

Holodeck Enterprise Edition

Help Documentation

Last Updated 01/04/06

Table of Contents

Getting Started	8
Getting started guide	9
Holodeck Tutorials.....	10
Holodeck support center.....	15
Learning Holodeck Basics	16
How to use this guide.....	17
Other resources	18
Installing and running Holodeck	19
System Requirements.....	20
Installing Holodeck.....	21
Running Holodeck as a Restricted User	23
EULA	25
What's new in this version.....	29
What's new in this version.....	30
Exploring the Holodeck workspace	33
Windows and panes overview	34
Project toolbar.....	36
Moving, detaching and hiding panels.....	38
Menus overview	39
Menus in Depth	40
File Menu	41
Session Menu	43
Application Menu	44
Log Menu	46
Tools Menu	47
View Menu	48
Help Menu	49
Holodeck Windows and Panes	50
Corrupted Files Pane.....	51
Dynamic Help Pane.....	53
Exception Pane	54
Faults Pane.....	56
File Corruption Details Pane.....	57
File Corruption Faults Pane.....	58
Help Pane	60
Limits Pane	61
Log Pane	62
Network Corruption Faults Pane	64
Network Logs Pane	66
Network Message Details Pane	68
Project Pane	70
Properties Pane.....	73
Resource Faults Pane	74
Test Pane	78

Tutorials 80

Beginner	81
Creating your first project.....	82
Introduction	83
Creating the Project.....	84
Saving the Project	88
How to Use Faults to Deprive Excel of Memory	89
Introduction	90
Set Insufficient Memory	91
Verify Insufficient Memory	92
Summary	93
How to Deprive Your Application of its Dependencies	94
Introduction	95
Investigate Resources	96
Set the Resource Fault.....	97
Restart BrokenApp with Resource Faults	98
Investigating the Failure	100
Summary	102
Dealing with Multiple Processes	103
Introduction	104
Adding a second process to a project	105
Setting per process items	107
Create a File in Use fault for Notepad.exe	108
Create a Memory Fault for BrokenApp.exe.....	109
Create a Memory Limit for Notepad.exe	110
Spawning Another Process from the BrokenApp.....	111
Summary	115
Intermediate.....	116
Using Holodeck to Corrupt Files and Test File Processing	117
Introduction	118
Corrupting Files	119
Using Corrupted Files to test file processing.....	123
Viewing the details of the corrupted file.....	124
Summary	126
Creating Network Corruption Using Regular Expressions.....	127
Introduction	128
Investigate Network Packets	129
Open Network Corruption Wizard	131
Select Which Data Should be Corrupted.....	132
Select Regular Expressions as the Method of Corruption	133
Create a Regular Expression and Expansion String	134
Verify the Fault is set	137
Summary	139
Failing a Single API Call	140
Introduction	141
Investigate Logs	142
Failing the ReadLine function	144
Verify ReadLine Fault is Set	151
Summary	153
Learning About API Failures During Startup and Comparing Logs	154
Introduction	155
Load project	156
Filtering the log result	157
Creating a Report for easy viewing	158
Exporting logs	159

Comparing logs	160
Making sense of the difference	162
Summary	164
Using Code Coverage with Record/Replay to Test the Application	165
Set up the project	166
Start Recording the Session.....	167
Begin Code Coverage Test Generation	169
Stop Test Generation	172
Replay the Recorded Session.....	173
Summary	174
Advanced.....	175
Test Harness and Code Coverage	176
Introduction.....	177
Create a Holodeck project file	178
Setup the Test Harness.....	179
Begin testing with Code Coverage	180
How to compare log files	181
Summary	184
Adding New Intercepts.....	185
Adding New Intercepts Introduction	186
Create a Test dll	187
Create the Test Application	189
Add a New Intercept Library.....	192
Compile the TestDLL Replacement Library	195
View Logs and Create a Test for the Test dll	196
Summary	199
Create a New Test Project.....	200
Create a New Test Project Introduction	201
Create the Custom Test Project.....	202
Use the Custom Test Project to Test Notepad.....	206
Summary	208
Holodeck in depth	209
Introduction to Holodeck in depth	210
New Project Wizard	211
New Project Wizard.....	212
Launch an Application.....	214
Attach to an Application	215
Start a Service.....	216
Attach to a Service	217
New Project Options	218
Default Logging	220
Creating Tests	221
Tests Overview	222
Creating and Deleting a test	223
Scheduled Test Wizard.....	225
This wizard will help you fail a certain function call.....	225
Modifying a test	232
Creating a Test Based on Stack Matching	233
Creating Limits.....	234
Limits overview.....	235
Creating a Disk Space limit.....	237
Creating a Memory Space limit.....	238
Creating a Network Bandwidth limit.....	239
Creating Faults	240
Faults Overview	241

Adding and Removing a fault.....	242
Adding custom faults.....	244
Types of faults.....	249
Fault Categories.....	250
Disk Faults.....	251
Memory Faults.....	252
Network Faults.....	253
Registry Faults.....	254
Process/Library Faults.....	255
Custom Faults	256
Resource Faults	257
Resource Fault Overview.....	258
Adding and Removing a Resource Fault.....	259
Create a New Resource Fault Wizard	262
Working with Logs	265
Logs Overview	266
API Log Categories	267
Custom Log Filtering and Sorting	268
To find text in the log files	269
Changing which functions are logged.....	270
Exporting Logs	271
Printing Logs	272
Exporting Resource Logs.....	273
Network Logs	274
Working with Reports	275
Reports Overview.....	276
Managing and Printing Reports	277
What's Contained in a Report.....	278
Working with Pivot Tables.....	279
Reports Tables.....	280
API Log Summary Table	281
Faults Summary Table	282
Tests Summary Table	283
Error Codes Summary Table.....	284
Resource Dependencies Summary Table.....	285
Resource Usage Summary Table	286
Return Values Summary Table	287
Using Automatic Test Generation	288
Introduction to Automatic Test Generation	289
Using Holodeck for Stress Test Generation	290
Using Holodeck for Code Coverage Test Generation	292
Modifying Testing Settings.....	295
Generating Network Corruption Faults	296
Using the Network Corruption Fault Wizard	297
Creating a fault using Random Corruption	298
Creating a fault using Find and Replace Corruption.....	302
Creating a fault using Regular Expressions.....	306
Generating File Corruption Faults	311
File Corruption Overview	312
Create a File Corruption Fault using Random Corruption	314
Create a File Corruption Fault using Find and Replace.....	317
Create a File Corruption Fault with Regular Expressions.....	320
Regular Expressions and Replacement Strings.....	323
Using regular expressions	324
Using replacement strings	326
Multiple Threads and Processes	328

Dealing with Multiple Processes	329
Working with Multi-threaded applications	331
Exceptions and mini-dumps	333
Exceptions Overview.....	334
Mini-dump Overview	335
Recording and replaying sessions	336
Introduction to recording and replaying	337
Creating a recorded session.....	338
Replaying a recorded session.....	339
Add Holodeck Intercepts	340
Add Holodeck Intercepts.....	341
Holodeck Intercepts Manager.....	343
Add Holodeck Intercepts Wizard.....	344
Select the functions to add to the Holodeck Database	345
Review the Definitions of the Functions	346
Review Your Choices	348
Replacement Library Results	349
Automated Testing with Holodeck	351
Introduction to Automating Holodeck	352
Using Holodeck with other Automated Test Frameworks	353
Using Holodeck with other Automated Test Frameworks.....	354
Using Holodeck as a command-line utility.....	355
Introduction to using Holodeck as a command-line Utility	356
Code Coverage testing from the command-line	358
Stress testing from the command-line	360
Using Recorded Sessions from the command-line.....	362
Running Holodeck silently.....	363
Create a Custom Test Project	364
Create a Custom Test Project Wizard	365
Create a Custom Test Project.....	366
Selecting Libraries and Methods.....	367
Specifying the Output Options	368
Custom Test Project Summary	369
HolodeckLib	370
Using HolodeckLib	371
HolodeckLib Method List.....	372
HolodeckLib Constructors and Destructors	374
HolodeckLib Application Execution and Control	375
HolodeckLib Log Operations.....	378
HolodeckLib Limit Control	379
HolodeckLib Fault Control.....	380
HolodeckLib Interception Control.....	381
Using HoloScript.....	382
Using HoloScript	383
HoloScript Method List.....	384
HoloScript Functions	390
Constructors and Destructors.....	391
Application and Service Launch	392
Process Chaining	395
Debugging Functions.....	397
Application and Service Control	398
Logging Control	404
Tests Control	407
Limits Control.....	408
Fault Control	410

Examples	413
First Example - Injecting Faults	414
Second Example - Examining AUT Execution Environment.....	416
Third Example - Modifying the AUT Execution Environment (Disk and Memory Limits)	418
Fourth Example - Modifying the AUT Execution Environment (Network Limits)	420
Using HEAT (Hostile Environment for Application Testing)	422
Using HEAT	423
Step by Step for HEAT API.....	424
HeatAPI Method List	425
Constructors and Destructors of HeatAPI.....	427
Interception Information and Control.....	428
Intercept Function Control.....	430
HEAT API Injection	432
HEAT API Example.....	433
Intercepting Overloaded Functions with HEAT.....	437
Appendices	438
About fault injection	439
How to Break Software.....	452
How to Break Software Security.....	453
Windows 2000 and attaching the Holodeck Debugger	454
API Call Categories	455
Error Codes	456
Exception Codes	457
Field Chooser	458
HEAT Application Execution	459
HEAT Error Codes	460

Getting Started

Getting started guide

The best place to start if you are a new user is with the Holodeck Tutorials. The tutorials have been set up so you can start with the first tutorial Setting up your first project and work through to the last while learning all the basics of Holodeck. Holodeck includes an application with easily discoverable bugs written into it, many of the tutorials have been created using this app.

Each of the tutorials should take approximately half an hour to complete, and will teach you not only the basics of Holodeck, but also how to track down the bugs once you've found them, and what many bugs look like when they are discovered in an app.

To learn more about the Holodeck workspace please see the Windows and panes overview.

Holodeck Tutorials

The Holodeck tutorials will help you to discover the power of Holodeck in an easy to learn step by step method. By completing these tutorials you will learn the visual environment of Holodeck as well as methods to more fully test your application.

Each tutorial focuses on a specific task that can help you test your application. We suggest you complete the tutorials in order, however if you already feel comfortable with Holodeck you may choose to review only the sections of interest to you.

Beginner:

Creating your first project – This tutorial will show how to create a project using the BrokenApp that came with Holodeck and save that project to be used in later tutorials.

Takes approximately 10-20 minutes to complete and focuses on the following tasks:

Setting up your first project

The Create a New Project Wizard

Saving the Project

How to Use Faults to Deprive Excel of Memory – This tutorial will not allow Excel to allocate anymore memory, through the use of faults.

Takes approximately 10-20 minutes to complete and focuses on the following tasks:

Introduction

Set Insufficient Memory

Verify Insufficient Memory

Summary

How to deprive your application of its dependencies – This tutorial will explain how to deprive your application of its dependencies such as needed dlls, files, registry keys, and others.

Takes approximately 20-30 minutes to complete and focuses on the following tasks:

Introduction

Investigate Resources

Set the Resource Fault

Restart BrokenApp with Resource Faults

Investigating the Failure

Summary

Dealing with Multiple Processes – This tutorial will help explain how to setup a project with multiple applications under test.

Takes approximately 20-30 minutes to complete and focuses on the following tasks:

Introduction

Adding a second process to a project

Setting per process items

Create a File in Use fault for Notepad.exe

Create a Memory Fault for BrokenApp.exe

Create a Memory Limit for Notepad.exe

Spawning Another Process from the BrokenApp

Summary

Intermediate:

Using Holodeck to Corrupt files and test file processing – In this tutorial we will corrupt a file using Holodeck's built in file corruption and load it into the application under test.

Takes approximately 20-30 minutes to complete and focuses on the following tasks:

Introduction

Corrupting Files

Using Corrupted Files to test file processing

Viewing the details of the corrupted file

Summary

Creating Network Corruption Using Regular Expressions– This tutorial covers how to create a network corruption fault, and use regular expressions to find and replace specified strings on the network.

Takes approximately 30-40 minutes to complete and focuses on the following tasks:

Introduction

Investigate Network Packets

Open Network Corruption Wizard

Select Which Data Should be Corrupted

Select Regular Expressions as the Method of Corruption

Create a Regular Expression and Expansion String

Verify the Fault is set

Summary

Failing a Single API Call – This tutorial covers how to set a Scheduled Test to fail a single API call.

Takes approximately 10-20 minutes to complete and focuses on the following tasks:

Introduction

Investigate Logs

Failing the ReadLine function

Verify ReadLine Fault is Set

Summary

Learning about API failures during startup and comparing logs – In this tutorial we will learn how to investigate API failures using Holodeck's log and reporting tools.

Takes approximately 20-30 minutes to complete and focuses on the following tasks:

Introduction

Load project

Filtering the log result

Creating a Report for easy viewing

Exporting logs

Comparing logs

Making sense of the difference

Summary

Using Code Coverage with Record/Replay to Test the Application – This tutorial covers how to use the Code Coverage test generation in conjunction with the Record/Replay functionality to maximize testing efficiency and reproducibility.

Takes approximately 30-40 minutes to complete and focuses on the following tasks:

Set up the project

Start Recording the Session
Begin Code Coverage Test Generation
Stop Test Generation
Replay the Recorded Session
Summary

Advanced:

Test Harness and Code Coverage – This tutorial will demonstrate how to use an external test harness and Holodeck's Code Coverage feature to find bugs quickly, easily and efficiently.

Takes approximately 30-40 minutes to complete, and focuses on the following tasks:

Introduction
Create a Holodeck project file
Setup the Test Harness
Begin testing with Code Coverage
How to compare log files
Summary

Adding New Intercepts – This tutorial shows how to add new intercepts to Holodeck for application specific testing.

Takes approximately 40-50 minutes to complete, and focuses on the following tasks:

Adding New Intercepts Introduction
Create a Test dll
Create the Test Application
Add a New Intercept Library
Compile the TestDLL Replacement Library
View Logs and Create a Test for the Test dll
Further Exploration
Summary

Create a New Intercepts – This tutorial covers how to test an application using HoloScript through the Custom Test Project Wizard.

Takes approximately 30-40 minutes to complete, and focuses on the following tasks:

Create a New Test Project **Introduction**

Create the Custom Test Project

Use the Custom Test Project to Test Notepad

Further Exploration

Summary

Holodeck support center

Questions about Holodeck? Get quality answers and responsive help from the Holodeck **e-mail support team** at:

support@sisecure.com

Feel free to call this support telephone line anytime, a support specialist will return your call promptly.

1-877-SI-HELP-5 (877-744-3575)

Additional Help is available at the **Holodeck Resource Center**.

<http://www.sisecure.com/holodeck/learn.shtml>

You can find tips to help you use Holodeck more easily in the Tips Section of the Resource Center:

http://www.sisecure.com/holodeck/holodeck_tips.shtml

Be sure to visit the Security Innovation website <http://www.securityinnovation.com/> where you will find:

- News
- Technical Papers from our founder James Whittaker and other members of the engineering team at Security Innovation.
- White papers describing some of the foundations behind Holodeck
- Other helpful online resources for securing your application

Learning Holodeck Basics

How to use this guide

This guide has been separated into several sections. You can explore each section individually to best suit your interests and experience; however the help documentation has been written in such a way that exploring each section in order will be most beneficial. If you are new to Holodeck please read through the Windows and panes overview to familiarize yourself with the layout of the Holodeck workspace; read through the Holodeck Tutorials to learn the features of Holodeck through easy to follow step by step examples.

For more information on what has changed since the last version of Holodeck, please see the What's new in this version section.

The Holodeck In Depth section covers each feature in Holodeck in detail. Research how to test your application more fully by understanding how to use the features of Holodeck completely.

The Automated Testing with Holodeck section includes more advanced topics about Holodeck such as using Holodeck with other test frameworks, using with a command line, and the customizing and extension of Holodeck.

Note: Many aspects of Holodeck have changed since the previous version; you may want to read each chapter to learn about each new part of this Holodeck version.

Other resources

Online Help – Search our website for new help.

Tutorials – Use the tutorials to learn, step by step, new functionality of Holodeck.

Animated Demos – Online Animated Demos show you how to get started with Holodeck as well as advanced topics.

PDF version of the help – Included in the install of Holodeck is a PDF version of these help files, feel free to print this out for offline help.

How to Break Software – Learn how to more fully test your software with this book written by our founder, James Whittaker.

How to Break Software Security – Learn more in depth software security testing with this book, by the same author.

About Fault Injection – Learn more about how Holodeck works with fault injection.

Installing and running Holodeck

System Requirements

The following hardware and software is required to run Holodeck version 2.6.

Minimum System requirements for Microsoft Windows:

An Intel Pentium 4 Processor or equivalent, 1.5 GHz or faster
Windows 2000, Windows XP, Windows 2003 Server, Windows Longhorn
Version 4.0 or later of Microsoft Internet Explorer
256 MB of physical random-access memory (RAM)
200 MB of available disk space (80 MB for install, 120 MB for virtual memory if needed)
A monitor and video card capable of 256 colors and 800 x 600 pixel resolution

Recommended System requirements for Microsoft Windows:

An Intel Pentium 4 Processor or equivalent, 2.0 GHz or faster
Windows 2000, Windows XP, Windows 2003 Server, Windows Longhorn
Version 4.0 or later of Microsoft Internet Explorer
512 MB of physical random-access memory (RAM)
200 MB of available disk space (80 MB for install, 120 MB for virtual memory if needed)
A monitor and video card capable of Millions of colors and 1024 x 768 pixel resolution

Installing Holodeck

Follow these steps to install Holodeck on a Windows computer.

Note: In certain operating systems, you can install or uninstall Holodeck only if you have Administrative privileges on your computer.

To install Holodeck:

1. Insert the Holodeck CD, or run the installer downloaded from the Security Innovation website
2. Follow the onscreen instructions
3. If prompted, restart your computer.

To register Holodeck:

1. Obtain a Serial Number and Registration Key by purchasing a license either over the phone or online.
2. Choose Help > Register Holodeck ...
3. Click Edit...
4. Enter the Serial Number and Registration Key into the text fields.
5. Click Register.
6. Verify License Details to see that registration completed successfully.



Running Holodeck as a Restricted User

To run Holodeck as a restricted user you must give the users group read/write access to both the HolodeckEE registry keys and the Holodeck install directory. To accomplish this follow these directions while logged on as an Administrator:

Note: Simple file sharing must be turned off to give the users group read/write permission to the Holodeck install directory. Follow these directions to turn off simple file sharing:

- 1) Open any folder
- 2) Select Tools > Folder Options
- 3) Select the View tab
- 4) Uncheck "Use simple file sharing (Recommended)"
- 5) Click OK

- 1) Grant the users group read/write access to the HolodeckEE registry keys
 - a) Run regedit.exe
 - i) Start > Run...
 - ii) Type **regedit.exe**
 - b) Right click the folder HolodeckEE found at HKEY_LOCAL_MACHINE\SOFTWARE\Security Innovation\Holodeck Enterprise Edition and select Permissions
 - c) Select the Users group
 - d) Grant the Users group full control by selecting the checkbox next to Full Control
 - e) Click OK to save the change
- 2) Grant the Users group read/write access to the install directory of Holodeck
 - a) If Holodeck was installed using the default directory it will be located at C:\Program Files\Security Innovation.
 - b) Browse to the install directory
 - c) Right click the folder Holodeck Enterprise Edition and select Sharing and Security...
 - d) Select the Security tab
 - e) Select the Users group
 - f) Grant the Users group full control by selecting the checkbox next to Full Control
 - g) Click OK to save the changes

Holodeck will now work as expected when launched from a restricted user account.

EULA

SOFTWARE LICENSE AGREEMENT

PLEASE READ THIS SOFTWARE LICENSE AGREEMENT CAREFULLY BEFORE USING THE HOODECK™ SOFTWARE ("SOFTWARE"). BY CLICKING "I ACCEPT" AND DOWNLOADING THE SOFTWARE, YOU ARE ACKNOWLEDGING THAT YOU HAVE READ AND UNDERSTAND THIS SOFTWARE LICENSE AGREEMENT AND THAT YOU AGREE TO BE BOUND BY ITS TERMS. IF YOU DO NOT AGREE TO BE BOUND BY THE TERMS OF THIS SOFTWARE LICENSE AGREEMENT, CLICK "I DO NOT ACCEPT" AND DELETE THE SOFTWARE FROM YOUR COMPUTER WITHOUT USING IT.

1. License Grant. Subject to the terms of this Software License Agreement ("License"), Security Innovation, Inc. ("Licensor") grants you a nontransferable, nonexclusive, limited license to use the Software and any associated documentation files (collectively, the "Products") with which this License is distributed. You may only use the Products for your internal business purposes or for your own personal use. You may not:

- Copy, translate, port, modify, or make derivative works of the Products;
- Assign this License without Licensor's prior written approval;
- Rent, disclose, publish, sell, assign, lease, sublicense, market, or transfer the Products or use them in any manner not expressly authorized by this Agreement; or
- Derive or attempt to derive the source code, source files, or structure of all or any portion of the Products by reverse engineering, disassembly, decompilation, or any other means.

2. Ownership. You have no ownership rights in the Products. Rather, you have a license to use the Products as long as this License remains in effect. Ownership of the Products and all intellectual property rights in them shall remain at all times with Licensor.

3. Copyright. The Products contain material that is protected by United States copyright law and trade secret law, and by international treaty provisions. All rights not granted to you by this License are expressly reserved by Licensor. You may not remove any proprietary notice of Licensor from any copy of the Products.

4. Confidentiality. You acknowledge that the Products contain proprietary trade secrets of Licensor and you hereby agree to maintain the confidentiality of the Products using at least as great a degree of care as you use to maintain the confidentiality of your own most confidential information, but in no event less than a reasonable degree of care. You agree to reasonably communicate the terms and conditions of this License to those persons employed by you who come into contact with the Products, and to use your best efforts to ensure their compliance with those terms and conditions, including, without limitation, not knowingly permitting such persons to use any portion of the Products for the purpose of deriving the source code for the Products.

5. Disclaimer of Warranties. YOU EXPRESSLY AGREE AND ACKNOWLEDGE THAT USE OF THE PRODUCTS IS AT YOUR SOLE RISK. THE PRODUCTS ARE PROVIDED "AS IS," WITH ALL FAULTS, AND WITHOUT WARRANTY OF ANY KIND. LICENSOR EXPRESSLY DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND TITLE/NON-INFRINGEMENT. LICENSOR DOES NOT WARRANT THAT THE PRODUCTS WILL MEET YOUR REQUIREMENTS, THAT THE PRODUCTS ARE COMPATIBLE WITH ANY PARTICULAR HARDWARE OR SOFTWARE PLATFORM, OR THAT THE OPERATION OF THE PRODUCTS WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT DEFECTS IN THE PRODUCTS WILL BE CORRECTED. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE PRODUCTS IS ASSUMED BY YOU. FURTHERMORE, LICENSOR DOES NOT WARRANT OR MAKE ANY REPRESENTATION REGARDING THE USE OR THE RESULTS OF THE USE OF THE PRODUCTS OR RELATED DOCUMENTATION IN TERMS OF THEIR CORRECTNESS, ACCURACY, QUALITY, RELIABILITY, APPROPRIATENESS FOR A PARTICULAR TASK OR APPLICATION, CURRENTNESS, OR OTHERWISE. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY LICENSOR OR LICENSOR'S AUTHORIZED REPRESENTATIVES SHALL CREATE A WARRANTY OR IN ANY WAY INCREASE THE SCOPE OF LICENSOR'S OBLIGATIONS HEREUNDER.

SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSIONS MAY NOT APPLY TO YOU. IN THAT EVENT, ANY SUCH IMPLIED WARRANTIES RELATING TO THE PRODUCTS ARE LIMITED IN DURATION TO FIFTEEN (15) DAYS FROM THE DATE OF FIRST USE OF THE PRODUCTS. THIS WARRANTY GIVES YOU SPECIFIC RIGHTS. YOU MAY HAVE OTHER RIGHTS, WHICH VARY FROM STATE TO STATE.

6. Limitation Of Liability. IN NO EVENT SHALL LICENSOR BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES (INCLUDING, WITHOUT LIMITATION, INDIRECT, SPECIAL, PUNITIVE, OR EXEMPLARY DAMAGES FOR LOSS OF BUSINESS, LOSS OF PROFITS, BUSINESS INTERRUPTION, LOSS OF DATA, OR LOSS OF BUSINESS INFORMATION) ARISING OUT OF OR CONNECTED IN ANY WAY WITH USE OF OR INABILITY TO USE THE PRODUCTS, OR FOR ANY CLAIM BY ANY OTHER PARTY, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. LICENSOR'S TOTAL LIABILITY TO YOU FOR ALL DAMAGES, LOSSES, AND CAUSES OF ACTION (WHETHER IN CONTRACT, TORT (INCLUDING NEGLIGENCE), OR OTHERWISE) SHALL NOT EXCEED THE AMOUNT OF THE LICENSE FEES PAID BY YOU FOR THE PRODUCTS AND DOCUMENTATION.

SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

7. Term and Termination. Upon payment by you of the first year's license fee, the License shall be in effect for an initial term of one (1) year, unless sooner terminated in accordance with this Section. If you are in compliance with the terms and conditions of this License, and provided that you pay the annual renewal license fee each year in the manner prescribed by Licensor, you will have the right to renew the License for successive renewal terms of one (1) year each. You may terminate this License at any time by destroying or returning to Licensor all known copies of the Products in your possession or under your control. This License shall terminate automatically in the event you breach any of the terms and conditions herein. Upon notification of termination, you agree to destroy or return to Licensor all copies of the Products and to certify in writing that all known copies, including backup copies, have been destroyed. All provisions relating to confidentiality, proprietary rights, and nondisclosure shall survive termination of this License.

8. License Fees. License fees are due and payable in advance on an annual basis. The license fee applicable for any renewal year shall be Licensor's then- current license fee for the Products.

9. Equitable Relief. You acknowledge and agree that Licensor will be irreparably injured if the provisions of Sections 1 (License Grant) and 4 (Confidentiality) were not capable of being specifically enforced, and agree that Licensor shall be entitled to equitable remedies for any breach of Sections 1 and 4, in addition to, and cumulative with, any legal rights or remedies, including the right to damages.

10. Severability. If any provision of this License is found to be invalid or unenforceable by any court, such provision shall be ineffective only to the extent that it is in contravention of applicable laws without invalidating the remaining provisions of the License.

11. Choice Of Law. This License shall be construed in accordance with the laws of the State of Florida without reference to the conflicts of law principles thereof.

12. Venue. You agree that all actions or proceedings arising in connection with the License shall be tried and litigated exclusively in the state courts located in or for Melbourne, Florida. This choice of venue is intended by the parties to be mandatory and not permissive in nature, and to preclude the possibility of litigation between the parties with respect to, or arising out of, this License in any jurisdiction other than that specified in this Section. Each party waives any right it may have to assert the doctrine of *forum non conveniens* or similar doctrine or to object to venue with respect to any proceeding brought in accordance with this Section.

13. Entire Agreement. This License constitutes the entire agreement between the parties with respect to the subject matter of this License, and supersedes all other prior and

contemporary agreements, understandings, and commitments between the parties regarding the subject matter of this License. This License may not be modified or amended except by a written instrument executed by the parties.

**YOU ACKNOWLEDGE THAT YOU HAVE READ THIS
SOFTWARE LICENSE AGREEMENT, UNDERSTAND IT, AND
AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS.**

What's new in this version

What's new in v. 2.6

Holodeck Version 2.6 includes the following upgrades:

- Compatible with .NET 2.0 Framework - You can now use Holodeck to test applications developed with the latest .NET Framework
- Full support for Windows 2003 Server - Now test your applications on the Win2K3 platform

.

What's new in v. 2.5

Holodeck Version 2.5 has many great features that will make Holodeck even more useful. With many features improved, added and revised Holodeck is by far the most powerful Fault Injection Tool available.

Add new Intercepts

You can now intercept any win32 DLL or .NET assembly and Holodeck will intercept any public methods just as it does for system API calls. This allows monitoring and testing of any public interface installed on the computer

Improved Scheduled Tests

You can now have a Scheduled test fire on with the following features

- Fire xx% of time
- Fire based on call stack matching

You can also pause the application when a test fires so you can set other tests, faults etc at that exact moment.

Improved control over the application

You can now remove, stop, or restart any application in the current project. This allows you to recover after the application crashes, or restart the application if it gets into an unpredictable or frozen state.

Improved File Corruption

File Corruption is integrated fully into Holodeck. Files are corrupted at the time of use and Holodeck redirects the application to the new corrupted file. This allows you to create a different corrupted file every time the application accesses the file.

Added intercepted API functions

- "Dangerous" APIs from Writing Secure Code
- Native COM methods
- Many more win32 and .NET functions

Improved .NET Interception Support

Holodeck now supports reference parameters, private methods, and system reflection.

Improved Fault Scenarios

Holodeck's out of the box fault scenarios are now more realistic, matching real world conditions. You can now modify existing faults or create your own custom faults.

Added Resource Based Faults

You can now fail access to a specific process, library, file, folder, registry key, registry value or COM object.

Improved API Logging

Holodeck now intercepts all API calls regardless of whether it was called directly by the application or not. This allows you to see if APIs called other APIs. Logs are displayed in a treeview showing how the call was made.

Fully Supported Windows Services

You can now launch or attach to any service regardless of whether it is a standalone service or part of a svchost process.

Added .NET to the Custom Test Project Generator

Using the Custom Test Project Holodeck can now generate test code for .NET as well as win32 APIs which allow you to create your own logic for intercepted functions.

Added Network Packet Logging

You can now see all the network packet traffic from the application and drill down into each packet to find out exactly what's being sent and received over the network to your application. If you've created a network corruption fault corrupted bytes are highlighted in the Network Message Details Pane.

Improved Code Coverage Test Generation

Holodeck now varies when the test will be applied to more thoroughly test the application. Added corruption and resource faults to the types of tests run under Code Coverage Test Generation.

Integrated Debugger

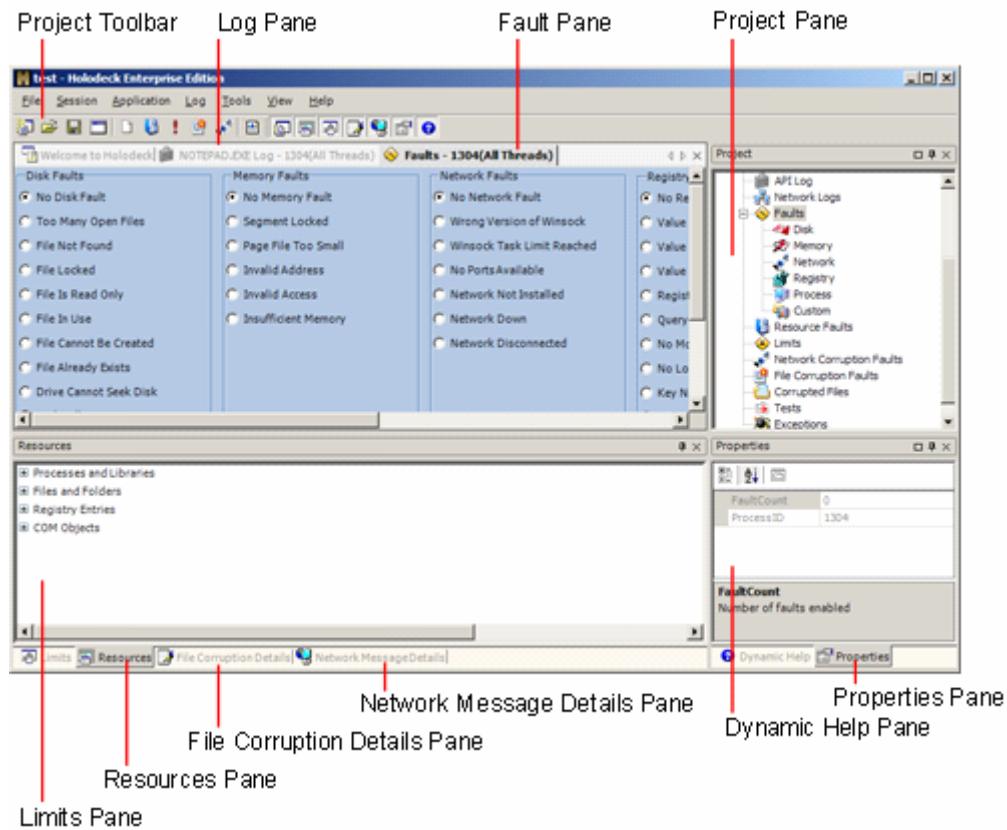
Holodeck now ships with an integrated debugger that catches all application crashes and exceptions and creates a mini-dump that can later be loaded into Visual Studio for debugging.

New Help Documentation

Revised and extended Help Documentation which are task based rather than feature based. We have added more tutorials and demonstration videos that walk the user through the most important tutorials. Holodeck now ships with a Broken Application that is used in many of the tutorials so you can see and verify failures in a known faulty program.

Exploring the Holodeck workspace

Windows and panes overview



Windows and Panes Visible by Default

Project toolbar – This toolbar contains items pertaining to your current project, creating a new project, and showing and hiding panes. Follow this link to find out what each does.

Log Pane – The log pane displays all the APIs Holodeck is currently intercepting.

Faults Pane – This pane allows you to quickly set faults exactly the same way as if the error happened within the Operating System.

Project Pane – The Project Pane will help you to display and organize your project entries into a tree view. The majority of the information about your project will be accessible from this pane.

Limits Pane – The limits pane makes it easy to limit the amount of disk, memory, or networking resources your application has available to it.

Resource Pane – Clicking this tab will show the Resource Pane which shows the file, folder, process, library, registry resources your application is currently using.

File Corruption Details Pane – Clicking this pane will the File Corruption.

Network Message Details Pane – This pane shows the data bytes sent in each network message.

Dynamic Help Pane – Watch the Dynamic Help Pane for information on the task you are currently doing. This pane will change and show information and tips to help you complete your task.

Properties Pane – Clicking this tab will show the properties pane will provide you with information regarding selected UI objects. Look here for additional information about the UI you are currently working with

Windows and Panes not Visible by Default

Corrupted Files Pane – Once your application access a file with a corruption fault the resultant corrupted file will show up as an entry in this pane.

Exception Pane – If you have enabled Holodeck as a debugger it will catch the exceptions your application throws and create a mini-dump file to aid in debugging.

File Corruption Faults Pane – Each File Corruption Fault you create will show up as an entry in this Pane.

Help Pane – Holodeck will display the full length help text items in this pane.

Network Corruption Faults Pane – Each Network Corruption Fault will show up as an entry in this pane.

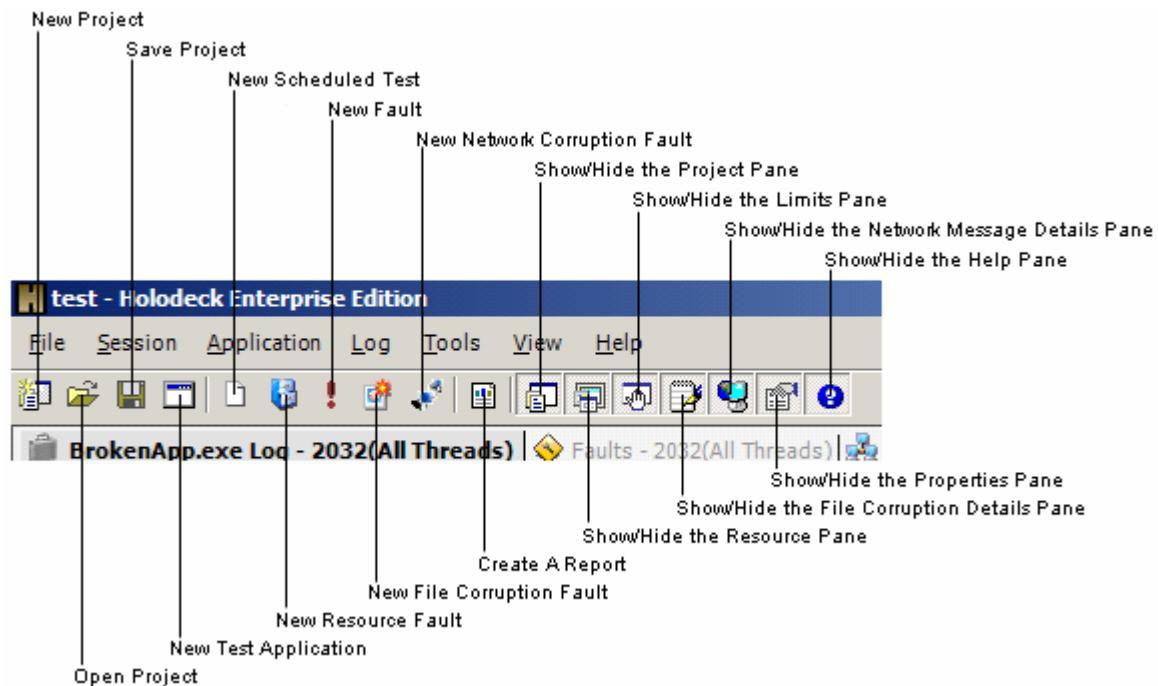
Network Logs Pane – Holodeck intercepts every packet your application sends and receives, look to this pane for information on those network packets.

Resource Faults Pane – Each Resource Fault will show up as an entry in this pane.

Test Pane – Each Scheduled Test you create will show up as an entry in this Pane.

Project toolbar

The project toolbar gives you easy access to some of the more common tasks that Holodeck provides; it also gives you an easy way to hide and show the different panes.



New Project – Opens the new project wizard to begin testing your application.

Open Project – Opens an open file dialog box so you can select a previously saved project.

Save Project – Saves the current project for later use

New Test Application – allows you to add another test application to the already running project

New Scheduled Test – Opens the new scheduled test wizard so you can create a scheduled test

New Resource Fault – Opens the new resource test wizard so you can create a resource test

New Fault – Opens the faults pane so you can set new faults for your application.

New File Corruption Fault – Opens the new file corruption fault wizard.

New Network Corruption Fault – Opens the new network corruption fault wizard.

Create a Report – Generates a Holodeck report for easy viewing of errors, API calls, and other information regarding your application.

Show/Hide the Project Pane – Shows or hides the project pane

Show/Hide the Resource Pane – Shows or hides the resource pane

Show/Hide the Limits Pane – Shows or hides the Limits pane

Show/Hide the File Corruption Details Pane – Shows or hides the File Corruption Details pane

Show/Hide the Network Message Details Pane – Shows or hides the network message details pane

Show/Hide the Properties Pane – Shows or hides the properties pane

Show/Hide the Help Pane – Shows or hides the dynamic help pane

Moving, detaching and hiding panels

Holodeck has been created to use a familiar layout design so things are placed in a easy to find and use location. For this reason you can detach, move, or hide any pane. This is especially useful for a multi-monitor environment; you can detach each of the panes and place them on the secondary monitor while having the main window of Holodeck and the Application Under Test in the main window.

To detach a complete pane click the title bar of the pane and drag to the location you desire. Valid docking locations are anywhere along the outer edge of the window. You may also "float" a window anywhere, including separate monitors.

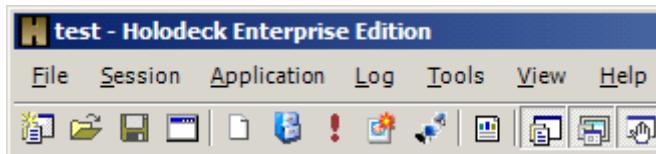
To detach a single pane click the tab, located at the bottom of the pane, and drag to the location you desire. You can dock or float this pane the same way you can a complete pane.

To hide a pane you can "unpin" the pane so it automatically hides itself while not in use. To pin or unpin a pane click the pin icon on the title bar.

To close the pane use the "X" on the title bar of the pane. Once a pane has been closed you can use the button bar at the top of the window, or the view menu open the pane again.

The pane layout is saved each time you close Holodeck

Menus overview



The File menu contains menu items that relate directly to your project file such as: New Project, New Test Application, Open, Close, and Save.

The Session menu contains menu items that relate to all the applications in the session you are currently working with. This includes Test Generation, Recording and Replaying sessions and turning on Per Thread functionality.

The Application menu contains items that apply to the application that currently has focus in Holodeck.

The Log menu contains items to find keywords in the log files and filter the log results.

The Tools menu gives you access to other tools useful for testing your application such as the Custom Test Project Generator and a method to add Holodeck Intercepts.

Use the View menu to show or hide the workspace panes.

The Help menu includes menu items pertaining to the help topics, how to report a bug in Holodeck, register Holodeck, and information about Holodeck.

Menus in Depth

File Menu

The file menu contains items standard to this menu, including items to create a new project, open an existing project, save the current project, close and exit this session.

New Project ... - More information on creating a new Holodeck Project.

New Test Application ... - Add another test application to the currently running project.

Open Project... – Open a previously saved project.

Close – Close this project, Holodeck will remain active.

Export Log to File ... - Export the currently selected API logs to a file to an easy to read csv spreadsheet file.

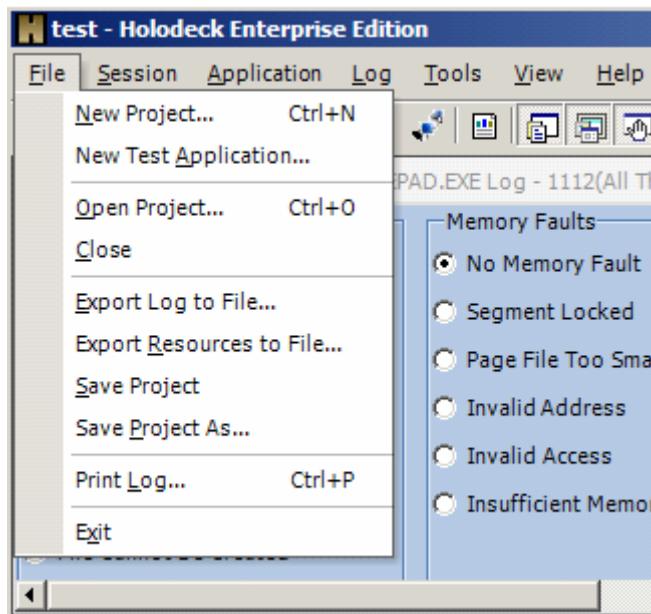
Export Resources to a File – Holodeck can export the current list of resources your application is using to an easy to read csv spreadsheet file.

Save Project – Save the current project for later use.

Save Project As... - Save the current project with a new name.

Print Log ... - Print the currently selected API or Network logs.

Exit – Close Holodeck completely



Session Menu

The session menu contains items that apply to all applications and threads in the session.

Code Coverage Test Generator ... - Holodeck can help you test your applications more thoroughly by setting faults, limits, and tests while your application is running.

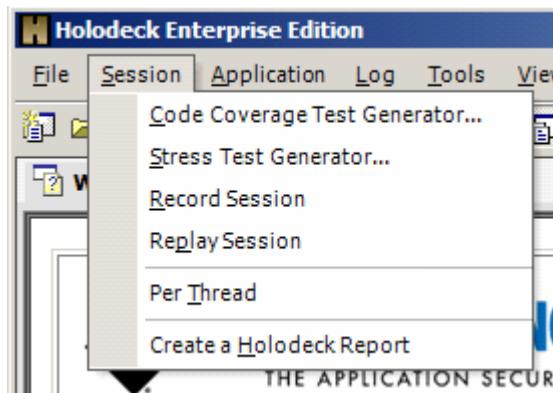
Stress Test Generator ... - Holodeck can help stress test your application by creating specific scheduled tests while your application is running.

Record Session – Record the tests, limits, faults etc your application undergoes by using this functionality.

Replay Session – Replay a recorded session to reproduce the test environment your application has undergone.

Per Thread – Holodeck automatically logs which APIs are called from which threads, create tests and faults specific to each thread by turning on Per Thread functionality.

Create a Holodeck Report – Holodeck reports help organize a Holodeck session into a single easy to read file.



Application Menu

The application menu contains items that are specific to each individual application. From this menu you can pause, stop, remove or restart an application in the current project without affecting the other applications in your project.

Copy selected item – copies whatever is currently selected to the clipboard.

Pause Application – Pauses the application from continuing, this allows you to set faults, tests, etc. at a precise moment.

Stop – Halts the application or process completely. You can restart the application using the Restart feature.

Restart – Restarts a previously stopped application. When an application is restarted all tests, faults, etc. are carried over to the new process. If you have set any faults or test in per-thread mode these faults and tests will be changed to per-process mode, since Holodeck can not map threads across a process change.

Remove – Removes the currently selected process from the project.

Attach Debugger and Log Exceptions – Holodeck defaults to log all exceptions your application creates, you can attach a debugger by double clicking on an exception from the exceptions pane. Toggle this functionality here.

Log First Chance Exceptions – Holodeck does not log first chance exceptions by default, however if you would like to see these exceptions you can configure to catch and log every exception that your application creates.

Child Process Inherits Settings – If your application spawns a child process, it will automatically inherit the same tests and faults that the parent process had set.

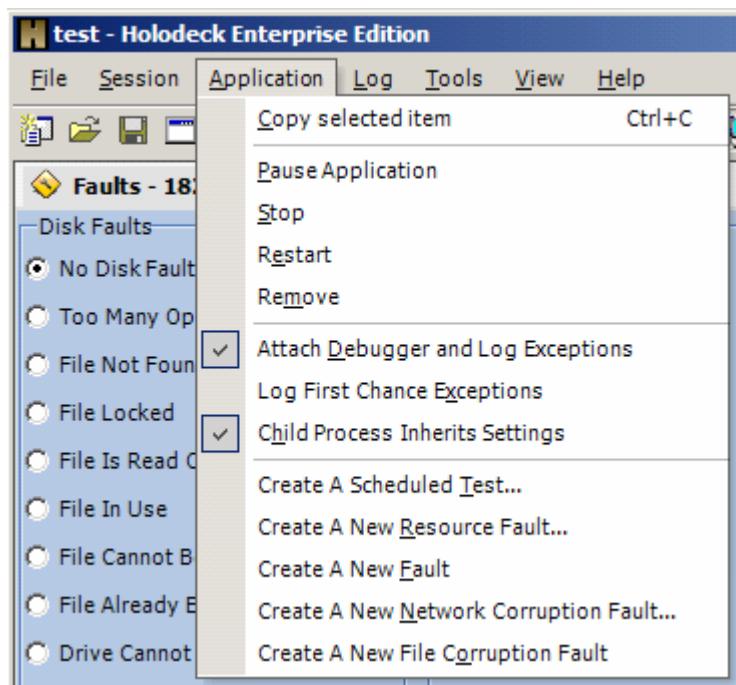
Create a new Schedule Test – Scheduled tests are specific to an individual API function.

Create a new Resource Fault – Resource faults are specific to an individual resource such as a library, process, registry key, or file.

Create a new Fault – Faults are an out of the box testing solutions to help simulate faults common to software and hardware.

Create a new Network Corruption Fault – Holodeck can corrupt network packets being sent or received, through the use of random, find and replace, or regular expression corruption.

Create a new File Corruption Fault – Holodeck can help test file parsing by corrupting individual files.



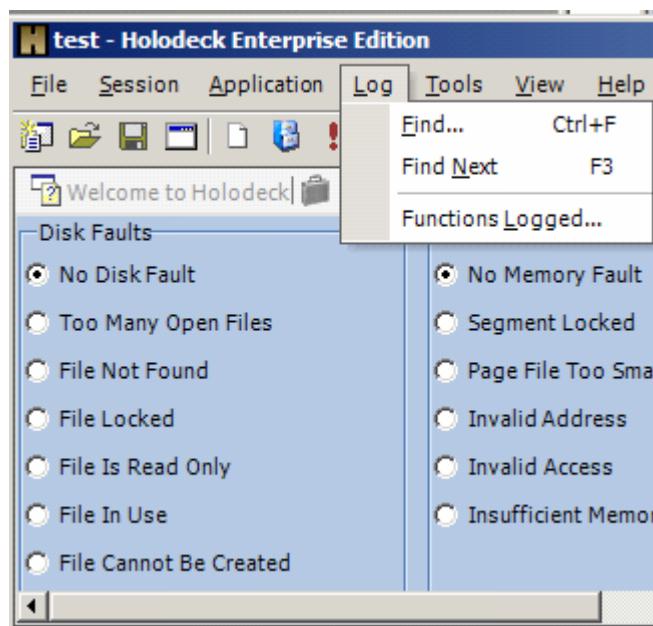
Log Menu

The log menu allows you to find specific logged functions in the logs pane, or change which APIs functions are logged.

Find – Quickly search for text within the API or Network logs

Find Next – Search through the logs to find the next occurrence of the text specified in the find window.

Functions Logged – Change which functions are intercepted and logged.



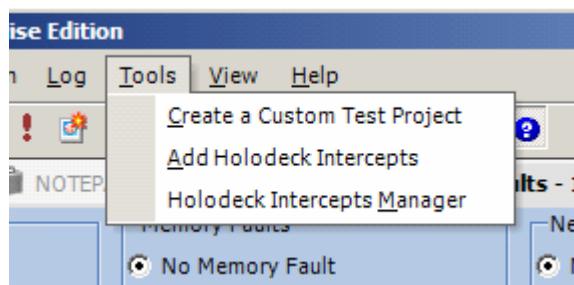
Tools Menu

Holodeck comes with a number of other useful tools.

Create a Custom Test Project – You can create a Visual Studio project to intercept custom libraries.

Add Holodeck Intercepts – You can add custom intercepts to Holodeck's list of APIs by running this wizard.

Holodeck Intercepts Manager – Add new or remove intercepts you have created through the Add new Holodeck Intercepts wizard.



View Menu

The view menu allows you to toggle which panes are visible, and choose which fields are viewable in the current window.

Project Pane – This menu item shows or hides the Project Pane

Resource Pane – This menu item shows or hides the Resource Pane

Limits Pane – This menu item shows or hides the Limits Pane

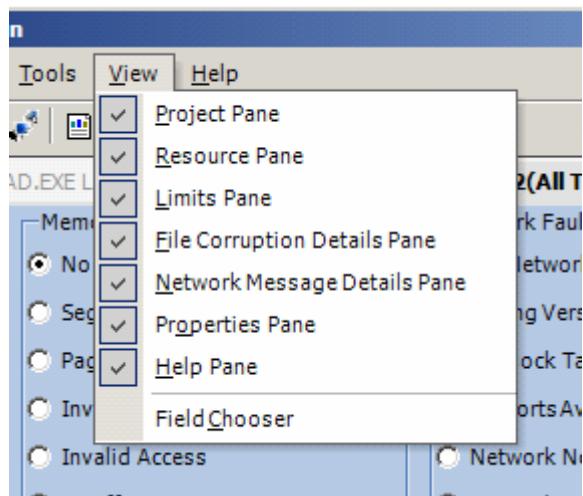
File Corruption Details Pane – This menu item shows or hides the File Corruption Details Pane

Network Message Details Pane – This menu item shows or hides the Network Message Details Pane

Properties Pane – This menu item shows or hides the Properties Pane

Help Pane – This menu item shows or hides the Help Pane

Field Chooser – The Field Chooser allows you to change which fields are shown in panes with selectable columns.



Help Menu

The Help menu includes menu items pertaining to the help topics, how to report a bug in Holodeck, register Holodeck, and information about Holodeck.

Help Topics... - This will display the complete Holodeck documentation system.

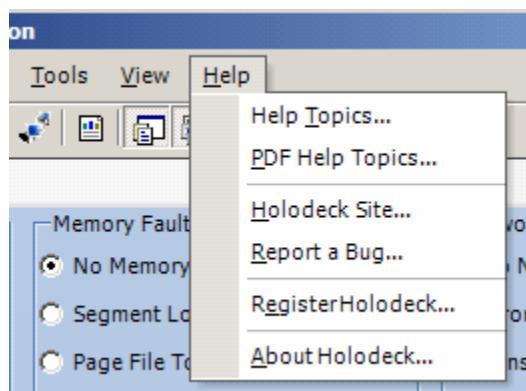
PDF Help Topics... - This will display the complete Help documentation in PDF form for printing.

Holodeck Site... - This menu item will take you to the Holodeck section of the Security Innovation website.

Report a bug... - If you think you've found a bug in Holodeck, please let us know so we can make the next version better.

Register Holodeck ... - Register Holodeck or get information to register Holodeck here.

About Holodeck – Displays information about Holodeck, the build number and a link to the Security Innovation Website.



Holodeck Windows and Panes

Corrupted Files Pane

Through the Corrupted Files Pane Holodeck allows you to save the original file, save the corrupted file, save the file corruption details, and delete the corrupted file. Saving the corrupted file is useful for reproducing file parsing bugs since Holodeck may not corrupt the file the same each time.

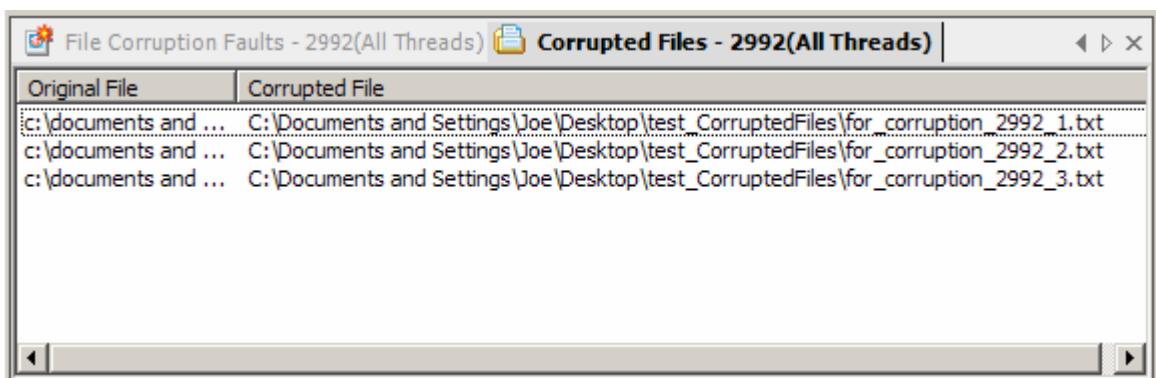
You can also save the corruption details to an RTF file by right clicking the corrupted file and selecting "Save RTF details." Saving the RTF details of a corrupted file exports the offset, hex value, and ascii representation of the file. Before each modification to the original file a red x is displayed; any other modification to the file is represented in both the hex values and ASCII in blue text. This is also good file to attach to a bug report since it may contain unprintable characters that will help a developer track down and fix the reported bug.

The items in the Corrupted Files pane show up as the application accesses the file with corruption. At the time of access Holodeck creates the corrupted file and redirects your application to it, leaving the original file in tact.

Show the Corrupted Files Pane by double-clicking on the Corrupted Files Node in the Project Pane.

Original File – The path to the uncorrupted original file.

Corrupted File – The path to the newly generated corrupted file



Original File	Corrupted File
c:\documents and ...	C:\Documents and Settings\Joe\Desktop\test_CorruptedFiles\for_corruption_2992_1.txt
c:\documents and ...	C:\Documents and Settings\Joe\Desktop\test_CorruptedFiles\for_corruption_2992_2.txt
c:\documents and ...	C:\Documents and Settings\Joe\Desktop\test_CorruptedFiles\for_corruption_2992_3.txt

Right click one of the corrupted files for the following options:

Save Original File – Saves the original, uncorrupted, file.

Save Corrupted File – Saves the corrupted file to a location you specify.

Save RTF Details – Saves the file corruption details in an RTF file, with all corruption changes highlighted.

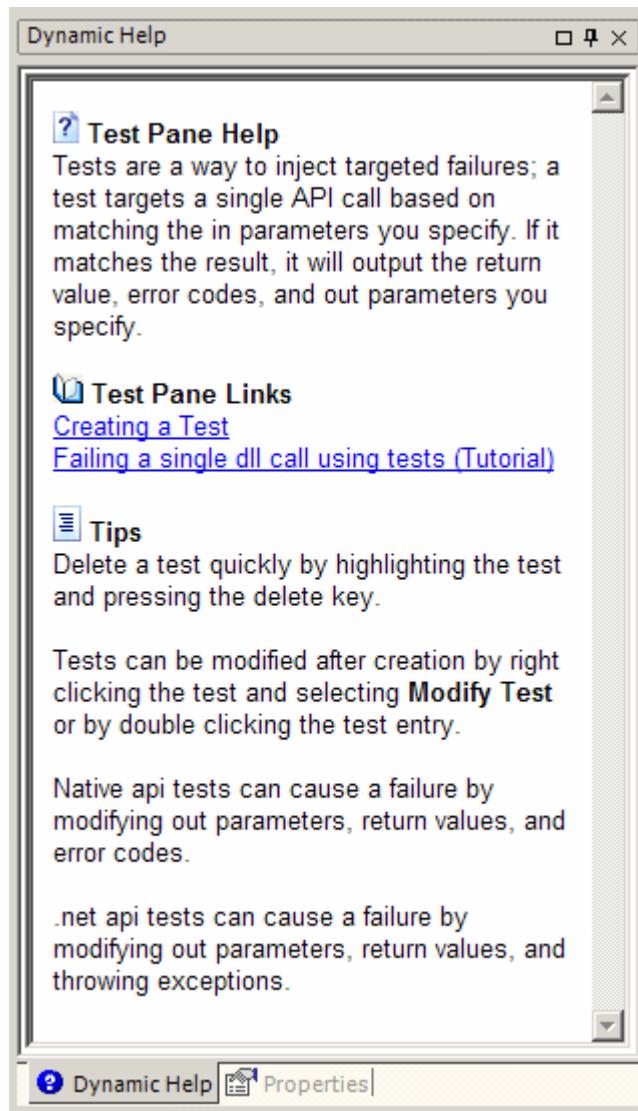
Delete Corrupted File – Deletes this corrupted file from the disk.

Select Visible Columns... - Allows you to choose which columns are displayed in this pane.

Dynamic Help Pane

The dynamic help pane will change to display help related to the pane that is currently selected. A brief introduction, links to useful help topics and quick tips will be displayed in this pane. Follow one of the provided links to learn more about the task at hand.

To show the Dynamic Help pane click the Show/Hide Dynamic Help button on the Project toolbar or click View > Dynamic Help on the menu.



Exception Pane

Holodeck can catch all exceptions your application produces; however by default Holodeck will only log second chance exceptions. Exceptions are routed through Holodeck first, as a first chance exception, then passed back to the application to handle. If the application does not handle the exception it is logged as a second chance exception.

View the Exception in a debugger by right clicking an exception in the Exception Pane and selecting "View Exception in Debugger." For more information on debugging Exceptions please see the Exceptions Overview.

To have Holodeck log both first and second chance exceptions either specify that on the last page of the New Project Wizard or select Session > "Attach Debugger and Log Exceptions" from the Menu.

You can save the mini-dump information for debugging the application at a later time. The mini-dump file is a snapshot of the computer state at the time of the crash. For more information on debugging using the mini-dump file please see the Mini-dump Overview.

To show the Exception pane double-click the Exception node in the Project Pane.

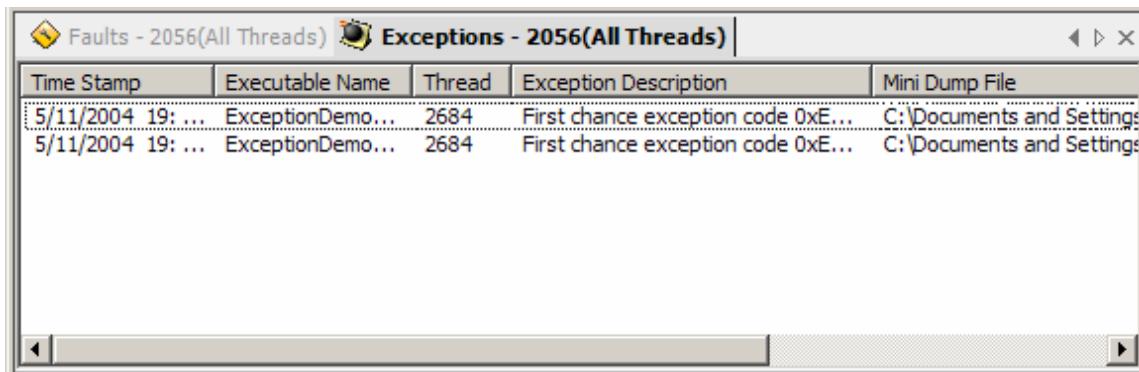
Time Stamp – The time in which the exception occurred.

Executable Name – The process name that caused the exception.

Thread – The thread ID that caused the exception.

Exception Description – a short description of a first chance or second chance exception and where the exception occurred in memory.

Mini Dump File – The location of the Mini-Dump Holodeck has created.



Time Stamp	Executable Name	Thread	Exception Description	Mini Dump File
5/11/2004 19: ...	ExceptionDemo...	2684	First chance exception code 0xE...	C:\Documents and Settings\...
5/11/2004 19: ...	ExceptionDemo...	2684	First chance exception code 0xE...	C:\Documents and Settings\...

Right click an exception for the following options:

Copy – Copy the details of the exception. Sample data follows:

Executable name: resizeImage.exe

Time stamp: 5/28/2004 15:35:58:919

Exception: First chance exception code 0xE0434F4D at 0x77E73887

Mini Dump File: C:\Documents and
Settings\user\Desktop\test_Exceptions\Exception1.dmp

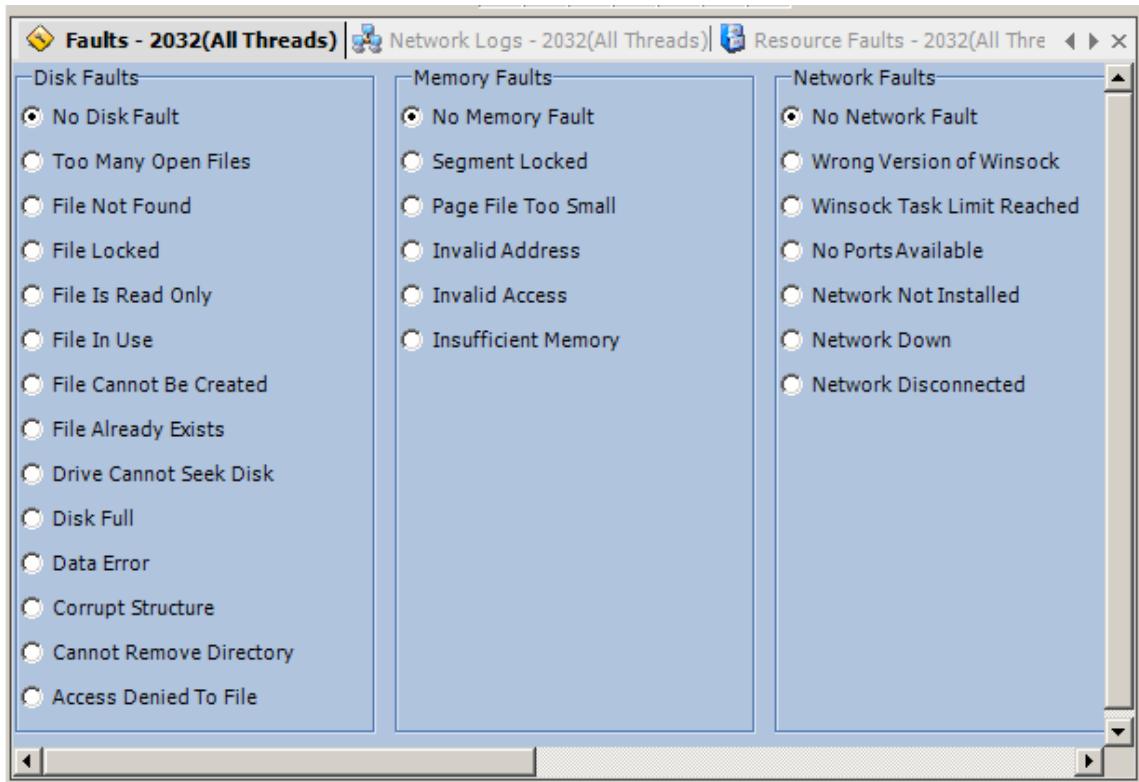
View Exception in Debugger – This will load the Exception and Mini-dump information into the Visual Studio so that you can debug the application. For more information on using the debugging information provided by Holodeck please see the Exceptions and Mini-dump overviews

Faults Pane

The Faults Pane shows all available "out of the box" faults Holodeck ships with. Use these faults to simulate different hardware and software failures, easily and precisely. To enable each fault simply select a bullet at the right of the fault you would like to enable. Holodeck supports one fault per category.

Holodeck shows the Faults Pane once the application under test has finished loading.

To view the Faults Pane otherwise, double click the Faults node or a Faults Category node in the Project Pane.



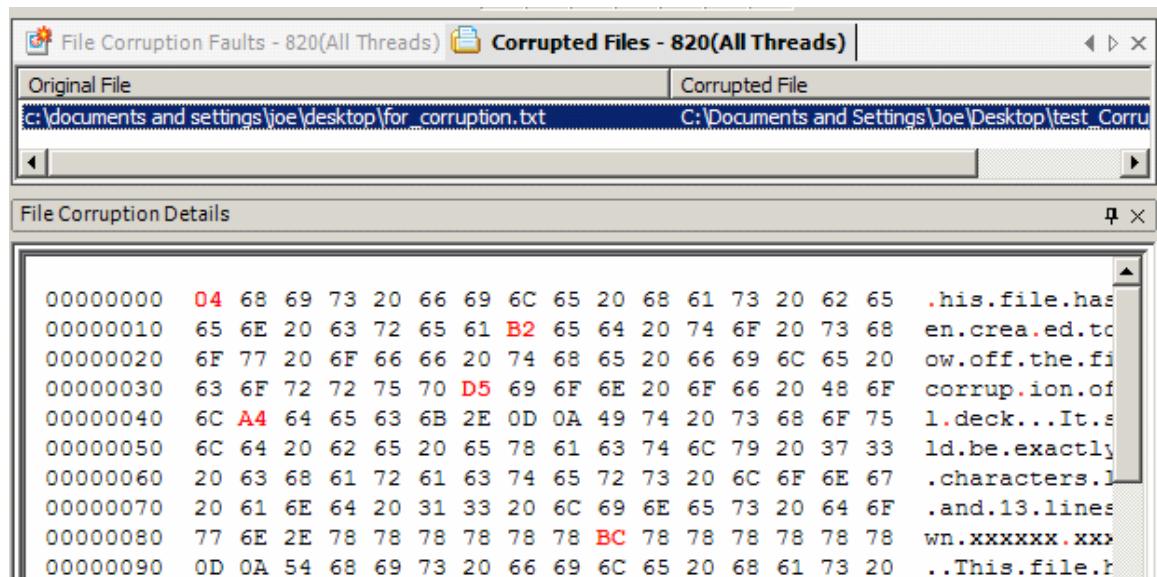
See the Faults Overview for more information concerning faults.

File Corruption Details Pane

The file corruption details pane shows exactly how a file has been corrupted. All changes Holodeck has made to a file through the file corruption faults will show up here. Holodeck will highlight any changes made in red. The center column is a hexadecimal display of the file, and how it has been changed by Holodeck; use this display to see changes in characters that may not be able to be able to be displayed. The right column shows the file in its ASCII representation so it is easy to read.

You must select an item from the Corrupted Files Pane to show the File Corruption Details Pane.

To show the File Corruption Details Pane double-click a Corrupted file in the Corrupted Files Pane.



The screenshot shows the 'Corrupted Files - 820(All Threads)' pane. It displays two files side-by-side: 'Original File' (c:\documents and settings\joe\desktop\for_corruption.txt) and 'Corrupted File' (C:\Documents and Settings\Joe\Desktop\test_Corru...). The 'File Corruption Details' pane below shows the hex and ASCII representations of the files. Red highlights are present in the hex and ASCII columns, indicating corrupted or modified data. The ASCII representation shows parts of the original file's content, such as '.his.file.ha...', 'en.crea.ed.t...', 'ow.off.the.fi...', 'corrup.ion.of...', 'l.deck...It.s...', 'ld.be.exactly...', '.characters.l...', '.and.13.lines...', 'wn.xxxxxxx.x...', and '..This.file.r...', with several characters highlighted in red.

Address	Original Hex	Corrupted Hex	Original ASCII	Corrupted ASCII
00000000	04 68 69 73 20 66 69 6C 65 20 68 61 73 20 62 65	04 68 69 73 20 66 69 6C 65 20 68 61 73 20 62 65	.his.file.ha...	.his.file.ha...
00000010	65 6E 20 63 72 65 61 B2 65 64 20 74 6F 20 73 68	65 6E 20 63 72 65 61 B2 65 64 20 74 6F 20 73 68	en.crea.ed.t...	en.crea.ed.t...
00000020	6F 77 20 6F 66 66 20 74 68 65 20 66 69 6C 65 20	6F 77 20 6F 66 66 20 74 68 65 20 66 69 6C 65 20	ow.off.the.fi...	ow.off.the.fi...
00000030	63 6F 72 72 75 70 D5 69 6F 6E 20 6F 66 20 48 6F	63 6F 72 72 75 70 D5 69 6F 6E 20 6F 66 20 48 6F	corrup.ion.of...	corrup.ion.of...
00000040	6C A4 64 65 63 6B 2E 0D 0A 49 74 20 73 68 6F 75	6C A4 64 65 63 6B 2E 0D 0A 49 74 20 73 68 6F 75	l.deck...It.s...	l.deck...It.s...
00000050	6C 64 20 62 65 20 65 78 61 63 74 6C 79 20 37 33	6C 64 20 62 65 20 65 78 61 63 74 6C 79 20 37 33	ld.be.exactly...	ld.be.exactly...
00000060	20 63 68 61 72 61 63 74 65 72 73 20 6C 6F 6E 67	20 63 68 61 72 61 63 74 65 72 73 20 6C 6F 6E 67	.characters.l...	.characters.l...
00000070	20 61 6E 64 20 31 33 20 6C 69 6E 65 73 20 64 6F	20 61 6E 64 20 31 33 20 6C 69 6E 65 73 20 64 6F	.and.13.lines...	.and.13.lines...
00000080	77 6E 2E 78 78 78 78 78 BC 78 78 78 78 78 78 78	77 6E 2E 78 78 78 78 78 BC 78 78 78 78 78 78 78 78	wn.xxxxxxx.x...	wn.xxxxxxx.x...
00000090	0D 0A 54 68 69 73 20 66 69 6C 65 20 68 61 73 20	0D 0A 54 68 69 73 20 66 69 6C 65 20 68 61 73 20	..This.file.r...	..This.file.r...

File Corruption Faults Pane

Each File Corruption Fault you create will show up as an item in this pane. You can turn faults on or off by toggling the checkmark beside each fault. Double click a fault to edit its corruption properties. Modify which columns are shown by right clicking anywhere in the pane and clicking "Select Visible Columns."

To create a new fault from the File Corruption Faults Pane right click anywhere in the pane and select "Create a new Fault"

When a file is accessed by the application under test it will show up in the Corrupted Files Pane

Turn a File Corruption Fault on or off quickly by toggling the check in the On/Off Column.

Show the File Corruption Faults pane by double-clicking the File Corruption Faults Node in the Project Pane.

On/Off – Signifies if this fault will execute or not.

Original File – The source file for corruption.

Type – The type of corruption as specified on the first page of the File Corruption Wizard.

Search For – The amount of corruption for random corruption, or the search string in find and replace or Regular Expression corruption.

Replace With – The string to replace the random corruption or matched search string.

Regenerate at Each Access – If Holodeck should regenerate the file at every access.

On/Off	Original File	Type	Search For	Replace With	Regene
<input checked="" type="checkbox"/>	C:\Documents...	Random	154 of 10000 bytes	Overwrite random single byte	No
<input checked="" type="checkbox"/>	C:\Documents...	Replace	Normal string "This"	Overwrite using Normal string "That"	No
<input checked="" type="checkbox"/>	C:\Documents...	RegExpr	[tT]his	aaaaaaaaaaaaaaaaaaaa...	Yes
<input type="checkbox"/>	C:\Documents...	Random	76 of 10000 bytes	Overwrite random single byte	No

Right click on one of the faults for the following options:

Copy – Copy the existing fault settings to the clipboard

Modify Fault... - Opens the last page of the new File Corruption fault wizard to allow you to edit the fault settings

Delete Fault – Removes the fault completely.

Right click anywhere in this pane for the following options:

Create a new Fault – Opens the new File Corruption Wizard

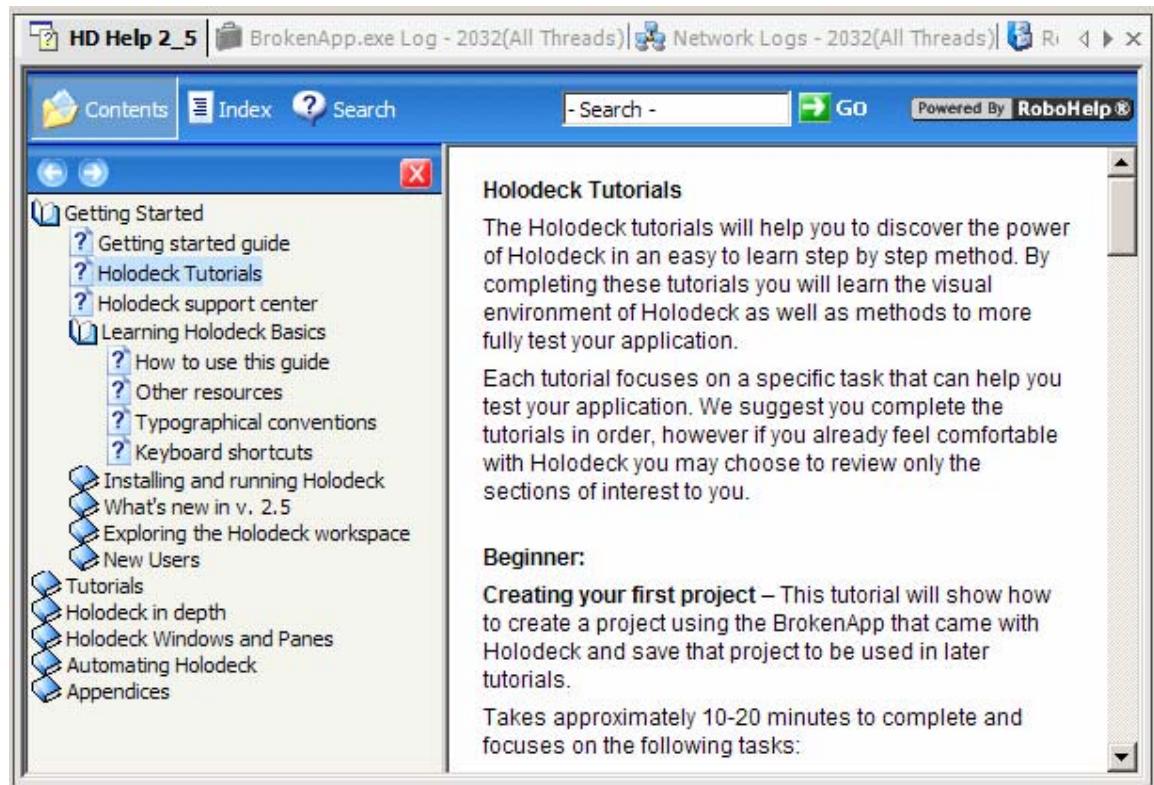
Select Visible Columns – Select which columns are visible in this pane.

Help Pane

The Help pane is the central pane that shows the main help topics. To print a help topic right click anywhere in main, lower right frame and select "Print..."

The complete help documentation in PDF form is available in the help directory or for download here.

