine (VM). This monitor works with a snapshot of the VM that provides a consistent VM environment throughout a fuzzing session. The *Vmware* monitor can start a VM snapshot and, optionally, reset the VM to a snapshot configuration each iteration.

The following illustration shows the VMWare Snapshot Manager, which is used for managing snapshots for a particular VM.



Parameters

Required:

Vmx

Path to the virtual machine.



When using with vSphere/ESXi, prefix the VM image name with the storage location in brackets. For example, **[ha-datacenter/datastore1] guest/guest.vmx**.

Optional:

SnapshotName

VM snapshot name.

Either *SnapshotName* or *SnapshotIndex* must be specified, but it is an error to specify both.

SnapshotIndex

VM snapshot index specification.

Either *SnapshotName* or *SnapshotIndex* must be specified, but it is an error to specify both.

The index specification is a list of zero-based index values delimited by a period (.). The specification resolves which leaf in a tree of snapshots should be used.

For example, in the following tree of snapshots, the snapshot named **Snapshot 1.2.1** would be used when **0.1.0** is specified.

- Snapshot 1
 - Snapshot 1.1
 - Snapshot 1.2
 - **Snapshot 1.2.1**
 - Snapshot 1.2.2
- Snapshot 2
 - Snapshot 2.1
 - Snapshot 2.2

Host

Hostname or IP address the VMware host.

HostPort

TCP/IP port of the VMware host.

Login

Username for authentication with the VMware host.

Password

Password for authentication with the VMware host.

HostType

Type of remote host, defaults to **Default**

VM Product	Description
VIserver	vCenter Server, ESX/ESXi hosts, VMWare Server 2.0
Workstation	VMWare Workstation
WorkstationShared	VMWare Workstation (Shared Mode)
Player	VMWare Player
Server	VMWare Server 1.0.x
Default	Default

ResetEveryIteration

If **true** is specified, reset the VM on every iteration. Defaults to **false**.

ResetOnFaultBeforeCollection

If **true** is specified, reset the VM after a fault is detected by any other monitor. Defaults to **false**.

StopOnFaultBeforeCollection

If **true** is specified, stop the VM after a fault is detected by any other monitor. Defaults to **false**.

WaitForToolsInGuest

If **true** is specified, wait for VMware tools to start within the guest whenever a VM is restarted. Defaults to **true**.

WaitTimeout

The number of seconds to wait for VMware tools to start within a guest. Defaults to **600**.

Headless

Run a VM without a GUI. Using this parameter can improve performance but may cause issues if the target interacts with the desktop. Defaults to **true**.

Examples

Example 243. Start Virtual Machine

```
<Agent name="Local">
  <Monitor class="Vmware">
    <Param name="Vmx" value="D:\VirtualMachines\OfficeWebTest\OfficeWebTest.vmx"
  />
    <Param name="HostType" value="Workstation" />
    <Param name="SnapshotName" value="Fuzzing" />
  </Monitor>
</Agent>
```

Example 244. Start Virtual Machine hosted on ESXi

```
<Agent name="Local">
  <Monitor class="Vmware">
    <Param name="Vmx" value="[ha-datacenter/datastore1] guest/guest.vmx" />
    <Param name="SnapshotName" value="Fuzzing" />
  </Monitor>
</Agent>
```

21.9.36. WindowsDebugger Monitor (Windows)

Monitor Categories: Automation, Data collection, Fault detection

The *WindowsDebugger* monitor controls a windows debugger instance.

This monitor launches an executable file, a process, or a service with the debugger attached; or, this monitor can attach the debugger to a running executable, process, or service.

The *WindowsDebugger* performs automation, fault detection, and data collection.

- Automation manages when the fuzzing target (process, service, etc) starts and restarts. This can occur at the start of a fuzzing session, at the start of each iteration, after detecting a fault, or upon receiving a call from the state model.
- Fault detection watches and reports crashes and faults—when the target exits prematurely, and when the target fails to exit.
- Data collection retrieves stack traces, logs, and other information provided by the debugger. Note that this monitor provides bucket information—major and minor hash values—as part of data collected on faults. For more information on bucketing provided by this monitor, see [!exploitable](#).

Parameters

Exactly one of the three required parameters is needed for each instance of this monitor. The other two required parameters are not used.

Executable

Executable to launch via the debugger. If the executable has command-line arguments, specify these in the "Arguments" parameter.

ProcessName

Name of the process that the debugger attaches.

Service

Name of the Windows process or service to attach to the debugger. If the service crashes or is stopped, this monitor will restart the service.

Optional:

Arguments

Command-line arguments for the executable file specified with the "Executable" parameter.

SymbolsPath

Path to the debugging symbol files. The default value is Microsoft public symbols server, SRV*<http://msdl.microsoft.com/download/symbols>.

WinDbgPath

Path to the Windows Debugger installation. If undeclared, Peach attempts to locate a local installation of the debugger.

StartOnCall

Defers launching the target until the state model issues a call to the monitor to begin. Upon receiving the call, the debugger attaches to the process, or starts the process or executable.

IgnoreFirstChanceGuardPage

Ignores faults from the first chance guard page. These faults are sometimes false positives or anti-debugging faults, defaults to false.

IgnoreSecondChanceGuardPage

Ignores faults from the second chance guard page. These faults are sometimes false positives or anti-debugging faults, defaults to false.

IgnoreFirstChanceReadAv

Ignores faults from the first chance read access violations. These faults are sometimes false positives, defaults to false. When monitoring a Java process, it is recommended to set this parameter to true.

NoCpuKill

Allows or disallows the CPU to idle. If true, the CPU can idle without terminating the target. If false, Peach polls and then terminates the target if it is caught idling. The default value is false.

CpuPollInterval

Specifies the time interval, expressed in milliseconds (ms), that the monitor waits between successive polls of the target. This argument is used when NoCpuKill is false. The default value is 200 ms.

FaultOnEarlyExit

Triggers a fault if the target exits prematurely, defaults to false.

RestartAfterFault

If "true", restarts the target when any monitor detects a fault. If "false", restarts the target only if the process exits or crashes.

This argument defaults to true.

RestartOnEachTest

Restarts the process for each iteration, defaults to false.

ServiceStartTimeout

When debugging a windows service, this specifies how long the monitor should wait for the service to start. The default value is 60 seconds.

WaitForExitOnCall

Exits the target upon receiving a call from the state model. If the call fails to occur within an acceptable waiting period, issue a fault and then exit. The WaitForExitTimeout parameter specifies the waiting period.

WaitForExitTimeout

Specifies the WaitForExitOnCall timeout value, expressed in milliseconds, defaults to 10000 ms (10 sec). The value -1 specifies an infinite waiting period.

Examples

Example 245. Command Line Configuration

```
<Agent name="Local">
  <Monitor class="WindowsDebugger">
    <Param name="Executable" value="CrashableServer.exe" />
    <Param name="Arguments" value="127.0.0.1 4244" />
    <!--<Param name="WinDbgPath" value="C:\Program Files (x86)\Debugging Tools
for Windows (x86)" />-->
  </Monitor>
</Agent>
```

Example 246. Service Configuration

```
<Param name="Service" value="WinDefend" />
```

Example 247. Process Configuration

```
<Param name="ProcessName" value="CrashableServer.exe" />
```

Example 248. StartOnCall Configuration

```
<StateModel name="TheState" initialState="initial">
    <State name="initial">
        <Action type="call" method="launchProgram" publisher="Peach.Agent"/>
    </State>
</StateModel>

<Agent name="Local">
    <Monitor class="WindowsDebugger">
        <Param name="Executable" value="CrashableServer.exe"/>
        <Param name="Arguments" value="127.0.0.1 4244"/>
        <Param name="StartOnCall" value="launchProgram"/>
    </Monitor>
</Agent>
```

Example 249. Exit Configurations

```
<Agent name="Local">
    <Monitor class="WindowsDebugger">
        <Param name="Executable" value="CrashableServer.exe"/>
        <Param name="Arguments" value="127.0.0.1 4244"/>
        <Param name="NoCpuKill" value="true"/>
        <Param name="FaultOnEarlyExit" value="false"/>
        <Param name="WaitForExitTimeout" value="250"/>
    </Monitor>
</Agent>
```

Example 250. WaitForExitOnCall Configuration

```
<StateModel name="TheState" initialState="initial">
    <State name="initial">
        <Action type="call" method="exitProgram" publisher="Peach.Agent"/>
    </State>
</StateModel>

<Agent name="Local">
    <Monitor class="WindowsDebugger">
        <Param name="Executable" value="CrashableServer.exe"/>
        <Param name="Arguments" value="127.0.0.1 4244"/>
        <Param name="WaitForExitOnCall" value="exitProgram"/>
    </Monitor>
</Agent>
```

21.9.37. WindowsKernelDebugger Monitor (Windows)

Monitor Categories: Data collection, Fault detection

The *WindowsKernelDebugger* monitor controls debugging of a remote windows kernel debugging instance. At least two machines are involved:

- The target machine that receives fuzzing data.
- The host machine that controls the debugging session. Peach resides on this machine

The *WindowsKernelDebugger* performs fault detection, and data collection.

- Fault detection watches and reports crashes and faults—when the target exits prematurely, and when the target fails to exit.
- Data collection retrieves stack traces, logs, and other information provided by the debugger. Note that this monitor provides bucket information—major and minor hash values—as part of data collected on faults. For more information on bucketing provided by this monitor, see [!exploitable](#).

The workflow for this configuration follows:

1. On the host machine, start Peach with the WindowsKernelDebugger Monitor.
This action starts the fuzzing and loads a copy of Windbg in kernel mode. The host machine enters a waiting state because the target machine will initiate and secure the connection for the debugging session.
2. On the target machine, open Windbg, and choose "Connect to Remote Session" from the File menu.
 - For the "Connection String", specify the "transport:port" and the server for the debugging session, as in this example: `net:port=5000,key=1.2.3.4`.
 - Click OK.

The debug session starts on the target machine. The target machine initiates contact with the host to establish the connection. Once the connection is secure, the host machine (with Peach) drives the activities.

 Instrumenting this configuration requires additional monitors external to the target. The monitors need to be able to restart the target after a crash, and to have the host resume fuzzing.

Parameters

KernelConnectionString

Connection string for attaching the debugger to a kernel-mode process (e.g. a driver). The connection string `net:port=5000,key=1.2.3.4` indicates this is the debug server and uses the tcp transport on port 5000. The Windows debugger offers a choice of transports; and, the port is arbitrary. For more information on using other values, see MSDN articles on live kernel mode,

debugging, remote debugging, and kernel connection strings.

Optional:

SymbolsPath

Path to the debugging symbol files. The default value is Microsoft public symbols server
`SRV*http://msdl.microsoft.com/download/symbols`.

WinDbgPath

Path to the windbg installation. If undeclared, Peach attempts to locate a local installation of the debugger.

ConnectTimeout

Duration, in milliseconds, to wait to receive a kernel connection. The default value is 3000 ms.

21.9.38. WindowsService Monitor (Windows)

Monitor Categories: Automation, Fault detection

The *WindowsService* monitor controls a windows service. When the monitor runs, it checks the state of the service. If the service exits prematurely, *WindowsService* generates a fault. If a fault is detected by from *windowsService* or from another monitor, the monitor collects status of the service and continues.

If the service is not running, this monitor attempts to restore the service to a running state, whether starting, restarting, or resuming from a paused state.

Parameters

Required:

Service

Name that identifies the service to the system, such as the display name of the service.

Optional:

FaultOnEarlyExit

Fault if the service exits early, defaults to false.

MachineName

Name of the computer on which the service resides, defaults to local machine.

Restart

Specifies whether to start the service on each iteration, defaults to false.

StartTimeout

Specifies the duration, in minutes, to wait for the service to start, defaults to 1 minute.

Examples

Example 251. Start IIS

```
<Agent name="Local">
  <Monitor class="WindowsService">
    <Param name="Service" value="World Wide Web Publishing Service" />
  </Monitor>
</Agent>
```

21.10. Publishers

Publishers are the I/O interfaces Peach uses to send and receive data.

Publishers support both stream and call based operations.

When the fuzzer is running, all [Actions](#) (except `changeState` and `slurp`) use a publisher to perform the action.

Different publishers support a different set of Action types. For example, the [File](#) publisher supports input for reading from a file, output for writing to a file, but does not support `accept` or `call`. This differs from the [COM](#) publisher which supports `call`, but not `input`, `output`, or `accept`.

All fuzzing definitions must use at least one Publisher and (optionally) can use multiple Publishers. When using multiple Publishers, each Action must specify which Publisher it is referencing by including the Publisher's `name` attribute in the Action's `publisher` attribute. If the `publisher` attribute is missing, Peach performs the Action using the first Publisher defined in the Test.

21.10.1. Network Publishers

When fuzzing network protocols, publishers typically uses the protocol that encompasses (operates one layer below) the target protocol. For example, when fuzzing the HTTP protocol, use the [TCP](#) publisher. When fuzzing TCP, use either the [RawV4](#) or [RawV6](#) publisher. When fuzzing IPv4 and IPv6, use the [RawEther](#) publisher.

21.10.2. Publishers

- [AndroidMonkey](#)
- [CAN](#)
- [Com](#)
- [Console](#)
- [ConsoleHex](#)
- [File](#)
- [FilePerIteration](#)
- [Http](#)
- [I2C](#)
- [RawEther](#)
- [RawIPv4](#)
- [RawV4](#)
- [RawV6](#)

- [Remote](#)
- [Rest](#)
- [SerialPort](#)
- [SslClient](#)
- [SslListener](#)
- [TcpClient](#)
- [TcpListener](#)
- [Udp](#)
- [Usb](#)
- [WebApi](#)
- [WebService](#)
- [WebSocket](#)
- [Zip](#)



A full list of publishers and parameter information for any specific version can be found in the output of `peach --showenv`

21.10.3. Syntax

```
<Test name="Default">
<!-- ... -->
  <Publisher class="ConsoleHex">
    <Param name="BytesPerLine" value="32"/>
  </Publisher>
<!-- ... -->
</Test>
```

21.10.4. Examples

Example 252. Multiple Publishers

Many of the licensed Pits use a single publisher; and, some Pits use multiple publishers. Peach imposes no limit on the number of publishers to use in a Pit.

For example, you might use multiple publishers to communicate with a network service that listens on multiple ports:

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

    <DataModel name="Hello">
        <Blob value="Hello Server"/>
    </DataModel>

    <DataModel name="Goodbye">
        <Blob value="Goodbye Server"/>
    </DataModel>

    <StateModel name="TheState" initialState="Initial">
        <State name="Initial">
            <Action type="output" publisher="FirstPort">
                <DataModel ref="Hello"/>
            </Action>
            <Action type="output" publisher="SecondPort">
                <DataModel ref="Goodbye"/>
            </Action>
        </State>
    </StateModel>
    <Agent name="TheAgent" />
    <Test name="Default">
        <Agent ref="TheAgent"/>
        <StateModel ref="TheState"/>
        <Publisher class="Tcp" name="FirstPort">
            <Param name="Host" value="localhost"/>
            <Param name="Port" value="12345"/>
        </Publisher>
        <Publisher class="Tcp" name="SecondPort">
            <Param name="Host" value="localhost"/>
            <Param name="Port" value="54321"/>
        </Publisher>
    </Test>
</Peach>

```

```
$ peach -1 --debug TwoPublisher.xml

[*] Test 'Default' starting with random seed 9324.
Peach.Core.MutationStrategies.RandomStrategy Iteration: Switch iteration, setting
controlIteration and controlRecordingIteration.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Updating action to original data model
Peach.Core.Dom.Action Updating action to original data model
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpClientPublisher start()
Peach.Core.Publishers.TcpClientPublisher open()
Peach.Core.Publishers.TcpClientPublisher output(12 bytes)
Peach.Core.Publishers.TcpClientPublisher

00000000  48 65 6C 6C 6F 20 53 65  72 76 65 72          Hello Server

Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.TcpClientPublisher start()
Peach.Core.Publishers.TcpClientPublisher open()
Peach.Core.Publishers.TcpClientPublisher output(14 bytes)
Peach.Core.Publishers.TcpClientPublisher

00000000  47 6F 6F 64 62 79 65 20  53 65 72 76 65 72          Goodbye Server

Peach.Core.Publishers.TcpClientPublisher close()
Peach.Core.Publishers.TcpClientPublisher Shutting down connection to 127.0.0.1:12345
Peach.Core.Publishers.TcpClientPublisher Read 0 bytes from 127.0.0.1:12345, closing
client connection.
Peach.Core.Publishers.TcpClientPublisher Closing connection to 127.0.0.1:12345
Peach.Core.Publishers.TcpClientPublisher close()
Peach.Core.Publishers.TcpClientPublisher Shutting down connection to 127.0.0.1:54321
Peach.Core.Publishers.TcpClientPublisher Read 0 bytes from 127.0.0.1:54321, closing
client connection.
Peach.Core.Publishers.TcpClientPublisher Closing connection to 127.0.0.1:54321
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.TcpClientPublisher stop()
Peach.Core.Publishers.TcpClientPublisher stop()

[*] Test 'Default' finished.
```

21.10.5. AndroidMonkey Publisher

The *AndroidMonkey* publisher is used to communicate with both Android emulators and physicals devices.

AndroidMonkey allows you to fuzz text inputs, screen touches and key presses. By using additional Android monitors, you can target specific Android applications for fuzzing and monitoring.

Syntax

```
<Publisher class="AndroidMonkey">
    <Param name="DeviceMonitor" value="App"/>
</Publisher>
```

Parameters

Required:

One of the following parameters is required.

DeviceSerial

The serial of the device to fuzz.

DeviceMonitor

Android monitor to get device serial from.

Optional:

AdbPath

Directory containing adb.

ConnectTimeout

Max seconds to wait for adb connection. Defaults to 5.

CommandTimeout

Max seconds to wait for adb command to complete. Defaults to 10.

Actions

start

Implicit Action to start the Publisher.

stop

Implicit Action to stop the Publisher.

call

This publisher supports several methods that result in different user interactions occurring.

tap

Specify on an X,Y coordinate on the screen. Requires two parameters providing a numerical value for X and Y.

keyevent

Send a key press to the target device. Requires a single string parameter. The parameter corresponds to the keyevent byte code used to for android key mapping. The entire list of android keyevent codes can be found [here](#).

text

Enter in text into a text field. Requires a single parameter with the text to enter.

Examples

Example 253. Basic Usage Example

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="TheDataModel">
    <Number size='32' signed="false" value="31337" />
</DataModel>

<DataModel name="X">
    <Number size='32' signed="false" value="230" />
</DataModel>

<DataModel name="Y">
    <Number size='32' signed="false" value="150" />
</DataModel>

<StateModel name="State" initialState="Initial" >
    <State name="Initial" >
        <Action type="call" method="OpenApplication" publisher="Peach.Agent"/>

        <Action type="call" method="tap">
            <Param>
                <DataModel ref="X"/>
            </Param>
            <Param>
                <DataModel ref="Y"/>
            </Param>
        </Action>
    </State>
</StateModel>
```

```

        </Param>
    </Action>
</State>
</StateModel>

<Agent name="TheAgent">
    <Monitor name="Emu" class="AndroidEmulator">
        <Param name="Avd" value="unused" />
        <Param name="EmulatorPath" value="C:\adt-bundle-windows-x86_64-20131030\sdk\tools"/>
    </Monitor>

    <Monitor name="App" class="Android">
        <Param name="ApplicationName" value="com.android.development" />
        <Param name="ActivityName" value=".BadBehaviorActivity" />
        <Param name="AdbPath" value="C:\adt-bundle-windows-x86_64-20131030\sdk\platform-tools"/>
        <Param name="StartOnCall" value="OpenApplication"/>
        <Param name="DeviceMonitor" value="Emu" />
    </Monitor>
</Agent>

<Test name="Default">
    <StateModel ref="State"/>
    <Agent ref="TheAgent" />

    <Publisher class="AndroidMonkey">
        <Param name="DeviceMonitor" value="App"/>
    </Publisher>
</Test>
</Peach>

```

21.10.6. CAN Publisher

Use the *CAN* Publisher to read and write bytes directly to and from a CAN bus.

CAN is designed to interface with the TotalPhase Komodo CAN SOLO or DUO (USB) Host Adapter.

Syntax

```
<Publisher class="CAN">
  <Param name="ID" value="5"/>
</Publisher>
```

Parameters

Required:

ID

The CAN identifier used for transmission and reception

Optional:

RTR

Is the frame requesting the transmission of a specific identifier. This parameter only effects output and does not effect input.

ExtendedFrame

Should extended frame format be used. This parameter only effects output and does not effect input.

SerialNumber

Serial number of Komodo CAN Interface to use. If omitted the first available interface will be used.

Bitrate

Controls the bitrate used for transmission and reception of CAN frames. Defaults to 125000 bps.

Timeout

How long to wait in milliseconds for incoming data to arrive. Defaults to 3000 ms.

SendTimeout

How long to wait in milliseconds for outgoing data to send. Defaults to 3000 ms.

Actions

start

Open and initialize the Komodo device.

stop

Close and clean up the Komodo device.

open

Enable the CAN port on the Komodo device.

close

Disable the CAN port on the Komodo device.

output

Data sent via output is written to the CAN bus.

input

Data received via input is read from the CAN bus.

Examples

Example 254. Read ADC from TotalPhase CAN Activity Board Pro

```
<Peach>
  <StateModel name='SM' initialState='Initial'>
    <State name='Initial'>
      <!-- Send query to ADC -->
      <Action type='output'>
        <DataModel name='DM' />
      </Action>

      <!-- Read response -->
      <Action type='input'>
        <DataModel name='DM'>
          <Blob length='3' />
        </DataModel>
      </Action>
    </State>
  </StateModel>

  <Test name='Default'>
    <StateModel ref='SM' />
    <Publisher class='CAN'>
      <Param name='ID' value='66' />
      <Param name='RTR' value='true' />
    </Publisher>
  </Test>
</Peach>
```

Example 255. Write LCD on TotalPhase CAN Activity Board Pro

```
<Peach>
  <DataModel name='DM'>
    <Number name='index' size='8' />
    <String name='value' length='4' />
  </DataModel>

  <StateModel name='SM' initialState='Initial'>
    <State name='Initial'>
      <Action type='output'>
        <DataModel ref='DM' />
        <Data>
          <Field name='index' value='0' />
          <Field name='value' value='abcd' />
        </Data>
      </Action>
      <Action type='output'>
        <DataModel ref='DM' />
        <Data>
          <Field name='index' value='1' />
          <Field name='value' value='efgh' />
        </Data>
      </Action>
    </State>
  </StateModel>

  <Test name='Default'>
    <StateModel ref='SM' />
    <Publisher class='CAN'>
      <Param name='ID' value='322' />
    </Publisher>
  </Test>
</Peach>
```

21.10.7. Com Publisher

The *Com* Publisher allows calling methods and properties on COM objects. Properties and Methods correlate directly to those in the corresponding COM definition.

The [setProperty](#) and [getProperty](#) Actions can only accept Strings or Numbers because binary arrays are incompatible with COM Properties.



Com Publisher only runs on Windows.

To use *Com*, Peach.Core.ComContainer.exe must be running in a separate command prompt (so it can host the COM object) while the Peach is running.

You can configure a debugger monitor to launch and monitor this process. See the [WindowsDebugger](#) monitor for more information about configuring a debugger monitor.

Syntax

```
<Publisher class="COM">
    <Param name="clsid" value="{d20ea4e1-3957-11d2-a40b-0c5020524153}" />
</Publisher>
```

Parameters

Required:

clsid

COM CLSID

Optional:

There are no optional parameters for this publisher.

Actions

start

Implicit Action to start the Publisher.

stop

Implicit Action to stop the Publisher.

call

Call a method

getProperty

Get a property value

setProperty

Set a property value

Examples

Example 256. Calling a method

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="CoolThingsModel">
        <String name="Username" value="username=user"/>
        <String name="Comma" value=","/>
        <String name="Password" value="password=admin"/>
    </DataModel>

    <StateModel name="TheStateModel">
        <State name="initial">
            <Action type="call" method="DoCoolThings">
                <Param name="AuthString">
                    <DataModel ref="CoolThingsModel"/>
                </Param>
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref=="TheStateModel" />
        <Publisher class="COM">
            <Param name="clsid" value="{d20ea4e1-3957-11d2-a40b-0c5020524153}" />
        </Publisher>
    </Test>
</Peach>
```

Example 257. Setting/Getting Properties

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

    <DataModel name="TheDataModel">
        <String name="Value" />
    </DataModel>

    <StateModel name="TheState" initialState="Initial">
        <State name="Initial">

            <Action type="call" method="Players[1].OpenUrl">
                <Param name="P1">
                    <DataModel ref="TheDataModel" />
                    <Data>
                        <Field name="Value" value="C:\labs\com\boyngirl.mov"/>
                    </Data>
                </Param>
            </Action>

            <Action type="setProperty" property="Players[1].QTControl.Movie.Height">
                <DataModel ref="TheDataModel" />
                <Data>
                    <Field name="Value" value="200"/>
                </Data>
            </Action>

            <Action type="setProperty" property="Players[1].QTControl.Movie.Width">
                <DataModel ref="TheDataModel" />
                <Data>
                    <Field name="Value" value="500"/>
                </Data>
            </Action>

            <Action type="call" method="Players[1].QTControl.Movie.Play" />
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>
        <Publisher class="Com">
            <Param name="clsid" value="QuickTimePlayerLib.QuickTimePlayerApp"/>
        </Publisher>
    </Test>
</Peach>

```

21.10.8. Console Publisher

The *Console* publisher outputs data to standard out.

Syntax

```
<Publisher class="Console" />
```

Parameters

There are no parameters for this publisher.

Actions

open

Initialize stream to standard out.

close

Close stream to standard out.

output

Data sent via output is written to the console.

Examples

Example 258. Display data to console

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="Data">
  <String name="Start" value="Start"/>
  <Blob name="Data" valueType="hex" value="BEEFEA7E41"/>
  <String name="Stop" value="Stop"/>
</DataModel>

<StateModel name="TheState" initialState="initial">
  <State name="initial">
    <Action type="output">
      <DataModel ref="Data" />
    </Action>
  </State>
</StateModel>

<Test name="Default">
  <StateModel ref="TheState"/>
  <Publisher class="Console" />
</Test>
</Peach>
```

21.10.9. ConsoleHex Publisher

The *ConsoleHex* publisher outputs data to standard out. The data is displayed in hex format.

Syntax

```
<Publisher class="ConsoleHex"/>
```

Parameters

Required:

There are no required parameters for this publisher.

Optional:

BytesPerLine

Number of bytes per row of text (optional, defaults to 16)

Actions

open

Initialize stream to standard out.

close

Close stream to standard out.

output

Data sent via output is written to the console.

Examples

Example 259. Display data to console

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="Data">
  <String name="Start" value="Start"/>
  <Blob name="Data" valueType="hex" value="BEEFEA7E41"/>
  <String name="Stop" value="Stop"/>
</DataModel>

<StateModel name="TheState" initialState="initial">
  <State name="initial">
    <Action type="output">
      <DataModel ref="Data" />
    </Action>
  </State>
</StateModel>

<Test name="Default">
  <StateModel ref="TheState" />
  <Publisher class="ConsoleHex"/>
</Test>
</Peach>
```

Example 260. Display data with custom bytes per line

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="Data">
  <String name="Start" value="Start"/>
  <Blob name="Data" valueType="hex" value="BEEFEA7E41"/>
  <String name="Stop" value="Stop"/>
</DataModel>

<StateModel name="TheState" initialState="initial">
  <State name="initial">
    <Action type="output">
      <DataModel ref="Data" />
    </Action>
  </State>
</StateModel>

<Test name="Default">
  <StateModel ref="TheState"/>
  <Publisher class="ConsoleHex">
    <Param name="BytesPerLine" value="8" />
  </Publisher>
</Test>
</Peach>
```

21.10.10. File Publisher

The *File* publisher opens a file for reading or writing.

Syntax

```
<Publisher class="File">
    <Param name="FileName" value="fuzzed.bin" />
</Publisher>
```

Parameters

Required:

FileName

Name of file to open

Optional:

Overwrite

Overwrite existing files, defaults to true.

Append

Append data to existing file, defaults to false.

Actions

open

Open file for reading/writing.

close

Close file stream.

output

Data to be written to file

input

Read data from file.

Examples

Example 261. Write to file

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

<DataModel name="TheDataModel">
    <Number name="Magic" size="8" value="47" token="true"/>
    <Number name="Length" size="8">
        <Relation type="size" of="Data"/>
    </Number>
    <Blob name="Data"/>
</DataModel>

<StateModel name="TheState" initialState="initial">
    <State name="initial">
        <Action type="output">
            <DataModel ref="TheDataModel" />
            <Data fileName="file.bin" />
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>
    <Publisher class="File">
        <Param name="FileName" value="fuzzed.bin" />
    </Publisher>
</Test>
</Peach>
```

Example 262. Read from file

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

<DataModel name="TheDataModel">
    <Number name="Magic" size="8" value="47" token="true"/>
    <Number name="Length" size="8">
        <Relation type="size" of="Data"/>
    </Number>
    <Blob name="Data"/>
</DataModel>

<StateModel name="TheState" initialState="initial">
    <State name="initial">
        <Action type="input">
            <DataModel ref="TheDataModel" />
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>
    <Publisher class="File">
        <Param name="FileName" value="file.bin" />
        <Param name="Overwrite" value="False"/>
    </Publisher>
</Test>
</Peach>
```

21.10.11. FilePerIteration Publisher

The *FilePerIteration* publisher creates an output file for each fuzzer iteration.

Use *FilePerIteration* when pre-generating fuzzing cases.

When generating the fuzzed files, Peach appends the iteration number to the fuzzed files name. For a file name format of "fuzzed_{0}.bin" the {0} will be replaced with the current iteration number which produces the following filenames: fuzzed_1.bin, fuzzed_2.bin, fuzzed_3.bin, fuzzed_4.bin, etc.

Syntax

```
<Publisher class="FilePerIteration">
    <Param name="FileName" value="fuzzed_{0}.bin" />
</Publisher>
```

Parameters

Required:

FileName

Name of file to create. Filename must contain "{0}" which will be substituted with the iteration count.

Optional:

There are no optional parameters for this publisher.

Actions

open

Open file for reading/writing

close

Close file stream

output

Data to be written to file

input

Read data from file

Examples

Example 263. Basic Usage Example

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="TheDataModel">
    <Number name="Magic" size="8" value="47" token="true"/>
    <Number name="Length" size="8">
        <Relation type="size" of="Data"/>
    </Number>
    <Blob name="Data"/>
</DataModel>

<StateModel name="TheState" initialState="initial">
    <State name="initial">
        <Action type="output">
            <DataModel ref="TheDataModel" />
            <Data fileName="file.bin"/>
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>
    <Publisher class="FilePerIteration">
        <Param name="FileName" value="fuzzed_{0}.bin" />
    </Publisher>
</Test>
</Peach>
```

21.10.12. Http Publisher (Deprecated)

This publisher is deprecated, please see [WebApi publisher](#).