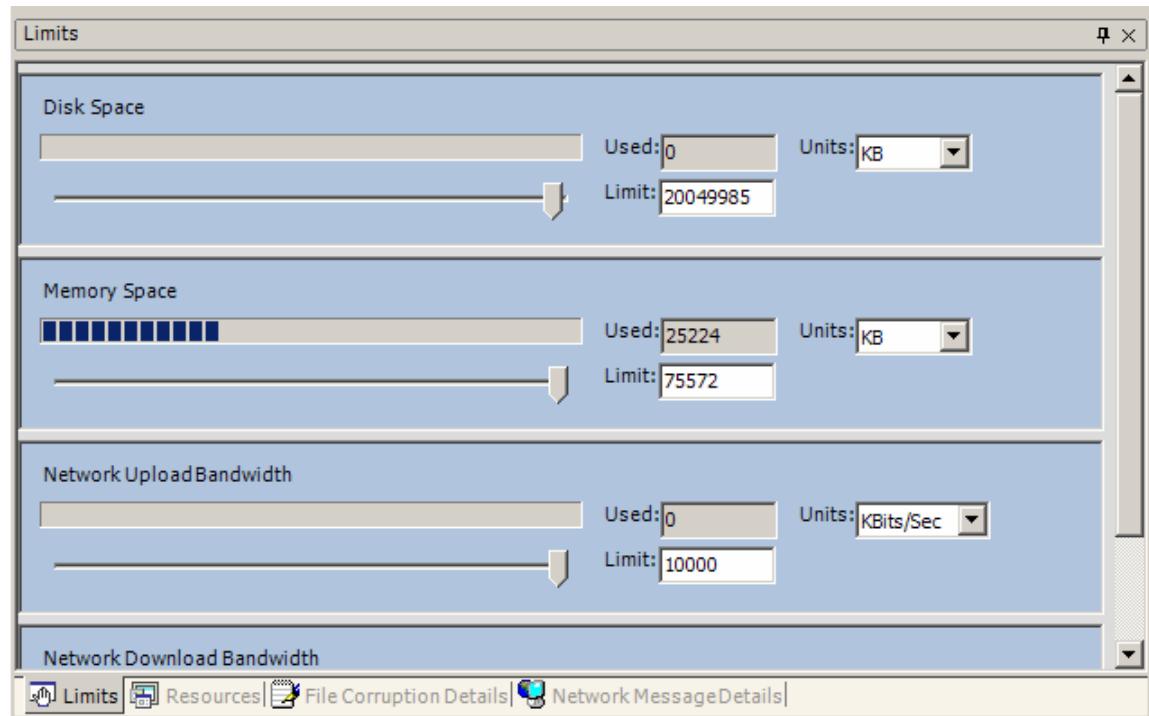
Show the Help pane by selecting Help > "Help Topics..." from the menu.



Limits Pane

The Limits Pane allows you to limit the amount of a resource your application has available to it. For instance, using Holodeck you could simulate a computer with a smaller Hard Drive, Less RAM, or a slower internet connection.

Show the limits pane by clicking the Show/Hide Limits Pane Node on the Project toolbar or by clicking View > Limits Pane from the Menu.



To find out more about Limits check out the Limits overview in the Holodeck in depth section.

Log Pane

As soon as you begin the Application Under Test Holodeck will start to log the API calls you have selected from the Logging Options page of the new project wizard. The log pane will hold any logs you are currently viewing. Each log will be titled by the application you are currently working with, in the format Appname processID(threadID) where All Threads for threadID means the log applies to all threads in this process.

Show the Log Pane by double-clicking the API node in the Project pane.

TimeStamp – This is the precise time in which Holodeck logged the API call the format of this category is Month/Day/Year Hour:Minute:Second:Milisecond.

Thread – All threads your application uses, will be listed here by thread number.

Category – This is the main category of the API call, Holodeck intercepts both win32 APIs and .NET APIs. For more information please see the API Log Categories

DLL – This column shows which .dll the intercepted API is located in.

Function – This shows which API calls the Application Under Test has called. The next few columns are information about that function call.

Return Value - The value returned from the function called, often these return values will be memory addresses, but more useful information might be looking for 1's or 0's (representing the call was made successfully or unsuccessfully) or values smaller than a memory address (actually useful return values) return values may also be TRUE, FALSE, or NULL.

Error Code – This column shows the error code returned by the API function. This is not set by all functions, some use return values to give status. Error Code List

Exception – If the function raised an exception it will show up in this column, this only applies while testing .NET applications. Exception List

List of Parameters – These columns contains all the parameter values.

TimeStamp	Thread	Category	DLL	Function	Return Value
05/10/2004 16:49:50:843	3672	DANGEROUS	kernel32.dll	EnterCriticalSection	NULL
05/10/2004 16:49:50:843	3672	LIBRARY	kernel32.dll	GetModuleHandleA	4194304
05/10/2004 16:49:50:843	3672	DANGEROUS	kernel32.dll	EnterCriticalSection	NULL
05/10/2004 16:49:50:843	3672	PROCESS	kernel32.dll	LeaveCriticalSection	NULL
05/10/2004 16:49:50:843	3672	DANGEROUS	kernel32.dll	EnterCriticalSection	NULL
05/10/2004 16:49:50:843	3672	PROCESS	kernel32.dll	LeaveCriticalSection	NULL
05/10/2004 16:49:50:843	3672	PROCESS	kernel32.dll	LeaveCriticalSection	NULL
05/10/2004 16:49:51:171	1496	DANGEROUS	kernel32.dll	EnterCriticalSection	NULL
05/10/2004 16:49:51:171	1496	PROCESS	kernel32.dll	LeaveCriticalSection	NULL
05/10/2004 16:49:51:171	1496	DANGEROUS	kernel32.dll	EnterCriticalSection	NULL
05/10/2004 16:49:51:171	1496	PROCESS	kernel32.dll	LeaveCriticalSection	NULL
+ 05/10/2004 16:49:51:171	1496	FILE	kernel32.dll	GetFileAttributesW	4294967295
05/10/2004 16:49:51:171	1496	DANGEROUS	kernel32.dll	EnterCriticalSection	NULL
05/10/2004 16:49:51:171	1496	PROCESS	kernel32.dll	LeaveCriticalSection	NULL

By default Holodeck will display the following information in the log pane: TimeStamp, Thread, Category, DLL, Function, Return Value, Error Code, Exception, and a list of parameters. For more information see the API Log Categories help topic.

Right click anywhere in this pane for the following options:

Copy – Copy the currently selected API function call; a sample copied function follows

Function: RegCloseKey(178)
Time: 05/28/2004 00:25:18.859
ThreadID: 1612
Category: REGISTRY
Dll Name: advapi32.dll
Error Code: 0
Exception:
Return Value: 0

Export Log to File – Export the currently displayed log into an easy to parse Comma Separated Value file.

Expand All – Expand all collapsed log entries showing functions called by other API functions

Collapse All – Collapse all log entries to show only the functions called directly from the main process.

Create a Scheduled Test – Open the Scheduled Test Wizard with the currently highlighted function loaded.

Select Visible Columns – Select which columns are displayed in the log pane.

For more information on working with logs, please see the Logs Overview in the Holodeck In Depth Section.

Network Corruption Faults Pane

The Network Corruption Faults Pane allows you to copy, modify, delete, and create new network corruption faults. To modify, copy or delete a fault right click the fault and select the correct operation from the Right-Click menu.

To Create a new fault by right clicking anywhere in the Network Corruption Faults Pane and selecting "Create a new Fault"

Choose which columns are visible by right clicking anywhere in pane and clicking "Select Visible Columns."

Turn a Network Corruption Fault on or off quickly by toggling the check in the On/Off Column.

To show the Network Corruption Faults pane double-click the Network Corruption Faults node in the Project Pane.

On/Off – If the network fault will execute on the next sent or received network message.

Port – The ports which messages will be corrupted.

Type – The type of corruption, Random, Find and Replace, or Regular Expression

Search For – The amount of corruption for random corruption, or the search string in find and replace or Regular Expression corruption.

Replace With – The string to replace the random corruption or matched search string.

Network Corruption Faults - 2840(All Threads)				
On/Off	Port	Type	Search For	Replace With
<input checked="" type="checkbox"/>	Recv on All	Random	1 of 10000 bytes	Overwrite random single byte
<input checked="" type="checkbox"/>	Recv on All	Replace	Normal string "www"	Replace with Normal string "ftp"
<input checked="" type="checkbox"/>	Send and Recv on All	RegExpr	www ftp	aaaaaaaaaaaaaaaaaaaaaaa...

Right click a network corruption fault for the following options:

Copy – Copy the fault details to the clipboard. Sample data follows.

Test On: True

Ports: Recv on All

Network Corruption Type: Random

Search For: 92 of 10000 bytes

Replace With: Overwrite random single byte

Network Corruption Type: Random

Modify Fault... – This will take you to the last page of the Network Corruption Faults Wizard so you can edit the settings of the fault.

Delete Fault – Remove this fault completely.

Right click anywhere in the pane for the following options:

Create a new Fault – Opens the Network Corruption Fault Wizard

Select Visible Columns – This option allows you to change which columns are visible in this pane.

Network Logs Pane

Holodeck automatically intercepts all messages sent and received over the network. These messages can be found in the network logs pane.

Select which columns are displayed by right clicking and selecting "Select Visible Columns..."

View the message details of a network message by opening the Network Message Details Pane

Export the contents of the Network logs by right clicking the Network Logs node in the Project Pane and selecting "Export Log to File"

Show the Network Logs pane by double-clicking on the Network Logs node in the Project Pane.

TimeStamp – The time at which the network message was sent or received.

Thread – The thread ID that sent or received the network message.

Direction – If the packet was sent or received.

Protocol – The protocol over which the packet was sent.

Msg Length – The length of this single message packet, long messages sometimes span multiple packets.

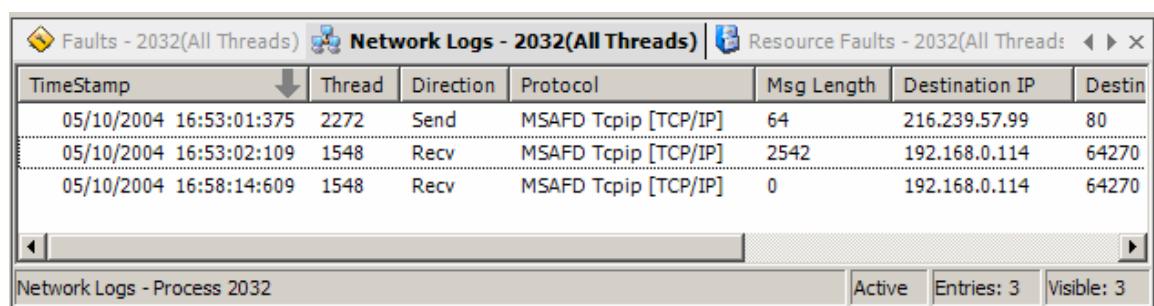
Destination IP – The IP the message was sent to.

Destination Port – The port used to receive the message.

Source IP – The IP the message was sent from.

Source Port – The port used to send the message

Start of Message – The first eight (8) bytes of the message.



TimeStamp	Thread	Direction	Protocol	Msg Length	Destination IP	Destin
05/10/2004 16:53:01:375	2272	Send	MSAFD Tcpip [TCP/IP]	64	216.239.57.99	80
05/10/2004 16:53:02:109	1548	Recv	MSAFD Tcpip [TCP/IP]	2542	192.168.0.114	64270
05/10/2004 16:58:14:609	1548	Recv	MSAFD Tcpip [TCP/IP]	0	192.168.0.114	64270

Right click anywhere in this pane for the following options:

Copy – Copy the currently selected Network message; a sample copied function follows

Type: Outgoing
TimeStamp: 05/28/2004 00:25:48:031
ThreadID: 3364
Protocol: MSAFD Tcpip [TCP/IP]
Destination IP Address: 216.239.53.99
Destination Port: 80
Source IP Address: 192.168.0.114
Source Port: 7685

Export Log to File – Export the currently displayed log into an easy to parse Comma Separated Value file.

Select Visible Columns – Select which columns are displayed in the log pane.

Note: In order to get network logs, Holodeck has to be attached when the connection is first established. This means if Holodeck has been attached to an already running instance of an application network logs will not show up.

For more information regarding the Network Logs see the help topic within Holodeck in Depth

Network Message Details Pane

This pane contains the details of the network message of the highlighted network log entry. Both the original packet and the corrupted packet are shown side by side. The offset, Hex Data and the ascii representation are shown for each packet. Any modification Holodeck has made to the packet through a Network Corruption Fault is highlighted in blue in the modified packet on the left. Modifications made to the packet are highlighted in red in the original packet data on the right.

This data is accessible through the Network logs pane. Find a packet you are interested in, and double click it. This will bring up the entire packet information into the Network Message Details Pane.

You must select an item from the network logs pane to view Network Message Details data.

Show the Network Logs Pane by double-clicking the Network Logs Node in the Project Pane.

Network Logs - 3712(All Threads) | **Network Corruption Faults - 3712(All Threads)**

TimeStamp	Thread	Direction	Protocol	Msg Length	Destination IP
05/24/2004 00:50:09:281	2872	Recv	MSAFD Tcpip [UDP/IP]	1	127.0.0.1
05/24/2004 00:50:09:281	2296	Send	MSAFD Tcpip [UDP/IP]	1	127.0.0.1
05/24/2004 00:50:09:281	2296	Send	MSAFD Tcpip [TCP/IP]	484	208.48.34.132
05/24/2004 00:50:10:390	2296	Recv	MSAFD Tcpip [TCP/IP]	1024	192.168.0.114

Network Logs - Process 3712

Active Entries: 431 Visible: 431

Network MessageDetails

Offset	Modified Hex Data	Modified ASCII
0x000000000	85 54 54 50 2f 31 2e 31	. T T P / :
0x000000010	0a 53 65 72 76 65 72 12	. S e r v e
0x000000020	66 74 2d 49 49 53 2f 35	f t - I I !
0x000000030	3a 20 4d 6f 6e 2c 0b 32	: . M o n ,
0x000000040	30 ec 20 30 36 3a 33 38	0 . . 0 6 :
0x000000050	0a 43 6f 6e 74 65 6e 74	. C o n t e
0x000000060	65 78 74 2f 68 74 6d 6c	e x t / h i
0x000000070	2d 52 61 6e 67 65 73 3a	- R a n g e
0x000000080	4c 61 73 74 2d 4d 6f 64	L a s t - 1
0x000000090	68 75 2c 20 33 30 20 4a	h u , . 3 (
0x0000000a0	31 36 3a 32 32 3a 34 37	1 6 : 2 2 :
0x0000000b0	61 67 3a 20 22 38 f3 32	a g : . " !
0x0000000c0	38 63 32 31 3a 37 63 63	8 c 2 1 :
0x0000000d0	6e 74 2d 4c 65 6e 67 74	n t - L e r
0x0000000e0	0d 0a 0d 0a 3c 68 74 6d	. . . < 1
0x0000000f0	42 65 67 69 6e 54 65 6d	B e g i n :
0x000000100	54 65 6d 70 6c 06 74 65	T e m p l .
0x000000110	62 6f 75 74 5f 75 73 2e	b o u t _ 1
0x000000120	0d 0a 3c 68 65 61 64 3e	. . < h e a
0x000000130	42 65 67 69 6e 45 04 69	R e n i n g

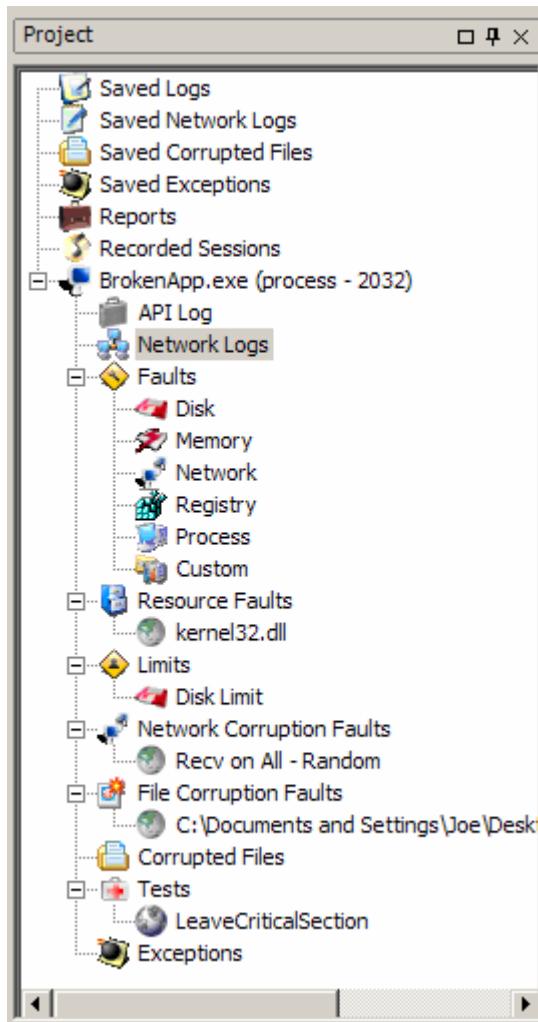
Limits | Resources | File Corruption Details | Network MessageDetails

Project Pane

The Project Pane will help you to display and organize your project entries into a tree view. The majority of the information about your project will be displayed here.

Double clicking on any node brings into focus the corresponding UI, for instance double clicking on the faults node will bring up the faults pane.

Show the Project Pane by clicking Project Pane button on the Project Toolbar, or click View > "Project Pane"



Global Nodes contain information that pertains to the entire session.

Saved Logs – API Logs which have been previously saved can be reviewed here.

Saved Network Logs – Network Logs which have been previously saved can be reviewed here.

Saved Corrupted Files – Corrupted Files which have been previously saved can be reviewed here.

Saved Exceptions – Exceptions which have been previously saved can be reviewed here.

Reports – View all reports created for this session.

Recorded Sessions – View and replay all Recorded Sessions for this session.

All nodes under a Process node contain information that pertains to the process.

If the Application Under Test has multiple threads or processes, each new thread or process will show up here, as a new node with its own Active Log, Resources, Faults, Limits and Tests.

Log Pane – View all the API function logs Holodeck has created.

Network Logs Pane – View the packets intercepted over the network.

Faults Pane – View and set faults.

Resource Faults Pane – View faults currently enabled, and create new ones.

Limits Pane – View and create hardware resource limits.

Network Corruption Faults Pane – View Network corruption currently enabled.

File Corruption Faults Pane – View File corruption currently enabled.

Corrupted Files Pane – View the Files that Holodeck has corrupted based on File Corruption Faults.

Test Pane – View the scheduled test currently enabled for your application.

Exception Pane – View the exceptions your application has produced and save mini-dump files.

Viewing with per-process granularity:

- 1) There are two scenarios in which Holodeck will show multiple processes.
 - a) Testing two applications at the same time.
 - b) The Test application spawns a child process.
 - i) Process Chaining must be turned on from the project wizard for Holodeck to follow the child processes

For more information on testing multiple processes see the Dealing with Multiple Processes help topic.

Viewing with per-thread granularity:

- 1) If your test application spawns new threads while running Holodeck will automatically begin logging these threads
- 2) By selecting the new thread you can view the active log for that thread, view the resources used by that specific thread, and set faults, limits or tests.

To turn on per-thread functionality Click Session > Per Thread.

When in per-thread mode, Holodeck automatically separates all relevant nodes in the project pane by ThreadID. This allows you to set faults and test to be specific to a single thread; you can also view API logs, Network Logs and Exceptions on a per-thread level.

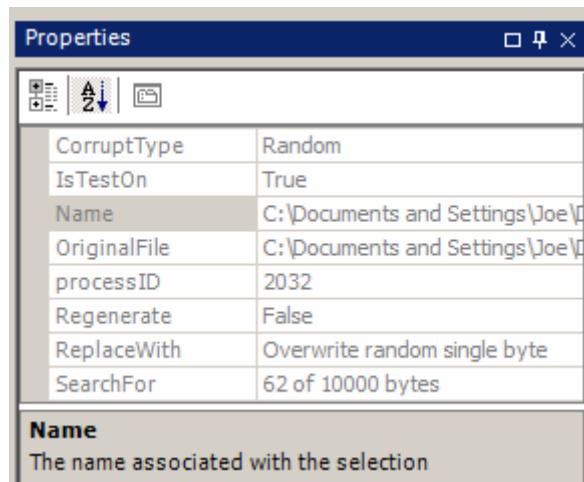
You can still set faults and tests to span an entire application, simply select the fault or tests node directly under the application node in the project pane. To view API logs, Network logs or Exceptions generated by the entire application, select the node directly under the application node in the Project Pane.

Properties Pane

The purpose of the property pane is to provide you with information regarding selected UI objects. Look here for additional information about the UI you are currently working with.

General Layout - The property pane toolbox window provides information in an expandable tree-view using a table. Each table can be expanded or collapsed using the +/- button in the upper left hand corner.

Show the Properties Pane by clicking the Show/Hide Property Pane button on the Project toolbar or Click View > "Properties Pane" from the menu.



Resource Faults Pane

The resource faults pane gives you access to each Resource Fault that has been created. Through it you can add, remove, disable, modify, or copy resource faults. Access these fault operations by right clicking a fault

Add a fault by right clicking anywhere in the Resource Faults Pane and selecting "Create a new Resource Faults."

Select which columns are visible in this pane by right clicking anywhere in the pane and selecting "Select Visible Columns"

Turn a Resource Fault on or off quickly by toggling the check in the On/Off Column.

To show the Resource Faults pane double-click the Resource Faults Node in the Project Pane.

On/Off – If the Resource Fault will execute on next access.

Resource Name – The resource the fault applies to.

Type – How access to the resource will be failed.

On/Off	Resource Name	Type
<input checked="" type="checkbox"/>	kernel32.dll	Access Denied
<input checked="" type="checkbox"/>	Kernel32	Not Enough Resources
<input type="checkbox"/>	HKEY_CURRENT_USER	Access Denied
<input checked="" type="checkbox"/>	HKEY_CURRENT_USER\Software\Mi...	Value Not Found

Right click a resource fault for the following options:

Copy – Copy the Fault Details to the clipboard. Sample data follows:

Resource: comctl32.dll

Fault: Invalid File Type

Test On: True

Modify Resource Fault ... - This option takes you to the last page of the Resource fault wizard where you can edit the specifics of the resource fault.

Delete Resource Fault – Removes the fault completely

Create a new Resource Fault – Opens the Create a New Resource Fault Wizard.

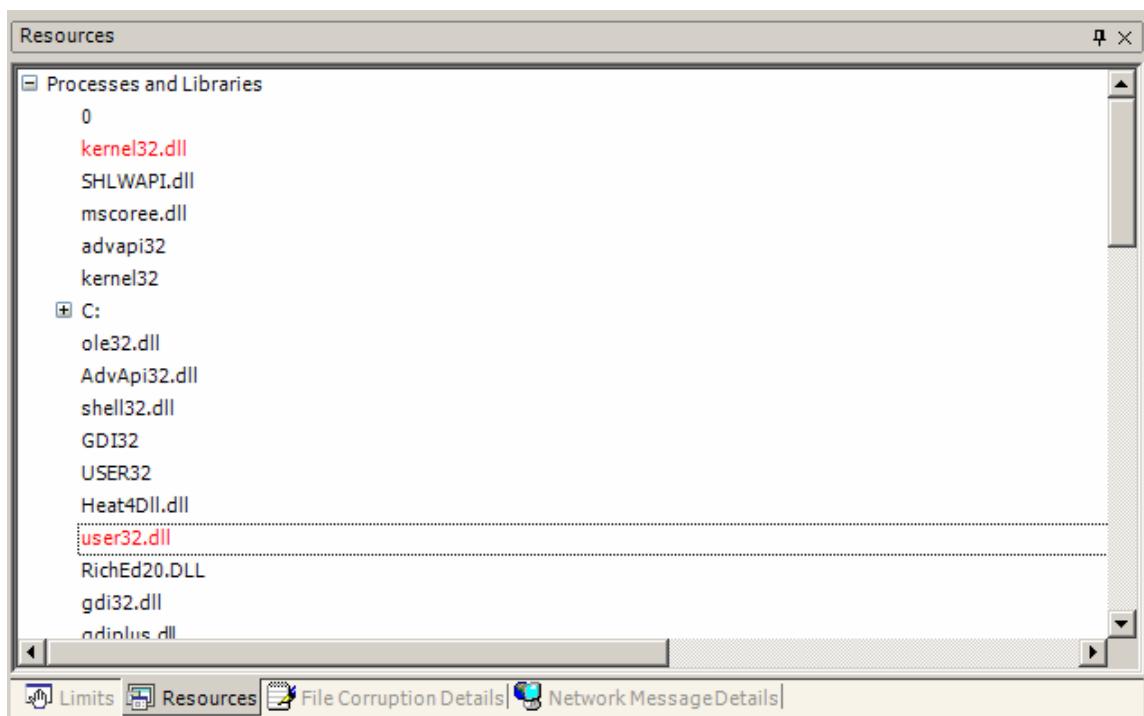
Resource Pane

Holodeck can also track external file, library, and registry resources and log every read, write, update, or access the Application Under Test makes. The resource pane shows all these resources in an easy to read tree view.

To show what resources the Application Under Test is using make sure the resources pane is open by clicking View and verifying there is a check mark by Resource Pane. From here you can view exactly which resources the Application Under Test is using. You can change which process the resources are displayed for by selecting different process nodes in the project pane.

You can set a resource fault easily by finding the resource in the treeview right clicking and selecting the desired fault. When a fault has been set for a resource the text will be displayed in red.

Show the Resource Pane by clicking the Show/Hide button on the Project toolbar or click View > "Project Pane" on the menu.



Right click anywhere in this pane for the following options:

Copy – Copy the currently selected resource and information related to it to the clipboard. Sample information follows:

Timestamp: 05/28/2004 13:21:27.531

Name: kernel32.dll

Description: N/A

Access count: 14

Last accessed by: GetModuleHandleA, which returned with value 2011430912 and error code

ERROR_FILE_NOT_FOUND

Export Resources to File... - Export the currently displayed resource list to a file. For more information see the Exporting Resource Logs help topic.

Select Most Recent Log Entry – selects the most recent occurrence of this resource in the API log pane.

Create a Fault – This will launch the Create a New Resource Fault Wizard with the currently selected resource selected.

Select a <type> fault – This menu will change depending on which type of resource you are selecting. This menu allows you to quickly set a fault without using the Wizard.

Select a Custom fault – Any custom faults you have created will be available to set in this menu.

Holodeck automatically combines the registry paths from multiple registry API calls to give you the full path, even if that was not what was used in the last API call. Holodeck also combines file paths to give you the full file path.

Jump to the last log entry that touched a resource by right clicking the resource in the resource pane and selecting "Select most recent log entry"

Export a complete list of which resources your application has accessed by clicking File > "Export Resource to File..."

For more information please see the Resource Fault Overview help topic in the Holodeck In Depth Section.

Test Pane

When you use the Scheduled Test Wizard to create a test it will show up in the test pane.

To show the Test Pane double click the Tests Node in the Project Pane.

On/Off – Turn this test on or off without interrupting other processing.

Target Thread – The thread this test will be testing on. A zero in this field means the test applies to all threads.

Function – The function to test.

Return Value – The value the test function will return.

Error Code – Upon return the function will set this as the last error

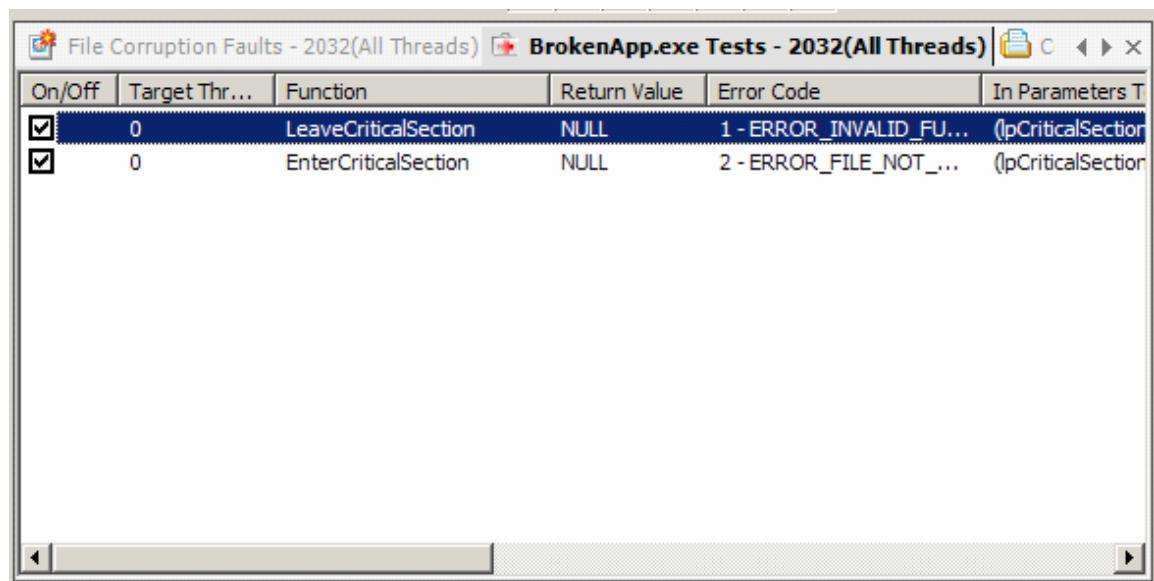
In Parameters to match – The in Parameters to match based on your specifications

Out Parameters to Change – If the in parameter specification has been met, out parameters will be changed to this.

Owner – The creator of the test

Pause – Whether or not the function pauses the application when it fires.

Fire Condition – If the test fires a percentage of the time, or based on the stack, or only when it is called from a certain other function.



The screenshot shows the Test Pane window with the title "File Corruption Faults - 2032(All Threads)" and "BrokenApp.exe Tests - 2032(All Threads)". The window contains a table with the following data:

On/Off	Target Thr...	Function	Return Value	Error Code	In Parameters T
<input checked="" type="checkbox"/>	0	LeaveCriticalSection	NULL	1 - ERROR_INVALID_FU... (lpCriticalSection)	
<input checked="" type="checkbox"/>	0	EnterCriticalSection	NULL	2 - ERROR_FILE_NOT_... (lpCriticalSection)	

Right click anywhere in this pane to display the following options:

Copy – Copy the fault details. Sample information follows:

Test On: True

Target Thread: 0

Test Function: GetModuleHandleW

Return Value: 2011430912

ErrorCode Value: 2 - ERROR_FILE_NOT_FOUND

Parameters values to match before executing test:

lpModuleName = kernel32.dll

Parameters values to change while executing test:

None

Test Owner: User

Test will pause Application Under Test after execution: No

Additional test firing options:

Test will fire 50% of the time

Modify Test... - This will open the last page of the New Scheduled Test Wizard so you can go back and edit the definition of the fault.

Delete Test – Remove this test completely

Create a new Test – Open the new Scheduled Test Wizard.

Select Visible Columns – This will allow you to modify which columns are visible in the Test Pane.

Tutorials

Beginner

Creating your first project

Introduction

In this tutorial we will cover how to create your first project and save it to the hard drive for later use. We will be using the BrokenApp that came with Holodeck to create this project so that in later Tutorials you can simply load this project without having to run through the entire Create a New Project Wizard.

Projects are Holodeck's way of keeping all your information together, they will hold all the information faults, limits, tests, etc. about your project. When you save a project Holodeck remembers all the changes you have made to that project so when you reload it the same applications will start up, with all of the faults, limits and tests you have saved.

The BrokenApp that ships with Holodeck is an application developed by the Holodeck team to show what different errors look like when discovered with Holodeck. BrokenApp contains a number of bugs, including memory leaks, calls to dangerous functions, crashing errors, and other errors that make it unstable under hostile conditions. We will be using this application in the tutorials to ensure bugs and errors are obvious and easy to find.

Saving the project will save each part of the project including API logs, Network logs, Corrupted Files, Exceptions, Recorded Sessions, Faults, Resource Faults, Limits, Network Corruption Faults, File Corruption Faults, and Scheduled Tests. All faults, limits, corruption faults, and tests will be automatically applied to the new instance of the application. If any tests or faults were created in per-thread mode those tests and faults will be applied to the entire process next time the project is loaded.

[Next >>](#)

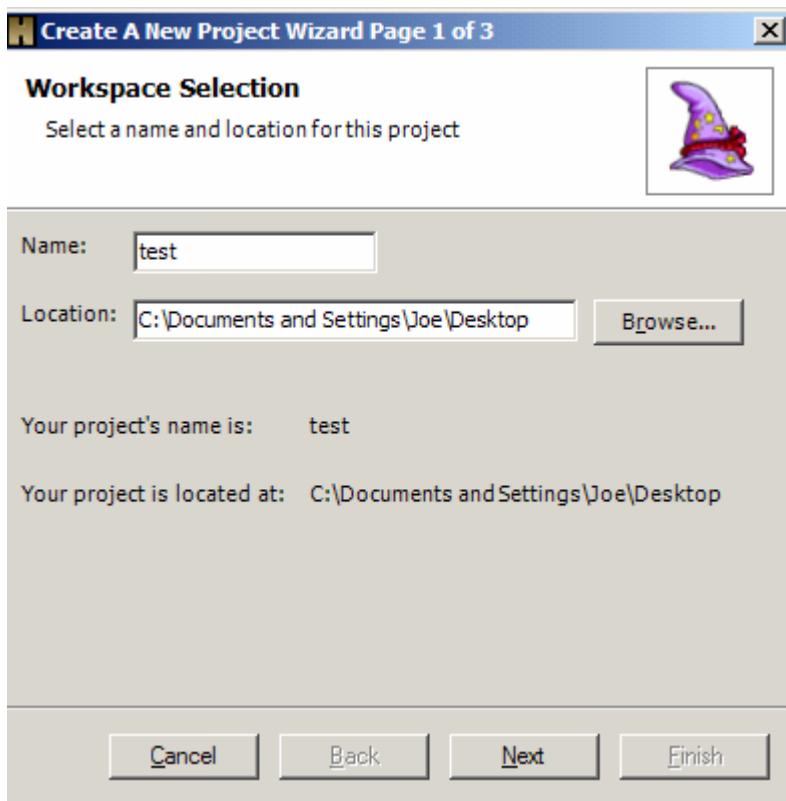
Creating the Project

The first page of the Create a New Project Wizard asks for a location to store your project or workspace. This can be anywhere on your hard drive or a network share. You must have complete read/write access to this location.

To open the New Project Wizard click File > "New Project ..."

Workspace Selection

The first page of the New Project Wizard lets you specify the name of your project and where you want to save it. In this case I have chosen to save the project on the Desktop, and name it test. You can browse to any location on your computer, removable media or a network share using the "Browse..." button. Click next when you have finished saving your project and have a location to save it to.

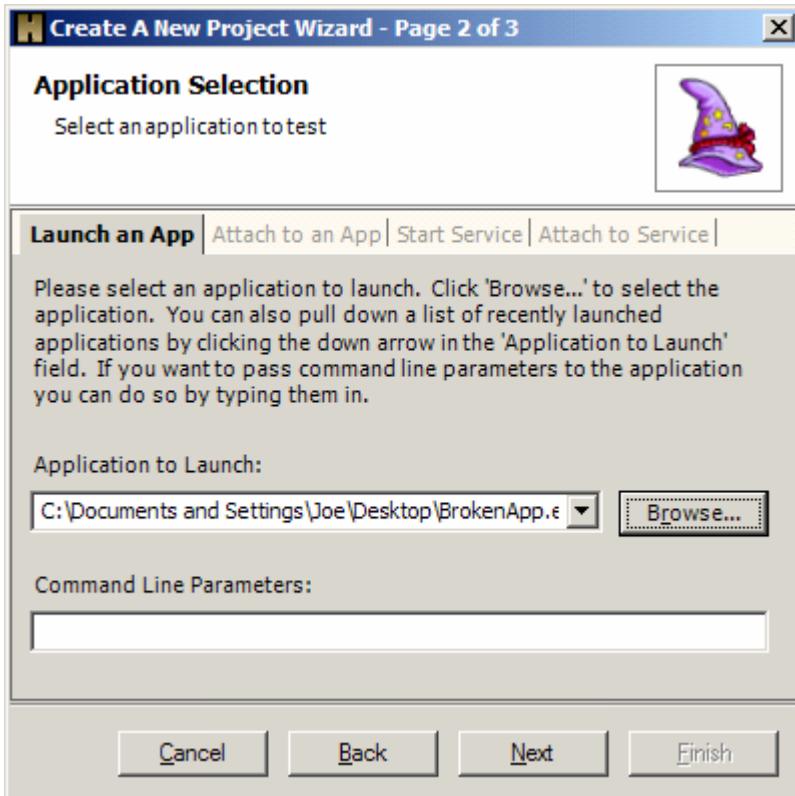


Application Selection

The second page of the Create a New Project Wizard is asking which application or service you would like to create this project with. Holodeck supports Launching a new application, attaching to an already running application, Starting a Service, or attaching to an already running service. For this project we want to launch the BrokenApp.

BrokenApp has been installed to the <HolodeckInstallDirectory>\BrokenApp\BrokenApp.exe
browse to this location to specify the BrokenApp for Holodeck to launch.

BrokenApp does not require any command line parameters so we can leave this text box blank.



Options Selection

The BrokenApp is written in Managed C++ code compiled as a .NET application. Holodeck will automatically recognize BrokenApp as a .NET application and will check the default logging for a .NET application however since this application also uses many Native Win32 API calls we will need to intercept these as well. For more information see the Default Logging help topic.

The Native functions that we are most interested in are as follows:

Category	Reason to Intercept
COM	COM resources will be available for fault testing in the Resource Pane.
File Functions	Files will show up in the Resource Pane and Corrupted files will show up in the Corrupted Files Pane.

Library Functions	Library (dll) resources will be available for fault testing in the Resource Pane.
Network Functions	Network Logs will be enabled.
Process Functions	Process resources will be available for fault testing in the Resource Pane.
Registry Functions	Registry resources will be available for fault testing in the Resource Pane.

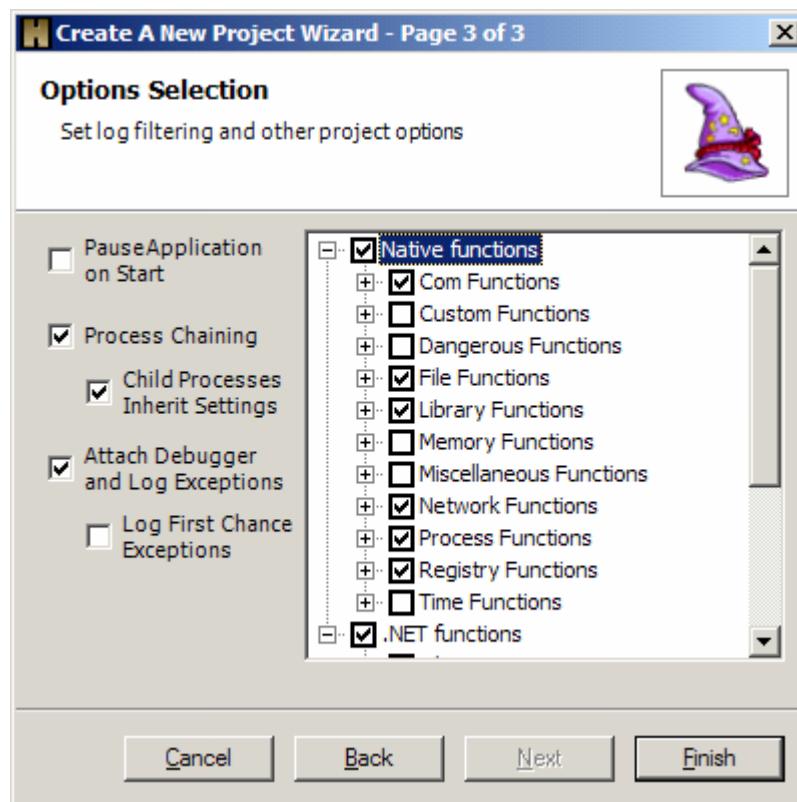
On this page we can specify to pause the application on start so we can set tests and faults before the application starts up. This is useful for testing the startup functions of your application.

Process Chaining sets up Holodeck to automatically attach to any processes your application spawns. This is extremely useful if your application uses any helper applications to do some of its processing.

Child Processes Inherit Settings allows the new process your application spawned to automatically inherit the same tests, limits, faults etc from the main application.

Holodeck can act as a debugger, log the exceptions your application creates and create mini-dump files for debugging the application by enabling the Attach Debugger and Log Exceptions option.

By default Holodeck does not log First Chance Exceptions as often they are quickly handled by the application under test, and does not create errors. Enable First Chance Exception logging if you would like to see each time your application creates a first chance exception



<< Previous Next >>

Saving the Project

Since this project will be used for the later tutorials we would like to save it so we don't have to setup the project each time. Instead we can load a project with all the right logging and pointed to the correct application. This can help save time if you are testing the same application daily. This is also necessary if you are running Holodeck from the command-line.

Since you specified a place for Holodeck to save all related project files, clicking File > Save Project will simply save the latest version of the project to that location.

If you would like to save the project to another location click File > Save Project As... This will bring up a save file dialog for you to specify a new location to save the project.

Saving the project will save each part of the project including API logs, Network logs, Corrupted Files, Exceptions, Recorded Sessions, Faults, Resource Faults, Limits, Network Corruption Faults, File Corruption Faults, and Scheduled Tests. All faults, limits, corruption faults, and tests will be automatically applied to the new instance of the application. If any tests or faults or tests were created for a specific thread those tests and faults will be applied to the entire application on the main process.

<< Previous

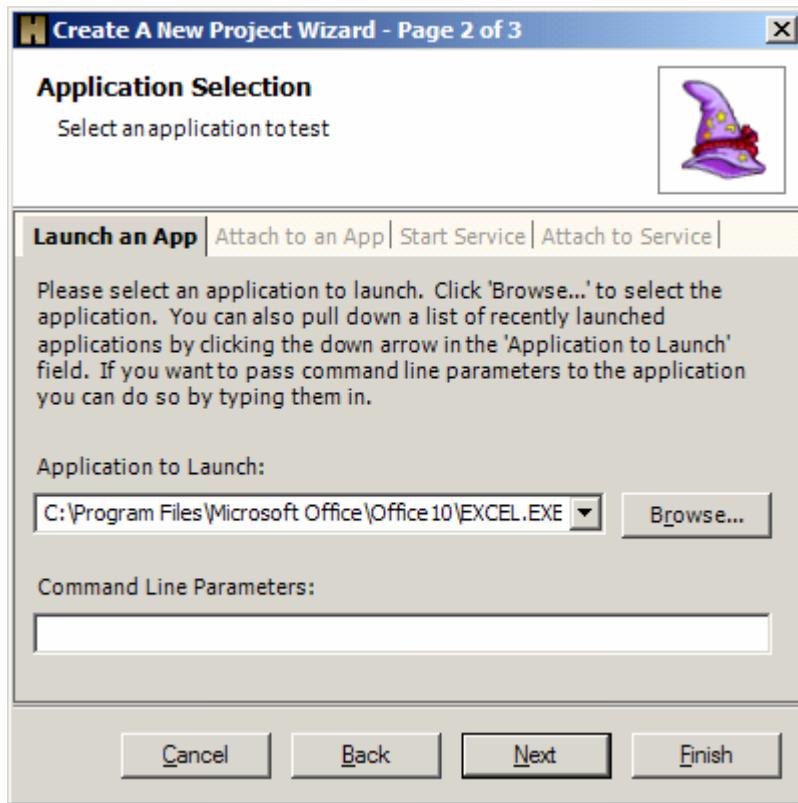
How to Use Faults to Deprive Excel of Memory

Introduction

This tutorial will cover how to deprive an application of memory by setting the Insufficient Memory fault. The Insufficient Memory fault fails any calls to API functions that try to allocate additional physical memory or virtual memory. Holodeck will not fail functions that de-allocate memory or re-allocate the same amount of memory.

To start this project create a new project using Microsoft Excel and default logging. Excel's default location is "C:\Program Files\Microsoft Office\Office10\EXCEL.EXE."

For more information on how to create a project please see the Setting up your first project tutorial.

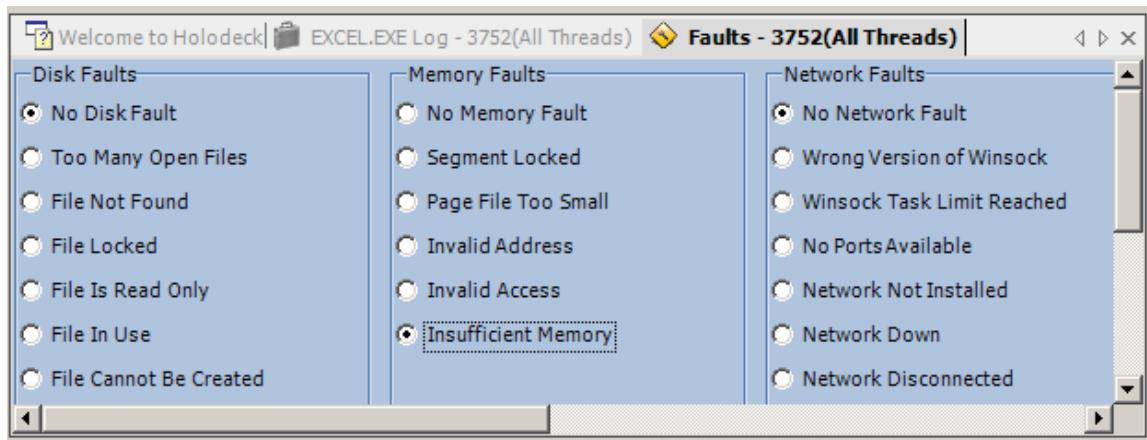


Next >>

Set Insufficient Memory

Once Excel has finished loading, navigate back to Holodeck. The faults pane should be visible from the default workspace of Holodeck. If the faults pane is not visible double click the faults node in the Project Pane.

In the Faults Pane select "Insufficient Memory" under the memory faults heading. In the Insufficient Memory fault calls to functions that allocate more memory are failed, this also includes functions that reallocate a greater amount of memory than was previously specified. For more information on Memory Faults see the help topic.



<< Previous Next >>

Verify Insufficient Memory

The Insufficient Memory fault is set as soon as the radio box is checked. To see how Excel does when it runs out of memory, navigate back to Excel and try to complete any task. Excel has completely frozen up; in fact Excel can't even allocate the memory necessary to redraw the window. This is what we would expect from an application that has no more memory resources whatsoever available. However, as soon as the memory fault is lifted, Excel returns to completely normal execution.

Delete the memory fault:

Navigate back to the faults pane in Holodeck and select "No Memory Fault"

Further Exploration:

Try this tutorial using BrokenApp that has shipped with Holodeck. When the Insufficient Memory fault is set BrokenApp quickly fails. You can investigate the failure through the first chance exception it produces. For more information on first chance exceptions and debugging using this information see the Exceptions and Mini-dump section or the resource faults tutorial.

<< Previous Next >>

Summary

Using the Insufficient Memory fault in Holodeck you can simulate an insufficient memory state within the Operating System exactly as if the fault actually happened.

Further Exploration

Try clicking on a number of buttons on the toolbar in Excel with the Insufficient Memory fault set. Excel will eventually crash under these conditions. Test Excel to see what functionality causes crashes when running in low memory and which doesn't. Is there any other odd behavior you can find in excel while this fault is enabled. Try enabling other memory faults to test Excel's memory management. Try enabling other categories of faults to test how Excel works with other hostile environments.

[**<< Previous**](#)

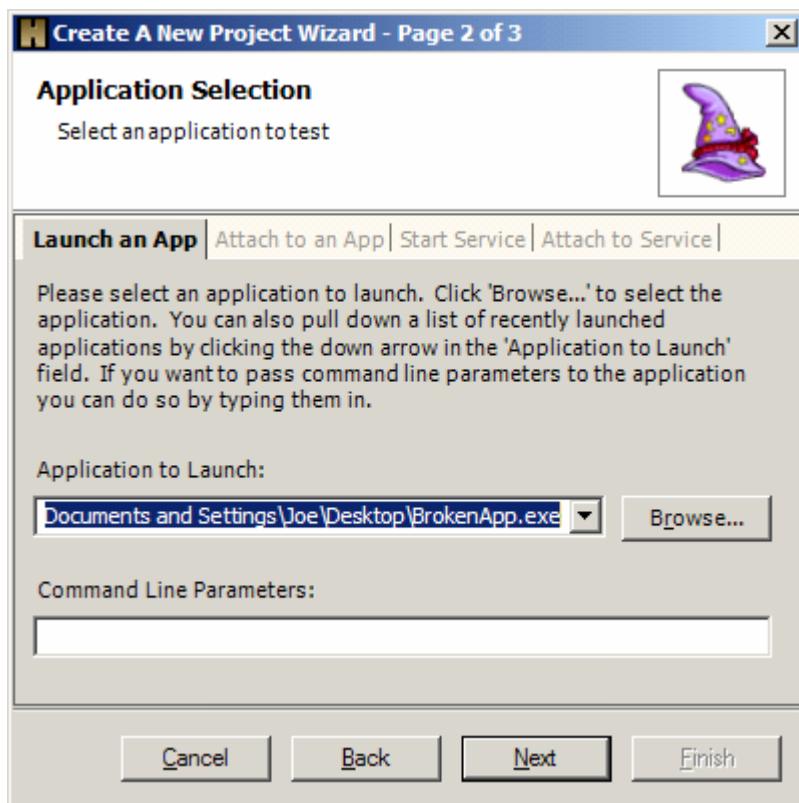
How to Deprive Your Application of its Dependencies

Introduction

In this Tutorial we will be exploring a method to discover which resources your application uses, and how to deny access to those resources to find new bugs. During this process we will uncover the resources that the BrokenApp uses while it loads, then set a resource fault so that it can no longer access it, then restart the app to find out what happens.

Create a new Project

To set up this Tutorial create a new project using BrokenApp as the Application Under Test, with default logging plus all the native API functions turned on. For more information on how to do this please see the Setting up a New Holodeck Project Tutorial.

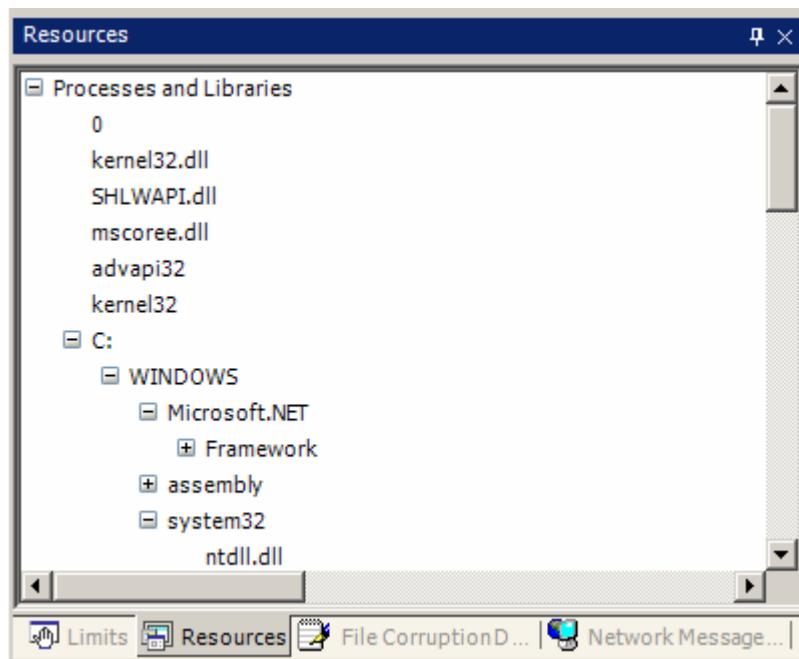


Next >>

Investigate Resources

Once the application has started up, navigate to the resource pane by clicking the tab in the lower pane that says "Resources." Holodeck will show all the resources that BrokenApp has touched on startup. More resources might show up here as the application is used, but we can find a pretty nice bug without having to do that.

If you drill down into the processes and libraries resource node you will find a dll called RichEd20.dll. This dll is responsible for all the rich text editing that BrokenApp is able to do. We want to see what happens when BrokenApp is unable to load this dll when it is starting up.

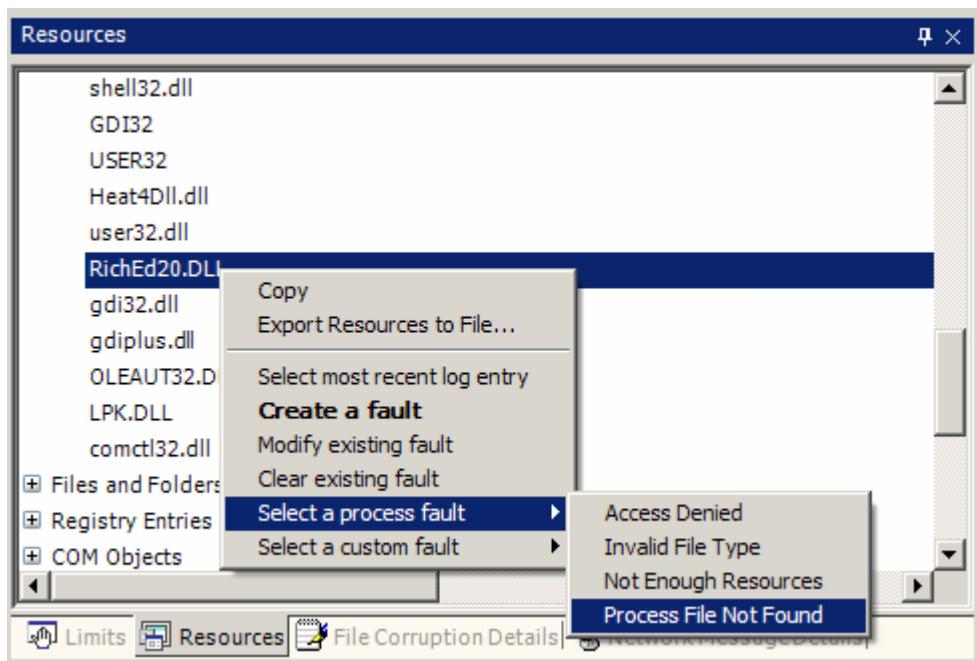


<< Previous Next >>

Set the Resource Fault

Now that we have done the research we need to do, the next step is to set a resource fault and restart the application. Often the application loads the entire library into memory, and doesn't access it after that; this is why we need to restart the application.

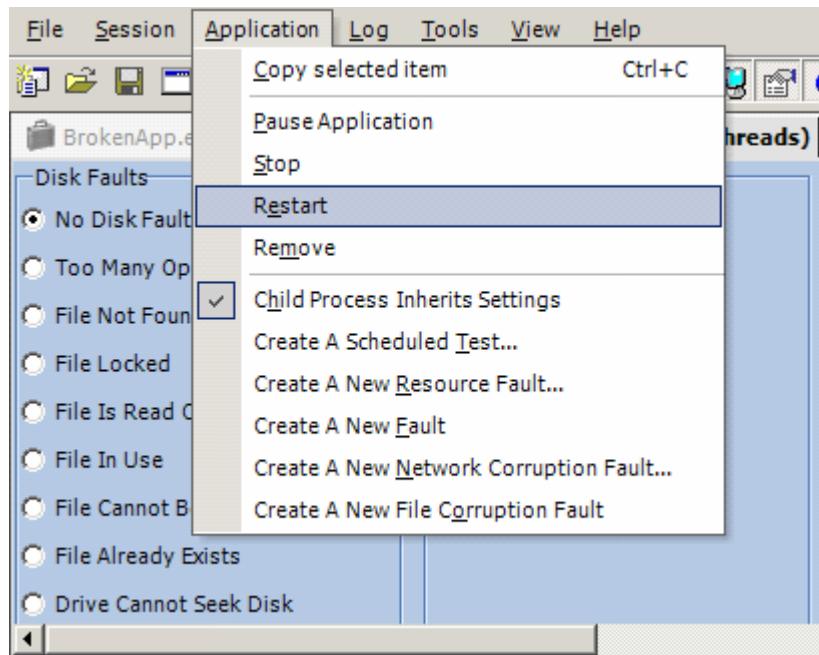
To set a resource fault, find the resource in the resource pane and right click it. If you select "Create a fault" Holodeck will launch the "Create a Fault Wizard" with the correct resource automatically checked. However Holodeck allows you to set a fault quickly by using the "Set a Process Fault" menu. After you select one of the faults in this menu that fault is set immediately. Set the "Process file not found" fault on RichEd20.dll. This fault will fail any attempts to load a library or create a process which is what happens when the BrokenApp tries to load this dll.



<< Previous Next >>

Restart BrokenApp with Resource Faults

Since this dll is loaded into memory when BrokenApp starts up we have to restart the application to see any changes. Restart BrokenApp by selecting Application > Restart. When you restart the BrokenApp Holodeck will remember any Resource Faults you have previously set and apply them to the application as it starts up.



As BrokenApp starts up it will fail as soon as it tries to load the RichEd20.dll library. Two exceptions will be thrown, the first chance exception is caught by Holodeck before the application has a chance to deal with it, first chance exceptions will be caught in any application every time there is an exception thrown. Holodeck immediately gives the exception back to the application so it can handle it. If the application does not handle it, which broken app does not, a second chance exception is thrown. At this point since BrokenApp does not handle the exception it crashes.

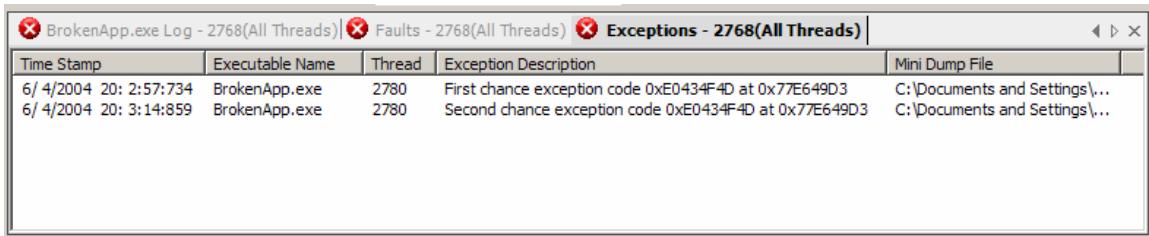
Faults - 3240(All Threads)				Exceptions - 3240(All Threads)			
Time Stamp	Executable Name	Thread	Exception Description	Time Stamp	Executable Name	Thread	Exception Description
5/24/2004 13:3:19:46	BrokenApp.exe	3524	First chance exception code 0xE0434F4D at 0x77E649D3				
5/24/2004 13:3:20:390	BrokenApp.exe	3524	Second chance exception code 0xE0434F4D at 0x77E649D3				

For more information see the Exceptions Overview help topic.

<< Previous Next >>

Investigating the Failure

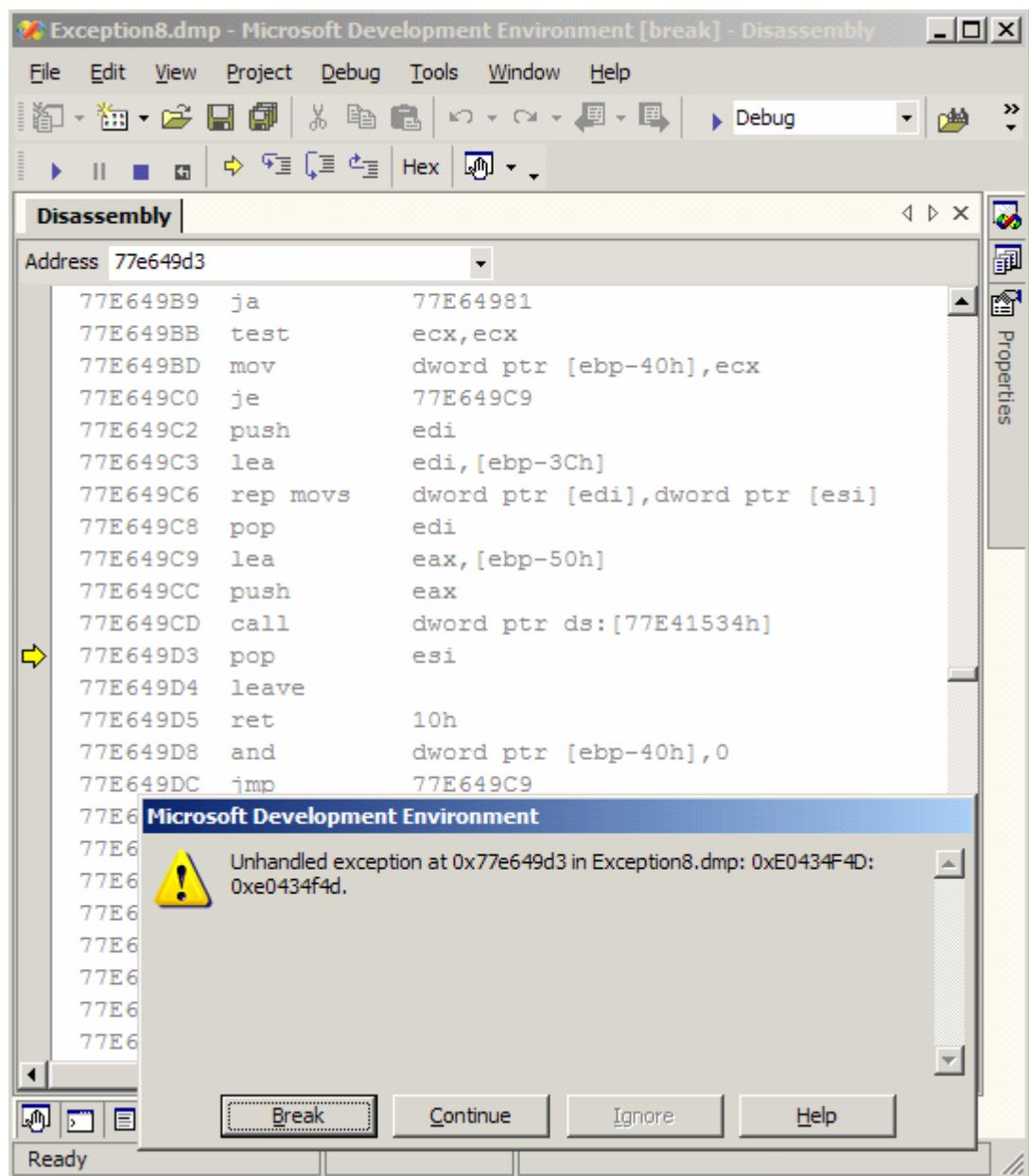
We can load the debugging information into Visual Studio.NET and reproduce the crash. To load the debugging information into Visual Studio double-click one of the exception nodes in the Project pane. This will show the Exception pane in the main window. The Exception pane shows every exception your application has thrown, whether it handled it or not. For more information see the Exceptions Overview.



Time Stamp	Executable Name	Thread	Exception Description	Mini Dump File
6/4/2004 20:2:57:734	BrokenApp.exe	2780	First chance exception code 0xE0434F4D at 0x77E649D3	C:\Documents and Settings\...
6/4/2004 20:3:14:859	BrokenApp.exe	2780	Second chance exception code 0xE0434F4D at 0x77E649D3	C:\Documents and Settings\...

Double-click the exception you want to debug in this pane and Holodeck will launch Visual Studio and load the exception debugging information. You can reproduce the crash or exception by pressing f5. If source code is available for the crash Visual Studio will load it and highlight the offending line. If no source code is available Visual Studio will ask to show the assembly code, and highlight the last instruction to be called.

For more information see the Exceptions Overview help topic.



<< Previous Next >>

Summary

Holodeck can easily test your applications ability to fail gracefully when resources are unavailable. You can quickly set resource faults from the resource pane by right clicking the resource and selecting the fault you would like to set on that specific resource.

Further Exploration:

Try failing access to a registry key that BrokenApp relies on. Go to the resource pane, research the registry keys BrokenApp touches while starting up, and fail access to them. Some Registry keys might cause BrokenApp to crash, others may only disable certain functionality within the application.

[**<< Previous**](#)

Dealing with Multiple Processes

Introduction

In this tutorial we will explore how to create a Holodeck project that will test multiple applications simultaneously. Doing so will allow you to set faults, tests, and limits on a per application basis. You will also be able to create reports and logs based on the entire project; gathering data from both applications.

To create a project that will eventually be used for multiple test applications simply choose one application first and we will add the second one later, using the New Test Application... functionality. In this example we will use notepad.exe and BrokenApp.exe as the two applications we will test.

Create a new project using BrokenApp.exe with full .NET logging and default win32 logging turned on, also enable process chaining on the last page of the New Project Wizard.

Once the BrokenApp has started up you should see the faults pane being displayed to set faults for the BrokenApp. Holodeck has already started intercepting and logging API functions. To see the API functions Holodeck has logged double click API logs in the Project Pane.

When BrokenApp has finished loading Add Another Process to the Project.

[Next >>](#)

Adding a second process to a project

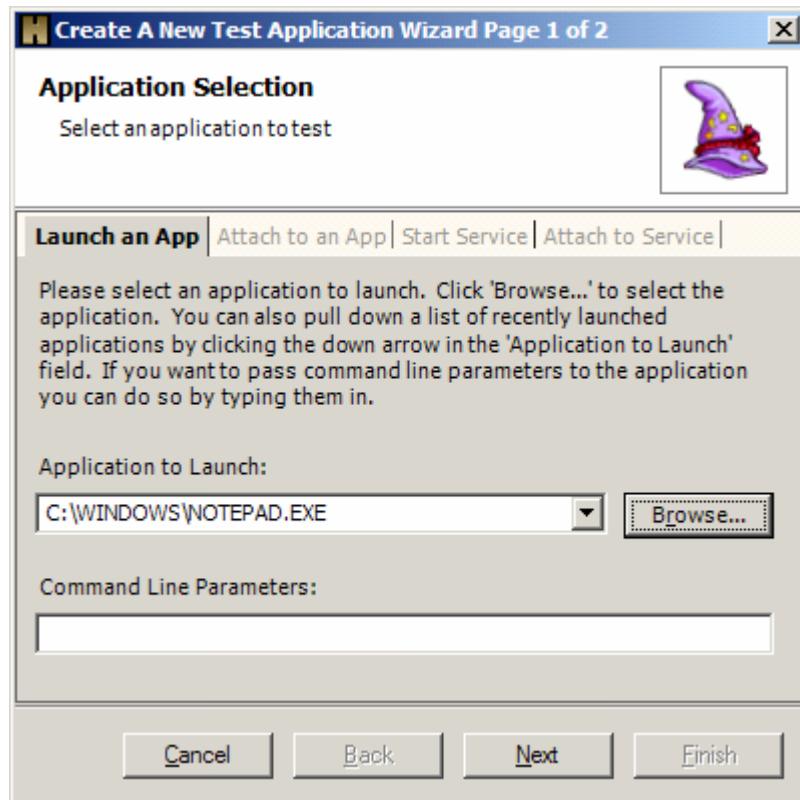
To add a second application to your project click File > "New Test Application... "

Application Selection

The Application Selection window allows you to add another application or service to the project. From this page you can launch an application, attach Holodeck to an already running application, start a service, or attach Holodeck to a service that is currently running.

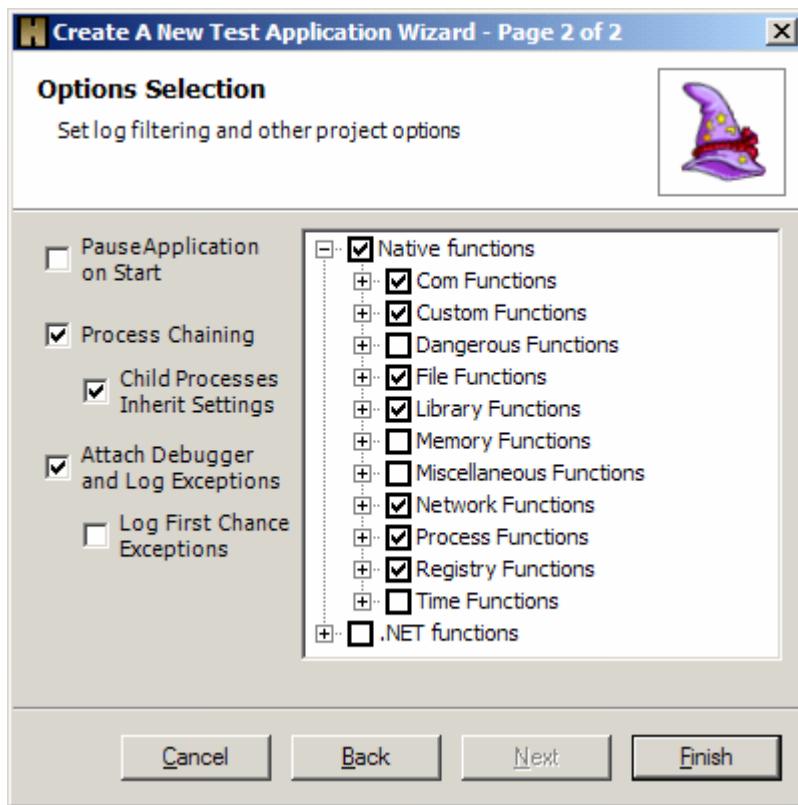
In this case we want to load the other application which is not currently running so we select Launch an App from the tab at the top (should already be selected by default) and browse to the location of notepad. Notepad is located in the "C:\WINDOWS" directory.

Notepad is a windows application and does not take any command line parameters so we can leave this text box blank



Options Selection

This page is identical to the last page of the New Project Wizard, the New Project Options page. For this case we want to leave all logging at the default settings. For more information on Default Logging please see the help topic.



This will create a new application parent node titled "Notepad.exe" in the Project pane. From here you can create Faults, Limits Corruption, and tests just for that application. In the next topic we will cover how to create these items at a per process level.

<< Previous Next >>

Setting per process items

In this topic we will create a Fault, Limit, Network corruption, and a test at a per process level.

Create a File in Use fault for Notepad.exe – Cause notepad to be unable to access any file.

Create a Memory Fault for BrokenApp.exe – Disallow BrokenApp to load, access, or allocate memory.

Create a Memory Limit for Notepad.exe – Limit Notepad to only 10 more bytes of memory than it is currently using.

<< Previous Next >>

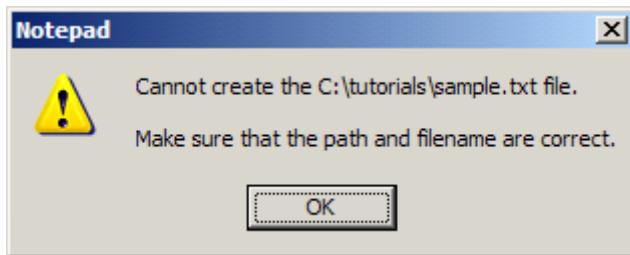
Create a File in Use fault for Notepad.exe

Before beginning this section, save the current text file as "sample.txt"

Once the file has been saved continue with these steps.

- 1) Select Notepad.exe > Faults > Disk in the Project Pane.
- 2) Select "File in Use" in the Faults Pane.
- 3) Verify failure

In notepad click File > Save, notepad will attempt to save the file to the same location as it had originally saved it. Since the "File in Use" fault is set, notepad gets an error and attempts to recover by asking you where else you would like to save the file, and give this error message.



After Clicking OK Notepad sluggishly shows the save dialog box. However if you try to save the file as something else, sample2.txt in this case, notepad fails to create the file again, as it is getting a file in use error for a file that has not yet been created.

- 4) Delete the fault
 - a) Without closing Notepad return to Holodeck
 - b) Delete the fault
 - c) Right click the fault and select Delete Fault
- 5) Notepad will return to normal execution once the fault has been lifted

Note: BrokenApp will continue to function normally; no fault has been set for this application.

<< Previous Next >>

Create a Memory Fault for BrokenApp.exe

Invalid Address faults can happen when a memory address has been stored incorrectly or has been changed by corrupt RAM.

- 1) Select BrokenApp.exe > Faults > Memory in the Project Pane.
- 2) In the Faults Pane select "Invalid Address."
- 3) Verify Failure:

In BrokenApp type a few words, then apply bold to the text by clicking Format > Bold on the menu. BrokenApp quickly crashes without throwing an error.

Investigating the error

We can investigate BrokenApp's failure many ways, but using the built in reporting feature of Holodeck is the easiest. To create a report right click the Reports node in the Project Pane and select "Create a Report." Holodeck will generate a report on all the processes currently running in this project in an easy to read file, for more information on this feature please see the Reports section of the help documents.

Amongst other things, the API failures of the BrokenApp process are recorded in the "Error Codes Summary Table." To research this failure drill down into the Error Codes of type "Fail" by clicking the plus [+] sign next to fail. This will show the error codes type "Fail" broken into the categories by Error Code. We want to know what functions failed right before the crash, so expand the "Total" Error Code, which was compiled by Holodeck to show all the error codes in chronological order. Click the plus [+] sign next to the Total row under the BrokenApp column to see all the errors that BrokenApp created.

Scroll all the way to the bottom of this list of error codes, to see the final calls BrokenApp made. Here we can see calls to memory allocation functions in the System.IO.MemoryStream class. Since we know the error happened after typing some text and applying the bold format to it, the developer could begin by investigating the source code around the function relevant to these operation and looking for any place he/she is allocating memory.

<< Previous Next >>

Create a Memory Limit for Notepad.exe

Memory Limits simulate low memory conditions in the operating system. This is a good test to do to find out how your application runs when very low on system resources. For more information on Limits in Holodeck please see the Limits overview help topic.

To create a memory limit in notepad.exe double click Notepad.exe > Limits, this will bring the limits pane into view.

In the Memory Space limit text box type in a number 10 KiloBytes larger than what Notepad is currently using, if notepad tries to allocate more than 10 KiloBytes of new memory Holodeck will fail those allocations as if the system is out of memory.

Verify failure

- 1) Switch focus to Notepad.exe
- 2) Begin typing, as soon as the memory buffer has been reached Notepad can no longer allocate memory resources and will not allow you to type any more. However, you can delete the text and type exactly the same amount; Notepad is deallocating and allocating memory properly.

Delete the limit

- 1) Right click the limit in the Project Pane.
- 2) Select delete

Return to notepad, notice the title bar has returned, and you can continue to type in the main window.

Further Exploration

Try the same memory limit using the BrokenApp. Follow the same steps, but replace notepad.exe with BrokenApp.exe. Since BrokenApp has many memory allocation errors it quickly fails.

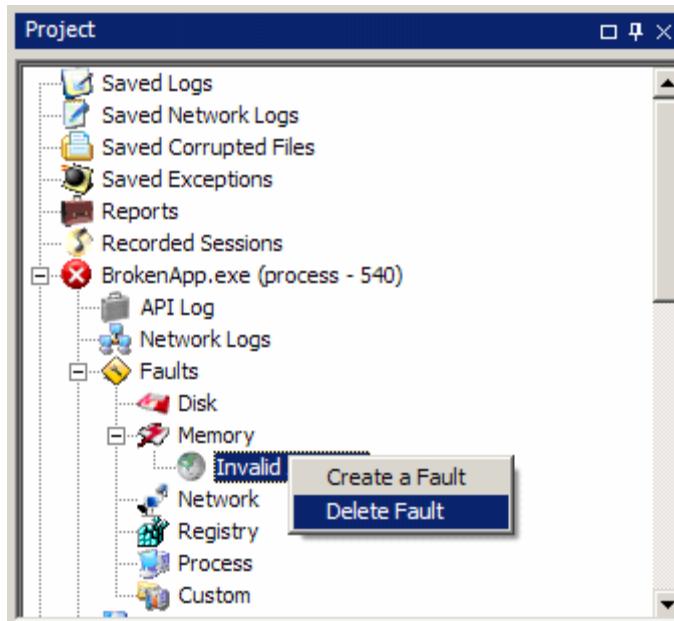
<< Previous Next >>

Spawning Another Process from the BrokenApp

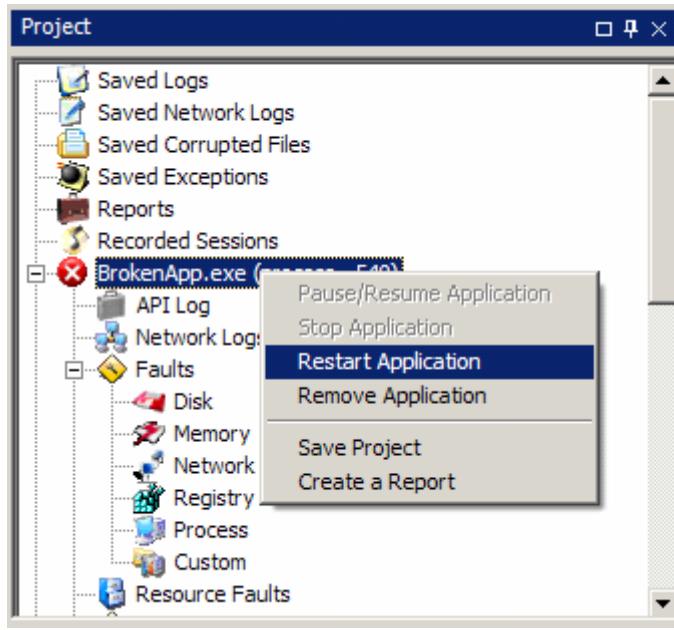
Restart BrokenApp:

Since the BrokenApp crashed after setting the memory fault, we will have to restart the application. Before restarting the application you must remove the memory fault, or it will be applied to the restarted application, immediately crashing the BrokenApp.

To remove the fault right click the fault and select "Delete Fault"



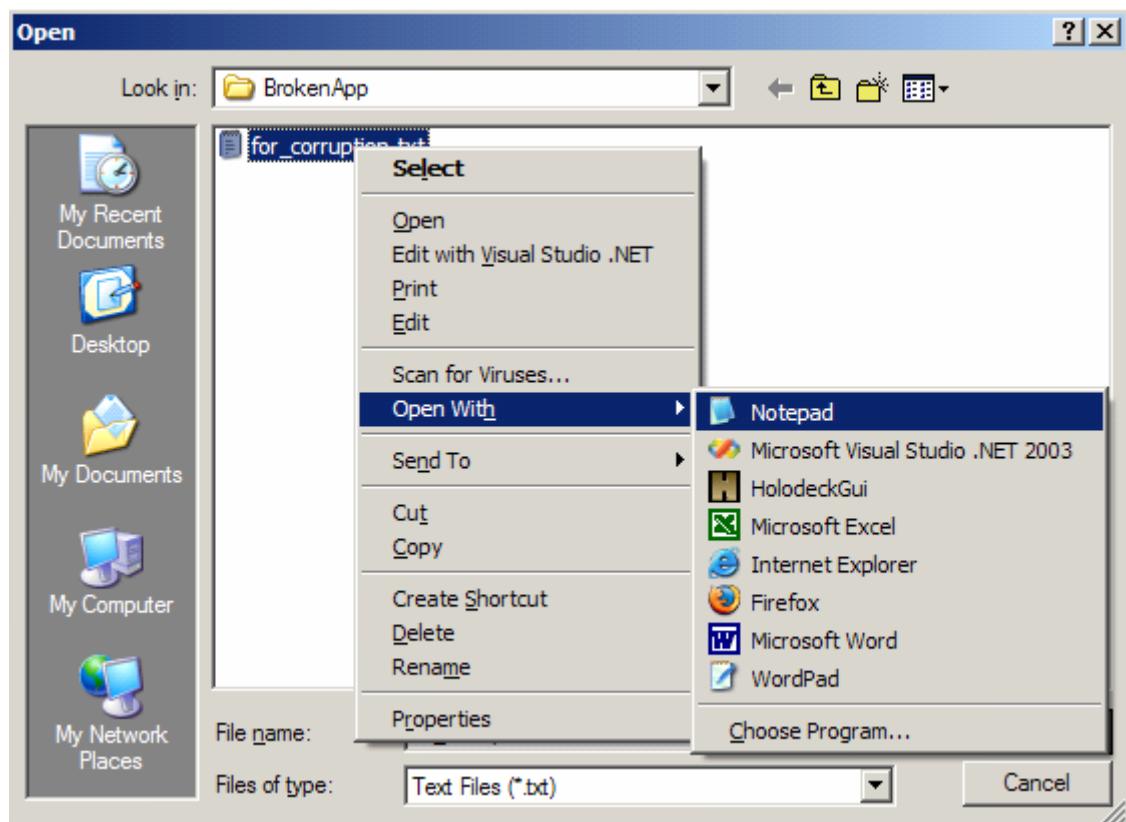
To restart the application right click the BrokenApp and select Restart Application.



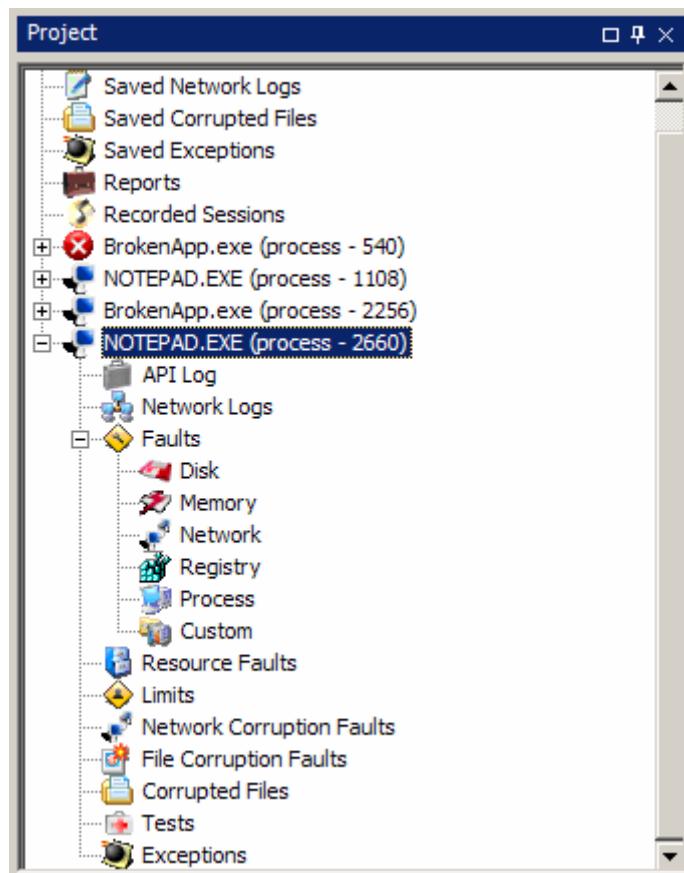
We will use the open file dialog box to spawn another notepad.exe process.

Launch Notepad from the Open Dialog:

Launch the Open File Dialog box from the restarted BrokenApp. Click File > Open ... Once the Open File Dialog is open right click any text file and select "Open With" > Notepad



This will launch Notepad from the BrokenApp's Open File Dialog and Holodeck will intercept this process and load it into the current project. This application is now available to test like any other application in the project.



<< Previous Next >>

Summary

Holodeck makes it easy to test multiple processes, and focus your tests to a single process. You can use Holodeck's Process Chaining to automatically attach Holodeck to a new process spawned by your application. Turn on "Child Processes Inherit Settings" to have any process your application spawns automatically inherit any hostile environment settings already set by the parent process.

Further Exploration

Try setting a memory fault for the parent application before it spawns a new process. If "Child Process Inherits settings" is turned on the startup code of the child application will have the fault already set. Try setting a file corruption fault for BrokenApp then load that file using Notepad, notice the fault is process specific.

[**<< Previous**](#)

Intermediate

Using Holodeck to Corrupt Files and Test File Processing

Introduction

This tutorial will cover how to corrupt a text file and test file processing using the Broken Application that was supplied with your Holodeck install. Holodeck's file corruption faults can be a very powerful tool to test file processing because it corrupts the file on a bitwise level. This simulates actual corruption that may occur because of corrupted RAM, Hard Drive sectors, or a bad network transfer.

The sections of this tutorial include:

Corrupting Files – Create a text document to be corrupted and set a File Corruption Fault for that file.

Using Corrupted Files to test file processing – Open the corrupted file using BrokenApp

Viewing the details of the corrupted file – View the actual file corruption through the File Corruption Details Pane.

Create a new project using BrokenApp.exe with full .NET logging and default win32 logging turned on, or load the project you saved using BrokenApp.exe from the first tutorial.

[Next >>](#)

Corrupting Files

Before we create a file corruption fault we need to create the file we want to test with. This is useful because it allows you to corrupt a file differently each time the application tries to access it to test all aspects of file processing.

Create a file to test:

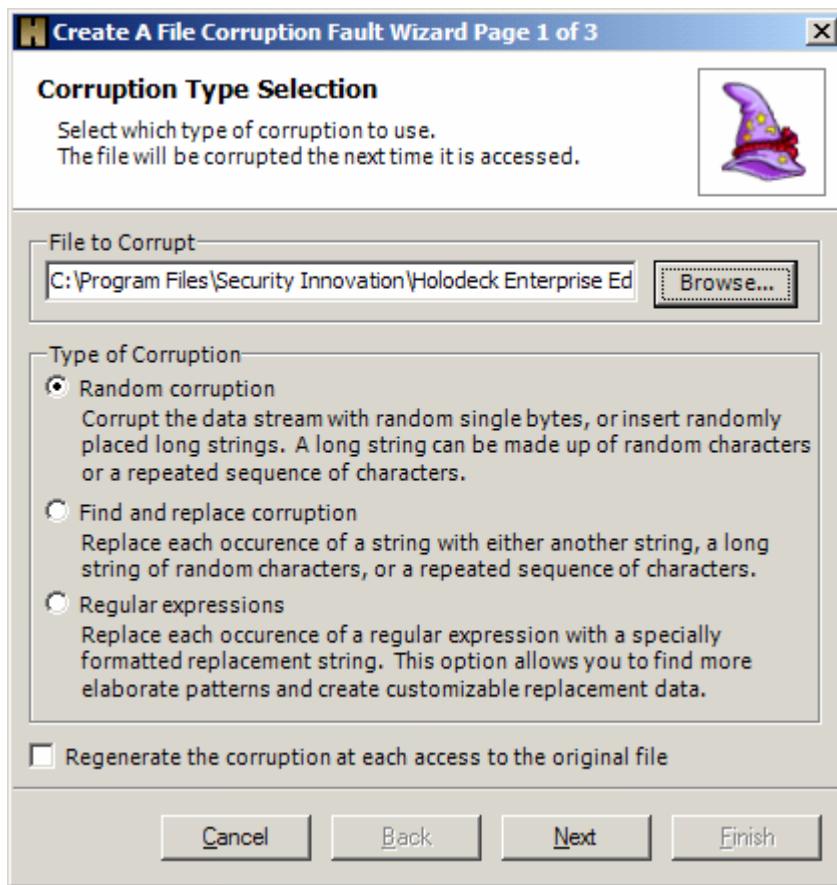
Create a text file with the following text, save it as for_corruption.txt to your desktop. We will use this file in corruption, because it is well formed and any corruption will be easy to see.

```
This file has been created to show off the file corruption of  
Holodeck.  
It should be exactly 73 characters long and 13 lines  
down.xxxxxxxxxxxxxx  
This file has been created to show off the file corruption of  
Holodeck.  
This file has been created to show off the file corruption of  
Holodeck.  
This file has been created to show off the file corruption of  
Holodeck.  
This file has been created to show off the file corruption of  
Holodeck.  
This file has been created to show off the file corruption of  
Holodeck.  
This file has been created to show off the file corruption of  
Holodeck.  
This file has been created to show off the file corruption of  
Holodeck.  
This file has been created to show off the file corruption of  
Holodeck.  
This file has been created to show off the file corruption of  
Holodeck.  
This file has been created to show off the file corruption of  
Holodeck.  
This file has been created to show off the file corruption of  
Holodeck.  
This file has been created to show off the file corruption of  
Holodeck.  
This file has been created to show off the file corruption of  
Holodeck.  
This file has been created to show off the file corruption of  
Holodeck.
```

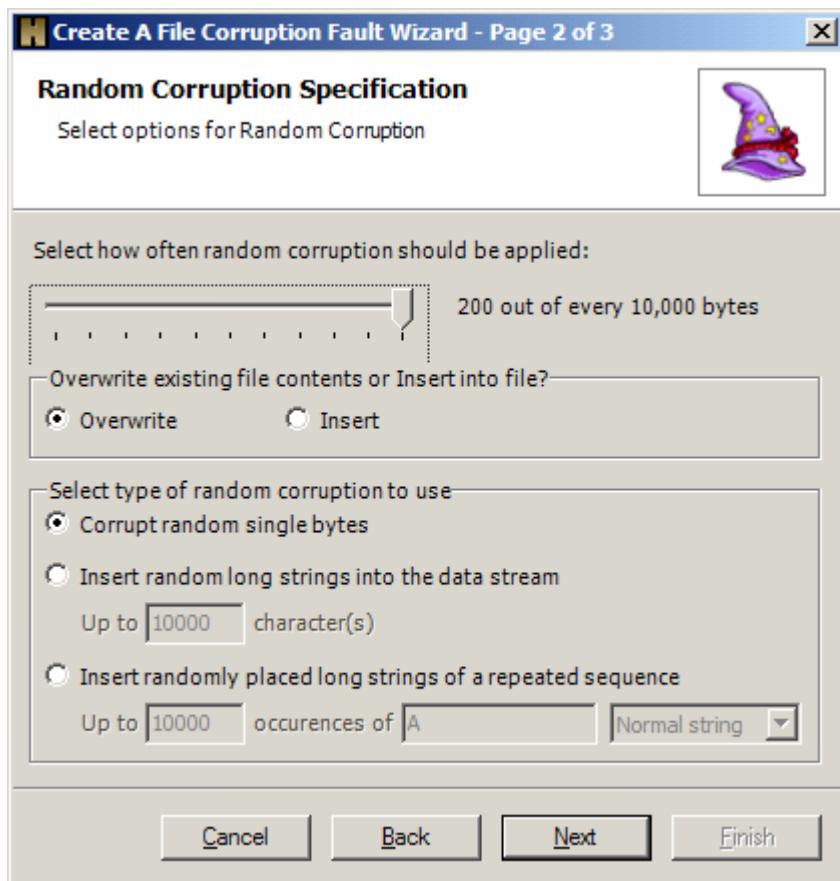
Create a New File Corruption Fault:

To create a file corruption fault click Application -> "Create a New File Corruption Fault..." This will bring up the file corruption wizard.

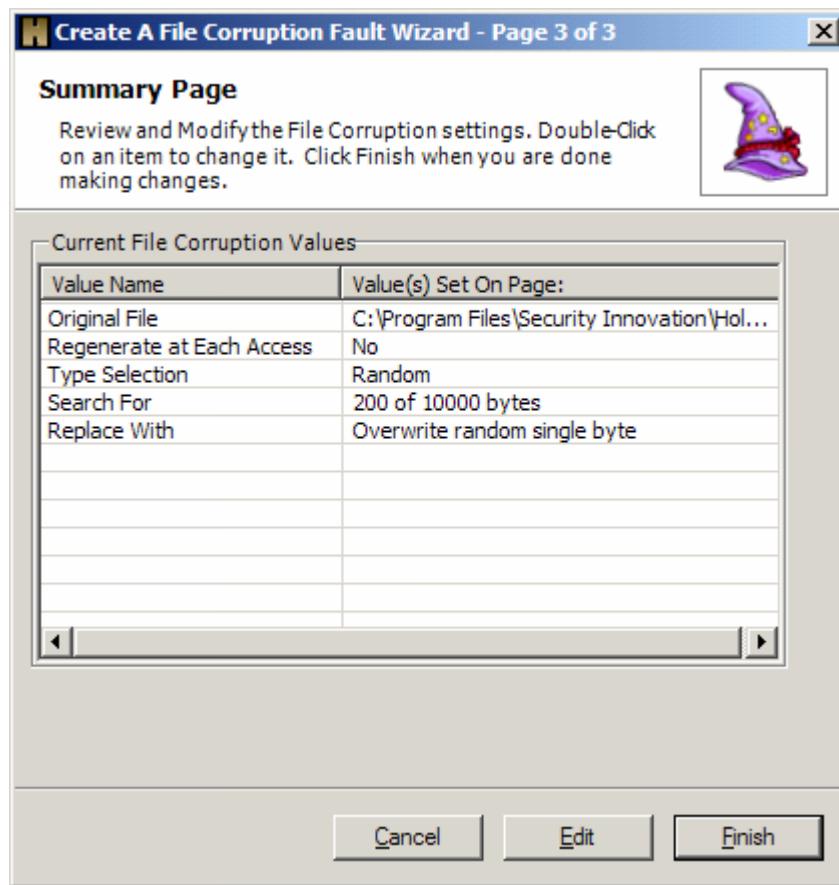
In this tutorial we want to create random corruption throughout this file. Holodeck's File corruption will select random bits to corrupt in random lengths throughout the file. This can generate anywhere from only slightly corrupted data, to extremely corrupted data.



For our purposes we want the file to be as corrupted as possible so the corruption will find any easy to find file reading bugs. Slide the corruption amount slider all the way to the right until it displays 200 out of every 10,000 bytes to be corrupted. Leave Overwrite and Corrupt random single bytes at their default values.



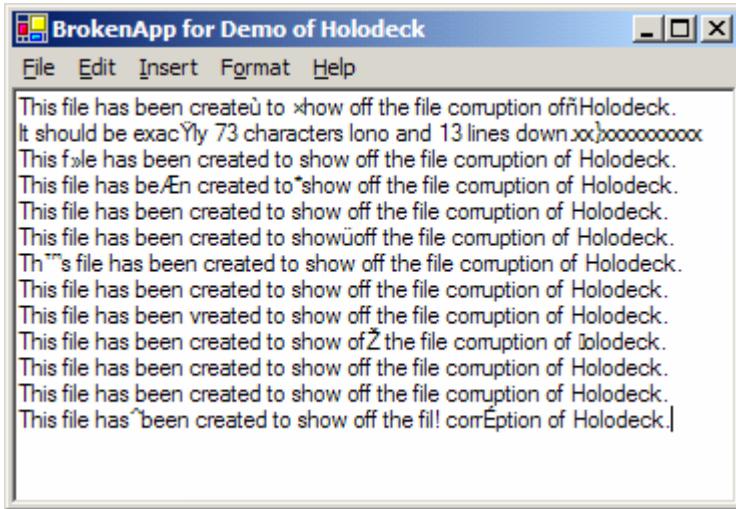
The final page shows a summary of the file corruption that you have selected. To return to any page, simply highlight the information you would like to change, and click edit; doing this will return you to the page of the wizard which contains that information.



<< Previous Next >>

Using Corrupted Files to test file processing

BrokenApp has been written to display any file corruption exactly as it exists in the file. This allows us to see exactly how Holodeck corrupted the file. Open the file for which you have just create a file corruption fault in BrokenApp by clicking File > Open on the menu bar. The following file corruption occurred on my computer, you can see many serious problems with the file. To find out exactly how the file was corrupted see the File Corruption Details Pane or continue to the next topic.

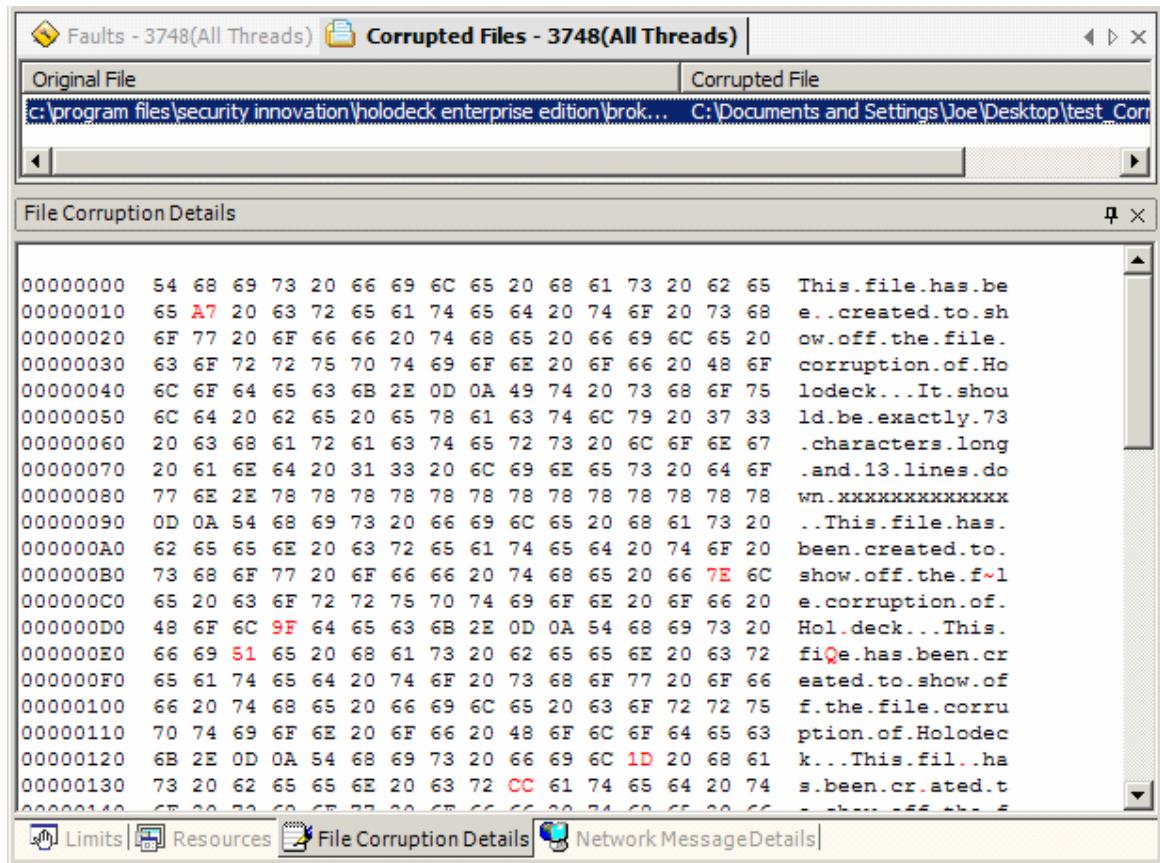


<< Previous Next >>

Viewing the details of the corrupted file

If the application you are testing does some amount of file recovery, you may want to use the File Corruption Details Pane to see exactly where the corruption occurs.

To view the File Corruption Details pane double click a corrupted file in the Project Pane and highlight the entry in the Corrupted Files Pane.



The File Corruption Details pane shows a hexadecimal view as well as the standard ASCII view of the data. The hexadecimal view can be very useful for finding out exactly how the Holodeck file corruptor has changed your file.

All changes made to the file are highlighted in red. For more information on File corruption please see the File Corruption Overview help topic.

<< Previous Next >>

Summary

In this tutorial we've discovered how to create file corruption faults for text files, however Holodeck's File Corruption will work for any type of file. Instead of using the BrokenApp to parse corrupted flat text, try corrupting an image or audio file and loading it into Windows Media Player or Microsoft Paint.

For more information with File Corruption see the [File Corruption Overview](#).

Further Exploration

Try setting a file corruption fault for other file types, including image files or audio files. Since Holodeck corrupts on a bitwise level file corruption for any type is possible. Use Windows Picture and Fax Viewer and try corrupting and loading a bitmap image, then try the same level of corruption with a compressed file type such as jpeg, or gif.

[**<< Previous**](#)

Creating Network Corruption Using Regular Expressions

Introduction

In this Tutorial we will discover how to find and replace network data using Regular Expressions and Replacement strings. This can be very useful when trying to match a certain string and test client or server parsing. In this tutorial we will search for any base URL and replace it with a long string of the letter ‘A’

Setup a new project using Internet Explorer as the application under test, with default logging turned on. For more information on how to setup a project see the Setting up your first project Tutorial in the Beginning Tutorials section.

Note: Regular expressions only work with uncompressed data, some web pages may be compressed for better data transfer; Regular Expressions will not match on these pages. For more information see the Regular Expression help topic.

[Next >>](#)

Investigate Network Packets

Once Internet Explorer has loaded, return to Holodeck. From the Project Pane you can investigate network logs, API logs, and exceptions the application has generated.

With Internet Explorer, navigate to the Intel website (www.intel.com); this website is large, unencrypted, and uncompressed which makes investigating network packets easy. Once the Intel page has loaded return to Holodeck and open the Network logs pane by double clicking on the Network Logs node in the Project pane.

TimeStamp	Thread	Direction	Protocol	Msg Length	Destination IP	Des
05/24/2004 15:24:23:890	1312	Send	MSAFD Tcpip [UDP/IP]	1	127.0.0.1	1
05/24/2004 15:24:23:890	3356	Recv	MSAFD Tcpip [UDP/IP]	1	127.0.0.1	1
05/24/2004 15:24:23:890	1312	Recv	MSAFD Tcpip [TCP/IP]	899	192.168.0.114	1
05/24/2004 15:24:23:890	1312	Send	MSAFD Tcpip [TCP/IP]	519	198.175.96.33	8
05/24/2004 15:24:23:921	1312	Recv	MSAFD Tcpip [TCP/IP]	1	192.168.0.114	1
05/24/2004 15:24:23:921	1312	Recv	MSAFD Tcpip [TCP/IP]	522	192.168.0.114	1
05/24/2004 15:24:23:921	1312	Recv	MSAFD Tcpip [TCP/IP]	3473	192.168.0.114	1
05/24/2004 15:24:23:953	1312	Send	MSAFD Tcpip [UDP/IP]	1	127.0.0.1	1

See the rest of the packet message by selecting the Network Message Details Pane which should be located at the bottom of the window. The Network Message Details Pane shows the entire Network Packet in its Hex and ASCII representation. Any Network corruption will also show up here.

Network MessageDetails			
Offset	Modified Hex Data	Modified ASCII	
0x00000000	48 54 54 50 2f 31 2e 31	20 32 30 30 20 4f 4b 0d	H T T P / 1 . 1
0x00000010	0a 43 6f 6e 74 65 6e 74	2d 4c 65 6e 67 74 68 3a	. C o n t e n t
0x00000020	20 36 36 33 0d 0a 43 6f	6e 74 65 6e 74 2d 54 79	. 6 6 3 . . C o
0x00000030	70 65 3a 20 69 6d 61 67	65 2f 67 69 66 0d 0a 4c	p e : . i m a g
0x00000040	61 73 74 2d 4d 6f 64 69	66 69 65 64 3a 20 46 72	a s t - M o d i
0x00000050	69 2c 20 30 32 20 41 70	72 20 32 30 30 34 20 31	i , . 0 2 . A p
0x00000060	36 3a 34 35 3a 30 39 20	47 4d 54 0d 0a 41 63 63	6 : 4 5 : 0 9 .
0x00000070	65 70 74 2d 52 61 6e 67	65 73 3a 20 62 79 74 65	e p t - R a n g
0x00000080	73 0d 0a 45 54 61 67 3a	20 22 62 38 33 65 36 64	s . . E T a g :
0x00000090	64 63 64 31 31 38 63 34	31 3a 31 30 37 61 22 0d	d c d 1 1 8 c 4
0x000000a0	0a 53 65 72 76 65 72 3a	20 4d 69 63 72 6f 73 6f	. S e r v e r :
0x000000b0	66 74 2d 49 49 53 2f 36	2e 30 0d 0a 50 33 50 3a	f t - I I S / 6
0x000000c0	20 43 50 3d 22 43 41 4f	20 44 53 50 20 43 4f 52	. C P = " C A O
0x000000d0	20 43 55 52 61 20 41 44	4d 61 20 44 45 56 61 20	. C U R a . A D

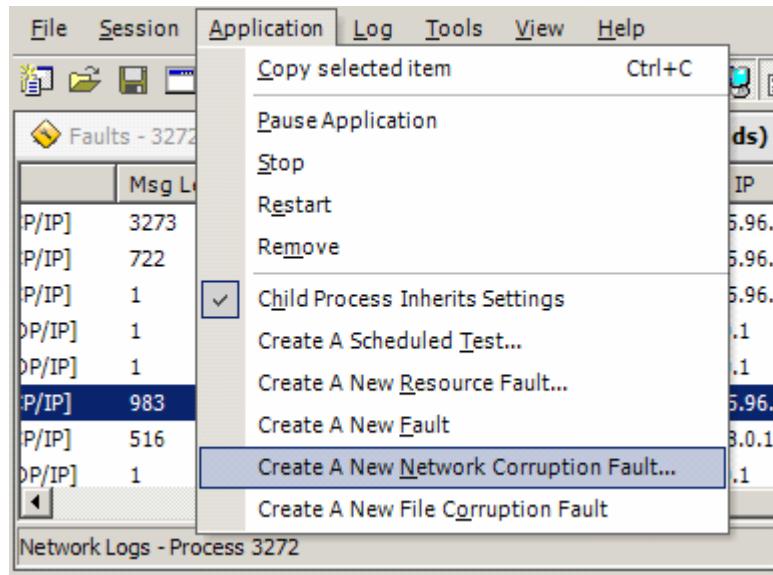
You can use these logs to investigate exactly how to corrupt the message for maximum application penetration, which will help find bugs deeper within your codebase.

<< Previous Next >>

Open Network Corruption Wizard

We want to corrupt any occurrence of a base URL, and replace it with a long string of the letter 'A'. This will help test packet parsing, and will make any errors we find very easy to discover in the web browser or the Network Message Details Pane.

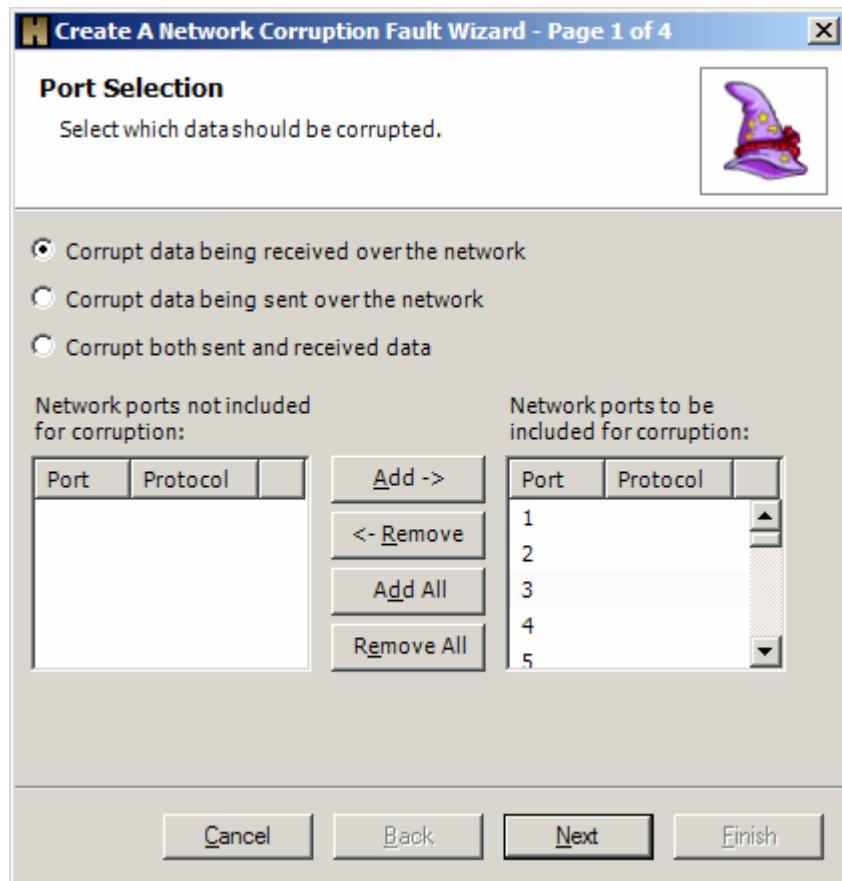
Open the Network Corruption Wizard by selecting Application > "Create A New Network Corruption Fault..."



<< Previous Next >>

Select Which Data Should be Corrupted

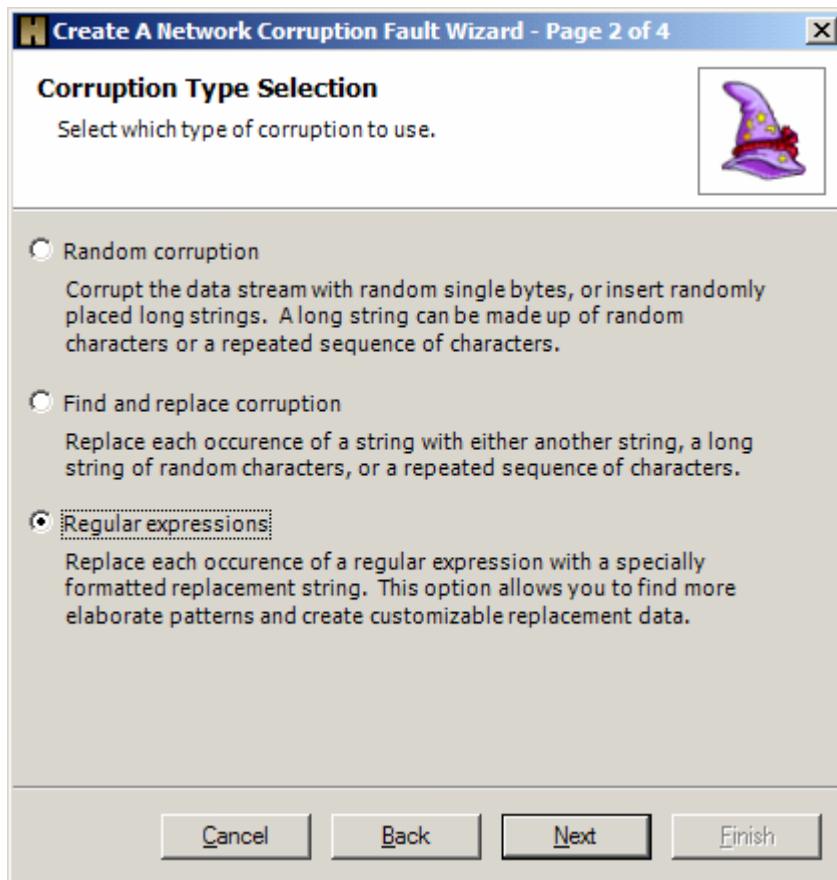
Select all ports and protocols and only data being received over the network since we only want to test the client. This will make sure that all data being sent to Internet Explorer will be corrupted. Since we are not testing the http server there is no need to corrupt outgoing data.



<< Previous Next >>

Select Regular Expressions as the Method of Corruption

Select the Regular Expressions radio button and click next; this will open the regular expression page of the network corruption wizard.



<< Previous Next >>

Create a Regular Expression and Expansion String

Create a regular expression to match on the domain of any URL (e.g. www.google.com, or slashdot.org) and replace it with 1000 to 5000 of the letter ‘A’.

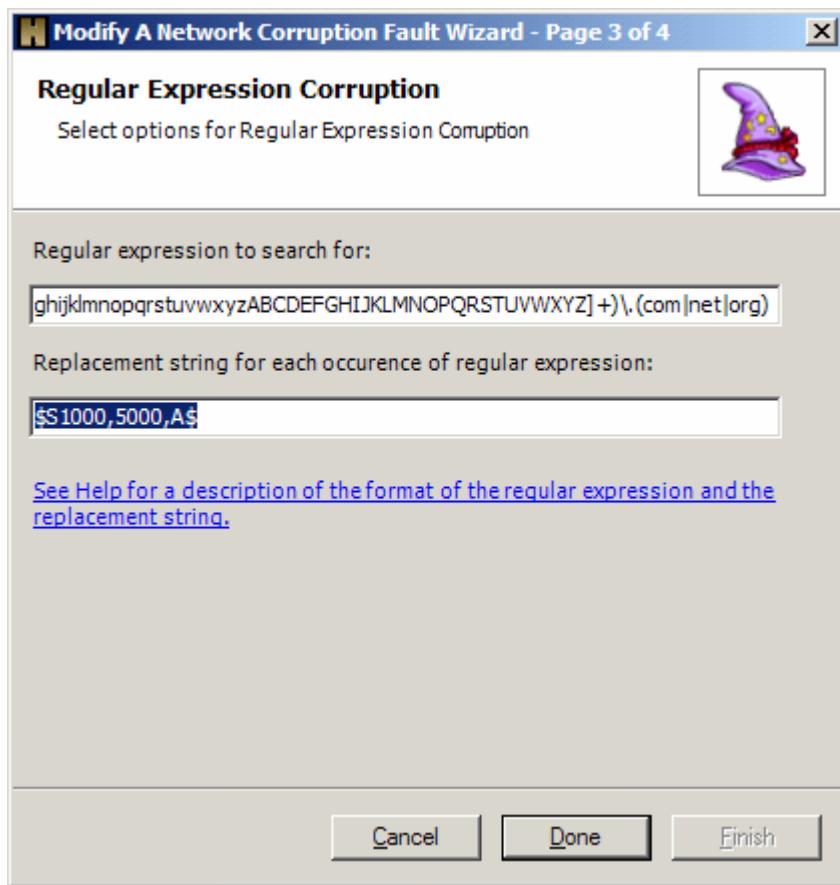
The regular expression to match any domain (www.intel.com) is as follows:

([wW]{3,}\.)? ([abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ]+) \.(com net org)	
([wW]{3,}\.)?	this part matches an optional capital or lowercase "www."
([abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ]+)	this part matches any length string consisting of upper or lowercase alphabet letters, greater than 1 (eg. "abc", "aRdfpQ", etc.)
\.(com net org)	this part matches a period followed by any one of the following, "com", "net", or "org"

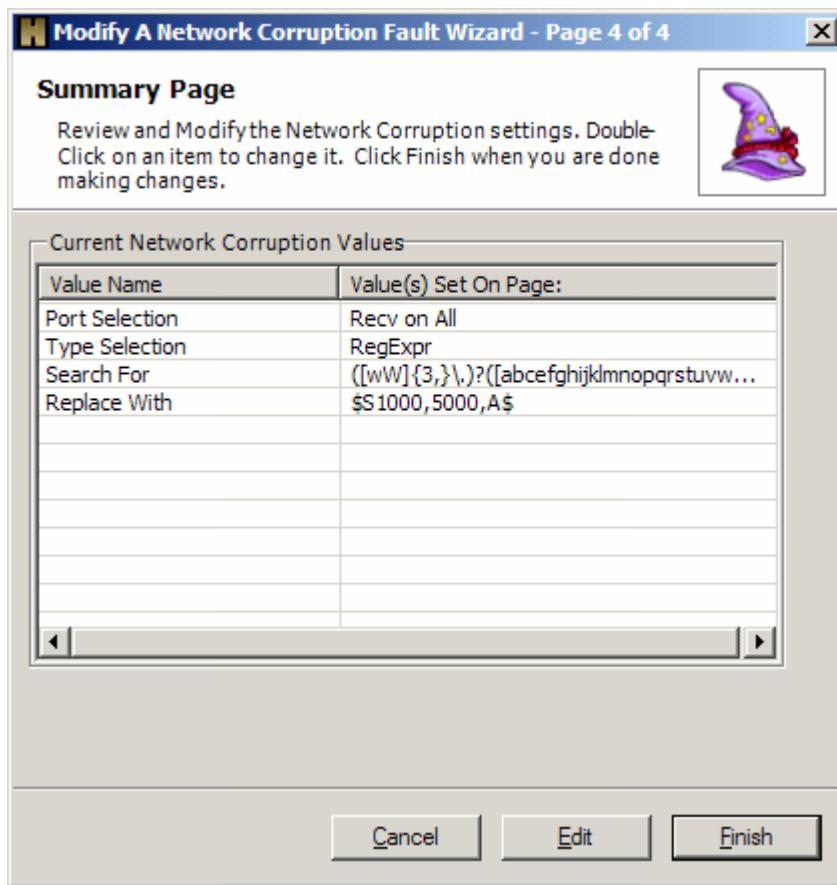
Example Expansion String:

The expansion string to expand between 1000 and 5000 ‘A’s is as follows:

\$	s	this will expand to a string
1000		the minimum number of repetitions
5000		the maximum number of repetitions
A		the string to expand
\$		By enclosing this string in ‘\$’ Holodeck knows this is a string to expand.



The final page of the wizard shows a summary of the corruption fault you have just created. You can go back and change any one of the settings for the fault from this page, simply select the value name you wish to change and click Edit.



[<< Previous](#) [Next >>](#)

Verify the Fault is set

When we click on any link on the Intel web page Holodeck immediately starts replacing all base URLs with long strings of the letter 'A'. This makes Internet Explorer unusable, but you can make sure the fault has set by checking the Network Logs in the Network Logs Pane, and the Network Message Details Pane.

To check the Network Logs open the Network Logs Pane by double-clicking on the Network Logs node in the Project Pane. In the screenshot below you can see two network messages that have been corrupted.

	Msg Length	Destination IP	Destination Port	Source IP	Source Port	Start of Msg
P/IP]	169	192.168.0.114	18181	216.239.57.104	80	HTTP/1.1
P/IP]	1024	192.168.0.114	17925	198.175.96.33	80	AAAAAAAAAA
P/IP]	1024	192.168.0.114	17925	198.175.96.33	80	AAAAAAAAAA
P/IP]	1024	192.168.0.114	17925	198.175.96.33	80	ites/nav
DP/IP]	1	127.0.0.1	1325	127.0.0.1	11525	!
DP/IP]	1	127.0.0.1	11525	127.0.0.1	1325	!
P/IP]	733	192.168.0.114	17925	198.175.96.33	80	his,a=Ma
P/IP]	5	192.168.0.114	18181	216.239.57.104	80	0....

The following is the Network Message Details for this Network Packet, all changes Holodeck makes show up in blue in the Modified columns, to the right of the Modified Columns are the Original Data columns, overwritten or changed data will show up in red here.

Offset	Modified Hex Data	Modified ASCII
0x00000000	41 41 41 41 41 41 41 41	A A A A A A A A
0x00000010	41 41 41 41 41 41 41 41	A A A A A A A A
0x00000020	41 41 41 41 41 41 41 41	A A A A A A A A
0x00000030	41 41 41 41 41 41 41 41	A A A A A A A A
0x00000040	41 41 41 41 41 41 41 41	A A A A A A A A
0x00000050	41 41 41 41 41 41 41 41	A A A A A A A A
0x00000060	41 41 41 41 41 41 41 41	A A A A A A A A
0x00000070	41 41 41 41 41 41 41 41	A A A A A A A A
0x00000080	41 41 41 41 41 41 41 41	A A A A A A A A
0x00000090	41 41 41 41 41 41 41 41	A A A A A A A A
0x000000a0	41 41 41 41 41 41 41 41	A A A A A A A A
0x000000b0	41 41 41 41 41 41 41 41	A A A A A A A A
0x000000c0	41 41 41 41 41 41 41 41	A A A A A A A A
0x000000d0	41 41 41 41 41 41 41 41	A A A A A A A A

<< Previous Next >>

Summary

This tutorial has shown how to create network packet corruption, by replacing certain regular expressions with a long string of ‘A’s. This helps to find obvious serious errors, however you can use the replacement and expansion strings to trick your application into parsing the input as valid. For instance, in this case you could replace any URL with a known outlying case such as www%2Eintel%2Ecom the %2Es represent the hexadecimal version of the period(.) however in Browsers such as Internet Explorer the period is used to specify zones and decide whether to allow or disallow certain scripts and executables. Using the Regular Expression functionality to replace all the URLs you will be able to check many different locations for the address at one time.

Further Exploration

You can use Holodeck to test server packet parsing as well. Try loading a client that connects to the server you wish to test into Holodeck. Create a Network Corruption Faults on outgoing packets. Holodeck will now send packets that are corrupted from the original packet from your client application.

[**<< Previous**](#)

Failing a Single API Call

Introduction

In this tutorial we will discover how to fail a single API test. This can be very useful when trying to test a single function within an application. In this example we will fail the Open URL functionality within BrokenApp.

To begin this tutorial load BrokenApp into Holodeck as the application under test, with only .NET logging turned on; or load the project created from the Creating your First Project Tutorial.

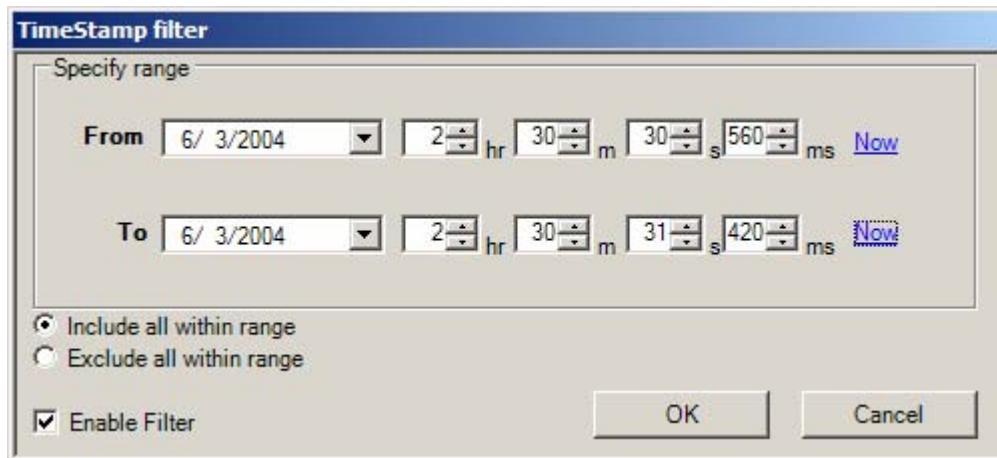
[Next >>](#)

Investigate Logs

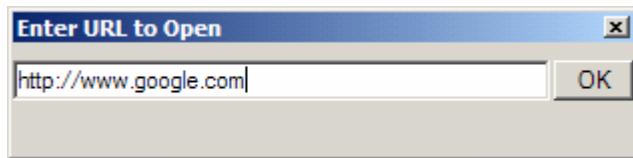
To investigate logs we first must do a little research to see how the BrokenApp handles the open URL menu item.

Once BrokenApp has finished loading return to Holodeck and open the API logs from the Project Pane by double clicking on the API log icon. For easier investigation we want to filter only the API calls that happen for one specific feature within the application; create a timestamp Custom Filter.

To create a custom filter right click the TimeStamp column header and select Custom Filter. This will bring up the Custom Filter dialog box for the TimeStamp column. Click the Now button on the From row, then click the Now button on the To row, this will set the viewable logs in the log pane to those that only show up between those two times. There should not be any log entries.

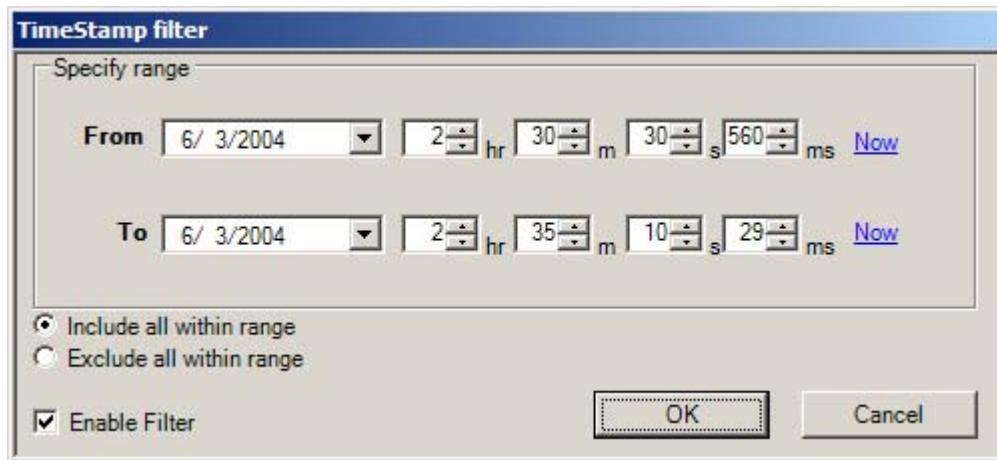


Return to BrokenApp and Click File > "Open URL..." This will bring up the Enter URL to Open dialog box. Type "http://www.google.com" into this box. Google was chosen because it is a small and lightweight webpage, and won't take long to download.



Once the BrokenApp has finished loading the URL return to Holodeck and expand your logs to include the log entries that were just created by this function. Right click the TimeStamp header

and when the TimeStamp filter dialog box pops up click the "Now" button on the "To" row; doing this will expand the logs available for viewing to all logs that have been recorded since the last TimeStamp filter until now.

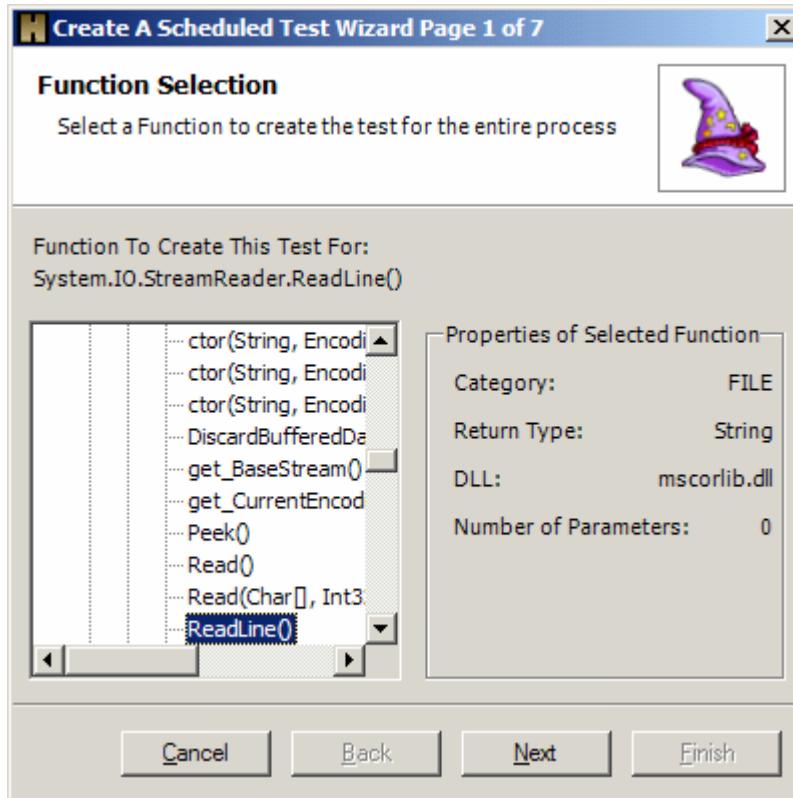


The logs pane now shows all the API functions BrokenApp calls to open a URL and output the source to the main window. To get all the API function calls grouped together click the header of the function column. Now scroll down, keeping an eye on the return value column. You might notice that there are a number of calls to the .NET function "System.IO.StreamReader.ReadLine()" BrokenApp uses this function to read from the incoming network stream; this is clear by the return values present in this function call. This is the function we will fail.

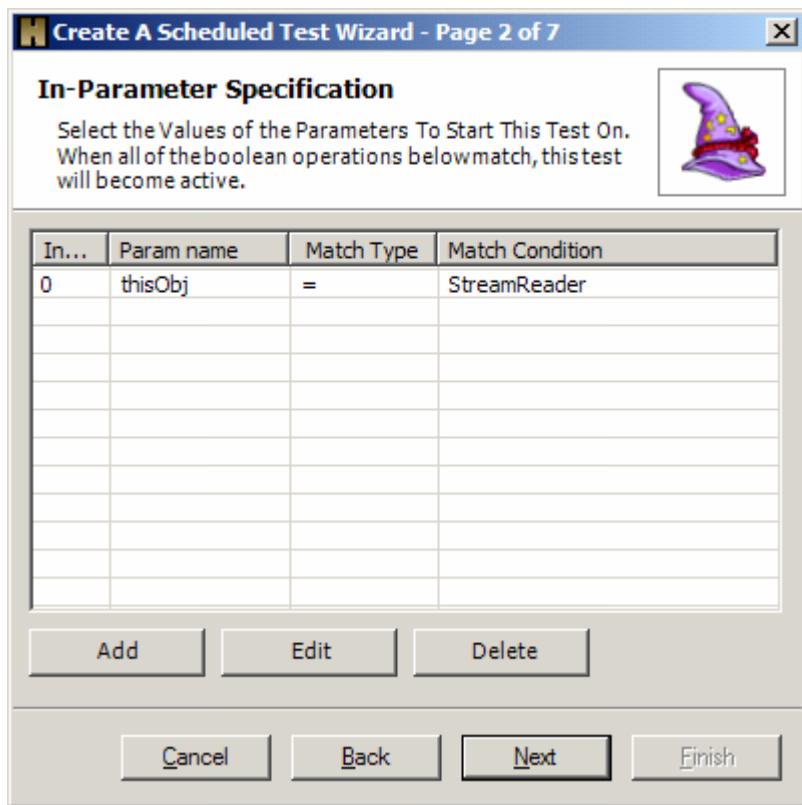
<< Previous Next >>

Failing the ReadLine function

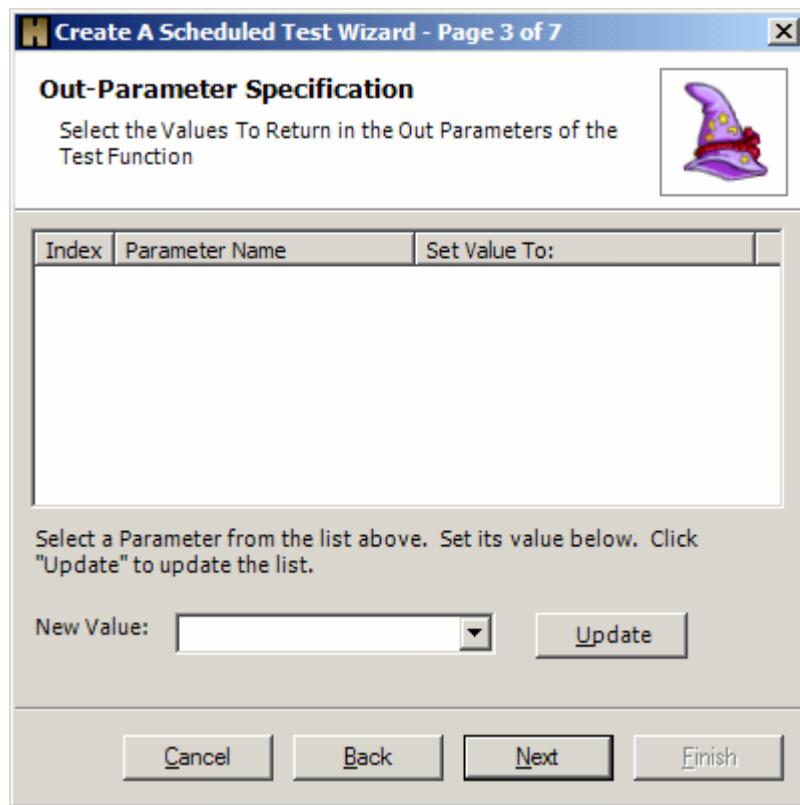
To set a fault for the System.IO.StreamReader.ReadLine() function call right click any of the log items for this function and select "Create a Scheduled Test..." This will bring up the Scheduled Test Wizard with the correct function already highlighted.



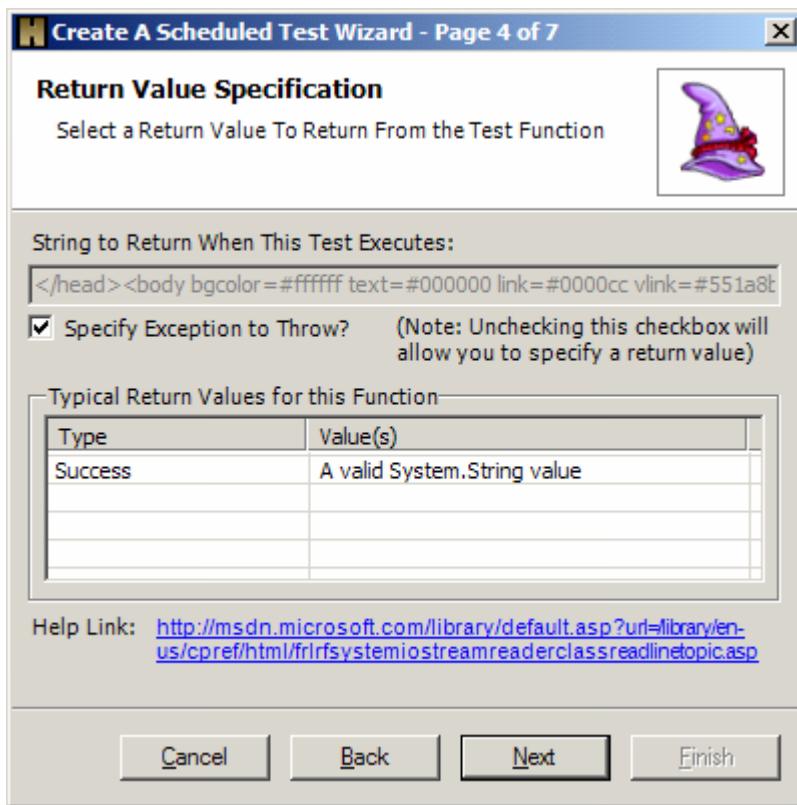
No In-Parameters need to be changed as the In-Parameter only specifies this is to be used as a streamreader.



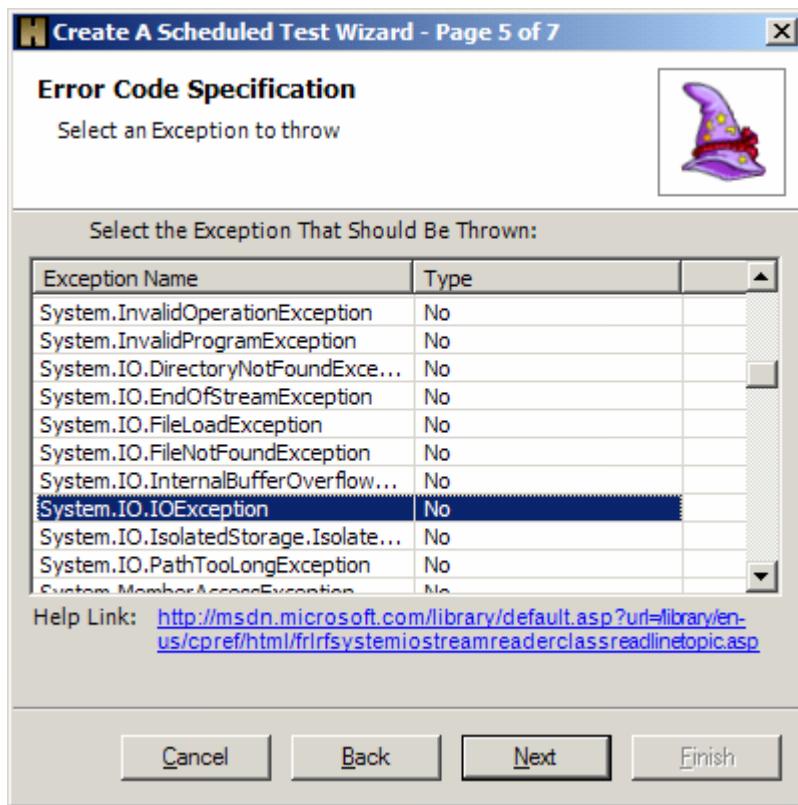
There are no Out-Parameters to change, so we can leave the third page untouched.



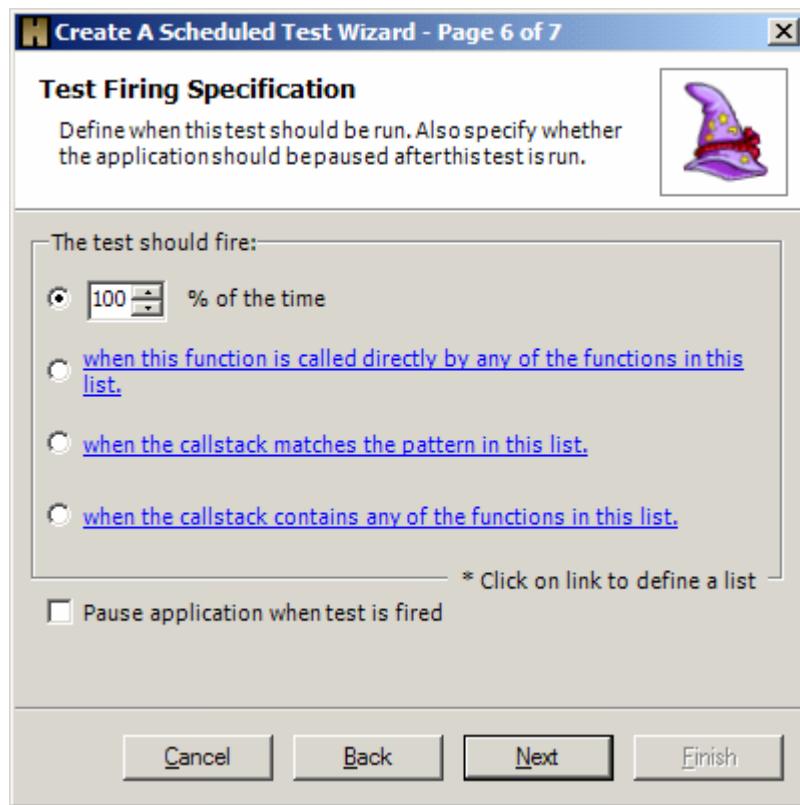
The fourth page allows you to specify a new return value, or throw an exception. In this case we want to throw an exception so leave this page untouched as well and click next.



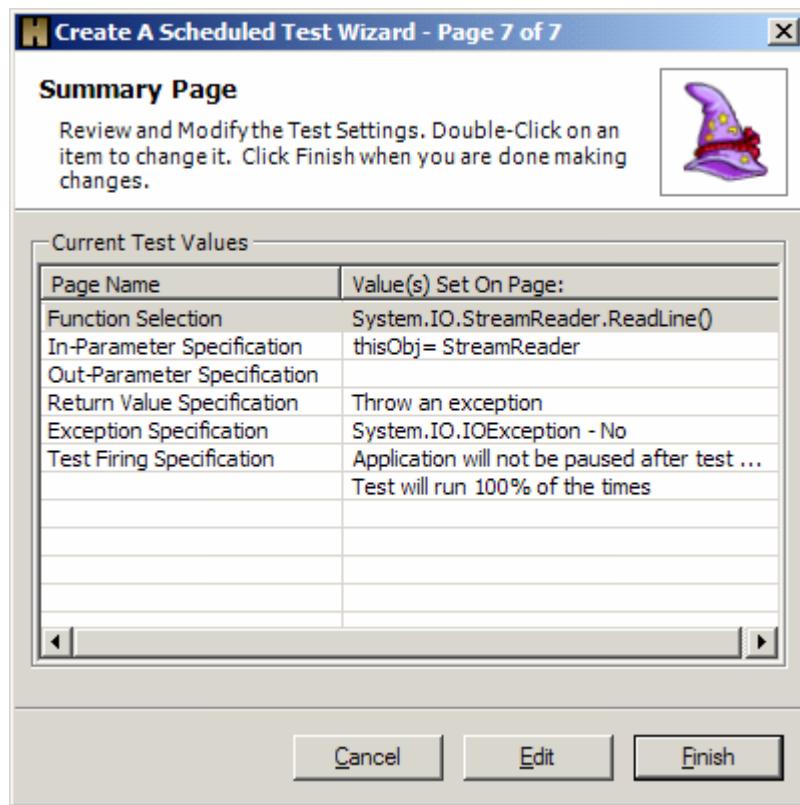
On the last page we can research which Exceptions this function calls by clicking the link at the bottom of the page. Opening this link will take you to the MSDN topic for the ReadLine method. This page shows there are two exceptions that are used by this method, OutOfMemoryException and IOException. Since we can simulate low and out of memory states using limits and faults we will select the IOException which MSDN states will happen when any I/O error occurs. Return to Holodeck and select the System.IO.IOException in the exception list to have Holodeck set this exception state every time the function is called.



To make sure this test will fire on the next time the function is called, we will specify the test to fire 100% of the time.



The final page of the wizard is a summary of the fault, if everything looks in order click finish and the fault will be set immediately.

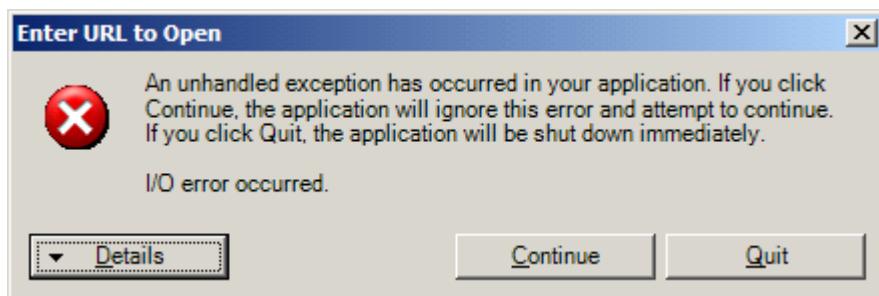


<< Previous Next >>

Verify ReadLine Fault is Set

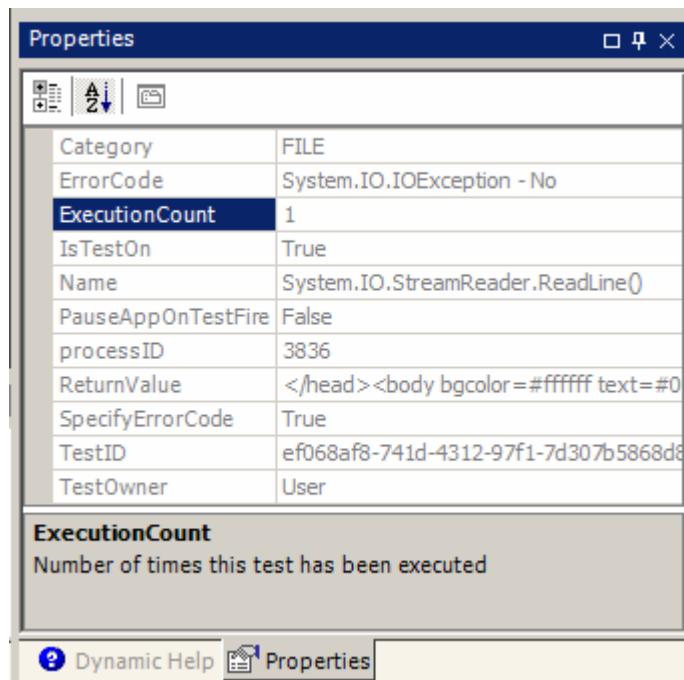
Once a fault has been set you can verify it one of two ways, verify the fault with the application, or check the number of times the test has executed in the Properties Pane.

Holodeck will remember all the information for each test until the project is closed so we can verify the test first in the application. The test we set was for when the BrokenApp attempted to load the URL information into the main window. Return to the BrokenApp and load www.google.com again; click File >Open URL... and type www.google.com into the text box. After BrokenApp reports the response stream has been received it will fail to catch the following error:



Either click Continue to keep the BrokenApp running, or Quit to close the BrokenApp.

Return to Holodeck and verify the failure through the properties pane. Show the Properties Pane by clicking the Properties pane tab at the bottom right of the screen. Now open the Test Pane and highlight the test you just created. In the properties pane the ExecutionCount property shows how many times this test has executed. This shows it has executed once, because the BrokenApp crashed the first time the test fired.



<< Previous Next >>

Summary

Scheduled tests can be applied to any API call, failing the readline function is an easy way to verify a test is set.

Further Exploration:

Try to create other tests on the BrokenApp that will cause crashing unhandled exceptions. Research the Insert Hyperlink and Insert Date or Time functionality in the Broken app, then fail the call to System.IO.MemoryStream.Write. Try researching the cause of these crashes using Holodeck's Reports tool, or by logging first chance exceptions and loading them into Visual Studio for debugging.

Try failing the call only 10% of the time, a robust application should be able to recover from this level of failure.

[<< Previous](#)

Learning About API Failures During Startup and Comparing Logs

Introduction

In this tutorial we will learn how to investigate API failures using Holodeck's logs and reporting tools. Using these features of Holodeck, you can investigate bugs more precisely and in ways never before possible.

The Reports feature allows you to drill down into a compiled version of the data accumulated during a test run. This data includes every log entry, fault, limit, resource dependency, and test that has been created for the current project. Each table in the report can be filtered, modified or manipulated however you see fit, providing a detailed yet precise view of the data.

Comparing logs is a good way to filter out unimportant data on similar runs of an application. Holodeck can export an easy to manipulate, tab delimited, text file containing all of the logs for the current session.

[Next >>](#)

Load project

- Load Excel into Holodeck
- 1) Start Holodeck and create a new project
 - a) File > New Project
 - b) Walk through the New Project Wizard
 - c) Select Excel to test (usually located at: "C:\Program Files\Microsoft Office\Office10\excel.exe")
 - d) Leave all other settings at their default values
 - 2) Excel will automatically be started when you click finish on the New Project Wizard.

<< Previous Next >>

Filtering the log result

You can sort, filter, add or remove columns in the log window to make your investigation easier. You can change the order of the columns by clicking and dragging the columns where you want them to go.

The screenshot shows the Holodeck Enterprise Edition interface with the title bar "Holodeck Enterprise Edition (Trial Version)". The menu bar includes File, Session, Application, Log, Tools, View, and Help. Below the menu is a toolbar with various icons. The main window is titled "EXCEL.EXE Log - 2564(*)". A context menu is open over the first log entry, listing "Custom Filter..." and "Sort column". The log pane displays five entries:

Time	Thread	Category	DLL	Error Code
2/19/2004 1...	1204	LIBRARY	kernel32.dll	G
2/19/2004 1...	1204	PROCESS	kernel32.dll	G
2/19/2004 1...	1204	LIBRARY	kernel32.dll	L
2/19/2004 1...	1204	REGISTRY	advapi32.dll	R
2/19/2004 1...	1204	REGISTRY	advapi32.dll	R
2/19/2004 1...	1204	REGISTRY	advapi32.dll	R

Filter the Error Code Column to show only non-success error codes by clicking on the header of the Error Code Column and clicking Custom Filter... In the filter window, add all error codes, then select ERROR_SUCCESS and click the << Remove button. This will show all the failed API calls in the log pane. While this allows you to view all the API failures you'll see that the Reports feature creates a more organized view based on error type, pass/fail, and will allow you to easily drill down on each API Failure.

Click on the header of any column to sort the column or to add a filter. Filtering the column here will only change the log entries you are currently viewing, Holodeck will continue to intercept and log all API calls. To change which API calls Holodeck will intercept and log click Log > Filter on the menu bar.

To add or remove a column click View > Field Chooser this will allow you to choose which fields show up in the log pane. This will not change what information is exported to the log. All fields will be exported.

<< Previous Next >>

Creating a Report for easy viewing

We can investigate any failures by wading through some 4000 log entries, but Holodeck includes reporting functionality that can make the process of investigating a run of any application under test very easy.

To create a report right click the Reports node in the Project Pane, and click Create a Report. Holodeck will now begin to compile the logs, resources, faults, limits and tests for this project into one easy to read report. Once the report has been created Holodeck will automatically display it in the main window. There is a plethora of information available here, but what we are interested in is the Error Codes Table.

From the Error Codes Table we can drill down to investigate each of the failures Excel encountered while running. Many of these errors are .dll failures which are the standard way of looking for DLLs within a search path, but others can be more serious functionality failures.

By clicking on the plus symbol to the left of Fail you can see a detailed version of the Error Code, and Error Code Type. This will also expand the Count of Occurrence columns to reflect the count of the failures of each Error Code.

By clicking on the plus symbol to the right of Fail you can see detailed information about each API call, this can be filtered by the different error codes. Any column can be sorted ascending or descending by right clicking the column and clicking Sort. Columns can be reordered by dragging them to a different location, or to easily remove a column simply drag it off the table. To add a column you have removed, right click the table and click Field List, this will bring up another window, where you can drag columns around to make data viewing easier. It is interesting to note here that during a regular startup, Excel encounters nearly 600 errors, which we can see by repeating the test. This would be a good place to look for exploitable bugs; if Excel failed to load a certain library, or failed a function call it might put the application in a useable but unstable state, which could be a spring board for finding more bugs.

<< Previous Next >>

Exporting logs

Holodeck also gives you the ability to export the logs generated when running the Application Under Test in Holodeck. This is useful if you would like to compare the logs in an external editor, such as Excel or Windiff.

Exporting log files

At any time during the execution of the Application Under Test you can export the current log to a text file. Export a log by click File > Export Log to File...

You may want to export the log files of two similar runs of an application to investigate differences between API calls, or the function calls leading up to a crash or bug.

For more information on exporting logs see the Exporting Logs page in the Holodeck in Depth section.

<< Previous Next >>

Comparing logs

You can quickly and easily compare the differences between two log files using Windiff, a utility for comparing two similar files which is bundled with Visual Studio.

Example Comparison

Let's compare the differences in API calls when Excel opens a document by double clicking on it, versus using the quick launch pane. We can do this with one instance of Holodeck by using the multiple test application feature.

To do this first create a new project, and set Excel to be the application under test. Once Excel has started, browse to a simple Excel document and open it by double clicking it. Once the document has finished loading, export the log file to a text document and save for later use.

Once the log has been exported you can close Excel, and open a new instance by clicking File > New Test Application, this will open the New Test Application Wizard which will allow you to select Excel to start again, and create a new API log.

Since you closed the first instance of Excel, the same document should show up in the quick launch bar on the right side of the window. This time, click the link on the sidebar to open the Excel document. Once the document has finished loading, export the log to a file under a different name.

These two logs were created at different times; therefore we have to remove the first two columns, TimeStamp and Thread. API calls often use memory addresses as parameter and return values, each time the application is run the OS may allocate a different memory location we have to remove the Return Value and the Parameter columns as well to ensure we are only comparing the order of API calls.

To open the log files in Excel right click the log file Open With... > Microsoft Excel. Excel will automatically put the data into each column. Click the column header of the TimeStamp column, right click the selected row, and click Delete. This will remove the column, and shift the rest of the data over. Repeat this for the Thread, Return Value and each of the Parameter Columns. Once these columns have been removed you can resave the log as a tab delimited text file. File > Save As...

<< Previous Next >>

Making sense of the difference

Now we can load the two files into Windiff, and see the difference in the way Excel loads a file by double clicking on it, or by loading it from the QuickLaunch Bar. We notice that the first 2000 lines of the logs are very similar; this is the startup of the application. The next 1000 lines are where we loaded the files. you may notice that, with the exception of a few operating system dependant calls, the two load methods take a very similar API path.

The Windiff window displays the differences between two log files: `\exceldoubleclick.txt` and `\excelquicklaunch.txt`. The left pane shows graphical representations of the log files, and the right pane shows a list of registry API calls. Red lines indicate differences from the first file, and yellow lines indicate differences from the second file. The log file contains numerous entries for `advapi32.dll` functions like `RegQueryValueExA`, `RegCloseKey`, etc. The differences are categorized into Application Loading, Opening Differences, and File Load and Closing Similarities.