Http publisher sends data over HTTP via a valid HTTP method type.

Http supports the following features:

- Authentication via Basic, Digest, or Windows integrated
- Definable method type
- Fuzzing and dynamic setting of headers (both key and value)
- Fuzzing and dynamic setting of query strings
- Optional cookie support
- SSL

See also the [WebApi](#) publisher.

Syntax

```
<Publisher class="Http">
    <Param name="Method" value="POST" />
    <Param name="Url" value="http://foo/user/create" />
</Publisher>
```

Parameters

Required:

Method

HTTP Method type (like GET and POST)

Url

URL of target

Optional:

BaseUrl

Base URL is used by some authentication types

Username

Username for authentication

Domain

Domain for authentication

Cookies

Enable cookie support. Defaults to true.

CookiesAcrossIterations

Track cookies across iterations. Defaults to false.

FailureStatusCodes

Comma separated list of status codes that are failures causing current test case to stop. Defaults to:
400,401,402,403,404,405,406,407,408,409,410,411,412,413,414,415,416,417,500,501,502,503,504,505

FaultOnStatusCodes

Comma separated list of status codes that are faults. Defaults to none.

Timeout

How long to wait in milliseconds for data/connection. Defaults to 3,000.

IgnoreCertErrors

Allow HTTPS regardless of cert status. Defaults to false.

Proxy

To use HTTP proxy, set the URL. Default is none. Example: <http://192.168.1.1:8080>.

The publisher will not use the default system proxy. If a proxy is required it must be explicitly set via the publisher parameter.

Please note that the host **localhost** and IP 127.0.0.1 will bypass the provided proxy. This is a behavior hardcoded into the underlying http networking code. For a discussion of options to deal with this limitation see the following article: [Fiddler - Monitoring Local Traffic](#).

Actions

call

Special method names used to fuzz the query string or a specific header.

Query

Specify as the method name for a call action, the first parameter is the query string

Header

Specify as the method name for a call action, the first parameter is the header name, the second is the value

start

Implicit Action to start the Publisher.

stop

Implicit Action to stop the Publisher.

open

Open and initialize the socket.

close

Close and clean up the socket.

output

Data sent via output is written to the open socket.

input

Data received via input is read from the open socket.

Scripting

The Http publisher exposes a public Headers dictionary that can be used to add/remove headers from Python scripting code. See the [WebApi](#) publisher examples.

Examples

Example 264. HTTP POST Request

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="PostBody">
  <String name="Username" value="username=user"/>
  <String name="Comma" value=","/>
  <String name="Password" value="password=admin"/>
</DataModel>

<StateModel name="TheState" initialState="initial">
  <State name="initial">
    <Action type="output">
      <DataModel ref="PostBody" />
    </Action>
  </State>
</StateModel>

<Test name="Default">
  <StateModel ref="TheState"/>
  <Publisher class="Http">
    <Param name="Method" value="POST" />
    <Param name="Url" value="http://foo/user/create" />
  </Publisher>
</Test>
</Peach>
```

Example 265. HTTP GET Request

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="TheDataModel">
    <Blob/>
</DataModel>

<StateModel name="TheState" initialState="initial">
    <State name="initial">
        <Action type="input" >
            <DataModel ref="TheDataModel" />
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>
    <Publisher class="Http">
        <Param name="Method" value="GET" />
        <Param name="Url" value="http://foo/user/create" />
    </Publisher>
</Test>
</Peach>
```

Example 266. Fuzzing HTTP GET Request Query String

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="QueryModel">
  <String value="key"/>
  <String value="=" token="true" />
  <String value="value"/>
</DataModel>

<DataModel name="GetInputModel">
  <Blob/>
</DataModel>

<StateModel name="TheState" initialState="initial">
  <State name="initial">
    <Action type="call" method="Query">
      <Param>
        <DataModel ref="QueryModel" />
      </Param>
    </Action>

    <Action type="input">
      <DataModel ref="GetInputModel"/>
    </Action>
  </State>
</StateModel>

<Test name="Default">
  <StateModel ref="TheState"/>
  <Publisher class="Http">
    <Param name="Method" value="GET" />
    <Param name="Url" value="http://foo/user/create" />
  </Publisher>
</Test>
</Peach>
```

Example 267. Fuzzing Cookie Value in Header

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="HeaderKey">
<String value="Cookie" />
</DataModel>

<DataModel name="HeaderValue">
<String value="user=newuesr" />
</DataModel>

<DataModel name="GetInputModel">
<Blob/>
</DataModel>

<StateModel name="TheState" initialState="initial">
<State name="initial">
<Action type="call" method="Header">
<Param>
<DataModel ref="HeaderKey" />
</Param>
<Param>
<DataModel ref="HeaderValue" />
</Param>
</Action>

<Action type="input">
<DataModel ref="GetInputModel"/>
</Action>
</State>
</StateModel>

<Test name="Default">
<StateModel ref="TheState"/>
<Publisher class="Http">
<Param name="Method" value="GET" />
<Param name="Url" value="http://foo/user/create" />
</Publisher>
</Test>
</Peach>
```

21.10.13. I2C Publisher

Use the *I2C* Publisher to read and write bytes directly to and from an I2C bus.

I2C is designed to interface with the Total Phase Aardvark I2C/SPI (USB) Host Adapter.

Syntax

```
<Publisher class="I2C">
  <Param name="Address" value="18"/>
  <Param name="SleepTime" value="5"/>
  <Param name="Bitrate" value="400"/>
</Publisher>
```

Parameters

Required:

Address

Address to write/read on target device.

Optional:

Bitrate

Bitrate supported by target device. Defaults to 100.

Port

USB port connected to Aardvark device. Defaults to 0.

SleepTime

Time to sleep between actions. Defaults to 100.

Actions

start

Implicit Action to start the Publisher.

stop

Implicit Action to stop the Publisher.

open

Open and initialize the aardvark handle.

close

Close and clean up the aardvark handle.

output

Data sent via output is written to the I2C bus.

input

Data received via input is read from the I2C bus.

Examples

Example 268. Basic Usage

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

<DataModel name="I2CModel">
    <Blob name="CtlCode" length="1" />
    <Number name="Length" size="8">
        <Relation type="size" of="Data" />
    </Number>
    <Blob name="Data" />
</DataModel>

<StateModel name="TheState" initialState="Initial">
    <State name="Initial">
        <Action type="open" />
        <Action type="output">
            <DataModel ref="I2C_Data_Reset"/>
            <Data>
                <Field name="CtlCode" value="1" />
                <Field name="Data" value="0x30313233"/>
            </Data>
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>

    <Publisher class="I2C">
        <Param name="Address" value="18"/>
        <Param name="SleepTime" value="5"/>
        <Param name="Bitrate" value="400"/>
    </Publisher>

    <Strategy class="Random" />
</Test>
</Peach>
```

21.10.14. RawEther Publisher

The *RawEther* publisher allows sending raw Ethernet packets.



This publisher requires administrative privileges.

Syntax

```
<Publisher class="RawEther">
    <Param name="Interface" value="eth0" />
</Publisher>
```

Parameters

Required:

Interface

Name of interface to bind to

Optional:

Timeout

Amount of time, in milliseconds, to wait for data or for a connection. The default value is 3,000.

MinMTU

Minimum packet size to transmit. The smallest value is 1280, which is the default value.

MaxMTU

maximum packet size to transmit. The largest value is 131070, which is the default value.

Filter

Filters the received frames using a libpcap-style filter string. For more information about libpcap style filters, see [this page](#).

Actions

start

Implicit Action to start the Publisher.

stop

Implicit Action to stop the Publisher.

open

Open and initialize the socket.

close

Close and clean up the socket.

output

Writes data through output to the open socket.

input

Reads data through input from the open socket.

Examples

Example 269. Sending data

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<Defaults>
    <Number endian="big"/>
</Defaults>

<DataModel name="TheDataModel">
    <Blob name="Dest" valueType="hex" mutable="false" length="6" value="FFFFFFFFFFFF"
/>
    <Blob name="Src" valueType="hex" mutable="false" length="6" value="FFFFFFFFFFFF
"/>
    <Number name="TypeOrLen" size="16" token="true" valueType="hex" value="0806"/>
    <Block name="Payload">
        <String name="name" value="Hello, scoobysnacks."/>
    </Block>
</DataModel>

<DataModel name="propertySize">
    <Number size="32" value="1500"/>
</DataModel>

<StateModel name="TheState" initialState="initial">
    <State name="initial">
        <Action type="setProperty" property="MaxMTU">
            <DataModel ref="propertySize"/>
        </Action>

        <Action type="setProperty" property="MinMTU">
            <DataModel ref="propertySize"/>
        </Action>
    </State>
</StateModel>
```

```
<Action type="output">
  <DataModel ref="TheDataModel"/>
</Action>
</State>
</StateModel>

<Test name="Default">
  <StateModel ref="TheState"/>
<Publisher class="RawEther">
  <Param name="Interface" value="eth0" />
  <Param name="Filter" value="ether proto 0x0806" />
</Publisher>
</Test>
</Peach>
```

21.10.15. RawIPv4 Publisher

The *RawIPv4* publisher sends raw IPv4 packets with IP headers.

Input received from *RawIPv4* includes the IPv4 header.

The *RawIPv4* maximum and minimum MTU size can be fuzzed by using the *setProperty* action.



This publisher runs best on Linux and requires root privileges.

Syntax

```
<Publisher class="RawV4">
    <Param name="Host" value="127.0.0.1" />
    <Param name="Protocol" value="17" />
</Publisher>
```

Parameters

Required:

Host

Hostname or IP address of remote host

Protocol

IP protocol to use (e.g. TCP)

Optional:

Interface

IP of interface to bind to

Timeout

How long to wait in milliseconds for data/connection. Defaults to 3,000.

MaxMTU

Maximum allowable MTU property value. Defaults to 131,070.

MinMTU

Minimum allowable MTU property value. Defaults to 1,280.

Actions

start

Implicit Action to start the Publisher.

stop

Implicit Action to stop the Publisher.

open

Open and initialize the socket.

close

Close and clean up the socket.

output

Data sent via output is written to the open socket.

input

Data received via input is read from the open socket.

getProperty

Get a property value.

This publisher supports two properties: + [horizontal]

MTU

The current MTU value

LastRecvAddr

The last receive address

setProperty

Set a property value. This can be used to fuzz properties exposed by the publisher. + This publisher supports one property:

[horizontal]

MTU

Set the current MTU value. The value is ignored if it is not within the set range.

Examples

Example 270. Sending data

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="propertySize">
    <Number size="32" value="1500"/>
</DataModel>
```

```

<DataModel name="Data">
    <String name="Start" value="Start"/>
    <Blob name="Data" valueType="hex" value="BEEFEA7E41"/>
    <String name="Stop" value="Stop"/>
</DataModel>

<DataModel name="UDPPacket">
    <Block name="Header">
        <Number name="SrcPort" size="16" endian="big" value="31337"/>
        <Number name="DestPort" size="16" endian="big" value="31337"/>
        <Number name="Length" size="16" endian="big">
            <Relation type="size" of="UDPPacket"/>
        </Number>
        <Number name="Checksum" size="16" endian="big">
            <Fixup class="UDPChecksumFixup">
                <Param name="ref" value="UDPPacket"/>
                <Param name="src" value="127.0.0.1"/>
                <Param name="dst" value="127.0.0.1"/>
            </Fixup>
        </Number>
    </Block>
    <Block name="UdpPayload" ref="Data"/>
</DataModel>

<DataModel name="Packet">
    <Block name="Header">
        <Flags size="16" endian="big">
            <Flag name="Version" position="0" size="4" valueType="hex" value="04" token="true"/>
            <Flag name="HeaderLength" position="4" size="4" valueType="hex">
                <Relation type="size" of="Header" expressionGet="size * 4" expressionSet="size / 4"/>
            </Flag>
            <Flag name="DSCP" position="8" size="6" valueType="hex" value="00"/>
            <Flag name="ECN" position="14" size="2" valueType="hex" value="00"/>
        </Flags>
        <Number name="TotalLength" size="16" endian="big" valueType="hex">
            <Relation type="size" of="Packet"/>
        </Number>
        <Number name="Identification" size="16" endian="big" value="0"/>
        <Flags name="Flags" size="16" endian="big">
            <Flag name="Reserved" position="0" size="1" valueType="hex" value="00"/>
            <Flag name="DF" position="1" size="1" valueType="hex" value="01"/>
            <Flag name="MF" position="2" size="1" valueType="hex" value="00"/>
            <Flag name="FragmentOffset" position="3" size="13" valueType="hex" value="0000"/>
        </Flags>
    </Block>
</DataModel>
```

```

<Number name="TTL" size="8" endian="big" valueType="hex" value="40"/>
<Number name="Protocol" size="8" endian="big" valueType="hex" value="11"/>
<Number name="Checksum" size="16" endian="big">
    <Fixup class="checksums.IcmpChecksumFixup">
        <Param name="ref" value="Header"/>
    </Fixup>
</Number>
<Block name="SrcBlock" length="4">
    <String name="SrcIP" value="127.0.0.1" mutable="false">
        <Transformer class="Ipv4StringToOctet" />
    </String>
</Block>
<Block name="DstBlock" length="4">
    <String name="DestIP" value="127.0.0.1" mutable="false">
        <Transformer class="Ipv4StringToOctet" />
    </String>
</Block>
<Block name="Payload" ref="UDPPacket"/>
</DataModel>

<StateModel name="TheState" initialState="initial">
    <State name="initial">
        <Action type="setProperty" property="MaxMTU">
            <DataModel ref="propertySize"/>
        </Action>

        <Action type="output">
            <DataModel ref="Packet" />
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="TheState"/>
    <Publisher class="RawV4">
        <Param name="Host" value="127.0.0.1" />
        <Param name="Protocol" value="17" />
    </Publisher>
</Test>
</Peach>

```

21.10.16. RawV4 Publisher

The *RawV4* publisher sends raw IPv4 packets without the IP headers.

Input received from *RawV4* includes the IPv4 header in the received data.

The *RawV4* maximum and minimum MTU size can be fuzzed by using the *setProperty* action.



This publisher runs best on Linux and requires root privileges.

Supported Protocol Values:

1

ICMP

2

IGMP

6

TCP

17

UDP

41

ENCAP

89

OSPF

132

SCTP

Syntax

```
<Publisher class="RawV4">
  <Param name="Host" value="127.0.0.1" />
  <Param name="Protocol" value="17" />
</Publisher>
```

Parameters

Required:

Host

Hostname or IP address of remote host

Protocol

IP protocol to use (e.g. TCP)

Optional:

Interface

IP of interface to bind to. Defaults to auto select.

Timeout

How long to wait in milliseconds for data/connection. Defaults to 3,000.

MaxMTU

Maximum allowable MTU property value. Only needed when fuzzing MTU. Defaults to 131,070.

MinMTU

Minimum allowable MTU property value. Only needed when fuzzing MTU. Defaults to 1,280.

Actions

start

Implicit Action to start the Publisher.

stop

Implicit Action to stop the Publisher.

open

Open and initialize the socket.

close

Close and clean up the socket.

output

Data sent via output is written to the open socket.

input

Data received via input is read from the open socket.

getProperty

Get a property value. This publisher supports two properties:

MTU

The current MTU value

LastRecvAddr

The last receive address

setProperty

Set a property value. This can be used to fuzz properties exposed by the publisher. This publisher supports one property: +

MTU

Set the current MTU value. The value is ignored if it is not within the set range.

Examples

Example 271. Sending data

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="propertySize">
    <Number size="32" value="1500"/>
</DataModel>

<DataModel name="Data">
    <String name="Start" value="Start"/>
    <Blob name="Data" valueType="hex" value="BEEFEA7E41"/>
    <String name="Stop" value="Stop"/>
</DataModel>

<DataModel name="Packet">
    <Block name="Header">
        <Number name="SrcPort" size="16" endian="big" value="31337"/>
        <Number name="DestPort" size="16" endian="big" value="31337"/>
        <Number name="Length" size="16" endian="big">
            <Relation type="size" of="Packet"/>
        </Number>
        <Number name="Checksum" size="16" endian="big">
            <Fixup class="UDPChecksumFixup">
                <Param name="ref" value="Packet"/>
                <Param name="src" value="127.0.0.1"/>
                <Param name="dst" value="127.0.0.1"/>
            </Fixup>
        </Number>
    </Block>
    <Block name="UdpPayload" ref="Data"/>
</DataModel>
```

```
<StateModel name="TheState" initialState="initial">
  <State name="initial">
    <Action type="setProperty" property="MaxMTU">
      <DataModel ref="propertySize"/>
    </Action>

    <Action type="output">
      <DataModel ref="Packet" />
    </Action>
  </State>
</StateModel>

<Test name="Default">
  <StateModel ref="TheState"/>
  <Publisher class="RawV4">
    <Param name="Host" value="127.0.0.1" />
    <Param name="Protocol" value="17" />
  </Publisher>
</Test>
</Peach>
```

21.10.17. RawV6 Publisher

The *RawV6* publisher sends raw IPv6 packets without the IP headers.

Input received from *RawV6* includes the IPv6 header in the received data.

The *RawV6* MTU size can be fuzzed by using the *setProperty* action.



This publisher runs best on Linux and requires root privileges.

Syntax

```
<Publisher class="RawV6">
    <Param name="Host" value="::1" />
    <Param name="Protocol" value="17" />
</Publisher>
```

Parameters

Required:

Host

Hostname or IP address of remote host

Protocol

IP protocol to use (value must be between 0 and 255).

Optional:

Interface

IP of interface to bind to. Uses all interfaces by default.

Timeout

How long to wait in milliseconds for data/connection. Defaults to 3,000.

MaxMTU

Maximum allowable MTU property value. Defaults to 131,070.

MinMTU

Minimum allowable MTU property value. Defaults to 1,280.

Actions

start

Implicit Action to start the Publisher.

stop

Implicit Action to stop the Publisher.

open

Open and initialize the socket.

close

Close and clean up the socket.

output

Data sent via output is written to the open socket.

input

Data received via input is read from the open socket.

getProperty

Get a property value. This publisher supports two properties:

MTU

The current MTU value

LastRecvAddr

The last receive address

setProperty

Set a property value. This can be used to fuzz properties exposed by the publisher. This publisher supports one property:

MTU

Set the current MTU value. The value is ignored if it is not within the set range.

Examples

Example 272. Sending data

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="propertySize">
    <Number size="32" value="1500"/>
</DataModel>

<DataModel name="Data">
```

```

<String name="Start" value="Start"/>
<Blob name="Data" valueType="hex" value="BEEFEA7E41"/>
<String name="Stop" value="Stop"/>
</DataModel>

<DataModel name="UDPPacket">
  <Block name="Header">
    <Number name="SrcPort" size="16" endian="big" value="31337"/>
    <Number name="DestPort" size="16" endian="big" value="31337"/>
    <Number name="Length" size="16" endian="big">
      <Relation type="size" of="UDPPacket"/>
    </Number>
    <Number name="Checksum" size="16" endian="big">
      <Fixup class="UDPChecksumFixup">
        <Param name="ref" value="UDPPacket"/>
        <Param name="src" value="::1"/>
        <Param name="dst" value="::1"/>
      </Fixup>
    </Number>
  </Block>
  <Block name="UdpPayload" ref="Data"/>
</DataModel>

<DataModel name="Packet">
  <Flags size="32" endian="big">
    <Flag name="Version" position="0" size="4" valueType="hex" value="06"/>
    <Flag name="DSCP" position="4" size="6" valueType="hex" value="00"/>
    <Flag name="ECN" position="10" size="2" valueType="hex" value="00"/>
    <Flag name="FlowLabel" position="12" size="20" valueType="hex" value="000000"/>
  </Flags>
  <Number name="PayloadLength" size="16" endian="big">
    <Relation type="size" of="IPv6Payload" />
  </Number>
  <Number name="NextHeader" size="8" value="17"/>
  <Number name="HopLimit" size="8" endian="big" valueType="hex" value="40"/>
  <Block name="SrcBlock" length="16">
    <Blob name="SrcIP" value="::1">
      <Transformer class="Ipv6StringToOctet"/>
    </Blob>
  </Block>
  <Block name="DstBlock" length="16">
    <Blob name="DestIP" value="::1">
      <Transformer class="Ipv6StringToOctet"/>
    </Blob>
  </Block>
  <Block name="IPv6Payload" ref = "UDPPacket"/>
</DataModel>

```

```
<StateModel name="TheState" initialState="initial">
  <State name="initial">
    <Action type="setProperty" property="MaxMTU">
      <DataModel ref="propertySize"/>
    </Action>

    <Action type="output">
      <DataModel ref="Packet" />
    </Action>
  </State>
</StateModel>

<Test name="Default">
  <StateModel ref="TheState"/>
  <Publisher class="RawV6">
    <Param name="Host" value="::1" />
    <Param name="Protocol" value="17" />
  </Publisher>
</Test>
</Peach>
```

21.10.18. Remote Publisher

Remote publisher runs another publisher from a Peach remote [Agent](#) process.

Remote resides on a separate (virtual or actual) machine from the remote [Agent](#). The remote [Agent](#) must have started before *Remote* is initiated. Peach will connect to the agent and provide the information required to start the remote publisher.

Syntax

```
<Publisher class="Remote">
    <Param name="Agent" value="RemoteAgent" />
    <Param name="Class" value="RawEther"/>

    <!-- Parameters for RawEther -->
    <Param name="Interface" value="eth0" />
</Publisher>
```

Parameters

Required:

Agent

Name of Agent to run the Publisher from

Class

Name of Publisher to run

Remoted Parameters

Parameters for Publisher being remoted

Optional:

There are no optional parameters for this publisher.

Actions

Any actions supported by remoted publisher.

Examples

Example 273. Remoting TCP Publisher

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
```

```

xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

<Defaults>
  <Number endian="big" />
</Defaults>

<DataModel name="TheDataModel">
  <Blob name="Dest" valueType="hex" mutable="false" length="6" value="FFFFFFFFFFFF"
/>
  <Blob name="Src" valueType="hex" mutable="false" length="6" value="FFFFFFFFFFFF
"/>
  <Number name="TypeOrLen" size="16" token="true" valueType="hex" value="0806"/>
  <Block name="Payload">
    <String name="name" value="Hello, scoobysnacks." />
  </Block>
</DataModel>

<DataModel name="propertySize">
  <Number size="32" value="1500" />
</DataModel>

<Agent name="RemoteAgent" location="tcp://192.168.1.1:9001" />

<StateModel name="TheState" initialState="initial">
  <State name="initial">
    <Action type="output">
      <DataModel ref="TheDataModel"/>
    </Action>
  </State>
</StateModel>

<Test name="Default">
  <Agent ref="RemoteAgent" />

  <StateModel ref="TheState"/>

  <Publisher class="Remote">
    <Param name="Agent" value="RemoteAgent" />
    <Param name="Class" value="RawEther" />

    <!-- Parameters for RawEther -->
    <Param name="Interface" value="eth0" />
  </Publisher>
</Test>
</Peach>

```

Starting Peach Agent Process on Remote Machine

```
peach.exe -a tcp
```

21.10.19. Rest Publisher (Deprecated)

This publisher is deprecated, please see [WebApi publisher](#).

The *Rest* Publisher is an I/O adapter that communicates using the Representational State Transfer (Rest) architecture. This publisher communicates using HTTP as the underlying transport.

This publisher supports sending body data as:

- JSON when models contain Json elements (JsonString, JsonObject, etc.) or the deprecated Json element.
- XML when models contain XmlElement
- As raw binary data when a custom content type is provided

Similar to the [Http](#) publisher, the *Rest* publisher supports setting headers by using a special action type *call* syntax. See the examples section for an example of setting a custom header field.

Several analyzers are useful when building Pits for RESTful APIs:

Json

Converts JSON documents or strings into Peach data models. Can be used both inside of DataModels with the String element or also via the command line.

Postman

Converts Postman Catalogs to Peach Pits.

Swagger

Converts Swagger JSON to Peach Pits



SSL/TLS is supported, just use [https](#) as the protocol in the URL.



By default the publisher will attempt to detect the body content type and set the correct ContentType header.

Syntax

```
<!-- AUtomatically detect content type -->
<Publisher class="Rest"/>

<!-- Custom content type header -->
<Publisher class="Rest">
    <Param name="ContentType" value="application/octet-stream" />
</Publisher>
```

Parameters

Required:

There are no required parameters.

Optional:

ContentType

Value for content-type header. Set this to allow sending of binary data.

BaseUrl

Base URL is used by some authentication types. Only used as part of authentication.

Username

Username for authentication

Password

Optional password associated with the Username in authentication

Domain

Domain for authentication

Cookies

Enable cookie support. The default value is true.

CookiesAcrossIterations

Track cookies across iterations. The default value is false.

FailureStatusCodes

Comma separated list of status codes that are failures causing current test case to stop. Defaults to: **400,401,402,403,404,405,406,407,408,409,410,411,412,413,414,415,416,417,500,501,502,503,504,505**

FaultOnStatusCodes

Comma separated list of status codes that are faults. Defaults to none.

Timeout

How long to wait in milliseconds for data/connection. the default value is 3,000.

IgnoreCertErrors

Allow HTTPS regardless of cert status. The default value is true.

Proxy

To use HTTP proxy, set the URL. Default is none. Example: <http://192.168.1.1:8080>.

Please note that the host **localhost** and IP 127.0.0.1 will bypass the provided proxy. This is a

behavior hardcoded into the underlying http networking code. For a discussion of options to deal with this limitation see the following article: [Fiddler - Monitoring Local Traffic](#).

Actions

call

Call actions are used to perform Rest calls.

The method attribute contains both the HTTP method (GET, POST, etc.) and the URL. The URL portion of the method attribute can contain parameter substitution tokens. These tokens are replaced with the parameters provided. One additional parameter can be provided that is used for the body of the request.

- The format of the method attribute is: `METHOD URL`
- This example places a call that expects to receive a single parameter: `GET /product/{0}`
- This example places a call that expects to receive two parameters: `GET /invoices?start_date={0}&end_date={1}`

A final Param can be added to provide the body content if needed.

This action also supports the `Result` element to capture the response payload.



The order of the parameters is important. In order they are matched to the substitutions of {0}, then {1}, etc. The final Param (optional) is used for the body payload.

call

Call actions are used to set HTTP headers.