Line Number	File	Function	Arguments	
205	REGISTRY	advapi32.dll	RegQueryValueExW	
206	REGISTRY	advapi32.dll	RegCloseKey	
207	REGISTRY	advapi32.dll	RegOpenUserClassesRoot	
208	REGISTRY	advapi32.dll	RegOpenKeyExW	
209	REGISTRY	advapi32.dll	RegCloseKey	
210	REGISTRY	advapi32.dll	RegEnumKeyExA	
211	REGISTRY	advapi32.dll	RegQueryValueW	
212	REGISTRY	advapi32.dll	RegCloseKey	
213	REGISTRY	advapi32.dll	RegOpenKey	
214	REGISTRY	advapi32.dll	RegQueryValueExA	
215	REGISTRY	advapi32.dll	RegCloseKey	
216	REGISTRY	advapi32.dll	RegOpenKeyExA	
[21]	>	REGISTRY	advapi32.dll	RegEnumKeyExA
[22]	>	REGISTRY	advapi32.dll	RegOpenKeyExA
[23]	>	REGISTRY	advapi32.dll	RegEnumKeyExA
[19]	>	REGISTRY	advapi32.dll	RegOpenKeyExA
217	REGISTRY	advapi32.dll	RegEnumKeyExA	
[30]	>	REGISTRY	advapi32.dll	RegOpenKeyExA
[27]	>	REGISTRY	advapi32.dll	RegCloseKey
218	<	REGISTRY	advapi32.dll	RegOpenKeyExA
219	<	REGISTRY	advapi32.dll	RegQueryValueExA
220	<	REGISTRY	advapi32.dll	RegCloseKey
221	<	REGISTRY	advapi32.dll	RegCloseKey
222	<	REGISTRY	advapi32.dll	RegOpenKeyExA
223	<	REGISTRY	advapi32.dll	RegCloseKey
224	REGISTRY	advapi32.dll	RegOpenKeyExA	
225	REGISTRY	advapi32.dll	RegEnumKeyExA	
226	REGISTRY	advapi32.dll	RegOpenKeyExA	
227	REGISTRY	advapi32.dll	RegEnumKeyExA	
228	REGISTRY	advapi32.dll	RegOpenKeyExA	
229	REGISTRY	advapi32.dll	RegQueryValueExA	
230	REGISTRY	advapi32.dll	RegCloseKey	
231	REGISTRY	advapi32.dll	RegCloseKey	
232	REGISTRY	advapi32.dll	RegCloseKey	
233	REGISTRY	advapi32.dll	RegOpenKeyExA	
[23]	>	REGISTRY	advapi32.dll	RegQueryInfoKeyA
234	<	REGISTRY	advapi32.dll	RegOpenKeyExA
235	REGISTRY	advapi32.dll	RegCloseKey	
236	REGISTRY	advapi32.dll	RegCloseKey	
237	REGISTRY	advapi32.dll	RegOpenKey	
238	REGISTRY	advapi32.dll	RegQueryValueExA	
239	REGISTRY	advapi32.dll	RegCloseKey	
240	REGISTRY	advapi32.dll	RegOpenKeyExA	
241	REGISTRY	advapi32.dll	RegEnumKeyExA	
242	REGISTRY	advapi32.dll	RegOpenKeyExA	
243	REGISTRY	advapi32.dll	RegEnumKeyExA	
244	REGISTRY	advapi32.dll	RegOpenKeyExA	
245	REGISTRY	advapi32.dll	RegEnumKeyExA	
246	REGISTRY	advapi32.dll	RegOpenKeyExA	
247	REGISTRY	advapi32.dll	RegCloseKey	
248	REGISTRY	advapi32.dll	RegOpenKeyExA	
249	REGISTRY	advapi32.dll	RegEnumKeyExA	
250	REGISTRY	advapi32.dll	RegOpenKeyExA	
251	REGISTRY	advapi32.dll	RegEnumKeyExA	
252	REGISTRY	advapi32.dll	RegOpenKeyExA	

Using Windiff we can see the differences in each file, the first column shows which line corresponds to each file. Red lines come from the first file, Yellow lines from the second file. On the far left a graphical representation of each file is shown where Windiff tries to match each line to the other file, if it can. Here we can see similarities to the way Excel starts up, and the similarities in the way Excel shuts itself down. The middle of the log file we see the drastic difference of how Excel loads by double clicking or by loading from Quick Launch pane.

<< Previous Next >>

Summary

Exporting and comparing logs is a good way to see changes in two different runs of the same application. Often times the difference in two runs of an application are system state differences; depending on other processes running on the system an application may take different code paths. If this is the case the API function logs can be drastically different.

Further Exploration

Try exporting the logs of a single thread. Turn on per-thread mode by clicking Session > "Per Thread." This can help separate differences in the logs that are only different because threads executed at different times.

[<< Previous](#)

Using Code Coverage with Record/Replay to Test the Application

Set up the project

In this tutorial we will cover how to set up Code Coverage testing within Holodeck to more fully test your application. Code Coverage sets a different fault, limit test or combination of the three each time the application is run. You can let Code Coverage test generation run in the background while you test your application, or have a test harness automatically run tests, to find more code paths than would be normally used.

New code paths are created by the hostile environment Holodeck creates while generating Code Coverage test cases. The hostile environment causes your application to call error catching code paths and recover from these errors on the fly.

Setup a project using the BrokenApp as the application under test, or load the project you created in the Creating your first project tutorial.

BrokenApp is so unstable that many of the faults, limits and tests will cause it to crash. This is fine because Holodeck will automatically restart the application when it crashes to continue testing.

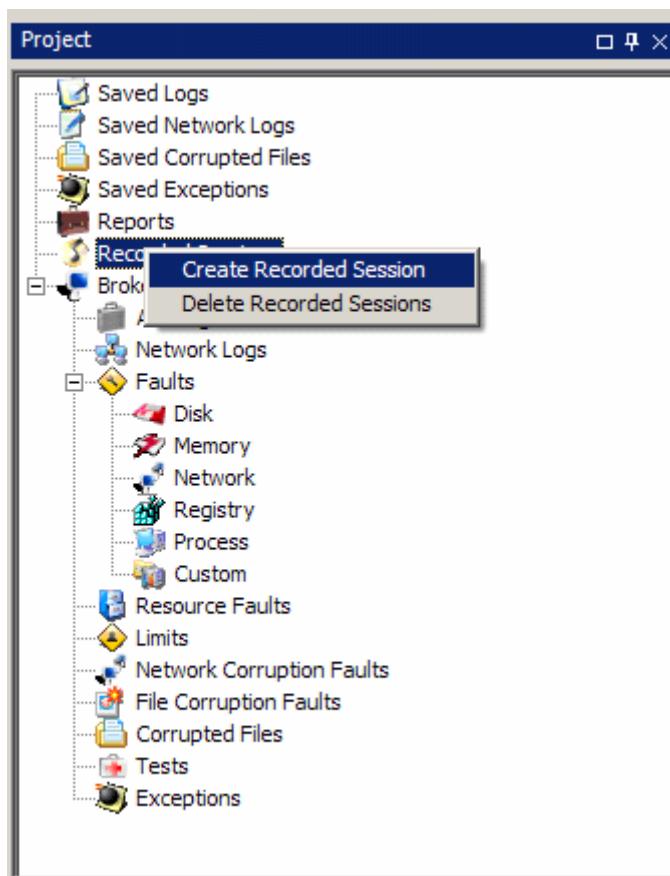
[Next >>](#)

Start Recording the Session

Recording the session will allow you to recreate the hostile environment to reproduce any bugs found in the application. Replaying a recorded session will recreate each fault, limit etc in the order they were set by the Code Coverage Generator.

Once the BrokenApp has finished loading and is ready for input and testing start recording the session.

To start recording the session right click the Recorded Session icon in the Project Pane and select "Create Recorded Session"

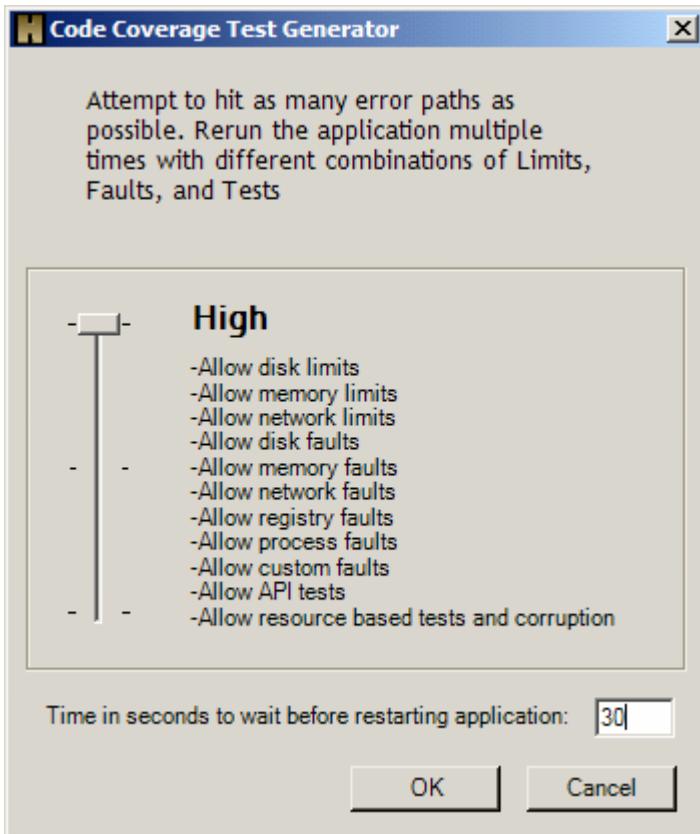


This will begin the recorded session, no further configuration is necessary to set up the recorded session.

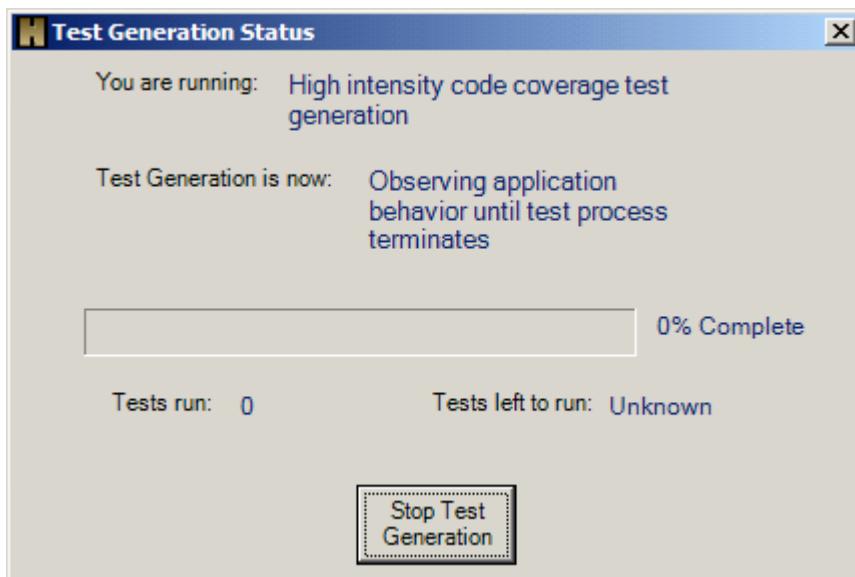
<< Previous Next >>

Begin Code Coverage Test Generation

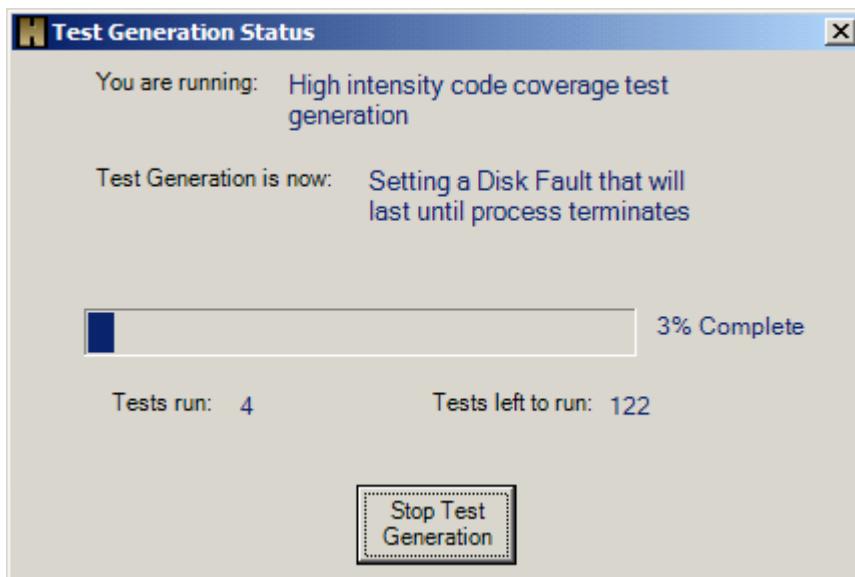
For this tutorial we will be setting High Code Coverage with a relatively short interval to wait before Holodeck restarts the application. High Code Coverage will allow us to discover relatively easy to find bugs in a short period of time. Since the errors will be found quickly an interval of 30 seconds can be set before Holodeck restarts the application.



Once OK is clicked Holodeck will begin observing the application to learn which API functions should be set to create the most hostile environment for your specific application. Holodeck will continue to observe the application until Test Generation has finished. This will allow Code Coverage to test more of the application based on the API calls it sees.



The first observation stage will finish as soon as the application has been closed by the user, or test harness. The next time the application starts up Holodeck will begin setting tests, faults, and limits while the user or test harness tests the application manually.



The testing will continue until the application is terminated by the user, test harness, or a crash. Once the application has terminated Holodeck will restart the application, wait for the application state to change that will trigger the set of a new test, limit or fault.

Use the BrokenApp while Holodeck sets tests, limits and faults to see how the BrokenApp responds to different tests. For this purpose we will test the "Insert Hyperlink" functionality in BrokenApp. Select Insert > "Insert Hyperlink..." this will bring up the insert hyperlink dialog box. Type "http://www.google.com" into this box and click "Insert Link"



Since this test will have to happen every time Holodeck sets a new Code Coverage test it is recommended it is automated using a test harness.

Since recorded sessions was started before starting the Code Coverage Test Generation there is no need to record steps, or remember which faults were set when, the Recorded sessions will be able to replay this information for you any time in the future.

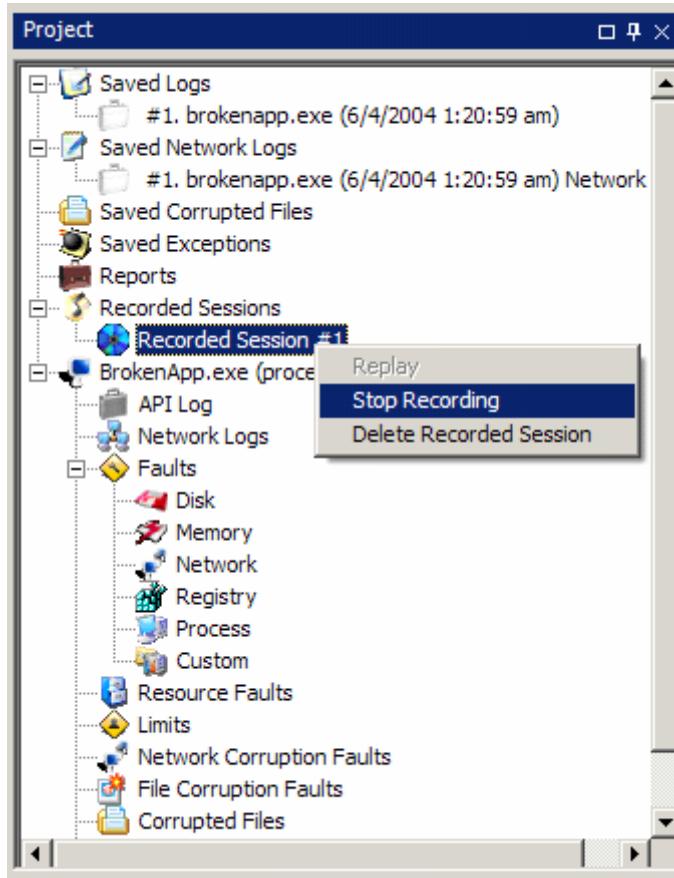
<< Previous Next >>

Stop Test Generation

At anytime before all the tests have finished, the test generation can be stopped by clicking the "Stop Test Generation" button at the bottom of the Test Generation Status window. To demonstrate the Record/Reply functionality of Holodeck stop test generation after a few tests have run. This will return you to Holodeck, and stop any further tests, limits or faults on the application.

Stop Recording the session:

Once the Code Coverage Test Generation tool has finished, you can stop the recorded session from recording. To do this right click the newly created Recorded Session named "Recorded Session #1" and select "Stop Recording"

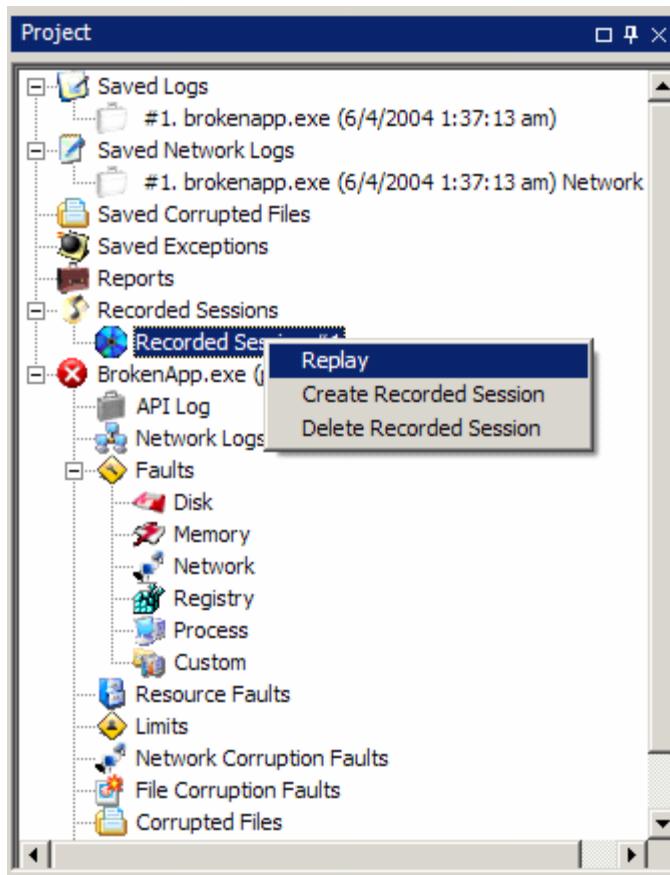


<< Previous Next >>

Replay the Recorded Session

Now that the recorded session has been created, all the faults, limits, tests and restarts of the application have been recorded to the application. Now you can replay the Recorded Session and test your application further, or run the same automated test harness tests on it you used to create the Recorded Session.

To Replay the Session right click the Recorded Session #1 and click "Replay"



Replay will restart the application and set all the faults at the same time based on the number of API logs coming into Holodeck. Replay will not operate the application the same way, no UI changes or User Input is recorded, only changes made to Holodeck are recorded.

<< Previous Next >>

Summary

Recorded Sessions can be extremely useful while trying to reproduce a bug in your application that was produced by a hostile environment produced by Holodeck. Record a Holodeck session when verifying a bug, then attach it to the bug report. This will allow the developer to verify the bug more easily and will allow you to verify the fix quickly when the developer has fixed the bug.

Further Exploration:

Recorded sessions will record any changes made to Holodeck while testing an application, it will not however record user input, UI changes, or test harness application changes. Try using Recorded Sessions while setting your own faults, limits, tests etc within Holodeck for the application. Then replay the same recorded session while doing other testing. You can use Recorded Sessions like a manual macro, which can help you find more bugs quickly. To reproduce the bug simply attach the Recorded Session to the bug report with the repro steps.

For more information see the Replay the Recorded Session overview and the Introduction to Automatic Test Generation help topics.

[**<< Previous**](#)

Advanced

Test Harness and Code Coverage

Introduction

This tutorial will demonstrate how to use an external test harness and Holodeck's Code Coverage feature to find bugs quickly, easily and efficiently. The Code Coverage feature runs tests based on the resources and API calls used by the program. For example: if the Application Under Test uses LoadLibraryA, an API used to load DLLs, the Code Coverage feature will fail that call on one of its runs. More information about the Code Coverage feature can be found in the Holodeck help documents.

By using an external test harness we can automate the use of Code Coverage test generation. The test harness will launch Holodeck; Holodeck in turn will launch the application under test and start testing. By using Holodeck's Recorded Session feature we can track which faults were set and when they were executed. Our test harness will catch exceptions thrown by the application under test and log them to a time stamped file. By comparing the two we will have a direct cause and effect relationship.

[Next >>](#)

Create a Holodeck project file

- 1) Open Holodeck
- 2) Create a new project
- 3) Store your project in a logical location (for this test we'll store our project at C:\test\notepadCodeCoverage.hdp)
 - a) Select the program you wish to test (for this test we'll use C:\windows\notepad.exe)
 - b) Check the APIs you wish to log (for this test we'll use the default logging)
 - c) Perform the action in the application under test that you would like to test (for example: opening a file)
- 4) Save your project and close Holodeck

If the test harness is set to log exceptions the Holodeck debugger must be turned off in the project.

<< Previous Next >>

Setup the Test Harness

The test harness is a piece of software that is responsible for launching Holodeck with command line options, this software is not part of Holodeck and must be supplied by the user. The Test Harness is also responsible for terminating the test at appropriate times and monitoring the application under test for exceptions. When launching Holodeck the following command line options are available:

/record – turns on session recording

/silent – runs Holodeck minimized and with no popup UI

/codecoverage:[high|medium|low] – turns on code coverage with either high, medium or low settings

/randomstress:[high|medium|low] – turns on random stress testing with either high, medium or low settings

/intelligentstress:[high|medium|low] – turns on intelligent stress testing with either high, medium or low settings

For more information on the command-line options in Holodeck see the Introduction to using Holodeck as a command-line Utility

The test harness must terminate the test process after Holodeck has finished running each test so that Holodeck can restart the Application Under Test with new faults set. The test harness is also responsible for monitoring and recording errors thrown by the Application Under Test, it is highly recommended that these are time stamped so that they may be easily compared to Holodeck's recorded session xml files.

<< Previous Next >>

Begin testing with Code Coverage

As Holodeck finds errors in the application under test unexpected error dialogues may appear, or for more serious errors, a debugger may be instantiated. Your test harness should be written in such a way that it can handle these problems and either continue on or terminate so the next test can run. Throughout the entire testing process, Holodeck is recording the faults and tests that are set, while the test harness is recording the errors. When you have collected all of the information needed or when test generation is complete, close the test harness.

Command to run Holodeck with code coverage silently and recording the session:

HolodeckGUI.exe /record /silent /codecoverage:high

<< Previous Next >>

How to compare log files

The two files we will compare are Holodeck's Recorded Session log (Recorded Sessions\Recorded Session #1.xml) and the log from our test harness. By matching the timestamps, or the order of execution, we can tell which faults caused which errors, making for easy reproduction. To match the timestamps, open the Recorded Session.xml file and look for the TimeStamp attribute of the <Log> tag, then look at your own timestamp from your test harness. When two match, you have found the cause of the fault (the <Inject> tag from Recorded Session.xml) and what happened (the log file from the test harness). The two log files are included in the resources section of this document.

How to Compare Log Files:

Using the Recorded Session #1.xml file and a test harness that logs each crash with a timestamp you can match which fault caused which timestamp. Notice the timestamp is set as soon as that fault was set, which may or may not cause an immediate failure in the Application Under Test.

Match timestamps by working backward in time from the Exception, you can see which faults, tests, and limits were set before the exception happened. The Test Harness Log timestamps are often a few seconds after the Recorded Sessions #1.xml timestamps, because the Application Under Test often does not fail immediately after a fault is set. Multiple Faults, Tests, or Limits might be set for any given Exception, so when working backwards you will have to keep track of how each Fault, Test, or Limit was set, deleted or modified. It is also important to make sure you consider Session numbers, each session is a new instance of the Application Under Test, which means all Faults, Tests, and Limits have been reset, be sure not to traverse Sessions or you will not be matching the proper Exceptions with the proper Recorded Session.

The following logs match up as follows:

Fault 1 (File not found) caused the first Exception

Fault 2 (Can't access file) caused the third Exception

Fault 3 (Access denied to file) caused the second Exception

Fault 4 (File is write protected) caused the fourth Exception

1. Recorded Session #1.xml

```
<Recorded_Session>
<Record>
<Log Index="1"
      TimeStamp="7/21/2003 17:8:12.906"
```

```

Session="1"

AppName="notepad.exe" />

<Inject Type="21" Name="File not found" Enabled="2" />

</Record>

<Record>

<Log Index="1"

TimeStamp="7/21/2003 17:8:20.109"

Session="2"

AppName="notepad.exe" />

<Inject Type="21" Name="Can't access file" Enabled="2" />

</Record>

<Record>

<Log Index="1"

TimeStamp="7/21/2003 17:8:26.718"

Session="3"

AppName="notepad.exe" />

<Inject Type="21" Name="Access denied to file" Enabled="2" />

</Record>

<Record>

<Log Index="1"

TimeStamp="7/21/2003 17:9:13.625"

Session="4"

AppName="notepad.exe" />

<Inject Type="21" Name="File is write protected" Enabled="2" />

</Record>

</Recorded_Session>

```

2. TestHarnessLog.txt

----- First chance exception, VERY LOW chance of exploitation

Exception #1 in test case #2

Process HolodeckGui.exe caused access violation:

Read of 0x00000000 at instruction 0x040f5314

Test case started at 7/21/2003 17:8:14.76

----- First chance exception, LOW chance of exploitation -----

Exception #2 in test case #2

Process HolodeckGui.exe caused access violation:

Read of 0x0000004c at instruction 0x040f19c6

Test case started at 7/21/2003 17:8:28.40

----- First chance exception, LOW chance of exploitation -----

Exception #3 in test case #2

Process HolodeckGui.exe caused access violation:

Read of 0x0000004c at instruction 0x77e73887

Test case started at 7/21/2003 17:8:22.9

----- First chance exception, VERY LOW chance of exploitation

Exception #4 in test case #2

Process HolodeckGui.exe caused access violation:

Read of 0x00000000 at instruction 0x0390e241

Test case started at 7/21/2003 17:9:16.10

<< Previous Next >>

Summary

Using Holodeck with a test harness can help ensure reproducibility of bugs and, when run on daily builds, can help test for obvious crashing bugs. Have the test harness launch Holodeck with code coverage or stress test automatic testing turned on and Holodeck will automatically set faults, tests and limits while the test harness commands the application.

Further Exploration

Try turning on Intelligent Stress Test Automatic testing while the test harness runs. Intelligent Stress Testing puts your application into a hostile environment based on the API function calls Holodeck sees from a test run.

Try recording a testing session, then replay the session while the test harness runs. This will make sure the same tests, faults, and limits are run each time the application is restarted.

[**<< Previous**](#)

Adding New Intercepts

Adding New Intercepts Introduction

In this tutorial you will learn how to generate a new intercept library for a custom dll, observe logging, and set a test that targets the newly intercepted method calls. This can be very useful when testing your application because you can intercept the library calls specific to your application. If you've created libraries that your application calls Holodeck can intercept these

Create a Test dll – Create the dll which we will be intercepting.

Create the Test Application – Create an application that uses the test dll that we just created.

Add a New Intercept Library – Add the dll to the Holodeck interception library.

View Logs and Create a Test for the Test dll – research how the test application uses the dll, and set a test for the function to fail.

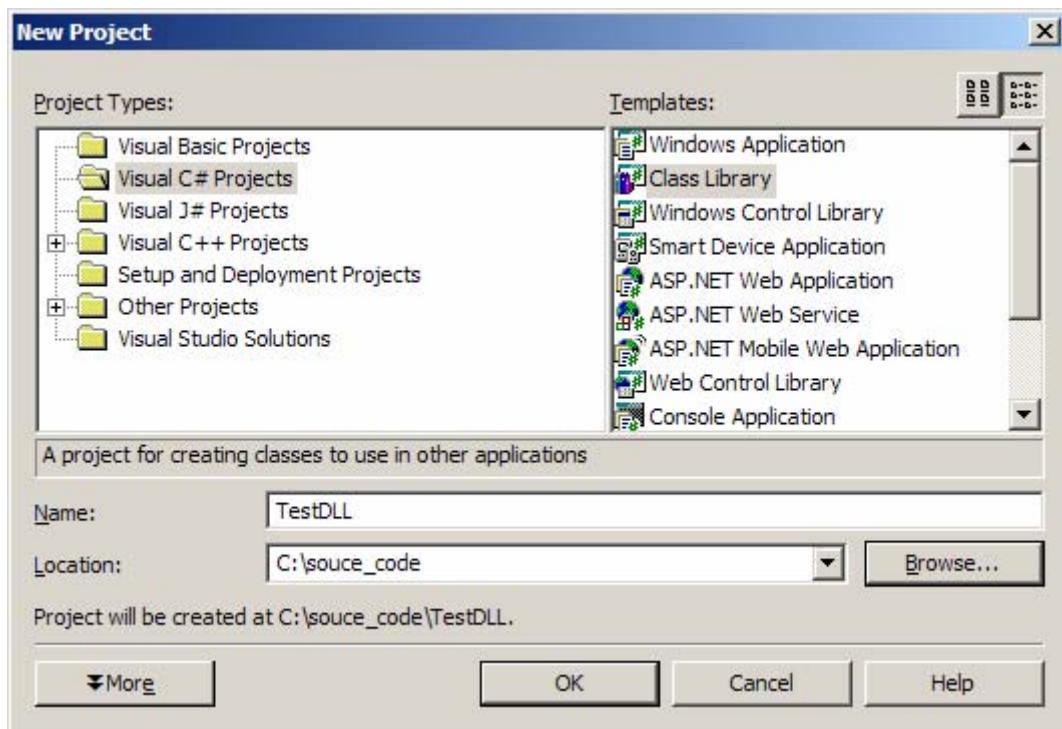
Further Exploration – modify the test, add other dlls to the interception list, and create a custom fault.

[Next >>](#)

Create a Test dll

In this project the first thing we will do is to create a test dll, this will give us a library to use in the test application. We will intercept this test dll within Holodeck to show how Holodeck can intercept any libraries, including dlls that are not supported out of the box.

Open Visual Studio.net and create a new test dll. Click New Project > Visual C#, class library. Name it TestDLL



Within the project create a new test method.

```
Just below the constructor in class1.cs (public Class1()) add this new method  
public bool TestMethod(int paramInt, bool boolParam)  
{  
    return true;  
}
```

Build the project, click Build > "Build Solution" from the menu.

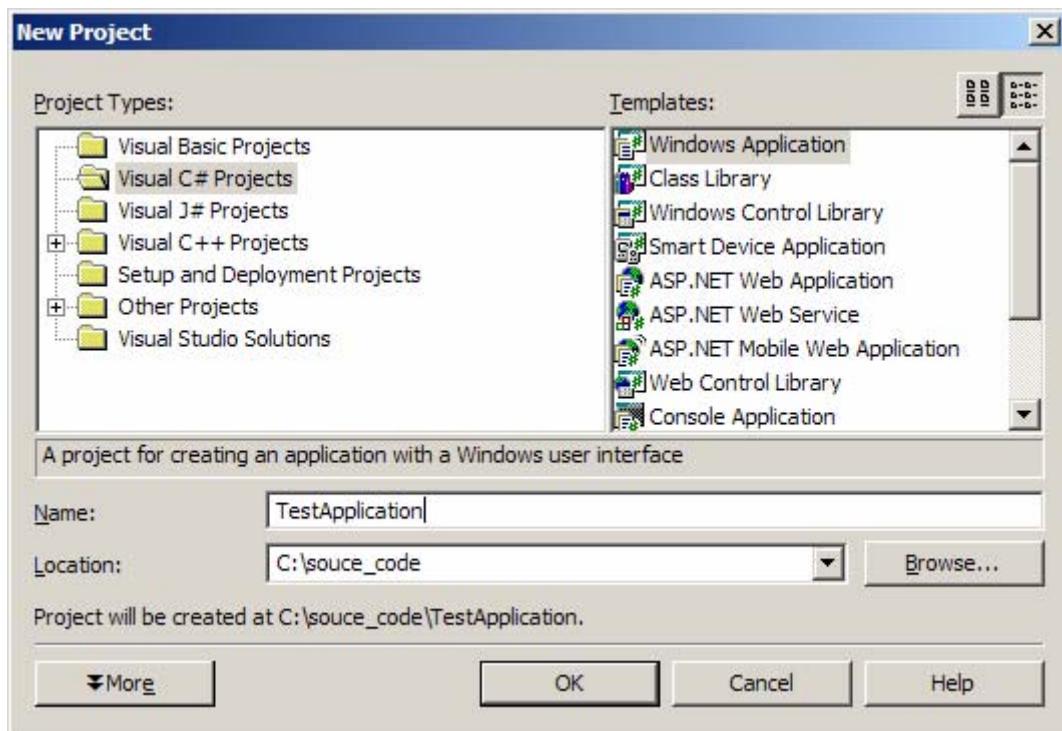
This will create the dll that we will use later in the application and add to Holodeck's interception list.

<< Previous Next >>

Create the Test Application

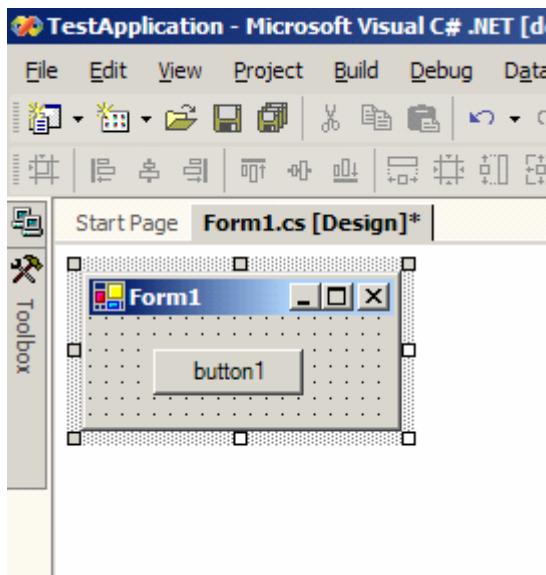
This test application will call the test dll we just created, after clicking a button. This is an extremely simple version of a program that uses a dll to finish its program operations. This test application will be the application we load into Holodeck, when it calls functions in the test dll Holodeck will be able to intercept these functions like it does with the native .NET and native functions out of the box.

Create a new project in Visual Studio.net Click New Project > Visual C# Windows Application, name it TestApplication.



Add a button to the form

Show the Toolbox pane and click button. Click anywhere on the form to insert the button.



Double click the button to create a method that will fire when button1 is clicked. Double clicking the button will take you directly to the code view of this application and automatically create a method called "button1_Click."

Add the following code to button1_Click

```
private void button1_Click(object sender, System.EventArgs e)
{
    Class1 testClass = new Class1();
    testClass.TestMethod(0, false);
}
```

Add a reference to TestDLL.dll

Adding the test dll by reference is how you can load external library functions, this will give you access to all the public methods in the library.

1. In the solutions pane, right click the references node and select "Add Reference..."
2. Click on the Projects tab
3. Click the Browse... Button
4. Browse to TestDll.dll and click OK
5. Add `using TestDLL;` to the top of From1.cs with the other using statements.

Build the project, click Built > Build Solution... on the menu.

This will create the application we will load into Holodeck that uses the new dll we have just created.

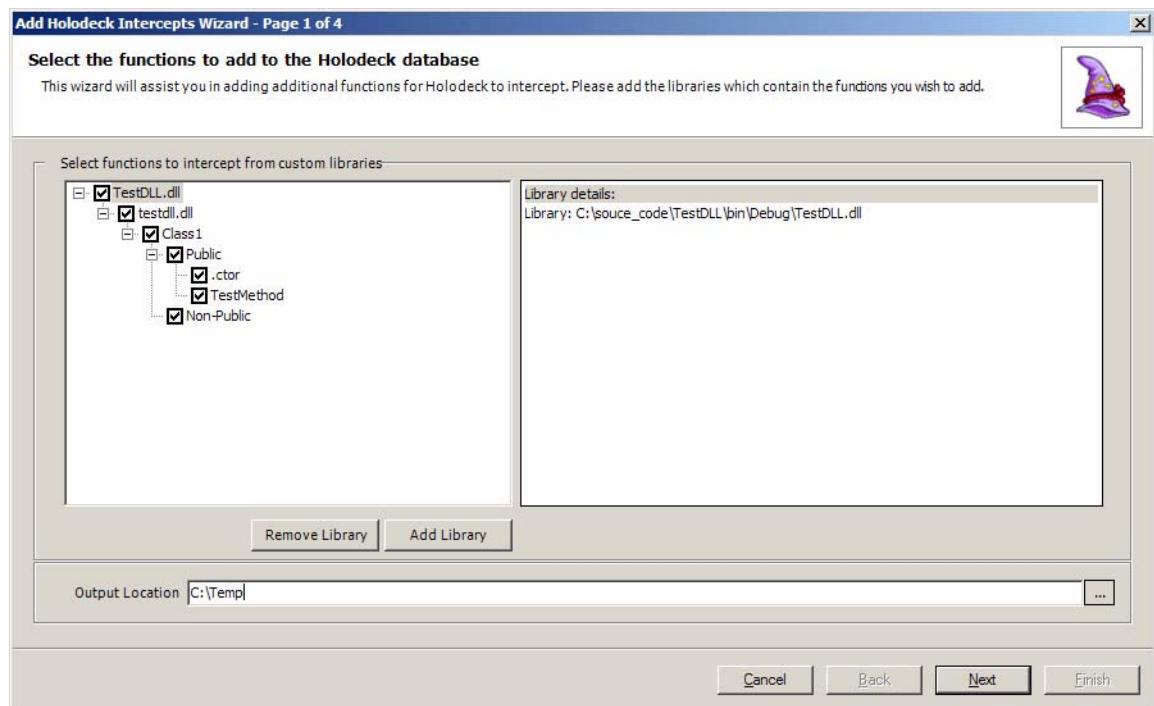
<< Previous Next >>

Add a New Intercept Library

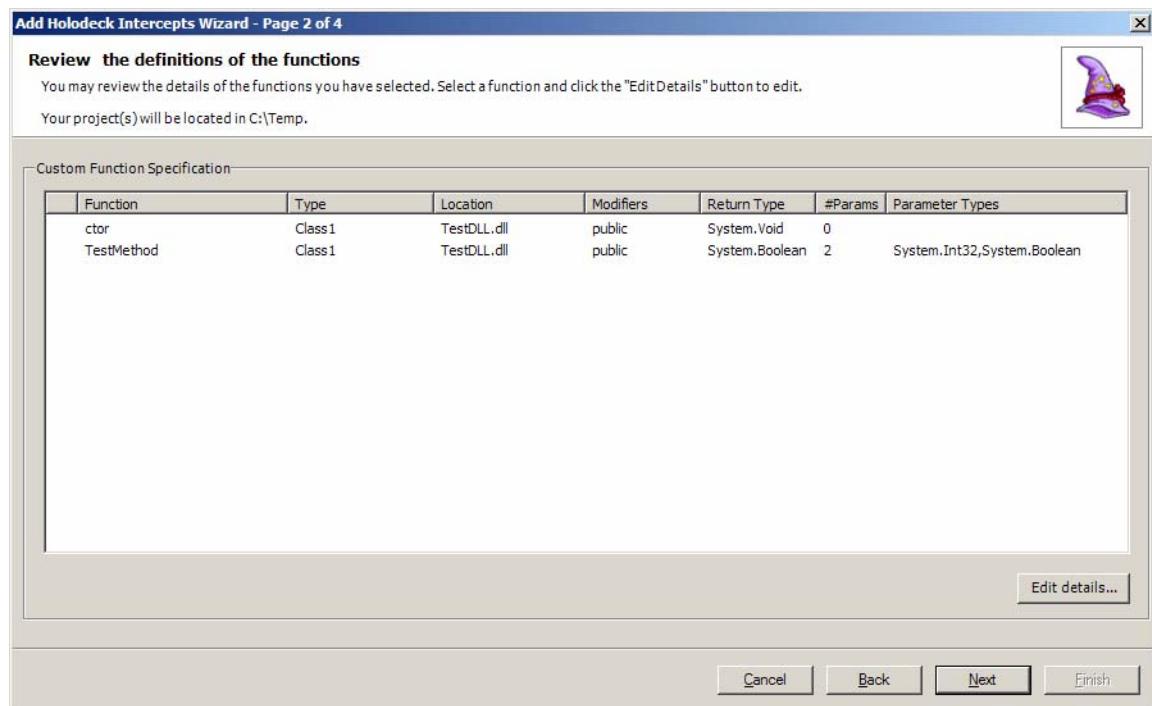
Launch Holodeck, once Holodeck has finished loading click Tools > "Add Holodeck Intercepts."

In the Add Holodeck Intercepts Wizard click Add Library, this will open an open dialog box where you can browse to the TestDLL.dll you created earlier. Click Open to load the library into the Add Intercepts Wizard.

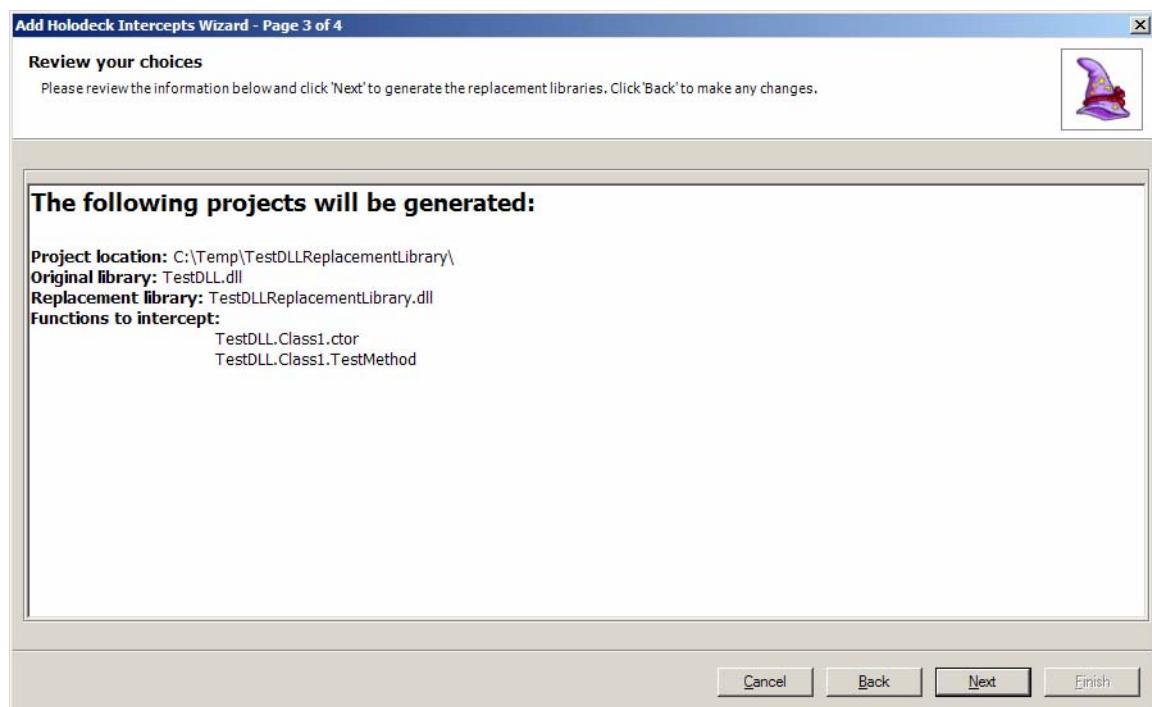
Check TestDll.dll in the treeview, select all the functions in the library for interception. Type C:\Temp in the output location text box, then click next:



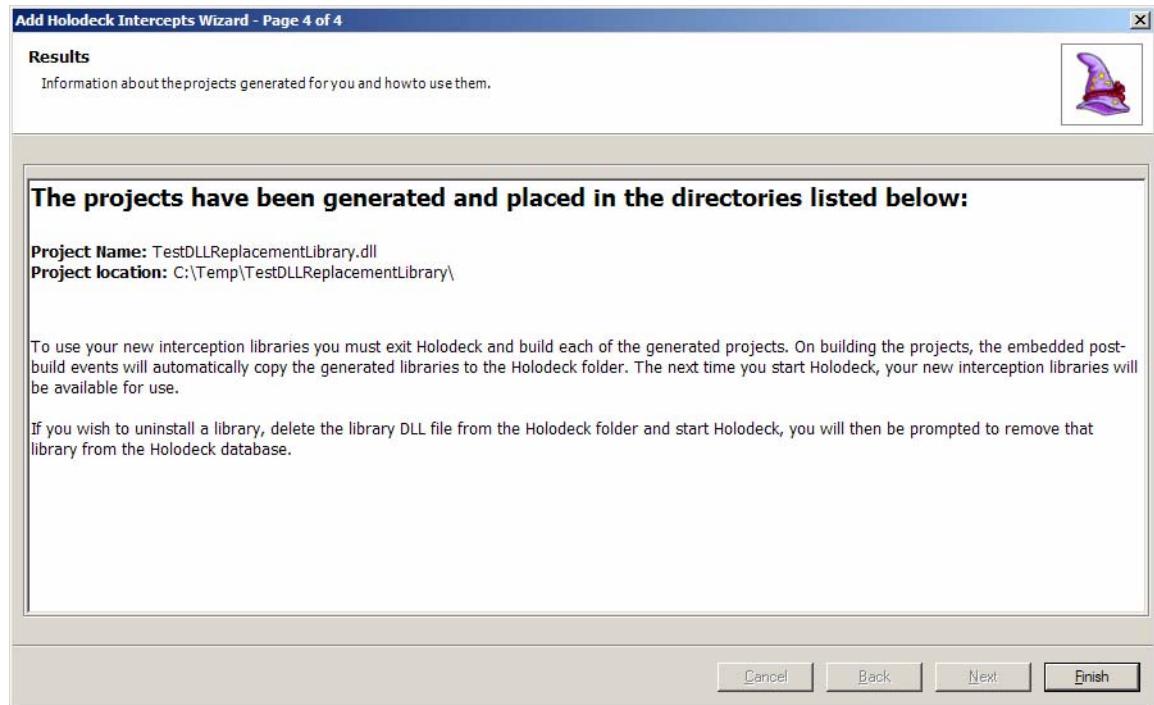
Holodeck has discovered all the information it needs to create this interception library, click next.



Holodeck reports back to make sure all the final choices are correct, after next is clicked on this page Holodeck will generate the project with the following functions to intercept.



Holodeck reports how and where each project has been created. In this case the project has been created in "C:\Temp\TestDLLReplacementLibrary\" click finish.



<< Previous Next >>

Compile the TestDLL Replacement Library

Open the project in Visual Studio.net, the project is located in "C:\Temp\TestDLLReplacementLibrary" double click the C# Project File to open the project n Visual Studio.net.

Click Build > "Build Solution" to build the replacement library. Answer yes when the dialog asks if you would like to add a registry key. This registry key tells Holodeck to regenerate its list of replacement libraries the next time it starts. The replacement library will be automatically added to the Holodeck replacement library list.

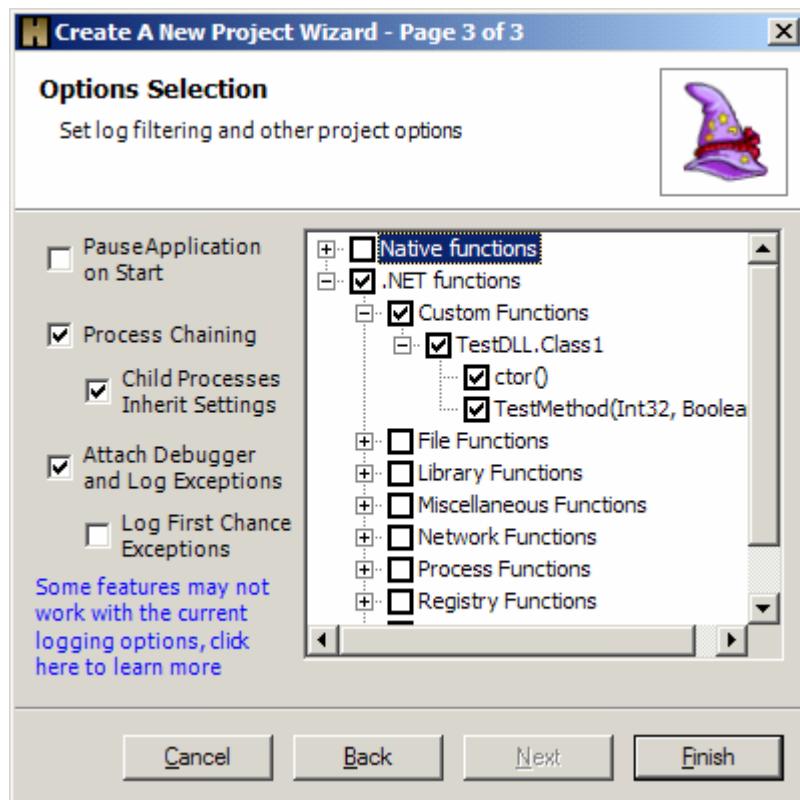
Relaunch Holodeck, wait for the .NET update to complete; the new intercept library has now been created and Holodeck will treat TestMethod just like it treats all the other system APIs that it intercepts out of the box.

<< Previous Next >>

View Logs and Create a Test for the Test dll

Create a New Project

Use TestApplication.exe as the application, turn off all logging except for .NET > Custom. Notice that TestMethod and the constructor for TestDLL are now in the logging treeview under custom.



View a Log

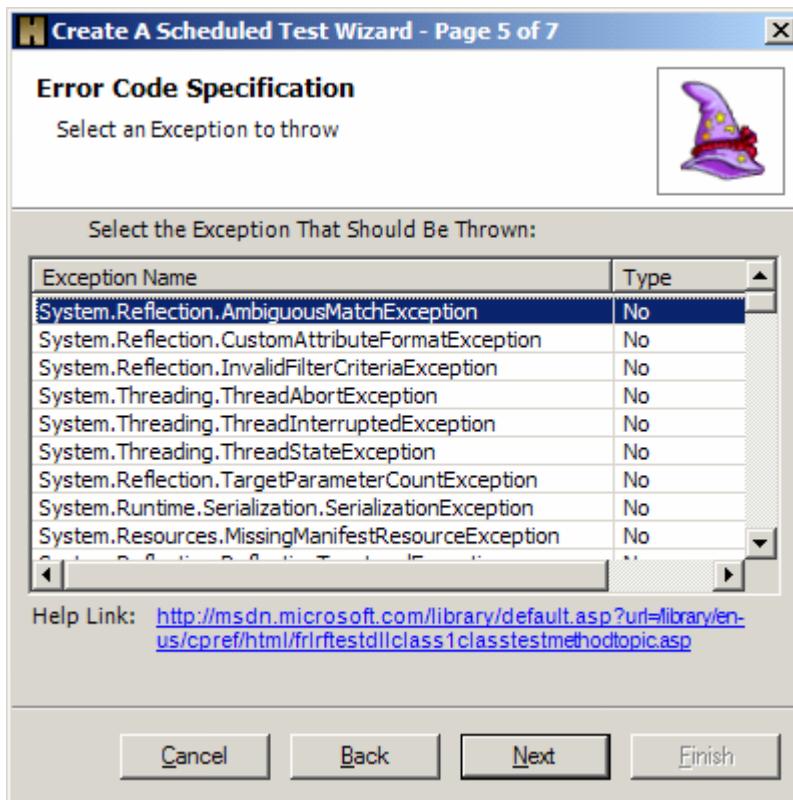
When TestApplication loads notice that no logs are visible in the Holodeck Log Pane, this is because it has not yet called the only function we are logging, the function in the TestDLL.

Click on the button in the TestApplication form. Notice that the constructor and TestMethod from TestDLL.dll are called by TestApplication and logged in the Holodeck log pane.

Welcome to Holodeck TestApplication.exe Log - 1088(All Threads) Faults - 1088(All Threads)							
TimeStamp	Thread	Category	DLL	Function	Return Value	Error Code	E
06/07/2004 15:53:52:437	2732	CUSTOM	TestDLL.dll	TestDLL.Class1.ctor()			
06/07/2004 15:53:52:437	2732	CUSTOM	TestDLL.dll	TestDLL.Class1.Tes...	True		
C:\source_code\TestApplication\bin\Debug\TestApplication.exe - Process 1088							
						Active	Entries: 2 Visible: 2

Create a Test

Double click on the call to test method in the Log Pane this will open the "Create a Scheduled Test Wizard" with the test application TestMethod already selected. No In-Parameter or Out-Parameter changes need to be made, however, specify "Ambiguous Match" as the .net exception to throw when the application calls TestFunction.



Click the button in the TestApplication again. Notice the unhandled exception dialog is shown and the log in the log pane shows an exception in the exception column. In this screen shot you can see the first two calls were executed properly, the last call to TestMethod was failed and threw the AmbiguousMatchException specified in the test.

Time	Thread	Category	DLL	Function	Return Value	Exception
07/2004 15:53:52:437	2732	CUSTOM	TestDLL.dll	TestDLL.Class1.ctor()		
07/2004 15:53:52:437	2732	CUSTOM	TestDLL.dll	TestDLL.Class1.Tes...	True	
07/2004 16:01:38:890	2732	CUSTOM	TestDLL.dll	TestDLL.Class1.ctor()		
07/2004 16:01:39:093	2732	CUSTOM	TestDLL.dll	TestDLL.Class1.Tes...	True	AmbiguousMatchException

<< Previous Next >>

Summary

This tutorial covered how to create an application that calls a library you created. An interception library was then created using Holodeck's Add New Intercept Library Wizard, which created a new intercept project. After an interception library is created Holodeck can intercept the methods of that library the same as it does with the native and .NET functions out of the box.

Further Exploration

Modify the test to fire 50% of the time, notice that it doesn't fire every time you hit the button, but randomly half the time.

Try creating a replacement library for a library function that the application you are testing is dependant upon and creating tests for it. When creating replacement libraries you don't have to create replacement functions for all the functions within the library.

Learn about custom faults and create a custom fault for the new library you are intercepting. This will set in stone a set of tests you can later turn on by going to the Faults Pane and clicking the radio button.

Try changing the test so it matches certain parameters. If you change the input parameters the test may not fire. This can be helpful if you only want a test to fire when certain parameters have been set.

[**<< Previous**](#)

Create a New Test Project

Create a New Test Project Introduction

In this tutorial you will learn how to inject your own test code into the application you are testing, opening up new and powerful test scenarios previously unavailable.

Create the Custom Test Project – Create the custom test project that will allow you to launch a new test application.

Use the Custom Test Project to Test Notepad – Compile the project, and setup the test.

Further Exploration – Modify the default values further to find other bugs within the application.

Note: The in the following tutorial notepad can easily be substituted for your application.

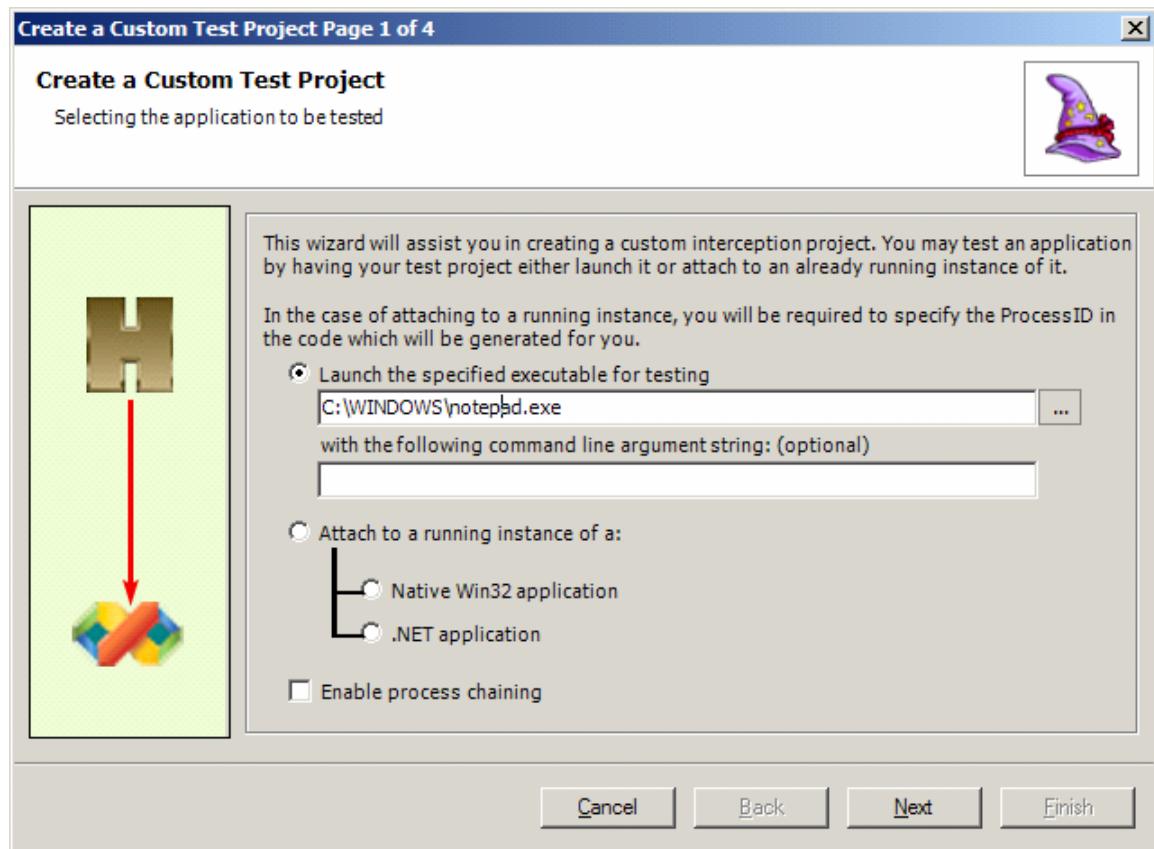
[Next >>](#)

Create the Custom Test Project

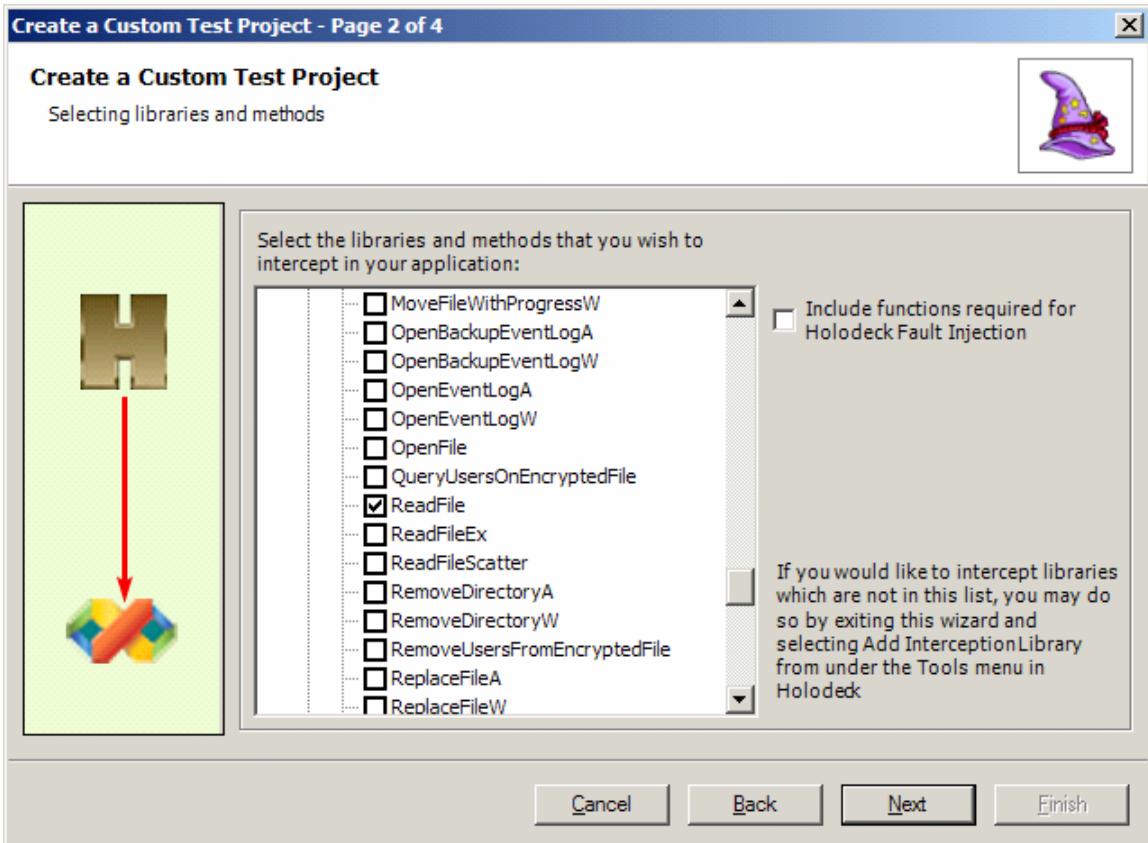
Launch Holodeck. Once Holodeck has finished loading click Tools > "Create a Custom Test Project" from the menu.

Select Notepad as the application to launch, and then click next.

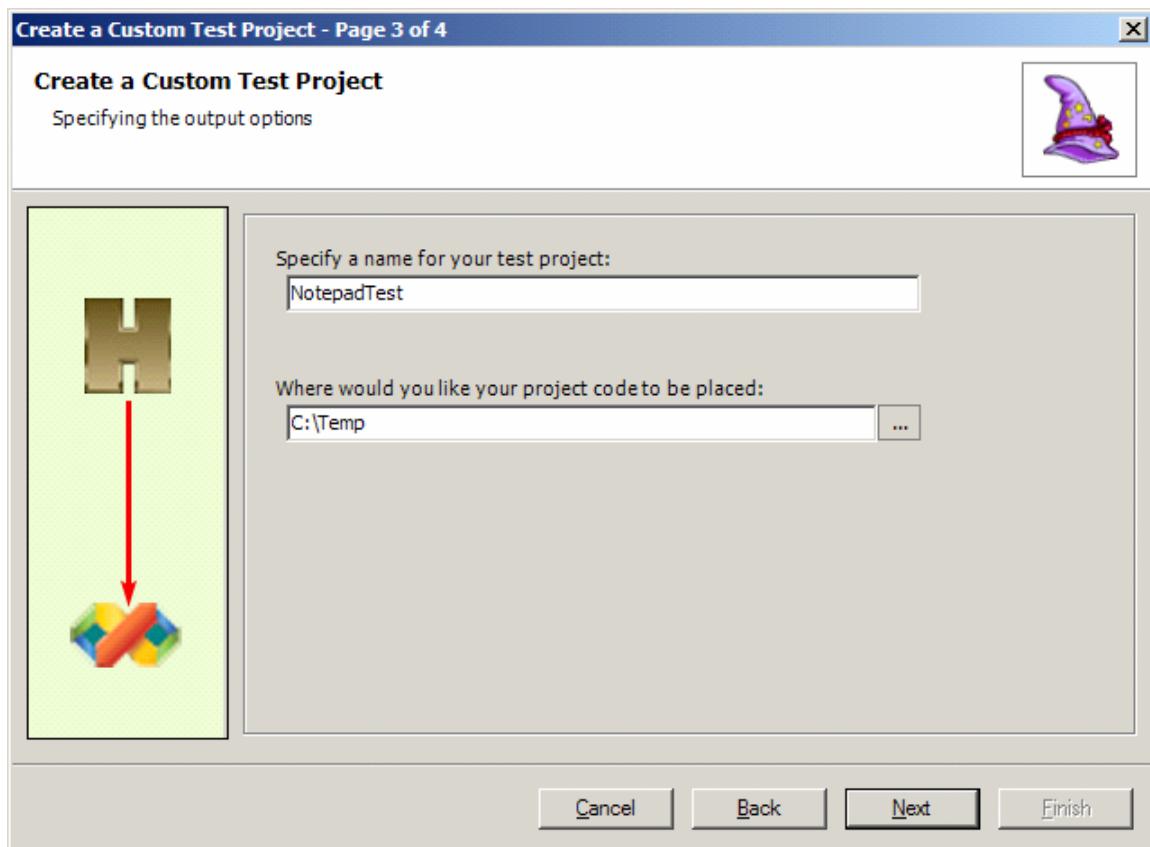
Notepad is being used in this tutorial for ease of demonstration, the techniques in this tutorial could be used on any application.



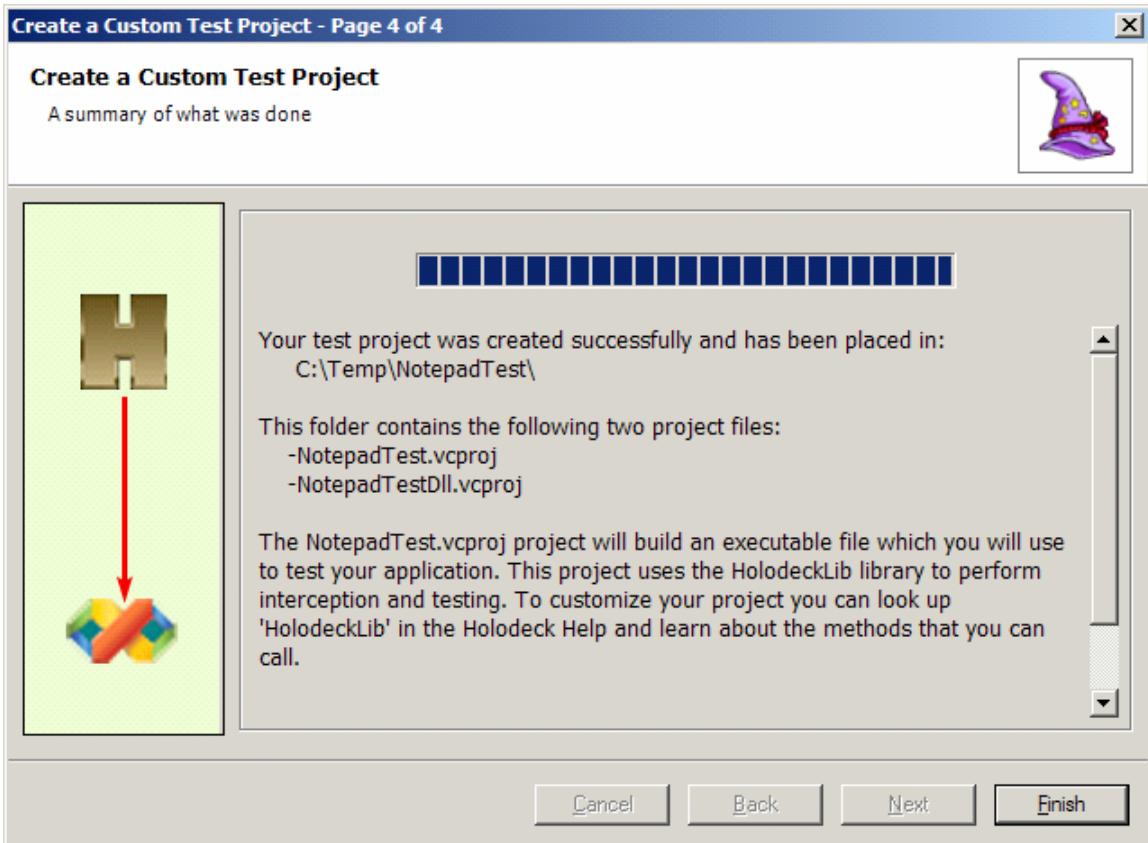
Expand the File Functions node in the treeview, then select ReadFile as the function to intercept and click next.



Specify "NotepadTest" as the name for the project, and place the project in "C:\Temp." Click next.



Holodeck has successfully created the test project, Click finish and close Holodeck.



<< Previous Next >>

Use the Custom Test Project to Test Notepad

Create the interception dll

Open the notepadtestdll.vcproj file from the "C:\temp\Notepadtest" directory.

Open the NotepadTestdll.cpp file and find the method "ReadFileReplacement." This will be called first at anytime notepad makes a call to readfile. We have the chance to add any test code we want and then pass on the parameters to the real ReadFile or just return control to notepad.

Add the following code to the top of the ReadFileReplacement function.

```
MessageBox(NULL, "Test is about to run", "", MB_OK);  
  
nNumberOfBytesToRead += 100;
```

This will trick ReadFile into thinking the buffer notepad has passed it is 100 bytes larger than it really is and will force a buffer overrun. This test case will cause the application to crash, however it is not a valid test. It is rather a quick and easy demonstration to show the power of injecting your test code into the intercept-path.

Build the notepadTestdll project by clicking Build > "Build Solution"

Launch Notepad

Open the notepadtest.vcproj file from C:\temp\notepadtest, compile the project and run it. Click Build > "Build Solution" this will create the application to launch. Run the application by clicking Debug > "Start" or by pressing f5. Once the application has launched the call to LaunchApplication will start notepad.

Once Notepad has finished loading, make a call to the library we have chosen to intercept. Click File > Open on the menu in Notepad, notice your dialog is displayed because notepad has made a call to the ReadFile function which we are intercepting. After you click OK to the dialog the forced buffer overrun test will execute and Notepad will crash immediately.



<< Previous Next >>

Summary

You can use Holodeck's Custom Test Project to launch any application and intercept function Holodeck has been set up to intercept, including custom libraries created with the Custom Intercepts Library Wizard. In this tutorial you created a simple project that launched notepad, but the same principles can be applied to testing any application.

Further Exploration

Try modifying the amount of bytes added to the buffer. Remove the message box and add just one byte; run notepad and see what happens. Subtract bytes and see what happens.

Add a custom test project for the application that you are testing and use it to create custom test scenarios that run without the need for the Holodeck UI. You can use the application the Custom Test Project creates to launch the application from a test harness. This allows you to control the application through the test harness while API interception takes place programmatically within the Custom Test Project.

[**<< Previous**](#)

Holodeck in depth

Introduction to Holodeck in depth

This section covers each feature in Holodeck in detail. Research how to test your application more fully by understanding how to use the features of Holodeck completely.

Each of the following chapters contain all the advanced features of Holodeck.

New Project Wizard – Holodeck can launch or attach to applications and services.

Creating Tests – Scheduled tests allow you to fail a single API call based on that function's input and output parameters.

Creating Limits – Holodeck can limit physical resources such as memory, hard drive space, and network bandwidth your application has available to it; use this to simulate different hardware configurations.

Creating Faults – Faults are common failures a system might have, such as an invalid address for memory, or a network disabled error for the network.

Resource Faults – Resource faults are faults that simulate problems accessing or loading a resource such as a process, library, file, folder, registry key, registry value or COM Object.

Working with Logs – Holodeck logs all the API's you select. You can filter, sort, change which functions are logged, export and more.

Working with Reports – Holodeck can generate an easy to read report on your current project. This report contains information for every process in the project including logs, faults, limits, etc..

Using Automatic Test Generation – Holodeck can automatically generate tests, limits, and faults for you through the use of code coverage or stress test generation.

Generating Network Corruption Faults – The Network Corruption Fault Wizard will help you create corruption within network packets.

Generating File Corruption Faults – Holodeck helps test file parsing by corrupting files.

Using regular expressions and Replacement strings – Regular Expressions and Replacement strings can be used in conjunction with Network and File Corruption Faults.

Multiple Threads and Processes – Holodeck can be focused on a single thread or follow multiple processes.

Exceptions and Mini-dumps – Holodeck can log exceptions and create mini-dump files to help debug the application.

Recording and Replaying Sessions – Holodeck allows you to record the tests Holodeck makes to help you reproduce bugs and verify failures.

Add New Holodeck Intercepts – You can add new intercepts to Holodeck easily to help you intercept custom libraries.

New Project Wizard

New Project Wizard

The New Test Project Wizard walks you through the steps to create a new project. Holodeck allows you to launch an application to test, attach to an already running application, launch a service, or attach to an already running service.

Launch an Application – Launch an application to test an application. This option allows you to test startup functions.

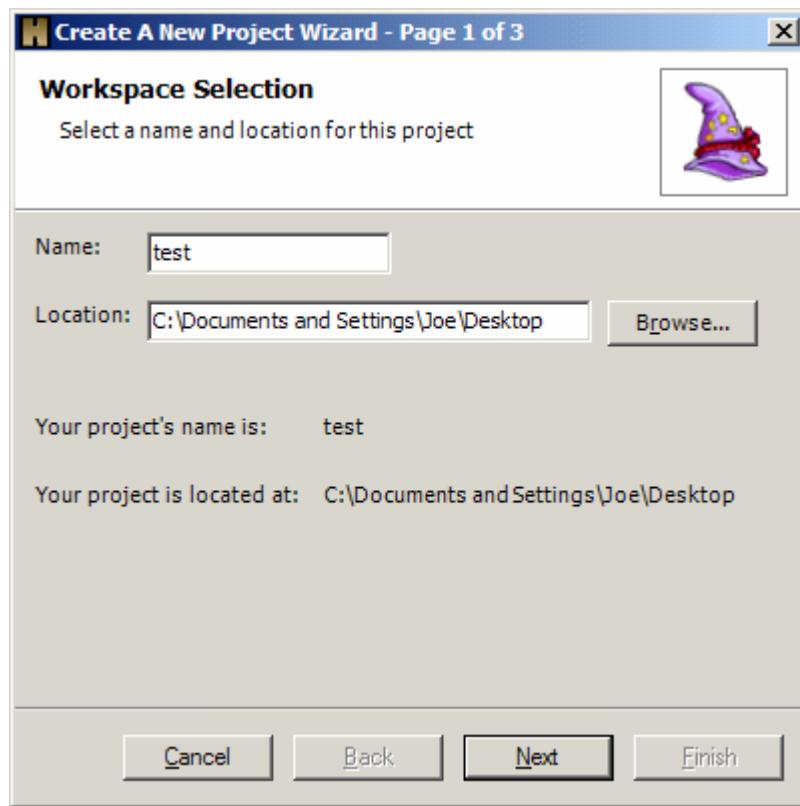
Attach to an Application – This option gives you the ability to attach Holodeck to an already running process.

Start a Service – Launch a service, all services which are enabled and not currently running are available for launch.

Attach to a Service – All services that are currently running are available to be attached.

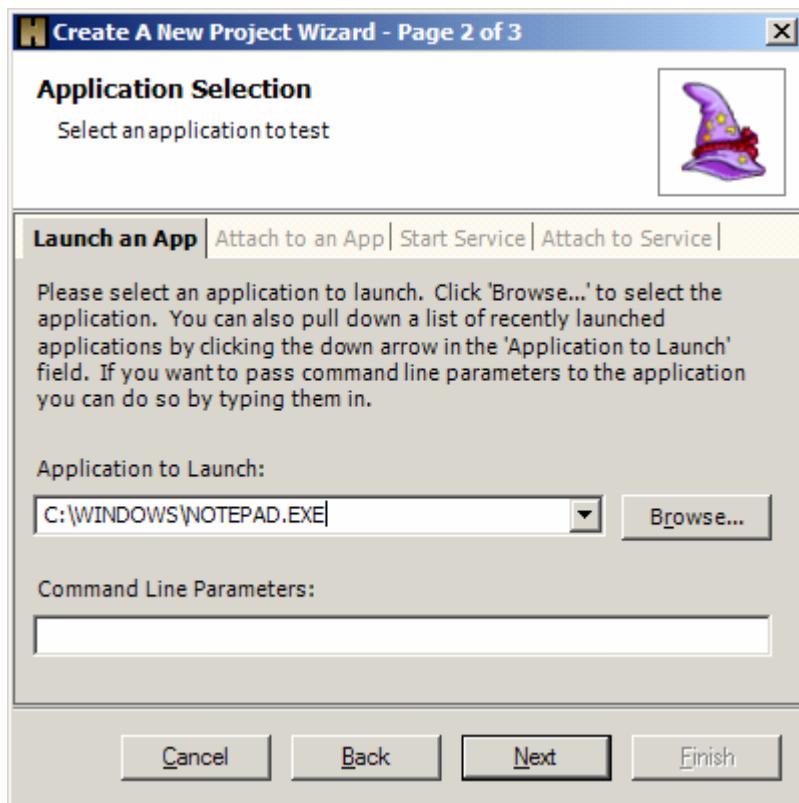
New Project Options – The final page of the Wizard allows you to select how Holodeck monitors your application or service.

Choose one of the above links for more information on that creating a project of that type.



Launch an Application

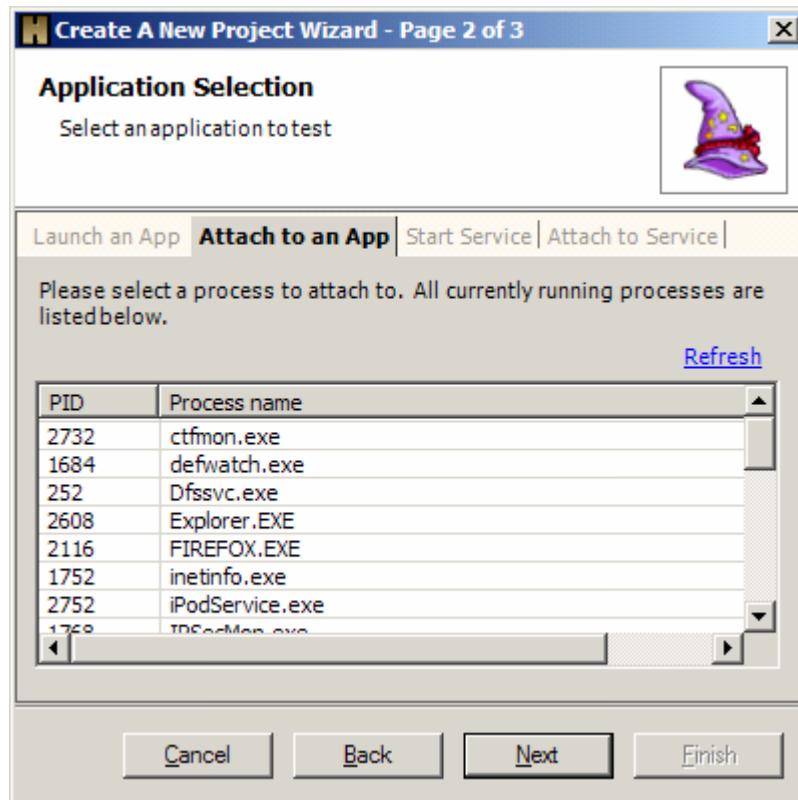
Use this option to test an application that is not currently running. You can pause the application on start to set tests, limits, etc. before the application loads, which enables you to test the startup of the application. If your application requires command line parameters you can specify them in the Command Line Parameters text box.



Continue to the New Project Options to complete the wizard.

Attach to an Application

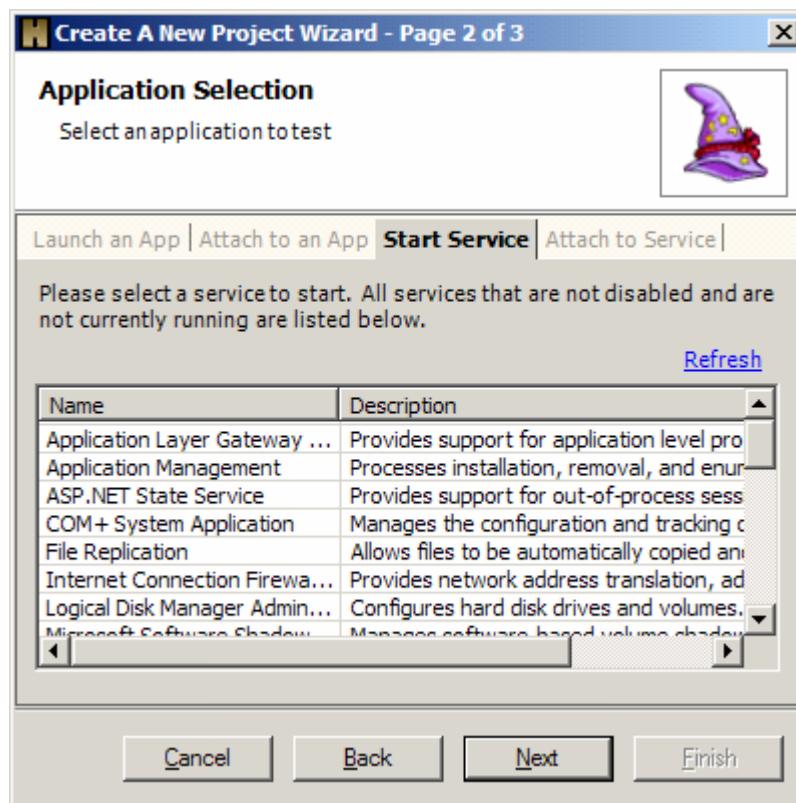
Use this option to attach Holodeck to a currently running application. This is useful for testing applications that start up with the operating system, run in the background, or are required to run for normal operation of the computer.



Continue to the New Project Options to complete the wizard.

Start a Service

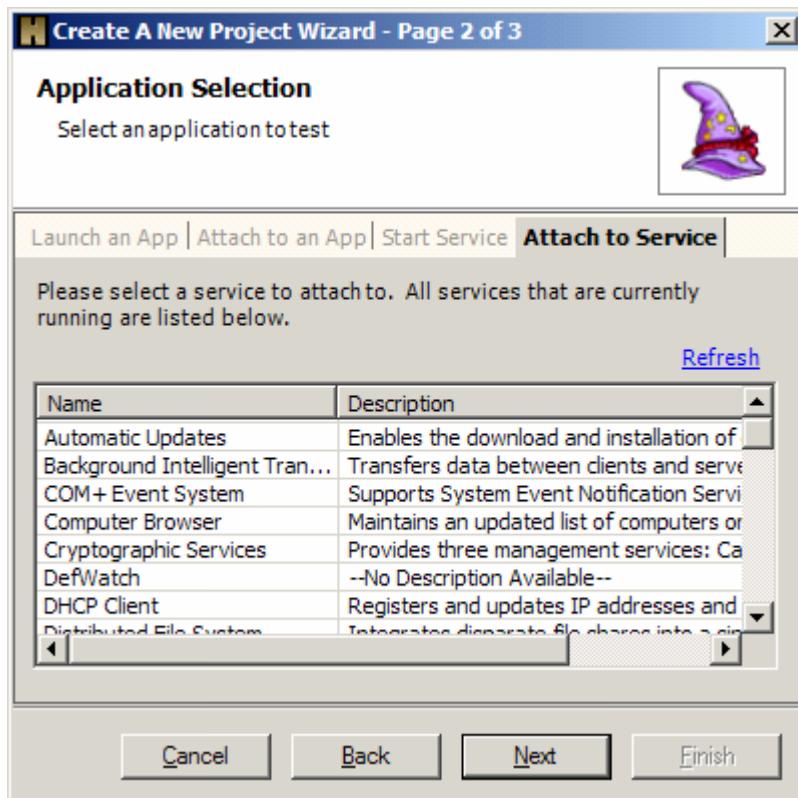
Holodeck can help test services including system services and application services. Holodeck will launch the service and begin logging immediately.



Continue to the New Project Options to complete the wizard.

Attach to a Service

You can attach Holodeck to a Service that is already running. Holodeck will begin logging API function calls as soon as the wizard has completed.



Continue to the New Project Options to complete the wizard.

New Project Options

This page allows you to set options on how Holodeck monitors your application. Turning on more function logging will increase the number of tests and faults that are available, however more logging may impair the performance of your application. For more information on working with logs please see the Logs Overview help topic.

You can pause your application on start to set tests, limits, faults etc. before your application starts up. If you are attaching Holodeck to an already running instance of an application or service Holodeck will pause the application or service when the wizard has finished; to resume execution of your application uncheck Application > Pause Application from the menu.

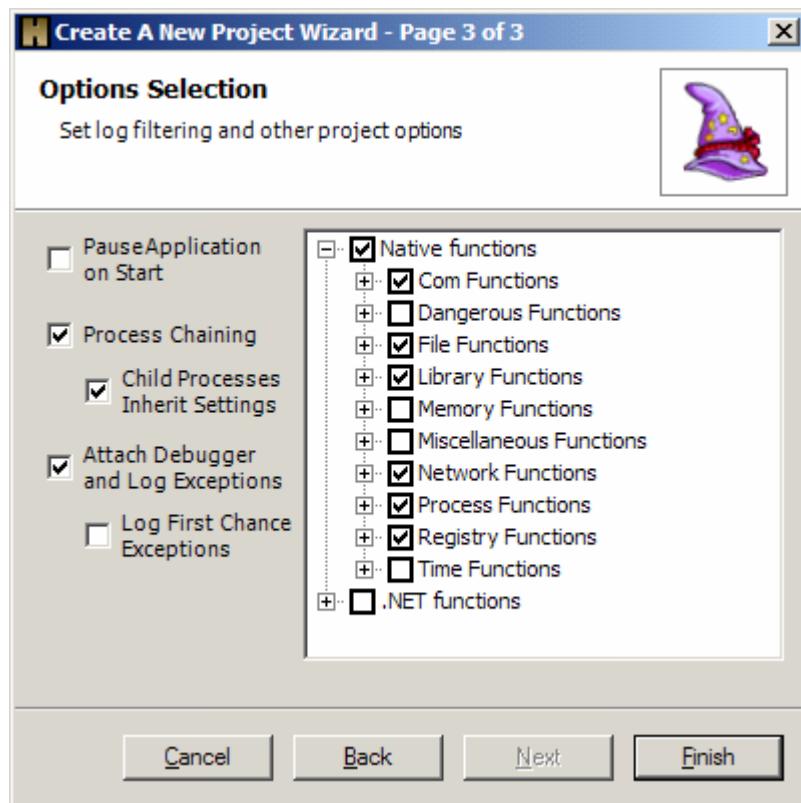
If Process Chaining is turned on Holodeck will automatically attach itself to any processes your application launches. If you check Child Process Inherit Settings, any process your application launches will automatically inherit the same tests, limits, faults etc. that were set for the parent application. For more information please see the Dealing with Multiple Processes help topic.

Holodeck comes with a built in Debugger which catches exceptions and creates mini-dump files. Check "Attach Debugger and Log Exceptions" to enable this feature.

If any of the following categories are not logged some of the features of Holodeck may not work properly or as expected, for more information please see the Default Logging help page. Function Category: COM Functions, File Functions, Library Functions, Network Functions, Process Functions or Registry Functions

If your application produces first chance exceptions that might help you debug the application upon a crash you can enable the logging of First Chance Exceptions. If you enable this option when your application produces a first chance exception Holodeck will generate a compressed mini-dump file that you can use to debug your application in Visual Studio. For more information regarding Exceptions and Mini-dumps please see the Exceptions and Mini-dump help pages.

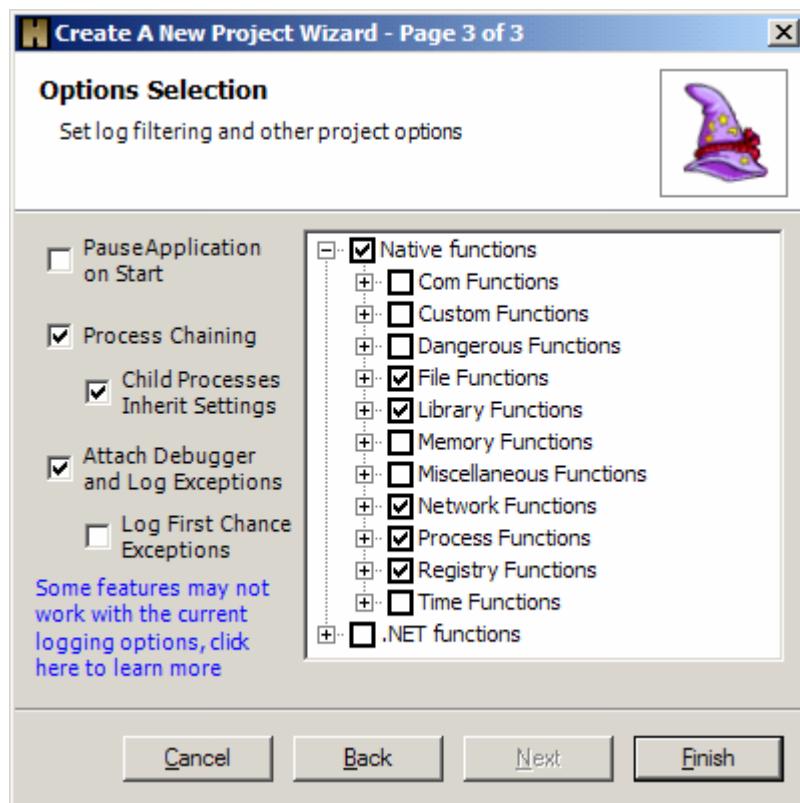
Note to Windows 2000 users: Windows 2000 is incapable of detaching a debugger already attached to an application. Thus if the Holodeck Debugger option is enabled, it isn't possible for Holodeck to detach from the application. For more information please see the Windows 2000 and attaching the Holodeck Debugger help topic.



Default Logging

Some of the features of Holodeck require you to intercept and log certain API categories. If you choose to turn off these categories the following features will be disabled, in addition to the function no longer being logged in the Logs Pane.

Function Category	Feature(s) Disabled
COM Functions	COM resources will not show up in the Resource Pane
File Functions	Files will not show up in the Resource Pane. Corrupted files will not show up in the Corrupted files pane, although file corruption will continue to work. Only the Holodeck UI is affected.
Library Functions	Library resources will not show up in the Resource Pane
Network Functions	Network Logs will be disabled
Process Functions	Process resources will not show up in the Resource Pane
Registry Functions	Registry resources will not show up in the Resource Pane



Creating Tests

Tests Overview

Tests are a way to inject targeted failures; a test targets a single API call based on matching the input parameters you specify. If the test firing criteria is matched Holodeck will change the function's output parameters and return value to match the test's specifications.

This is a good way to test specific function calls to find out what might happen if external API calls fail.

Creating and Deleting a test – Overviews the different ways to create and delete a test in Holodeck

Scheduled Test Wizard – This wizard will help walk you through the steps to create a scheduled test.

Modifying a test – Once a test has been created you can go back to modify it.

Creating a Test Based on Stack Matching – Holodeck allows you to fire tests based on which functions are currently on the stack, or only fire a certain percentage of the time.

How to tell if a test fires:

You can tell if a test has fired by looking in the API logs. If the test has fired you will see the error-code and return values set by the test.

You can see how many times a test has fired by selecting the test in the project pane, and looking to the Properties Pane.

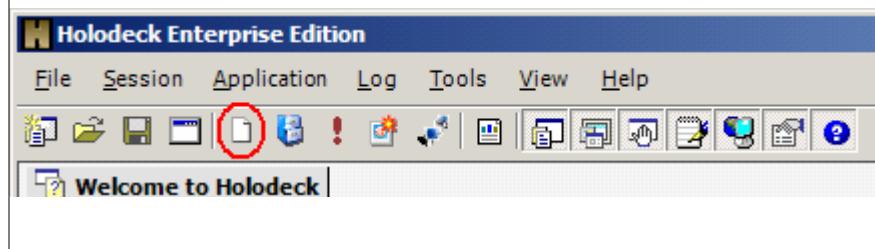
Creating and Deleting a test

You can create a **New Scheduled Test** by any of the following methods

- 1) From Log entries
 - a) Double Click any Log entry to open the Scheduled Test Wizard

Note: this is a good way to create a scheduled test because it then loads the Wizard with the correct API function selected.

- 2) From the Context menu.
 - a) Click the **New Scheduled Test Button**

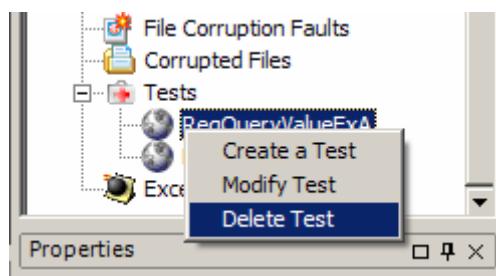


- 3) From the Menu
 - a) Click **Application > Create a Scheduled Test**
- 4) From the Project Pane
 - a) Right click the Tests Node under the Application for which you would like to create a test.
 - b) Select Create a new Test.

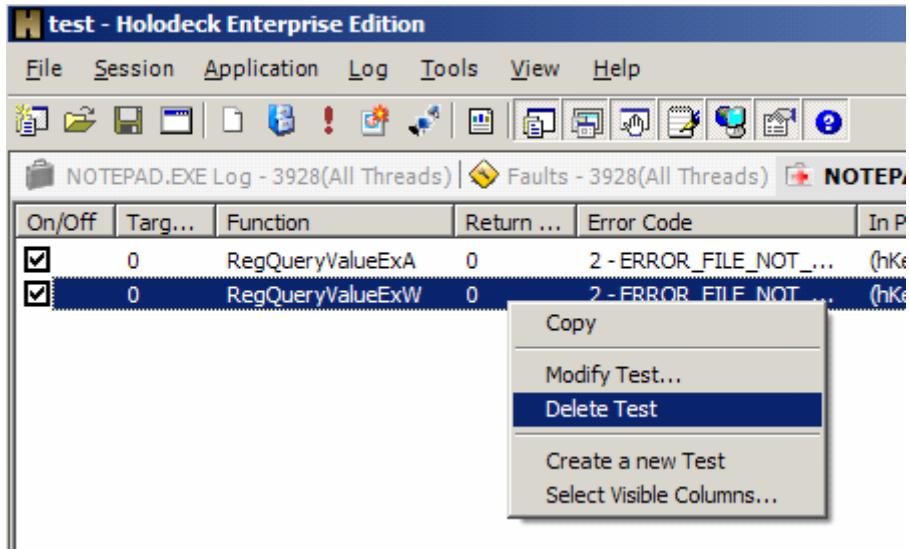
Note: if you create a test in per-thread mode you can create a test for a single thread. If you are in per-thread mode and create a test from the main log window the test will apply to all threads; conversely, if you are in per-thread mode and you create a test in one of the thread specific API logs the test will apply only to that thread.

Deleting a test

To delete a test, find the test in the project pane, right click and select Delete test.



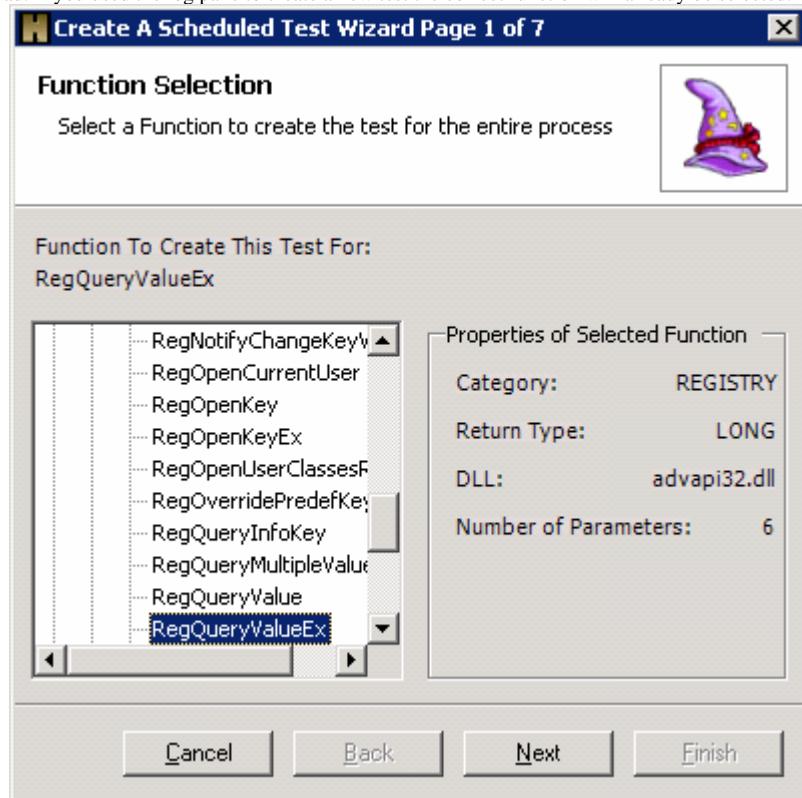
You can also delete a test by right clicking the test in the tests pane and selecting Delete test.



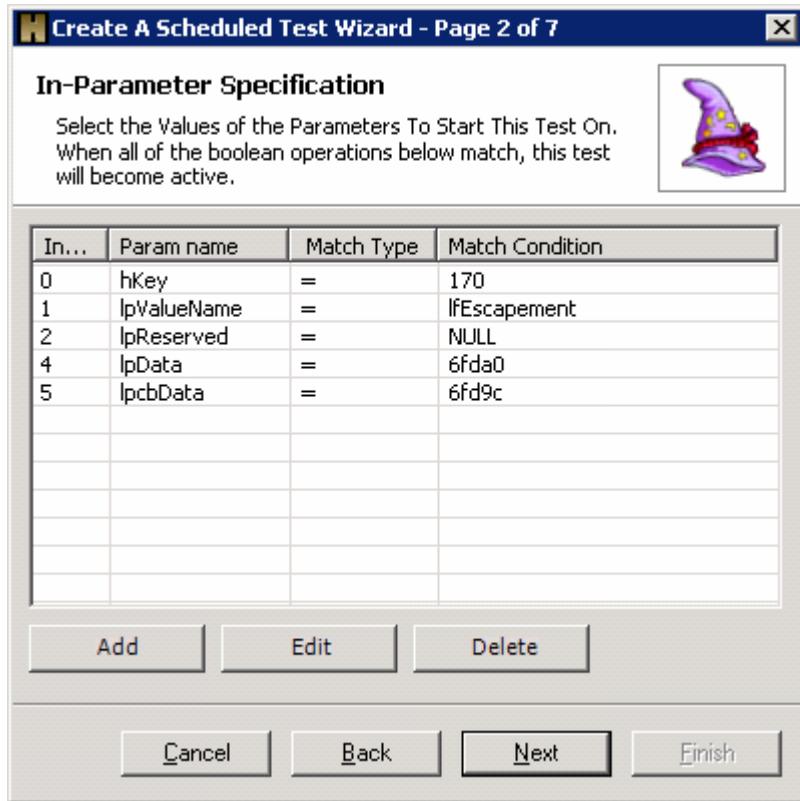
Scheduled Test Wizard

This wizard will help you fail a certain function call.

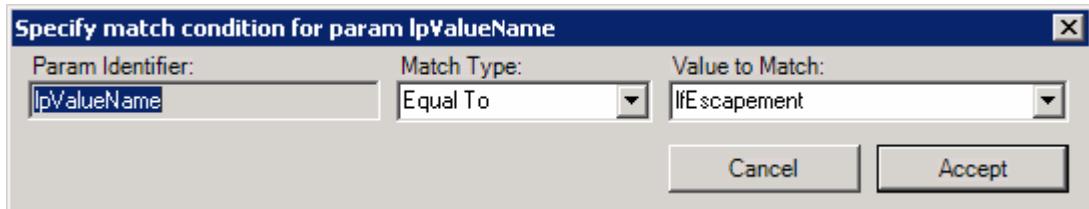
- 1) The Function Selection Window allows you to select a function to create the test for the entire process or a single thread. If you used the log pane to create a new test the correct function will already be selected.



- 2) The In-Parameter Specification Window allows you to select the Values of the Parameters that when matched will cause this test to fire. When all of the Boolean operations below match, this test will fire. In-parameters are most likely sent from the application under test, which make it easy to fire a controlled test this way.



To edit the in parameter specifications highlight a parameter and click edit. The following dialog will appear.



In the Value to match drop down you can type new values to match on, or use one of the special values.

Special Values:

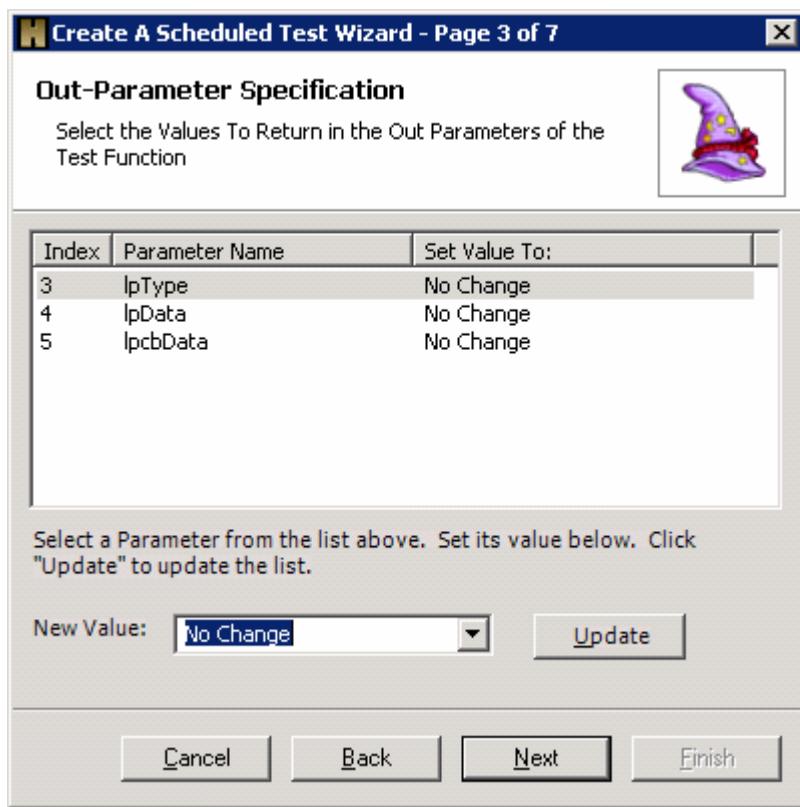
- {NULL} – A null parameter.
- {TRUE} – A Boolean true value
- {FALSE} – A Boolean false value
- {ANY} – Any value

You can match the values in a number of ways, including the following:

- Equal To – The parameter equals the value to match.
- Not Equal To – The parameter does not match the value to match.
- Less Than – The parameter is less than the value to match.
- Greater Than – The parameter is greater than the value to match.
- Less Than or Equal To – The parameter is less than or equal to the value to match.
- Greater Than or Equal To – The parameter is greater than or equal to the value to match.
- Starts With – The parameter is compared to the value to match by a string comparison, and starts with the value to match.
- Ends With – The parameter is compared to the value to match by a string comparison, and ends with the value to match.
- Contains Substring – The value to match is contained anywhere in the parameter.
- Doesn't Contain Substring – The value to match is not contained anywhere in the parameter.
- Equal to Resource Path – The parameter equals the resource path, based on a string comparison with the full path based on resource stitching, handle matching, etc. for the parameter.
- Not Equal to Resource Path – The parameter does not equal the resource path, based on a string comparison with the full path based on resource stitching, handle matching, etc. for the parameter.

- Contains Resource Path – The parameter contains any part of the resource path, based on a string comparison with the full path based on resource stitching, handle matching, etc. for the parameter.
- Doesn't Contain Resource Path – The parameter does not contain any part of the resource path, based on a string comparison with the full path based on resource stitching, handle matching, etc. for the parameter.
- Starts with the resource Path – The parameter starts with the resource path, based on a string comparison with the full path based on resource stitching, handle matching, etc. for the parameter.
- Binary Contains – The parameter and value to match are cast to their binary representation, and'ed together and compared to the value to match (i.e. (parameter & value) == value). This is useful if the parameter is a flag that can have values or'd together, use binary contains to and the value out.
- Binary Not Contains – The parameter and value to match are cast to their binary representation, and'ed together and compared to the value to match, matches if this test fails.

3) The **Out-Parameter Specification Window** allows you to select or change the values to return in the out parameters of the test function. When the test fires Holodeck will replace whatever the original out-parameter value was with the out-parameter value you specify in this page of the wizard. Often the application under test may assume certain values for each out-parameter, this is a likely place to find bugs. By changing the out-parameters to unexpected values you can ensure your application will fail properly if there is an error in the API call, or the API is called incorrectly.

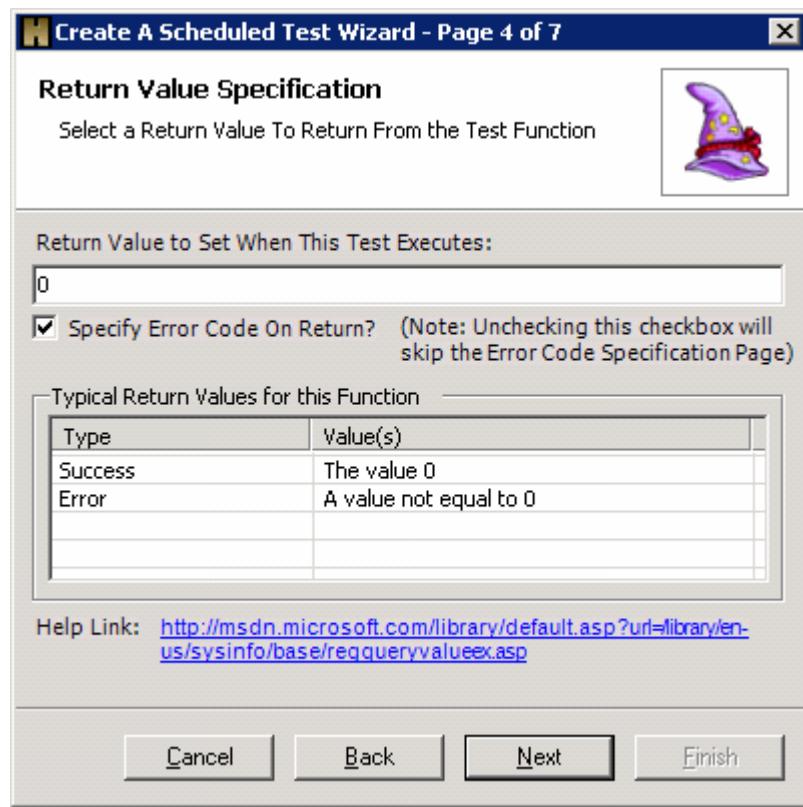


4) The **Return Value Specification Window** allows you to select a return value to return from the test function.

Check the "Specify Error Code On Return" Checkbox if you want to specify an explicit error code when the function returns.

Return values that are typical of this function are the return values that occur under most circumstances, including error values and success values.

The Help link takes you to the MSDN help page regarding the selected API function.

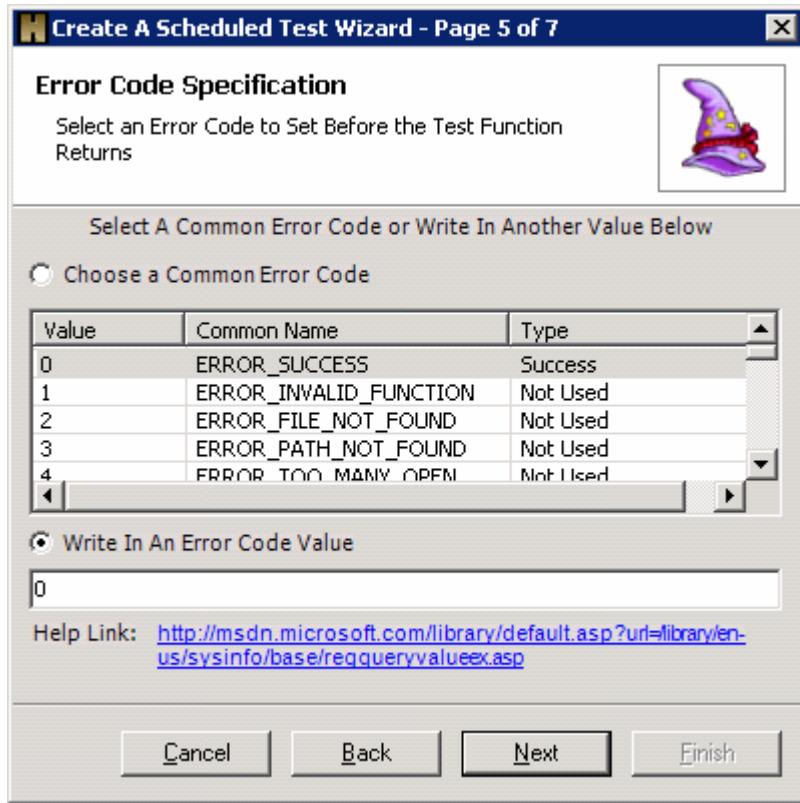


5) The **Error Code Specification Window** allows you to select an Error Code to be Set Before the Test Function Returns. Holodeck will set the specified error code using the SetLastError system call. Doing this will make the application under test believe the last API call had some error, which could uncover bugs in the application code.

Select the "Choose a Common Error Code" radio button if you wish to select an error code from the list.

You can write in your own error code in the Error Code Text box. If you write in your own error code it must be a whole number greater than or equal to zero.

The Help link takes you to the MSDN help page regarding the selected API function.



6) The **Test Firing Specification** allows you to change the amount a specific test fires during the run of the Application Under Test. Often an error does not happen every time an API call is made. On this page you can specify how often a test fires. It may be useful to specify a certain application state such as a function on the call stack.

You can have the test fire a certain percentage of the time. Holodeck will fire the test randomly the set percentage of the time.

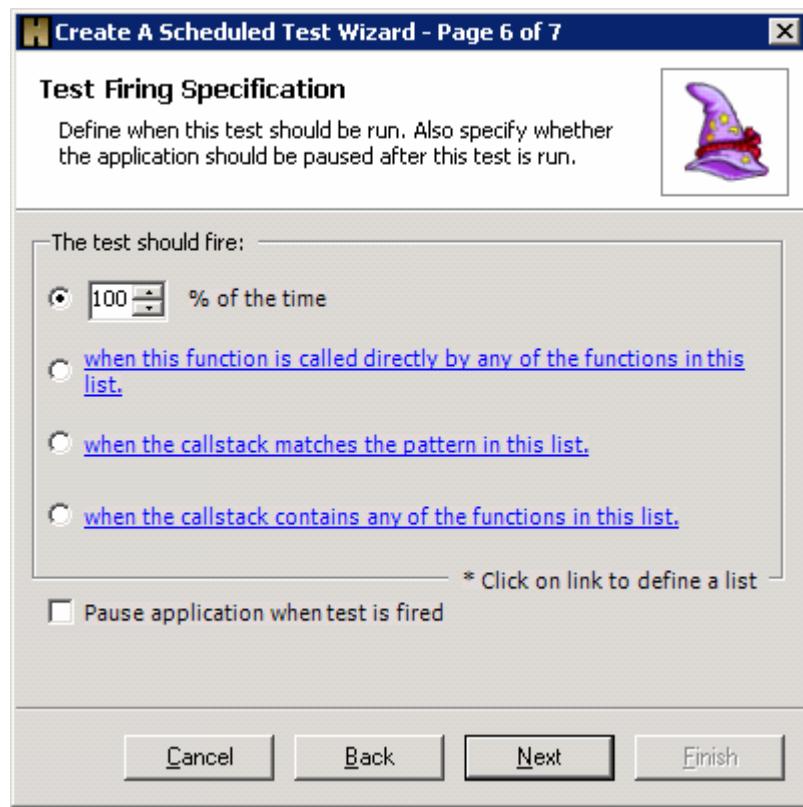
Select the second option to have this test fire whenever it is called by another test. Specify which function will fire the test by following the link

You can specify a specific callstack pattern so the function will only fire when the functions you specify are on the stack in the order you specify.

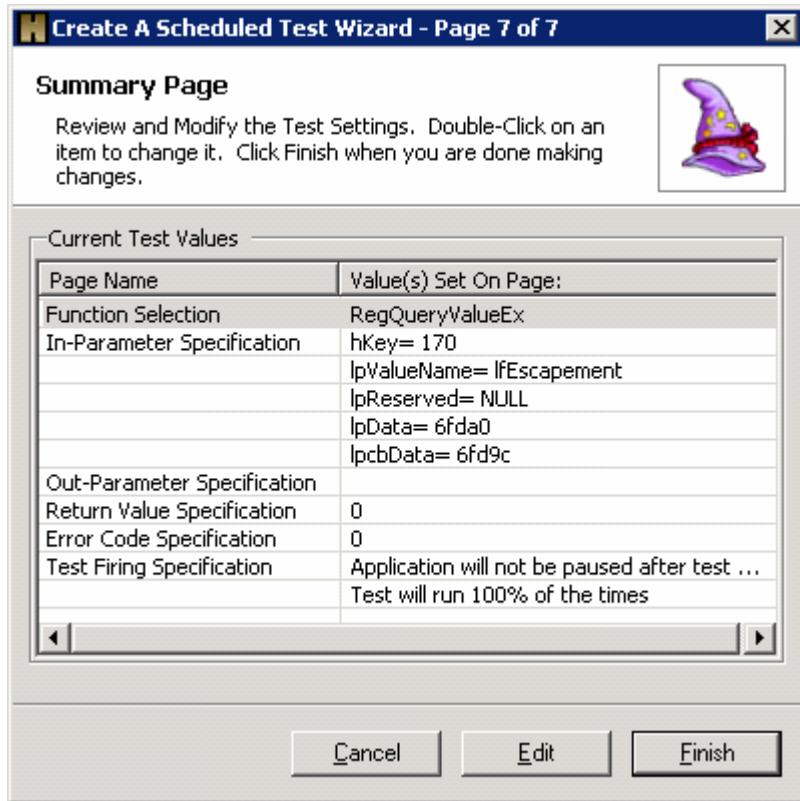
You can have the test fire when any of the function you specify are on the stack with the last option.

Please see the Creating a Test Based on Stack Matching for more information on this subject.

Pause the application when the test is fired to set other tests, faults limits etc. while the application is paused.

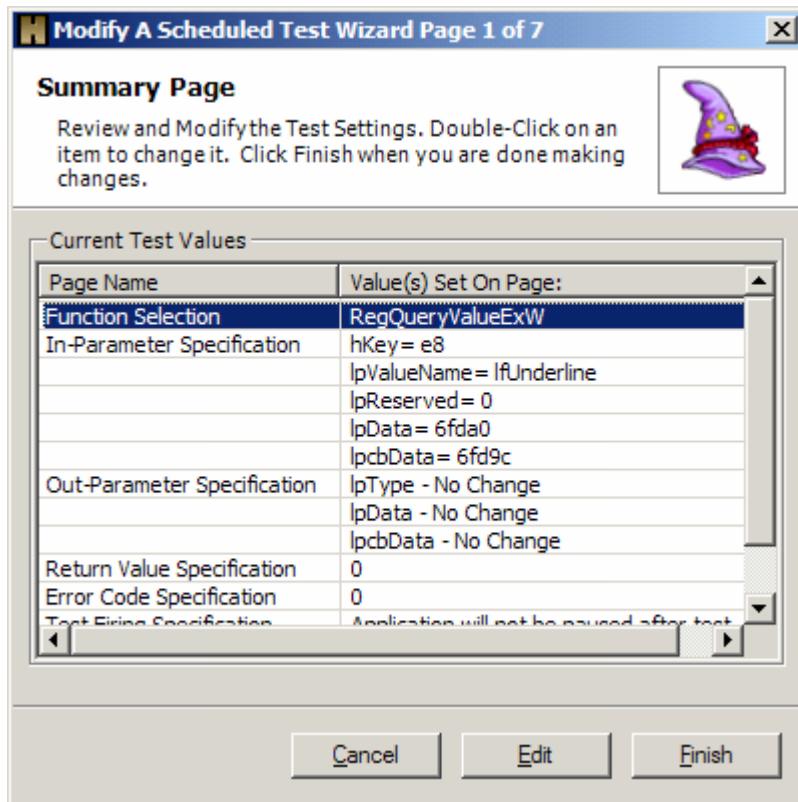


- 7) The **Summary / Modifications Page** allows you to review and Modify Test Settings. If you wish to change any of the settings on this page, highlight the setting you would like to change and click edit. This will return you to the page of the wizard where that information can be changed.



Modifying a test

Modifying Tests - Right click the test you wish to Modify in the Project Pane and click Modify Test or double click the test you would like to modify. This will bring up the Summary / Modifications page of the New Scheduled Test Wizard



To modify any part of the test simply select the page name in the table and click edit, this will bring you directly to that page of the wizard.

Creating a Test Based on Stack Matching

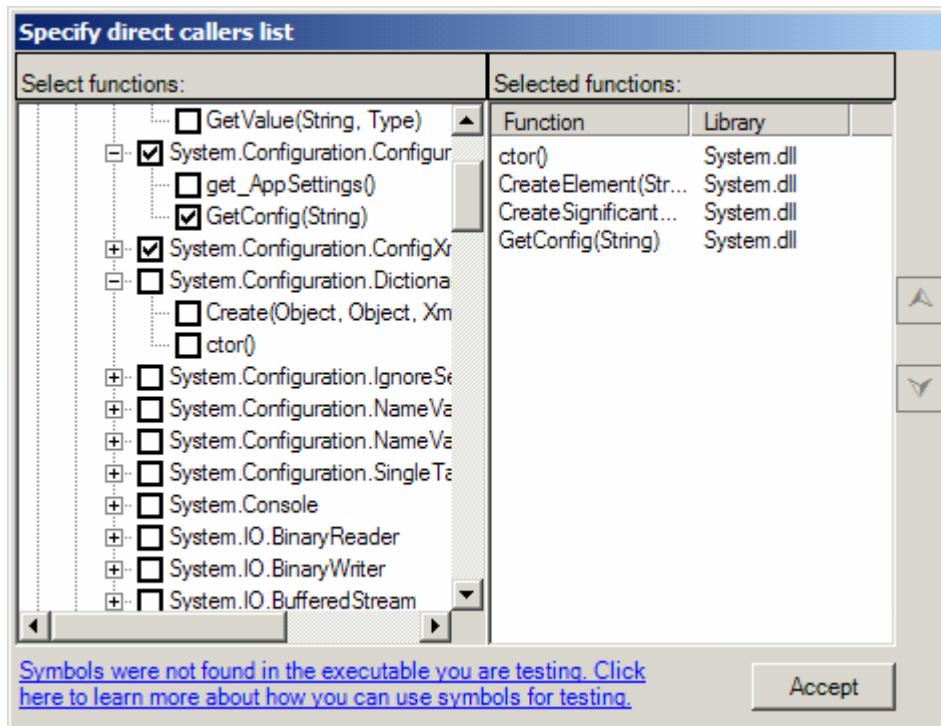
When testing a debug version of the application Holodeck can read the symbols and give you the ability to fire a test only when certain functions are on the system stack. You can match on functions internal to the application as well as any other intercepted functions.

For example:

With symbols information, a test could be created which would only fire if the function under test was called from a particular function inside the application.

This can be very useful when trying to focus a test to fire only when a certain system state is present, ie. a certain function has called the function for which the test was set.

Add functions to the selected function list on the right by checking the function. Change which order the functions will be matched on the stack by clicking the up or down arrows to the right of the Selected Functions column.



Creating Limits

Limits overview

In Holodeck it is possible to view how much of the disk or memory resources the Application Under Test is using and to restrict those resources to simulate different test scenarios without having to physically change the memory or disk availability on the test machine. These limits include Disk Limits, Memory Limits, and Network Limits. Use these limits to simulate different hardware configurations without having to change your system hardware. Use the limits pane to set and remove limits from your test application.

Limits:

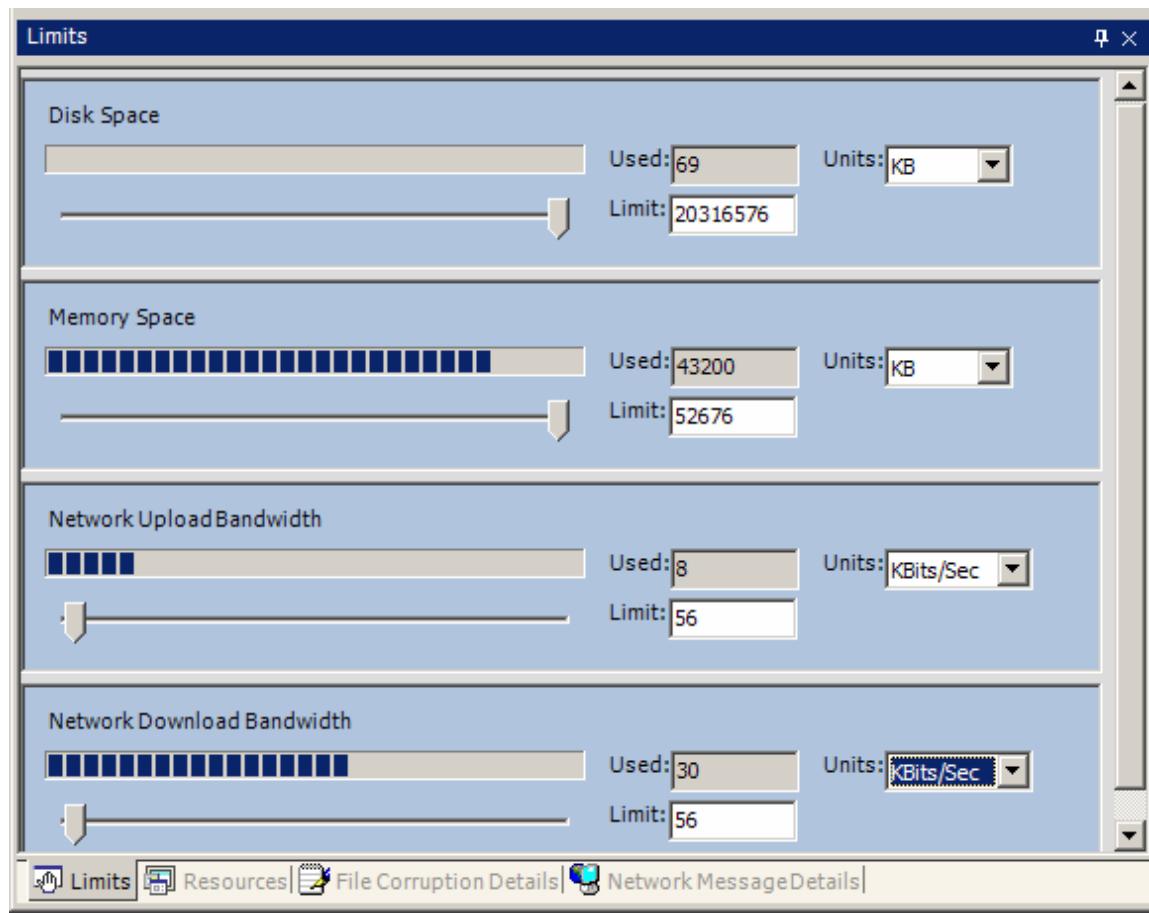
To view the amount of disk or memory resources currently being used by the Application Under Test make sure Holodeck is configured to show the Limits pane by clicking View on the Menubar and making sure Limits Pane has a check beside it.

Once the Limits Pane has been selected under the View menu, click the Limits tab at the bottom of the screen in the Limits Pane. This will bring up the Limits pane which shows the total amount of each resource used. If a limit has been set the limit will be shown on the slider-bar and in the limit text box. Each resource can be limited to see how your program will handle under limited resource conditions.

For example if you wanted to see how your application dealt with a small amount of memory you could slide the slider bar down until Holodeck was limiting the memory resources to extreme levels. This is useful for finding out how your application reacts to low memory conditions.

The Used Text Box contains how much of a resource is in use by the Application Under Test.

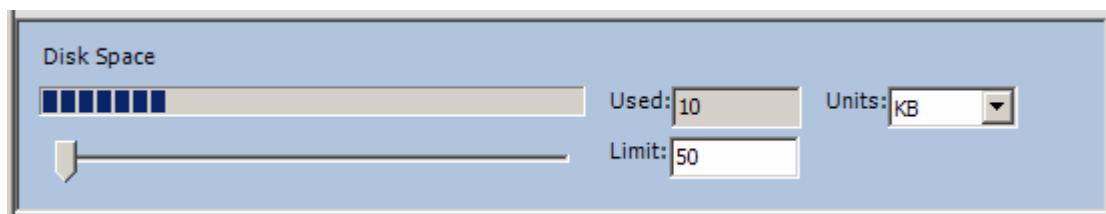
Note: If there is no limit set Holodeck will display the hardware limit which will change over time based on how much memory or disk space is being used.



Creating a Disk Space limit

This limits the amount of Hard Drive Space available to the Application Under Test. Using this feature you can test the Application Under Test's ability to handle errors such as Low or Filled Disk Space.

Open the limits pane by clicking **View** and making sure there is a checkmark by the **Limits Pane**.



Create a limit:

With the Limits Pane visible use the slider bar or the text box labeled **Limit** to set a limit, or type in a specific disk limit to the disk limit text box. You can choose to display the available and limited disk space in Bytes, Kilobytes, or Megabytes.

Note: The minimum limit for Disk Space is the amount the Application Under Test is currently using.

Remove a limit:

To remove a limit, simply slide the slider bar all the way to the right, or delete the node in the project pane.

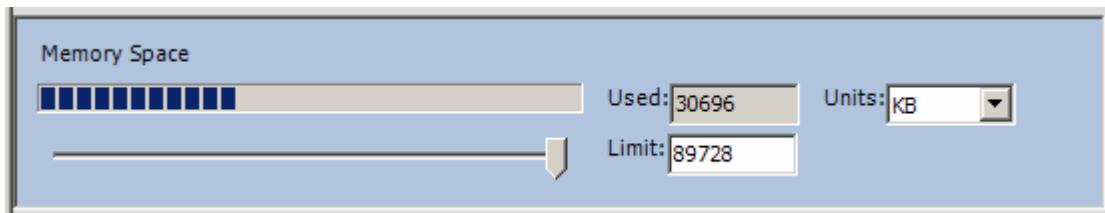
How Limits and Resources are calculated:

The amount used is updated through the monitor of API calls that allocate disk space. The total disk space amount is then incremented by the allocation request amount in these API calls. Before each request for disk space is permitted, Holodeck checks to ensure the amount being allocated is below the set limit. If the Application Under Test goes above the limit the API call requesting disk space is failed. If a file is moved and the size is smaller than the original file size the amount used is decremented and the limit won't fire. Holodeck uses win32 API's to get the max disk space available. Percent used is calculated using the disk space used through the API calls and the max disk space available. When a limit is set the max disk space available is artificially reduced and the progress bar is recalculated (percentage of limit, instead of percentage of hardware capacity).

Creating a Memory Space limit

This limits the amount of Memory space available to the Application Under Test. Test the Application Under Test's ability to function under low memory conditions using memory limits.

Open the limits pane by clicking **View** and making sure there is a checkmark by the **Limits Pane**.



Create a limit:

With the Limits Pane visible use the slider bar or the text box labeled **Limit** to set a limit, or type in a specific memory amount. You can choose to display the available and limited memory space in Bytes, Kilobytes, or Megabytes.

Note: The minimum limit for Memory Space is the amount the Application Under Test is currently using.

Remove a limit:

To remove a limit, simply slide the slider bar all the way to the right, or delete the node in the project pane.

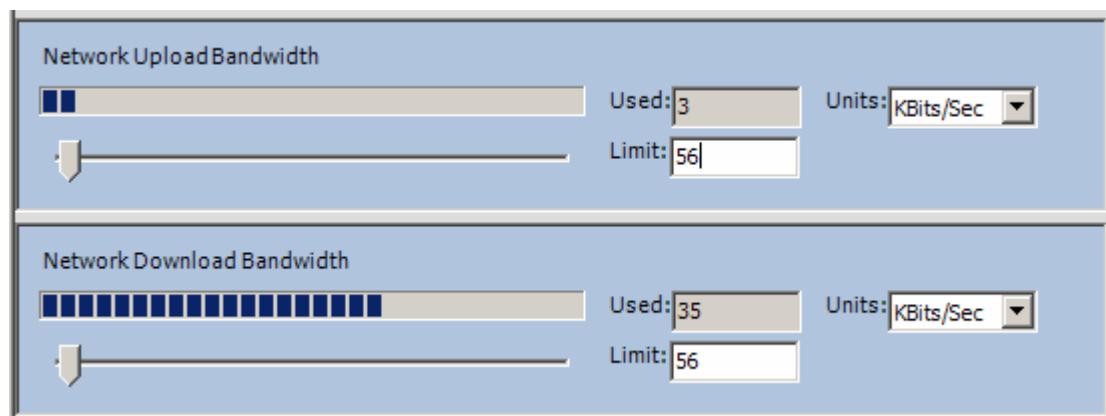
How Limits and Resources are calculated:

The amount used is updated through the monitor of API calls that allocate memory. The total memory amount is then incremented by the allocation request amount in these API calls. Before each request for memory is permitted, Holodeck checks to ensure the amount being allocated is below the set limit. If the Application Under Test goes above the limit the API call requesting memory is failed. Holodeck uses win32 API's to get the max memory available. Percent used is calculated using the memory used through the API calls and the max disk space available. When a limit is set the max memory available is artificially reduced and the progress bar is recalculated (percentage of limit, instead of percentage of hardware capacity).

Creating a Network Bandwidth limit

This feature is useful for network dependent applications; simulate any network connection speed, or completely stop your network connection for a moment to see if the Application Under Test can recover. You can have different upload and download limits in a single project by changing either the Network Upload Bandwidth or the Network Download Bandwidth.

Open the limits pane by clicking **View** and making sure there is a checkmark by the **Limits Pane**.



Create a limit:

With the Limits Pane visible use the slider bar or the text box labeled **Limit** to set a limit, or type in a specific network speed. For greater precision you can choose between bits, Kilobits, and Megabits per second.

Remove a limit:

To remove a limit, slide the slider bar all the way to the right, or delete the limit node from the project pane.

How Network Limits work:

Holodeck intercepts all Winsock APIs, among which are functions which send and receive network data. By buffering the incoming and outgoing data Holodeck is capable of limiting the speed of the network to a user specified value. If this specified value for speed is greater than what a network connection can handle, the network speed will automatically be set to the maximum speed which the connection is capable of handling.

note: upload and download limits work for all stream based sockets.

Creating Faults

Faults Overview

Faults are preset error conditions that will help you test your application easily. There are six categories including one to create your own faults to test your application specifically. Faults set and fail an array of tests that simulate common errors.

For instance, the "File Not Found" fault fails all calls that have to do with copying, deleting, moving, replacing, searching, opening and operations having to do with attributes for files. These faults simulate the same errors the operating system would throw.

Adding and Removing a fault – Learn each of the ways to add and remove faults.

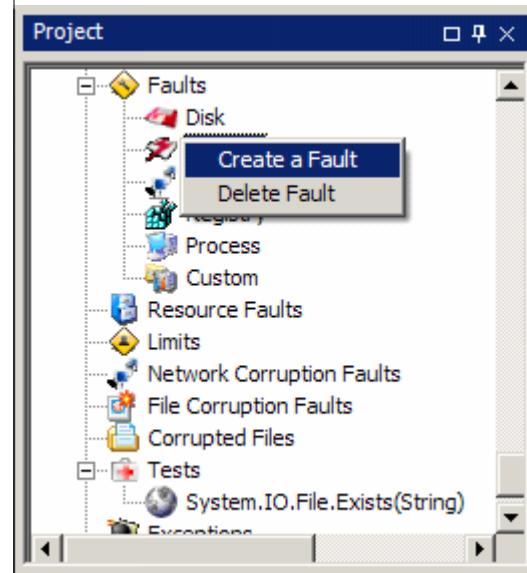
Adding custom faults – Holodeck allows you to create your own faults, learn how to create faults focused to fail a set of API functions you specify.

Types of faults – a list of the types of faults and a description of each category.

Adding and Removing a fault

To add a Fault simply select the fault in the Faults Pane.

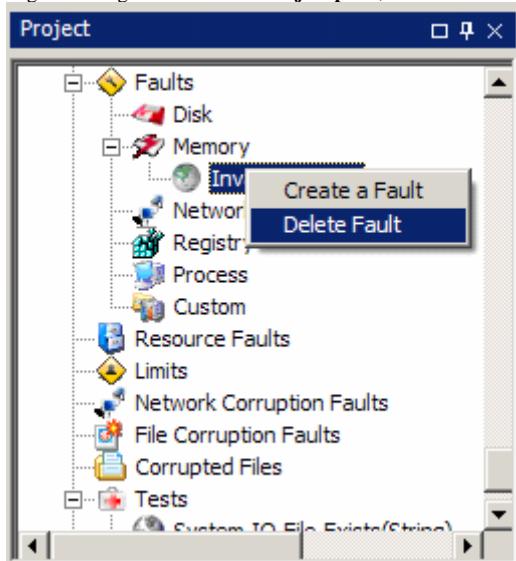
- 1) Click the Create new Fault button on the Project Toolbar or right click one of the fault nodes in the Project pane and select "Create a Fault".



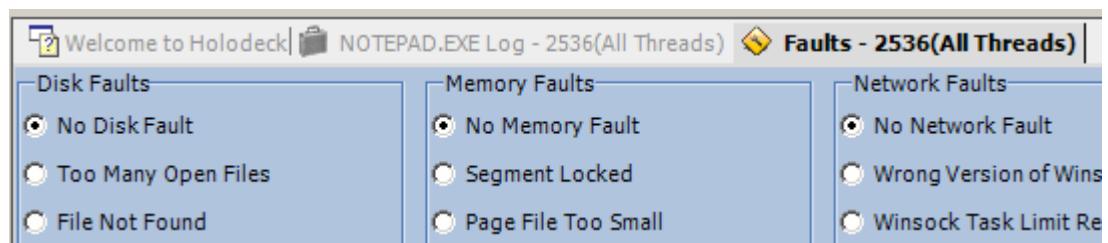
- 2) With the Faults pane visible select up to one fault from each category.

You can remove a fault by either:

Right clicking the fault in the **Project pane**, under **Faults** and selecting **Delete Fault**



Or selecting No Disk Fault, No Memory Fault, No Network Fault etc. in the Faults pane.



Adding custom faults

You can create your own faults to fail a specific set of API functions simply by editing the faults.xml file located in the function_db folder within the Holodeck install folder (default: C:\Program Files\Security Innovation\Holodeck Enterprise Edition\function_db).

Open faults.xml in a text or XML editor, and within the beginning and ending <Faults> tag insert your custom fault as follows.

Example:

This XML snippet, when added to faults.xml will create a fault named "My Fault"

```
<Fault Name="My_Fault" Type="Custom" ReturnValue="0" ErrorCode="112">
  <Function Name="CopyFileA" Allocation="GT">
    <CheckResource ParamIndex="1" Exists="2"/>
    <MatchParams>
      <MatchParam Name="bFailIfExists" TestOperator="=" TestValue="1"
        CompareAsType="3" ID="2"/>
    </MatchParams>
  </Function>
</Fault>
```

The fault name will show up in the faults pane as "My Fault" under the Custom column, Holodeck converts underscores '_' to spaces.

Unless the function tag overrides the setting, all function in this fault will use a returnValue of 0 and an ErrorCode of 112

Each function to be failed has to have its own <Function> tag, to have this function fail under every circumstance simply use the closed tag form (<Function Name="CopyFileB" />)

In this case the function to be failed is the "CopyFileA" API function, and will only fire when this function tries to allocate more space than it had originally had.

The CheckResource tag checks to see if the resource with handle 1 does not exist.

The MatchParams tag allows you to only fire when certain parameters are matched. In this case this test will only fire if the parameter named bFailIfExists matches the test value, 1, exactly. Holodeck will compare the actual value and the TestValue as a Boolean. ID is the zero based index of the parameter to match.

In summary CopyFileA will fail only if the filename we are copying to does not yet exist, the parameter bFailIfExists equals 1 and the file we are copying is larger than 0 bytes (ie. Allocation is increasing)

<Fault> tag – the main tag for the custom fault

Attribute	Description	Example	Required
Name	The Name of your custom fault	My_Custom_Fault	Required
Type	Which category this will show up. Valid values are: Disk, Memory, Network, Registry, Process, Custom	Custom	Required
ReturnValue	The value to return when the API call is failed	0	Required
ErrorCode	The error code to set using the setlasterror system call	8	Required

<Function> tag – the individual function to fail

Attribute	Description	Example	Required
Name	The Name of the API function to fail	LoadLibraryA	Required
OverrideErrorCode	You can override the error code implied from the parent fault tag by setting the code here	8	Implied
OverrideReturnValue	You can override the return value implied from the parent fault tag by setting the return value here	0	Implied
PassThrough	If PassThrough is set to true the actual API function will be called and Holodeck will use that return value, instead of the error value specified in the fault element. Valid Values are: True, False	True	Implied
Exception	While using .NET function calls it might be necessary to specify an exception code to be set	3221225495	Implied
Allocation	The amount of memory or disk use caused by this call. Valid Values are: <ul style="list-style-type: none">• GT – only fire if disk or mem usage increases by this call• LT – only fire if disk or mem usage decreases by this call• GTE – only fire if disk or mem usage increases or is unchanged	GT	Implied

- LTE – only fire if disk or mem usage decreases or is unchanged
- E – only fire if disk or mem usage unchanged

<MatchParams> tag – Fire the test if all the attributes match the api call.

Attribute	Description	Example	Required
Name	The Name of the parameter to match	lpFileName	Required
TestOperator	<p>How the test value will be compared to the incoming parameter value. Valid values are:</p> <ul style="list-style-type: none"> • = – test value matches param value exactly • != – test value does not match param value • < – test value is less than the param value • > – test value is greater than the param value • <= – test value is less than or equal to the param value • >= - test value is greater than or equal to the param value • starts with – A string parameter starts with the test value • ends with – a string parameter ends with the test value • contains – a string parameter contains the test value (could also start or end with the value) • not contains – a string parameter does not contain the test value • binary contains – a binary flag parameter has OR'd test test value in • binary not contains – a binary flag parameter hasn't OR'd the test 	not contains	Required

- value in
- `=resource` – a path or handle parameter represents the test value resource path
 - `!=resource` – a path or handle parameter does not represent the test value resource path
 - `contains resource` – a path or handle value contains the test value resource path
 - `not contains resource` – a path or handle value does not contain the test value resource path
 - `starts with resource path` – a path or handle value represents a resource path that starts with the test value resource path

TestValue	The value to compare.	"\\."	Required
	All characters except ‘(’, ‘)’, ‘{’, ‘}', ‘\$', ‘\’, ‘?’, ‘/’, ‘:’, ‘*’, ‘+’, ‘^’, and ‘ ’ can be used normally. To use these characters you must precede that character with the ‘\’ character.		
CompareAsType	The type of the test value to compare:	1	Required
	Valid Values: 0 - NullType 1 - StringType 2 - WideStringType 3 - BooleanType 4 - IntegerType 5 - UnsignedLongType 6 - RealType 7 - PointerType 8 - PointerPointerType 9 - Integer64Type		

	10 -OutStringType 11 -OutWideStringType 12 -IgnoreType		
ID	The zero based index of the parameter to match.	0	Required

<CheckResource> tag - check for a resource existence or non-existence before firing the test

Attribute	Description	Example	Required
ParamIndex	The index of the parameter that represents the resource to check. Could be a handle or a string resource path.	0	Required
Exists	<ul style="list-style-type: none"> • 1 – only fire if the resource exists • 2 – only fire if the resource doesn't exist • 3 – fire if the resource doesn't exist then create it 	1	Required

Types of faults

Types of Faults

- 1) **Disk Faults** – These faults include faults that have to do with Hard Drive and File I/O corruption.
- 2) **Memory Faults** – Memory faults include any problem the operating system may have accessing the system memory space.
- 3) **Network Faults** – Network faults can be anything from a wrong version of Winsock, to the Network is completely disconnected.
- 4) **Registry Faults** – Manipulate how your application is able to read and write registry values.
- 5) **Process/Library Faults** – Process and Library faults can be very helpful while testing process manipulation. You can also remove access to important libraries using these faults.
- 6) **Custom Faults** – Custom faults, as set in the Faults.xml file will be listed here.

Holodeck sets all the error conditions exactly as the operating system would if the failure scenario actually occurred.

Fault Categories

Disk Faults

Faults that are common to storage media, such as the Hard Drive, Flash media, Optical etc.

Too Many Open Files – Any attempts to open, create, or find a file will be failed; sets the error code 4, Error too many open files, and return value -1.

File Not Found – Any attempts to copy, create, delete move, replace, find, open or get/change file attributes are failed in this fault; sets the error code 2, Error File Not Found, and return value 0.

File Locked – Any attempt to access the file for any reason is failed; sets the error code 33, Error Lock Violation, and return value 0.

File Is Read Only – Fails copy, create, delete, move, replace, write, or change attributes on the file; sets the error code 6009, Error File Read Only, and return value 0.

File In Use – Functions copy, create, delete, move, replace open and setting file attributes are failed; sets the error code 32, Error Sharing Violation, and return value 0.

File Cannot Be Created – Any attempts to create a file, including copying, moving, and replacing files are failed; sets the error code 82, Error Cannot Make, and return value 0.

File Already Exists – Fails attempts to copy, move, or create a file which manipulates the filename; sets the error code 183, Error Already Exists, and return value 0.

Drive Cannot Seek Disk – Fails any attempts to access or modify any file for any reason; sets the error code 25, Error Seek, and return value 0.

Disk Full – Fails attempts to allocate more space on the disk, this includes copying, creating, replacing, and writing files; sets the error code 112, Error Disk Full, and return value 0.

Data Error – Fails any attempt to access any file for any reason; sets the error code 23, Error CRC, and return value 0.

Cannot Remove Directory – Fails calls to RemoveDirectoryA and RemoveDirectoryW; sets the error code 16, Error Current Directory, and return value 0.

Access Denied To File – Any attempt to access any file for any reason is failed; sets the error code 5, Error Access Denied, and return value 0.

Note: Some faults will return a variety of error codes and return values depending on the function called and the condition. The list above specifies what the majority behavior is for each fault. For more information see the faults.xml file located in <Holodeck Enterprise Edition Install Path>\function_db\faults.xml

Memory Faults

Memory Faults are faults focused to simulate failures in system memory, these faults extend to virtual memory.

Segment Locked – Calls to functions freeing, allocating, or discarding are failed; sets the error code 212, Error Locked, and return value 0.

Page File Too Small – Attempts to allocate more memory are failed; sets the error code 1454, Error Pagefile Quota, and return value 0.

Invalid Address – Calls to functions that free or allocate virtual memory or map user pages are failed; sets the error code 487, Error Invalid Address, and return value 0.

Invalid Access – Attempts to access or allocate memory are failed; sets the error code 998, Error No Access, and return value 0.

Insufficient Memory – Calls to functions that allocate more memory are failed, this also includes functions that reallocate a greater amount of memory than was previously specified; sets the error code 8, Error Not Enough Memory, and return value 0.

Note: Some faults will return a variety of error codes and return values depending on the function called and the condition. The list above specifies what the majority behavior is for each fault. For more information see the faults.xml file located in <Holodeck Enterprise Edition Install Path>\function_db\faults.xml

Network Faults

Network faults help simulate failures that are common to network protocols. These include faults that might happen physically, as well as software problems.

Network Disconnected – Fails calls that attempts to connect, send and receive on the network; sets the error code 10050, WSA net Down, and return value -1.

Network Not Installed = Fails all calls to WSASStartup and returns a return value of 10091, Network subsystem is unavailable.

Wrong Version of Winsock - Fails all calls to WSASStartup; and returns a return value of 10092, WinSock.dll version out of range.

Winsock Task Limit Reached - Fails all calls to WSASStartup and returns a return value of 10067, too many processes.

No Ports Available – Fails calls to bind, connect, and WSACConnect; sets the error code 10048, WSA address in use, and return value -1.

Network Down – Fails any calls to any API having to do with the network; sets the error code 10050, WSA net down, and return value -1.

Note: Some faults will return a variety of error codes and return values depending on the function called and the condition. The list above specifies what the majority behavior is for each fault. For more information see the faults.xml file located in <Holodeck Enterprise Edition Install Path>\function_db\faults.xml

Registry Faults

Registry Faults simulate failures common to registry values. These faults can be applied without running the risk of corrupting or deleting actual registry values.

Value Not Found – Attempts to delete or query registry values are failed; returns 2, the system cannot find the file specified, as the return value.

Value Cannot Be Written – Any attempt to write (set) a registry value is failed; returns 1013, the configuration registry key could not be written, as the return value.

Value Cannot Be Read – Any attempt to read a registry value is failed, this includes enumerating and querying registry values; and returns 1012, the configuration registry key could not be read, as the return value.

Registry Is Corrupt – Any attempt to access a registry key for any reason is failed; returns 1015, the registry is corrupted. The structure of one of the files containing registry data is corrupted, or the system's memory image of the file is corrupted, or the file could not be recovered because the alternate copy or log was absent or corrupted, as the return value.

Query Buffer Too Small – Attempts to query a registry value are failed; returns 234, more data is available, as the return value.

No More Query Items – Functions trying to enumerate registry values are failed; returns 259, no more data is available, as the return value.

No Log Space – All attempts to access a registry key for any reason are failed; returns 1019, System could not allocate the required space in a registry log, as the return value.

Key Not Found – Calls to functions that delete, open, replace, restore or unload a registry value are failed; returns 2, the system cannot find the file specified, as the return value.

Key Marked For Deletion – Attempts to delete, enumerate, flush, query, save, set or get the registry key's security is failed; returns 1018, illegal operation attempted on a registry key that has been marked for deletion, as the return value.

Key Is Corrupt – Any attempt to access a registry key for any reason is failed; returns 1010, the configuration registry key is invalid, as the return value.

Key Cannot Be Opened – Attempts to create, enumerate, open, query, save or set registry values are failed; returns 1011, the configuration registry key could not be opened, as the return value.

IO Operation Failed – Attempts to load, restore, save and unload registry keys are failed; returns 1016, an I/O operation initiated by the registry failed unrecoverably. The registry could not read in, or write out, or flush, one of the files that contain the system's image of the registry, as the return value.

Access Denied – Any attempt to access a registry key for any reason is failed; returns 5, access is denied, as the return value.

Note: Some faults will return a variety of error codes and return values depending on the function called and the condition. The list above specifies what the majority behavior is for each fault. For more information see the faults.xml file located in <Holodeck Enterprise Edition Install Path>\function_db\faults.xml

Process/Library Faults

Faults of the Process/Library type are failures in loading external processes and libraries.

Process File Not Found – Calls to functions trying to load a library, create a process or load a module are failed; sets the error code 2, Error File Not Found, on return and returns 0 as the return value.

Not Enough Resources – Calls to functions trying to load a library, create a process or load a module are failed; sets the error code 8, Error Not Enough Memory, on return and returns 0 as the return value.

Invalid File Type – Calls to functions trying to load a library are failed and sets the error code 126, Error Mod Not Found, and return a value of 1, functions trying to create a process, or load a module are failed and set the error code 193, invalid file type and use a return value of 0.

Access Denied - Calls to functions trying to load a library, create a process or load a module are failed; sets the error code 5, Error Access Denied, on return and returns 0 as the return value.

Note: Some faults will return a variety of error codes and return values depending on the function called and the condition. The list above specifies what the majority behavior is for each fault. For more information see the faults.xml file located in <Holodeck Enterprise Edition Install Path>\function_db\faults.xml

Custom Faults

Any faults of the type Custom or that are not from the other categories will be listed in this column.

COM Object Does Not Exist – Fails CoCreateInstance and CoCreateInstanceEx with a return value of 2147746132, class is not registered. Fails OleCreate, and OleCreateEx with a return value of 2147942414, interface is not registered.

Refer to the Adding custom faults section for more information on adding your own faults.

Resource Faults

Resource Fault Overview

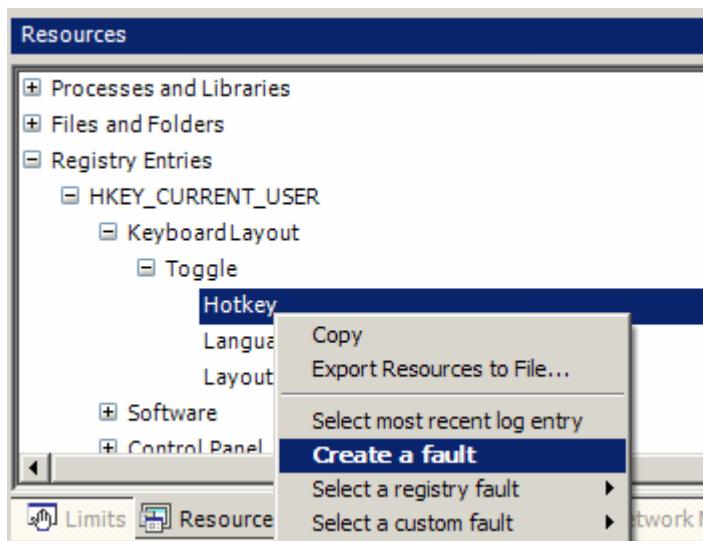
Resource faults are faults that are focused on a single Process, Library, Files, Folders, and Registry Entries. Holodeck allows you to create many different types of faults that are related to errors that commonly happen with each category of resource.

Adding and Removing a Resource Fault – Learn how to add or remove a resource fault.

Create a New Resource Fault Wizard – The resource fault wizard helps you create a Resource Fault.

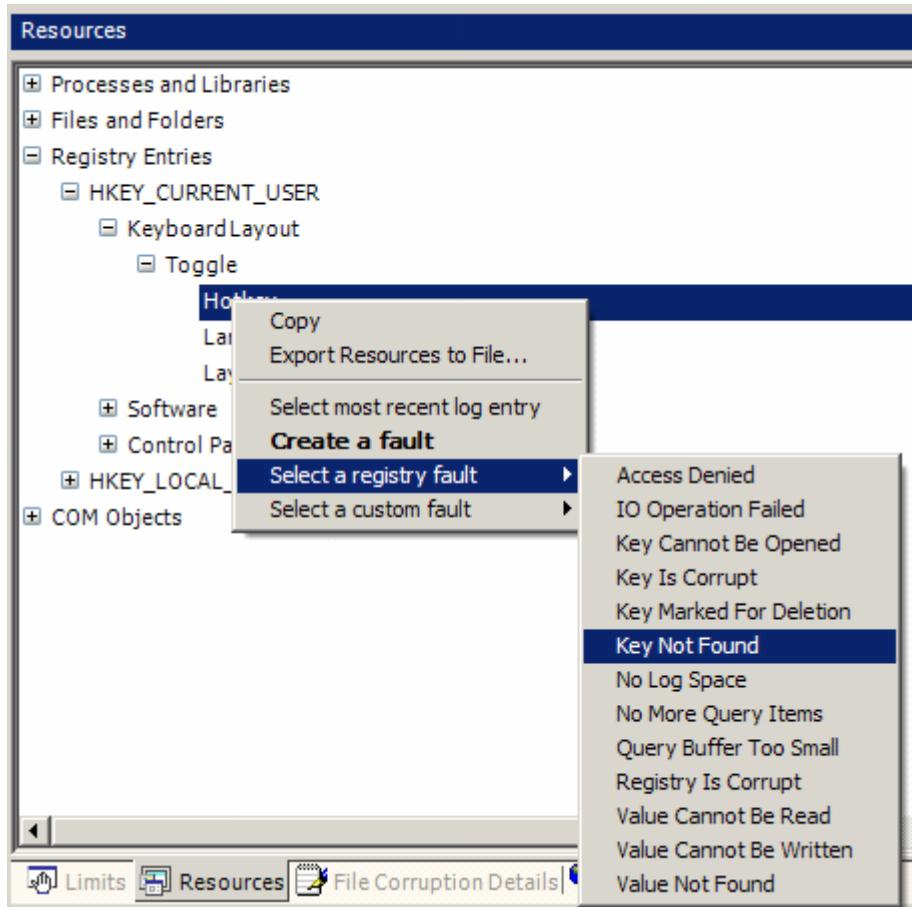
Adding and Removing a Resource Fault

A resource fault can be added by either selecting Application > Create a new Resource Fault... or by finding the resource in the resource pane, right-clicking and selecting "Create a Fault." You can also set a resource fault by right clicking the Resource Fault Node in the Project Pane and selecting "Create a Resource Fault." Once a Resource Fault has been set that resource will be displayed in red text.



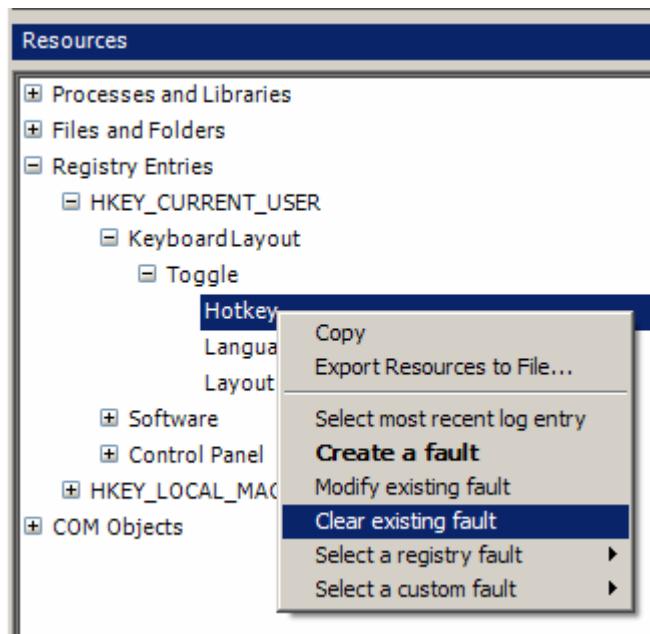
If you create the fault by the later method the Create a New Resource Fault wizard will have the correct resource already selected, otherwise you will have to find the correct resource in the resource tree on the first page of the wizard.