To set a header using a call action you must:

- Set the method to "Header"
- Have two parameters named `Name` and `Value`
- The `Name` parameter is the header key name
- The `Value` parameter is the value for the header

By default they are fuzzed. This can be disabled using the `mutable` attribute on the data model and data elements or using the `Include/Exclude` elements in the `Test` portion of the xml.

An example of settings headers both via Python and also the `call` action are provided in the Example section.

Scripting

The Rest publisher exposes a public Headers dictionary that can be used to add/remove headers from Python scripting code. See example *Setting Custom Authentication Header via Python* below.

Example

Example 274. Calling Rest Services with Result

The following example provides three fragments using the GET and POST methods. For the GET request, the Result element is used to capture any returned data.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="postData">
    <JsonObject>
        <JsonString propertyName="Name" value="Widget" />
        <JsonDouble propertyName="Price" value="1.99" />
        <JsonInteger propertyName="Quantify" value="1" />
    </JsonObject>
</DataModel>

<DataModel name="RestString">
    <String name="value" value="">
        <Hint name="Peach.TypeTransform" value="false" />
    </String>
</DataModel>

<DataModel name="RestResult">
    <Choice name="ResultOrEmpty">
        <String name="Result">
            <Analyzer class="Json" />
        </String>
        <Block name="Empty" />
    </Choice>
</DataModel>

<StateModel name="Default" initialState="FirstState">
    <State name="FirstState">
        <Action type="call" method="GET http://www.example.com/product/{0}">
            <!-- {0} -->
            <Param name="Id">
                <DataModel ref="RestString" />
                <Data>
```

```

        <Field name="value" value="1"/>
    </Data>
</Param>

        <!-- Capture Response (optional) -->
<Result>
    <DataModel ref="RestResult" />
</Result>
</Action>

<Action type="call" method="GET
http://www.example.com/invoices?start_date={0}&end_date={1}">
<!-- {0} -->
<Param name="StartDate">
    <DataModel ref="RestString" />
    <Data>
        <Field name="value" value="11-21-2011"/>
    </Data>
</Param>

<!-- {1} -->
<Param name="EndDate">
    <DataModel ref="RestString" />
    <Data>
        <Field name="value" value="11-21-2015"/>
    </Data>
</Param>

        <!-- Capture Response (optional) -->
<Result>
    <DataModel ref="RestResult" />
</Result>
</Action>

<Action type="call" method="POST http://www.example.com/product/{0}">
<!-- {0} -->
<Param name="Id">
    <DataModel ref="RestString" />
    <Data>
        <Field name="value" value="100"/>
    </Data>
</Param>

        <!-- POST Body -->
<Param name="postData">
    <DataModel ref="PostData" />
</Param>
</Action>
```

```

</State>
</StateModel>

<Test name="Default">
    <StateModel ref="Default" />
    <Publisher class="Rest" />
</Test>

</Peach>

```

Example 275. Posting XML

The following example provides three fragments using the GET and POST methods.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="RestString">
    <String name="value" value="">
        <Hint name="Peach.TypeTransform" value="false" />
    </String>
</DataModel>

<DataModel name="PostData">
    <XmlElement elementName="Product">
        <XmlAttribute attributeName="Name">
            <String value="Widget" />
        </XmlAttribute>
        <XmlAttribute attributeName="Price">
            <String value="1.99" />
        </XmlAttribute>
        <XmlAttribute attributeName="Quantity">
            <String value="1" />
        </XmlAttribute>
    </XmlElement>
</DataModel>

<StateModel name="Default" initialState="FirstState">
    <State name="FirstState">

        <Action type="call" method="POST http://www.example.com/product/{0}">
            <!-- {0} -->
            <Param name="Id">
                <DataModel ref="RestString" />

```

```
<Data>
    <Field name="value" value="1" />
</Data>
</Param>

<!-- POST Body -->
<Param name="PostData">
    <DataModel ref="PostData" />
</Param>
</Action>

</State>
</StateModel>

<Test name="Default">
    <StateModel ref="Default" />
    <Publisher class="Rest" />
</Test>

</Peach>
```

Example 276. Posting Binary

The following example provides three fragments using the GET and POST methods.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="RestString">
  <String name="value" value="">
    <Hint name="Peach.TypeTransform" value="false" />
  </String>
</DataModel>

<DataModel name="PostData">
  <Blob />
</DataModel>

<StateModel name="Default" initialState="FirstState">
  <State name="FirstState">

    <Action type="call" method="POST http://www.example.com/product/{0}/image">
      <!-- {0} -->
      <Param name="Id">
        <DataModel ref="RestString" />
        <Data>
          <Field name="value" value="1" />
        </Data>
      </Param>

      <!-- POST Body -->
      <Param name="postData">
        <DataModel ref="PostData" />
        <Data fileName="image.png" />
      </Param>
    </Action>

  </State>
</StateModel>

<Test name="Default">

  <StateModel ref="Default"/>
  <Publisher class="Rest">
    <Param name="ContentType" value="application/octet-stream" />
  </Publisher>

</Test>

</Peach>

```

Example 277. Setting Custom Header via Pit

The following example shows how to set a custom header via the Pit XML. The custom header is named "X-CustomHeader" with a value of "Hello World".

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="RestString">
    <String name="value" value="">
        <Hint name="Peach.TypeTransform" value="false" />
    </String>
</DataModel>

<DataModel name="RestResult">
    <Choice name="ResultOrEmpty">
        <String name="Result">
            <Analyzer class="Json" />
        </String>
        <Block name="Empty" />
    </Choice>
</DataModel>

<StateModel name="Default" initialState="FirstState">
    <State name="FirstState">

        <!-- Add X-CustomHeader header -->
        <Action type="call" method="Header">
            <Param name="Name">
                <DataModel ref="RestString" />
                <Data>
                    <Field name="value" value="X-CustomHeader"/>
                </Data>
            </Param>
            <Param name="Value">
                <DataModel ref="RestString" />
                <Data>
                    <Field name="value" value="Hello World!"/>
                </Data>
            </Param>
        </Action>

        <Action type="call" method="GET http://www.example.com/product/{0}">
            <!-- {0} -->
            <Param name="Id">
```

```

<DataModel ref="RestString" />
<Data>
    <Field name="value" value="1"/>
</Data>
</Param>

<!-- Capture Response (optional) -->
<Result>
    <DataModel ref="RestResult" />
</Result>
</Action>

</State>
</StateModel>

<Test name="Default">
    <StateModel ref="Default" />
    <Publisher class="Rest" />
</Test>

</Peach>

```

Example 278. Setting Custom Authentication Header via Python

The following example shows how to add custom authentication via a python script. In this example we will configure a pit for fuzzing an Amazon AWS S3 service endpoint. This is only an example and should not actually be used to fuzz AWS.

```

import base64
import hmac
from hashlib import sha1
from email.Utils import formatdate

AWS_ACCESS_KEY_ID = "44CF9590006BF252F707"
AWS_SECRET_KEY = "0txrzxIsfpFjA7SwPzILwy8Bw21TLhquhboDYROV"

def AwsAuthGen(context, action):

    # Get the Publisher (RestPublisher)
    if action.publisher:
        publisher = context.test.publishers[action.publisher]
    else:
        publisher = context.test.publishers[0]

    XAmzDate = formatdate()

    h = hmac.new(AWS_SECRET_KEY, "PUT\n\napplication/json\n\nnx-amz-date:%s\n\n/?policy"
    % XAmzDate, sha1)
    authToken = base64.encodestring(h.digest()).strip()

    publisher.Headers.Add("x-amz-date", XAmzDate)
    publisher.Headers.Add("Authorization", "AWS %s:%s" % (AWS_ACCESS_KEY_ID,
    authToken))

# end

```

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<Import import="aws_s3_example"/>

<!--
{
"Version": "2008-10-17",
"Id": "aaaa-bbbb-cccc-dddd",
"Statement" : [
    {
        "Effect": "Allow",
        "Sid": "1",
        "Principal" : {
            "AWS": ["111122223333", "444455556666"]
        },

```

```

        "Action":["s3:*"],
        "Resource":"arn:aws:s3:::bucket/*"
    }
]
}
-->
<!-- Generated using the JSON analyzer -->
<DataModel name="Policy">
    <JsonObject>
        <JsonString propertyName="Version" name="Version" value="2008-10-17" />
        <JsonString propertyName="Id" name="Id" value="aaaa-bbbb-cccc-dddd" />
        <JsonArray propertyName="Statement" name="Statement">
            <JsonObject propertyName="Statement" name="Statement">
                <JsonString propertyName="Effect" name="Effect" value="Allow" />
                <JsonString propertyName="Sid" name="Sid" value="1" />
                <JsonObject propertyName="Principal" name="Principal">
                    <JsonArray propertyName="AWS" name="AWS">
                        <JsonString propertyName="AWS" name="AWS" value="111122223333" />
                        <JsonString value="444455556666" />
                    </JsonArray>
                </JsonObject>
            </JsonArray>
            <JsonArray propertyName="Action" name="Action">
                <JsonString propertyName="Action" name="Action" value="s3:*" />
            </JsonArray>
            <JsonString propertyName="Resource" name="Resource" value=
"arn:aws:s3:::bucket/*" />
        </JsonObject>
    </JsonArray>
</JsonObject>
</DataModel>

<StateModel name="TheStateModel" initialState="Initial">
    <State name="Initial">
        <Action type="call" method="PUT http://XXXXX.s3.amazonaws.com/?policy"
               onStart="aws_s3_example.AwsAuthGen(context, action)">
            <Param name="Body">
                <DataModel ref="Policy" />
            </Param>
        </Action>
    </State>
</StateModel>

<Test name="Default" maxOutputSize="20000000">
    <StateModel ref="TheStateModel"/>
    <Publisher class="Rest">

```

```
<Param name="FaultOnStatusCodes" value="500,501,502,503,504,505" />
</Publisher>
</Test>
</Peach>
```

21.10.20. SerialPort Publisher

The *SerialPort* publisher allows Peach to communicate with a device using a serial port and serial transmissions.

Syntax

```
<Publisher class="SerialPort">
    <Param name="PortName" value="/dev/ttyUSB0"/>
    <Param name="Baudrate" value="115200"/>
    <Param name="DataBits" value="8"/>
    <Param name="Parity" value="None"/>
    <Param name="StopBits" value="One"/>
</Publisher>
```

Parameters

Required:

PortName

Serial port device name

Baudrate

Serial baud rate. The transmission rate that accounts for data bits and more: start bit, stop bits, parity bits, and commands.

Parity

The parity-checking protocol.

DataBits

Standard length of data bits per byte.

StopBits

The standard number of stop bits per byte.

Optional:

Handshake

The handshaking protocol for serial port data transmission. The default value is none.

Timeout

Maximum waiting period, specified in milliseconds, between transmissions. The default value is 3,000.

Actions

open

Open the device

input

Receive data

output

Send data

Examples

Example 279. Sending and receiving data

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

  <DataModel name="DataModel">
    <Blob name="Hello" value="Hello Serial Device!\n"/>
  </DataModel>

  <StateModel name="TheState" initialState="Initial">
    <State name="Initial">
      <Action type="open"/>
      <Action type="output">
        <DataModel ref="DataModel"/>
      </Action>
      <Action type="input">
        <DataModel ref="DataModel"/>
      </Action>
    </State>
  </StateModel>

  <Test name="Default">
    <StateModel ref="TheState"/>
    <Publisher class="SerialPort">
      <Param name="PortName" value="/dev/ttyUSB0"/>
      <Param name="Baudrate" value="115200"/>
      <Param name="DataBits" value="8"/>
      <Param name="Parity" value="None"/>
      <Param name="StopBits" value="One"/>
    </Publisher>  </Test>
  </Peach>
```

21.10.21. Ssl Publisher

The *Ssl* Publisher enables a pit to communicate via an SSL over TCP socket.

Ssl is used to fuzz protocols built inside an SSL encrypted channel (such as HTTPS).

Since it is easy to confuse *Ssl* and *SslListner*, here's the difference:

- The *Ssl* Publisher connects out
- The *SslListner* Publisher accepts (inward) connections

Syntax

```
<Publisher class="Ssl">
    <Param name="Host" value="192.168.48.128" />
    <Param name="Port" value="433" />
</Publisher>

<Publisher class="Ssl">
    <Param name="Host" value="192.168.48.128" />
    <Param name="Port" value="433" />
    <Param name="Alpn" value="h2;spdy/3.1;http/1.1" />
</Publisher>
```

Parameters

Required:

Host

Hostname to connect to.

Port

Port to connect to.

Optional:

ClientCert

Provide client certificate for server verification. Path to client certificate in PEM format

ClientKey

Provide client key for server verification. Path to client private key in PEM format

Alpn

Enable ALPN TLS extension (RFC 7301). Example value: `h2;spdy/3.1;http/1.1`

VerifyServer

Verify the server certificate. Defaults to false.

ConnectTimeout

Max milliseconds to wait for connection (default 10000).

Timeout

How many milliseconds to wait for data (default 3000).

Actions

start

Implicit Action to start the Publisher.

stop

Implicit Action to stop the Publisher.

open

Open and initialize the SSL connection.

close

Close and clean up the SSL connection.

output

Data sent via output is written to the SSL stream.

input

Data received via input is read from the SSL input buffer.

Examples

31. Sending and receiving data

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

<DataModel name="OutModel">
    <Block name="Headers">
        <String value="POST /testsslpage.html HTTP/1.0\r\n" />
        <String value="User-Agent: TestAgent/1.0\r\n"/>
        <String value="Content-Length: " />
        <String name="ContentLen">
            <Relation of="Data" type="size"/>
        </String>
        <String value="\r\n\r\n"/>
        <String name="Data"/>
    </Block>
</DataModel>

<DataModel name="InModel">
    <String value="{{$ token=true }} />
    <String />
    <String value="{}" token=true />
</DataModel>

<StateModel name="State" initialState="First">
    <State name="First">
        <Action type="output">
            <DataModel ref="OutModel">
        </Action>
        <Action type="input">
            <DataModel ref="InModel">
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="State"/>

    <Publisher class="Ssl">
        <Param name="Host" value="localhost"/>
        <Param name="Port" value="31337"/>
        <Param name="Timeout" value="3000"/>
    </Publisher>
</Test>
</Peach>
```

21.10.22. SsListener Publisher

The *SsListener* Publisher enables a pit to accept SSL connections over a TCP socket.

SsListener is used to fuzz protocol built inside an SSL encrypted channel, such as HTTPS.

Since it is easy to confuse *Ssl* and *SsListener*, here's the difference:

- The *Ssl* Publisher connects out(ward)
- The *SsListener* Publisher accepts (inward) connections

The *SsListener* is similar to the *TcpListener* because both accept connections over a TCP port and block until a connection has been established.

Syntax

```
<Publisher class="SsListener">
    <Param name="Interface" value="0.0.0.0"/>
    <Param name="Port" value="31337"/>
    <Param name="Timeout" value="3000"/>
    <Param name="AcceptTimeout" value="3000"/>
    <Param name="ServerCertPath" value="cert.pfx" />
</Publisher>
```

Parameters

Required:

Interface

IP of interface to bind to. Use **0.0.0.0** for all interfaces.

Port

Local port to listen on.

ServerCertPath

Path to server certificate file.

Optional:

AcceptTimeout

How many milliseconds to wait for a connection. Defaults to 3000.

CheckCertRevocation

Check revocation of certificate. Defaults to false.

ClientCertRequired

Require client to authenticate via certificate. Defaults to false.

ServerCertPass

Password for cert file. Defaults to none.

Timeout

How many milliseconds to wait for data. Defaults to 3000.

Actions

start

Implicit Action to start the Publisher.

stop

Implicit Action to stop the Publisher.

open

Open and initialize the SSL connection.

close

Close and clean up the SSL connection.

accept

Block until an incoming SSL connection has been received.

output

Data sent via output is written to the SSL stream.

input

Data received via input is read from the SSL input buffer.

Examples

Example 280. Sending and receiving data

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

<DataModel name="OutModel">
    <Block name="Headers">
        <String value="{" />
        <String value="key1" />
        <String value="," />
        <String value="value1" />
        <String value="}" />
    </Block>
</DataModel>

<DataModel name="InModel">
    <String value="GET /testsslpage.html HTTP/1.0\r\n" />
</DataModel>

<StateModel name="State" initialState="First">
    <State name="First">
        <Action type="accept" />
        <Action type="input">
            <DataModel ref="InModel">
        </Action>
        <Action type="output">
            <DataModel ref="OutModel">
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="State"/>

    <Publisher class="SslListener">
        <Param name="Interface" value="0.0.0.0"/>
        <Param name="Port" value="31337"/>
        <Param name="Timeout" value="3000"/>
        <Param name="AcceptTimeout" value="3000"/>
        <Param name="ServerCertPath" value="cert.pfx" />
    </Publisher>
</Test>
</Peach>
```

21.10.23. Tcp Client Publisher

The *Tcp Client* publisher connects to a remote TCP service.

Since it is easy to confuse *Tcp Client* and *TcpListener*, here's the difference:

- The *Tcp Client* Publisher connects out(ward)
- The *TcpListener* Publisher listens to (inward) connections

Syntax

```
<Publisher class="Tcp">
    <Param name="Host" value="127.0.0.1" />
    <Param name="Port" value="8080" />
</Publisher>
```

Parameters

Required:

Host

Hostname or IP address of remote host

Port

Destination port number

Optional:

RetryMode

Connection retry method, defaults to **FirstAndAfterFault**.

Options:

Never

Never attempt a reconnection

FirstAndAfterFault

Only reattempt a connection when the remote host may be have restarted by Peach. This will be on first iteration and after a fault was detected.

Always

Always attempt a reconnection

FaultOnConnectionFailure

Log a fault when unable to connect to remote host. Defaults to true.

Lifetime

Lifetime of connection. Defaults to [Iteration](#).

Options:

Iteration

Connection lasts lifetime of a testcase/iteration.

Session

Connection lasts lifetime of testing session. Reconnects if connection is closed, but otherwise the connection will be left open.

Timeout

How long to wait in milliseconds for data. Defaults to 3,000.

ConnectTimeout

How long to wait in milliseconds for a new connection. Defaults to 10,000.

Actions

output

Send data to remote host

input

Receive data from remote host

Examples

Example 281. Sending and receiving data

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

    <DataModel name="TheDataModel">
        <String name="value" length="4" />
    </DataModel>

    <StateModel name="TheState" initialState="Initial">
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
                <Data>
                    <Field name="value" value="mike" />
                </Data>
            </Action>

            <!-- receive 4 bytes -->
            <Action type="input">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="TheAgent"/>

    <Test name="Default">
        <Agent ref="TheAgent"/>
        <StateModel ref="TheState"/>
        <Publisher class="Tcp">
            <Param name="Host" value="127.0.0.1" />
            <Param name="Port" value="8080" />
        </Publisher>
    </Test>
</Peach>
```

21.10.24. TcpListener Publisher

The *TcpListener* publisher is able to listen for incoming connections on a TCP port.

Since it is easy to confuse *Tcp Client* and *TcpListener*, here's the difference:

- The *Tcp Client* Publisher connects out(ward)
- The *TcpListener* Publisher listens to (inward) connections

Syntax

```
<Publisher class="TcpListener">
    <Param name="Interface" value="127.0.0.1" />
    <Param name="Port" value="8080" />
</Publisher>
```

Parameters

Required:

Interface

IP of interface to bind to

Port

Destination port number

Optional:

Timeout

How long to wait in milliseconds for data once a connection has been established. Defaults to 3,000.

AcceptTimeout

How long to wait in milliseconds for a new connection. Defaults to 3,000.

Actions

accept

Wait for incoming connection

output

Send data to remote host

input

Receive data from remote host

Examples

Example 282. Sending and receiving data

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

  <DataModel name="TheDataModel">
    <String name="value" length="4" />
  </DataModel>

  <StateModel name="TheState" initialState="Initial">
    <State name="Initial">
      <Action type="accept" />
      <Action type="output">
        <DataModel ref="TheDataModel"/>
        <Data>
          <Field name="value" value="mike" />
        </Data>
      </Action>

      <!-- receive 4 bytes -->
      <Action type="input">
        <DataModel ref="TheDataModel"/>
      </Action>
    </State>
  </StateModel>

  <Agent name="TheAgent"/>

  <Test name="Default">
    <Agent ref="TheAgent"/>
    <StateModel ref="TheState"/>
    <Publisher class="TcpListener">
      <Param name="Interface" value="127.0.0.1" />
      <Param name="Port" value="8080" />
    </Publisher>
  </Test>
</Peach>
```

21.10.25. Udp Publisher

The *Udp* publisher sends and receives UDP packets.

The underlying stack correctly pads out the UDP packet on output and strips the UDP padding on input. Because of the padding and stripping, a slight difference occurs between the lengths of an incoming packet and the corresponding outgoing packet. This has no effect on the actual data sent; the integrity of the actual data remains intact.

Each *Udp* output action results in a single packet being sent. Each *Udp* input action receives a single packet.

Syntax

```
<Publisher class="Udp">
    <Param name="Host" value="127.0.0.1" />
    <Param name="Port" value="8000" />
</Publisher>
```

Parameters

Required:

Host

Host or IP address or remote host

Optional:

Port

Destination port number, only optional when first packet is sent by target.

SrcPort

Source port

Interface

IP of interface to bind to

Timeout

How long to wait in milliseconds for data/connection. Defaults to 3,000.

MaxMTU

Maximum allowable MTU property value. Defaults to 131,070.

MinMTU

Minimum allowable MTU property value. Defaults to 1,280.

Actions

output

Send a single UDP packet to remote host.

input

Receive a single UDP packet from remote host.

getProperty

Get a property value.

This publisher supports two properties:

LastRecvAddr

The last receive address

Port

Get the **Port** parameter.

MTU

Get the **MTU** of the interface

SrcPort

Get the **SrcPort** parameter.

setProperty

Set a property value.

This publisher supports two properties:

Port

Change the **Port** parameter.

MTU

Change the **MTU** parameter.

SrcPort

Change the **SrcPort** parameter.

Examples

Example 283. Connect Example

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

    <DataModel name="TheDataModel">
        <String name="value" length="4" />
    </DataModel>

    <StateModel name="TheState" initialState="Initial">
        <State name="Initial">
            <Action type="output">
                <DataModel ref="TheDataModel"/>
                <Data>
                    <Field name="value" value="mike" />
                </Data>
            </Action>

            <!-- receive 4 bytes -->
            <Action type="input">
                <DataModel ref="TheDataModel"/>
            </Action>
        </State>
    </StateModel>

    <Agent name="TheAgent"/>

    <Test name="Default">
        <Agent ref="TheAgent"/>
        <StateModel ref="TheState"/>
        <Publisher class="Udp">
            <Param name="Host" value="127.0.0.1" />
            <Param name="Port" value="8000" />
        </Publisher>

    </Test>
</Peach>
```

Example 284. Listener

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

    <DataModel name="TheDataModel">
        <String name="value" length="4" />
    </DataModel>

    <StateModel name="TheState" initialState="Initial">
        <State name="Initial">
            <Action type="open"/>
            <Action type="output">
                <DataModel ref="TheDataModel"/>
                <Data>
                    <Field name="value" value="mike" />
                </Data>
            </Action>

            <!-- receive 4 bytes -->
            <Action type="input">
                <DataModel ref="TheDataModel"/>
            </Action>
            <Action type="close"/>
        </State>
    </StateModel>

    <Agent name="TheAgent"/>

    <Test name="Default">
        <Agent ref="TheAgent"/>
        <StateModel ref="TheState"/>
        <Publisher class="Udp">
            <Param name="Host" value="127.0.0.1" />
            <Param name="Interface" value="127.0.0.1" />
            <Param name="SrcPort" value="8000" />
            <Param name="Port" value="8001" />
        </Publisher>

    </Test>
</Peach>
```

Example 285. Multicast

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

    <DataModel name="TheDataModel">
        <String name="value" length="4" />
    </DataModel>

    <StateModel name="TheState" initialState="Initial">
        <State name="Initial">
            <Action type="open"/>
            <Action type="output">
                <DataModel ref="TheDataModel"/>
                <Data>
                    <Field name="value" value="mike" />
                </Data>
            </Action>

            <!-- receive 4 bytes -->
            <Action type="input">
                <DataModel ref="TheDataModel"/>
            </Action>
            <Action type="close"/>
        </State>
    </StateModel>

    <Agent name="TheAgent"/>

    <Test name="Default">
        <Agent ref="TheAgent"/>
        <StateModel ref="TheState"/>
        <Publisher class="Udp" name="ListenPublisher">
            <Param name="Host" value="224.0.0.1"/>
            <Param name="Port" value="8000"/>
            <Param name="SrcPort" value="8001"/>
            <Param name="Interface" value="127.0.0.1"/>
        </Publisher>
    </Test>
</Peach>
```

21.10.26. USB Publisher

The *Usb* publisher is used to test devices via the USB hardware interface. This publisher is used to test devices connected to a host via USB. It cannot be used to test USB drivers.

The *Usb* publisher is a wrapper around libusb. As such it can be useful at times to refer to the libusb documentation regarding questions about publisher parameters specific to USB or to diagnose error messages.

It is recommended that users have a basic understanding of how USB works prior to using the *Usb* publisher.

A number of tools exist that can assist in identifying USB devices and capturing traffic. One tool that is highly recommended is [Wireshark](#). Wireshark has the ability to capture USB traffic on Windows and Linux. More information about capture USB with Wireshark can be found [here](#).

Syntax

```
<Publisher class="Usb">
    <Param name="VendorId" value="4817" />
    <Param name="ProductId" value="5340" />

    <Param name="Configuration" value="1"/>
    <Param name="Interface" value="0"/>
    <Param name="ReadEndpoint" value="Ep02" />
    <Param name="WriteEndpoint" value="Ep02" />
</Publisher>
```

Parameters

Required:

VendorId

USB vendor IDs (VID) are 16-bit numbers used in combination with USB vendor IDs to identify USB devices to a host. Each vendor ID is assigned by the USB Implementers Forum to a specific company. The VID is embedded in the product and communicated to the computer when the device is plugged in.

ProductId

USB product IDs (PID) are 16-bit numbers used in combination with USB vendor IDs to identify USB devices to a host. Each vendor assigns a PID to individual products. The PID is then embedded in the product and communicated to the computer when the device is plugged in.

Optional:

Configuration

USB configuration to set for device. Defaults to 1. Use -2 to skip and -1 to set unconfigured state.

If the device is already configured with the selected configuration, a lightweight device reset will occur: a SET_CONFIGURATION is issued causing most USB-related device state to be reset.

The configuration cannot be set if other applications or drivers have claimed interfaces. In this case a value of -2 can be used to skip setting the configuration.

A configuration value of -1 will put the device into unconfigured state. The USB specifications state that a configuration value of 0 does this, however buddy devices exist which actually have a configuration 0.

A USB device configuration descriptor specifies how the device is powered, the number of interfaces it has, etc.

Interface

USB device interface descriptor to claim for use. Defaults to 0.

The interface groups endpoints for a single feature of the device.

ReadEndpoint

USB endpoint to read from. Defaults to *Ep01*. Options are: Ep01 through Ep15.

ReadEndpointType

USB endpoint read type. Defaults to *Bulk*. Options are: Bulk, Control, Interrupt, Isochronous

WriteEndpoint

USB endpoint to write to. Defaults to *Ep01*. Options are: Ep01 through Ep15.

WriteEndpointType

USB endpoint write type. Defaults to *Bulk*. Options are: Bulk, Control, Interrupt, Isochronous

Timeout

How long to wait in milliseconds for reading and writing. Defaults to 3,000.

Actions

start

Open USB device, set configuration, claim interface and open endpoints.

Performed once at the start of fuzzing.

stop

Free all USB resources. Performed once at the end of fuzzing.

output

Transmit data via the USB writer endpoint.

input

Receive data via the USB reader endpoint.

Examples

Example 286. USB Fuzzing Example

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach-
pro/output/win_x64_debug/bin/peach.xsd">

    <Import import="time" />

    <Defaults>
        <Number endian="big" signed="false"/>
    </Defaults>

    <DataModel name="TheDataModel">
        <Number name="MessageType" size="32" endian="little" value="1" />
        <Number name="MessageLength" size="32" endian="little">
            <Relation type="size" of="TheDataModel"/>
        </Number>
        <Number name="DataOffset" size="32" endian="little">
            <Relation type="offset" of="Frame"/>
        </Number>
        <Number name="DataLength" size="32" endian="little">
            <Relation type="size" of="Frame"/>
        </Number>
        <Number name="OOBDataOffset" size="32" endian="little" value="0" />
        <Number name="OOBDataLength" size="32" endian="little" value="0" />
        <Number name="NumOOBDataElements" size="32" endian="little" value="0" />
        <Number name="PerPacketInfoOffset" size="32" endian="little" value="0" />
        <Number name="PerPacketInfoLength" size="32" endian="little" value="0" />
        <Number name="VcHandle" size="32" endian="little" value="0" />
        <Number name="Reserved" size="32" endian="little" value="0" />

        <Blob name="Frame" value="AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" />
    </DataModel>

    <StateModel name="State" initialState="State1" >
        <State name="State1">
```

```
<Action type="output" onStart="time.sleep(1)">
    <DataModel ref="TheDataModel"/>
</Action>
</State>
</StateModel>

<Test name="Default">
    <StateModel ref="State"/>

    <Publisher class="Usb">
        <Param name="VendorId" value="4817" />
        <Param name="ProductId" value="5340" />
        <Param name="Configuration" value="1"/>
        <Param name="Interface" value="0"/>
        <Param name="ReadEndpoint" value="Ep02" />
        <Param name="WriteEndpoint" value="Ep02" />
    </Publisher>
</Test>
</Peach>
```

21.10.27. WebApi Publisher

The *WebApi* Publisher is an I/O adapter that communicates with various web API/HTTP end points. This publisher is used with the [web](#) action type.

This is the recommended method to perform web requests using Peach. The [Rest](#) and [Http](#) publishers are deprecated.

This publisher, in combination with the [web](#) action gives full control over:

- Path
- Query string
- Form data
- Body
 - Json
 - XML
 - Binary

Several analyzers are useful when building Pits for use with this publisher:

[Json](#)

Converts JSON documents or strings into Peach data models. Can be used both inside of DataModels with the String element or also via the command line.

[Postman](#)

Converts Postman Catalogs to Peach Pits.

[Swagger](#)

Converts Swagger JSON to Peach Pits

[WebRecordProxy](#)

Recording proxy captures web requests and generates full pit. This makes creating the base pit easy, simply use as your HTTP proxy.

[Xml](#)

Converts XML documents or string into Peach data models. Can be used both inside of DataModels with the String element or also via the command line.



SSL/TLS is supported, just use [https](#) as the protocol in the URL.

Syntax

```
<Publisher class="WebApi"/>
```

Parameters

Required:

There are no required parameters.

Optional:

BaseUrl

The base URL to use for authentication.

Example: <http://myservice.domainname.example>.

This parameter is required if the **Username** and **Password** parameters are specified.

Username

Username for authentication

Password

Password for authentication

Domain

Domain for authentication

FailureStatusCodes

Comma separated list of status codes that are failures causing current test case to stop. Defaults to:
400,401,402,403,404,405,406,407,408,409,410,411,412,413,414,415,416,417,500,501,502,503,504,505

FaultOnStatusCodes

Comma separated list of status codes that are faults. Defaults to none.

Timeout

How long to wait in milliseconds for data/connection. the default value is 3,000.

IgnoreCertErrors

Allow HTTPS regardless of cert status. The default value is true.

Proxy

To use HTTP proxy, set the URL. Default is none. Example: <http://192.168.1.1:8080>.

The publisher will not use the default system proxy. If a proxy is required it must be explicitly set via the publisher parameter.

Please note that the host **localhost** and IP 127.0.0.1 will bypass the provided proxy. This is a behavior hardcoded into the underlying http networking code. For a discussion of options to deal

with this limitation see the following article: [Fiddler - Monitoring Local Traffic](#).

Actions

web

Web actions are used to perform WebApi calls.

This action provides full control over the HTTP request. The *method* attribute contains the HTTP method (GET, POST, etc.). The *url* attribute contains the URL with optional parameter substitution tokens. These tokens are replaced with the parameters provided by the *Path* elements.

- This example places a call that expects to receive a single parameter: `http://localhost/product/{id}`
- Query parameters are set using *Query* elements.

This action also supports the **Response** element to capture the response status code, headers and body.

Example

Example 287. Calling WebApi Services with Result

The following example provides three fragments using the GET and POST methods. For the GET request, the **Result** element is used to capture any returned data.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="postData">
    <JsonObject>
        <JsonString propertyName="Name" value="Widget" />
        <JsonDouble propertyName="Price" value="1.99" />
        <JsonInteger propertyName="Quantify" value="1" />
    </JsonObject>
</DataModel>

<DataModel name="WebApiResult">
    <Choice name="ResultOrEmpty">
        <String name="Result">
            <Analyzer class="Json" />
        </String>
        <Block name="Empty" />
    </Choice>
</DataModel>
```

```

<StateModel name="Default" initialState="FirstState">
    <State name="FirstState">
        <Action type="web" method="GET" url="http://www.example.com/product/{id}">
            <Path name="Id" key="id" value="1"/>

            <Response />
                <DataModel ref="WebApiResult" />
            </Response>
        </Action>

        <Action type="web" method="GET" url="http://www.example.com/invoices">
            <Query name="StartDate" key="start_date" value="11-21-2011" />
            <Query name="EndDate" key="end_date" value="11-21-2015" />

            <Response>
                <DataModel ref="WebApiResult" />
            </Response>
        </Action>

        <Action type="call" method="POST" url="http://www.example.com/product/{id}">
            <Path name="Id" key="id" value="100" />
            <Body name="PostData">
                <DataModel ref="PostData" />
            </Body>
        </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="Default" />
    <Publisher class="WebApi" />
</Test>

</Peach>

```

Example 288. Posting XML

The following example provides three fragments using the GET and POST methods.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="postData">
    <XmlElement elementName="Product">
        <XmlAttribute attributeName="Name">
            <String value="Widget" />
        </XmlAttribute>
        <XmlAttribute attributeName="Price">
            <String value="1.99" />
        </XmlAttribute>
        <XmlAttribute attributeName="Quantity">
            <String value="1" />
        </XmlAttribute>
    </XmlElement>
</DataModel>

<StateModel name="Default" initialState="FirstState">
    <State name="FirstState">

        <Action type="web" method="POST" url="http://www.example.com/product/{id}">
            <Path name="Id" key="id" value="1"/>

            <Body name="postData">
                <DataModel ref="postData" />
            </Body>
        </Action>

    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="Default" />
    <Publisher class="WebApi" />
</Test>

</Peach>
```

Example 289. Posting Binary

The following example provides three fragments using the GET and POST methods.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<DataModel name="PostData">
    <Blob />
</DataModel>

<StateModel name="Default" initialState="FirstState">
    <State name="FirstState">

        <Action type="web" method="POST" url=
"http://www.example.com/product/{id}/image">
            <Path name="Id" key="id" value="1"/>
            <Body name="postData">
                <DataModel ref="PostData" />
                <Data fileName="image.png" />
            </Body>
        </Action>

    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="Default"/>
    <Publisher class="WebApi"/>
</Test>
</Peach>
```

Example 290. Setting Custom Header via Pit

The following example shows how to set a custom header via the Pit XML. The custom header is named "X-CustomHeader" with a value of "Hello World".

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<StateModel name="Default" initialState="FirstState">
    <State name="FirstState">

        <Action type="web" method="GET" url="http://www.example.com/product/{id}">
            <Path name="Id" key="id" value="1"/>
            <Header name="x-custom" key="X-CustomHeader" value="Hello World!" />
            <Response/>
        </Action>

    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="Default" />
    <Publisher class="WebApi" />
</Test>

</Peach>
```

Example 291. Setting Custom Authentication Header via Python

The following example shows how to add custom authentication via a python script. In this example we will configure a pit for fuzzing an Amazon AWS S3 service endpoint. This is only an example and should not actually be used to fuzz AWS.

```

import base64
import hmac
from hashlib import sha1
from email.Utils import formatdate

AWS_ACCESS_KEY_ID = "44CF9590006BF252F707"
AWS_SECRET_KEY = "0txrzxIsfpFjA7SwPzILwy8Bw21TLhquhboDYROV"

def AwsAuthGen(context, action):

    # Get the Publisher (WebApiPublisher)
    if action.publisher:
        publisher = context.test.publishers[action.publisher]
    else:
        publisher = context.test.publishers[0]

    XAmzDate = formatdate()

    h = hmac.new(AWS_SECRET_KEY, "PUT\n\napplication/json\n\nnx-amz-date:%s\n\n/?policy"
    % XAmzDate, sha1)
    authToken = base64.encodestring(h.digest()).strip()

    publisher.Headers.Add("x-amz-date", XAmzDate)
    publisher.Headers.Add("Authorization", "AWS %s:%s" % (AWS_ACCESS_KEY_ID,
    authToken))

# end

```

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

<Import import="aws_s3_example"/>

<!--
{
"Version": "2008-10-17",
"Id": "aaaa-bbbb-cccc-dddd",
"Statement" : [
    {
        "Effect": "Allow",
        "Sid": "1",
        "Principal" : {
            "AWS": ["111122223333", "444455556666"]
        },

```

```

        "Action":["s3:*"],
        "Resource":"arn:aws:s3:::bucket/*"
    }
]
}
-->
<!-- Generated using the JSON analyzer -->
<DataModel name="Policy">
    <JsonObject>
        <JsonString propertyName="Version" name="Version" value="2008-10-17" />
        <JsonString propertyName="Id" name="Id" value="aaaa-bbbb-cccc-dddd" />
        <JsonArray propertyName="Statement" name="Statement">
            <JsonObject propertyName="Statement" name="Statement">
                <JsonString propertyName="Effect" name="Effect" value="Allow" />
                <JsonString propertyName="Sid" name="Sid" value="1" />
                <JsonObject propertyName="Principal" name="Principal">
                    <JsonArray propertyName="AWS" name="AWS">
                        <JsonString propertyName="AWS" name="AWS" value="111122223333" />
                        <JsonString value="444455556666" />
                    </JsonArray>
                </JsonObject>
            </JsonArray>
            <JsonArray propertyName="Action" name="Action">
                <JsonString propertyName="Action" name="Action" value="s3:*" />
            </JsonArray>
            <JsonString propertyName="Resource" name="Resource" value=
"arn:aws:s3:::bucket/*" />
        </JsonObject>
    </JsonArray>
</JsonObject>
</DataModel>

<StateModel name="TheStateModel" initialState="Initial">
    <State name="Initial">
        <Action type="web" method="PUT" url="http://XXXXX.s3.amazonaws.com/?policy"
               onStart="aws_s3_example.AwsAuthGen(context, action)">
            <Body name="Body">
                <DataModel ref="Policy" />
            </Body>
        </Action>
    </State>
</StateModel>

<Test name="Default" maxOutputSize="20000000">
    <StateModel ref="TheStateModel"/>
    <Publisher class="WebApi">

```

```
<Param name="FaultOnStatusCodes" value="500,501,502,503,504,505" />
</Publisher>
</Test>
</Peach>
```

21.10.28. WebService Publisher

The *WebService* publisher is able to call SOAP and WCF based web services.

WebService attempts to locate a service definition, or you can provide one.

Syntax

```
<Publisher class="WebService">
    <Param name="Url" value="http://localhost:7789/TestService/Service.asmx" />
    <Param name="Service" value="Service" />
</Publisher>
```

Parameters

Required:

Url

WebService URL

Service

Service Name

Optional:

Wsdl

Path or URL to WSDL for web service

ErrorOnStatusCode

Error when status code isn't 200, defaults to true.

Timeout

How long to wait in milliseconds for data/connection, defaults to 3,000.

Throttle

Time in milliseconds to wait between connections, defaults to 0.

Actions

call

Method attribute is method on web service to call.

Examples

Example 292. Example calling web service

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

  <DataModel name="ArbitraryData">
    <String name="value" />
  </DataModel>

  <StateModel name="TheState" initialState="Initial">
    <State name="Initial">

      <Action type="call" method="Login">
        <Param name="name">
          <DataModel ref="ArbitraryData"/>
          <Data>
            <Field name="value" value="mike" />
          </Data>
        </Param>
        <Param name="passwd">
          <DataModel ref="ArbitraryData"/>
          <Data>
            <Field name="value" value="Password!" />
          </Data>
        </Param>
      </Action>

    </State>
  </StateModel>

  <Agent name="TheAgent"/>

  <Test name="Default">
    <Agent ref="TheAgent"/>
    <StateModel ref="TheState"/>
    <Publisher class="WebService">
      <Param name="Url" value="http://localhost:7789/TestService/Service.asmx" />
      <Param name="Service" value="Service" />
    </Publisher>
  </Test>

</Peach>
```

21.10.29. WebSocket Publisher

The *WebSocket* publisher allows rapid delivery of fuzzing data to browsers using a web socket.

The web socket publisher works through a simple JSON based protocol. The browser evokes a JavaScript handler to accept data from the fuzzer and display it. The iterations wait for the data to be fully loaded prior to completing an iteration.

Peach provides a fuzzer template with a placeholder to drop the encoded data. The client code loads the template into an iframe.

Syntax

```
<Publisher class="WebSocket">
  <Param name="Port" value="8080"/>
  <Param name="Template" value="peach_ws_template.html"/>
  <Param name="Publish" value="base64"/>
</Publisher>
```

Parameters

Required:

Template

Data template for publishing. The template contains the HTML that is loaded into an iframe element in the browser each iteration. It must contain the *DataToken* placeholder which will be replaced with the data based on the *Publish* parameter.

Optional:

Port

Port to listen for connections on. Defaults to **8080**.

Publish

How to publish data, base64 or URL. Defaults to **base64**.

DataToken

Token to replace with data in template. Defaults to **##DATA##**.

Timeout

Time in milliseconds to wait for client response. Defaults to **60000**.

Actions

output

Generate and send *Template*

Examples

Example 293. Basic Usage Example

A working example of the WebSocket publisher comes with the Peach Fuzzer Professional binary distribution in the *samples* folder. The full example is a combination of four files and is too long to list in this document. The list of files is below.

websocket.xml

Pit file

peach_ws_client.html

HTML file that is loaded into browser

peach_ws_template.html

Template for HTML generated and sent to target

peach_ws_client.js

JavaScript code used by *peach_ws_client.html*

32. websocket.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach /peach/peach.xsd">

<DataModel name="TheDataModel">
  <Blob/>
</DataModel>

<StateModel name="TheState" initialState="Initial">
  <State name="Initial">
    <Action type="output">
      <DataModel ref="TheDataModel"/>
      <Data fileName="samples_png" />
    </Action>
  </State>
</StateModel>

<Agent name="TheAgent">
  <Monitor class="WindowsDebugger">
    <Param name="Executable" value="C:\Program Files (x86)\Mozilla
Firefox\firefox.exe" />
    <Param name="Arguments" value="peach_ws_client.html" />
  </Monitor>
</Agent>

<Test name="Default">
  <Agent ref="TheAgent"/>
  <StateModel ref="TheState"/>

  <Publisher class="WebSocket">
    <Param name="Port" value="8080"/>
    <Param name="Template" value="peach_ws_template.html"/>
    <Param name="Publish" value="base64"/>
  </Publisher>
</Test>

</Peach>
```

21.10.30. Zip Publisher

The Zip publisher opens a zip file for writing. The Zip publisher enumerates all Stream data elements in the DataModel and creates a zip entry for each stream.

Syntax

```
<Publisher class="Zip">
    <Param name="FileName" value="output.zip" />
</Publisher>
```

Parameters

Required:

FileName

Name of file to open

Actions

open

Open file for reading/writing.

close

Close file stream.

output

Data to be written to file

Examples

Example 294. Write two text files (file1.txt and file2.txt) to a zip file

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">

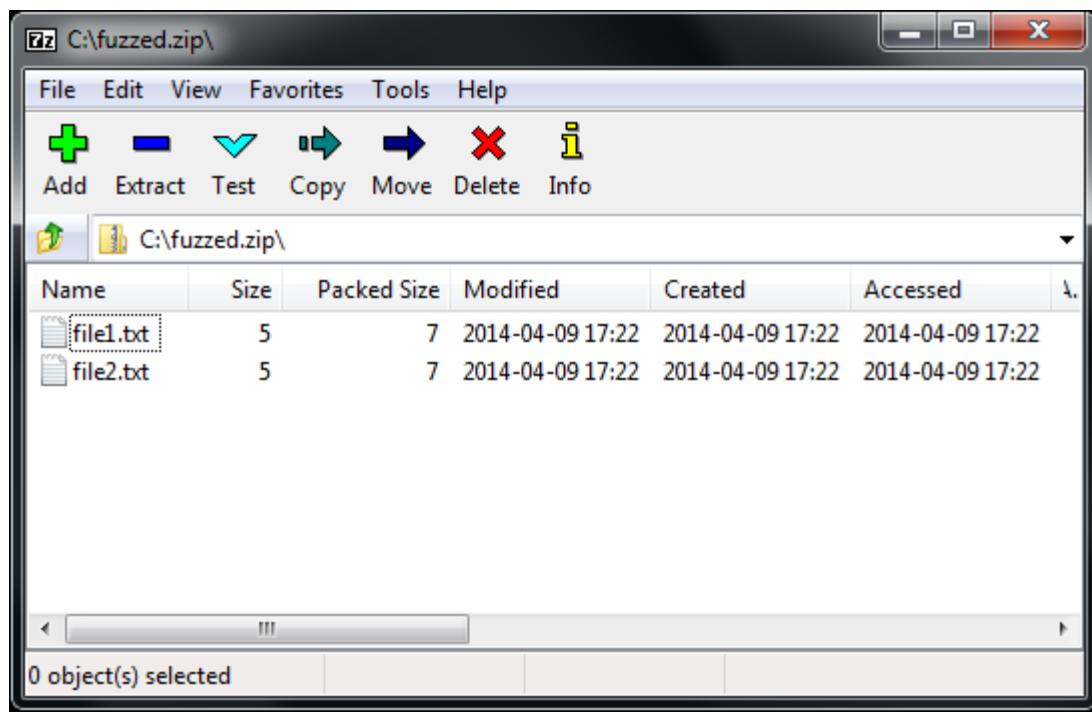
    <DataModel name="TheDataModel">
        <Stream streamName='file1.txt'>
            <String value='Hello' />
        </Stream>
        <Stream streamName='file2.txt'>
            <String value='World' />
        </Stream>
    </DataModel>

    <StateModel name="TheState" initialState="initial">
        <State name="initial">
            <Action type="output">
                <DataModel ref="TheDataModel" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="TheState"/>
        <Publisher class="Zip">
            <Param name="FileName" value="fuzzed.zip" />
        </Publisher>
    </Test>
</Peach>

```

View of fuzzed.zip in 7Zip



21.11. Mutators

Mutators perform the actual data changes to models that produce fuzzed output. The fuzzing strategy decides how and when to apply mutators. During a fuzzing session, Peach weights mutators based on the number of variations of data that each mutator generates. Peach selects mutators that generate more variations more frequently than mutators that generate fewer variations of fuzzed data.

A typical mutator performs a single type of change. For example, the *BlobBitFlipperMutator* performs bit flipping on *Blob* elements. Most mutators have no or very little state or complex logic.

Mutators that are associated with state have a prefix for identification, either State or Action.

Custom mutators are relatively easy to write.

To focus on using a specific set of mutators in a fuzzing session, specify the mutators to use or those not to use during a fuzzing run in the *Test* element of the Pit. Most often, Peach considers all mutations and uses all mutations appropriate for the DataModel.

The following sections provide descriptions of each mutator, as well as hints and configuration options.

21.11.1. ArrayEdgeCase

This mutator produces test cases in which the size of an array expands or contracts to have item counts near certain numerical edge cases. An algorithm helps in producing a distribution with focal points at numerical edge cases:

- Where an array index transitions from 0 to 8 bits, from 8 to 16 bits, 16 to 32 bits, and from 32 to 64 bits.
- Where numeric values are interpreted differently for signed and unsigned integer types.

This mutator focuses on integer issues that lead to memory corruption.



Currently this mutator limits the number of items in an array to 65K.

Supported Elements

This mutator supports any array element. Array elements are those that have the *occurs*, *minOccurs*, or *maxOccurs* attribute.

- `occurs` attribute
- `minOccurs` attribute
- `maxOccurs` attribute

Hints

This mutator does not support hints.



A previous version of this mutator supported a hint. An update to the underlying algorithm removed the need for a hint.

21.11.2. ArrayRandomizeOrder

This mutator randomizes the order of items in an array.

Operation with deterministic strategies

With deterministic strategies, the number of test cases this mutator generates is the lesser of 100, or the factorial of the number of array items.

You can repeat this mutator with a different seed value to produce different randomized array orders.

Operation with non-deterministic strategies

With non-deterministic strategies, each call to this mutator produces a randomized order of the elements. The weighting of this mutator is based the lesser of 100, or the factorial of the number of array items.

Supported Elements

This mutator supports any array element. Array elements are those that have the `occurs`, `minOccurs`, or `maxOccurs` attribute.

- `occurs` attribute
- `minOccurs` attribute
- `maxOccurs` attribute

Hints

This mutator does not support hints.



A previous version of this mutator supported a hint. An update to the underlying algorithm removed the need for a hint.

21.11.3. ArrayReverseOrder

This mutator reverses the order of items in an array.

Supported Elements

This mutator supports any array element. Array elements are those that have the *occurs*, *minOccurs*, or *maxOccurs* attribute.

- [occurs](#) attribute
- [minOccurs](#) attribute
- [maxOccurs](#) attribute

Hints

This mutator does not support hints.

21.11.4. ArrayVarianceMutator

This mutator produces test cases in which the array expands or contracts to produce sizes that are in a distribution with the center of the distribution the current count of the array. For example, if an array contains 100 elements, this mutator produces test cases in which the array has grown to larger sizes and smaller sizes with the sizes clustered around 100.



Currently this mutator limits the number of items in an array to 65K.

Supported Elements

This mutator supports any array element. Array elements are those that have the *occurs*, *minOccurs*, or *maxOccurs* attributes set.

- [occurs](#) attribute
- [minOccurs](#) attribute
- [maxOccurs](#) attribute

Hints

This mutator does not support hints.



A previous version of this mutator supported a hint. An update to the underlying algorithm removed the need for a hint.

21.11.5. BlobChangeFromNull

This mutator produces test cases in which a random number of contiguous null bytes in a [Blob](#) are changed. The location where the change occurs is randomly-determined as is the number of bytes altered in the mutation. The number of contiguous bytes that receive new and different values range from 1 to 100.

Supported Elements

- [Blob](#)

Hints

BlobChangeFromNull-N

Standard deviation of the number of bytes to change.

BlobMutator-N

Standard deviation of the number of bytes to change. The value of this Hint affects other Blob mutators.

21.11.6. BlobChangeRandom

This mutator produces test cases in which a random number of contiguous bytes in a [Blob](#) are changed. The location where the change occurs is randomly-determined as is the number of bytes altered in the mutation. The number of contiguous bytes that receive new and different values range from 1 to 100.

Supported Elements

- [Blob](#)

Hints

BlobChangeRandom-N

Standard deviation of the number of bytes to change.

BlobMutator-N

Standard deviation of the number of bytes to change. The value of this Hint affects other Blob mutators.

21.11.7. BlobChangeSpecial

This mutator produces test cases in which a random number of contiguous bytes in a [Blob](#) are individually changed using a small set of replacement values. The location where the change occurs is randomly-determined as is the number of bytes altered in the mutation. The number of bytes that receive new values range from 1 to 100.

Each altered value randomly receives one of the following replacement values: 0x00, 0x01, 0xFE, or 0xFF.

Supported Elements

- [Blob](#)

Hints

BlobChangeSpecial-N

Standard deviation of the number of bytes to change.

BlobMutator-N

Standard deviation of the number of bytes to change. The value of this Hint affects other Blob mutators.

21.11.8. BlobChangeToNull

This mutator produces test cases in which a random number of contiguous bytes in a [Blob](#) are changed to null. The location where the change occurs is randomly-determined as is the number of bytes altered in the mutation. The number of bytes that receive the null value range from 1 to 100.

Supported Elements

- [Blob](#)

Hints

BlobChangeToNull-N

Standard deviation of the number of bytes to change.

BlobMutator-N

Standard deviation of the number of bytes to change. The value of this Hint affects other Blob mutators.

21.11.9. BlobExpandAllRandom

This mutator produces test cases in which a random number of contiguous bytes are inserted in a [Blob](#). The location where the insertion starts is randomly determined, as is the number of inserted bytes. The number of inserted bytes range from 1 to 255. The value of each inserted byte is a different randomly-selected value.

Supported Elements

- [Blob](#)

Hints

BlobExpandAllRandom-N

Standard deviation of the number of bytes to change.

BlobMutator-N

Standard deviation of the number of bytes to change. The value of this Hint affects other Blob mutators.

21.11.10. BlobExpandSingleIncrementing

This mutator produces test cases in which a random number of contiguous bytes are inserted in a [Blob](#). The location where the insertion starts is randomly-determined as is the number of inserted bytes. The number of inserted bytes range from 1 to 255.

The value of the first inserted byte is randomly selected and ranges from 0x00 to 0xff. The value of the second inserted byte increments by 1. The value of the third byte increments again. When the value 0xff occurs, value of the subsequent byte rolls over to 0x00. Thereafter the values continue to increment until all of the inserted bytes have values.

Supported Elements

- [Blob](#)

Hints

BlobExpandAllRandom-N

Standard deviation of the number of bytes to change.

BlobMutator-N

Standard deviation of number of the bytes to change. The value of this Hint affects other Blob mutators.

21.11.11. BlobExpandSingleRandom

This mutator produces test cases in which a random number of contiguous bytes having the same value are inserted in a [Blob](#). The location where the insertion starts is randomly-determined as is the number of inserted bytes. The number of inserted bytes range from 1 to 255.

The value used for the inserted bytes is randomly selected from the range of 0x00 to 0xff.

Supported Elements

- [Blob](#)

Hints

BlobExpandSingleRandom-N

Standard deviation of the number of bytes to change.

BlobMutator-N

Standard deviation of the number of bytes to change. The value of this Hint affects other Blob mutators.

21.11.12. BlobExpandZero

This mutator produces test cases in which a random number of contiguous null bytes are inserted in a [Blob](#). The location where the insertion starts is randomly-determined as is the number of inserted bytes. The number of inserted bytes range from 1 to 255.

Supported Elements

- [Blob](#)

Hints

BlobExpandZero-N

Standard deviation of the number of bytes to change.

BlobMutator-N

Standard deviation of the number of bytes to change. The value of this Hint affects other Blob mutators.

21.11.13. BlobReduce

This mutator produces test cases in which a random number of contiguous bytes are removed from a [Blob](#). The location where the deletion starts is randomly-determined as is the number of bytes to delete. The number of deleted bytes range from 1 to 255.

Supported Elements

- [Blob](#)

Hints

BlobReduce-N

Standard deviation of the number of bytes to change.

BlobMutator-N

Standard deviation of the number of bytes to change. The value of this Hint affects other Blob mutators.

21.11.14. ChoiceSwitch

This mutator produces test cases for [Choice](#) elements by changing the selected choice.

Supported Elements

- [Choice](#)

Hints

This mutator does not support hints.

21.11.15. DataElementBitFlipper

This mutator produces test cases by changing the values of individual bits (bit flipping) within data produced by the model. The number of changed bits is a Gaussian distribution with a range of 1 to 6.

If Transformers are used in the model, this mutator affects both the pre- and post- transformed data.

Supported Elements

This mutator can attach to the following data elements:

- Container elements that have a transformer attached. Container elements typically have one or more child elements. Containers include Data Models, Blocks, Choices and arrays.
- Data elements that are not containers.

The lone exception: a data element that does not perform a type transformation on itself is unsupported. A type transformation occurs when an element packs or encodes its data in another format. A type transformation can occur with elements whose parent elements have types such as XmlElement or XmlAttribute. A couple of type transformation examples follow:

- A Number element that packs its value into a binary format
- A String element that encodes the characters into a representation such as ASCII or UTF-8

Hints

This mutator does not support hints.

21.11.16. DataElementDuplicate

This mutator produces test cases for all data elements by duplicating them in the model. The number of duplicated elements range from 1 to 50 and form a distribution. The distribution is configurable.

Supported Elements

All data elements are supported.

Hints

DataElementDuplicate-N

Standard deviation of the number of duplications.

21.11.17. DataElementRemove

This mutator produces a single test case by removing all data elements from the model.

Supported Elements

All data elements are supported.

Hints

This mutator does not support hints.

21.11.18. DataElementSwapNear

This mutator produces a single test case for all data elements by swapping the element with its neighbor.

Supported Elements

All data elements are supported.



This mutator binds to any container data element (DataModel, Block, Choice, Flags, or array) that contains 2 or more child elements.

Hints

This mutator does not support hints.

21.11.19. DoubleRandom

This mutator produces test cases for Double and String elements. String elements are only supported when the default data they contain is a number. The test cases produced are random floating point numbers in the numerical space of the element. For a 32-bit floating point number, the values range from -3.402823E+38 to 3.4028234E+38. For String elements, the range of values is identical to that of a 64-bit floating point number, from -1.79769313486232E+308 to 1.7976931348623157E+308.