You can quickly set Resource faults without having to go through the Resource Fault Wizard by finding the correct resource, right-clicking and selecting "Select a Fault > Fault Name." This will set the fault immediately.

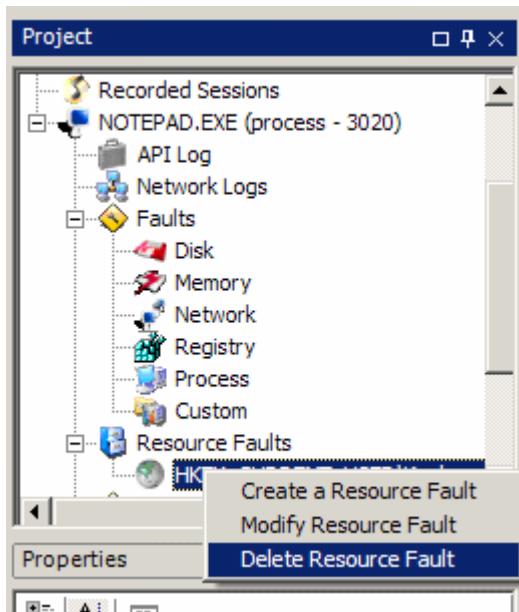


To remove a Resource fault:

Find the resource in the Resource Pane right-click and select "Clear Existing Fault"

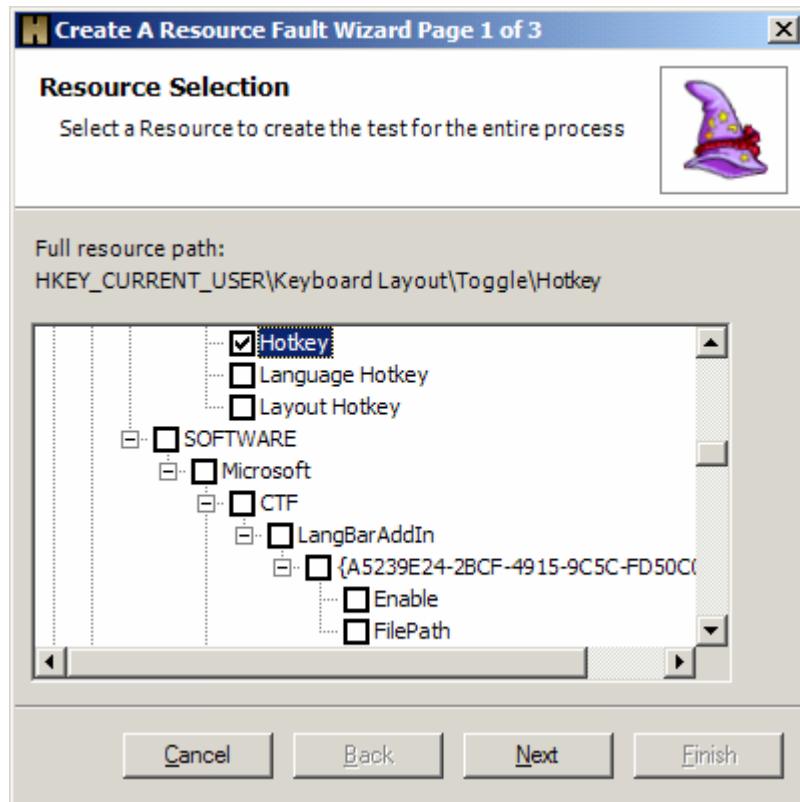


Or right-click the fault in the Project pane under the Resource Faults Node and select "Delete Resource Fault"

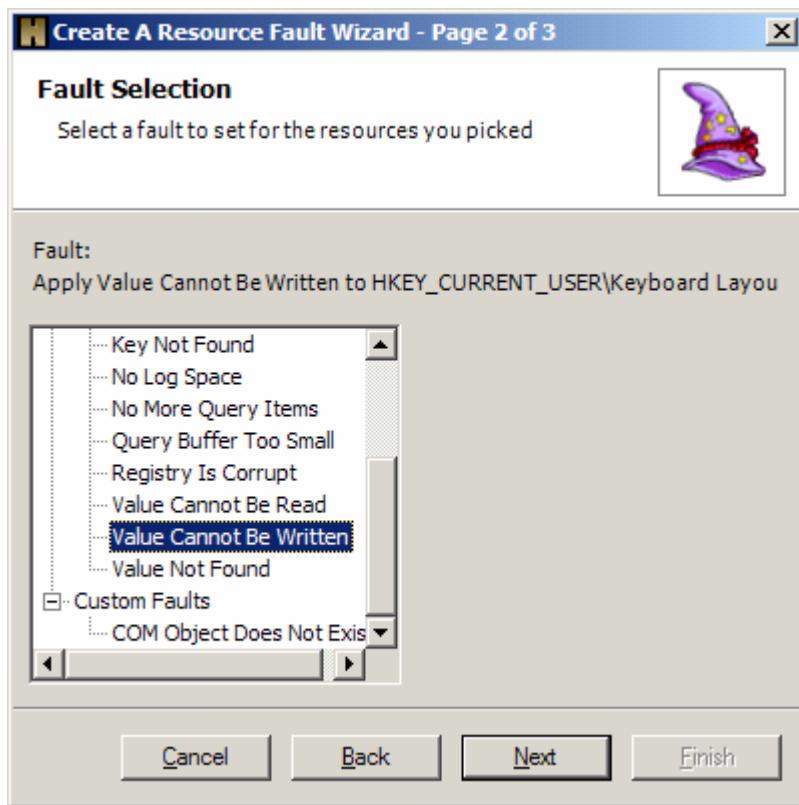


Create a New Resource Fault Wizard

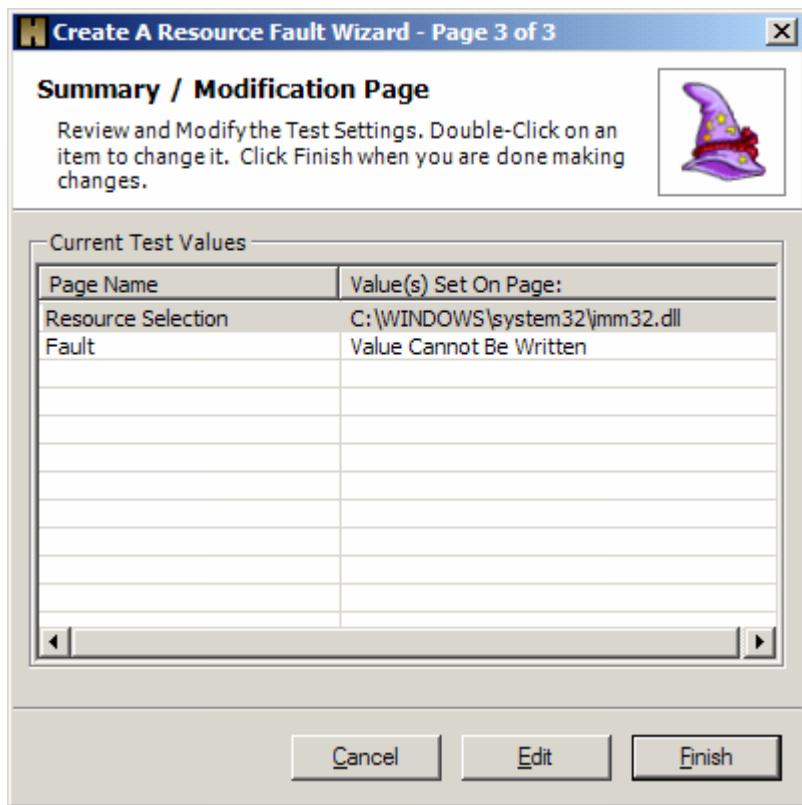
The first page of the Create a New Resource Fault Wizard allows you to select which resource you would like to set a fault for. You can select more than one of the same type of Resource; multiple selections between categories (e.g. registry and process) are not allowed.



The second page shows only the Resource Faults that are allowed for the category of fault you have selected. Custom Resource Faults will also be shown, only one fault may be selected.



The final page is a summary page, you can return to either page quickly by selecting the Page Name and clicking edit. This will return you to the page that value was set on, so you can edit it.



Working with Logs

Logs Overview

As soon as you attach to the Application Under Test Holodeck will start to log the API calls you have selected. Logs can be filtered, sorted, or exported to make working with the data Holodeck provides as easy as possible.

For more information on working with the log files Holodeck creates see the following help topics:

Log Categories – a complete list of the Log Categories, and an explanation of each.

Custom Log Filtering and Sorting – You can sort and filter log files without affecting which functions are intercepted and logged.

To find text in the log files – Find specific text within the API function Logs

Changing which functions are logged – Change which API functions are intercepted and logged.

Exporting logs – Export log files into easy to parse Excel Comma Separated Value File (csv)

Printing Logs – Holodeck can print the API logs for easy offline review

Exporting Resource Logs – Export a complete list of the resources your

Network Logs Heledeck intercepts and logs each Network Packet. View the Network Logs for inspection.

Network Logs: Holodeck intercepts and logs each Network Packet. View the Network Logs for information regarding Network Packets.

API Log Categories

Holodeck automatically sorts each API call into categories. These categories can help you to investigate failures or decide which functions to set tests for.

TimeStamp – This is the precise time in which Holodeck logged the API call. The format of this category is Month/Day/Year Hour:Minute:Second:Milisecond.

Thread – The ID of the Thread in the Application Under Test that called this function.

Category – This is the main category of the API call, Holodeck intercepts both win32 APIs and .NET APIs
Categories include:

COM – API functions included in the COM (Common Object Model)

DANGEROUS – These are dangerous APIs as listed in the Writing Secure Code Book.

FILE – API functions having to do with file manipulation such as: file attributes, disk space, file size, creating and deleting files, etc.

LIBRARY – API functions having to do with library functions such as: loading libraries and getting module handles.

MISCELLANEOUS – API functions that don't fit into any other category.

MEMORY – API functions having to do with Memory calls such as: LocalAlloc, LocalFree, HeapSize etc.

NETWORK – API functions having to do with Network calls including socket manipulation and sending and receiving calls.

PROCESS – Includes thread and process specific function calls including CreateThread, and GetStartupInfo

REGISTRY – Registry Specific API calls are contained in this category, including Opening, Querying, Closing and others.

SECURITY – API functions that are related to security including cryptography and licencing.

TIME – Functions dealing with timers, are included in this category.

DLL – This column shows which .dll the intercepted API is located in.

Function – This shows which API calls the Application Under Test has called. The next few columns are information about that function call.

Return Value - The value returned from the function called, often these return values will be memory addresses, but more useful information might be looking for 1's or 0's (representing the call was made successfully or unsuccessfully) or values smaller than a memory address (actually useful return values) return values may also be TRUE, FALSE, or NULL.

Error Code – This column shows the error code returned by the API function. This is not set by all functions, some use return values to give status. Error Code List

Exception – If the function raised an exception it will show up in this column, this only applies while testing .NET applications. Exception List

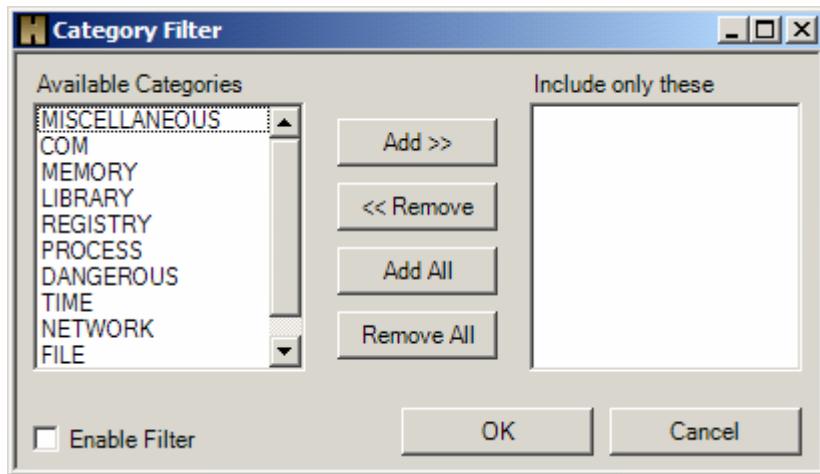
List of Parameters – These columns contains all the parameter values.

Custom Log Filtering and Sorting

Each log view can be filtered and sorted based on each header column. Filtering a log does not affect which API calls are logged; it simply changes the view in the current log pane. The log pane is, by default, sorted by the timestamp of the API call. This can be changed to sort on any column.

To filter on a column:

- 1) Right-click the column to filter on.
- 2) Select Custom Filter...
- 3) The following shows up for Category filtering



- 4) Select which categories to include and click OK.

Enable filter will automatically be selected. If a column has a filter set the column text will be shown in blue.

To remove a filter reselect Custom Filter... from the log column and uncheck "Enable Filter"

To sort a column:

Clicking the column header once sorts the column in ascending order; clicking the column again will sort the column in descending order.

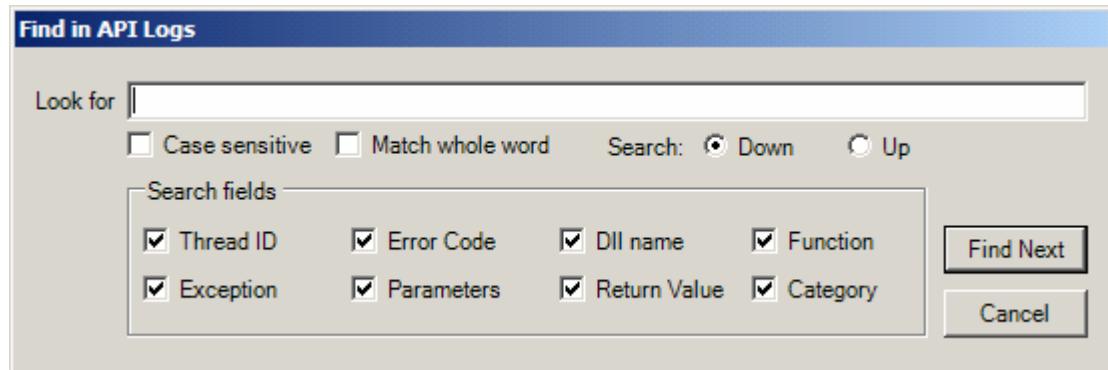
The column the log file is currently is sorted by is represented by a grey arrow to the right of the column header.

Note: Function filters in the log pane only apply to top level nodes. For instance, if you filter out a function, it may still show as a child of some other function. This also means that if you filter a function that has unfiltered children, those children will never show in the log since the parent was filtered out. This does not apply if you change which functions are logged in this case the functions are never intercepted.

To find text in the log files

With the log pane open, either select Log > Find... or press Ctrl-f. This will bring up the "Find in API Logs" dialog box. This find field works similarly to most other find commands. You can select to only search on certain fields, or change case sensitivity. Matching the whole word will only return matches that the word stands alone for instance searching for "free" would match only cases of "free" but not "FreeLibrary"

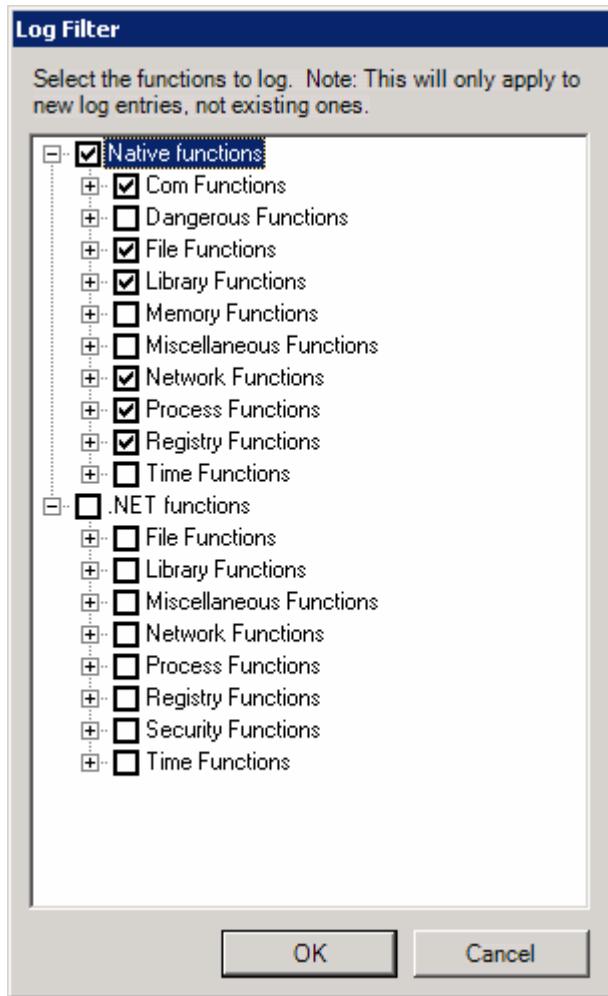
Note: Find functionality works with network logs as well.



Changing which functions are logged

Enabling more API functions to be logged increases the number of tests available to you. Decreasing the number of API function logged increases Holodeck's performance and uses less RAM.

To enable or disable APIs to be logged click Log > Functions Logged... from the menu.



Note: Disabling COM, File, Library, Network, Process or Registry function may disable some features of Holodeck. For more information please see the Default Logging help topic.

Exporting Logs

Holodeck can export a log file for external manipulation in an easy to parse Excel Comma Separated Value File. This can be helpful when trying to compare two different runs of the same application.

To export a log click File > Export Log to File...

Once the log file has been exported it can be easily manipulated by any spreadsheet program.

Exporting a log file will ignore any Log Filters and field settings you may have set during testing, the exported log file will contain all available data for the process you save.

Log files contain the following information for each log entry:

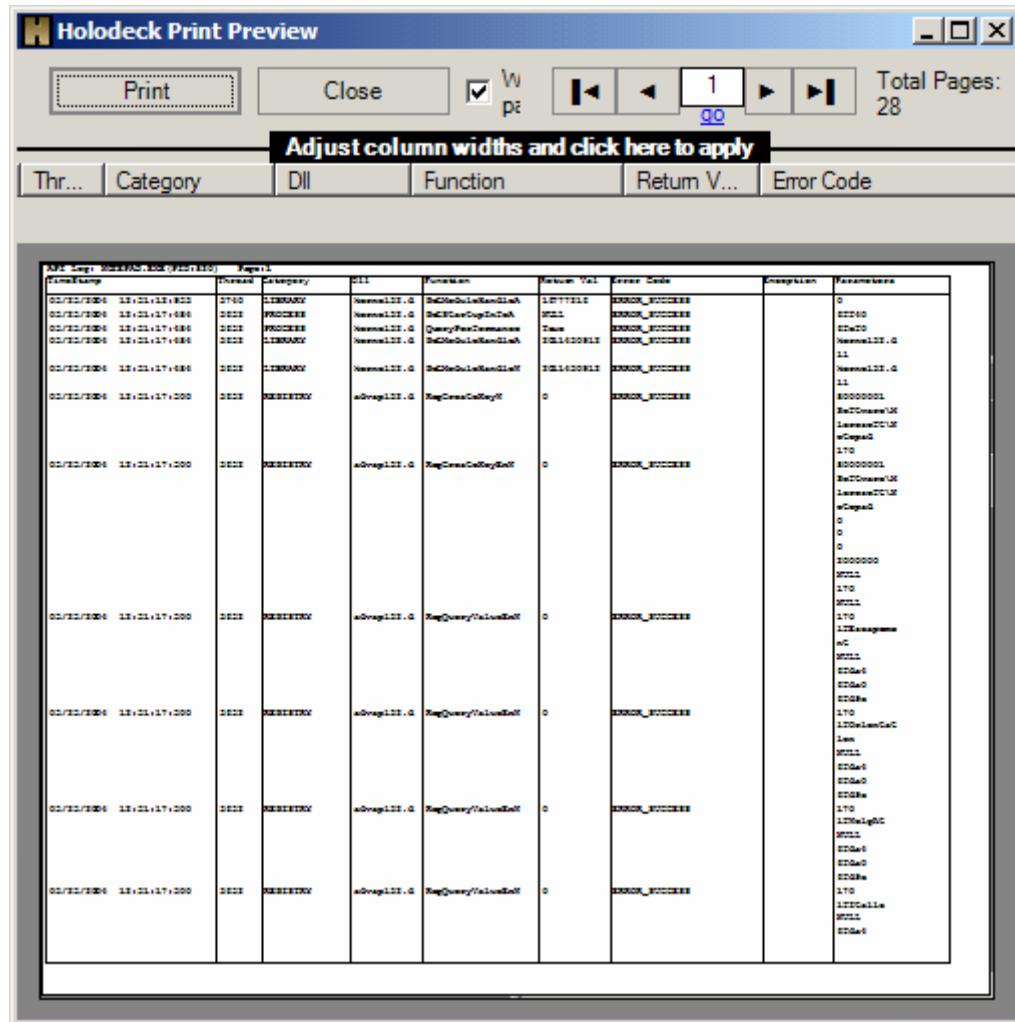
- Category
- DLL Name
- Error Code
- Function
- Parameters
- Return Value
- Time Stamp
- Exceptions

For more information on these categories please see the Log Categories section.

Printing Logs

Holodeck allows you to print the complete list of API logs easily through the Print Preview Dialog (below). Preview individual pages using the forward and back buttons located at the top right of the window. Adjust columns widths by dragging the dividers of the column headers, after the column has been resized click the button above to apply the new column size settings.

Note: Filters and sorting does affect the print output; use filtering to narrow down what you want to print.



Exporting Resource Logs

Holodeck can export a log file of all the resources your application uses. This is useful when comparing different builds, or to track resource usage changes.

To export the resource log click File > Export Resources to File...

Each log entry contains the following information:

- Type – The type of the resource, FILE, PROCESS, or REGISTRY
- Resource – The path to the resource.
- TimeStamp – The time when the Resource was last used.
- LastFunction – The last function to use this resource.
- ErrorCode – The Error Code set by the last function.
- ReturnValue – The return value returned by the last function.
- NumHits – The number of times this resource was touched by the application

Network Logs

Network logs are logs specific to data being sent and received over the network. These can be extremely powerful when trying to see exactly what data was sent or received over the network and how it was sent or received.

Network logs include information on each network message including:

TimeStamp – The time at which the message was sent or received.

Thread – The thread number that called the API to send or receive the message.

Direction – If the message was sent or received.

Protocol – Which network protocol was used to communicate the message.

Msg Length – The complete message length in bytes.

Start of Msg – The first 8 bytes of the message.

TimeStamp	Thread	Direction	Protocol	Msg Length	Destination IP
05/24/2004 00:47:38:437	2176	Send	MSAFD Tcpip [UDP/IP]	1	127.0.0.1
05/24/2004 00:47:38:437	2872	Recv	MSAFD Tcpip [UDP/IP]	1	127.0.0.1
05/24/2004 00:47:39:078	2176	Send	MSAFD Tcpip [TCP/IP]	362	216.239.57.10
05/24/2004 00:47:39:093	2872	Recv	MSAFD Tcpip [UDP/IP]	1	127.0.0.1
05/24/2004 00:47:39:093	2176	Send	MSAFD Tcpip [UDP/IP]	1	127.0.0.1
05/24/2004 00:47:40:156	2176	Recv	MSAFD Tcpip [TCP/IP]	1024	192.168.0.114
05/24/2004 00:47:40:343	780	Recv	MSAFD Tcpip [TCP/IP]	373	192.168.0.114
05/24/2004 00:47:46:937	748	Send	MSAFD Tcpip [TCP/IP]	408	66.102.7.104
05/24/2004 00:47:47:750	748	Recv	MSAFD Tcpip [TCP/IP]	170	192.168.0.114
05/24/2004 00:47:49:953	748	Recv	MSAFD Tcpip [TCP/IP]	5	192.168.0.114
05/24/2004 00:47:50:171	2176	Send	MSAFD Tcpip [UDP/IP]	1	127.0.0.1
05/24/2004 00:47:50:171	2872	Recv	MSAFD Tcpip [UDP/IP]	1	127.0.0.1
05/24/2004 00:47:50:171	2176	Send	MSAFD Tcpip [TCP/IP]	603	216.239.57.10
05/24/2004 00:47:50:953	2176	Recv	MSAFD Tcpip [TCP/IP]	1024	192.168.0.114

To see the complete message highlight the network log entry and look at the Network message details. For more information on the Message Details see the Network Message Details Pane help item in the Holodeck Windows and Panes Section.

Export the network logs by right clicking any entry in the log and selecting Export Log to file, or click File >Export Logs to File... with the Network Logs Pane open.

Note: Network logs will not be generated when the connection was already open before HD attaches to the application. In order to get network logs, Holodeck must to be attached when the connection is first established.

Working with Reports

Reports Overview

Holodeck makes it easy to investigate application failures by organizing the logged API calls, Faults, Tests, Error Codes, Resource Dependencies, Resource Usage, and Return Values into easy to read pivot tables.

For more information regarding Holodeck's reporting feature see the following help topics:

[Managing and Printing Reports](#)

[What's Contained in a Report](#)

[Working with Pivot Tables](#)

Managing and Printing Reports

To create a report on all processes and threads in the current project right click the Reports node in the project pane and click Create a Report. This will create a report under the reports global node named Holodeck Report

Manipulating generated reports

To save a report Right-Click the report in the Project Pane and click Save Report, this will allow you to save the report to a remote location to manipulate and compare different reports.

Note: If you create a report and want to share it with someone else you will need to send the html file plus its support files folder "reportname_files"

Renaming Reports

- do either:
- 1) To rename a report Right-Click the report in the Project Pane and click Rename Report
 - 2) Single click the name of the report you would like to rename.

Note: Rename a report before generating a new one to avoid overwriting the old report.

What's Contained in a Report

Each report Holodeck creates an easy to read summary of the log file created by the Application Under Test. This summary is displayed in a PivotTable which can be filtered, modified or manipulated however you see fit. The log summary is useful to see where the Application Under Test failed or which category of API calls are most frequently used.

API Log Summary Table - This report shows summary data for APIs logged by each process. Use this data to determine how many API calls were made for each category.

Faults Summary Table - This report shows summary data for the faults set for each process. Use this data to determine which faults were active at the time of report creation.

Tests Summary Table - This report shows summary data for the tests set for each process. Use this data to determine which tests were active at the time of report creation.

Error Codes Summary Table - This report shows summary data for the error codes set by APIs called from each process. Use this data to examine any API calls that set failure error codes.

Resource Dependencies Summary Table - This report summarizes the dependencies for each process. Use this data to see the number of dependencies as well as the number of times each dependency was touched. If you see a dependency with a very high hit-count this may be, or related to, a performance bug.

Resource Usage Summary Table - This report shows maximum and average resource usage for each process.

Return Values Summary Table - This report shows summary data for the values returned by APIs called from each process. Use this data to examine any API calls that returned failure error codes.

Working with Pivot Tables

The Holodeck report uses PivotTables which are interactive tables that quickly combine and compare large amounts of data. You can rotate its rows and columns to see different summaries of the data, and you can expand the details of the table to drill down on more important information.

You can drag irrelevant information out of the table to remove it from view. Filter the table views to show only pertinent information, by right clicking the table header and clicking AutoFilter. Sort your tables ascending or descending by right clicking the table header and clicking either Filter Ascending or descending.

Reorder your tables to view certain information near to one another by dragging and dropping entire rows from one place to the other, the PivotTable functionality will keep your information in order.

For more information about PivotTables please see this Microsoft Help Demo.

Reports Tables

API Log Summary Table

This report shows summary data for APIs logged by each process. Use this data to determine how many API calls were made for each category.

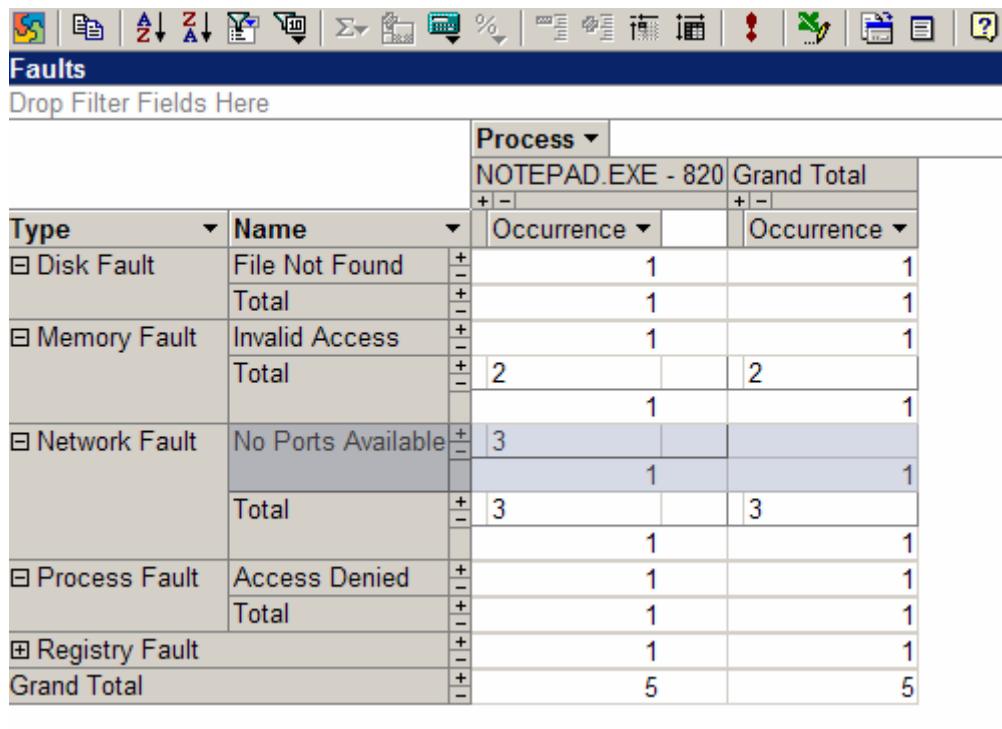
Drill down into each category to find which API calls failed, and a record of error codes, exceptions, return values and parameters.

Process			
NOTEPAD.EXE - 1112		Grand Total	
Type	Function	Log Entry Count	Log Entry Count
+ COM		55	55
+ FILE		858	858
+ LIBRARY		240	240
+ NETWORK		22	22
+ PROCESS		166	166
+ REGISTRY		2571	2571
Grand Total		3912	3912

Faults Summary Table

This report shows summary data for the faults set for each process. Use this data to determine which faults were active at the time of report creation.

Expand fault types to find out which faults were set during the test.



		Process ▾	
		NOTEPAD.EXE - 820 Grand Total	
Type	Name	Occurrence ▾	Occurrence ▾
Disk Fault	File Not Found	+ 1	1
	Total	+ 1	1
Memory Fault	Invalid Access	+ 1	1
	Total	+ 2	2
		+ 1	1
Network Fault	No Ports Available	+ 3	
		+ 1	1
	Total	+ 3	3
Process Fault	Access Denied	+ 1	1
	Total	+ 1	1
Registry Fault		+ 1	1
Grand Total		+ 5	5

Tests Summary Table

This report shows summary data for the tests set for each process. Use this data to determine which tests were active at the time of report creation.

Expand the type to find out which tests of each type were set in the test run. Expand the functions to find out the test details.

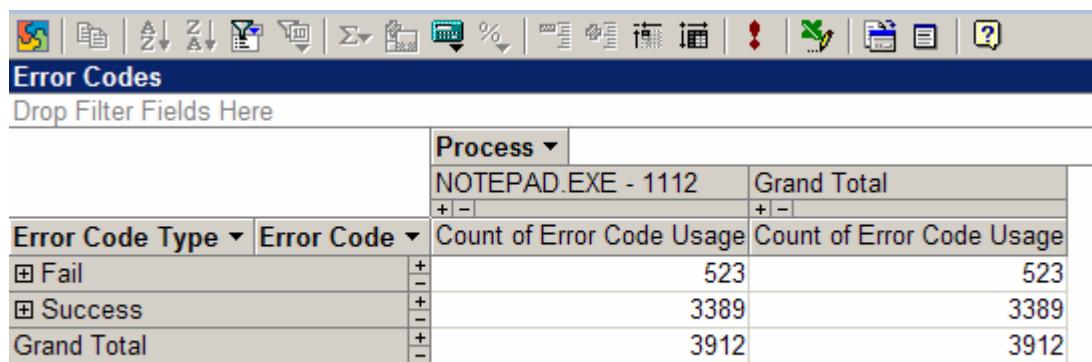
The screenshot shows a software interface with a toolbar at the top containing various icons. Below the toolbar is a dark blue header bar with the word "Tests" in white. Underneath the header is a search bar labeled "Drop Filter Fields Here". The main area is a data grid with the following columns: Type, Function, Param1, Param2, Param3, and Number of Tests. The grid displays the following data:

		Process	NOTE PAD.EXE - 820			Grand Total
Type	Function	Param1	Param2	Param3	Number of Tests	
REGISTRY	RegQueryValueExA	= {Any}	= {Any}	= {Any}	1	
	RegQueryValueExW	= {Any}	= {Any}	= {Any}	1	
	Total				2	
Grand Total					2	

Error Codes Summary Table

This report shows summary data for the error codes set by APIs called from each process. Use this data to examine any API calls that set failure error codes.

Drill down into failure codes to find out why and when your application failed. Some failures are expected, but others may be unnoticed bugs.

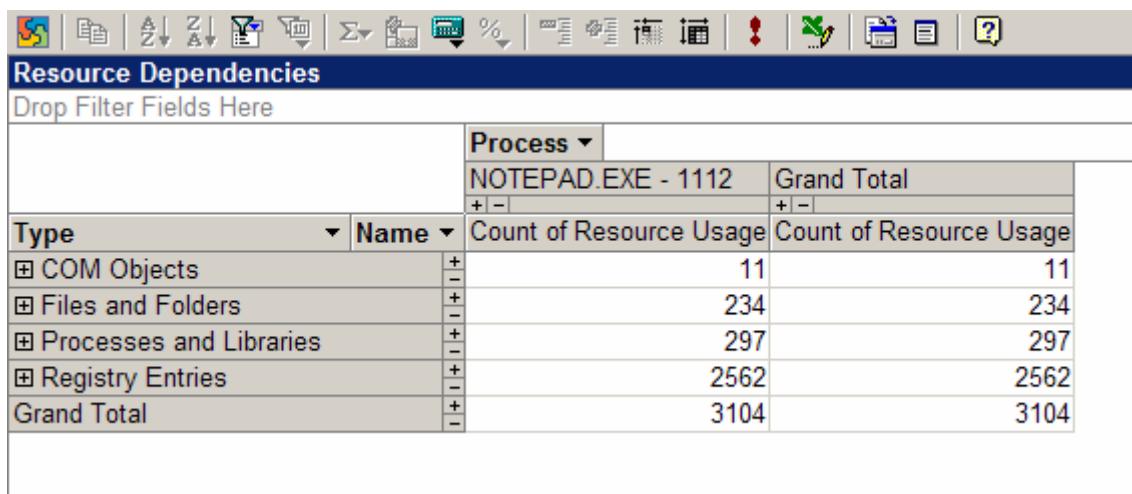


Process ▾			
Error Code Type ▾		Error Code ▾	Count of Error Code Usage
Fail		+	523
Success		-	3389
Grand Total		+	3912
		-	3912

Resource Dependencies Summary Table

This report summarizes the dependencies for each process. Use this data to see the number of dependencies as well as the number of times each dependency was touched. If you see a dependency with a very high hit-count this may be, or related to, a performance bug.

Investigate which resources your application uses the most. Some performance issues may be discovered if a resource is unnecessarily accessing a resource too many times.



The screenshot shows a software interface titled "Resource Dependencies". At the top is a toolbar with various icons. Below it is a header bar with the title "Resource Dependencies" and a sub-header "Drop Filter Fields Here". A dropdown menu labeled "Process" is open, showing "NOTEPAD.EXE - 1112" and "Grand Total". The main area is a table with the following data:

Type	Name	Count of Resource Usage	Count of Resource Usage
COM Objects		11	11
Files and Folders		234	234
Processes and Libraries		297	297
Registry Entries		2562	2562
Grand Total		3104	3104

Resource Usage Summary Table

This report shows maximum and average resource usage for each process.

Investigate which physical resources your application uses to find average and max disk, memory and network usage.

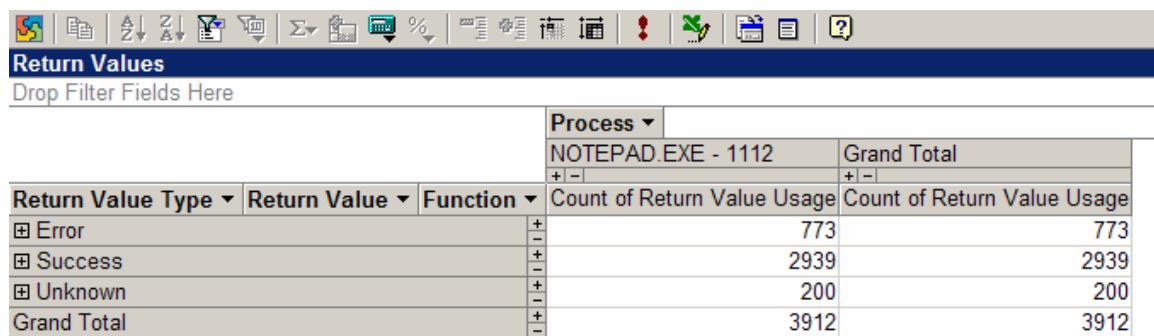
The screenshot shows a software interface titled "Resource Usage" with a toolbar at the top containing various icons. Below the title is a placeholder text "Drop Filter Fields Here". The main area is a data grid with the following columns: "Process" (with a dropdown arrow), "Name" (with a dropdown arrow), "KiloBytes Used" (with a dropdown arrow), and another "KiloBytes Used" column. The data grid displays the following information for the process IEXPLORE.EXE - 3668:

Process	Name	KiloBytes Used	KiloBytes Used
IEXPLORE.EXE - 3668	Average Disk Usage	0	
	Average Memory Usage	1	
	Average Network Download Usage per second	0	
	Average Network Upload Usage per second	0	
	Max Disk Usage	410	
	Max Memory Usage	54128	
	Max Network Download Usage per second	74	
	Max Network Upload Usage per second	27	
	Grand Total	0 410 1 54128 0 27 0 74	0 410 1 54128 0 27 0 74

Return Values Summary Table

This report shows summary data for the values returned by APIs called from each process. Use this data to examine any API calls that returned failure error codes.

Many API functions use return values to report failures, use this in conjunction with the error codes to investigate unexpected failures.



NOTEPAD.EXE - 1112		Grand Total		
+ -		+ -		
Return Value Type	Return Value	Function	Count of Return Value Usage	Count of Return Value Usage
☒	Error		773	773
☒	Success		2939	2939
☒	Unknown		200	200
	Grand Total		3912	3912

Using Automatic Test Generation

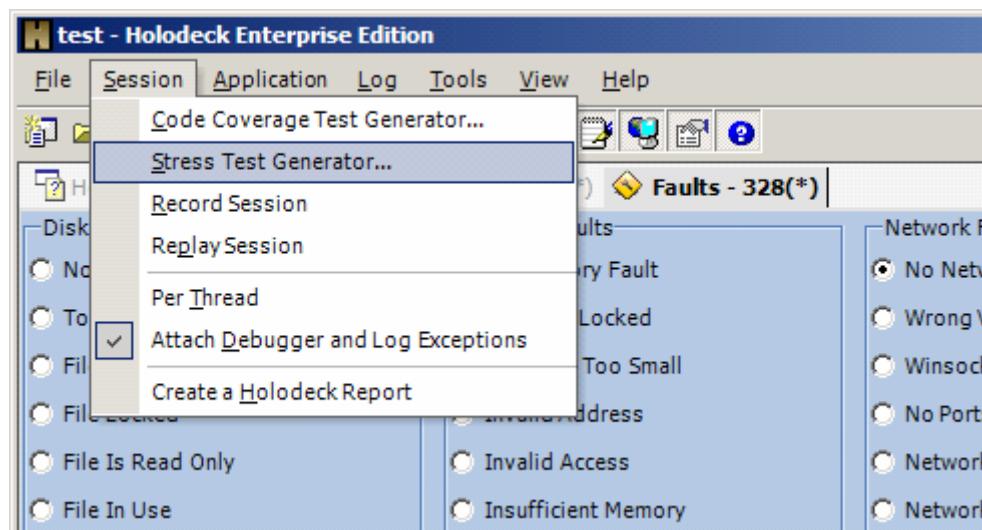
Introduction to Automatic Test Generation

Holodeck makes it easy to set and run different tests on the Application Under Test through the use of Code Coverage Test Generator and Stress Test Generator. These help to test each line of code in the Application Under Test, and simulate stressful situations for your software. When using Holodeck to automatically test your application, turn off logging options that are unnecessary to test. Some Test Generation features may stop working if all logging is turned off, however the less logging used, the less system memory will be used which will allow stress testing to last longer.

Using Holodeck for Stress Test Generation – Holodeck can automatically generate test cases that will help you stress test your application.

Using Holodeck for Code Coverage Test Generation – Holodeck can automatically set faults, limits, etc. that will help ensure a thorough test pass.

Modifying Testing Settings – You can modify the automatic test settings by editing an XML file.



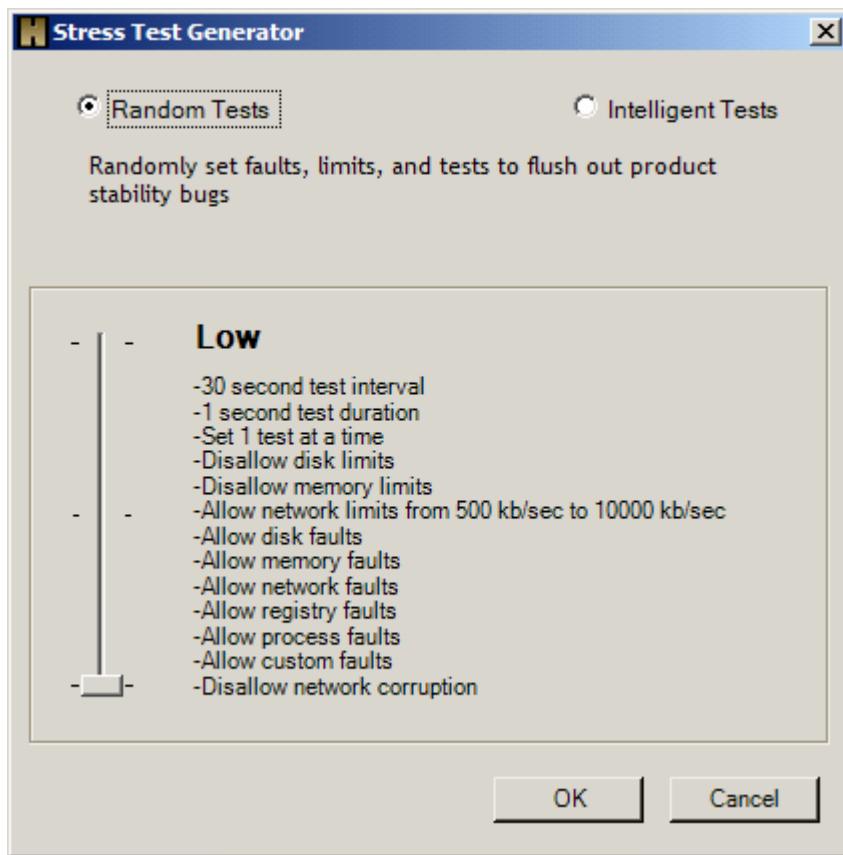
To ensure reproducibility of the bugs that you may find using Holodeck's Stress Test Generation and Code Coverage Test Generation you may want to record the test run. To record your test session simply start session recording before starting your test and stop recording the session once the tests have finished. See Recording and Replaying Sessions for help.

Note: some features may not work if certain functions are not being logged, for more information please see the Default Logging help topic.

Using Holodeck for Stress Test Generation

Using Holodeck's Stress Test Generation to test your application is a great way to ensure your software will continue to perform well even in stressful situations. Under Stress Testing Holodeck injects faults intermittently to cause stress failures to occur more rapidly than they would under normal circumstances.

Get to the Stress Test Generation dialog by clicking Session > Stress Test Generator... on the menu.



Holodeck can either set random or intelligent tests.

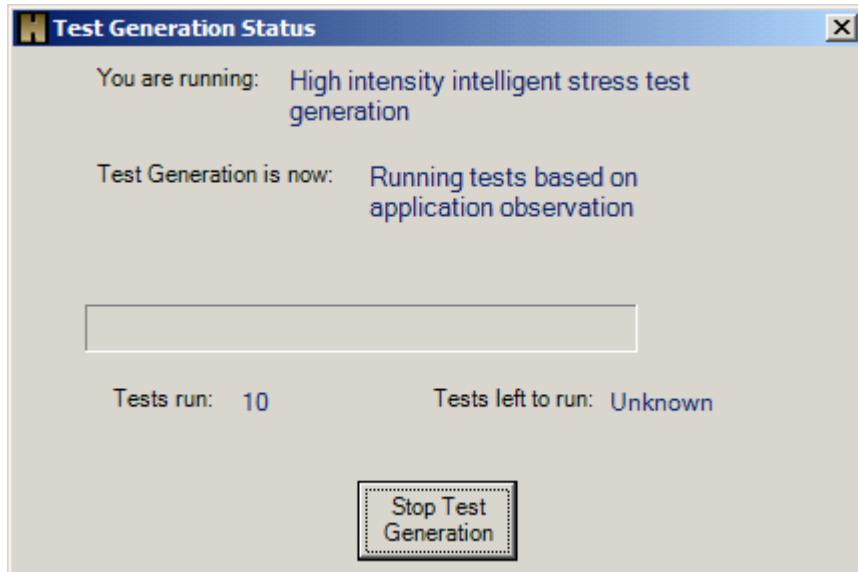
Random Tests:

Random tests starts setting tests, faults and limits as soon as Stress Testing is enabled. It randomly selects test, faults and limits within the specifications of the low, medium and high categories. While Stress Testing is running you can either attach an external test harness to run through an automated test set, or test by hand.

Intelligent Tests:

Holodeck begins the testing session by listening to the application. It sets tests, faults and limits based on what it has seen from the test run. Holodeck continues to listen to the application while further tests are made. While Stress Testing is running you can either attach an external test harness to run through an automated test set, or test by hand.

The test generation status dialog shows all the information having to do with the currently running Stress test. Since these tests can be run indefinitely there are an unknown number of tests left to run. While the tests are running look here for updates on what Holodeck is testing on your application.



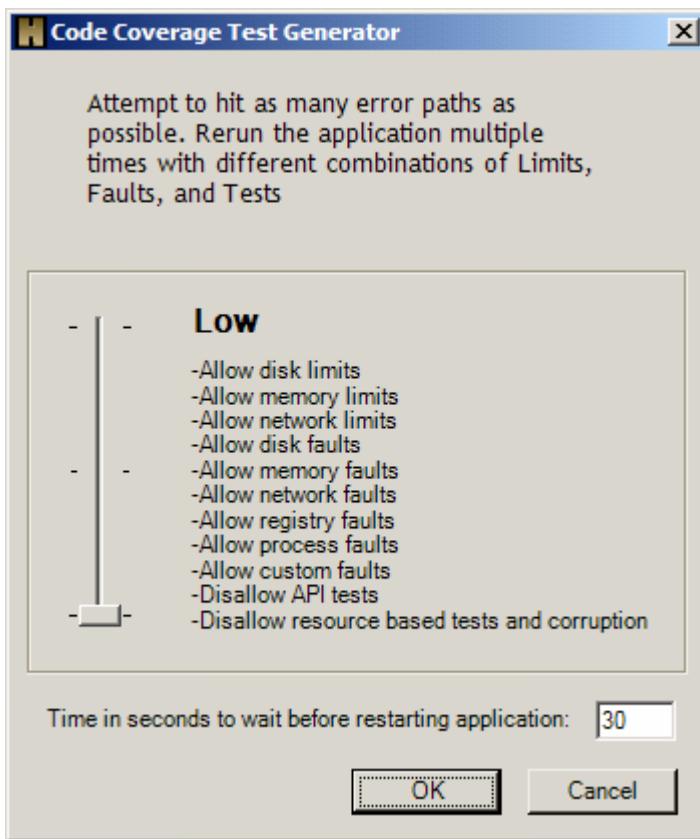
note: When stress test generation is run for multiple processes each time it creates a set of tests it will randomly select one of the processes to test.

Using Holodeck for Code Coverage Test Generation

Code coverage test generation attempts to hit as many error paths as possible. It reruns the application multiple times, each time setting different combinations of Limits, Faults, and Tests. Each time the Application Under Test terminates, by way of the script terminating the Application gracefully or Holodeck finding a crashing bug, Holodeck will relaunch the Application Under Test and start the next test, until there are none left. Holodeck assumes that each testing script will close the application under test at the end of the test; this allows Holodeck to know when to start the next test.

Once the application has stopped Holodeck will wait the number of seconds in the text box in the lower right to restart the application.

To open the Code Coverage Test Generator click Test -> Code Coverage Test Generator... on the menu.



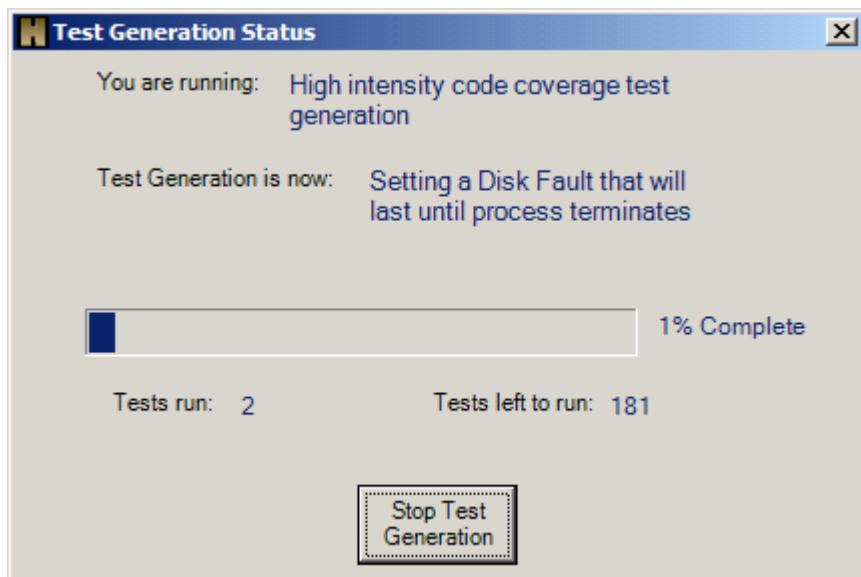
Initial Recording

The first thing Holodeck does when running Code Coverage Testing is record what the application is doing to create an intelligent test plan. Holodeck records resource usage and unique API calls until the application is terminated. It then uses the information found in the recording phase to build the next tests. Each time the application is restarted Holodeck sets a different single limit, fault or test.

- Exact order of execution:
- 1) Set each fault one at a time
 - 2) Set a limit
 - a) For disk or memory
 - i) Set a limit for max used – 1 byte
 - ii) Set a limit for average used – 1 byte
 - iii) Set a limit to minimum possible (0 bytes)
 - b) For Network
 - i) Set a limit to 50%
 - ii) Set a limit to 0%
 - 2) Set a test for each unique API call seen in the recording phase.
 - 3) Resource faults are set targeting each dependency
 - i) Faults are either two of the following
 - (1) Corruption of the Resource
 - (2) Any one of the resource faults as found in the faults pane

Note: Test/limit/fault is set randomly between start and finish of the application running so that different code paths are hit.

The test generation status dialog shows all the information having to do with the currently running test. This session of Code Coverage testing will run for 181 tests. Each time the application under test terminates, either from a crash or when the test harness closes, Holodeck will restart the application and set a new test.



Note: You should never set timeout to shorter than the original run of the test as that will cause no tests to be set on some of the restarts.

Modifying Testing Settings

You can change how Holodeck Stress and Code Coverage tests your application by modifying the TestGeneration.xml file found in the Test Generation folder in the Holodeck install directory (default: C:\Program Files\Security Innovation\Holodeck Enterprise Edition\Test Generation).

This XML file holds all the settings for how Holodeck tests your application. Before making any changes to this file it is recommended that you make a backup copy.

TestInterval – Time between setting individual tests, in milliseconds.

TestDuration – Time to enable each test, in milliseconds.

AllowNetworkLimit – Boolean, allow or disallow network limits.

AllowDiskLimit - Boolean, allow or disallow disk limits.

AllowMemoryLimit - Boolean, allow or disallow memory faults.

AllowNetworkFaults - Boolean, allow or disallow network faults.

AllowRegistryFaults - Boolean, allow or disallow registry faults.

AllowProcessFaults - Boolean, allow or disallow process faults.

AllowCustomFaults - Boolean, allow or disallow custom faults.

AllowDiskFaults - Boolean, allow or disallow disk faults.

AllowMemoryFaults - Boolean, allow or disallow memory faults.

AllowNetworkCorruption - Boolean, allow or disallow network corruption.

NumOverlappingTests – The number of tests allowed to execute at the same time.

NetworkLowRange – Lowest the network limit can be, in bytes per second.

NetworkHighRange - Highest the network limit can be, in bytes per second.

DiskLimitSet – set value 0 for at current usage, set value 1 for above current usage

DiskLimitMax – limits will be set from current usage to current usage plus this number

MemoryLimitSet– set value 0 for at current usage, set value 1 for above current usage

MemoryLimitMax– limits will be set from current usage to current usage plus this number.

Generating Network Corruption Faults

Using the Network Corruption Fault Wizard

Often network traffic can become corrupted in transfer. Holodeck allows you to simulate these errors through network corruption faults. Using this wizard you can simulate random corruption or find and replace interesting string to test network stream parsing.

Please see the following pages for more information on the network corruption fault wizard.

[Creating a fault using Random Corruption](#)

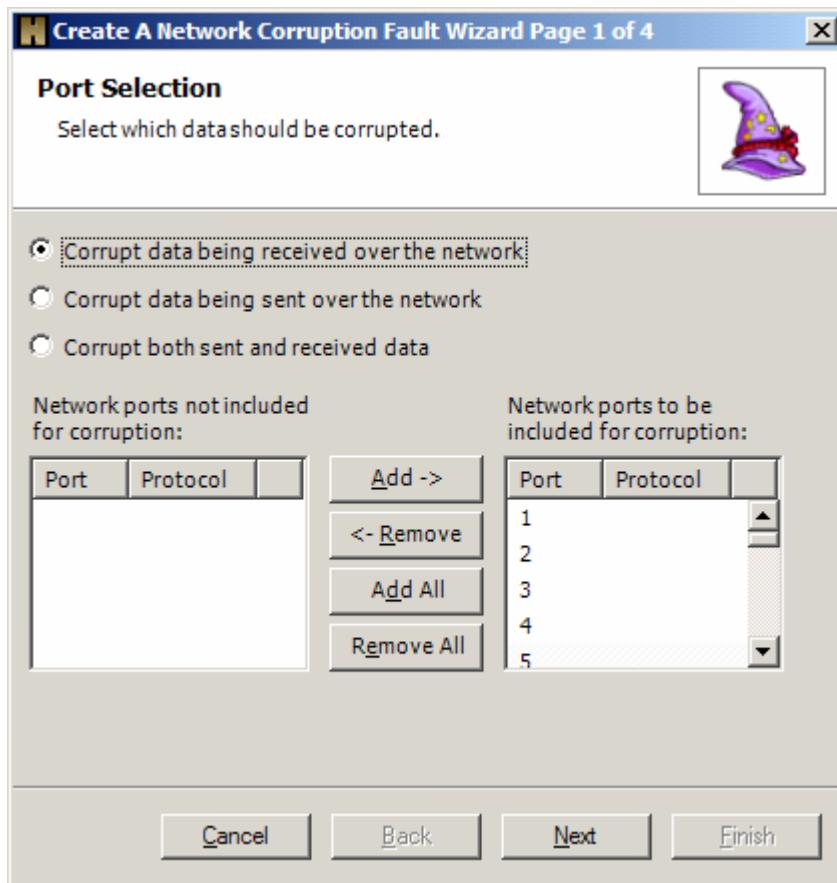
[Creating a fault using Find and Replace Corruption](#)

[Creating a fault using Regular Expressions](#)

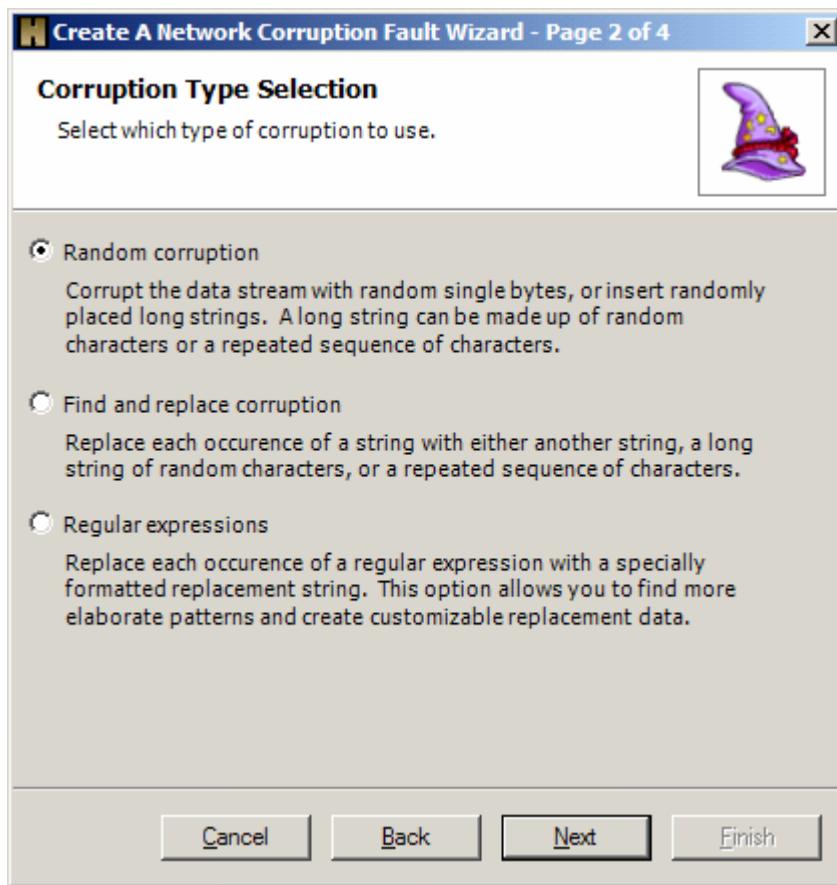
Creating a fault using Random Corruption

Random Corruption is a good way of testing for the type of corruption that would happen as a result of common line noise.

On the first page of the wizard you can specify that Corruption should happen on data being sent, received, or sent and received. Corruption being received over the network will test your client's ability to parse corrupted network messages and corruption being sent over the network will test your server's ability to receive corrupted network messages. You can also specify to corrupt messages only being sent and received over certain network ports. You might want to focus your testing on a specific service that uses a specific port.



The Default Random Corruption should already be set.

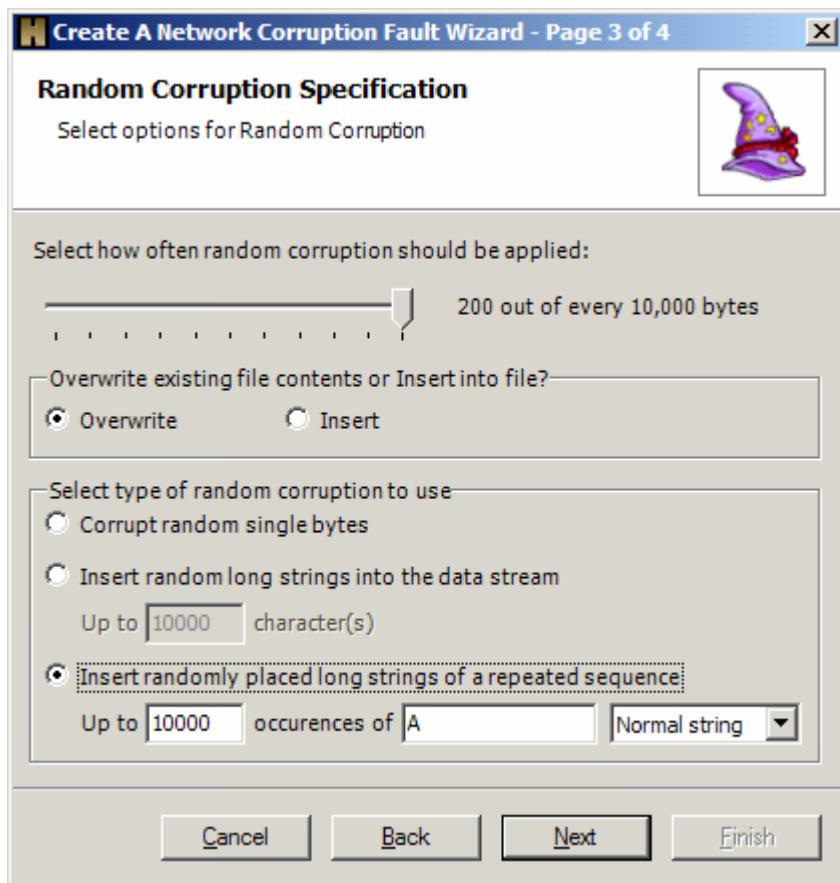


Random Corruption can happen in a variety of ways.

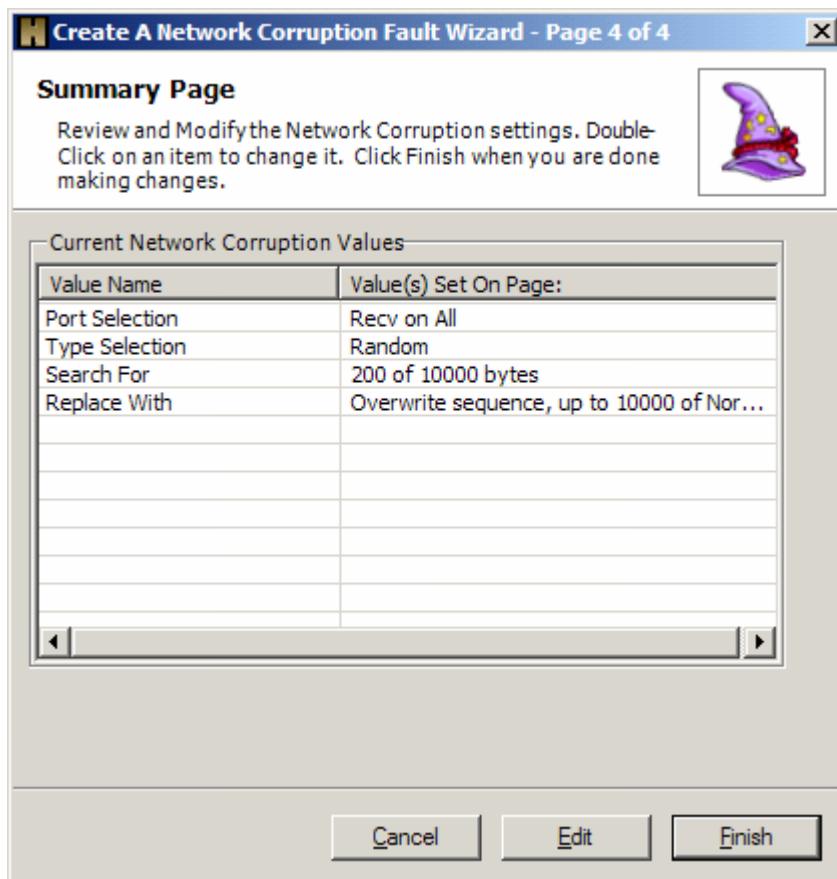
Depending on the severity of the test you may opt to corrupt more data. Sending an extremely corrupted packet may be rejected by the application early on, but is good for severe corruption smoke testing. On the other hand corrupting fewer bytes may allow the corrupted packet to masquerade as a legitimate packet and find bugs deeper in the network code.

Overwriting data in a packet replaces already existing data, this includes any header or footer information your application may be looking for.

Inserting data into the packet can find buffer overflow bugs. It may also make it easier to find the corrupted string in memory if an error occurs.



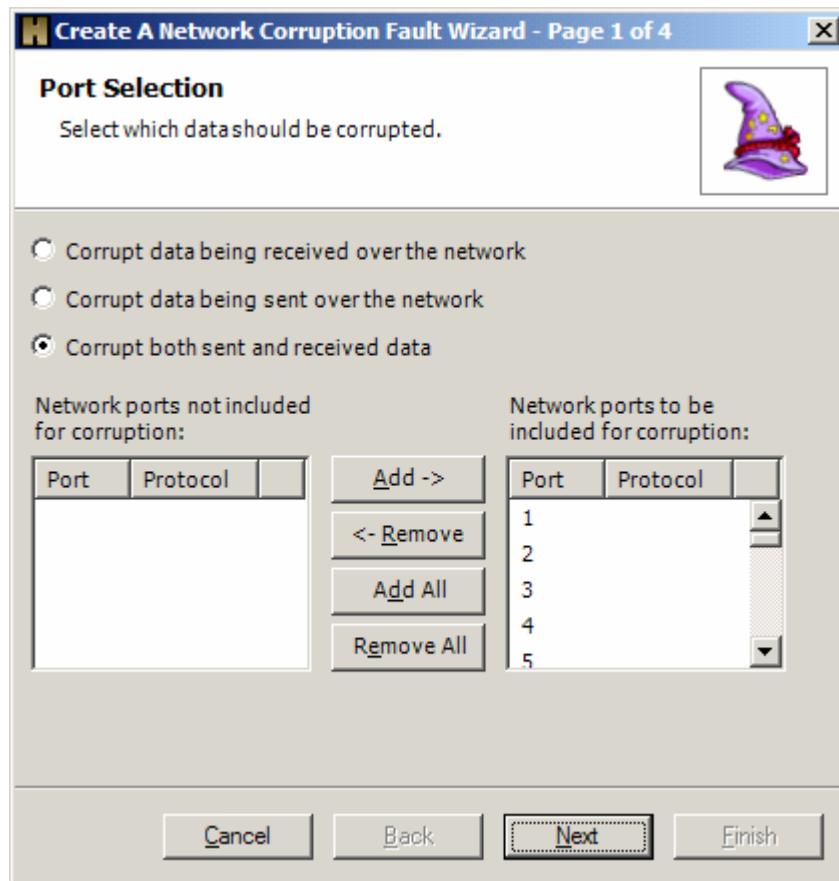
The final page shows a summary of the corruption fault you just created. To edit any information on this page simply select it, and click edit. This will take you back to that page to make any necessary changes.



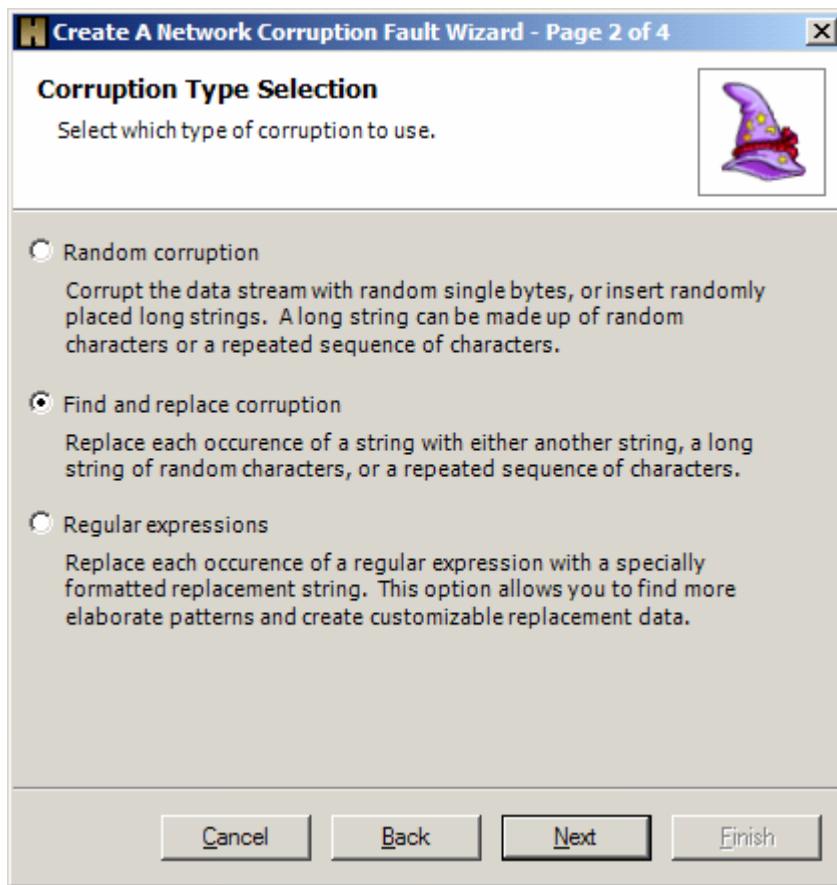
Creating a fault using Find and Replace Corruption

Random Corruption is a good way of testing for the type of corruption that would happen as a result of common line noise.

On the first page of the wizard you can specify that Corruption should happen on data being sent, received, or sent and received. Corruption being received over the network will test your client's ability to parse corrupted network messages and corruption being sent over the network will test your server's ability to receive corrupted network messages. You can also specify to corrupt messages only being sent and received over certain network ports. You might want to focus your testing on a specific service that uses a specific port.



Select Find and Replace Corruption from the list.



In the example below all occurrences of the word "coffee" will be replaced by the word decaf. You can use hex strings and escape string to find or replace strings that would otherwise be unprintable.

Hex Strings:

Hex strings will take any number of hex digits (0-9, a-f or A-F) with any amount of whitespace, where each two digits represent a byte. There must be an even number of digits, or this will result in an incomplete byte.

Example of a Hex String:

5465737404 - Converts to the word Test with an "End of Transmission" character at the end.

54 65 73 74 04 – This is the same string, but with whitespace, and easier to read.

Escaped Strings:

Escaped strings allow you to inject unprintable characters into your Normal String. Escaped strings take the following escape sequences:

\f Form feed

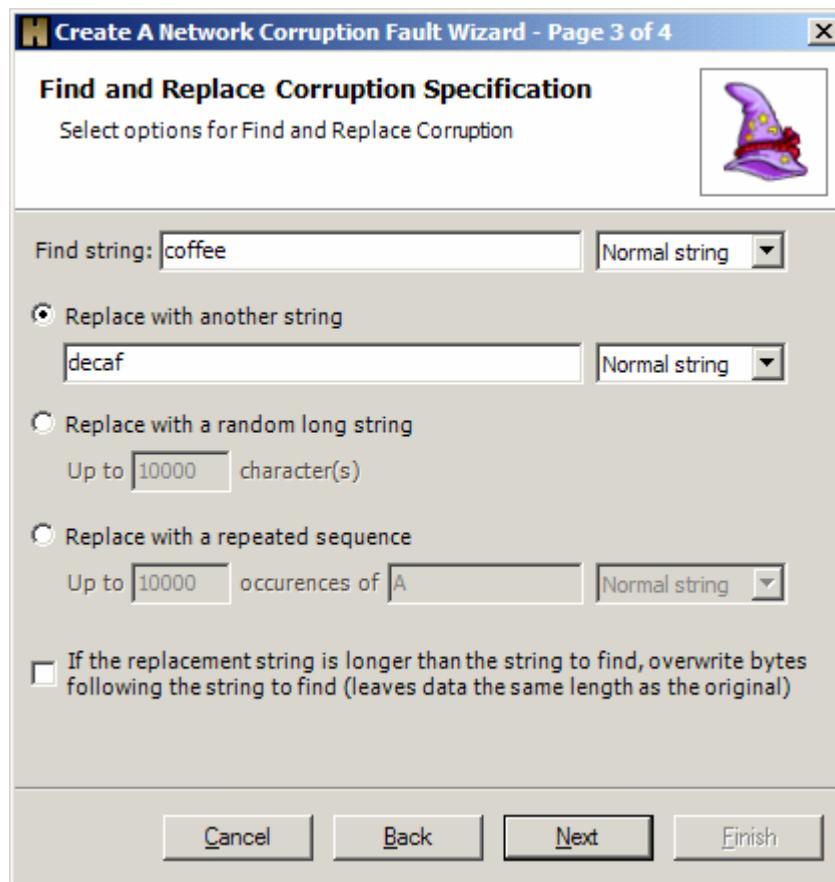
\n Newline

\r	Carriage Return
\t	Tab
\v	Vertical Tab
\xNN	Hexadecimal character (ex \x20 represents a space)
\\	Single backslash

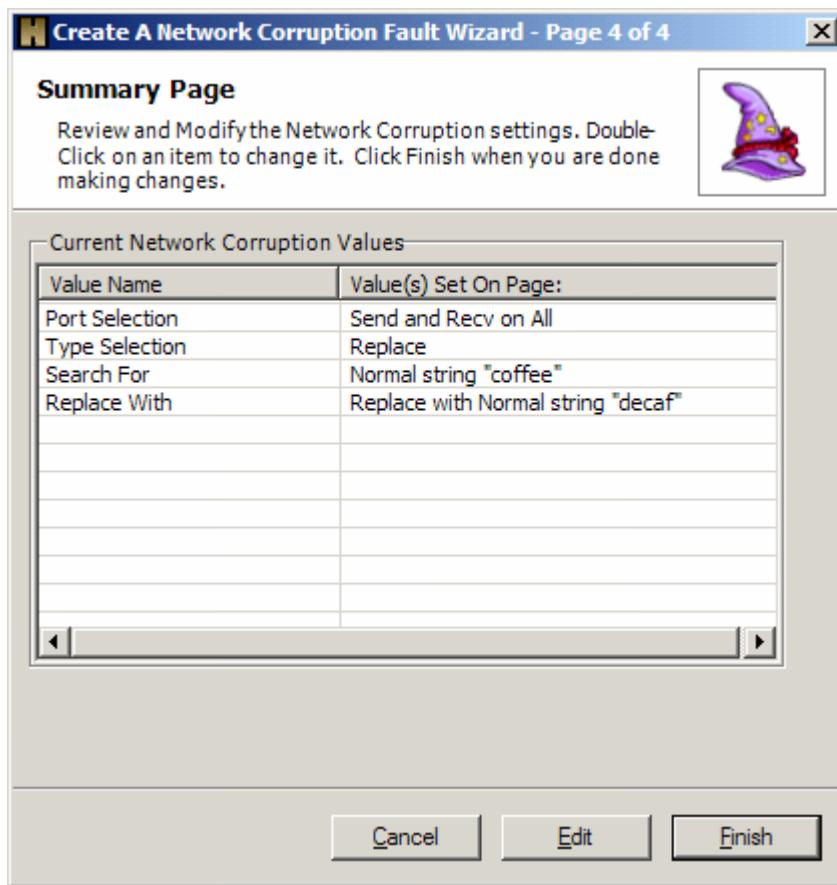
Example of an Escaped String:

\t\tTest\x04 – Converts to two tab spaces the word "Test" and the "End of Transmission" character at the end.

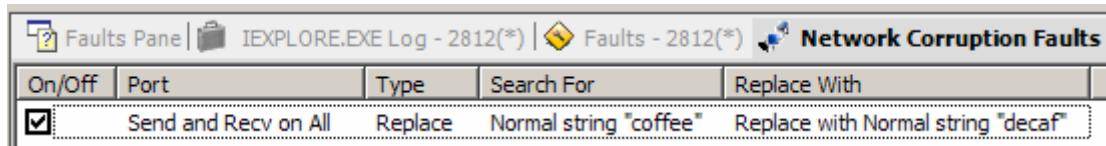
You can replace the matched string with a normal string, hex strings, escaped strings, random strings or a repeated sequence of a string. Since packet size is so important in Network messages you may opt to overwrite bytes following the string to find. If you overwrite bytes following the string the data length will be the same total length as the original.