For weighting purposes and for deterministic strategies where an estimate of the number of generated test cases is needed, this mutator reports that it generates 5,000 test cases. In actuality, the number of test cases can exceed 5000.

Supported Elements

- [String](#)
- [Double](#)

Hints

This mutator does not support hints.

21.11.20. DoubleVariance

This mutator produces test cases for Double and String elements. String elements are only supported when the default data they contain is a number. The test cases are produced using a Gaussian distribution with the current default value as the center of the distribution.

Supported Elements

- [String](#)
- [Double](#)

Hints

This mutator does not support hints.

21.11.21. ExtraValues

This mutator allows the user to supply additional values to use in new test cases. Each additional value generates its own test case. The additional values are tested "as is"; that is they are not altered.

If the *ExtraValues* hint is mandatory. If the hint is not provided, this mutator is not used.

Supported Elements

- [String](#)
- [Number](#)
- [Blob](#)

Hints

ExtraValues

Specify a semicolon-separated list of the values to use in new test cases.



The former hint name *ValidValues*, is deprecated in favor of *ExtraValues*.

Examples

Example 295. Providing Extra Values for a String

In this example, three test cases are added to the string *FirstName*.

```
<String name="FirstName" value="Josh">
    <!-- This list adds three test cases -->
    <Hint name="ExtraValues" value="O'Brian;O-Brian;Josh III" />
</String>
```

21.11.22. NumberEdgeCase

This mutator produces test cases for Number and String elements. String elements are only supported when the default data they contain is an integer. The test cases are generated using a Gaussian distribution around numerical edge cases. Edge cases are defined as boundaries between signed/unsigned and bit boundaries (0, 8, 16, 32, 64).

Supported Elements

- [String](#)
- [Number](#)

Hints

This mutator does not support hints.

21.11.23. NumberRandom

This mutator produces test cases for Number and String elements. String elements are only supported when the default data they contain is an integer. The test cases produced are random integers in the numerical space of the element. For an unsigned 8-bit number that range is 0 to 255. For String elements, the range is that of a signed 64-bit integer, -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.

For weighting purposes and for deterministic strategies where an estimate of the number of generated test cases is needed, this mutator reports that it generates 5,000 test cases. In actuality, the number of test cases can exceed 5000.

Supported Elements

- [String](#)
- [Number](#)

Hints

This mutator does not support hints.

21.11.24. NumberVariance

This mutator produces test cases for Number and String elements. String elements are only supported when the default data they contain is an integer. The test cases are produced using a Gaussian distribution with the current default value as the center of the distribution.

Supported Elements

- [String](#)
- [Number](#)

Hints

This mutator does not support hints.

21.11.25. SampleNinja

The SampleNinja Mutator swaps elements between different sample files using a [Sample Ninja](#) database.



This mutator is enabled only if a sample ninja database exists. You can use PitTool to create a sample ninja database.

Supported Elements

All elements are supported.

Hints

This mutator does not support hints.

21.11.26. SizedDataEdgeCase

This mutator produces test cases for elements that are part of a size relationship. The SizedDataEdgeCase Mutator causes the data in a sized element to expand such that the relationship is invalid. The data portion of the relationship expands, while the physical size of the field is static.

Test cases are generated using a Gaussian distribution around numerical edge cases. Edge cases for integers are defined as boundaries between signed/unsigned representations and bit boundaries (0, 8, 16, 32, 64).

Values used in the test cases never require more storage than the maximum size the size provider can handle. For example, values used for a signed 16-bit integer do not exceed 32,767.



This mutator is similar to the [NumberEdgeCase](#) Mutator.

Supported Elements

Elements that are part of a size relationship: numbers and strings. String elements are only supported when the default data they contain is integer values.

Hints

This mutator does not support hints.

21.11.27. SizedDataVariance

This mutator produces test cases for elements that are part of a size relationship. The SizedDataVariance Mutator causes the data in a sized element to expand such that the relationship is invalid. The data portion of the relationship expands, while the physical size of the field is static.

The test cases are produced using a Gaussian distribution with the current default value as the center of the distribution.

Values used in the test cases never require more storage than the maximum size the size provider can handle. For example, values used for a signed 16-bit integer do not exceed 32,767.



This mutator is similar to the [NumberVariance](#) Mutator.

Supported Elements

Elements that are part of a size relationship: numbers and strings. String elements are only supported when the default data they contain is integer values.

Hints

This mutator does not support hints.

21.11.28. SizedEdgeCase

This mutator produces test cases for elements that are part of a size relationship. The SizedEdgeCase Mutator causes the data in a sized element to expand and, as needed, expands the physical size of the field to keep the relationship valid.

The test cases are generated using a Gaussian distribution around numerical edge cases. Edge cases for integers are defined as boundaries between signed/unsigned representations and bit boundaries (0, 8, 16, 32, 64).

Values used in the test cases never require more storage than the maximum size the size provider can handle. For example, values used for a signed 16-bit integer do not exceed 32,767.



This mutator is similar to the [NumberEdgeCase](#) Mutator.

Supported Elements

Elements that are part of a size relationship: numbers and strings. String elements are only supported when the default data they contain is integer values.

Hints

This mutator does not support hints.

21.11.29. SizedVariance

This mutator produces test cases for elements that are part of a size relationship. The SizedVariance Mutator causes the data in a sized element to expand and, as needed, expands the physical size of the field to keep the relationship valid.

The test cases are generated using a Gaussian distribution with the current default value as the center of the distribution.

Values used in the test cases never require more storage than the maximum size the size provider can handle. For example, values used for a signed 16-bit integer do not exceed 32,767.



This mutator is similar to the [NumberVariance](#) Mutator.

Supported Elements

Elements that are part of a size relationship: numbers and strings. String elements are only supported when the default data they contain is integer values.

Hints

This mutator does not support hints.

21.11.30. StateChangeRandom

This mutator produces test cases that affect the flow of the state model. During a state change, starting with selecting the initial state, the state change can land on a randomly-selected state. This mutator can be enabled if more than one state is available.



Currently, state model mutations are disabled by default.

Supported Elements

All states defined in the state model.

Hints

This mutator does not support hints.

21.11.31. StringAsciiRandom

This mutator produces test cases by generating random ASCII strings. Characters that comprise the strings use the 7-bit ASCII collating sequence, values 0x00 - 0x7F.

The lengths of the generated strings form a distribution that centers on the length of the current string and ranges from 0 to 65K characters.

Supported Elements

- [String](#)

Hints

This mutator does not support hints.

21.11.32. StringCaseLower

This mutator produces a single test case by making the characters in the string all lower case. This mutator is not enabled for strings that contain all lowercase data.

Supported Elements

- [String](#)

Hints

This mutator does not support hints.

21.11.33. StringCaseRandom

This mutator produces test cases by randomly changing the case of a string.

Supported Elements

- [String](#)

Hints

This mutator does not support hints.

21.11.34. StringCaseUpper

This mutator produces a single test case by making the characters in the string all upper case. This mutator is not enabled for strings that contain all upper case data.

Supported Elements

- [String](#)

Hints

This mutator does not support hints.

21.11.35. StringLengthEdgeCase

This mutator produces test cases for String elements. The test cases produced are length edge cases created by shrinking and growing the string as needed.

The lengths are generated using a Gaussian distribution around numerical edge cases. Edge cases are defined as integer boundaries for signed/unsigned number representations and for bit boundaries (0, 8, 16, 32, 64).

The maximum size string generated by this mutator is 65K characters.

Supported Elements

- [String](#)

Hints

This mutator does not support hints.

21.11.36. StringLengthVariance

This mutator produces test cases for String elements. The test cases produced are variances of the string length.

The string lengths generated for the test cases form a Gaussian distribution with the current length as the center of the distribution.

Supported Elements

- [String](#)

Hints

This mutator does not support hints.

21.11.37. **StringList**

This mutator produces test cases for String elements. Test cases are generated by rotating through a list of strings read from a file.

This mutator is enabled for String elements that have the **StringList** hint, providing a valid file of strings.

Supported Elements

- [String](#)

Hints

StringList

Name of a file that contains one or more strings. The format of the file requires the placement of one string per line.

21.11.38. StringStatic

This mutator produces test cases for String elements by accessing an internal list of strings that have caused or that might cause a data consumer to misbehave.

The strings in the list include special filenames, characters, numbers, and other strings. Currently, the list produces 1667 unique test cases.

Supported Elements

- [String](#)

Hints

This mutator does not support hints.

21.11.39. StringUnicodeAbstractCharacters

This mutator produces test cases for Unicode String elements by generating new strings and then populating the strings with a random set of Unicode abstract characters. The lengths of the generated strings form a distribution that centers on the current string length and ranges from 1 to 65k Unicode characters.

This mutator requires String elements that have a Unicode encoding type: UTF-8, UTF-16, or UTF-32.

Supported Elements

- [String](#)

Hints

This mutator does not support hints.

21.11.40. StringUnicodeFormatCharacters

This mutator produces test cases for Unicode String elements by generating new strings and then populating the strings with a random set of Unicode format characters. A format character has no visible appearance, but can affect the appearance or behavior of neighboring or subsequent characters. The lengths of the generated strings form a distribution that centers on the current string length and ranges from 1 to 65k Unicode characters.

This mutator requires String elements that have a Unicode encoding type: UTF-8, UTF-16, or UTF-32.

Unicode 7.0 contains 152 format characters. A couple of format character examples follow:

- U+200C ZERO WIDTH NON-JOINER is a format character that inhibits ligatures.
- U+200D ZERO WIDTH JOINER is a formation character that requests a ligature formation.

Supported Elements

- [String](#)

Hints

This mutator does not support hints.

21.11.41. StringUnicodeInvalid

This mutator produces test cases for Unicode String elements by generating new strings and then populating the strings with randomly-selected values from a set of invalid Unicode characters. The codes for these characters are included in the Unicode space, but are currently unused or are specifically excluded.

The lengths of the generated strings form a distribution that centers on the current string length and ranges from 1 to 65K Unicode characters.

This mutator requires String elements that have a Unicode encoding type: UTF-8, UTF-16, or UTF-32.

Supported Elements

- [String](#)

Hints

This mutator does not support hints.

21.11.42. StringUnicodeNonCharacters

This mutator produces test cases for Unicode String elements by generating new strings and then populating the strings with randomly-selected values from a set of Unicode code points that do not map to actual characters. The lengths of the generated strings form a distribution that centers on the length of the current string and ranges from 1 to 65K (ushort max) Unicode characters.

This mutator requires String elements that have a Unicode encoding type: UTF-8, UTF-16, or UTF-32.

The Unicode standard includes 60 non-character code points. The set of non-character code points is stable and mature. No new non-character code points will be defined. Non-character code points have either of the following characteristics:

- code point values in the range U+FDD0 – U+FDEF
- code point values that end with FFFE or FFFF (U+FFFE, U+FFFF, U+1FFE, U+1FFF, . . . , U+10FFFe, U+10FFFF).

Supported Elements

- [String](#)

Hints

This mutator does not support hints.

21.11.43. StringUnicodePlane0

This mutator produces test cases for Unicode String elements by generating new strings and populating the strings with randomly-selected values from Unicode plane 0 (basic multilingual plane) characters. The lengths of the generated strings form a distribution that centers on the length of the current string and ranges from 1 to 65K (ushort max) Unicode characters.

This mutator requires Strings that have a Unicode encoding type: UTF-8, UTF-16, or UTF-32.

Supported Elements

- [String](#)

Hints

This mutator does not support hints.

21.11.44. StringUnicodePlane1

This mutator produces test cases for Unicode String elements by generating new strings and populating the strings with randomly-selected values from Unicode plane 1 (supplementary multilingual plane) characters. The lengths of the generated strings form a distribution that centers on the length of the current string and ranges from 1 to 65K (ushort max) Unicode characters.

This mutator requires String elements that have a Unicode encoding type: UTF-8, UTF-16, or UTF-32.

Supported Elements

- [String](#)

Hints

This mutator does not support hints.

21.11.45. StringUnicodePlane14

This mutator produces test cases for Unicode String elements by generating new strings and populating the strings with randomly-selected values from Unicode plane 14 (supplementary special-purpose plane) characters. The lengths of the generated strings form a distribution that centers on the length of the current string and ranges from 1 to 65K (ushort max) Unicode characters.

This mutator requires String elements that have a Unicode encoding type: UTF-8, UTF-16, or UTF-32.

Supported Elements

- [String](#)

Hints

This mutator does not support hints.

21.11.46. StringUnicodePlane15And16

This mutator produces test cases for Unicode String elements by generating new strings and populating the strings with randomly-selected values from Unicode plane 15 and 16 (private use area planes) characters. The lengths of the generated strings form a distribution that centers on the length of the current string and ranges from 1 to 65K (ushort max) Unicode characters.

This mutator requires String elements that have a Unicode encoding type: UTF-8, UTF-16, or UTF-32.

Supported Elements

- [String](#)

Hints

This mutator does not support hints.

21.11.47. StringUnicodePlane2

This mutator produces test cases for Unicode String elements by generating new strings and populating the strings with randomly-selected values from Unicode plane 2 (supplementary ideographic plane) characters. The lengths of the generated strings form a distribution that centers on the length of the current string and ranges from 1 to 65K (ushort max) Unicode characters.

This mutator requires String elements that have a Unicode encoding type: UTF-8, UTF-16, or UTF-32.

Supported Elements

- [String](#)

Hints

This mutator does not support hints.

21.11.48. StringUnicodePrivateUseArea

This mutator produces test cases for Unicode String elements by generating new strings and populating the strings with randomly-selected values from characters U+E000 – U+F8FF of Unicode plane 0 (private use area of the BMP). The lengths of the generated strings form a distribution that centers on the length of the current string and ranges from 1 to 65K (ushort max) Unicode characters.

This mutator requires String elements that have a Unicode encoding type: UTF-8, UTF-16, or UTF-32.

Supported Elements

- [String](#)

Hints

This mutator does not support any hints.

21.11.49. StringUtf8BomLength

The *StringUtf8BomLength* mutator produces test cases for ASCII and Unicode UTF-8 strings by altering an individual string in two ways:

- Adjust the length of the string using the [StringLengthVariance](#) mutator
- Insert 1 to 6 Unicode Byte Ordering Marks (BOMs) in arbitrary places in the string

BOMs are not considered characters, and using them can cause buffer length calculations to be incorrect.

Supported Elements

- [String](#)

Hints

This mutator does not support any hints.

21.11.50. StringUtf8BomStatic

The *StringUtf8BomStatic* mutator produces test cases for ASCII and Unicode UTF-8 strings by creating a mutated string in two ways:

- Create a string using the [StringStatic](#) mutator
- Inject 1 to 6 Unicode Byte Ordering Marks (BOMs) in arbitrary places in the string

BOMs are not considered characters, and injecting BOMs into a string that might be filtered could cause the filtering not to work and the string would be accepted.

Supported Elements

- [String](#)

Hints

This mutator does not support any hints.

21.11.51. StringUtf8ExtraBytes

This mutator produces test cases for UTF-8 String elements by generating new strings and then populating the strings with randomly-selected values. The interesting aspect of this mutator is that the storage allocated for each character ranges from 1 to 6 bytes, rather than using a single byte. The codes for these characters are included in the Unicode space and are unremarkable.

The lengths of the generated strings form a distribution that centers on the current string length and ranges from 1 to 65K Unicode characters.

Supported Elements

- [String](#)

Hints

This mutator does not support hints.

21.11.52. StringUtf8Invalid

This mutator produces test cases for UTF-8 and ASCII String elements by generating new strings and then populating the strings with randomly-selected values.

The interesting part of this mutator flips the control bits of each generated character. The control bits manage the underlying storage for the byte sequence of the character. Each character ranges from 1 to 6 bytes. The codes for these characters are included in the Unicode space and are unremarkable.

The lengths of the generated strings form a distribution that centers on the current string length and ranges from 1 to 65K Unicode characters.

Supported Elements

- [String](#)

Hints

This mutator does not support hints.

21.11.53. StringUtf16BomLength

The *StringUtf16BomLength* mutator produces test cases for Unicode UTF-16 strings by altering an individual string in two ways:

- Adjust the length of the string using the [StringLengthVariance](#) mutator
- Insert 1 to 6 Unicode Byte Ordering Marks (BOMs) in arbitrary places in the string

BOMs are not considered characters, and using them can cause buffer length calculations to be incorrect.

BOM marks, whether for Big Endian-ness or for Little Endian-ness, are selected arbitrarily.

Supported Elements

- [String](#)

Hints

This mutator does not support any hints.

21.11.54. StringUtf16BomStatic

The `StringUtf16BomStatic` mutator produces test cases for Unicode UTF-16 strings by creating a mutated string in two ways:

- Create a string using the [StringStatic](#) mutator
- Inject 1 to 6 Unicode Byte Ordering Marks (BOMs) in arbitrary places in the string

BOM are not considered characters, and injecting BOMs into a string that might be filtered could cause the filtering not to work and the string would be accepted.

BOM marks, whether for Big Endian-ness or for Little Endian-ness, are selected arbitrarily.

Supported Elements

- [String](#)

Hints

This mutator does not support any hints.

21.11.55. StringUtf32BomLength

The *StringUtf32BomLength* mutator produces test cases for Unicode UTF-32 strings by altering an individual string in two ways:

- Adjust the length of the string using the [StringLengthVariance](#) mutator
- Insert 1 to 6 Unicode Byte Ordering Marks (BOMs) in arbitrary places in the string

BOM are not considered characters, and using them can cause buffer length calculations to be incorrect.

BOM marks, whether for Big Endian-ness or for Little Endian-ness, are selected arbitrarily.

Supported Elements

- [String](#)

Hints

This mutator does not support any hints.

21.11.56. StringUtf32BomStatic

The *StringUtf32BomStatic* mutator produces test cases for Unicode UTF-32 strings by creating a mutated string in two ways:

- Create a string using the [StringStatic](#) mutator
- Inject 1 to 6 Unicode Byte Ordering Marks (BOMs) in arbitrary places in the string

BOMs are not considered characters, and injecting BOMs into a string that might be filtered could cause the filtering not to work and the string would be accepted.

BOM marks, whether for Big Endian-ness or for Little Endian-ness, are selected arbitrarily.

Supported Elements

- [String](#)

Hints

This mutator does not support any hints.

21.11.57. StringXmlW3C

This mutator produces XML parser test cases for String elements. The test cases are defined in the [XML W3C Conformance Test Suite](#).

This mutator is enabled for String elements that have the [XML](#) hint.

Supported Elements

- [String](#)

Hints

XML

Enables W3C XML tests. The hint name is "XML". The hint value is "xml".

21.12. Common Attributes and Parameters

The following sections contain common attributes and parameters used by other Peach Pit files elements. Other sections of the documentation reference these attributes and parameters.

21.12.1. Constraint Attribute

The constraint attribute specifies a scripting expression that helps Peach identify whether the parser has cracked incoming data correctly for the data element. Peach uses the constraint expression as a post-data-cracking validation filter.

If the constraint expression evaluates `true`, the data successfully loaded into the element. Otherwise, a cracking error occurred and Peach will report the details of the error.

The constraint attribute does not affect how Peach mutates the value.



Constraints are only executed when parsing (cracking) data into a data element.



Constraints are typically slower than using either `Choice` or the *token* attribute.

The following special variables are available to the expression:

element

The data element instance.

value

The value of the data cracked into the data element. This value is cast to a string or to a byte array.

Example

```
<!-- Operate on number and check result -->
<Number name="constrainedNum" size="32" constraint="int(value) & 0xfefe == 5" />
```

```
<!-- Case insensitive equals -->
<String constraint="value.lower() == 'peach'" />

<!-- Value contains peach -->
<String constraint="value.find('peach') != -1" />
```

```
<!-- Length is less than 100 bytes -->
<Blob constraint="len(value) < 100" />
```

21.12.2. Endian Attribute

Specify the byte order of a field.

For more information about endian-ness, see [the Endian Wikipedia article](#).

Valid Values:

- big or network — Most significant byte first
- little — Least significant byte first

Examples:

```
<DataModel name="NumberExample6">
  <Number name="Hi5" value="AB CD" valueType="hex" size="16" signed="false" endian="big"
  />
</DataModel>
```

Produces the bytes in the following order.

```
AB CD
```

```
<DataModel name="NumberExample7">
  <Number name="Hi5" value="AB CD" valueType="hex" size="16" signed="false" endian="little"
  />
</DataModel>
```

Produces the bytes in the following order.

```
CD AB
```

21.12.3. Field

The Field element specifies the data that replaces the default value in a DataModel element.

Attributes

Required:

name

Name of the DataModel element.

Optional:

value

The value to set.

valueType

The format in which the default value is expressed (i.e. hex, string, or literal).

Examples

Example 296. Basic Example

Data used to overwrite *Good Afternoon World!* with *Hello World!*.

```
<DataModel name="TheDataModel">
  <Block name="Block1">
    <String name="Key" value="Output: " />
    <String name="Value" value="Good Afternoon World!" />
  </Block>
</DataModel>

<StateModel name="TheState">
  <State name="initial">
    <Action type="output">
      <DataModel ref="TheDataModel" />
      <Data name="SampleData">
        <Field name="Block1.Value" value="Hello World!" />
      </Data>
    </Action>
  </State>
</StateModel>
```

21.12.4. fileName

When used with the [Data](#) element, *fileName* can specify any of the following:

- A single file ("`sample.png`"),
- Multiple files by pointing to a folder ("`samples`"), or
- Multiple files by providing a file glob ("`samples_*.jpg`").

If a folder or file glob specifies multiple files, use the *switchCount* attribute with the [random strategy](#) to specify the number of iterations performed before switching to the next file.



Multiple files are ONLY supported by the random mutation strategy.

21.12.5. Hint

Hints are a [Mutator](#) extension; they can be attached to data elements to provide the Peach engine more information about parsed data (including how to treat it). For example, when a [String](#) contains a number, only the numerical tests contained within the string mutator would execute.

```
<String value="250">
  <Hint name="NumericalVarianceMutator-N" value="100" />
</String>
```

Available Hints:

- ArrayVarianceMutator-N
- BitFlipperMutator-N
- DWORDSliderMutator
- FiniteRandomNumbersMutator-N
- NumericalEdgeCaseMutator-N
- NumericalVarianceMutator-N
- SizedDataNumericalEdgeCasesMutator-N
- SizedDataVaranceMutator-N
- SizedNumericalEdgeCasesMutator-N
- SizedVaranceMutator-N
- type
- ValidValues

21.12.6. Length Attribute

The *length* attribute defines the Blob or String size in bytes.

Examples

```
<Blob name="MagicNumber" length="8" value="01 02 03 04 05 06 07 08" valueType="hex" />
<String name="Token" value="MAGIC" length="5"/>
```

21.12.7. Length Type Attribute

The *lengthType* attribute defines the units of measure of the *length* attribute.



A *length* attribute must be defined before using *lengthType*.

bits

Length is specified as a number of bits.

bytes

Length is specified as a number of bytes (default).

chars

Length is specified as a number of characters (only applies to [String](#)).

Examples

```
<Blob length="8" lengthType="bytes" value="01 02 03 04 05 06 07 08" valueType="hex" />  
<String value="MAGIC" length="5" lengthType="chars"/>  
<String value="MAGIC" lengthType="calc" lengthCalc="4+1"/>
```

21.12.8. Maximum Occurrence Attribute

maxOccurs specifies the maximum number of times an element can occur. Peach treats a data element with *maxOccurs* as an array.

Peach uses the occurrence attributes in cracking, producing, and mutating data.

The *maxOccurs* attribute specifies an upper limit on the number of array elements that can occur. A similar attribute, the [minOccurs](#) attribute specifies a lower limit on the number of array elements that can occur. When used together, these two attributes define a range of an element's occurrence.

Related attributes: [minOccurs](#), [occurs](#).

Examples

```
<!-- Can occur a maximum of two times -->  
<Block name="OtherThings" maxOccurs="2">  
  <String name="A" value="A" />  
  <String name="B" value="B" />  
  <String name="C" value="C" />  
</Block>  
  
<!-- Can occur a maximum of 1000 times -->  
<String name="OptionalValue" maxOccurs="1000" />
```

21.12.9. Minimum Occurrence Attribute

minOccurs specifies the minimum number of times an element can occur. Peach treats a data element with *minOccurs* as an array.

Occurrence attributes are used when cracking, producing, and mutating data.

The *minOccurs* attribute specifies a lower limit on the number of array elements that can occur. A similar attribute, the [maxOccurs](#) attribute specifies an upper limit on the number of array elements that can occur. When used together, these two attributes define a range of an element's occurrence.

Related attributes: [maxOccurs](#), [occurs](#).

Examples

```
<!-- Must occur at least twice -->
<Block name="OtherThings" minOccurs="2">
  <String name="A" value="A" />
  <String name="B" value="B" />
  <String name="C" value="C" />
</Block>

<!-- Can occur 0 or more times -->
<String name="OptionalValue" minOccurs="0" />
```

21.12.10. Mutable Attribute

Mutable declares whether to fuzz this element, block, or data type. The default value is true, meaning that Peach fuzzes the element, block, or data type.



Marking an element as non-mutable disables the mutators that normally operate on that element. Even when an element is marked as non-mutable, it may still be modified as the fuzzer fuzzes the other elements.



Marking elements as non-mutable usually leads to missing faults. With that in mind, withhold use of this attribute unless:

- 1) You know that the marked elements have already undergone fuzzing, and
- 2) You are very familiar with the effects of this attribute (that is, you know what you are doing).

Examples

```
<DataModel name="Header">
  <Number name="ReservedForFutureuse" size="8" mutable="false" />
  <Number size="8" />
  <Number size="8" />
  <Number size="8" />
<DataModel>
```

21.12.11. Name Attribute

Virtually all the Peach Pit file elements support the *name* attribute. Names are used for readability and [referencing](#) other elements in a Pit file.



Names should not contain punctuation. Punctuation marks like period (.), slash (\), and colon (:) have special [references](#) meanings.

Names are case sensitive and must be unique at the current scope level.

Examples

Correct:

The following example has unique names for each element.

```
<Block name="Header">
  <Number name="Value1" size="8"/>
  <Number name="Value2" size="8"/>
  <Number name="Value3" size="8"/>
  <Number name="Value4" size="8"/>
</Block>
```

The following example does not provide names for all values. This practice is okay, as long as you don't want to access the unnamed element.

```
<Block name="Header">
  <Number size="8"/>
  <Number size="8"/>
  <Number size="8"/>
  <Number size="8"/>
</Block>
```

Incorrect:

The following has duplicate names at the same document level. This causes an error.

```
<Block name="Header">
  <Number name="Value" size="8"/>
  <Number name="Value" size="8"/>
  <Number name="Value" size="8"/>
  <Number name="Value" size="8"/>
</Block>
```

21.12.12. Occurs Attribute

occurs is a combination of [minOccurs](#) and [maxOccurs](#).

The *occurs* value (for the specified data element) is the exact number of element occurrences. Peach treats *occurs* as an array.

Example

The following block is an array that occurs a fixed number of times.

```
<Block name="OccursBlock" occurs="5">
  <String name="occurs" value="A"/>
</Block>
```

Produces the output:

```
AAAAA
```

21.12.13. State onStart Attribute

The *onStart* attribute is a python expression that is evaluated prior to performing the actions in a [State](#). This expression can be used to increment a counter or to call some other function defined in a separate file included with the [Import](#) element.



onComplete is the sibling attribute.

Keeping State

State bags are defined as `Dictionary<string, object>` in C#. If you use one of two state bags exposed through the `RunContext` instance, you can store state during the lifetime of the current iteration or for the entire fuzzing session.

A full example of using the iteration state bag is provided in the examples section.

Using the Session State:

This state stored in this state bag persists for the entire fuzzing session, from the point of creation to the end of the session.

```
context.stateStore['my_counter'] = 0
```

Using the Iteration State:

The state stored in this state bag exists for the current iteration.

```
context.iterationStateStore['my_counter'] = 0
```

Syntax

```
<State name="Initial" onStart="xyz.reset_counter(self)">  
  <Action type="output">  
    <DataModel ref="TheDataModel" />  
  </Action>  
</State>
```

Scripting Scope

context

RunContext instance

state

State instance

stateModel

StateModel instance

self

State instance

test

Test instance

Examples**Example 297. Display a Message from *onStart***

This example prints a message when the *onStart* statement is run.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="File1Model">
        <String value="Data for the file" />
    </DataModel>

    <StateModel name="State" initialState="Initial">
        <State name="Initial" onStart="print 'Hello from onStart!'">
            <Action type="output">
                <DataModel ref="File1Model" />
            </Action>

            </State>
        </StateModel>

        <Test name="Default">
            <StateModel ref="State"/>

            <Publisher class="File">
                <Param name="FileName" value="fuzzed.txt" />
            </Publisher>
        </Test>
    </Peach>
```

The following is the resulting output:

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 23438.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Hello from onStart! ①
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(17 bytes)
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.FilePublisher stop()

[*] Test 'Default' finished.
```

① Output from *onStart*

Example 298. Loop Using Iteration State Bag

This example uses the iteration state bag to simulate a for loop.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String value="Looping!\n" />
    </DataModel>

    <StateModel name="State" initialState="Initial">
        <State name="Initial" onStart="context.iterationStateStore['count'] = 0">

            <!-- Initialize our counter -->
            <Action type="changeState" ref="Loop" />

        </State>

        <State name="Loop" onStart="context.iterationStateStore['count'] =
context.iterationStateStore['count'] + 1">

            <!-- onStart will increment counter -->
            <Action type="output">
                <DataModel ref="TheDataModel" />
            </Action>

            <!-- Loop until our counter is greater than 3 -->
            <Action type="changeState" ref="Loop"
when="context.iterationStateStore['count'] < 3" />

        </State>

    </StateModel>

    <Test name="Default">
        <StateModel ref="State"/>

        <Publisher class="Console"/>
    </Test>

</Peach>

```

The following is the resulting output:

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 28742.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.ChangeState
Peach.Core.Dom.Action Changing to state: Loop
Peach.Core.Dom.StateModel Run(): Changing to state "Loop".
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(9 bytes)
Looping! ①
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.ChangeState
Peach.Core.Dom.Action Changing to state: Loop
Peach.Core.Dom.StateModel Run(): Changing to state "Loop".
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(9 bytes)
Looping! ②
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.ChangeState
Peach.Core.Dom.Action Changing to state: Loop
Peach.Core.Dom.StateModel Run(): Changing to state "Loop".
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(9 bytes)
Looping! ③
Peach.Core.Dom.Action Run: action 'Action_1' when returned false ④
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

① Output from iteration 1

② Output from iteration 2

③ Output from iteration 3

④ *when* expression returning false causing exit from loop

21.12.14. State `onComplete` Attribute

The `onComplete` attribute is a python expression that is evaluated after performing the actions in a [State](#). This expression can be used to increment a counter or to call some other function defined in a separate file included with the [Import](#) element.



`onStart` is the sibling attribute.

Keeping State

State bags are defined as `Dictionary<string, object>` in C#. If you use one of two state bags exposed through the `RunContext` instance, you can store state during the lifetime of the current iteration or for the entire fuzzing session.

A full example of using the iteration state bag is provided in the examples section.

Using the Session State:

This state stored in this state bag persists for the entire fuzzing session, from the point of creation to the end of the session.

```
context.stateStore['my_counter'] = 0
```

Using the Iteration State:

The state stored in this state bag exists for the current iteration.

```
context.iterationStateStore['my_counter'] = 0
```

Syntax

```
<State name="Initial" onComplete="xyz.reset_counter(self)">  
  <Action type="output">  
    <DataModel ref="TheDataModel" />  
  </Action>  
</State>
```

Scripting Scope

context

RunContext instance

state

State instance

stateModel

StateModel instance

self

State instance

test

Test instance

Examples**Example 299. Display a Message from *onComplete***

This example prints a message when the *onComplete* statement is run.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="File1Model">
        <String value="Data for the file" />
    </DataModel>

    <StateModel name="State" initialState="Initial">
        <State name="Initial" onComplete="print 'Hello from onComplete!'">
            <Action type="output">
                <DataModel ref="File1Model" />
            </Action>
        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="State"/>

        <Publisher class="File">
            <Param name="FileName" value="fuzzed.txt" />
        </Publisher>  </Test>
    </Peach>

```

This is the resulting output:

```
> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 27537.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(17 bytes)
Hello from onComplete! ①
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.FilePublisher stop()

[*] Test 'Default' finished.
```

① Output from *onComplete*

Example 300. Loop Using Iteration State Bag

This example uses the iteration state bag to simulate a for loop.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String value="Looping!\n" />
    </DataModel>

    <StateModel name="State" initialState="Initial">
        <State name="Initial" onStart="context.iterationStateStore['count'] = 0">

            <!-- Initialize our counter -->
            <Action type="changeState" ref="Loop" />

        </State>

        <State name="Loop" onComplete="context.iterationStateStore['count'] =
context.iterationStateStore['count'] + 1">

            <!-- onStart will increment counter -->
            <Action type="output">
                <DataModel ref="TheDataModel" />
            </Action>

            <!-- Loop until our counter is greater than 3 -->
            <Action type="changeState" ref="Loop"
when="context.iterationStateStore['count'] < 3" />

        </State>

    </StateModel>

    <Test name="Default">
        <StateModel ref="State"/>

        <Publisher class="Console"/> </Test>

    </Peach>

```

This is the resulting output:

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 28742.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.ChangeState
Peach.Core.Dom.Action Changing to state: Loop
Peach.Core.Dom.StateModel Run(): Changing to state "Loop".
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(9 bytes)
Looping! ①
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.ChangeState
Peach.Core.Dom.Action Changing to state: Loop
Peach.Core.Dom.StateModel Run(): Changing to state "Loop".
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(9 bytes)
Looping! ②
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.ChangeState
Peach.Core.Dom.Action Changing to state: Loop
Peach.Core.Dom.StateModel Run(): Changing to state "Loop".
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(9 bytes)
Looping! ③
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.ChangeState
Peach.Core.Dom.Action Changing to state: Loop
Peach.Core.Dom.StateModel Run(): Changing to state "Loop".
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(9 bytes)
Looping! ④
Peach.Core.Dom.Action Run: action 'Action_1' when returned false ⑤
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

- ① Output from iteration 1
- ② Output from iteration 2
- ③ Output from iteration 3
- ④ Output from iteration 4
- ⑤ *when* expression returning false causing exit from loop

21.12.15. Action onComplete Attribute

The `onComplete` attribute is an expression that is evaluated after performing an action. This expression can be used to increment a counter or to perform other functions defined in separate files, such as those brought into the model using the [Import](#) tag.

Keeping State

Peach provides a mechanism for the user to store state during for the lifetime of the current iteration, or the fuzzing session. This is accomplished using one of two state bags exposed through the `RunContext` instance. The state bags are defined as `Dictionary<string, object>` in C#. A full example of using the iteration state bag is provided in the examples section.

Using the Session State:

This state stored in this state bag persists for the entire fuzzing session, from the point of creation to the end of the fuzzing session.

```
context.stateStore['my_counter'] = 0
```

Using the Iteration State:

The state stored in this state bag exists only for the current iteration.

```
context.iterationStateStore['my_counter'] = 0
```

Syntax

```
<State name="Initial">
  <Action type="changeState" ref="NextState" onComplete="xyz.reset_counter(self)"/>
</State>
```

Scripting Scope

action

Action instance

context

RunContext instance

state

State instance

stateModel

StateModel instance

self

Action instance

test

Test instance

Examples

Example 301. Display a Message from *onComplete*

This example prints a message when the *onComplete* expression is run.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="File1Model">
        <String value="Data for the file" />
    </DataModel>

    <StateModel name="State" initialState="Initial">
        <State name="Initial">

            <Action type="output" onComplete="print 'Hello from onComplete!'">
                <DataModel ref="File1Model" />
            </Action>

        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="State"/>

        <Publisher class="File">
            <Param name="FileName" value="fuzzed.txt" />
        </Publisher>  </Test>

    </Peach>

```

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 27073.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(17 bytes)
Hello from onComplete! ①
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.FilePublisher stop()

[*] Test 'Default' finished.

```

① Output from *onComplete*

Example 302. Loop Using Iteration State Bag

This example uses the iteration state bag to simulate a for loop.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String value="Looping!\n" />
    </DataModel>

    <StateModel name="State" initialState="Initial">
        <State name="Initial">

            <!-- Initialize our counter -->
            <Action type="changeState" ref="Loop"
onStart="context.iterationStateStore['count'] = 0" />

        </State>

        <State name="Loop">

            <!-- onComplete will increment counter -->
            <Action type="output" onComplete="context.iterationStateStore['count'] =
context.iterationStateStore['count'] + 1">
                <DataModel ref="TheDataModel" />
            </Action>

            <!-- Loop until our counter is greater than 3 -->
            <Action type="changeState" ref="Loop"
when="context.iterationStateStore['count'] < 3" />

        </State>

    </StateModel>

    <Test name="Default">
        <StateModel ref="State"/>

        <Publisher class="Console"/> </Test>
    </Peach>

```

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 28742.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.ChangeState
Peach.Core.Dom.Action Changing to state: Loop
Peach.Core.Dom.StateModel Run(): Changing to state "Loop".
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(9 bytes)
Looping! ①
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.ChangeState
Peach.Core.Dom.Action Changing to state: Loop
Peach.Core.Dom.StateModel Run(): Changing to state "Loop".
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(9 bytes)
Looping! ②
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.ChangeState
Peach.Core.Dom.Action Changing to state: Loop
Peach.Core.Dom.StateModel Run(): Changing to state "Loop".
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(9 bytes)
Looping! ③
Peach.Core.Dom.Action Run: action 'Action_1' when returned false ④
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

① Output from iteration 1

② Output from iteration 2

③ Output from iteration 3

④ *when* expression returning false causing exit from loop

21.12.16. Action onStart Attribute

The onStart attribute is an expression that is evaluated before performing an action. This expression can be used to increment a counter or to perform other functions defined in separate files, such as those brought into the model using the [Import](#) tag.



If the *onStart* expression is part of the very first explicit action, it executes prior to the implicit actions.

Keeping State

Peach provides a mechanism for the user to store state during for the lifetime of the current iteration, or the fuzzing session. This is accomplished using one of two state bags exposed through the RunContext instance. The state bags are defined as *Dictionary<string, object>* in C#. A full example of using the iteration state bag is provided in the examples section.

Using the Session State:

This state stored in this state bag persists for the entire fuzzing session, from the point of creation to the end of the fuzzing session.

```
context.stateStore['my_counter'] = 0
```

Using the Iteration State:

The state stored in this state bag exists only for the current iteration.

```
context.iterationStateStore['my_counter'] = 0
```

Syntax

```
<State name="Initial">  
  <Action type="changeState" ref="NextState" onStart="xyz.reset_counter(self)"/>  
</State>
```

Scripting Scope

action

Action instance

context

RunContext instance

state

State instance

stateModel

StateModel instance

self

Action instance

test

Test instance

Examples**Example 303. Display a Message from *onStart***

This example prints a message when the *onStart* expression is run.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="File1Model">
        <String value="Data for the file" />
    </DataModel>

    <StateModel name="State" initialState="Initial">
        <State name="Initial">

            <Action type="output" onStart="print 'Hello from onStart!'">
                <DataModel ref="File1Model" />
            </Action>

        </State>
    </StateModel>

    <Test name="Default">
        <StateModel ref="State"/>

        <Publisher class="File">
            <Param name="FileName" value="fuzzed.txt" />
        </Publisher>  </Test>

    </Peach>

```

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 27537.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Hello from onStart! ①
Peach.Core.Publishers.FilePublisher start()
Peach.Core.Publishers.FilePublisher open()
Peach.Core.Publishers.FilePublisher output(17 bytes)
Peach.Core.Publishers.FilePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.FilePublisher stop()

[*] Test 'Default' finished.

```

① Output from *onStart*

Example 304. Loop Using Iteration State Bag

This example use the iteration state bag to simulate a for loop.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String value="Looping!\n" />
    </DataModel>

    <StateModel name="State" initialState="Initial">
        <State name="Initial">

            <!-- Initialize our counter -->
            <Action type="changeState" ref="Loop"
onStart="context.iterationStateStore['count'] = 0" />

        </State>

        <State name="Loop">

            <!-- onStart will increment counter -->
            <Action type="output" onStart="context.iterationStateStore['count'] =
context.iterationStateStore['count'] + 1">
                <DataModel ref="TheDataModel" />
            </Action>

            <!-- Loop until our counter is greater than 3 -->
            <Action type="changeState" ref="Loop"
when="context.iterationStateStore['count'] < 3" />

        </State>

    </StateModel>

    <Test name="Default">
        <StateModel ref="State"/>

        <Publisher class="Console"/> </Test>
    </Peach>

```

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 28742.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.ChangeState
Peach.Core.Dom.Action Changing to state: Loop
Peach.Core.Dom.StateModel Run(): Changing to state "Loop".
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(9 bytes)
Looping! ①
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.ChangeState
Peach.Core.Dom.Action Changing to state: Loop
Peach.Core.Dom.StateModel Run(): Changing to state "Loop".
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(9 bytes)
Looping! ②
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.ChangeState
Peach.Core.Dom.Action Changing to state: Loop
Peach.Core.Dom.StateModel Run(): Changing to state "Loop".
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(9 bytes)
Looping! ③
Peach.Core.Dom.Action Run: action 'Action_1' when returned false ④
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

① Output from iteration 1

② Output from iteration 2

③ Output from iteration 3

④ *when* expression returning false causing exit from loop

21.12.17. Ref Attribute

The ref attribute specifies a reference to another element by name or by class structure. References can be relative or fully qualified.

Examples

Relative by Name:

```
<Block ref="ElementName"/>
```

Fully qualified with namespace prefix:

```
<Block ref="namespace:parent.parent.child"/>
```

Fully qualified:

```
<Block ref="parent.parent.child"/>
```

21.12.18. Signed Attribute

Specifies whether a number contains a signed value. The default value is true.

signed is useful when parsing length data.

Examples:

Specify a data element is unsigned by setting the signed attribute to "false".

```
<DataModel name="NumberExample3">
  <Number name="Hi5" value="5" size="32" signed="false"/>
</DataModel>
```

21.12.19. Size Attribute

The *size* attribute specifies the size of a data element in BITS. Size typically applies to **Number** or bit **Flags** elements.

Examples

This example creates an 8-bit number with a default value of 42.

```
<Number name="TheAnswer" size="8" value="42" />
```

21.12.20. Token Attribute

A token is a data element that Peach seeks when parsing incoming data. Tokens do not affect how elements are mutated. If a token attribute is specified the element must have a default value specified in the XML.

Examples

If you are trying to parse "length:42" and you want the header and value to be considered separate elements, label the colon delimiter ":" as a token.

33. Example Data to Crack

```
length:42
```

The following model looks for a ":" and puts the value of "length" into the header string and a 42 into the val string.

If a ":" is not found in the data stream, an exception will be raised and cracking will fail.

```
<String name="header" />
<String name="delimiter" value=":" token="true"/>
<String name="val" />
```

34. Example Data

```
Host: www.peachfuzzer.com\r\n
```

35. Example xml to parse

```
<String name="header" />
<String name="delimiter" value=":" token="true"/>
<String name="val" />
<String name="CRLF" value="\r\n" token="true" />
```

21.12.21. Value Attribute

The *value* attribute specifies a default value for a data element when Peach is not mutating the element.

All Peach data element types support *value*.

Examples

```
<Number name="five" size="8" value="5" />  
<String name="five" value="five" />
```

21.12.22. ValueType Attribute

The *valueType* attribute specifies what type of data is contained in the *value* attribute. Valid values are **string**, **hex**, and **literal**.

string

String is the default *valueType*.

The value provided is a string value. String value is converted into the appropriate data element type. The *Number* element expects numerical values while *String* and *Blob* accept any values.

hex

The value specified as a stream of hex bytes.

Peach is fairly good about figuring out different types of hex strings like "0x00 0x00" or "\x00 \x00" or "00 00 00 00 00".

literal

Specify a python or ruby statement that evaluates to the wanted value.

The evaluation occurs only once when the Pit is parsed into an internal DOM tree.



A literal should **not** be used in place of a [Relation](#) or a [Fixup](#). Literals are only to be used for quick one-time expressions.

Examples

Example 305. string

A numerical representation of value 65.

```
<Number valueType="string" value="65" />
```

Output

```
A
```

A string representation of value 65.

```
<Number valueType="string" value="65" />
```

Output

```
65
```

Example 306. hex

A numerical representation of value 0x41.

```
<String valueType="hex" value="41" />
```

Output

```
A
```

A numerical representation of value 0x41.

```
<Number valueType="hex" value="41" />
```

Output

```
A
```



The hex values must be EVEN numbered! This is WRONG: "0x000" and this is CORRECT: "0x0000"

Example 307. literal

```
<String valueType="literal" value="(1 + 2 + 3)**2" />
```

Output

```
36
```

```
<String valueType="literal" value="'hello world!'.upper()" />
```

Output

```
HELLO WORLD!
```

21.12.23. Action when Attribute

The *when* attribute is a Python Boolean expression that is evaluated before performing an action. If the expression evaluates to true, the action is performed; otherwise, the action is skipped.

Keeping State

Peach provides a mechanism for the user to store state during for the lifetime of the current iteration, or the fuzzing session. This is accomplished using one of two state bags exposed through the RunContext instance. The state bags are defined as *Dictionary<string, object>* in C#. A full example of using the iteration state bag is provided in the examples section.

Using the Session State:

This state stored in this state bag persists for the entire fuzzing session, from the point of creation to the end of the fuzzing session.

```
context.stateStore['my_counter'] = 0
```

Using the Iteration State:

The state stored in this state bag exists only for the current iteration.

```
context.iterationStateStore['my_counter'] = 0
```

Syntax

```
<State name="Initial">  
    <Action type="changeState" ref="State2" when=  
        "int(state.actions[0].dataModel.find('Type').DefaultValue) == 2"/>  
</State>
```

Scripting Scope

action

Action instance

context

RunContext instance

state

State instance

stateModel

StateModel instance

self

Action instance

test

Test instance

Examples**Example 308. Conditional *changeState* Based on *when* Expression**

The following example changes its behavior based on input received from the target client.

```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="Ping">
        <Choice>
            <String name="PingPingStr" value="PINGPING" token="true" />
            <String name="PingStr" value="PING" token="true"/>
        </Choice>
    </DataModel>

    <DataModel name="Pong">
        <String value="PONG" />
    </DataModel>

    <DataModel name="PongPong">
        <String value="PONGPONG" />
    </DataModel>

    <StateModel name="TheStateModel" initialState="InitialState">
        <State name="InitialState">

            <Action type="accept" />

            <Action type="input">
                <DataModel ref="Ping"/>
            </Action>
        </State>
    </StateModel>

```

```

<!-- Switch states only when input was PINGPING -->
<Action type="changeState" ref="PongPongBack"
    when="state.actions[1].dataModel.find('PingPingStr') != None" />

<Action type="output">
    <DataModel ref="Pong"/>
</Action>

</State>

<!-- This state is only reached when input was PINGPING -->
<State name="PongPongBack">

<Action type="output">
    <DataModel ref="PongPong"/>
</Action>

</State>

</StateModel>

<Test name="Default">
    <StateModel ref="TheStateManager"/>
    <Publisher class="TcpListener">
        <Param name="Interface" value="0.0.0.0" />
        <Param name="Port" value="31337" />
        <Param name="AcceptTimeout" value="10000" />
        <Param name="Timeout" value="10000" />
    </Publisher>

    <Logger class="File" >
        <Param name="Path" value="logs"/>
    </Logger>
</Test>
</Peach>

```

Example 309. Loop Using Iteration State Bag

This example uses the iteration state bag to simulate a for loop.

```

<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://peachfuzzer.com/2012/Peach peach.xsd">

    <DataModel name="TheDataModel">
        <String value="Looping!\n" />
    </DataModel>

    <StateModel name="State" initialState="Initial">
        <State name="Initial">

            <!-- Initialize our counter -->
            <Action type="changeState" ref="Loop"
onStart="context.iterationStateStore['count'] = 0" />

        </State>

        <State name="Loop">

            <!-- onComplete will increment counter -->
            <Action type="output" onComplete="context.iterationStateStore['count'] =
context.iterationStateStore['count'] + 1">
                <DataModel ref="TheDataModel" />
            </Action>

            <!-- Loop until our counter is greater than 3 -->
            <Action type="changeState" ref="Loop"
when="context.iterationStateStore['count'] < 3" />

        </State>

    </StateModel>

    <Test name="Default">
        <StateModel ref="State"/>

        <Publisher class="Console"/> </Test>
    </Peach>

```

```

> peach -1 --debug example.xml

[*] Test 'Default' starting with random seed 28742.

[R1,-,-] Performing iteration
Peach.Core.Engine runTest: Performing recording iteration.
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.ChangeState
Peach.Core.Dom.Action Changing to state: Loop
Peach.Core.Dom.StateModel Run(): Changing to state "Loop".
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher start()
Peach.Core.Publishers.ConsolePublisher open()
Peach.Core.Publishers.ConsolePublisher output(9 bytes)
Looping! ①
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.ChangeState
Peach.Core.Dom.Action Changing to state: Loop
Peach.Core.Dom.StateModel Run(): Changing to state "Loop".
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(9 bytes)
Looping! ②
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.ChangeState
Peach.Core.Dom.Action Changing to state: Loop
Peach.Core.Dom.StateModel Run(): Changing to state "Loop".
Peach.Core.Dom.Action Run: Adding action to controlRecordingActionsExecuted
Peach.Core.Dom.Action ActionType.Output
Peach.Core.Publishers.ConsolePublisher output(9 bytes)
Looping! ③
Peach.Core.Dom.Action Run: action 'Action_1' when returned false ④
Peach.Core.Publishers.ConsolePublisher close()
Peach.Core.Engine runTest: context.config.singleIteration == true
Peach.Core.Publishers.ConsolePublisher stop()

[*] Test 'Default' finished.

```

① Output from iteration 1

② Output from iteration 2

③ Output from iteration 3

④ *when* expression returning false causing exit from loop

21.12.24. xpath

22. Extending Peach

Peach Fuzzer Professional was designed to be extended, allowing you to increase Peach's capabilities. The most common reasons for extending Peach include:

- Custom validation algorithms (such as checksums)
- Custom publishers (I/O adapters)
- Monitoring

Peach can use custom implementations of the following to extend its functionality:

- [Agents](#)
- [Analyzers](#)
- [Fixups](#)
- [Loggers](#)
- [Monitors](#)
- [Publishers](#)
- [Transformers](#)

All extensions follow the same paradigm:

1. Create a C# class that derives from the appropriate Peach base class.
2. Decorate the class with the appropriate plugin attribute.
3. Compile the class into an assembly and place the assembly in the Plugins directory.

This section covers the most common types of extensions for Peach.

22.1. Peach Plug-ins

If your target has something in its data model that Peach does not support (like a custom checksum algorithm or a custom data format), you may need to extend Peach's functionality by using Peach plugins.

Peach plugins are either C# class libraries or python files that implement C# classes. Custom Peach plugins should be placed in the [Plugins](#) subdirectory of the Peach installation folder. When Peach starts it scans all `.dll` and `.py` files in the folder looking for plugins.

To see a list of all detected plugins and their parameters, run `peach --showenv`.

All Peach plugins take a C# dictionary as a construction argument. This argument dictionary is populated with the Param elements names and values defined in the Peach Pit.

Peach plugins use ParameterAttribute to define all of their supported configuration arguments (name, type, description, and optional default value).

- The name and type specified in the ParameterAttribute must match a public property on the plugin class.
- If the name parameter happens to be a C# reserved keyword, prefix the associated class property with an underscore.
- The default value is always given as a string, regardless of the type of the parameter.



If the default value is omitted from the ParameterAttribute, Peach treats the parameter as required.

ParameterParser.Parse() uses the arguments dictionary to automatically populate all plugin properties.

- When populating the properties, type conversion converts the string specified in the pit to the concrete type defined in the plugin.
- When performing automatic type conversion, the ParameterParser supports System.Enum as well as any type that implements IConvertible.

This method uses reflection to:

- Enforce missing required parameters.
- Initialize optional parameters from default values.
- Convert types from string to a concrete type.
- Validate all type conversions
 - -100 is valid int.
 - -100 is not a valid byte.

General Peach plugin best practices:

- Use stream operations whenever possible.
- Avoid using large arrays of contiguous data.
- Validate parameter types in the plugin constructors.
- Provide useful error messages.
- Throw PeachException() when non-recoverable errors occur such as:
 - A parameter is out of range.
 - A user lacks appropriate permissions.
- Throw SoftException when recoverable errors occur such as:
 - A file is in use.

- An error occurs when starting a process.
- A socket connection is refused.

22.1.1. Examples

Example 310. Transformer

This example shows the arguments in the JsEncode transformer constructor.

```
[Transformer("JsEncode")]
public class JsEncode : Transformer
{
    public JsEncode(Dictionary<string, Variant> args)
        : base(args)
    {
    }
}
```

Example 311. Fixup

This example shows the class definition for a custom fixup called "MyFixup".

```
[Fixup("MyFixup", true)]
public class MyFixup : Peach.Core.Fixup
{}
```

Example 312. ParameterAttribute

This example shows the ParameterAttribute usage.

```
[Transformer("MyTransformer")]
[Parameter("level", typeof(int), "Specifies the level, default: 9", "9")]
[Parameter("class", typeof(string), "Specifies the class, required!")]
public class MyTransformer : Transformer
{
    // Type matches typeof(int) in ParameterAttribute
    public int level { get; set; }

    // Name is '_class' since 'class' is a C# keyword
    public string _class { get; set; }

    public MyTransformer(Dictionary<string, Variant> args)
        : base(args)
    {
    }
}
```

Example 313. Transformer

This example shows the class definition for a custom transformer called "MyTransformer".

```
[Transformer("MyTransformer", true)]
public class MyTransformer : Peach.Core.Transformer
{}
```

Example 314. Publisher

This example shows the class definition for a custom publisher called "MyPublisher".

```
[Publisher("MyPublisher", true)]
public class MyPublisher : Peach.Core.Publisher
{}
```

Example 315. Monitor

This example shows the class definition for a custom monitor called "MyMonitor".

```
[Monitor("MyMonitor", true)]
public class MyMonitor : Peach.Core.Monitor
{}
```

22.2. Fixup

A fixup computes a value based on one or more data elements. Since the value is based on other elements, each fixup tracks whenever a referenced element changes so it can recompute its value. For example, the Crc32 fixup needs to know whenever the referenced element changes, so that a new CRC can be computed.

Each fixup implementation needs to implement a single function: `fixupImpl()`. This function returns the result of the fixup computation.

Since fixups compute their value from other elements, each fixup must register its dependent elements with the fuzzing engine. The tracking of dependent elements is important so that the fixup will recompute its value when any of its dependent elements are mutated. The common convention for specifying dependent elements to a fixup is to use a `ref` parameter.

Each Fixup implementation must tell the base Fixup class which parameters refer to data elements. This is accomplished by passing the appropriate parameter names to the base fixup constructor. Inside of `fixupImpl()`, the resolved data element references are obtained through the dictionary `this.elements`. The key to this dictionary is the name of the parameter (eg: `ref`) and the value is the appropriate data element.

Example 316. Fixup that references a dependent element

Consider a custom fixup that needs to MD5 an element named *DateHeader*.

First, use a *ref* param in the pit to tell the fixup the specific element to target.

```
<Fixup class="DateMd5">
  <Param name="ref" value="DateHeader" />
</Fixup>
```

Second, obtain the target element in the fixupImpl().

```
protected override Variant fixupImpl()
{
    // Find the element specified in the 'ref' parameter
    var dateHeaderElement = this.elements["ref"];
    // Compute the MD5 of said element's value
    return MD5(dateHeaderElement.Value);
}
```

It is important to maintain the element type information when returning results from a fixup. The data element type returned by the fixup should match the data element type of the resulting field; the only exception is a string that can be the resulting field when the fixup element inputs a number.

If the fixup evaluates to an integer, an int type should be returned. Similarly, if the fixup evaluates to a string, a string type should be returned. This allows Peach to perform proper byte encoding when creating the final value for a data element.

Because the Crc fixup returns its value in an unsigned integer type, Peach outputs the bytes in the endianness defined on the parent data element.

22.2.1. Examples

Example 317. Single data element

This is an example of a fixup with a single data element reference.

```
[Fixup("CustomFixupOne", true)]
[Parameter("ref", typeof(string), "Reference to data element")]
[Serializable]
public class CustomFixupOne : Fixup
{
    public CustomFixupThree(DataElement parent, Dictionary<string, Variant> args)
        : base(parent, args, "ref")
    {
    }
}
```

Example 318. Multi-data element

This is an example of a fixup with three data element references.