The final page shows a summary of the corruption fault you just created. To edit any information on this page simply select it, and click edit. This will take you back to that page to make any necessary changes.



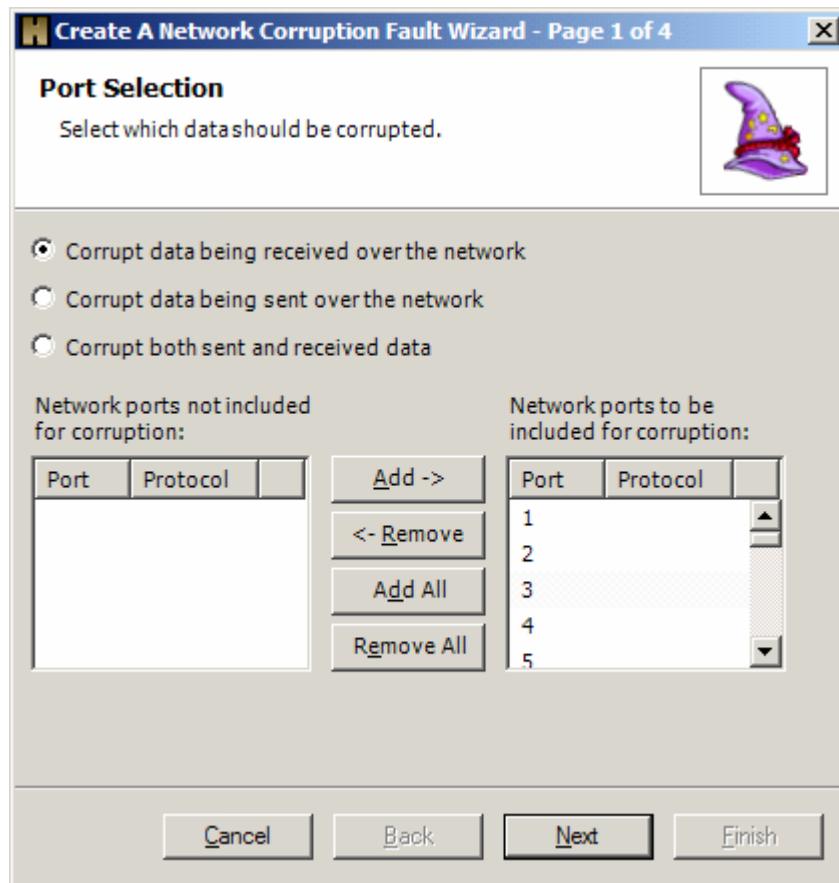
In the network corruption pane of Holodeck we can see the fault has been set to search for the normal string "coffee" and replace it with the Normal string "decaf"



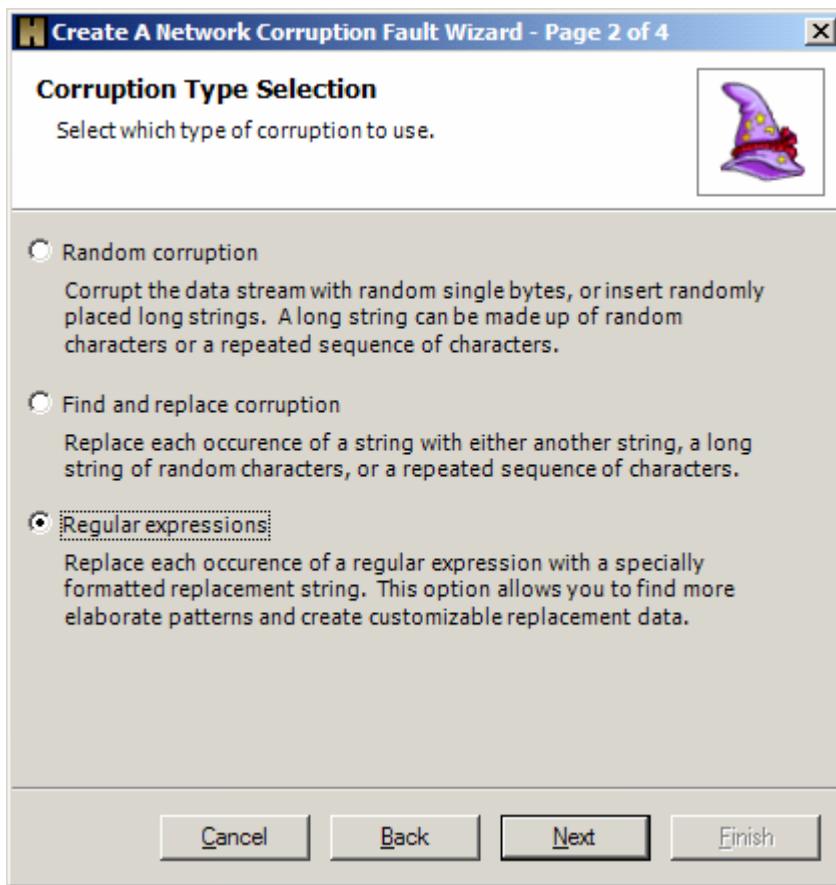
Creating a fault using Regular Expressions

Random Corruption is a good way of testing for the type of corruption that would happen as a result of common line noise.

On the first page of the wizard you can specify that Corruption should happen on data being sent, received, or sent and received. Corruption being received over the network will test your client's ability to parse corrupted network messages and corruption being sent over the network will test your server's ability to receive corrupted network messages. You can also specify to corrupt messages only being sent and received over certain network ports. You might want to focus your testing on a specific service that uses a specific port.



Select Regular Expressions as the method of corruption



The following examples used on the Regular Expression Corruption page of the Network Corruption Fault Wizard will replace any domain (ie www.intel.com) with a long string of "A's between 1000 and 5000 characters long.

Example Regular Expression:

The regular expression to match any domain (www.intel.com) is as follows:

```
([wW]{3,}\. )?([a b c e f g h i j k l m n o p q r s t u v w x y z ABC D E F G H I J K L M N O P Q R S T U V W X Y Z ]+)\.(com|net|org)
```

([wW]{3,}\.)?

this part matches an optional capital or lowercase "www."

([a b c e f g h i j k l m n o p q r s t u v w x y z ABC D E F G H I J K L M N O P Q R S T U V W X Y Z]+)

this part matches any length string consisting of upper or lowercase alphabet letters, greater than 1 (eg. "abc", "aRdfpQ", etc.)

\.(com|net|org)

this part matches a period

followed by any one of
the following, "com",
"net", or "org"

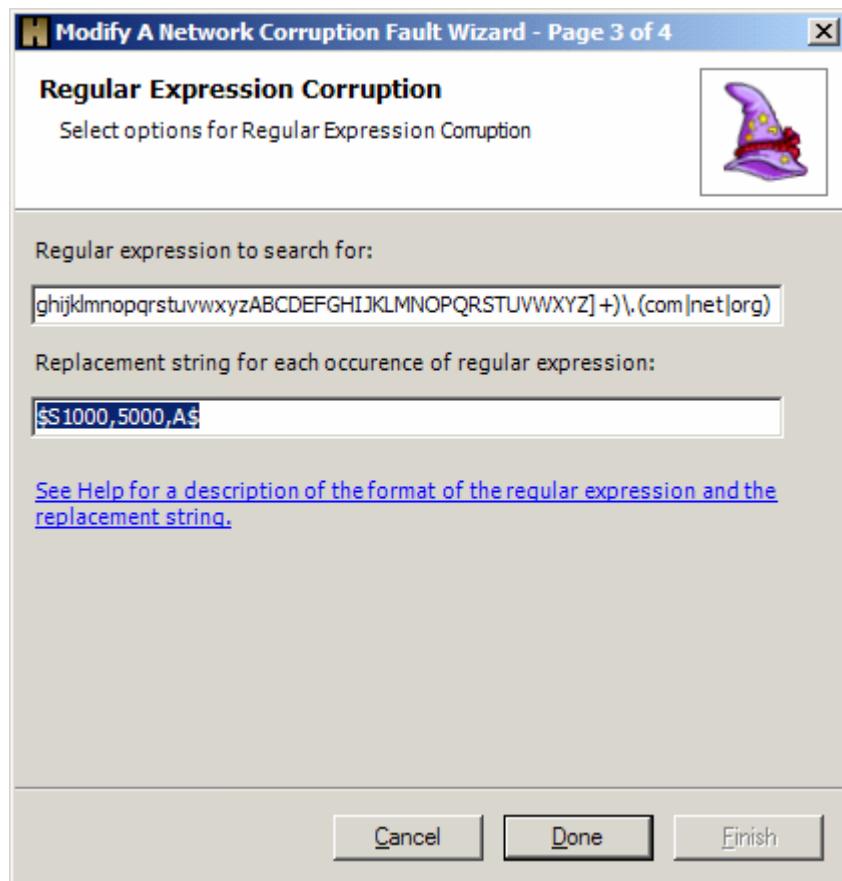
Example Expansion String:

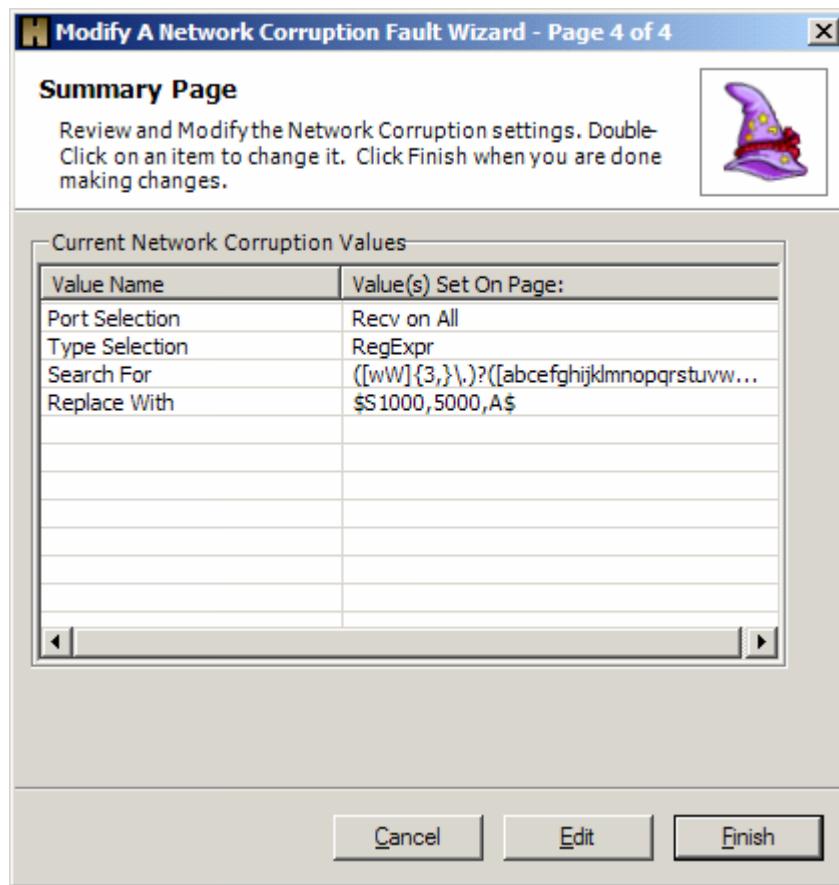
The expansion string to expand between 1000 and 5000 'A's is as follows:

\$S1000,5000,A\$

S	this will expand to a string
1000	the minimum number of repetitions
5000	the maximum number of repetitions
A	the string to expand
\$	By enclosing this string in '\$' Holodeck knows this is a string to expand.

For more information on Regular Expressions and Replacement Strings please see the related help topics.





Generating File Corruption Faults

File Corruption Overview

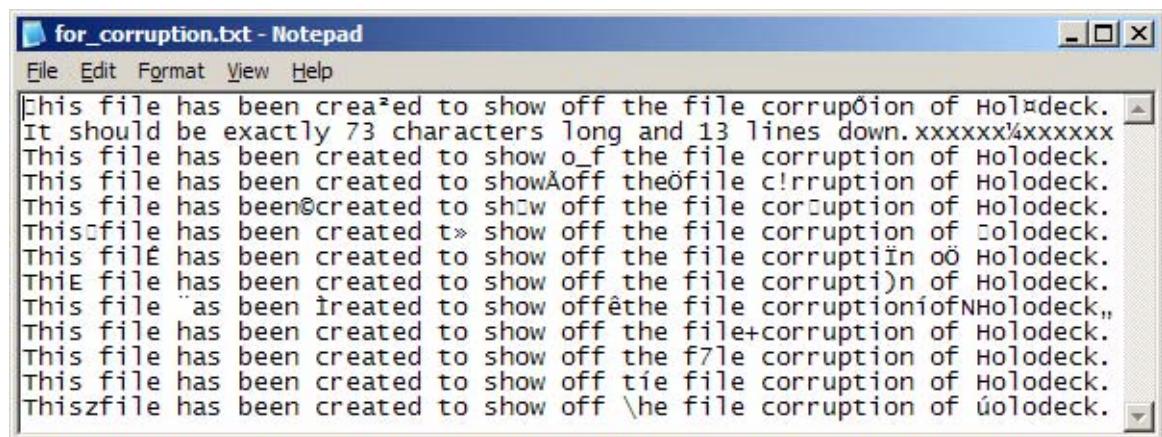
When you create a File corruption fault Holodeck does not corrupt the original file, but rather when the Application Under Test attempts to access that file it redirects the Application Under Test to a corrupted version of the file. This is useful because it allows you to corrupt a single file many different ways to test all aspects of file processing.

Please see the following pages for more information on File Corruption:

- [Create a File Corruption Fault using Random Corruption](#)
- [Create a File Corruption Fault using Find and Replace](#)
- [Create a File Corruption Fault with Regular Expressions](#)

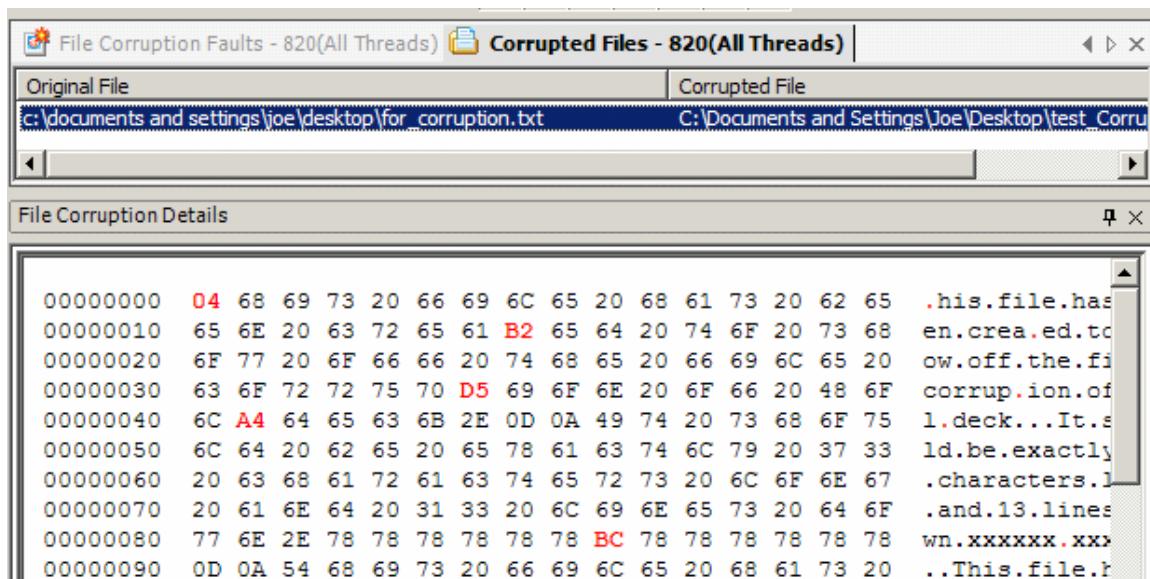
To create a file corruption fault click Application > "Create a New File Corruption Fault..." This will bring up the file corruption wizard.

Example of a corrupted file:



The original file remains untouched; Holodeck creates a new file in the Corrupted Files directory, the redirects the application to that file. All changes are stored in an XML file with the same name as the corrupted file.

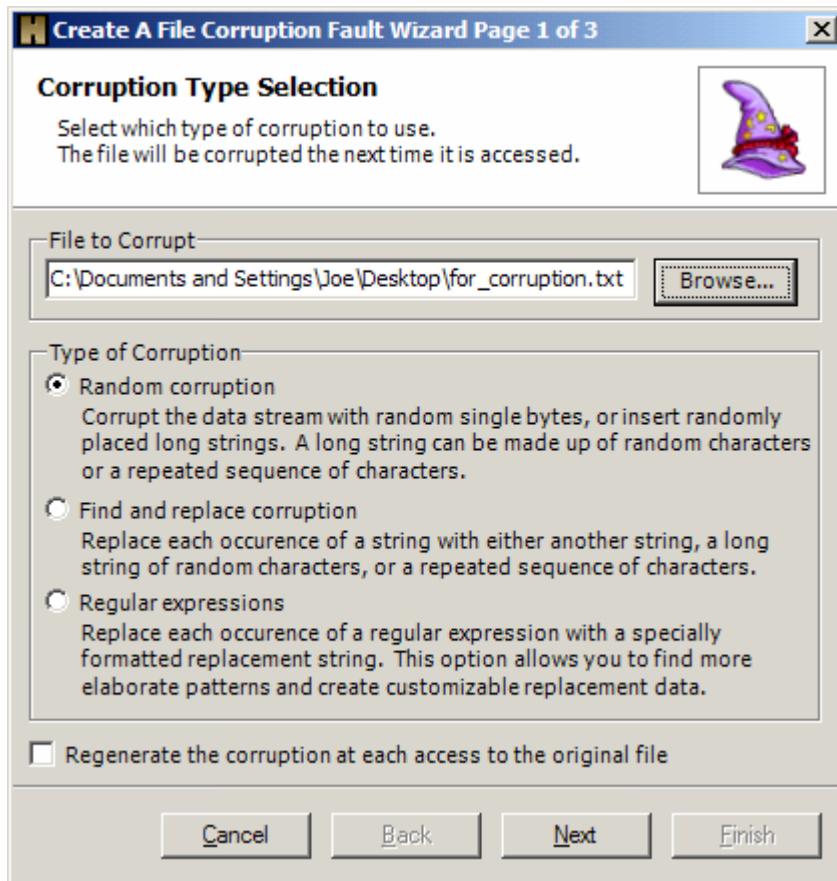
In the image below a corrupted file has been selected in the Corrupted Files Pane after the application had tried to access it. Holodeck then generated the corrupted file and added an entry in the Corrupted Files Pane. Selecting the corrupted file will load the corrupted file, with all corruption changes highlighted, in the File Corruption Details Pane.



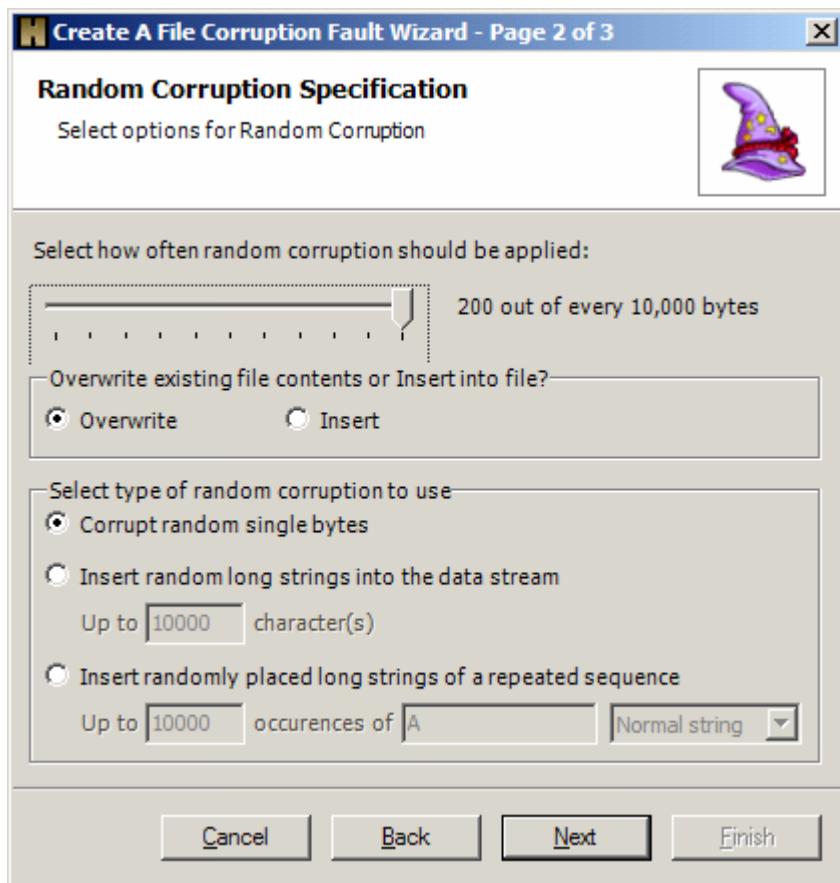
Create a File Corruption Fault using Random Corruption

Select the file you wish to corrupt by browsing to it using the Browse... button or type it directly into the "File to Corrupt" text box. Select Random Corruption as the type of corruption. This is a good way to flush out file parsing bugs, Holodeck generates completely random bytes which also generates many unprintable and unexpected characters.

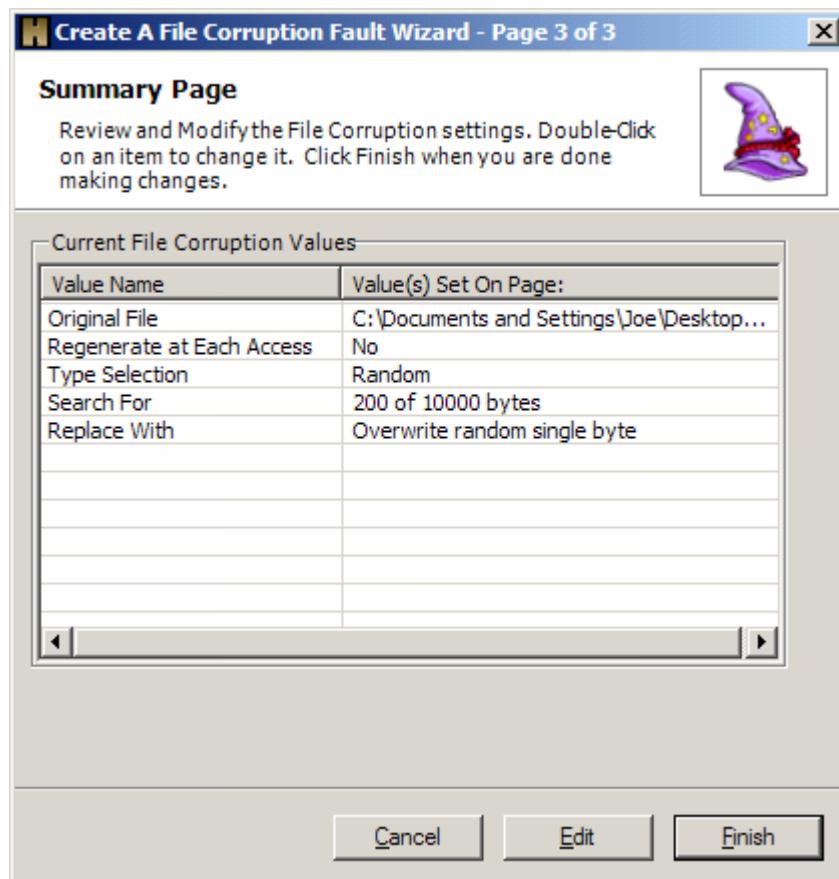
Holodeck creates a copy of the file to be corrupted and corrupts file on the first access, then uses that same file each time it is accessed unless "Regenerate the corruption at each access to the original file" is checked. Check this checkbox for better overall testing when using any kind of random corruption, uncheck this checkbox for help reproducing bugs.



In this case we want to corrupt the file as much as possible, to flush out easy file parsing bugs. More corruption will also be easier to find in the application once it has opened the file. We could opt to insert random characters into the file instead of overwriting them. Sometimes it is useful to insert a repeating character into the file for easy discovery within the application.



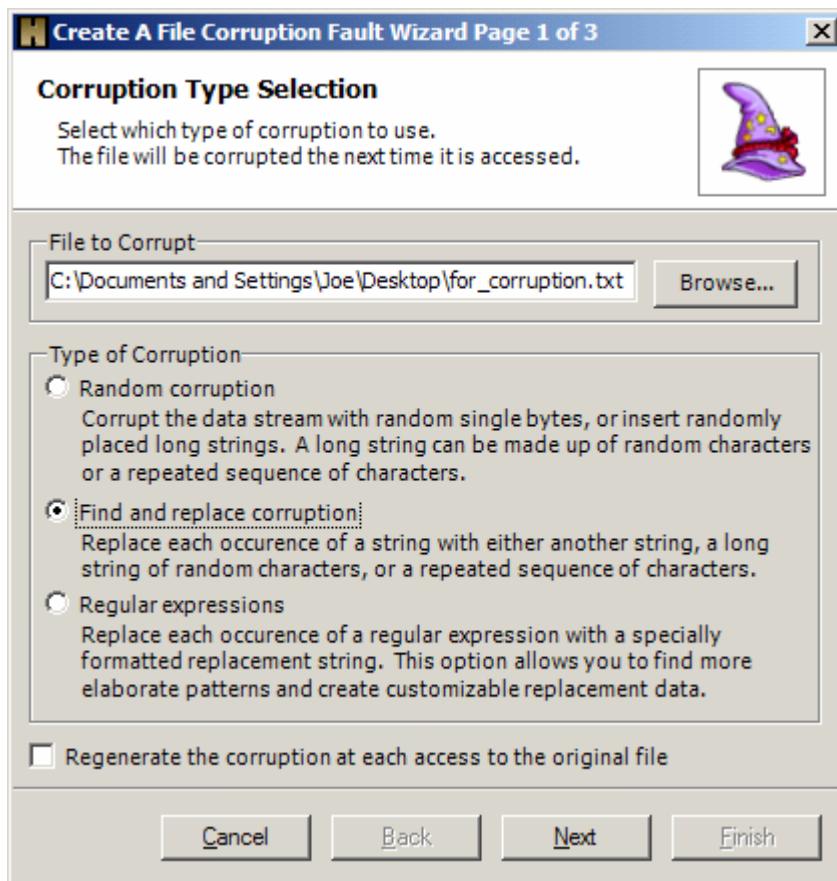
The Summary Page shows all the information about the wizard pertaining to this File Corruption Fault. You can go back and change any part of the fault by highlighting the value name and clicking edit. This will return you to the page in the wizard so you can change the options.



Create a File Corruption Fault using Find and Replace

Select the file you wish to corrupt by browsing to it using the Browse... button or type it directly into the "File to Corrupt" text box. Select Find and Replace as the type of corruption. This is a good way to focus your corruption to a single word, escaped string, or hex string.

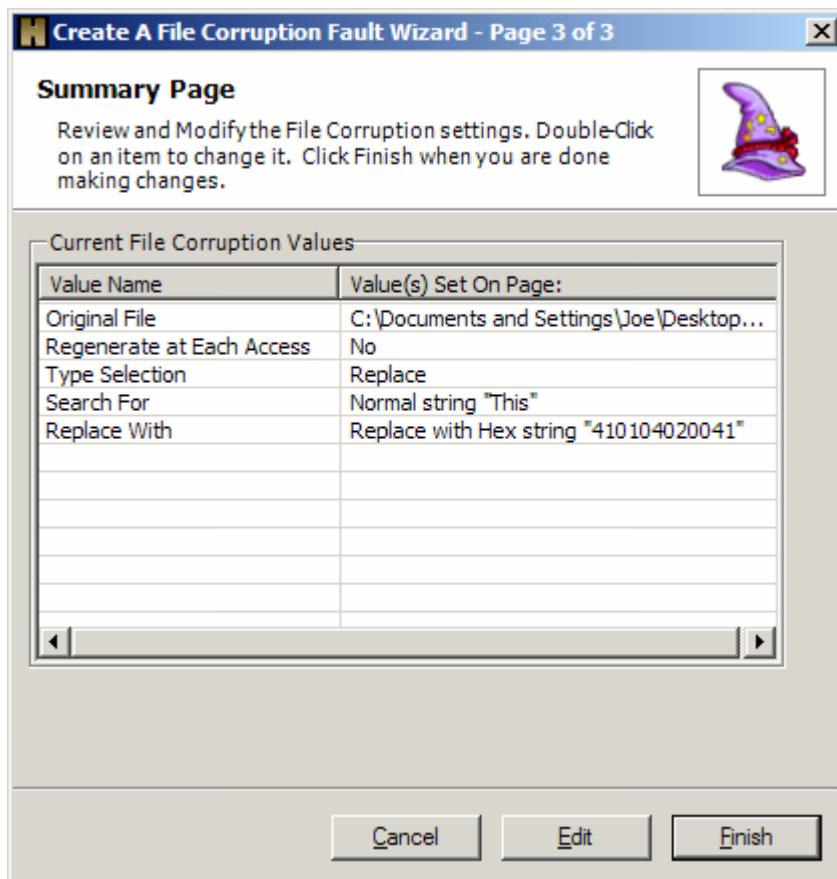
Holodeck creates a copy of the file to be corrupted and corrupts file on the first access, then uses that same file each time it is accessed unless "Regenerate the corruption at each access to the original file" is checked. Check this checkbox for better overall testing when using any kind of random corruption, uncheck this checkbox for help reproducing bugs or if you are performing the same find and replace corruption.



In this case we want to search for any occurrence of the normal string "This" and replace it with a number of unprintable Hex characters. The example here shows a string of five unprintable characters preceded and followed by the Hex value 41, which represents 'A'. I did this so that if the program parses the string properly, I will have a starting and ending place. If an error occurs I will be able to find the character that caused the error.



The final page shows a summary of the corruption fault you just created. To edit any information on this page simply select it, and click edit. This will take you back to that page to make any necessary changes.

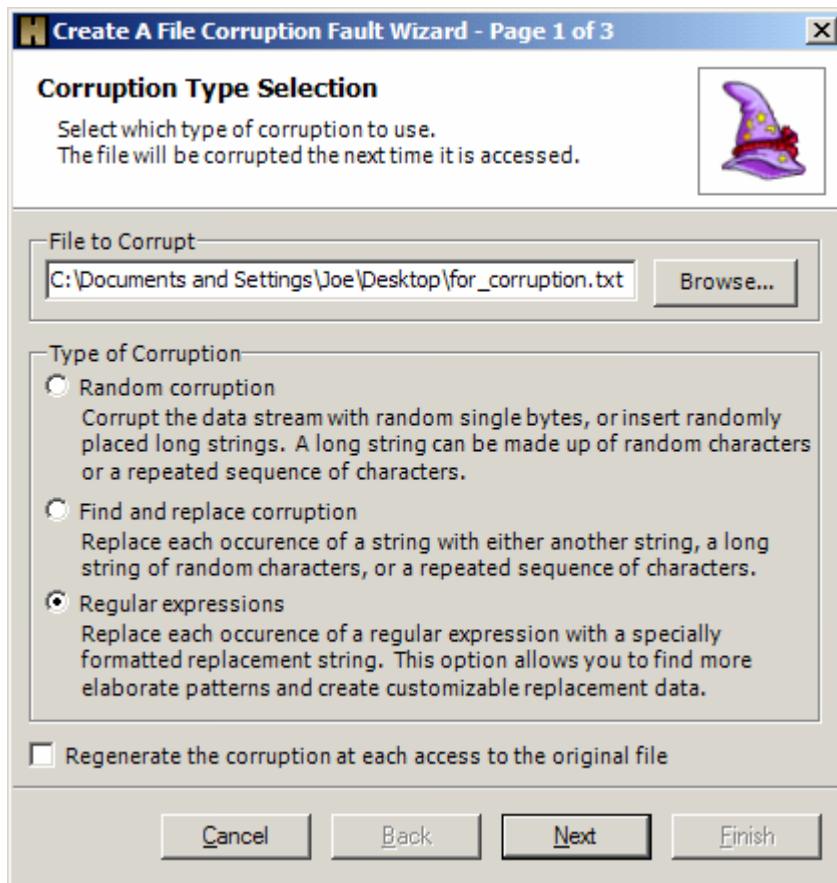


Create a File Corruption Fault with Regular Expressions

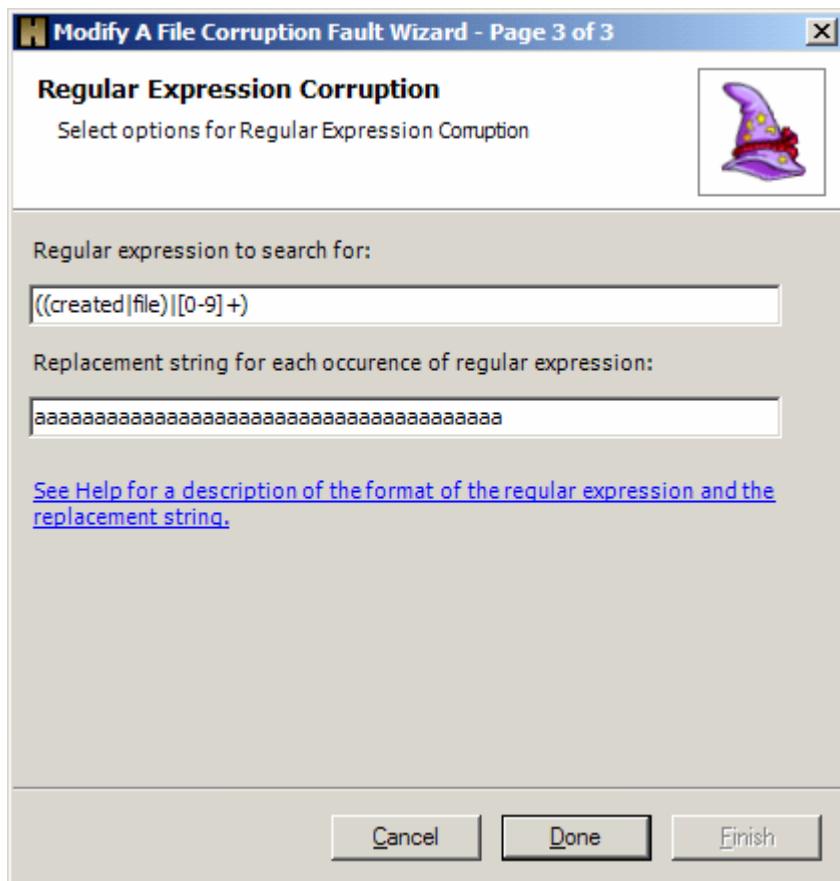
Select the file you wish to corrupt by browsing to it using the Browse... button or type it directly into the "File to Corrupt" text box.

Select Regular expressions. This will allow you to search for a number of different strings, and replace them with normal strings or with expansion strings. This is very useful for matching on filenames, web addresses, or other well formed, but unique, strings.

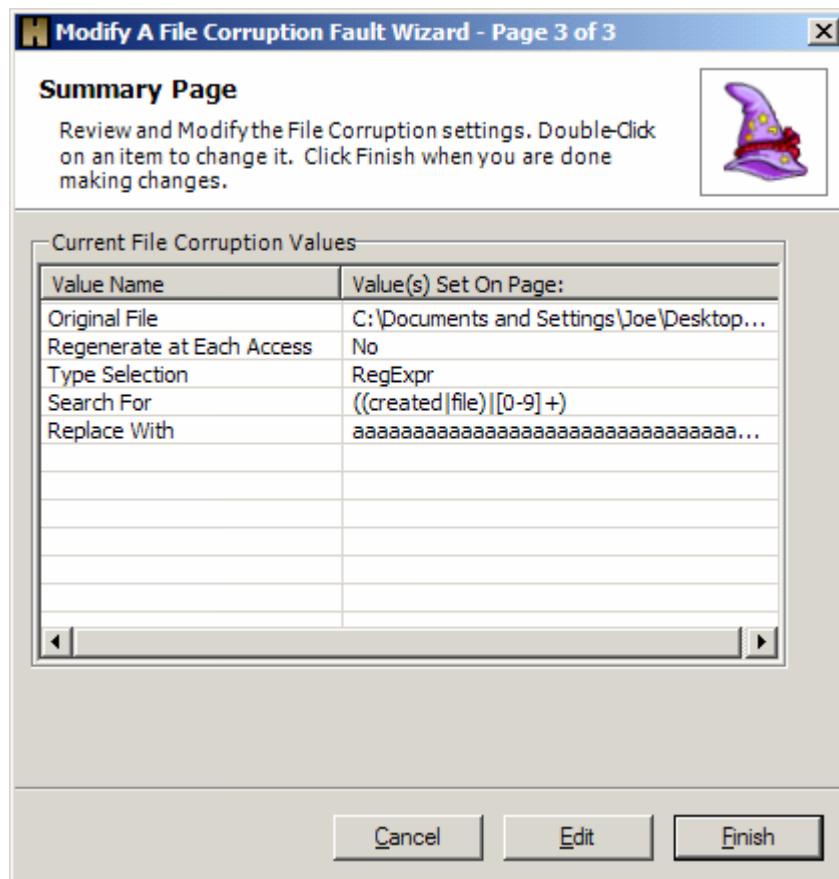
Holodeck creates a copy of the file to be corrupted and corrupts file on the first access, then uses that same file each time it is accessed unless "Regenerate the corruption at each access to the original file" is checked. Check this checkbox for better overall testing when using any kind of random corruption, uncheck this checkbox for help reproducing bugs or if you are performing the same find and replace corruption.



This regular expression: ((created|file)|[0-9]+) will match the word "created" or the word "file" or a string of numbers of any length. My file contains all three of these things so Holodeck will replace any of these occurrences with the long string of 'a's below.



The final page shows a summary of the corruption fault you just created. To edit any information on this page simply select it, and click edit. This will take you back to that page to make any necessary changes.



Regular Expressions and Replacement Strings

Using regular expressions

Regular expressions can be helpful while trying to match a well formed string that may change over time. Regular expressions are used in matching data being sent over the network in the network corruption faults as well as use in file corruption faults.

Note: to replace a string please see the Using replacement strings section.

Regular expression examples:

Match any IP address, with optional port number:

[0-9]+\. [0-9]+\. [0-9]+\. [0-9]+(:[0-9]+)?

[0-9]+ – represents a random number between 0 and 9 the following ‘+’ means there can be a string of 1 or more of these numbers.

\. – Since ‘.’ character is a reserved character it must be escaped. Escape reserved characters by using the ‘\’

((:[0-9]+)?) – This section matches a colon first, then a string of random numbers, the final ‘?’ makes this entire section optional.

Match a sequence of 10 or more consecutive "A" characters:

A{10,} – A sequence of the preceding group is represented by {min, max} this regular expression matches a sequence of 10 or more of the character ‘A’. If you leave a number for the max out of the expression Holodeck assumes any length.

Match any domain is as follows:

([wW]{3,}\.)?([a-zA-Z]+)\.(com|net|org)

([wW]{3,}\.)? - this part matches an optional capital or lowercase "www."

([a-zA-Z]+) – this part matches any length string consisting of upper or lowercase alphabet letters, greater than 1 (eg. "abc", "aRdfpQ", etc.)

\.(com|net|org) – this part matches a period followed by any one of the following, "com", "net", or "org"

Regular Expression Definition:

All characters except "()", "{}", "\$", "\", "?", "/", ".", "*", "+", "^", and "|" can be used normally. C escape sequences are defined. To use a reserved character, place a "\" character in front of it (for example, "\+" represents a literal "+" character).

The "." character matches any single character.

A "*" after an expression matches repetition of zero or more times, whereas a "+" character represents repetition of one or more times. A range of repetition counts can be specified with {}, i.e. "a{2,5}" matches between 2 and 5 repetitions of "a".

A "?" following an expression makes it optional (can be present exactly zero or one times).

A "|" between two expressions represents an either-or match. The expression "get|set" will match either "get" or "set".

Character sets can be enclosed in []. The expression "[Gg]et" would match either "Get" or "get".

Sub-expressions can be enclosed in (). These sub-expressions can be referred to directly in the replacement string by numerical order starting at one.

Nonprintable characters can be represented with a C-style octal or hexadecimal escape sequences, i.e. "\0123" for octal or "\xAB" for hexadecimal.

Null characters can be matched using "\00".

Regular expressions match the string closest to the start of the buffer first, and the length of the matched string second.

Using replacement strings

Replacement strings are useful when trying to insert a string into a stream of data that might have to be formed in a precise manner. This is useful for trying to create an improper string that will be checked as correct by the first or second string checking statements. These replacement strings replace whatever is found when using find and replace corruption or regular expression corruption.

Replacement string examples:

Replace with a random long string of between 500 and 1000 characters, followed by a string of 100 "A" characters:

`$R500,1000$$S100,100,A$` – This will replace the matched string (regular expression, or find and replace string) with a random string followed by a 100 'A' characters.

`$R500,1000$` - the first 'R' tells Holodeck to use Random strings, and expand it from 500 to 1000 characters long.

`$S100,100,A$` - the first 'S' tells Holodeck to use the following string and expand it exactly 100 times, since the minimum and maximum are the same.

Duplicate the matched string 500 times:

`$S500,500,$$&$` – The first 'S' tells Holodeck to use the following string.

`$&` – expands to the previously matched string (from the find and replace or regular expression corruption.)

Insert 100 random characters just before the matched string

`$R100,100$$&`

`$R100,100$` – expands to exactly 100 random characters.

`$&` – expands to the previously matches string (from the find and replace or regular expression corruption)

Replacement strings definition:

All characters except "(", ")", "\$", "\", "?", ",", and "/" can be used normally. C escape sequences are defined, as well as the following expansions:

`$`` Expands to all data from the end of the previous match to the start of the current match.

`$&` Expands to all of the current match.

`$0` Expands to the entire current match.

`$N` (`N` is 0-9) Expands to matched sub-expression `N`.

`$Rmin,max$` Expands to a random string of between `min` and `max` characters.

`$Smin,max,seq$` Expands to a string of between `min` and `max` repetitions of character sequence `seq`.

`?Ntrue:false` If subexpression `N` has matched, expand true expression, otherwise expand false expression (or nothing if false expression is not specified).

`\` Expands to a normal / character

Multiple Threads and Processes

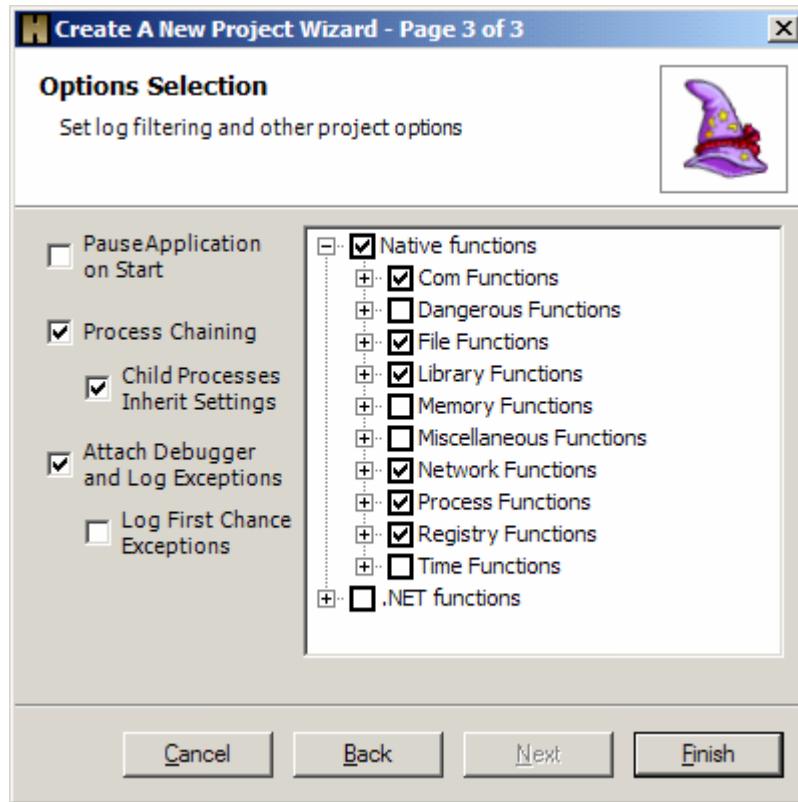
Dealing with Multiple Processes

This option is available if you check the Process Chaining option on the last page of the Holodeck Project Setup Wizard. If the application spawns multiple processes, or if you choose to test multiple applications simultaneously Holodeck will allow you to set faults, tests etc for each process.

Holodeck allows you to test multiple processes or applications at the same time. This is useful if your application spawns new processes while running or if you simply want to add another application to the project to test two separate applications in the same Holodeck Project.

Automatically Testing new processes your application spawns:

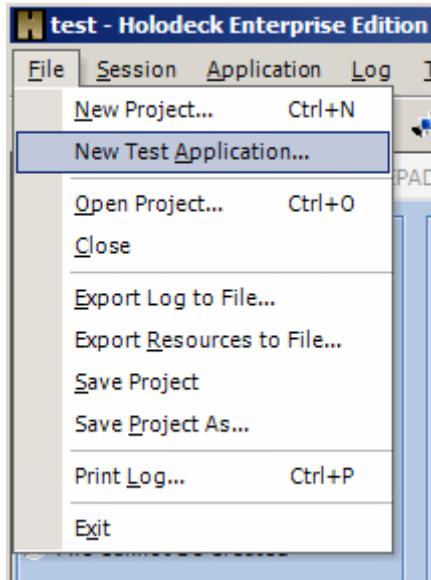
To have Holodeck automatically attach to any process your application spawns check "Process Chaining" on the last page of the New Project Wizard. If you would like the new process to inherit all the tests, limits, faults, etc. that you are currently using with the main application check "Child Processes Inherit Settings" on the last page of the New Project Wizard.



Each new process the Application Under Test spawns can be treated like its own application. You can set limits, faults, and tests on each new process, as well as viewing resources specific to that process.

Testing multiple applications at the same time:

To test multiple applications at the same time select File > New Test Application, this will bring up the New Test Application Wizard to select an additional Application Under Test. You can specify different APIs to log for each Application Under Test, by selecting the functions under Log > Functions Logged...

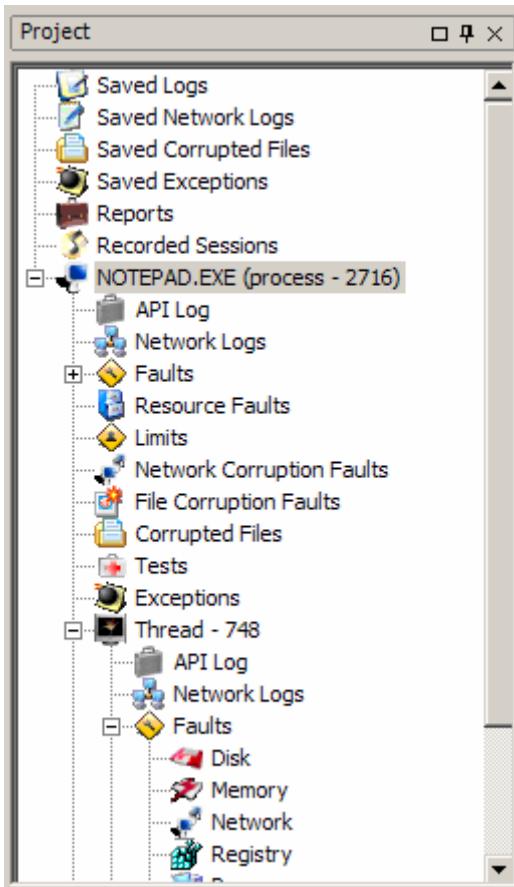


Working with Multi-threaded applications

Holodeck will automatically track any new threads that are created by the process you are testing. If Holodeck is in Per-Thread mode each new thread will show up in the Project Pane as a child of its parent process. This can be extremely useful for testing specific threads, such as threads in charge of reading a file or parsing incoming network traffic.

To turn on Per-Thread mode click Session > Per Thread from the Menu.

Each new thread the application spawns can be treated like its own application, allowing you to set limits, faults, and tests on each thread individually. Each fault, limit, or test set on a single thread will only apply to that thread, however, setting a fault, limit or test on the main process node will create that fault, limit or test for all Threads of that process.



To change limits, set faults etc. expand the thread by clicking the '+' sign next to the thread.

Double clicking an active log node in the Project Pane will open a log pane that will show just entries for the thread you chose.

Exceptions and mini-dumps

Exceptions Overview

Holodeck catches all exceptions your application produces. Exceptions are routed through Holodeck first, as a first chance exception, then passed back to the application to handle. If the application does not handle the exception it is logged as a second chance exception.

View the Exception in a debugger by right clicking an exception in the Exception Pane and selecting "View Exception in Debugger." If you have a debugger attached to the application the debugger may catch the second chance exception allowing you to handle the exception in other ways. If source code is available your debugger may be able to use the exception information provided by Holodeck to step into the exception causing function.

Exceptions will only be handled by Holodeck if the attach debugger option is selected. To enable or disable check the "Attach Debugger and Log Exceptions" option in the Application menu.

Mini-dump Overview

You can save the mini-dump information for debugging the application at a later time. The mini-dump file is a snapshot of the computer state at the time of the crash. Matching images must be available because mini-dump files store very little information; they store only some of the volatile information at the time of the crash. They do not store the basic code streams that the computer loaded into memory. Instead, to save space, the mini-dump file stores only the name and time stamp of the images loaded on the crashing computer.

To examine the code that was running on the crashing computer, the debugger must be given access to the same binaries that the crashing computer was running. The debugger uses the name and time stamp stored in the mini-dump file to uniquely match and load the binaries when the developer wants to debug the crash.

Recording and replaying sessions

Introduction to recording and replaying

You can create a Recorded Session while testing the Application Under Test using Holodeck by using the **Record Session** Functionality. This will allow you to reproduce a test or bug more precisely than trying to remember and write down steps. Recorded Sessions are global to your entire project, so they will record changes to all of the process and threads being tested by Holodeck.

While recording Holodeck records each change in Holodeck including when faults, tests, limits etc are set and lifted. Recording does not save UI changes, input, or any changes from a test harness.

Holodeck records every application in the current Holodeck Session, however, if no changes are made to the application it will not interfere with the recording of the other applications.

Holodeck's record adding, deleting or modifying the following things:

- Limits
- Tests
- Faults
- Resource Faults
- Network Corruption
- File Corruption

Use Recorded Sessions as a Macro

You can use a recorded session as a macro. Start a recorded session, apply faults, limits, tests etc and stop the recorded session. Now at anytime during testing you can restart your application and replay exactly the same faults, limits tests etc. based on the incoming logs.

Creating a recorded session – Create a recorded session to replay Holodeck's interaction with the application.

Replaying a recorded session – Replay a recorded session to have Holodeck automatically set faults, tests limits etc.

[Next >>](#)

Creating a recorded session

A Recorded Session can be created by any of the following methods

- 1) From the Project Pane
- a) Right Click the Recorded Sessions node in the Project Pane
- b) Click Create a Recorded Session
- 2) From the Menu
- a) Click Session -> Record Session

Once a Recorded Session has been started Holodeck will log every change within Holodeck, faults, limits, tests, etc are all recorded in the recordedSession.xml file.

Note: Holodeck does not record UI testing, or input that must be handled by the test harness.

When you are finished recording your test session stop recording by...

- 1) Click Session -> Stop Recording
- 2) Right click the newly created test and click Stop Recording

Note: when recording a session Holodeck records all changes made in Holodeck to all applications for the current session.

<< Previous Next >>

Replaying a recorded session

Replaying a recorded session and ensure Holodeck sets and lifts the same hostile environments every time it is run. Use this every time test scripts are run so you can ensure reproducibility of bugs, and ensure you are testing the same things on each different build.

When a recorded session is replayed Holodeck will set each fault, test, limit etc. according to the logs coming in, as it was set when recording.

- 1) Right click the Recorded Session you would like to replay and select Replay from the menu.
- 2) Select the Recorded Session you would like to replay and select Replay Session from the Tests Menu
- 3) This will start the recorded session; Holodeck will automatically restart the Application Under Test and begin the repro steps exactly as they were recorded. Playing a Recorded session will set all the faults, limits, tests, etc. that were set while recording at the same relative time to application start.

Add Holodeck Intercepts

Add Holodeck Intercepts

Using Holodeck's add new Intercepts Wizard you can easily add the ability to intercept any function in any dll. Holodeck can scan a dll for public functions and generate source code to create an interception dll with your replacement function. This allows you to intercept any custom dll.

Create Imposter dlls with the Add Holodeck Intercepts Wizard

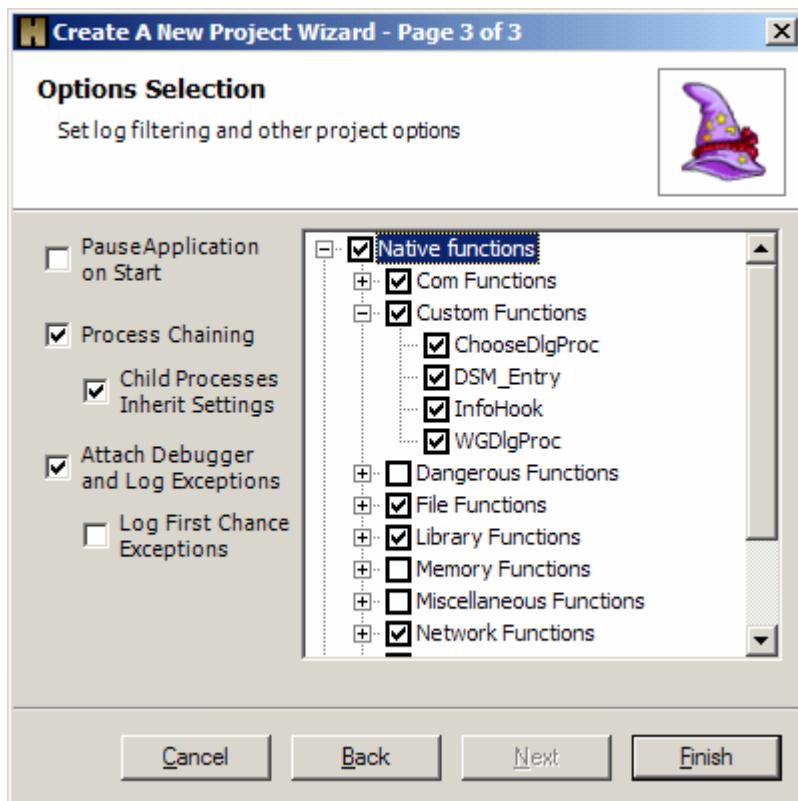
Select the functions to add to the Holodeck Database – The first page of the wizard allows you to select which replacement libraries you would like to create and which functions you would like to intercept.

Review the Definitions of the Functions – If there is any more information Holodeck requires to create the intercepting functions you may specify that here.

Review Your Choices – Before Holodeck generates the source code for the replacement library you can review the choices you have made thus far.

Replacement Library Results – Once the replacement library source code has been generated Holodeck gives you instructions on how to build your new project using Visual Studio.

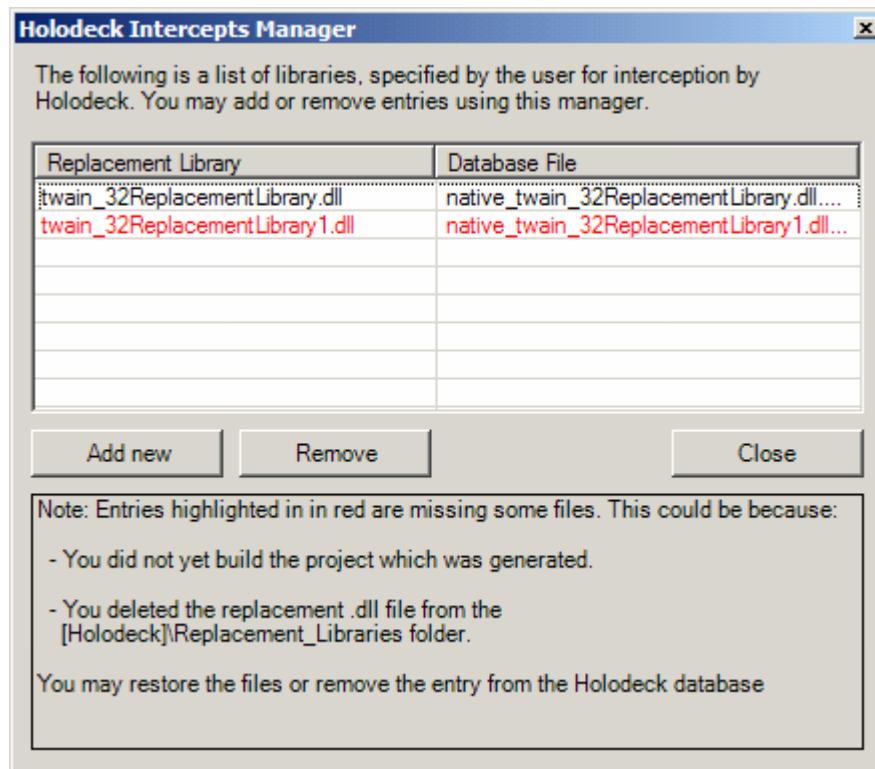
After a replacement library has been created and loaded into Holodeck you can intercept it like any other function supported by Holodeck out of the box. You can view logs, create tests, create custom faults etc. to the new Custom intercept functions.



Once you have created a replacement dll Holodeck will automatically load it into the replacement dll folder in the Holodeck install directory. To add, remove or change any of the replacement dlls you have created use the Holodeck Intercepts Manager.

Holodeck Intercepts Manager

The Holodeck Intercepts Manager allows you to add or remove new interception libraries. Libraries that require your attention are highlighted in red. You can either remove these libraries, or build them in Visual Studio. If the library is missing Holodeck will ignore the replacement functions.

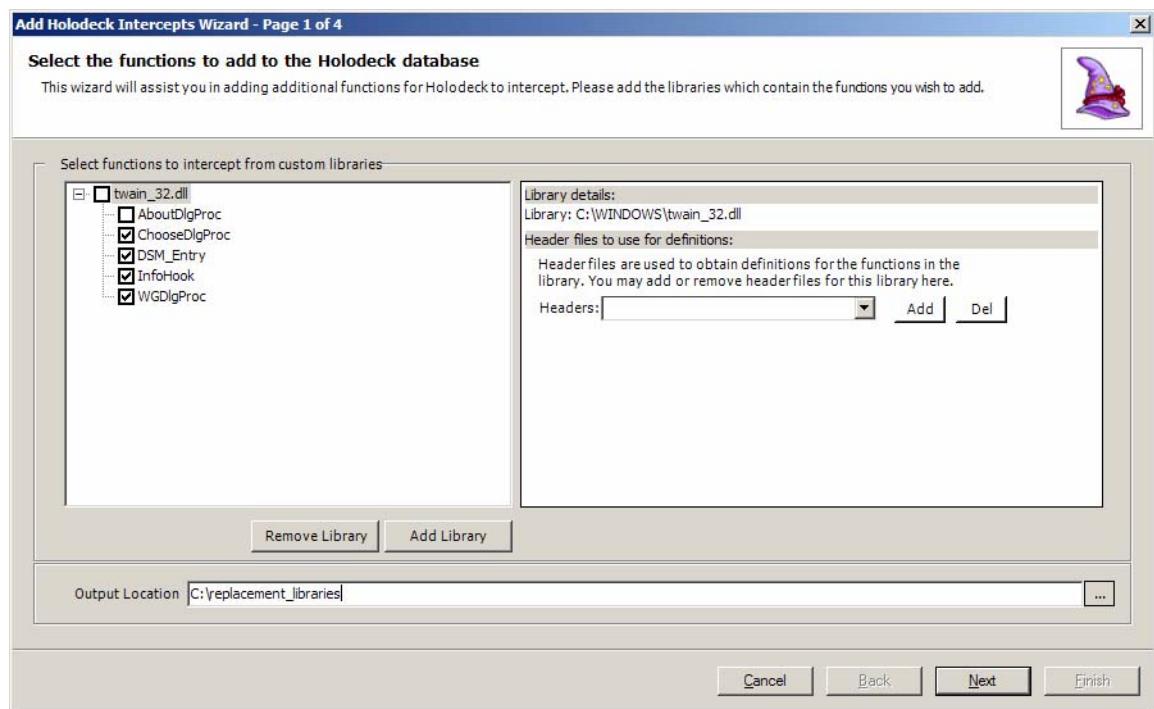


Add Holodeck Intercepts Wizard

Select the functions to add to the Holodeck Database

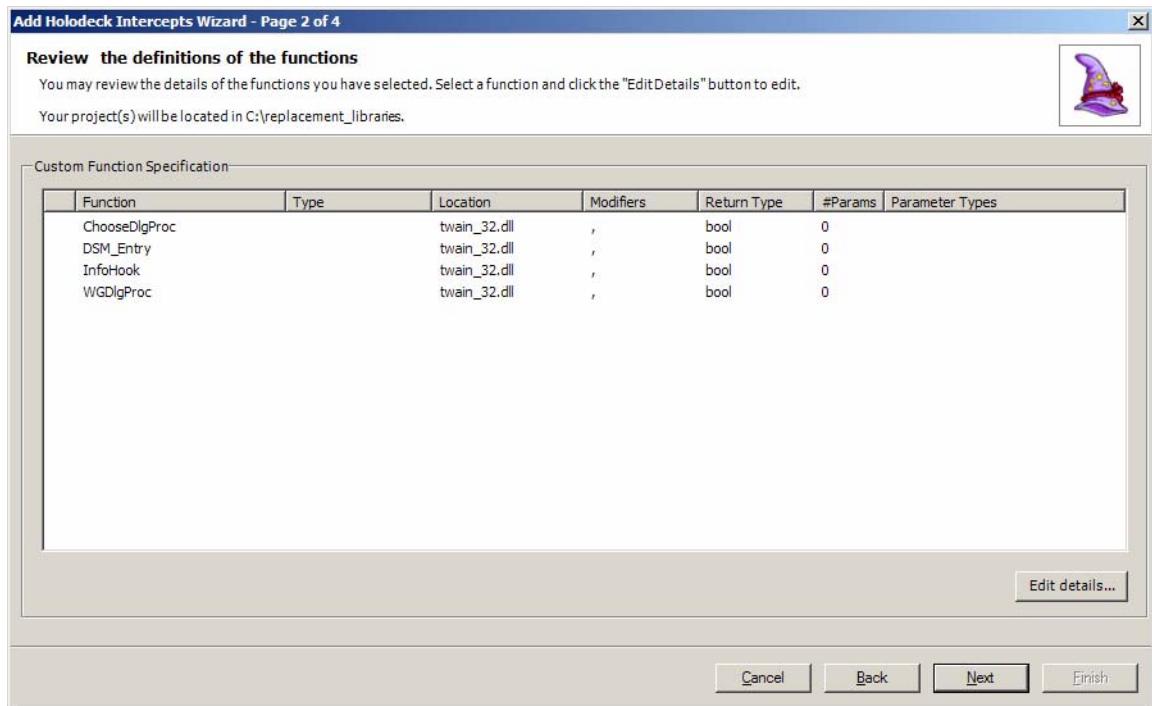
This page of the wizard allows you to add libraries to your project. Holodeck will automatically scan the dll to find the correct functions. Check the functions you would like to intercept. If you have a header file Holodeck can parse the Header file for additional information regarding the intercept functions.

Next Review the Definitions of the Functions to add any more information required to generate the project.



Review the Definitions of the Functions

If Holodeck requires any additional information to create the project you can specify that here. Each replacement function can be edited by clicking the edit button in the lower right of the window.



The Edit Function Details dialog box allows you to change which category Holodeck intercepts it as, change the return type, and any type modifiers.

Category – The category Holodeck will use when intercepting and logging this function.

Return Type – The type of variable this function will return, this field is required and must be supplied by the user or a header file.

Type Modifiers – These control how the function is stored in the binary for example: __declspec noreturn

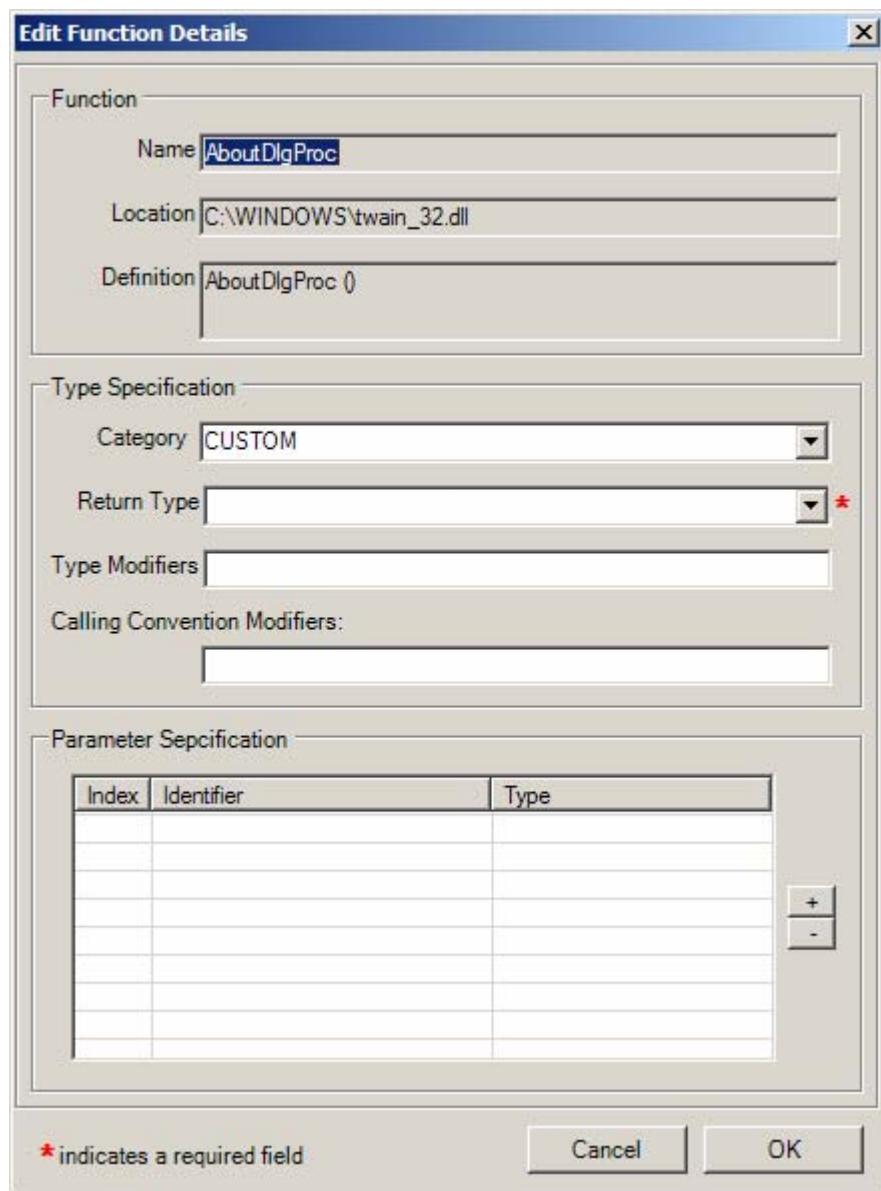
Calling Convention Modifiers – These are keywords which control the way the function is called, or a mechanism to decide what and in what order to put the function on the stack. Examples are __stdcall, __pascal or __fastcall.

Parameter Specification – This table contains information about the parameters of this intercepted function.

Index – the zero based index of the parameter.

Identifier – the name given to the parameter.

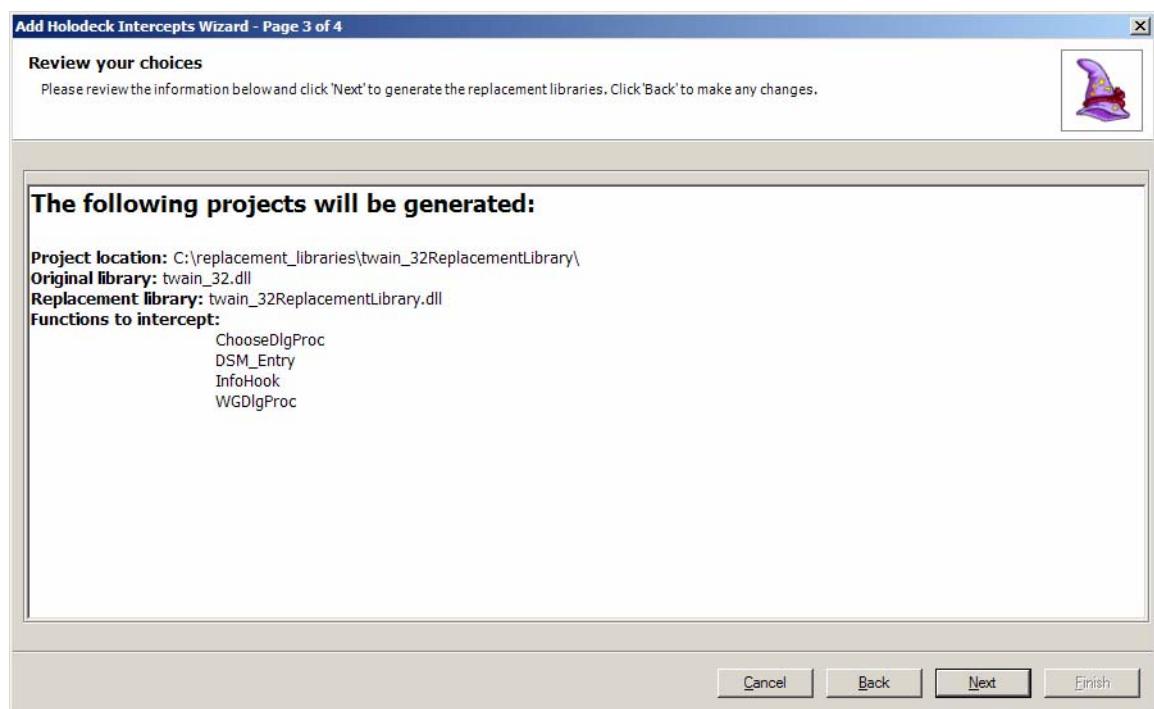
Type – the type of the variable



Review Your Choices

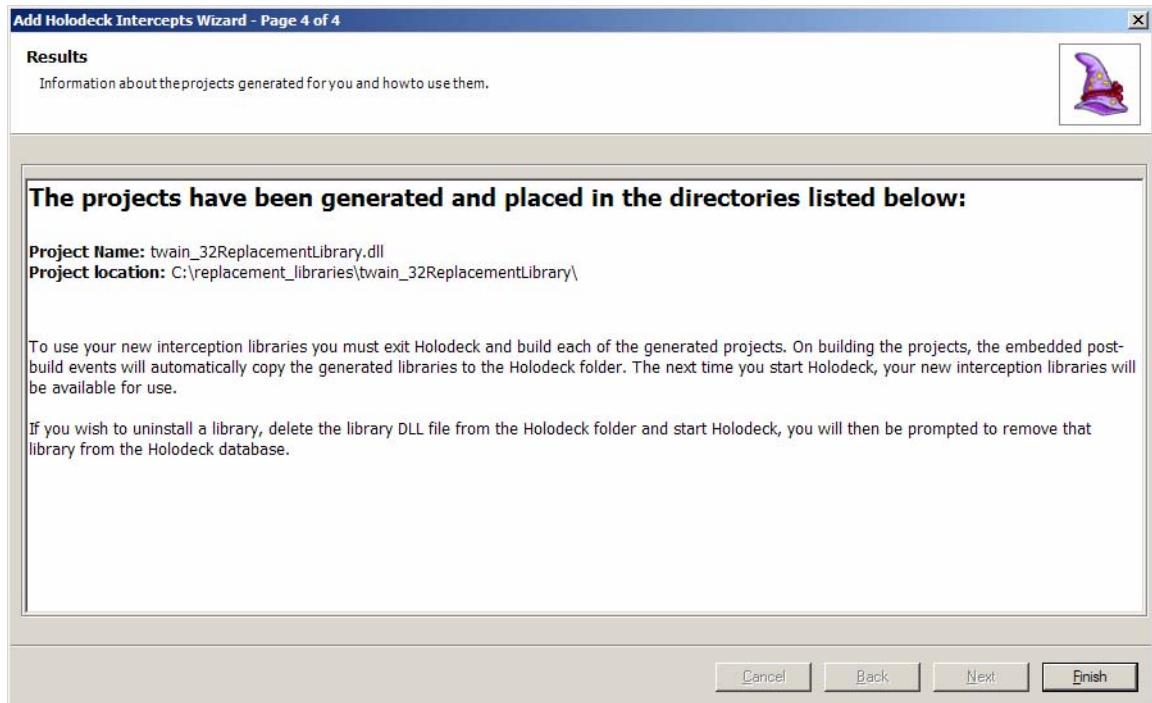
This is the final page of the wizard before Holodeck generates your project. If you see any problems with how Holodeck is going to generate the project this is your final chance to go back and edit the settings. Edit the settings for this project by clicking the back button. If all the settings are in order click next and Holodeck will generate the source files and Visual Studio project files.

Continue to the Replacement Library Results page where Holodeck gives you instructions on building your project.



Replacement Library Results

This page shows you information pertaining to the recently generated project. Load your new project into Visual Studio, change the functions Holodeck has created and build the project. When you build the project the embedded post-build events will automatically copy the generated libraries to the correct Holodeck folder.



The next time you start Holodeck, your new interception libraries will be available for use in the Options Selection page of the New Project Wizard. You can intercept it like any other function supported by Holodeck out of the box. You can view logs, create tests, create custom faults etc. to the new Custom intercept functions.

Note: Under certain circumstances the generated solution will not compile. On compile Visual Studio will throw an error. To fix the error add the necessary reference.

Add a reference to Sample.dll

Adding a dll by reference is how you can load external library functions, this will give you access to all the public methods in the library.

1. In the solutions pane, right click the references node and select "Add Reference..."
2. Click on the Projects tab

3. In the "Add Reference" dialog choose the necessary dll and click OK.
4. Add `using NameOfDLL;` (no ".dll") to the top of the source file with the other using statements.

Automated Testing with Holodeck

Introduction to Automating Holodeck

This section includes more advanced topics about Holodeck such as using Holodeck with other test frameworks, using with a command line, and the customizing and extension of Holodeck.

Using Holodeck with other Automated Test Frameworks - If you already have an automated test framework in use you can use Holodeck to test your applications further. You can use the test framework to record the session or to catch errors as they happen.

Introduction to using Holodeck as a command-line Utility - Holodeck can be run by on the command line, which makes it easy to launch with test scripts.

Create a Custom Test Project Wizard - To insert your own custom intercepts for testing custom dlls or functions Holodeck has included the Custom Intercept Generator which will walk you through the process of creating an impostor dll.

Using HolodeckLib – a simplified wrapper for HoloScript, HolodeckLib has much of the needed functionality without some of the complexity.

HoloScript – a set of libraries built on top of the HeatAPI that can be used to create new Holodeck interception frameworks.

HEAT – (The Hostile Environment for Application Testing) is the foundation upon which Holodeck has been built.

Using Holodeck with other Automated Test Frameworks

Using Holodeck with other Automated Test Frameworks

Holodeck can be used in conjunction with your already existing Automated Test Frameworks to increase the probability of finding bugs. Holodeck can be used for ad-hoc testing by manipulating tests, faults, limits in the UI while running your test framework. Holodeck can also be used in a lab environment or stress test environment. For stress testing Holodeck expects a stress test harness to launch it, telling it the stress options and Application Under Test on the command line. For code coverage testing Holodeck expects to be launched by a test harness as well. If there is a crash in the Application Under Test the harness is responsible to do the right thing (launch debugger, record crash, etc).