```
[Fixup("CustomFixupThree", true)]
[Parameter("refOne", typeof(string), "Reference to first data element")]
[Parameter("refTwo", typeof(string), "Reference to second data element")]
[Parameter("refThree", typeof(string), "Reference to third data element")]
[Serializable]
public class CustomFixupThree : Fixup
{
    public CustomFixupThree(DataElement parent, Dictionary<string, Variant> args)
        : base(parent, args, "refOne", "refTwo", "refThree")
    {
    }
}
```

Example 319. fixupImpl function

This is an example of the fixupImpl function of a Crc fixup.

```
protected override Variant fixupImpl()
{
    // Get the element we need to compute the CRC of
    var elem = this.elements["ref"];
    // Get the stream of data for the target element
    var data = elem.Value;

    // Ensure we are at the beginning of the data
    data.Seek(0, System.IO.SeekOrigin.Begin);

    // Initialize the CRCTool
    CRCTool crcTool = new CRCTool();
    crcTool.Init(CRCTool.CRCCode.CRC32);

    // Return the CRC of data as a uint
    return new Variant((uint)crcTool.crcTableFast(data));
}
```

22.3. Monitor

Custom monitors must implement the following functions:

- StopMonitor()
- SessionStarting()
- SessionFinished()
- IterationStarting()
- IterationFinished()
- DetectedFault()
- GetMonitorData()
- MustStop()
- Message()

The SessionStarting and SessionFinished functions are called once per fuzzing session. These functions are responsible for any initialization and cleanup required by the monitor. Monitors can run remotely and multiple monitors can be defined in a Peach pit. It is possible that calling SessionStarting on the first monitor triggers some behavior on the target that allows subsequent monitors to be able to start without error. This means things like parameter validation and initialization should occur in the

SessionStarting function as opposed to the monitor's constructor.

The IterationStarting and IterationFinished functions are called once per fuzzing iteration. All per-iteration logic is implemented in these functions. Monitors that perform per-iteration process control will start and stop the target process in these functions. Most importantly, any per-iteration state must be reset in calls to IterationFinished(). The current fuzzing iteration is driven by the Peach engine, and monitors must not use past iterations to make any assumptions about future iterations.

The Message function is used to signal monitors at specific places in the StateModel execution. This functionality is primarily used for state synchronization. It allows the monitor to pause the execution of the StateModel at a specific point until some desired event happens. For example, a custom monitor could use the Message function to wait until a proprietary embedded device has rebooted and is ready to accept input data.

It is important to note that call actions on the "Peach.Agent" publisher result in the Message function being called on every monitor. Each monitor must filter for their desired messages, ignoring all unexpected message calls by returning null.

The DetectedFault and GetMonitorData functions are used to report faults back to Peach. At the end of each iteration, the Peach engine calls DetectedFault() on every monitor. If any monitor reports that a fault occurred, Peach calls GetMonitorData() on every monitor. The GetMonitorData function returns a Fault record, and the type of the record can be either Fault or Data.

Fault Detection:

- Return true from DetectedFault().
- Any monitor that detects a fault causes Peach to call ALL monitors to return monitor data.
- GetMonitorData() returns applicable data.
- Returned fault can be **fault** or **data**.
 - Debugger stack trace is **fault**.
 - Network packet capture is **data**.
 - Log file is **data**.
- Faults can include a hash for bucketing information.

Monitor best practices:

- Indicate errors with SoftExceptions().
- Keep monitors simple.
- Prefer multiple small monitors.
- Agent/Monitor order within a pit is honored by Peach.
 - Starting functions are called in order.
 - Finished functions are called in reverse order.

- Avoid one large complicated monitor.
- Remember monitors can run in remote agents.
- Maintain cross platform compatibility.
 - Windows Peach with Linux Agent.

22.4. Publisher

Publishers need to be extended whenever Peach needs to send or receive data through a custom IO channel. Publishers support both a *stream* view (open, input, output, close) and a *function* view (call, setProperty, getProperty). Each publisher method corresponds to a single action type used in the StateModel. While publishers can support all action types, most do not need to. For example, the File publisher does not implement call, accept, setProperty and getProperty. It is up to the developer to implement the appropriate set of functions.

All of the publisher's public functions are implemented in the base class. The public functions ensure the publisher is in the proper state and that they call the protected implementation functions when needed. For example, the public open function calls the protected OnOpen function only when the publisher is not open. When developers implement custom publishers, they override the protected functions. The default implementation of these protected functions is to throw a NotImplementedException().

The list of functions that users may override are:

- OnStart()
- OnStop()
- OnOpen()
- OnClose()
- OnAccept()
- OnInput()
- OnOutput()
- OnCall()
- OnSetProperty()
- OnGetProperty()

The OnStart and OnStop functions perform initialization and teardown. These functions are normally called once per test. The Peach engine automatically calls stop on all publishers when the test completes. The OnOpen and OnClose functions control access to the underlying resource. These functions are normally called once per iteration. The Peach engine automatically calls close on all publishers when each iteration completes.

The OnOutput function writes all the provided data to the underlying resource. The data is provided in

a stream, and it is best to write the data to the resource one block at a time instead of making a single large contiguous buffer. If the underlying resource only consumes data in a single contiguous buffer, it is best to have a configurable maximum size and truncate data that exceeds it.

The OnInput function tells the publisher to read data from the underlying resource. The data is then stored internally in a seek-able stream and used by Peach for cracking the data model. If the underlying resource already provides a seek-able stream (eg: a file stream) then OnInput doesn't have to do anything. However, for publishers like Udp, the OnInput function causes the next packet to be received. When implementing the OnInput function for publishers that block, it is best to expose a user configurable input timeout. If no data is received from the underlying resource after the timeout interval, a SoftException() should be thrown to indicate to the Peach engine that the action did not complete successfully.

To assist developers in quickly writing publishers, Peach comes with two helper classes: StreamPublisher and BufferedStreamPublisher. Both of these classes make it easy to quickly write publishers for IO interfaces that already implement the C# System.IO.Stream interface.

- StreamPublisher is used when the underlying stream supports the Seek() function (eg: file streams). To use the StreamPublisher, the developer only needs to override OnOpen() and set the *this.stream* property.
- BufferedStreamPublisher is used when the underlying stream does not support the Seek() function (eg: TCP Streams, SerialPort). The BufferedStreamPublisher automatically performs asynchronous reads on the underlying stream, and buffers the accumulated data in a seek-able stream.

To use the BufferedStreamPublisher, the developer needs to do two things.

- a. Override OnOpen() and set *this._client* to the System.IO.Stream and call the StartClient() function.
- b. If extra cleanup code is required other than simply closing the stream, the developer needs to override ClientClose() and clean up any additional resources acquired during OnOpen().

An example of a File publisher deriving from StreamPublisher.

```

[Publisher("File", true)]
[Publisher("FileStream")]
[Publisher("file.FileWriter")]
[Publisher("file.FileReader")]
[Parameter("FileName", typeof(string), "Name of file to open for reading/writing")]
[Parameter("Overwrite", typeof(bool), "Replace existing file? [true/false, default true]", "true")]
[Parameter("Append", typeof(bool), "Append to end of file [true/false, default false]", "false")]
public class FilePublisher : StreamPublisher
{
    private static NLog.Logger logger = LogManager.GetCurrentClassLogger();
    protected override NLog.Logger Logger { get { return logger; } }

    public string FileName { get; set; }
    public bool Overwrite { get; set; }
    public bool Append { get; set; }

    private FileMode fileMode = FileMode.OpenOrCreate;

    public FilePublisher(Dictionary<string, Variant> args)
        : base(args)
    {
        if (Overwrite && Append)
            throw new PeachException("File publisher does not support Overwrite and Append being enabled at once.");
        else if (Overwrite)
            fileMode = FileMode.Create;
        else if (Append)
            fileMode = FileMode.Append | FileMode.OpenOrCreate;
        else
            fileMode = FileMode.OpenOrCreate;
    }

    protected override void OnOpen()
    {
        stream = System.IO.File.Open(FileName, FileMode);
    }
}

```

An example of a Serial publisher deriving from BufferedStreamPublisher.

```

[Publisher("SerialPort", true)]
[Parameter("PortName", typeof(string), "Com interface for the device to connect to")]
[Parameter("BaudRate", typeof(int), "The serial baud rate.")]
[Parameter("Parity", typeof(Parity), "The parity-checking protocol.")]

```

```

[Parameter("DataBits", typeof(int), "Standard length of data bits per byte.")]
[Parameter("StopBits", typeof(StopBits), "The standard number of stop bits per byte.")]
public class SerialPortPublisher : BufferedStreamPublisher
{
    private static NLog.Logger logger = LogManager.GetCurrentClassLogger();
    protected override NLog.Logger Logger { get { return logger; } }

    public string PortName { get; protected set; }
    public int Baudrate { get; protected set; }
    public Parity Parity { get; protected set; }
    public int DataBits { get; protected set; }
    public StopBits StopBits { get; protected set; }

    protected SerialPort _serial;

    public SerialPortPublisher(Dictionary<string, Variant> args)
        : base(args)
    {
    }

    protected override void OnOpen()
    {
        base.OnOpen();

        try
        {
            _serial = new SerialPort(PortName, Baudrate, Parity, DataBits, StopBits);
            _serial.Handshake = Handshake;
            _serial.DtrEnable = DtrEnable;
            _serial.RtsEnable = RtsEnable;
            _serial.Open();
            // Set _clientName so logs from the base class are pretty
            _clientName = _serial.PortName;
            // Set _client to use for async IO
            _client = _serial.BaseStream;
        }
        catch (Exception ex)
        {
            string msg = "Unable to open Serial Port {0}. {1}.".Fmt(PortName, ex.Message);
            Logger.Error(msg);
            throw new PeachException(msg, ex);
        }

        // Start the async read operations
        StartClient();
    }
}

```

```

protected override void ClientClose()
{
    base.ClientClose();

    // No custom closing required
    _serial = null;
}

```

Some Publishers use the Function view metaphor (call, setProperty, getProperty). For this type of Publisher, sharing information occurs by implementing properties of the publisher that other parts of the Peach can access. For example, the StateModel or a script could make use of the ports that a Publisher uses, as in the following example.



Remote agents can host Publishers; therefore, Publishers should not directly use the IterationStateStore bag.

An example of a Serial publisher deriving from BufferedStreamPublisher.

```

protected override Variant OnGetProperty(string property)
{
    switch(property)
    {
        case "Port":
            return new Variant(Port);
        case "SrcPort":
            return new Variant(SrcPort);
    }
    return base.OnGetProperty(property);
}

```

The property is then accessible in the state model using the getProperty action, or using a script as in the following:

```
Port = int(context.test.publishers[0].getProperty('Port'))
```

22.5. Transformer

Peach includes support for common encoding transformations such as encryption and compression. Custom transformers are used whenever the target requires special encoding of the data not included with Peach. Custom transformer implementations must implement two functions: internalEncode and internalDecode.

Peach calls the internalEncode function when outputting data to a publisher. Inside this function, the provided source data of the parent data element is transformed into a new stream and the new stream is returned to Peach. For example, the AES transformer's implementation of this function encrypts the source data and returns the encrypted data.

Peach calls the internalDecode function when cracking input data into the transformer's parent data element. This function call results from an input action or from a DataSet being applied. The internalDecode function is given a stream of encoded data. The function performs the necessary decoding logic and returns a new stream of decoded data. The decoded data is then used by the transformer's parent element for cracking. For example, the AES transformer's implementation of this function decrypts the source data and returns the decrypted data.

22.6. Mutator

Peach includes mutators for common data types and patterns. Occasionally it is necessary to extend the mutation capabilities with a custom mutator. Custom mutators follow a similar pattern to all other Peach plug-ins, implement a class that derives from the base class, Peach.Core.Mutator class, and decorate the class with a MutatorAttribute.

In order for Peach to select a mutator when fuzzing, every mutator must implement a static supportedDataElement function. This function gets called with every data element in the DOM and returns true if the mutator can mutate the data element.

An example of the supportedDataElement function for a string mutator.

```
public new static bool supportedDataElement(DataElement obj)
{
    if (obj is Dom.String && obj.isMutable)
        return true;

    return false;
}
```

When the Peach engine determines that a mutator is capable of mutating a data element, an instance of the mutator class is created for each supported data element. Once Peach creates all of the mutator instances, each mutator needs to provide the number of mutations that can be performed. The supported mutation count is provided with the *count* property.

Each mutator implements two functions for performing the actual mutation: sequentialMutation and randomMutation. These functions correspond to the type of mutation strategy defined in the Peach pit. When the Peach configuration uses the Random strategy, the randomMutation function performs mutations. When the Peach configuration uses the Sequential strategy, the mutator's *mutation* attribute will be set to the desired value (between 0 and count) and the sequentialMutation function performs mutations.

When picking random numbers inside the mutator, it is important to use the random number generator provided by the mutation strategy. Mutators access the random number generator with the `this.context.Random` variable. The mutation strategy guarantees that the random number generator is different across different fuzzing iterations. Additionally, the mutation strategy guarantees the random number generator is identical for the same iterations. This allows Peach to produce identical mutations when replaying the same fuzz iteration to reproduce faults or when re-running a test of the same seed at a future date.

An example of a string mutator that mutates string elements with the values "Hello", "World" or "Hello World".

```

[Mutator("StringMutator")]
[Description("Replace strings with hello world")]
public class HelloWorldMutator : Peach.Core.Mutator
{
    uint pos = 0;
    static string[] values = new string[] { "Hello", "World", "Hello World" };

    public StringMutator(DataElement obj)
    {
        pos = 0;
        name = "HelloWorldMutator";
    }

    public new static bool supportedDataElement(DataElement obj)
    {
        if (obj is Dom.String && obj.isMutable)
            return true;

        return false;
    }

    public override int count
    {
        get { return values.Length; }
    }

    public override uint mutation
    {
        get { return pos; }
        set { pos = value; }
    }

    public override void sequentialMutation(DataElement obj)
    {
        obj.mutationFlags = MutateOverride.Default;
        obj.MutatedValue = new Variant(values[pos]);
    }

    public override void randomMutation(DataElement obj)
    {
        obj.mutationFlags = MutateOverride.Default;
        obj.MutatedValue = new Variant(this.context.Random.Choice<string>(values));
    }
}

```

In order for Peach to select a mutator when fuzzing, every mutator must implement a static

`supportedDataElement` function. This function is called with every data element in the DOM and returns true if the mutator can mutate the data element.

An example of the `supportedDataElement` function for a string mutator.

```
public new static bool supportedDataElement(DataElement obj)
{
    if (obj is Dom.String && obj.isMutable)
        return true;

    return false;
}
```

22.7. Agent

Custom agents are useful when the target system does not support a .NET runtime (like Mono) or the device is too slow (speed is a common problem for embedded devices that require running a native agent in C/C++).

Agents in Peach communicate over protocols called channels. While you can develop custom channel protocols, an existing channel usually creates a custom Peach agent. Agents can be written in any language; To make it easy to author your own, Peach comes with some example implementations designed for languages like as Python and C++. The REST based protocol (which transmits data in JSON messages) is the easiest channel protocol to use with custom agents.

When you write a custom publisher, we recommend you start with one of the examples in the SDK and extend it to meet your requirements. The examples in the SDK already implement the agent channel protocol with stub methods ready to be implemented. If this is not an option, the following example channel sessions can be used as documentation for the protocol. The second example includes the use of a remote publisher.

Example 320. Sample session

This example shows a complete agent session using the REST JSON agent channel (protocol prefix `http`) with matching pit.

```

<Agent name="TheAgent" location="http://127.0.0.1:9980">
    <Monitor class="WindowsDebugger">
        <Param name="Executable" value="mspaint.exe" />
        <Param name="Arguments" value="fuzzed.png" />
        <Param name="WinDbgPath" value="C:\Program Files (x86)\Debugging Tools for Windows (x86)" />
        <Param name="StartOnCall" value="ScoobySnacks"/>
    </Monitor>
    <Monitor class="PageHeap">
        <Param name="Executable" value="mspaint.exe"/>
        <Param name="WinDbgPath" value="C:\Program Files (x86)\Debugging Tools for Windows (x86)" />
    </Monitor>
</Agent>

<Test name="Default">
    <Agent ref="TheAgent"/>
    <StateModel ref="TheState"/>

    <Publisher class="File">
        <Param name="FileName" value="fuzzed.png"/>
    </Publisher>

</Test>

```

```

GET /Agent/AgentConnect
<< { "Status": "true" }

POST /Agent/StartMonitor?name=Monitor_0&cls=WindowsDebugger
>> {"args": {"Executable": "mspaint.exe", "Arguments": "fuzzed.png", "WinDbgPath": "C:\\\\Program Files (x86)\\\\Debugging Tools for Windows (x86)", "StartOnCall": "ScoobySnacks"}}
<< { "Status": "true" }

POST /Agent/StartMonitor?name=Monitor_1&cls=PageHeap
>> {"args": {"Executable": "mspaint.exe", "WinDbgPath": "C:\\\\Program Files (x86)\\\\Debugging Tools for Windows (x86)"}}
<< { "Status": "true" }

GET /Agent/SessionStarting
<< { "Status": "true" }

GET /Agent/IterationStarting?iterationCount=1&isReproduction=False
<< { "Status": "true" }

GET /Agent/IterationFinished

```

```

<< { "Status":"true" }

GET /Agent/DetectedFault
<< { "Status":"true" }
// Status of true indicates a fault was detected. False for no fault.

GET /Agent/GetMonitorData
<< {
    "Results": [
        {
            "iteration": 0,
            "controlIteration": false,
            "controlRecordingIteration": false,
            "type": 0, (0 unknown, 1 Fault, 2 Data)
            "detectionSource": null,
            "title": null,
            "description": null,
            "majorHash": null,
            "minorHash": null,
            "exploitability": null,
            "folderName": null,
            "collectedData": [
                {"Key": "data1", "Value": "AA=="}
            ]
        }
    ]
}

GET /Agent/IterationStarting?iterationCount=1&isReproduction=True
<< { "Status":"true" }

GET /Agent/IterationFinished
<< { "Status":"true" }

GET /Agent/DetectedFault
<< { "Status":"true" }
// Status of true indicates a fault was detected. False for no fault.

GET /Agent/GetMonitorData
<< {
    "Results": [
        {
            "iteration": 0,
            "controlIteration": false,
            "controlRecordingIteration": false,
            "type": 0, (0 unknown, 1 Fault, 2 Data)
            "detectionSource": null,
            "title": null,
            "description": null,
            "majorHash": null,
            "minorHash": null,
            "exploitability": null,
            "folderName": null,
            "collectedData": [
                {"Key": "data1", "Value": "AA=="}
            ]
        }
    ]
}

```

```

        "description":null,
        "majorHash":null,
        "minorHash":null,
        "exploitability":null,
        "folderName":null,
        "collectedData": [
            {"Key": "data1", "Value": "AA=="}
        ]
    }
]

}

GET /Agent/Publisher/stop
<< { "Status": "true" }

GET /Agent/SessionFinished
<< { "Status": "true" }

GET /Agent/StopAllMonitors
<< { "Status": "true" }

GET /Agent/AgentDisconnect
<< { "Status": "true" }

```

Example 321. Sample session with remote publisher

This example shows the channel messages when a remote publisher is in use.

```

<Agent name="TheAgent" location="http://127.0.0.1:9980">
    <Monitor class="WindowsDebugger">
        <Param name="Executable" value="mspaint.exe" />
        <Param name="Arguments" value="fuzzed.png" />
        <Param name="WinDbgPath" value="C:\Program Files (x86)\Debugging Tools for Windows (x86)" />
        <Param name="StartOnCall" value="ScoobySnacks"/>
    </Monitor>
    <Monitor class="PageHeap">
        <Param name="Executable" value="mspaint.exe"/>
        <Param name="WinDbgPath" value="C:\Program Files (x86)\Debugging Tools for Windows (x86)" />
    </Monitor>
</Agent>

<Test name="Default">
    <Agent ref="TheAgent"/>
    <StateModel ref="TheState"/>

    <Publisher class="Remote">
        <Param name="Agent" value="TheAgent"/>
        <Param name="Class" value="File"/>
        <Param name="FileName" value="fuzzed.png"/>
    </Publisher>

</Test>

```

```

GET /Agent/AgentConnect
<< { "Status":"true" }

POST /Agent/StartMonitor?name=Monitor_0&cls=WindowsDebugger
>> {"args": {"Executable": "mspaint.exe", "Arguments": "fuzzed.png", "WinDbgPath": "C:\\\\Program Files (x86)\\\\Debugging Tools for Windows (x86)", "StartOnCall": "ScoobySnacks"}}
<< { "Status":"true" }

POST /Agent/StartMonitor?name=Monitor_1&cls=PageHeap
>> {"args": {"Executable": "mspaint.exe", "WinDbgPath": "C:\\\\Program Files (x86)\\\\Debugging Tools for Windows (x86)"}}
<< { "Status":"true" }

GET /Agent/SessionStarting
<< { "Status":"true" }

GET /Agent/IterationStarting?iterationCount=1&isReproduction=False
<< { "Status":"true" }

```

```

POST /Agent/Publisher/Set_Iteration
>> {"iteration":1}
<< { "error":"false", "errorString":null }

POST /Agent/Publisher/Set_IsControlIteration
>> {"isControlIteration":true}
<< { "error":"false", "errorString":null }

POST /Agent/Publisher/Set_IsControlIteration
>> {"isControlIteration":true}
<< { "error":"false", "errorString":null }

POST /Agent/Publisher/Set_Iteration
>> {"iteration":1}
<< { "error":"false", "errorString":null }

GET /Agent/Publisher/start
<< { "error":"false", "errorString":null }

GET /Agent/Publisher/open
<< { "error":"false", "errorString":null }

POST /Agent/Publisher/output
>> {"data":"SGVsbG8gV29ybGQ="}
<< { "error":"false", "errorString":null }

GET /Agent/Publisher/close
<< { "error":"false", "errorString":null }

POST /Agent/Publisher/call
>> {"method":"ScoobySnacks", "args": [{"name":"p1", "data":"SGVsbG8gV29ybGQ=", "type":0}]
]}
<< { "error":"false", "errorString":null }

GET /Agent/IterationFinished
<< { "Status":"true" }

GET /Agent/DetectedFault
<< { "Status":"true" }
// Status of true indicates a fault was detected. False for no fault.

GET /Agent/GetMonitorData
<< {
  "Results": [
    {
      "iteration":0,
      "controlIteration":false,

```

```

    "controlRecordingIteration":false,
    "type":0, (0 unknown, 1 Fault, 2 Data)
    "detectionSource":null,
    "title":null,
    "description":null,
    "majorHash":null,
    "minorHash":null,
    "exploitability":null,
    "folderName":null,
    "collectedData":[
        {"Key":"data1","Value":"AA=="}
    ]
}
]
}

```

GET /Agent/IterationStarting?iterationCount=1&isReproduction=True
 << { "Status":"true" }

POST /Agent/Publisher/Set_Iteration
 >> {"iteration":1}
 << { "error":"false", "errorString":null }

POST /Agent/Publisher/Set_IsControlIteration
 >> {"isControlIteration":true}
 << { "error":"false", "errorString":null }

POST /Agent/Publisher/Set_IsControlIteration
 >> {"isControlIteration":true}
 << { "error":"false", "errorString":null }

POST /Agent/Publisher/Set_Iteration
 >> {"iteration":1}
 << { "error":"false", "errorString":null }

GET /Agent/Publisher/start
 << { "error":"false", "errorString":null }

GET /Agent/Publisher/open
 << { "error":"false", "errorString":null }

POST /Agent/Publisher/output
 >> {"data":"SGVsbG8gV29ybGQ="}
 << { "error":"false", "errorString":null }

GET /Agent/Publisher/close
 << { "error":"false", "errorString":null }

```

POST /Agent/Publisher/call
>> {"method":"ScoobySnacks", "args": [{"name":"p1", "data":"SGVsbG8gV29ybGQ=", "type":0}]
]}
<< { "error":"false", "errorString":null }

GET /Agent/IterationFinished
<< { "Status":"true" }

GET /Agent/DetectedFault
<< { "Status":"true" }
// Status of true indicates a fault was detected. False for no fault.

GET /Agent/GetMonitorData
<< {
    "Results": [
        {
            "iteration":0,
            "controlIteration":false,
            "controlRecordingIteration":false,
            "type":0, (0 unknown, 1 Fault, 2 Data)
            "detectionSource":null,
            "title":null,
            "description":null,
            "majorHash":null,
            "minorHash":null,
            "exploitability":null,
            "folderName":null,
            "collectedData": [
                {"Key": "data1", "Value": "AA=="}
            ]
        }
    ]
}

GET /Agent/Publisher/stop
<< { "Status":"true" }

GET /Agent/SessionFinished
<< { "Status":"true" }

GET /Agent/StopAllMonitors
<< { "Status":"true" }

GET /Agent/AgentDisconnect
<< { "Status":"true" }

```

23. Tutorials

The following tutorials provide information on how to get started fuzzing with Peach.

- Beginner: [Dumb File Fuzzing \(command line\)](#)
- Beginner: [File Fuzzing](#)

23.1. Tutorial: Dumb Fuzzing

Welcome to the dumb fuzzing tutorial. In this tutorial, we are going to build a simple dumb fuzzer for PNG graphics files (.png). Our dumb fuzzer will use several sample files (also known as seed files) to mutate using methods like bit flipping, dword slides, etc.

The target of this fuzzer is `mspaint` on Windows, `feh` on Linux, and `Safari` on OSX.

Tutorial Outline

1. [Development environment](#)
2. [Creating the data model](#)
3. [Creating the state model](#)
4. [Configuring a publisher](#)
5. [Adding an agent and monitor](#)

23.1.1. The Peach Development Environment

Before we start building our Peach fuzzer let's first take a moment to talk about a normal Peach development environment. Here is what a typical environment looks like:

XML Editor

A good XML editor is definitely a must. A good free XML editor for Windows is [Microsoft Visual Studio Express](#) edition. If you already have an XML editor that supports intelliSense via XML Schema, you're all set.

Latest Peach

Always stay up to date on the version of Peach to ensure you have the latest bug fixes.

Debugger and support tools

There are several support tools that make things work a lot nicer:

- On Windows, download the following for this lab:
 - [Microsoft Debugging Tools](#) (pick the latest version available)
- On OS X, download the following for this lab:
 - [CrashWrangler](#) (Apple Developer Connection account required)

23.1.2. Creating Data Models

Now we are going to dive right in. Let's start by making a copy of `template.xml` (found in your Peach folder) to `png.xml`. This will hold all of the information about our PNG dumb fuzzer. Also, You want several samples PNG files; start with 10.

Go ahead and load up `png.xml` into your XML editor.

For our dumb fuzzer, we only need one data model to hold all the data from the PNG file. The data model does not know anything about the PNG data structure, instead keeping all the data in a "Blob" element (binary large object or byte array).

Creating the DataModel

Okay, head over to your `png.xml` file and let's start writing some XML! Locate the `DataModel` called `TheDataModel` is should look something like this:

```
<!-- TODO: Create data model -->
<DataModel name="TheDataModel">
</DataModel>
```

Read more about: [DataModel](#)

We are going to add a single data element to our data model as follows:

```
<DataModel name="TheDataModel">
  <Blob />
</DataModel>
```

Read more about: [DataModel](#), [Blob](#)

Okay, that's all we need for our data model. The "Blob" element will end up holding all of our PNG data.

23.1.3. Create State Model

Now that you have created the data model we can create the state model. For file fuzzing, the state model is very simple. All we want to do is write out the file and launch the target process. We can do this using three actions:

- `output` — Write the file
- `close` — Close the file
- `call` — Launch the application

Go ahead and locate the state model in the `png.xml` file called `TheState`. We will expand on this state model to include our three actions as follows:

```

<!-- This is our simple png state model -->
<StateModel name="TheState" initialState="Initial">
  <State name="Initial">

    <!-- Write out our png file -->
    <Action type="output">
      <DataModel ref="TheDataModel"/>

      <!-- This is our folder of sample files to read in -->
      <Data name="data" fileName="samples_png/*.png"/>
    </Action>

    <Action type="close"/>

    <!-- Launch the target process -->
    <Action type="call" method="LaunchViewer" publisher="Peach.Agent"/>
  </State>
</StateModel>

```

Read more about: [StateModel](#), [State](#), [Action](#), [DataModel](#), [Data](#)

In the final "call" action, notice that we have configured an attribute called "publisher" with the value "Peach.Agent". This attribute causes this action to send a "call" message to any configured Agent (see the next page) with the message "LaunchViewer". This is how the debugger monitor knows to launch the process.

We are all set! Next we just need to configure our debugger and publishers.

23.1.4. Configure Publisher

The last thing we need to do before we can try out our nifty fuzzer is to configure a [Publisher](#). [Publishers](#) are I_O connectors that implement the plumbing between actions like *output*, *input*, and *call*. For our file fuzzer, we will use the [Publisher](#) called [File](#). This publisher allows us to write out a file.

Configuring our publisher is easy, just locate the following XML near the bottom of the [png.xml](#) file; it is a child of [Test](#).

```

<!-- TODO: Complete publisher -->
<Publisher />

```

Now, this publisher takes a single parameter called *FileName* that contains the file name of the fuzzed file.

```
<Publisher class="File">
    <Param name="FileName" value="fuzzed.png"/>
</Publisher>
```

What's Next?

We will need to configure a way to detect when our target crashes. We will also want to collect some information like a stack trace to look at later on. Head to the next section to learn how to configure an agent and monitor.

23.1.5. Agent and Monitor

Now we are ready to configure our agent and monitors. Agents are special Peach processes that can be run locally in process or remote over a network connection. These agents host one or more monitors that can perform such actions as attaching debuggers, watching memory consumption, etc. For this tutorial we are going to configure Peach to use monitors specific to each target platform. Windows will be configured to use Microsoft WinDbg to monitor `mspaint.exe` for exceptions and other common issues. Additionally on Windows we will enable the HEAP debugging for the target process. Linux will be configured to monitor for the presence of core files. OSX will be configured to use CrashWrangler to monitor `Safari` for exceptions and other common issues.

Configure the Agent and Monitor

First, let's locate the commented out `Agent` element in the template file, it looks something like this:

```
<!-- TODO: Configure agent -->
<Agent name="TheAgent" location="http://127.0.0.1:9000"/>
```

We are going to uncomment this section and remove the "location" attribute. When no "location" attribute is present, Peach automatically starts a local Peach Agent. We will configure three agents, one for Windows, one for Linux and one for OSX. The Windows agent will be comprised of two monitors: WindowsDebugger and PageHeap. The Linux agent will also be comprised of one monitor: Gdb. The OSX agent will only be comprised of a single monitor: CrashWrangler.

```
<Agent name="WinAgent">
    <Monitor class="WindowsDebugger">

        <!-- The command line to run. Notice the filename provided matched up
            to what is provided below in the Publisher configuration -->
        <Param name="Executable" value="mspaint.exe" />
        <Param name="Arguments" value="fuzzed.png" />

        <!-- This parameter will cause the debugger to wait for an action-call in
            the state model with a method="LaunchViewer" before running
```

```

        program.

-->
<Param name="StartOnCall" value="LaunchViewer" />

<!-- This parameter will cause the monitor to terminate the process
     once the CPU usage reaches zero.
-->
<Param name="CpuKill" value="true"/>

</Monitor>

<!-- Enable heap debugging on our process as well. -->
<Monitor class="PageHeap">
    <Param name="Executable" value="mspaint.exe"/>
</Monitor>

</Agent>

<Agent name="LinAgent">
    <!-- Register for core file notifications. -->
    <Monitor class="Gdb"/>

    <!-- This is the program we're going to run inside of the debugger -->
    <Param name="Executable" value="feh"/>

    <!-- These are arguments to the executable we want to run -->
    <Param name="Arguments" value="fuzzed.png"/>

    <!-- This parameter will cause the monitor to terminate the process
         once the CPU usage reaches zero.
    -->
    <Param name="CpuKill" value="true"/>

</Agent>

<Agent name="OsxAgent">
    <Monitor class="CrashWrangler">
        <!-- The executable to run. -->
        <Param name="Command" value="/Applications/Safari.app/Contents/MacOS/Safari" />

        <!-- The program arguments. Notice the filename provided matched up
             to what is provided below in the Publisher configuration -->
        <Param name="Arguments" value="fuzzed.png" />

        <!-- Do not use debug malloc. -->
        <Param name="UseDebugMalloc" value="false" />

        <!-- Treat read access violations as exploitable. -->

```

```

<Param name="ExploitableReads" value="true" />

<!-- Path to Crash Wrangler Execution Handler program. -->
<Param name="ExecHandler" value="/usr/local/bin/exc_handler" />

<!-- This parameter will cause the monitor to wait for an action-call in
     the state model with a method="LaunchViewer" before running
     program.
-->
<Param name="StartOnCall" value="LaunchViewer" />

</Monitor>
</Agent>

```

Configure Test

Okay, now we just need to enable the agent for our test. Head down to the **Test** element, specifically we are looking to uncomment this line, and modify our Launcher publisher.

```
<!-- <Agent ref="LocalAgent"/> -->
```

Leaving us with this:

```

<Test name="Default">
    <Agent ref="WinAgent" platform="windows"/>
    <Agent ref="LinAgent" platform="linux"/>
    <Agent ref="OsxAgent" platform="osx"/>

    <StateModel ref="TheState"/>

    <Publisher class="File">
        <Param name="FileName" value="fuzzed.png"/>
    </Publisher>
</Test>

```

Configure Fuzzing Strategy

Since we are dumb fuzzing with multiple files, we want to change the default fuzzing strategy Peach uses to one more suited to our needs. The best fuzzing strategy for dumb fuzzing is the random strategy. We can configure it by adding a **Strategy** element to our **Test** section.

We add this:

```
<Strategy class="Random"/>
```

Leaving us with this:

```
<Test name="Default">
  <Agent ref="WinAgent" platform="windows"/>
  <Agent ref="LinAgent" platform="linux"/>
  <Agent ref="OsxAgent" platform="osx"/>

  <StateModel ref="TheState"/>

  <Publisher class="File">
    <Param name="FileName" value="fuzzed.png"/>
  </Publisher>

  <Strategy class="Random"/>
</Test>
```

23.1.6. Running the Fuzzer

Now, let's actually kick off our fuzzer for real! Every 200 or so iterations, the strategy will switch to a different sample file.

```
peach png.xml
```

What's Next?

From here you will want to:

1. Collect additional samples files
2. Run minset on the sample files to remove any files that cause duplicate code paths
3. Collect bugs!

23.2. Tutorial: File Fuzzing

Welcome to the file fuzzing tutorial. In this tutorial, we are going to build a wave (.wav) file fuzzer. Wave files are based on the RIFF file format. This format is not overly complex and will show off several features of Peach. The target of this fuzzer is [mplayer](#), an open-source, cross-platform, command line media player.

Tutorial Outline

1. [Development environment](#)
2. [Creating the data model](#)
3. [Creating the state model](#)
4. [Configuring a publisher](#)
5. [Adding an agent and monitor](#)
6. [Optimizing test count](#)

23.2.1. The Peach Development Environment

Before we start building our Peach fuzzer let's first take a moment to talk about a normal Peach development environment. Here is what a typical environment looks like:

XML Editor

A good XML editor is definitely a must. A good free XML editor for Windows is [Microsoft Visual Studio Express](#) edition. If you already have an XML editor that supports intelliSense via XML Schema, you're all set.

Latest Peach

Always stay up to date on the version of Peach you are using to get the latest bug fixes.

Debugger and support tools

There are several support tools that make things work a lot nicer.

- On Windows, download the Microsoft Debugging Tools for Windows.
- On OS X, download CrashWrangler.

Both of these tools are covered in the installation chapter.

23.2.2. Creating Wave Data Models

Now we are going to dive right in. Let's start by making a copy of `template.xml` (found in your Peach folder) to `wav.xml`. This will hold all of the information about our WAV fuzzer. You will also want a sample WAV file, grab [this one](#).

Go ahead and load up `wav.xml` into your XML editor.

Now, you want to check out the following two specifications to get an idea for the format of WAV:

1. [Wave File Format](#)
2. RIFF File Specification (Microsoft)

If you glance through the wave file format, notice that the wave file is composed of a file header followed by a number of chunks. This is fairly common for file formats and also for network packets. Typically, each chunk has the same format that follows some form of T-L-V or Type-Length-Value. In fact, wave file chunks are just that, a type followed by length followed by data. Each chunk type will define what its data looks like.

Based on this basic information we can plan out our fuzzer. We have several top level "[DataModel](#)" elements that will be called:

- Chunk
- ChunkFmt
- ChunkData
- ChunkFact
- ChunkCue
- ChunkPlst
- ChunkList
- ChunkLabl
- ChunkLtxt
- ChunkNote
- ChunkSmpl
- ChunkInst
- Wav

The [DataModel](#) called *Chunk* will be a template for each of the following types of chunks and we will pull all of it together, and also define our header in the last [DataModel](#) called *Wav*.

Setting Defaults for Number element

The majority of numbers used in WAV are unsigned. We can make that the default by adding this XML to our PIT:

```
<Defaults>
  <Number signed="false" />
</Defaults>
```

Creating the Wav DataModel

Okay, head over to your `wav.xml` file and let's start writing some XML! Locate the `DataModel` called `TheDataModel` is should look something like this:

```
<!-- TODO: Create data model -->
<DataModel name="TheDataModel">
</DataModel>
```

Read more about: [DataModel](#)

We are going to rename this `Wav` and define the header for our wave file. Looking at the specification I notice that the format for the wave header is:

- File magic: 4-character string, always "RIFF"
- Length of file: 32-bit unsigned integer
- Riff type: 4-character string, always "WAVE"

We can define that in Peach using the following XML:

```
<!-- Defines the format of a WAV file -->
<DataModel name="Wav">
  <!-- wave header -->
  <String value="RIFF" token="true" />
  <Number size="32" />
  <String value="WAVE" token="true"/>
</DataModel>
```

Read more about: [DataModel](#), [String](#), [Number](#)

Otherwise, we can continue and define the `Chunk` DataModel.

Chunk DataModel

The `Chunk` data model should be the first data model in the Peach Pit file, so let's add it in above the `Wav` data model as follows:

```

<!-- Defines the common wave chunk -->
<DataModel name="Chunk">
</DataModel>

<!-- Defines the format of a WAV file -->
<DataModel name="Wav">
    <!-- wave header -->
    <String value="RIFF" token="true" />
    <Number size="32" />
    <String value="WAVE" token="true"/>
</DataModel>

```

Read more about: [DataModel](#), [String](#), [Number](#)

Notice that the *Chunk* data model occurs before the *Wav* data model. This is important, we will reference this data model later and it must be defined before we use it.

Looking at the specification, we know that the wave chunk format is as follows:

- ID: 4-character string padded with spaces
- Size: 4-byte unsigned integer
- Data: bytes of data the size of Size

We can model that in Peach using the following XML:

```

<!-- Defines the common wave chunk -->
<DataModel name="Chunk">
    <String name="ID" length="4" padCharacter=" " />
    <Number name="Size" size="32" >
        <Relation type="size" of="Data" />
    </Number>
    <Blob name="Data" />
    <Padding alignment="16" />
</DataModel>

```

Read more about: [DataModel](#), [String](#), [Number](#), [Relation](#), [Blob](#), [Padding](#)

Notice that we have created a size [relationship](#) between *Size* and *Data*. By doing this, *Size* automatically updates with the size of *Data* when we produce data. When we parse in a sample file to use as default values, this instructs the parser that it can find the size of *Data* by looking at *Size*.

Now we can use a Padding type to pad out our DataModel correctly. Notice that the alignment attribute is set to 16. This tells the [Padding](#) element to automatically size itself so that the *Chunk* DataModel ends on a 16-bit (2-byte) boundary.

Format Chunk

Now we are going to define the details of the format chunk. We will use the generic chunk we already defined as a template for this chunk. That allows us to specify only the specifics of this chunk and save some typing.

Looking at the wave specification, we can tell that the format chunk is as follows:

- ID: Always 'fmt'
- Data:
 - Compression code: 16-bit unsigned int
 - Number of channels: 16-bit unsigned int
 - Sample rate: 32-bit unsigned int
 - Average bytes per second: 32-bit unsigned int
 - Block align: 16-bit unsigned int
 - Significant bits per sample: 16-bit unsigned int
 - Extra format bytes: 16-bit unsigned int

The *ChunkFmt* data model will be defined after *Chunk* but before *Wav*:

```
<DataModel name="ChunkFmt" ref="Chunk">
  <String name="ID" value="fmt" token="true"/>
  <Block name="Data">
    <Number name="CompressionCode" size="16" />
    <Number name="NumberOfChannels" size="16" />
    <Number name="SampleRate" size="32" />
    <Number name="AverageBytesPerSecond" size="32" />
    <Number name="BlockAlign" size="16" />
    <Number name="SignificantBitsPerSample" size="16" />
    <Number name="ExtraFormatBytes" size="16" />
    <Blob name="ExtraData" />
  </Block>
</DataModel>
```

Read more about: [DataModel](#), [Block](#), [String](#), [Number](#), [Blob](#)

Now, if you look at this, notice a number of cool things. First off, if you check out the *DataModel* element, you can see an attribute called *ref* which has a value of *Chunk*. This tells Peach to copy the *Chunk* data model and make it the basis for our new data model called *ChunkFmt*. This means that all the elements defined in *Chunk* are in our new *ChunkFmt* by default! This is way cool and our first look at re-use in Peach. Next, we have two elements in our data model that have the same name as elements in the *Chunk* model (ID and Data). This causes the old elements to be replaced with our new ones. This allows us to override the old elements based on the needs of our format chunk type.

Now, you might be asking why we needed to override ID. This is a good question. We override ID here to specify the static string that it will always be for this format chunk. Later, we will specify a sample wave file to use and the parser will need hints on how to select the correct chunk. More on that later when we introduce the [Choice](#) element :)

Otherwise, I think things should largely make sense.

Data Chunk

Next up is the data chunk. This one is easy because the *Data* portion of the packet has no structure. We can define this chunk as follows:

```
<DataModel name="ChunkData" ref="Chunk">
    <String name="ID" value="data" token="true"/>
</DataModel>
```

Read more about: [DataModel](#), [String](#)

Fact Chunk

Okay, now we have the fact chunk. This chunk is defined as follows:

- ID: "fact", 4-character string
- Data:
 - Number of samples: 32-bit unsigned int
 - Unknown? Unknown trailing bytes

Another easy one to define in XML:

```
<DataModel name="ChunkFact" ref="Chunk">
    <String name="ID" value="fact" token="true"/>
    <Block name="Data">
        <Number size="32" />
        <Blob/>
    </Block>
</DataModel>
```

Read more about: [DataModel](#), [Block](#), [String](#), [Number](#), [Blob](#)

Notice that I was lazy and decided not to name the [Number](#) or [Blob](#) here. Peach does not require that all elements have names, only those that are being referenced.

Wave List Chunk

This chunk is a bit different. The wave list chunk is comprised of a silent chunk and data chunks, alternating in a list. So, before we can complete the wave list chunk, we will need to define the silent chunk. Let's do that now.

The silent chunk is easy, it's just a 4-byte unsigned integer. The data model looks like this:

```
<DataModel name="ChunkSint" ref="Chunk">
    <String name="ID" value="sInt" token="true"/>
    <Block name="Data">
        <Number size="32" />
    </Block>
</DataModel>
```

Read more about: [DataModel](#), [Block](#), [String](#), [Number](#)

Now that that's out of the way, we can get on with our wave list chunk. The data portion is an array of silent and data chunks. Here is how we do that:

```
<DataModel name="ChunkWav1" ref="Chunk">
    <String name="ID" value="wav1" token="true"/>
    <Block name="Data">
        <Block name="ArrayOfChunks" maxOccurs="3000">
            <Block ref="ChunkSint"/>
            <Block ref="ChunkData" />
        </Block>
    </Block>
</DataModel>
```

Read more about: [DataModel](#), [Block](#), [String](#)

This definition introduces the concept of arrays, or repeating elements. Notice that we have a [Block](#) element that has an attribute *maxOccurs*. This tells Peach that this block occurs once or more, up to 3,000 times. Also, notice that we are using the *ref* attribute with the [Block](#) element. This is just like using it with the data model, but allows us to get re-use inside of the data model as well.

Cue Chunk

Now onto the cue chunk. This chunk should be easy now that we know about the *maxOccurs* attribute. This chunk is also an array. The array is comprised of the following:

- ID: 4 bytes
- Position: 4-byte unsigned integer
- Data Chunk ID: 4-byte RIFF ID

- Chunk start: 4-byte unsigned integer offset of data chunk
- Block start: 4-byte unsigned integer offset to sample of first channel
- Sample offset: 4-byte unsigned integer offset to sample byte of first channel

We don't have to worry about the fact the last 3 numbers are offsets. The data is already parsed in the wave list chunk, we just need to read them in. So let's build the XML!

```
<DataModel name="ChunkCue" ref="Chunk">
  <String name="ID" value="cue " token="true"/>
  <Block name="Data">
    <Block name="ArrayOfCues" maxOccurs="3000">
      <String length="4" />
      <Number size="32" />
      <String length="4" />
      <Number size="32" />
      <Number size="32" />
      <Number size="32" />
    </Block>
  </Block>
</DataModel>
```

Read more about: [DataModel](#), [Block](#), [String](#), [Number](#)

There shouldn't be any surprises here, we are just re-using the same stuff as before. Once again, I'm being a bit lazy and not giving everything a name. This is okay, but it can be nice sometimes to use the names as documentation :)

Playlist Chunk

Looking at this chunk, I notice that *Data* will be comprised of an array (again); but, this time the count will be included before the array. We use a count-of relationship to model this.

```

<DataModel name="ChunkPlst" ref="Chunk">
    <String name="ID" value="plst" token="true"/>
    <Block name="Data">
        <Number name="NumberOfSegments" size="32" >
            <Relation type="count" of="ArrayOfSegments"/>
        </Number>
        <Block name="ArrayOfSegments" maxOccurs="3000">
            <String length="4" />
            <Number size="32" />
            <Number size="32" />
        </Block>
    </Block>
</DataModel>

```

Read more about: [DataModel](#), [Block](#), [Number](#), [String](#), [Relation](#)

Notice in this XML, that we setup a relationship between *NumberOfSegments* and *ArrayOfSegments* that will indicate the count.

Associated Data List Chunk

This chunk is an array of label chunks, name chunks, and text chunks. We will not know in what order they will appear, so we will need to support them in any order. This is actually be fairly easy, but first we need to define each of the tree chunks before we define our data list chunk. Let's do that now.

Label Chunk

First up is the label chunk. Its data portion contains a null-terminated string, and possibly, a single pad byte.

```

<DataModel name="ChunkLabl" ref="Chunk">
    <String name="ID" value="lbl" token="true"/>
    <Block name="Data">
        <Number size="32" />
        <String nullTerminated="true" />
    </Block>
</DataModel>

```

Read more about: [DataModel](#), [Block](#), [Number](#), [String](#)

We will automatically get the pad byte from the *Chunk*.

Note Chunk

Now onto the note chunk. It turns out this chunk is exactly the same as the label chunk! So, we will just create an alias for it like this:

```
<DataModel name="ChunkNote" ref="ChunkLabl">
    <String name="ID" value="note" token="true"/>
</DataModel>
```

Read more about: [DataModel](#), [String](#)

Yup, that's it! Nice and easy :)

Labeled Text Chunk

This one is also very similar to the note and label chunks but has several more numbers included in it. I'll copy the data model for the label chunk and expand it like this:

```
<DataModel name="ChunkLtxt" ref="Chunk">
    <String name="ID" value="ltxt" token="true"/>
    <Block name="Data">
        <Number size="32" />
        <Number size="32" />
        <Number size="32" />
        <Number size="16" />
        <Number size="16" />
        <Number size="16" />
        <Number size="16" />
        <String nullTerminated="true" />
    </Block>
</DataModel>
```

Read more about: [DataModel](#), [Block](#), [Number](#), [String](#)

As we can see, it's very similar to the label chunk.

Back to Associated Data List Chunk

Okay, we are ready to combine all the chunks into an array. It will end up looking like this:

```

<DataModel name="ChunkList" ref="Chunk">
    <String name="ID" value="list" token="true"/>
    <Block name="Data">
        <String value="adtl" token="true" />
        <Choice maxOccurs="3000">
            <Block ref="ChunkLabl"/>
            <Block ref="ChunkNote"/>
            <Block ref="ChunkLtxt"/>
            <Block ref="Chunk"/>
        </Choice>
    </Block>
</DataModel>

```

Read more about: [DataModel](#), [Block](#), [Number](#), [String](#), [Choice](#)

Here we are introducing the [Choice](#) element. This element will try each of the [Blocks](#) we specify looking for the best match. Notice that *Chunk*, our catch-all, is at the end of the list. The specification indicates that other types of blocks could show up here.

Sampler Chunk

The sampler chunk is similar to what we have already seen. It contains several numbers and an array of some values. We will define it as follows:

Read more about: [DataModel](#), [Block](#), [Number](#), [String](#)

Again, that was straight forward. :)

Instrument Chunk

Whew! This is our last chunk to define and it's an easy one. It's comprised of just seven (7) 8-bit numbers. This will be super easy.

Read more about: [DataModel](#), [Block](#), [Number](#), [String](#)

Notice that the numbers in this case are signed. The values they can have include both negative to positive numbers.

23.2.3. Finishing the Wav Model

Time to wrap this modeling up! Let's head down to the *Wav* chunk. The last time we touched, it looked like this:

```
<!-- Defines the format of a WAV file -->
<DataModel name="Wav">
    <!-- wave header -->
    <String value="RIFF" token="true" />
    <Number size="32" />
    <String value="WAVE" token="true"/>
</DataModel>
```

Read more about: [DataModel](#), [Number](#), [String](#)

We are going to add an array of chunks; however, we don't know in what order all these chunks will occur. So, we will use our friend, the [Choice](#) element, to have Peach decide this, using the input file.

```
<!-- Defines the format of a WAV file -->
<DataModel name="Wav">
    <!-- wave header -->
    <String value="RIFF" token="true" />
    <Number size="32" />
    <String value="WAVE" token="true"/>

    <Choice maxOccurs="3000">
        <Block ref="ChunkFmt"/>
        <Block ref="ChunkData"/>
        <Block ref="ChunkFact"/>
        <Block ref="ChunkSint"/>
        <Block ref="ChunkWavl"/>
        <Block ref="ChunkCue"/>
        <Block ref="ChunkPlst"/>
        <Block ref="ChunkLtxt"/>
        <Block ref="ChunkSmpl"/>
        <Block ref="ChunkInst"/>
        <Block ref="Chunk"/>
    </Choice>
</DataModel>
```

Read more about: [DataModel](#), [Block](#), [Number](#), [String](#), [Choice](#)

That wasn't so hard! Was it?

Next Steps

All the hard work is over, but there is still stuff we need to do before we can run our fuzzer!

23.2.4. Create State Model

Now that you have created the data models we can create the state model. For file fuzzing the state model is very simple. All we want to do is write out the file and launch the target process. We can do this using three actions:

- output — Write the file
- close — Close the file
- call — Launch the application

Go ahead and locate the state model in the `wav.xml` file called *TheState*. We will expand on this state model to include our three actions as follows:

```
<!-- This is our simple wave state model -->
<StateModel name="TheState" initialState="Initial">
    <State name="Initial">

        <!-- Write out our wave file -->
        <Action type="output">
            <DataModel ref="Wav"/>
            <!-- This is our sample file to read in -->
            <Data fileName="sample.wav"/>
        </Action>

        <Action type="close"/>

        <!-- Launch the target process -->
        <Action type="call" method="StartMPlayer" publisher="Peach.Agent" />
    </State>
</StateModel>
```

Read more about: [StateModel](#), [State](#), [Action](#), [DataModel](#), [Data](#), [Field](#)

Now on to configuring our Publisher!

23.2.5. Configure Publisher

The last thing we need to do before we can try out our nifty fuzzer is to configure a [Publisher](#).

[Publishers](#) are I_O connectors that implement the plumbing between actions like *output*, *input*, and *call*. For our file fuzzer, we will use the [Publisher](#) called [File](#). This publisher allows us to write out a file and then launch a process using the *call* action like we setup in last section.

Configuring our publisher is easy, just locate the following XML near the bottom of the [wav.xml](#) file, it is a child of [Test](#).

```
<!-- TODO: Complete publisher -->
<Publisher />
```

Now, this publisher takes a single parameter called *FileName* that contains the file name of the fuzzed file. This should be the same as the file name we specified in the *call* action ([fuzzed.wav](#)).

```
<Publisher class="File">
  <Param name="FileName" value="fuzzed.wav"/>
</Publisher>
```

What's Next?

Now we need a way to detect when our target crashes and also run our target. We will also want to collect some information like a stack trace to look at later on. Head to the next section to learn how to configure an agent and monitor.

23.2.6. Agent and Monitor

Now we are ready to configure our agent and monitors. Agents are special Peach processes that can run locally or remote. These processes host one or more monitors that can perform such actions as attaching debuggers, watching memory consumption, etc. For this tutorial, we are going to configure Microsoft WinDbg to monitor [mplayer.exe](#) for exceptions and other common issues. Additionally, we will enable the HEAP debugging for the target process.

Configure the Agent and Monitor

First lets locate the commented out [<Agent>](#) element in the template file, it looks something like this:

```
<!-- TODO: Configure agent -->
<Agent name="TheAgent" location="http://127.0.0.1:9000"/>
```

We are going to uncomment this section and remove the "location" attribute. When no "location" attribute is present, Peach automatically starts a local Peach Agent. We will configure three agents, one for Windows, one for Linux and one for OSX. The Windows agent will be comprised of two monitors: WindowsDebugger and PageHeap. The Linux agent will also be comprised of one monitor: Gdb. The OSX agent will only be comprised of a single monitor: CrashWrangler.

```

<Agent name="WinAgent">
  <Monitor class="WindowsDebugger">

    <!-- The command line to run. Notice the filename provided matched up
        to what is provided below in the Publisher configuration -->
    <Param name="Executable" value="c:\mplayer\mplayer.exe" />
    <Param name="Arguments" value="fuzzed.wav" />

    <!-- This parameter will cause the debugger to wait for an action-call in
        the state model with a method="StartMPlayer" before running
        program.
    -->
    <Param name="StartOnCall" value="StartMPlayer" />

    <!-- This parameter will cause the monitor to terminate the process
        once the CPU usage reaches zero.
    -->
    <Param name="CpuKill" value="true"/>

  </Monitor>

  <!-- Enable heap debugging on our process as well. -->
  <Monitor class="PageHeap">
    <Param name="Executable" value="mplayer.exe"/>
  </Monitor>

</Agent>

<Agent name="LinAgent">
  <!-- Register for core file notifications. -->
  <Monitor class="Gdb">

    <!-- This is the program we're going to run inside of the debugger -->
    <Param name="Executable" value="mplayer"/>

    <!-- These are arguments to the executable we want to run -->
    <Param name="Arguments" value="sample.wav"/>

    <!-- This parameter will cause the monitor to terminate the process
        once the CPU usage reaches zero.
    -->
    <Param name="CpuKill" value="true"/>

  </Monitor>

</Agent>
```

```

<Agent name="OsxAgent">
  <Monitor class="CrashWrangler">
    <!-- The executable to run. -->
    <Param name="Command" value="mplayer" />

    <!-- The program arguments. Notice the filename provided matched up
        to what is provided below in the Publisher configuration -->
    <Param name="Arguments" value="fuzzed.wav" />

    <!-- Do not use debug malloc. -->
    <Param name="UseDebugMalloc" value="false" />

    <!-- Treat read access violations as exploitable. -->
    <Param name="ExploitableReads" value="true" />

    <!-- Path to Crash Wrangler Execution Handler program. -->
    <Param name="ExecHandler" value="/usr/local/bin/exc_handler" />

    <!-- This parameter will cause the monitor to wait for an action-call in
        the state model with a method="StartMPlayer" before running
        program.
    -->
    <Param name="StartOnCall" value="StartMPlayer" />

  </Monitor>
</Agent>

```

Read more about: [Agent](#), [WindowsDebugger](#), [PageHeap](#)

Configure Test

Okay, now we just need to enable the agent for our test. Head down to the `<Test>` element, specifically we are looking to uncomment this line, and add a new parameter to the Publisher indicating an agent has been configured.

```

<!-- <Agent ref="LocalAgent"/> -->

```

Leaving us with this:

```
<Test name="Default">
    <Agent ref="WinAgent" platform="windows"/>
    <Agent ref="LinAgent" platform="linux"/>
    <Agent ref="OsxAgent" platform="osx"/>

    <StateModel ref="TheState"/>

    <Publisher class="File">
        <Param name="FileName" value="fuzzed.wav"/>
    </Publisher>
</Test>
```

Read more about: [Test](#)

Running the Fuzzer

Now let's actually kick off our fuzzer for real!

```
c:\wav>c:\peach\peach.exe wav.xml
```

23.2.7. Optimize Testing

At this point, we could just kick off our fuzzer and grab a beer; but, we can optimize our fuzzer in a few ways to reduce the number of iterations it will perform. For example, all of the actual PCM/WAV samples and music data is probably something that doesn't need much fuzzing. All that fuzzing the music data will do is create unhappy sounds. So, let's dial down the mutators that will run against them.