Using Holodeck as a command-line utility

Introduction to using Holodeck as a command-line Utility

Usage

HolodeckGui [startwizard_option] [project_option] <options>

[startwizard_option] choices

/addnewintercepts	starts the 'Add Holodeck Intercepts' wizard on startup
/customtestproject	starts the 'Create a custom test project' wizard on startup

[project_option] choices

projectfile.hdp	loads the specified project file on startup
/launch:exefilename	launches the specified exe with default options*
/attach:<process_id>	attaches to the specified processID with default options*
/launchservice:servicename	launches the specified service with default options*
/attachservice:servicename	attaches to the specified running service with default options*

<options> choices*

/randomstress:[low,medium,high]	starts the random test generator with the specified intensity
/intelligentstress:[low,medium,high]	starts the intelligent test generator with the specified intensity
/codecoverage:[low,medium,high][,timeout]	starts the code coverage generator with the specified intensity and optional timeout in seconds
/record	starts recording the session on startup
/silent	starts Holodeck in silent mode, messages are logged to ErrorLog.txt

***Default Options:**

Pause on start	false
Process chaining	true
Inherit settings	true

Attach debugger	true
Logging options	same defaults as in New Project Wizard

Examples:

HolodeckGui myproject.hdp

HolodeckGui myproject.hdp /codecoverage:low

HolodeckGui myproject.hdp /codecoverage:high,15

HolodeckGui myproject.hdp /record

HolodeckGui /launch:c:\windows\notepad.exe

HolodeckGui /attach:2146 /randomstress:high

HolodeckGui /launchservice:"Indexing Service"

HolodeckGui /attachservice:"ClipBook" /record /slientÿÿÿÿ

Code Coverage testing from the command-line

Code coverage test generation attempts to hit as many error paths as possible. It reruns the application multiple times, each time setting different combinations of Limits, Faults, and Tests. Each time the Application Under Test terminates, by way of the script terminating the Application gracefully or Holodeck finding a crashing bug, Holodeck will relaunch the Application Under Test and start the next test, until there are none left. Holodeck assumes that each testing script will close the application under test at the end of the test; this allows Holodeck to know when to start the next test.

To turn on Code Coverage testing from the command-line use the /codecoversion switch followed by colon (:) high, medium, or low.

Example: HolodeckGui.exe projectName /codecoversion:high

- 1) **High**
 - a) Allow network limits
 - b) Allow disk limits
 - c) Allow memory limits
 - d) Allow all faults
 - e) Allow all tests
- 2) **Medium**
 - a) Allow network limits
 - b) Allow disk limits
 - c) Allow memory limits
 - d) Allow all faults
 - e) Allow resource based tests
 - f) Disallow all other tests
- 3) **Low**
 - a) Allow network limits
 - b) Allow disk limits
 - c) Allow memory limits
 - d) Disallow all faults
 - e) Disallow all tests

Initial Recording

The first thing Holodeck does when running Code Coverage Testing is record what the Application Under Test is doing to create an intelligent test plan. Holodeck records resource usage and unique API calls until the Application Under Test is terminated. It then uses the information found in the recording phase to build the next tests.

Exact order of execution:

- 1) Set each fault one at a time
 - a) For disk or memory
 - i) Set a limit for max used – 1 byte
 - ii) iSet a limit for average used – 1 byte
 - iii) iiSet a limit to minimum possible (0 bytes)

- b) For Network
- i) Set a limit to 50%
- ii) iSet a limit to 0%
- 2) Set a test for each unique API call seen in the recording phase.

Stress testing from the command-line

Using Holodeck's Stress Test Generation to test your application is a great way to ensure your software will continue to perform well even in stressful situations. This is useful for server side applications or applications that must perform well in multi-user environments. Holodeck will perform a random or intelligent test for a short period of time, if the test crashes the application under test Holodeck will restart the application and continue testing with the next test.

Random Tests – Randomly set faults, limits, and tests will be set at a given interval, at a given test duration, to flush out stability bugs.

To turn on Random Stress testing from the command-line use the /randomstress switch followed by colon (:) high, medium, or low.

Example: HolodeckGui.exe projectName /randomstress:high

- 1) **High** – test the Application Under Test with a very hostile environment
 - a) Every 15 seconds a random test will occur
 - b) Each Test will last two seconds
 - c) Holodeck will set five tests each time
 - d) Allow network limits from 0% to 25% of throughput
 - e) Allow disk limits down to the amount currently being used by the Application Under Test
 - f) Allow memory limits down to the amount currently being used by the Application Under Test
 - g) Allow all faults
- 2) **Medium** – test the Application Under Test with a semi hostile environment
 - a) Every 15 seconds a random test will occur
 - b) Each Test will last two seconds
 - c) Holodeck will set one test each time
 - d) Allow network limits from 25% to 75% of throughput
 - e) Allow disk limits down to one kilobyte over the amount currently being used by the Application Under Test
 - f) Allow memory limits down to one kilobyte over the amount currently being used by the Application Under Test
 - g) Allow all faults
- 3) **Low** – test the Application Under Test with some tests and faults
 - a) Every 30 seconds a random test will occur
 - b) Each test will last one second
 - c) Holodeck will set one test each time
 - d) Allow network limits from 75% to 100% of throughput
 - e) No disk limits will be set
 - f) No memory limits will be set
 - g) Allow all faults

Intelligent Tests – Create tests based on observation of API calls made by the application under test

To turn on Intelligent Stress testing from the command-line use the /intelligentstress switch followed by colon (:) high, medium, or low.

Example: HolodeckGui.exe projectName /intelligentstress:high

- 1) **High**
 - a) Each test will occur at 15 second intervals
 - b) Holodeck will set five tests at a time
- 2) **Medium**
 - a) Each test will occur at 30 second intervals
 - b) Holodeck will set two tests at a time
- 3) **Low**
 - a) Each test will occur at 60 second intervals

- b) Holodeck will set only one test at a time

Using Recorded Sessions from the command-line

Recording each session is a very good idea when running Holodeck from the command line, especially if it is run using the silent switch.

To enable recording of the session on the command line use the /record switch.

Example: HolodeckGui.exe projectName /record

This will create a recordedSession.xml file in the recorded session folder. This will allow you to replay the session to reproduce any errors Holodeck finds for a developer or bug triage team.

For more information on recording and replaying sessions please see the Introduction to recording and replaying section in the Holodeck in Depth Section.

Running Holodeck silently

You can run Holodeck minimized and without any UI notifications. You can run Holodeck more smoothly, especially in a test harness situation, if the UI notifications are suppressed. To test your application without any UI, and with as little memory as possible other Automated Testing options might be better such as the Custom Intercept Generator, HoloScript, and HEAT.

To run Holodeck silently use the /silent switch.

Example: HolodeckGui.exe projectName /silent

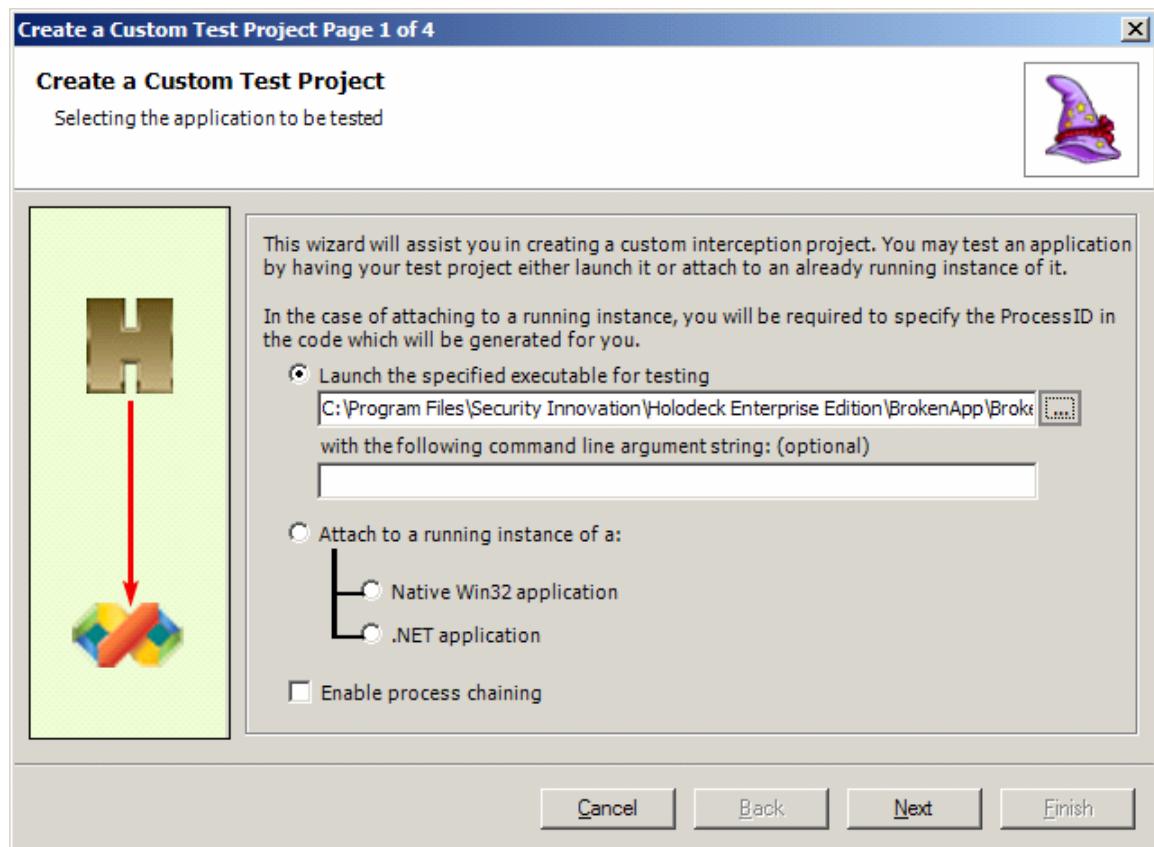
Create a Custom Test Project

Create a Custom Test Project Wizard

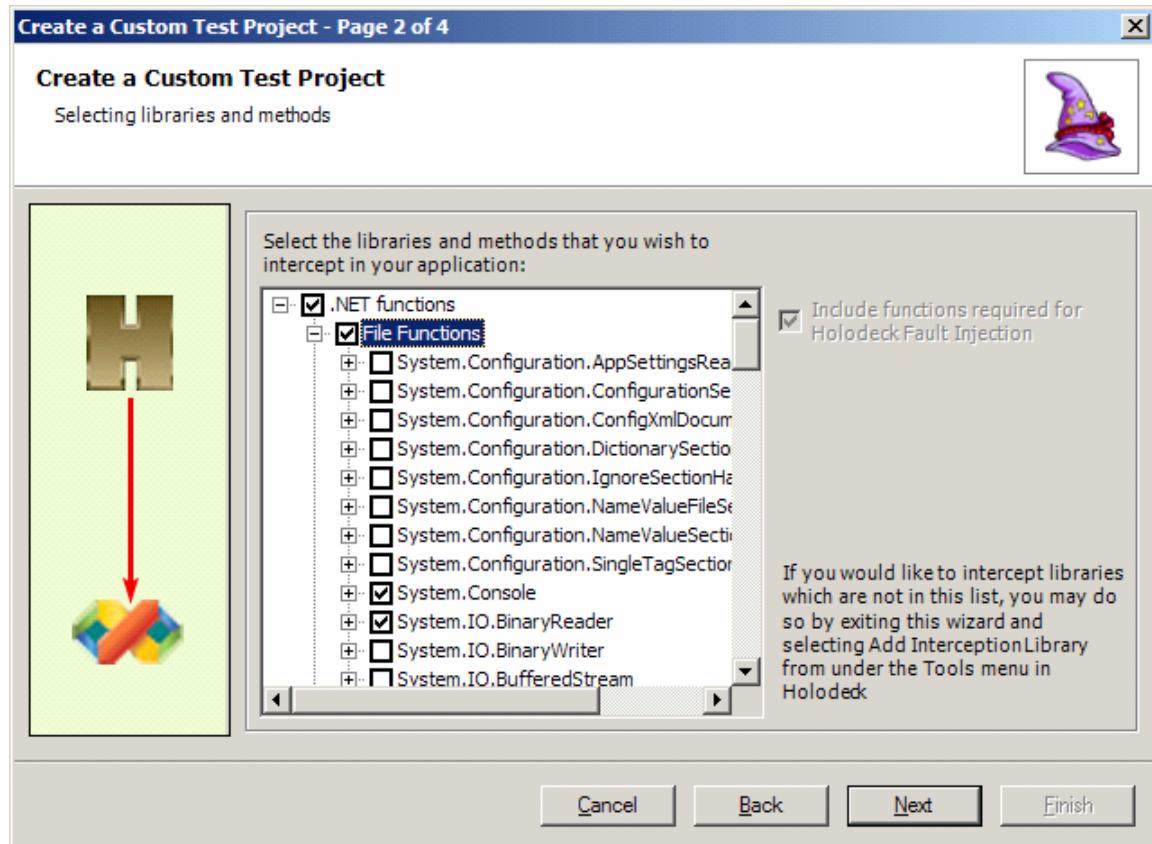
The custom test project wizard helps you to create a custom intercept project so that you can test an application without having to load all the features of Holodeck.

The Custom Test Project Wizard can be run from within Holodeck by clicking Tools > "Custom Test Project", or by itself, by selecting it in the Security Innovation Start Menu folder.

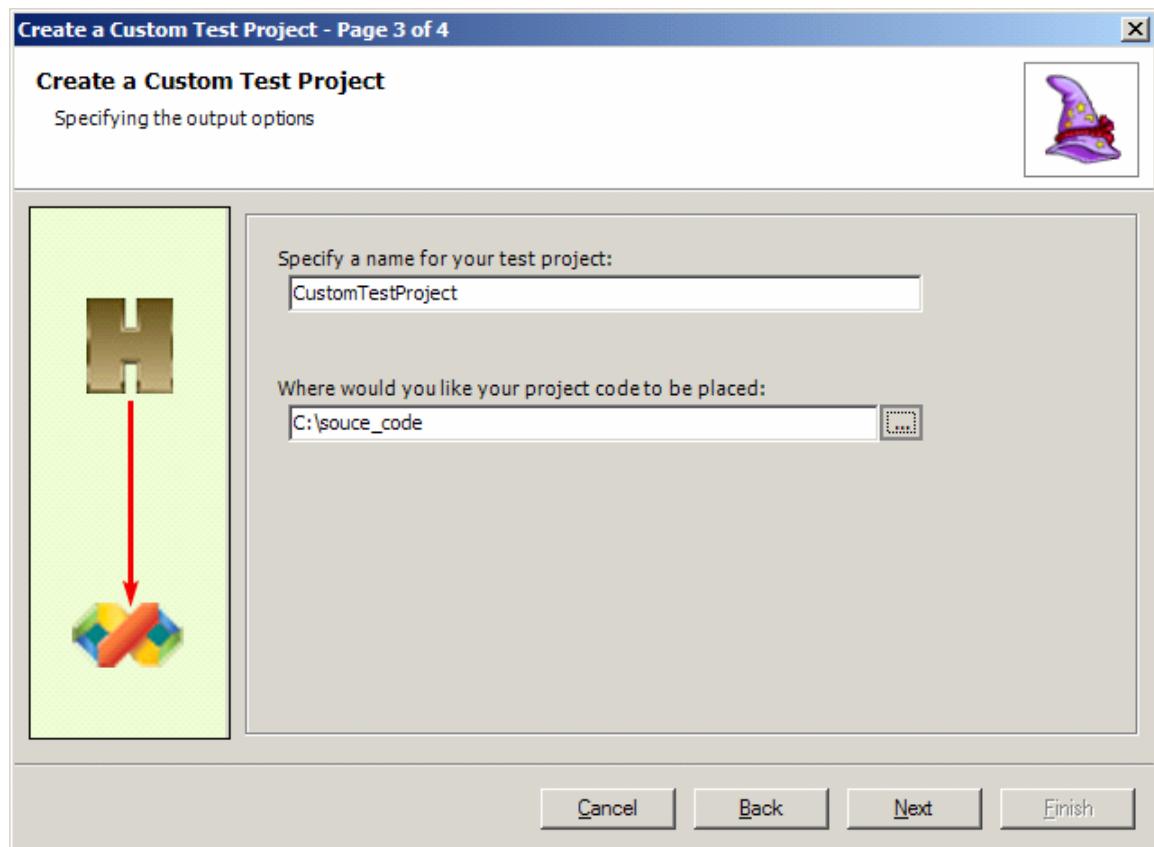
Create a Custom Test Project



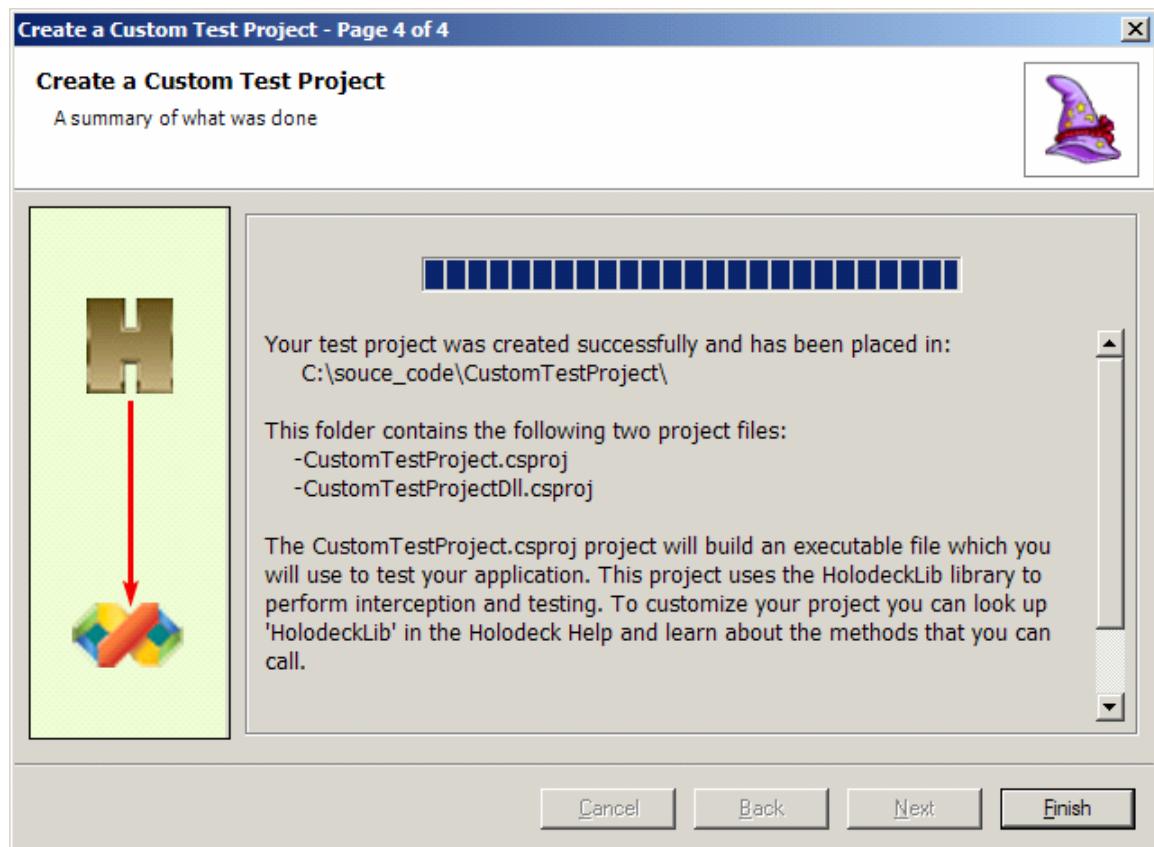
Selecting Libraries and Methods



Specifying the Output Options



Custom Test Project Summary



HolodeckLib

Using HolodeckLib

HolodeckLib is a .NET wrapper built around HoloScript to give you much of the power of HoloScript in an easy to learn friendly interface. HolodeckLib uses HoloScript, and many of the methods within this framework (**InterceptFunction**, **LaunchApplication**, **AttachApplication**, etc) are lightweight wrappers that directly call HoloScript routines with the same names.

HolodeckLib is the framework used by the Custom Test Project, you can use any of the HolodeckLib functions to help test your application. You can also create a project from scratch using these functions.

HolodeckLib Method List

Constructors and Destructors

TestApplication	The default constructor for a new Test Application
~TestApplication	The default destructor for a Test Application

Application Execution and Control

LaunchApplication	Launches an application that is not currently running
AttachApplication	Attaches to an application that is currently running
LaunchService	Launches a service that is not currently running
AttachService	Attaches to a service that is currently running
PauseApplication	Pauses the execution of the application
ResumeApplication	Resumes a paused application
TerminateApplication	Terminates execution of the application
GetApplicationProcessID	Returns the ProcessID of the currently running application
GetApplicationThreadID	Returns the ThreadID of the currently running thread
SetUseProcessChaining	Enable or disable Process Chaining in the current application
SetDotNetLibraryInitList	Set an external .NET library list

Log Operations

EnableFunctionLogging	Enable function logging of the function name in the specified category
DisableFunctionLogging	Disable function logging of the function name in the specified category

Limit Control

SetMaximumDiskUsage	Sets the maximum disk usage limit for the application
SetMaximumMemoryUsage	Sets the maximum memory usage limit for the application
SetNetworkUploadSpeed	Sets the maximum network upload speed limit for the application
SetNetworkDownloadSpeed	Sets the maximum network download speed limit for the application

Fault Control

InjectFault	Inject the specified fault to the target thread
RemoveFault	Remove the specified fault from the target thread

Interception Control

InterceptFunction	Intercepts a function and replaces with the replacement function
-------------------	--

HolodeckLib Constructors and Destructors

TestApplication();

- The default constructor for a new Test Application.

~TestApplication();

- The default destructor for a Test Application.

HolodeckLib Application Execution and Control

```
bool LaunchApplication(String * applicationName, String * cmdLine, bool startPaused);
```

- **Description** - Launch an application that is not currently running
 - **applicationName** – the full path to the application to launch
 - **cmdLine** – any command line arguments the application requires to start properly
 - **startPaused** – Start the application in a paused state
- **Return Value** - Returns true if all operations are successful, false otherwise

```
bool AttachApplication(unsigned long processID, bool pauseOnAttach);
```

- **Description** - Attach to an application that is currently running
 - **processID** – the ProcessID of which to attach.
 - **pauseOnAttach** – Pause the application as soon as the Test Application has attached
- **Return Value** - Returns true if all operations are successful, false otherwise

```
bool LaunchService(String *serviceName, bool startPaused);
```

- **Description** - Launch a service that is not currently running
 - **serviceName** – the name of the service to launch
 - **startPaused** – Start the service in a paused state.
- **Return Value** - Returns true if all operations are successful, false otherwise

```
bool AttachService(String *serviceName, bool startPaused);
```

- **Description** - Attach to a service that is currently running
 - **serviceName** – the name of the service to launch
 - **startPaused** – Pause the service as soon as the Test Application has attached
- **Return Value** - Returns true if all operations are successful, false otherwise

```
bool PauseApplication();
```

- **Description** - Pause the application

- **Return Value** - Returns true if all operations are successful, false otherwise

```
bool ResumeApplication();
```

- **Description** - Resume the application from a paused state
- **Return Value** - Returns true if all operations are successful, false otherwise

```
bool TerminateApplication();
```

- **Description** - Terminate Execution of the application
- **Return Value** - Returns true if all operations are successful, false otherwise

```
bool GetApplicationProcessID(UInt32 &processID);
```

- **Description** - Retrieves the process ID for the current application
 - **processID** – the process ID of the current application
- **Return Value** - Returns true if all operations are successful, false otherwise

```
bool GetApplicationThreadID(UInt32 &threadID);
```

- **Description** - Retrieve the test application's ThreadID
 - **threadID** – the Thread ID of the current thread.
- **Return Value** - Returns true if all operations are successful, false otherwise

```
bool SetUseProcessChaining(bool useChaining);
```

- **Description** - Set HolodeckLib to use process chaining
 - **useChaining** – Use process chaining
- **Return Value** - Returns true if all operations are successful, false otherwise

```
void SetDotNetLibraryInitList(ArrayList * externalLibsList);
```

- **Description** - Passes the specified list of .NET dll names to the Kernel32 replacement library so that at load time the intercepted versions of the dlls are used rather than the originals.
 - **externalLibsList** – an arraylist of .NET dll names that will be intercepted.
- **Return Value** - Returns true if all operations are successful, false otherwise

HolodeckLib Log Operations

- ```
bool EnableFunctionLogging(String *category, String *functionName);
```
- **Description** - Enable function logging of the function name in the specified category
    - **category** – the Holodeck Log Category of the function
    - **functionName** – the name of the function to enable logging on.
  - **Return Value** - Returns true if all operations are successful, false otherwise

- ```
bool DisableFunctionLogging(String *category, String *functionName);
```
- **Description** - Disable function logging of the function name in the specified category
 - **category** – the Holodeck Log Category of the function
 - **functionName** – the name of the function to enable logging on.
 - **Return Value** - Returns true if all operations are successful, false otherwise

HolodeckLib Limit Control

- ```
bool SetMaximumDiskUsage(UInt64 maximumDiskUsage, bool isEnabled);
```
- **Description** - Sets the maximum disk usage limit for the application
    - **maximumDiskUsage** – the maximum disk usage allowed to the application in bytes
    - **isEnabled** – Enable or disable the limit
  - **Return Value** - Returns true if all operations are successful, false otherwise

- ```
bool SetMaximumMemoryUsage(UInt64 maximumMemoryUsage, bool isEnabled);
```
- **Description** - Sets the maximum memory usage limit for the application
 - **maximumMemoryUsage** – the maximum memory usage allowed to the application in bytes
 - **isEnabled** – Enable or disable the limit
 - **Return Value** - Returns true if all operations are successful, false otherwise

- ```
bool SetNetworkUploadSpeed(UInt64 uploadSpeed, bool isEnabled);
```
- **Description** - Sets the maximum upload limit for the application
    - **uploadSpeed** – the maximum upload speed allowed to the application in bytes/second
    - **isEnabled** – Enable or disable the limit
  - **Return Value** - Returns true if all operations are successful, false otherwise

- ```
bool SetNetworkDownloadSpeed(UInt64 downloadSpeed, bool isEnabled);
```
- **Description** - Sets the maximum download limit for the application
 - **downloadSpeed** – the maximum download speed allowed to the application in bytes/second
 - **isEnabled** – Enable or disable the limit
 - **Return Value** - Returns true if all operations are successful, false otherwise

HolodeckLib Fault Control

- ```
bool InjectFault(Array *targetThreadID, int faultID);
```
- **Description** - Injects the specified fault to the target thread.
    - **targetThreadID** – the thread ID of the thread for which to set the fault
    - **faultID** – the FaultID of the fault you wish to set
  - **Return Value** - Returns true if all operations are successful, false otherwise

- ```
bool RemoveFault(Array *targetThreadID, int faultID);
```
- **Description** - Removes a set fault from the target thread
 - **targetThreadID** – the thread ID of the fault to remove
 - **faulted** – the faulted of the fault you wish to remove
 - **Return Value** - Returns true if all operations are successful, false otherwise

HolodeckLib Interception Control

```
bool InterceptFunction (String * functionName, String * replacementFunctionName, String *  
replacementDllName);
```

- **Description** - Intercepts a function and replaces with the replacement function
 - **functionName** – the name of the function you are replacing
 - **replacementFunctionName** – the name of the replacement function
 - **replacementDllName** – the name of the dll in which the replacement function resides.
- **Return Value** - Returns true if all operations are successful, false otherwise

Using HoloScript

Using HoloScript

HoloScript is a tool built on top of the HeatAPI that can be used to create new Holodeck interception frameworks. You may already be familiar with the **HolodeckLib::TestApplication** framework that is generated by the Holodeck **Custom Test Project**.

One of the benefits of using HoloScript to implement your advanced interceptions is that it is highly portable. Since HoloScript is provided as a .NET interface, it can be instantiated from any managed programming language, giving the developer a high degree of flexibility.

Note: Since HoloScript implements a number of Holodeck specific functions such as logging intercepted API calls, it may not be suitable for some low level applications that are not intended to be integrated into Holodeck (these applications could better utilize the HeatAPI directly).

Holoscript.dll is installed into the default install folder (by default C:\Program Files\Security Innovation\Holodeck Enterprise Edition)

HoloScript Method List

Constructors and Destructors

HoloScriptApplication	This default Constructor creates a new instance of a HoloScriptApplication containing a CHeatApp
~HoloScriptApplication	This default Destructor releases any base resources allocated by the HoloScriptApplication

Application and Service Launch

LaunchApplication	Launches the specified application, passing it the specified command line parameters and begins interception.
LaunchService	Launches a service that is not currently running.
AttachApplication	Attaches HoloScript to the process to an already running application.
AttachService	Attaches to a service that is currently running.

Process Chaining

get_InheritSettings	Gets a Boolean value if the child process should inherit the settings of the parent.
get_IsChainedProcess	True if the process was spawned as part of process chaining
get_ProcessChaining	Can be used to determine if any child processes of the current Application Under Test will monitored
set_InheritSettings	Sets whether or not the child process should inherit the settings of the parent.
set_IsChainedProcess	Sets the value to be a Chained Process.
set_ProcessChaining	Can be used to specify that any child processes of the current Application Under Test should be monitored
SetUseProcessChainin g	Enables (if useChaining is true) or Disables (if useChaining is false) monitoring of any child processes of the current Application Under Test

Debugging Functions

get_AttachDebugger	Returns whether or not Holodeck will be used as a debugger.
get_LogFirstChance	Gets whether or not Holodeck will log first chance exceptions.
get_ProcessSymbols	Gets the symbol list for the current process
set_AttachDebugger	Sets if Holodeck should be used as a debugger.
set_LogFirstChance	Sets Holodeck to log all first chance exceptions.

Application and Service Control

AddExternalDotNetLibrary	Sets a .NET library to be intercepted to the Kernel32 so at load time the intercepted version of the dll is used instead of the original.
get_ApplicationName	Can be used to retrieve the name of the current Application Under Test (for example, MyApp.exe)
get_ApplicationPath	Can be used to retrieve the full path name of the current Application Under Test (for example, C:\MyHScriptProject\MyApp.exe)
get_ApplicationPause	Can be used to detect if the interception framework is currently intercepting calls from the Application Under Test
get_CommandLineParams	Can be used to retrieve the command line parameters that were passed to the current Application Under Test
get_IsPaused	Can be used to detect if the interception framework has been properly initialized and is currently intercepting calls from the Application Under Test
get_IsRunning	Returns if the current application is running.
get_IsService	Gets a Boolean value if the current process is a service.
get_PauseOnStart	Can be used to detect if the Application Under Test has been requested to pause upon launch
get_ProjectFolder	Returns the folder the project is contained in.
get_ProjectPath	Returns the path of the current project

get_ProjectTitle	Returns the Title name of the current project.
get_ServiceName	Returns the name of the current service.
get_ShouldRestartWithProject	Gets a Boolean value if the process should restart with the project.
get_TerminateOnClose	Gets whether the application will be terminated when the project closes.
GetApplicationProcessID	Gets the process ID of the application currently being monitored and stores it in processID
GetApplicationThreadID	Gets the thread ID of the application currently being monitored and stores it in threaded
GetApplicationThreadList	Stores the thread identifiers of all the application's threads in the array pointed to by threadID and returns the number of threads
InterceptFunction	Can be used to specify that a call to functionName from the Application Under Test that should be intercepted
IsSystemProcess	Checks to see if the specified process is a system process or not.
IsSystemService	Checks to see if the specified service is a system service or not.
PauseApplication	Pauses interception for the current application
ResumeApplication	Resumes interception for the current application
set_ApplicationName	Can be used to set the full path name for the current Application Under Test (for example, C:\MyHScriptProject\MyApp.exe)
set_isService	Sets whether or not the process is a service.
set_PauseOnStart	Sets the application to be paused on launch.
set_ProjectPath	Sets the path where the current project file will be saved.
set_ServiceName	Sets the service name to the value passed in.
set_ShouldRestartWithProject	Sets whether the process should restart with the project.
set_TerminateOnClose	Sets the application to be terminated when the project closes.
setAppPauseState	Pauses or unpauses the current application.
TerminateApplication	Terminates the current application

Logging Control

DeleteSavedLogFile	Deletes the specified saved log file.
DisableFunctionLogging	Disables logging for the function belonging to category specified by functionName
EnableFunctionLogging	Enables logging for the function belonging to category specified by functionName
ExportToDiffFile	Writes the current log file to the specified file in plain text format
get_FilteredFunctionList	Can be used to retrieve a list of InterceptedFunction pointers that specify functions that are not currently being intercepted and logged by the interception framework
get_LogEntryCount	Can be used to retrieve the number of entries in the current application log file
get_SavedLog	Gets a Boolean value if the current log is a saved log.
GetFilteredFunctionList	Returns a pointer to an Array of InterceptedFunction strings that specify functions that are not currently being logged by the interception framework.
GetLogEntryAt	Returns a pointer to an instance of LogItem containing the log entry located at the specified integer index
GetLogEntryAtFilePath	Returns a pointer to an instance of LogItem containing the log entry located at the specified absolute position in the log file
GetLogFileName	Simply returns a pointer to a String that contains the name of the interception log file previously opened by OpenLogFile
OpenLogFile	Opens the specified file as the interception log file
SearchInLogFile	Returns the integer index of the first log entry containing the specified string (starting from the specified starting record index)
set_FilteredFunctionList	Can be used to specify the functions that are not currently being intercepted and logged by the interception framework as a list of InterceptedFunction pointers

Tests Control

CreateScheduledTest	Creates a test to be performed upon the next call
---------------------	---

	to functionName from the Application Under Test
DeleteScheduledTest	Removes the test identified by testID

Limits Control

GetMonitorInformation	Gets information about the current disk and memory usage for the application currently being monitored and stores them in the appropriate parameters
GetNetworkDownloadBandwidth	Gets information about the system's maximum download network throughput (in Bytes/sec) that will be allocated to the Application Under Test
GetNetworkUploadBandwidth	Gets information about the system's maximum upload network throughput (in Bytes/sec) that will be allocated to the Application Under Test
GetRealSpaceInformation	Gets information about the system's currently available disk and memory space
SetMaximumDiskUsage	Sets the maximum amount of disk space that will be allocated to the Application Under Test
SetMaximumMemoryUsage	Sets the maximum amount of memory that will be allocated to the Application Under Test
SetNetworkDownloadBandwidth	Sets the system's maximum network download throughput (in bytes/sec) that will be allocated to the Application Under Test
SetNetworkUploadBandwidth	Sets the system's maximum network upload throughput (in bytes/sec) that will be allocated to the Application Under Test

Fault Control

CreateFileCorruptionFault	Creates a file corruption fault on the specified file.
CreateNetworkCorruption	Creates a network corruption fault where send specifies that outgoing packets should be corrupted and recv specifies that incoming packets should be corrupted
DeleteFileCorruptionFault	Removes a file corruption fault from the specified file.
DeleteNetworkCorruption	Removes the network corruption fault identified by testID (value set by previous call to CreateNetworkCorruptionFault)

InjectFault	Injects the fault specified by faultID into the thread(s) specified in targetThreadID
InjectResourceFault	Injects a resource fault to the specified thread.
RegenerateFileCorruption	Regenerates file corruption for the specified file.
RemoveFault	Removes the fault specified by faultID from the thread(s) specified in targetThreadID
RemoveResourceFault	Removes a resource fault from the specified thread.
SetLastFileCorruption	

HoloScript Functions

Constructors and Destructors

HoloScriptApplication()

This default Constructor creates a new instance of a HoloScriptApplication containing a CHeatApp

~HoloScriptApplication()

This default Destructor releases any base resources allocated by the HoloScriptApplication

Application and Service Launch

```
bool LaunchApplication( String *ApplicationToLaunch,
                      String *CommandLineParams,
                      bool PauseOnStart,
                      ArrayList *FilteredFunctionList,
                      String * exporterLog,
                      bool UseExternalInterceptionList)
```

- **Description** - Launches the specified application, passing it the specified command line parameters and begins interception
 - **ApplicationToLaunch** – the full path to the application to launch (i.e. "C:\\programDirectory\\myProgram.exe")
 - **CommandLineParams** – any command line parameters your application needs to operate properly.
 - **PauseOnStart** – interception will pause when the application first starts up if true.
 - **FilteredFunctionList** - a list of pointers to Strings containing the names of functions that will not be logged by the interception framework
 - **exporterLog** – the current log file for the application – if this value is NULL, the default path will be used for the application's current log file
 - **UseExternalInterceptionList** – true if functions to be intercepted are to be specified by an external list (such as the one created by the **Custom Test Project**)
- **Return Value** - Returns true if all operations are successful, false otherwise

```
HoloScriptApplication hsapp = new HoloScriptApplication();
ArrayList list = new ArrayList();
list.Add("WriteFile");
hsapp.FilteredFunctionList = list;

hsapp.LaunchApplication("c:\\windows\\notepad.exe", null,
true, list, null, false);
hsapp.ResumeApplication();

System.Console.WriteLine("{0} filtered functions",
hsapp.FilteredFunctionList.Count);
```

Note: the above code will print 1, as you would expect. However if UseExternalExceptionList is true LaunchApplication will reset the list and the above code will print 485. Do not test this parameter to true if you are using a list

```
bool LaunchApplication( String *ApplicationToLaunch,
                      String *CommandLineParams,
                      bool PauseOnStart,
                      ArrayList *FilteredFunctionList)
```

- **Description** - Launches the specified application, passing it the specified command line parameters and begins interception The default path will be used for the application's current log file.

- **ApplicationToLaunch** – the full path to the application to launch (i.e. "C:\\programDirectory\\myProgram.exe")
 - **CommandLineParams** – any command line parameters your application needs to operate properly.
 - **PauseOnStart** – interception will pause when the application first starts up if true.
 - **FilteredFunctionList** - a list of pointers to Strings containing the names of functions that will not be logged by the interception framework
- **Return Value** - Returns true if all operations are successful, false otherwise

```
bool LaunchService(    String *serviceToLaunch,
                      bool pauseOnStart,
                      ArrayList *filteredFunctionList);
```

- **Description** - Launches the specified Service.
 - **serviceToLaunch** – the name of the service to launch.
 - **pauseOnStart** – if true the service will start paused.
 - **FilteredFunctionList** contains an ArrayList of pointers to Strings containing the names of functions that will not be logged by the interception framework
- **Return Value** - Returns true if all operations are successful, false otherwise

```
bool AttachApplication(  unsigned long processID,
                        bool pauseOnAttach,
                        ArrayList *filteredFunctionList,
                        String * exporterLog,
                        bool UseExternalInterceptionList,
                        bool suspendedApp,
                        bool injectOnlyServiceExeLibrary);
```

- **Description** - Attaches HoloScript to the process to an already running application.
 - **processID** – the ID of the process to attach to.
 - **pauseOnAttach** – pause the application when Holodeck attaches.
 - **filteredFunctionList** – an ArrayList of pointers to Strings containing the names of functions that will not be logged by the interception framework
 - **exporterLog** – the path for the log file to be saved.
 - **UseExternalInterceptionList** – Boolean value to use an external Interception list, or use the default interception libraries provided by Holodeck

- **suspendedApp** – true if the application is already suspended.
 - **injectOnlyServiceExeLibrary** –
- **Return Value** - Returns true if all operations are successful, false otherwise

```
bool AttachApplication(    unsigned long processID,
                           bool pauseOnAttach,
                           ArrayList *filteredFunctionList,
                           bool suspendedApp);
```

- **Description** - Attaches HoloScript to the process to an already running application.
 - **processID** – the processID of the application to attach to.
 - **pauseOnAttach** – Boolean to pause the application when the interception library is attached.
 - **filteredFunctionList** – an ArrayList of pointers to Strings containing the names of functions that will not be logged by the interception framework
 - **suspendedApp** – true if the application is already suspended.
- **Return Value** - Returns true if all operations are successful, false otherwise

```
bool AttachService(        String *serviceToAttachTo,
                           bool pauseOnAttach,
                           ArrayList *filteredFunctionList,
                           bool terminateOnExit);
```

- **Description** - Attaches to a service that is currently running
 - **serviceToAttachTo** – the name of the service to attach to.
 - **pauseOnAttach** – enable to pause the service when the interception library attaches.
 - **filteredFunctionList** – an ArrayList of pointers to Strings containing the names of functions that will not be logged by the interception framework
 - **terminateOnExit** – terminate the application when the interception framework exits.
- **Return Value** - Returns true if all operations are successful, false otherwise

Process Chaining

__property bool get_ApplicationAttachedTo()

- **Description** – return if the application was attached to.
- **Return Value** –
 - True if user has attached to a running application process.
 - False if the user has launched an application or service, or attached to a service.

__property bool get_InheritSettings()

- **Description** - Gets a Boolean value if the child process should inherit the settings of the parent.
- **Return Value** – True if the child process will inherit the settings of the parent.

__property bool get_IsChainedProcess()

- **Description** – Gets a Boolean value if the process is a child of a parent process from process chaining.
- **Return Value** - True if the process was spawned as part of process chaining.

__property bool get_ProcessChaining()

- **Description** - Can be used to determine if any child processes of the current Application Under Test will monitored
- **Return Value** – True if new processes will be automatically attached to by Holodeck

Example usage:

```
HoloScriptApplication *holoApp =  
new HoloScriptApplication();  
. . .  
if (holoApp->ProcessChaining) {  
// WATCHING CHILDREN  
} else {  
// NOT WATCHING CHILDREN  
}
```

__property void set_InheritSettings(bool value)\

- **Description** - Sets whether or not the child process should inherit the settings of the parent.
- **Return Value** - none

`__property void set_ProcessChaining(bool value)`

- **Description** - Can be used to specify that any child processes of the current Application Under Test should be monitored
- **Return Value** - none

Example usage:

```
HoloScriptApplication *holoApp =
new HoloScriptApplication();
.
.
.
holoApp->ProcessChaining = true;
```

`bool SetUseProcessChaining(bool useChaining)`

- **Description** - Enables or Disables monitoring of any child processes is enabled of the current Application Under Test
 - **useChaining** – if true monitoring of any child processes is enabled of the current Application Under Test
- **Return Value** - Returns true if operation is successful, false otherwise

Debugging Functions

`_property bool get_AttachDebugger()`

- **Description** - Returns whether or not Holodeck will be used as a debugger.
- **Return Value** – True if Holodeck will be used as a debugger.

`_property bool get_LogFirstChance()`

- **Description** - Gets whether or not Holodeck will log first chance exceptions.
- **Return Value** – True if Holodeck will log first chance exceptions

`_property Array* get_ProcessSymbols()`

- **Description** - Gets the symbol list for the current process
- **Return Value** – A pointer to an array of the symbols for the current process

`_property void set_AttachDebugger(bool value);`

- **Description** - Sets if Holodeck should be used as a debugger.
- **Return Value** - none

`_property void set_LogFirstChance(bool value);`

- **Description** - Sets Holodeck to log all first chance exceptions.
- **Return Value** - none

Application and Service Control

```
void AddExternalDotNetLibrary(String * externalLibrary);
```

- **Description** - Sets a .NET library to be intercepted to the Kernel32 so at load time the intercepted version of the dll is used instead of the original.
 - **externalLibrary** – the external library to use as a replacement library.
- **Return Value** - none

```
__property String *get_ApplicationName()
```

- **Description** - Can be used to retrieve the name of the current Application Under Test
- **Return Value** – a string of the application name, (for example, **MyApp.exe**)

Example usage:

```
HoloScriptApplication *holoApp =
    new HoloScriptApplication();

.

.

String *appName =
    holoApp->ApplicationName;
```

```
__property String *get_ApplicationPath()
```

- **Description** - Can be used to retrieve the full path name of the current Application Under Test
- **Return Value** – a string of the entire application path. (for example, **C:\MyHScriptProject\MyApp.exe**)

Example usage:

```
HoloScriptApplication *holoApp =
    new HoloScriptApplication();

.

.

String *pathName =
    holoApp->ApplicationName;
```

```
__property bool get_ApplicationPaused()
```

- **Description** - Can be used to detect if the interception framework is currently intercepting calls from the Application Under Test

- **Return Value** – true if the application is currently paused

Example usage:

```
HoloScriptApplication *holoApp =
    new HoloScriptApplication();
.

.

if (holoApp->ApplicationPaused) {
    // NOT INTERCEPTING
} else {
    // INTERCEPTING
}
```

| | **__property String *get_CommandLineParams()**

- **Description** - Can be used to retrieve the command line parameters that were passed to the current Application Under Test
- **Return Value** – returns a string of the command line parameters that were passed to the current Application.

Example usage:

```
HoloScriptApplication *holoApp =
    new HoloScriptApplication();
.

.

String *cmdLine =
    holoApp->CommandLineParams;
```

| | **__property bool get_IsPaused()**

- **Description** - Can be used to detect if the interception framework has been properly initialized and is currently intercepting calls from the Application Under Test
- **Return Value** – true if the application is currently paused

Example usage:

```
HoloScriptApplication *holoApp =
    new HoloScriptApplication();
.

.

if (holoApp->IsPaused) {
    // NO HEAT
} else {
    // HEAT ON
}
```

| | **__property bool get_IsRunning();**

- **Description** - Returns if the current application is running.
- **Return Value** – true if the application is currently running

__property bool get_IsService()

- **Description** - Gets a Boolean value if the current process is a service.
- **Return Value** – true if the application is a service

 __property bool get_PauseOnStart()

- **Description** - Can be used to detect if the Application Under Test has been requested to pause upon launch
- **Return Value** – true if the application will be paused on start

Example usage:

```
HoloScriptApplication *holoApp =  
    new HoloScriptApplication();  
  
    .  
    .  
    .  
if (holoApp->PauseOnStart) {  
    // WILL PAUSE  
} else {  
    // WON'T PAUSE  
}
```

 __property String* get_ProjectFolder()

- **Description** - Returns the folder the project is contained in.
- **Return Value** – returns the folder the application is contained in.

 __property String* get_ProjectPath()

- **Description** - Returns the path of the current project
- **Return Value** – the entire project path for the current project

 __property String* get_ProjectTitle()

- **Description** - Returns the Title name of the current project.
- **Return Value** – a string of the project path

 __property String *get_ServiceName()

- **Description** - Returns the name of the current service.
- **Return Value** – a string of the name of the current service.

 __property bool get_ShouldRestartWithProject()

- **Description** - Gets a Boolean value if the process should restart with the project.
 - **Return Value** – true if the current process will restart when the project is restarted

__property bool get_TerminateOnClose()

- **Description** - Gets whether the application will be terminated when the project closes.
 - **Return Value** – true if the application will be closed when the project terminates.

bool GetApplicationProcessID(UInt32 &processID)

- **Description** - Gets the process ID of the application currently being monitored and stores it in processID
 - **processID** – The process id for the current application
 - **Return Value** - Returns true if all operations are successful, false otherwise

bool GetApplicationThreadID(UInt32 &threadID)

- **Description** - Gets the thread ID of the application currently being monitored and stores it in `threadID`
 - **Return Value** - Returns true if all operations are successful, false otherwise

```
int GetApplicationThreadList(DWORD * threadID)
```

- **Description** - Stores the thread identifiers of all the application's threads in the array pointed to by **threadID**
 - **threadID** - all the application's threads in the array pointed to by **threadID**.
if NULL, the number of thread is returned
 - **Return Value** – the number of threads.

```
bool InterceptFunction(  String * functionName,
                        String * redirectedFunctionName,
                        String * redirectedDllName)
```

- **Description** - Can be used to redirect a specific function to an intercept function
 - **functionName** – the original function to be intercepted
 - **redirectedFunctionName** – the replacement function.
 - **redirectedDlLName** - the DLL file where the replacement function is located.
 - **Return Value** - Returns true if operation is successful, false otherwise

```
static bool IsSystemProcess(unsigned long processIDToCheck)
```

- **Description** - Checks to see if the specified process is a system process or not.

- **processIDToCheck** – the process ID to check.
- **Return Value** – true if the current process is a system process

| | static bool IsSystemService(String *serviceNameToCheck)

- **Description** - Checks to see if the specified service is a system service or not.
 - **serviceNameToCheck** – the service name to check
- **Return Value** – true if the current process is a system service.

| | bool PauseApplication()

- **Description** - Pauses interception for the current application
- **Return Value** - Returns true if all operations are successful, false otherwise

| | bool ResumeApplication()

- **Description** - Resumes interception for the current application
- **Return Value** - Returns true if all operations are successful, false otherwise

| | __property void set_ApplicationName(String *value)

- **Description** - Can be used to set the full path name for the current Application Under Test (for example, C:\MyHScriptProject\MyApp.exe)
- **Return Value** - none

Example usage:

```
HoloScriptApplication *holoApp =
    new HoloScriptApplication();
String *appToLaunch =
    new String(
        "C:\MyHScriptProject\MyApp.exe");
holoApp->ApplicationName = appToLaunch;
```

| | __property void set_IsService(bool value)

- **Description** - Sets whether or not the process is a service.
 - **value** – true if the process is a service
- **Return Value** - none

| | __property void set_PauseOnStart(bool value)

- **Description** - Sets the application to be paused on launch.
 - **Value** – true if the application should pause on start
- **Return Value** - none

```
    __property void set_ProjectPath(String* path)
```

- **Description** - Sets the path where the current project file will be saved.
 - **Path** – a string of where the current project file will be saved ("C:\myProject")
- **Return Value** - none

```
    __property void set_ServiceName(String *value)
```

- **Description** - Sets the service name to the value passed in.
 - **Value** – the name of the service
- **Return Value** - none

```
    __property void set_ShouldRestartWithProject(bool value)
```

- **Description** - Sets whether the process should restart with the project.
 - **Value** – true if the application should restart with the project
- **Return Value** - none

```
    __property void set_TerminateOnClose(bool value)
```

- **Description** - Sets the application to be terminated when the project closes.
 - **Value** – true if the application will terminate on close
- **Return Value** - none

```
    void SetAppPauseState(bool value)
```

- **Description** - Pauses or unpauses the current application
 - **Value** – true to pause the application, false to unpause the application
- **Return Value** - none

```
    bool TerminateApplication()
```

- **Description** - Terminates the current application
- **Return Value** - Returns true if all operations are successful, false otherwise

Logging Control

- ```
bool DeleteSavedLogFile(String *logFileName)
```
- **Description** - Deletes the specified saved log file
    - **logFileName** – the full path and name to a logfile.
  - **Return Value** - Returns true if operation is successful, false otherwise
  - **Note:** be sure to verify the correct delete path, this function will delete any file.
- 
- ```
bool DisableFunctionLogging(String *category, String *functionName)
```
- **Description** - Disables logging for the function belonging to **category** specified by **functionName**
 - **category** – the category of the function name, for more information see the API Log Categories section.
 - **functionName** – the actual function name to disable logging for.
 - **Return Value** - Returns true if operation is successful, false otherwise
-
- ```
bool EnableFunctionLogging(String *category, String *functionName)
```
- **Description** - Enables logging for the function belonging to **category** specified by **functionName**
    - **category** – the category of the function name, for more information see the API Log Categories section.
    - **functionName** – the actual function name to enable logging for.
  - **Return Value** - Returns true if operation is successful, false otherwise
- 
- ```
void ExportToDiffFile(String *diffFileName)
```
- **Description** - Writes the current log file to the specified file in plain text format
 - **diffFileName** – the file name and path to export the current log to.
 - **Return Value** - none
-
- ```
_property ArrayList *get_FilteredFunctionList()
```
- **Description** - Can be used to retrieve a list of **InterceptedFunction** pointers that specify functions that are not currently being logged by the interception framework
  - **Return Value** – an array list of all the functions that are not currently being logged

*Example usage:*

```
HoloScriptApplication *holoApp =
 new HoloScriptApplication();
.
.
.
ArrayList *filteredFuncs =
 holoApp->FilteredFunctionList;
```

`__property unsigned long get_LogEntryCount()`

- **Description** - Can be used to retrieve the number of entries in the current application log file
- **Return Value** – the number of log entries in the current log file

*Example usage:*

```
HoloScriptApplication *holoApp =
 new HoloScriptApplication();
.
.
.
unsigned long numEntries =
 holoApp->LogEntryCount;
```

`__property bool get_SavedLog()`

- **Description** - Gets a Boolean value if the current log is a saved log
- **Return Value** – true if the current log is saved

`LogItem *GetLogItemAt(int index, unsigned long &filePosition)`

- **Description** – Gets a log entry at the specified file position and index.
  - **filePosition** – the absolute position in the log file where the **LogItem** is located
  - **index** – the index of the log item to return.
- **Return Value** - Returns a pointer to an instance of **LogItem** containing the log entry located at the specified integer index

`LogItem *GetLogEntryAtFilePosition(unsigned long filePos)`

- **Description** – gets a lot item at the specified file position
  - **filePos** – the absolute position of the file.
- **Return Value** - Returns a pointer to an instance of **LogItem** containing the log entry located at the specified absolute position in the log file

`String *GetLogFileName()`

- **Description** - Simply returns a pointer to a String that contains the name of the interception log file previously opened by **OpenLogFile**
- **Return Value** - a pointer to a String that contains the name of the interception log file previously opened by **OpenLogFile**

`void OpenLogFile(String *newLogFileName)`

- **Description** - Opens the specified file as the interception log file
  - **newLogFileName** – the complete filename and path to the log file you wish to open
- **Return Value** - none

```
int SearchInLogFile(String *searchString, int startRecord)
 • Description - Searches the current log file for the search string.
 ○ searchString – the text to search for in the current log.
 ○ startRecord – the first record to search.
 • Return Value - the integer index of the first log entry containing the specified string
 (starting from the specified starting record index)
 ○ Returns -1 if no matches are found before the end of the log file
```

```
__property void set_FilteredFunctionList(ArrayList *value)
 • Description - Can be used to specify the functions that are not currently being
 logged by the interception framework as a list of InterceptedFunction pointers
 ○ value – an array list of functions that should not be logged.
 • Return Value - none
```

*Example usage:*

```
HoloScriptApplication *holoApp =
 new HoloScriptApplication();
ArrayList *filteredFuncs = new ArrayList();
.
.
.
holoApp->FilteredFunctionList =
 filteredFuncs;
```

## Tests Control

```
bool CreateScheduledTest(Array *targetThreadID,
 String *category,
 String *functionName,
 String *returnValue, String *errorCode,
 ArrayList *matchParams,
 ArrayList *changeParams,
 String **testID)
```

- **Description** - Creates a test to be performed upon the next call to **functionName** from the Application Under Test thread(s) in **targetThreadID** (a pointer to an Array of UInt32 pointers) if the specified conditions are met
  - **category** is the category of the function used by the settings manager
  - **returnValue** specifies the String to return if the test returns a positive result
  - **errorCode** specifies the error code to set if the test returns a positive result
  - **matchParams** is a pointer to a list of **InterceptedFunctionParameter** pointers that specify the IN parameters to match for the test to execute
  - **changeParams** is a pointer to a list of **InterceptedFunctionParameter** pointers that specify the OUT parameters that will change when the test executes
  - **testID** - will contain a pointer to a unique identifier for this test, following this call
- **Return Value** - Returns true if all operations are successful, false otherwise

```
bool DeleteScheduledTest(Array *targetThreadID, String *testID)
```

- **Description** - Removes the test identified by **testID** (value set by previous call to **CreateScheduledTest**) from the thread(s) specified in **targetThreadID** (a pointer to an Array of UInt32 pointers)
  - **targetThreadID** – the thread to delete the scheduled test from, if null this function will delete the specified test from all threads
  - **testID** – the test ID of the test you would like to remove
- **Return Value** - Returns true if all operations are successful, false otherwise

## Limits Control

```
bool GetMonitorInformation(UInt64 &maximumDiskUsage,
 UInt64 ¤tDiskUsage,
 UInt64 &maximumMemoryUsage,
 UInt64 ¤tMemoryUsage)
```

- **Description** - Gets information about the current disk and memory usage for the application currently being monitored and stores them in the appropriate parameters
  - **maximumDiskUsage** stores the maximum amount of disk space that will be allocated to the Application Under Test
  - **currentDiskUsage** stores the current amount of disk space being used by the Application Under Test
  - **maximumMemoryUsage** stores the maximum amount of memory that will be allocated to the Application Under Test
  - **currentMemoryUsage** stores the current amount of memory being used by the Application Under Test
- **Return Value** - Returns true if operations are successful, false otherwise

```
|bool GetNetworkDownloadBandwidth(UInt64 &downloadBandwidth);
```

- **Description** - Gets information about the system's maximum download network throughput (in Bytes/sec) that will be allocated to the Application Under Test
  - **downloadBandwidth** – the current maximum download bandwidth
- **Return Value** - Returns true if operations are successful, false otherwise

```
|bool GetNetworkUploadBandwidth(UInt64 &uploadBandwidth);
```

- **Description** - Gets information about the system's maximum upload network throughput (in Bytes/sec) that will be allocated to the Application Under Test
  - **uploadBandwidth** – the current maximum upload bandwidth
- **Return Value** - Returns true if operations are successful, false otherwise

```
|bool GetRealSpaceInformation(UInt64 &maximumDiskUsage,
 UInt64 &maximumMemoryUsage)
```

- **Description** - Gets information about the system's currently available disk and memory space – this call ignores any limits that have been set within HoloScript
  - **maximumDiskUsage** stores the maximum amount of disk space available to the Application Under Test
  - **maximumMemoryUsage** stores the maximum amount of memory available to the Application Under Test
- **Return Value** - Returns true if operations are successful, false otherwise

```
bool SetMaximumDiskUsage(UInt64 maximumDiskUsage, bool isEnabled)
```

- **Description** - Sets the maximum amount of disk space that will be allocated to the Application Under Test
  - **maximumDiskUsage** – the maximum about of disk space available to the application in bytes
  - **isEnabled** – tell the settings manager to turn disk limits on (true) or off (false)
- **Return Value** - Returns true if operation is successful, false otherwise

- ```
bool SetMaximumMemoryUsage(UInt64 maximumMemoryUsage, bool isEnabled)
• Description - Sets the maximum amount of memory that will be allocated to the Application Under Test
    ○ maximumMemoryUsage – the maximum about of memory space available to the application in bytes
    ○ isEnabled – tell the settings manager to turn disk limits on (true) or off (false)
• Return Value - Returns true if operation is successful, false otherwise

bool SetNetworkDownloadBandwidth(UInt64 downloadBandwidth , bool IsDownloadControlEnabled);
• Description - Sets the system's maximum network download throughput (in bytes/sec) that will be allocated to the Application Under Test
    ○ downloadBandwidth – the amount of bandwidth available to the application in bytes/second
    ○ IsDownloadControlEnabled – tell the settings manager to turn network limits on (true) or off (false)
• Return Value - Returns true if operation is successful, false otherwise

bool SetNetworkUploadBandwidth(UInt64 uploadBandwidth , bool IsUploadControlEnabled);
• Description - Sets the system's maximum network upload throughput (in bytes/sec) that will be allocated to the Application Under Test
    ○ uploadBandwidth – the amount of bandwidth available to the application in bytes/second
    ○ IsUploadControlEnabled – used to tell the settings manager to turn network limits on (true) or off (false)
• Return Value - Returns true if operation is successful, false otherwise
```

Fault Control

```
bool CreateFileCorruptionFault(    String* fileName,  
                                  bool alwaysRegen,  
                                  String* searchRegExpr,  
                                  String* replacementStr,  
                                  String **testId);
```

- **Description** - Creates a file corruption fault on the specified file.
 - **filename** – the file to corrupt, complete path.
 - **alwaysRegen** – Regenerate the file every time the application accesses it
 - **searchRegExpression** – The regular expression to search for
 - **repalcementStr** – The string to replace the regular expression with.
 - **testId** – The identifier of the test.
- **Return Value** - Returns true if all operations completed, false otherwise.

```
bool CreateNetworkCorruptionFault(      bool send,  
                                         bool recv,  
                                         Array* portList,  
                                         String* searchRegExpr,  
                                         String* replacementStr,  
                                         String **testId);
```

- **Description** - Creates a network corruption fault where send specifies that outgoing packets should be corrupted and recv specifies that incoming packets should be corrupted
 - **send** – Corrupt data being sent over the network.
 - **recv** – Corrupt data being received over the network.
 - **portList** – a list of each port on which corruption will occur.
 - **searchRegExpression** – The regular expression to search for
 - **repalcementStr** – The string to replace the regular expression with.

- **testId** – a pointer to a unique identifier for the test.
- **Return Value** - Returns true if all operations completed, false otherwise.

```
bool DeleteFileCorruptionFault(String *testId);
• Description - Removes a file corruption fault from the specified file.
○ testId – The test id of the file corruption fault you would like to remove.
• Return Value - Returns true if all operations completed, false otherwise.
```

```
bool DeleteNetworkCorruptionFault(String *testId);
• Description - Removes a network corruption fault.
○ testId – The test id of the network corruption fault you would like to remove.
• Return Value - Returns true if all operations completed, false otherwise.
```

```
bool InjectFault(Array *targetThreadID, FaultType faultID)
• Description - Injects the fault specified by faultID into the thread(s) specified in targetThreadID (a pointer to an Array of UInt32 pointers) – if targetThreadID is NULL, the fault is injected into all thread(s)
○ targetThreadID – the thread to inject the fault to
○ faultID – the fault to inject
• Return Value - Returns true if operation(s) are successful, false otherwise
```

```
ArrayList *InjectResourceFault(    Array *targetThreadID,
                                    int faultID,
                                    String *resourceName);
• Description - Injects a resource fault to the specified thread.
○ targetThreadID – The thread(s) you would like to set the resource fault on. If NULL the resource fault will be applied to all threads.
○ faultID – the fault id of the fault to inject.
○ resourceName – the resource name for the fault (ie. kernel.dll)
• Return Value – an array list of the threads that the resource fault was inject to.
```

```
bool RegenerateFileCorruptionFault(String *fileName);
```

- **Description** - Forces regeneration of file corruption for the specified file.
 - **filename** – the file to corrupt
- **Return Value** - Returns true if all operations completed, false otherwise

```
bool RemoveFault(Array *targetThreadID, FaultType faultID)
• Description - Removes the fault specified by faultID from the thread(s) specified in
  ◦ targetThreadID - a pointer to an Array of UInt32 pointers
    • if targetThreadID is NULL, the fault is removed from all
      thread(s)
• Return Value - Returns true if operation(s) are successful, false otherwise
```

```
bool RemoveResourceFault(Array *targetThreadID, ArrayList *testIDs);
• Description - Removes a resource fault from the specified thread.
  ◦ targetThreadID – the threadID of the thread you wish to remove the fault,
    a NULL value will remove the fault from all threads.
  ◦ testID – the tests to delete.
• Return Value - Returns true if operation(s) are successful, false otherwise
```

```
bool SetLastFileCorruption(String *fileName, String *corruptedFile, String
*changesXmlFile);
• Description - Sets the value of the last corrupted file; so that file corruption code
  may repro file corruption if required.
  ◦ filename – original file name
  ◦ corruptedFile – the corrupted file
  ◦ changesXmlFile – the XML file outlining the changes made to the
    corrupted file.
• Return Value - Returns true if operation(s) are successful, false otherwise
```

Examples

First Example - Injecting Faults

The following is a simple example of a Managed C++ Application using Holoscript to launch the Windows Notepad application and inject an **AccessDenied** fault into it. Every time Windows tries to access a file (or directory), it will receive the **AccessDenied** error, and it will not be able to access the file.

```
#include "stdafx.h"
#using <FunctionDatabase.dll>
#using <mscorlib.dll>
#include <tchar.h>

#using <HoloScript.dll>

using namespace System;
using namespace HoloScript;

int _tmain(void)
{
    // create a new interception framework
    HoloScriptApplication *holoApp =
        new HoloScriptApplication();

    // launch Notepad as the application under test, this call
    // disables logging of all
    // functions in order to increase performance.
    holoApp->LaunchApplication(
        "C:\\\\WINDOWS\\\\NOTEPAD.EXE", NULL,
        false,
        holoApp->GetFilteredFunctionList());

    /* Notepad will now receive AccessDenied every
     * time it tries to access a file (or directory)
     */
    holoApp->InjectFault(NULL,
        FunctionDatabase::FaultType::AccessDenied);

    /* wait for user input to terminate the
     * interception
     */
    Console::WriteLine(
        "Press [ENTER] to Terminate Application");
    Console::ReadLine();

    // terminate the interception
    holoApp->TerminateApplication();

    return 0;
}
```


Second Example - Examining AUT Execution Environment

The following is a simple example of a Managed C++ Application that uses Holoscript to launch the Windows Notepad application. Once Notepad is running, its process ID and thread ID are written to the console. The application also gets the current Disk and Memory usage information and writes it to the console.

```
#include "stdafx.h"
#using <FunctionDatabase.dll>
#using <mscorlib.dll>
#include <tchar.h>
|
#using <HoloScript.dll>

using namespace System;
using namespace HoloScript;

int _tmain(void)
{
    UInt32      processId, threadId;
    UInt64      maxDiskUse, curDiskUse,
                maxMemUse, curMemUse;

    // create a new interception framework
    HoloScriptApplication *holoApp =
        new HoloScriptApplication();

    // launch Notepad as the application under test
    holoApp->LaunchApplication(
        "C:\\\\WINDOWS\\\\NOTEPAD.EXE", NULL,
        false,
        holoApp->GetFilteredFunctionList());

    /* get the application process ID and write it
     * to the console
     */
    holoApp->GetApplicationProcessID(processId);
    Console::Write("Process ID: ");
    Console::WriteLine(processId);

    /* get the application thread ID and write it
     * to the console
     */
    holoApp->GetApplicationThreadID(threadId);
    Console::Write("Thread ID: ");
    Console::WriteLine(threadId);

    /* get the resource limits and utilization
     * and write them to the console
     */
    holoApp->GetMonitorInformation(maxDiskUse,
                                    curDiskUse, maxMemUse, curMemUse);
    Console::Write("Disk Usage: ");
```

```
Console::Write(maxDiskUse);
Console::Write(" (bytes max), ");
Console::Write(curDiskUse);
Console::WriteLine(" (bytes current)");
Console::Write("Memory Usage:   ");
Console::Write(maxMemUse);
Console::Write(" (bytes max), ");
Console::Write(curMemUse);
Console::WriteLine("(bytes current)");
Console::WriteLine();

/* wait for user input to terminate the
 * interception
 */
Console::WriteLine(
    "Press [ENTER] to Terminate Application");
Console::ReadLine();

// terminate the interception
holoApp->TerminateApplication();

return 0;
}
```

Third Example - Modifying the AUT Execution Environment (Disk and Memory Limits)

The following is an extension of the last example, Examining Application Under Test Execution Environment. This application sets the maximum amount of disk that can be allocated by the Application Under Test to 128 bytes and sets the maximum amount of memory that can be allocated by the Application Under Test to 640 kilobytes.

```
#include "stdafx.h"
#using <FunctionDatabase.dll>

#using <mscorlib.dll>
#include <tchar.h>

#using <HoloScript.dll>

using namespace System;
using namespace HoloScript;

int _tmain(void)
{
    UInt64      maxDiskUse, curDiskUse,
                maxMemUse, curMemUse;

    // create a new interception framework
    HoloScriptApplication *holoApp =
        new HoloScriptApplication();

    // launch Notepad as the application under test
    holoApp->LaunchApplication(
        "C:\\\\WINDOWS\\\\NOTEPAD.EXE", NULL,
        false,
        holoApp->GetFilteredFunctionList());

    // set the maximum disk usage to 128 bytes
    holoApp->SetMaximumDiskUsage(128, true);

    // set the maximum memory usage to 640 kilobytes
    holoApp->SetMaximumMemoryUsage(640*1024, true);

    /* get the resource limits and utilization
     * and write them to the console
     */
    holoApp->GetMonitorInformation(maxDiskUse,
                                    curDiskUse, maxMemUse, curMemUse);
    Console::Write("Disk Usage: ");
    Console::Write(maxDiskUse);
    Console::Write(" (bytes max), ");
    Console::Write(curDiskUse);
    Console::WriteLine(" (bytes current)");
    Console::Write("Memory Usage: ");
```

```
Console::Write(maxMemUse);
Console::Write(" (bytes max), ");
Console::Write(curMemUse);
Console::WriteLine("(bytes current)");
Console::WriteLine();

/* wait for user input to terminate the
 * interception
 */
Console::WriteLine(
    "Press [ENTER] to Terminate Application");
Console::ReadLine();

// terminate the interception
holoApp->TerminateApplication();

return 0;
}
```

Fourth Example - Modifying the AUT Execution Environment (Network Limits)

The following application sets the maximum speed of the system network throughput that can be allocated by the Application Under Test to maximum 50 bytes/sec. In other words, the Application Under Test can only use up to 50 bytes/sec capacity.

```
#include "stdafx.h"
#using <FunctionDatabase.dll>
#using <mscorlib.dll>
#include <tchar.h>

#using <HoloScript.dll>

using namespace System;
using namespace HoloScript;

int _tmain(void)
{
    UInt64 netDwldBandwidth;
    UInt64 netUpLdBandwidth;

    // create a new interception framework
    HoloScriptApplication *holoApp = new HoloScriptApplication();

    /* launch Internet Explorer as the
     * application under test
     */
    holoApp->LaunchApplication(
        "C:\\\\PROGRAM FILES\\\\INTERNET EXPLORER\\\\IEEXPLORE.EXE",
        NULL,
        false,
        holoApp->GetFilteredFunctionList());

    // set upload and download speeds to max 50 bytes/sec
    holoApp->SetNetworkUploadBandwidth(50, true);
    holoApp->SetNetworkDownloadBandwidth(50, true);

    // get the network speeds and write them to the console
    holoApp->GetNetworkUploadBandwidth(netUpLdBandwidth);
    Console::Write("Network upload speed: ");
    Console::WriteLine(netUpLdBandwidth);
    holoApp->GetNetworkDownloadBandwidth(netDwldBandwidth);
    Console::Write("Network download speed: ");
    Console::WriteLine(netDwldBandwidth);
    Console::WriteLine();

    /* wait for user input to terminate the
     * interception
     */
    Console::WriteLine(
        "Press [ENTER] to Terminate Application");
    Console::ReadLine();
}
```

```
// terminate the interception  
holoApp->TerminateApplication();  
  
return 0;  
}
```

Using HEAT (Hostile Environment for Application Testing)

Using HEAT

HEAT (Hostile Environment for Application Testing) is the foundation on which Holodeck and its supporting tools like HoloScript are built. All of the standard interception procedures within HEAT can be accessed by developers using the HeatAPI. This allows you to use C++ to create your own separate executables that utilize HEAT technology or extend Holodeck by creating separate DLLs that implement extended external API calls.

Important:

initializeApp, **attachToApp**, or **attachToRunningApp** must be called before calling other functions.

forceInject needs to be called before interception will actually take place

initializeApp starts the process suspended, so **runApp** has to be called before it will begin executing

interceptFunc can be called before **forceInject** to queue up interceptions before actually performing the injection into the process.

Note: when defining replacement libraries you must ensure that the calling convention is maintained (e.g. __stdcall etc.) If this is not properly maintained, several problems may occur, including crashes, hangs, etc. which may be hard to debug.

Step by Step for HEAT API

- 1) User will be required to create an impostor dll, and an exe.
- 2) In the impostor dll, the user must create impostor functions for every function that he plans to intercept.
- 3) The impostor function must have the same signature as the function it is replacing (same return value, same parameters).
- 4) The impostor function should call the real function once the user is done manipulating the parameters etc, unless the impostor function does all that the real function does.
- 5) Make sure all the impostor functions are exported by the impostor dll. HEAT will not be able to intercept real functions properly if imposter functions are not exported by the imposter dll.
- 6) Once the impostor dll is complete, write the code for the exe.
- 7) Include CHeatApp.h, and specify HeatApi.Lib in the dependencies\libraries for the project (both files are present in : Holodeck Install Directory\HEAT folder).
- 8) After creating a CHeatApp object, the user can use the various functions in the help to launch/manipulate application under test.
- 9) Functions to be intercepted can be specified using InterceptFunc from the Heat API.
- 10) It is recommended to start the application using initializeApp(), and then inject the heat dll into the target process using forceInject().
- 11) Once done injecting HEAT and setting up function intercepts, use runApp() to run the target application.

HeatAPI Method List

Constructors and Destructors of HeatAPI

CHeatApp()	This default Constructor creates a new instance of a CHeatApp
~CHeatApp()	This default Destructor releases any base resources allocated by the CHeatApp

Interception Information and Control

initializeApp	Launches an application with the given path and executable name. The app will be initially suspended.
attachToApp	Attaches to a closed application.
attachToRunningApp	Attaches to a running application
detachFromApp	Detaches from (stops interception of) the application that is currently being intercepted.
deinitializeApp	Terminates the process HEAT is currently attached to and removes all intercepts from the intercept list.
isReady	Returns true if an application is ready to run and be intercepted, false otherwise
isRunning	Returns true if the application is running
setIsRunning	Allows procedure to set running state
setIsReady	Allows procedure to set ready state
setIsInjected	Allows procedure to set injected state
getProcessId	Returns the ID of the current process
getProcessHandle	Returns the handle of the current process.
getThreadId	Returns the ID of the current thread.
getThreadsInCurrentProcess	Returns an array of thread IDs being used by the process

HEAT Application Execution

runApp	Begins execution of the process
pauseApp	Suspends execution of the process
setPauseState	For Externally setting the pause state of the application
resumeApp	Continues execution of the process
terminateApp	Terminates execution of the process

Intercept Function Control

interceptFunc	Intercepts a function and redirects it.
unInterceptFunc	Stops intercepting calls from the Application Under Test
deInitOnProcessTerminate	Automatically unintercept all functions and detach from the process if the given process ID terminates.

HEAT API Injection

forceInject	Injects HEAT into the attached process.
delayedInject	Injects HEAT into a process that hasn't been initialized yet.

Application Execution Information and Control

runApp	Resumes execution of the process.
pauseApp	Pauses execution of the process.
setPauseState	Forces the pause state of the process.
resumeApp	Resumes execution of the current process if it is paused
terminateApp	Terminates execution of the current process
getProcessId	Returns the process ID of the current process handled by this CHeatApp instance.
getThreadId	Returns the thread ID of the current process handled by this CHeatApp instance
getThreadsInCurr	The thread IDs for all of the threads associated with the current
entProcess	process handled by this CHeatApp are stored in threadList
isRunning	Returns true if an application is currently running and being intercepted, false otherwise
getProcessHandle	Gets a handle to the attached process.
isReady	Returns true if HEAT is ready to accept commands. If false, only initializeApp, attachToApp, or attachToRunningApp can be called.
isRunning	Determines if the app is executing. Returns false once the app has terminated.
setIsRunning	Allows the caller to override the value returned from isRunning.
setIsReady	Allows the caller to override the value returned from isReady
setIsInjected	Allows the caller to override the current injection state

Constructors and Destructors of HeatAPI

`CHeatApp()`

- This default Constructor creates a new instance of a **CHeatApp**

`~CHeatApp()`

- This default Destructor releases any base resources allocated by the **CHeatApp**

Interception Information and Control

DWORD initializeApp(LPSTR pszAppPath, LPSTR pszAppName)

- **Description** - Launches an application with the given path and executable name. The app will be initially suspended.
 - **pszAppPath** specifies the application path
 - **pszAppName** specifies the application executable name
- **Return Value** - Returns **HEAT_SUCCESS** if operation succeeds, otherwise **HEAT_ALREADY_ATTACHED**, **HEAT_INVALID_PARAMETERS** or **HEAT_FAILED_CREATE_PROCESS**

DWORD attachToApp(const PROCESS_INFORMATION& procInfo)

- **Description** - Attaches to an application using a PROCESS_INFORMATION structure filled in from a previous call to CreateProcess. This form should be used if the app has just been created and it was created suspended. Use attachToRunningApp if the app was running normally. The app will be in the same state as it was before this call.
 - **procInfo** specifies the process on which to begin interception
- **Return Value** - Returns **HEAT_SUCCESS** if operation succeeds, otherwise **HEAT_ALREADY_ATTACHED** or **HEAT_FAILED_OPEN_PROCESS**

DWORD attachToApp(DWORD dwProcessId)

- **Description** - Attaches to an application using a process ID. This form should be used if the app has just been created and it was created suspended. Use attachToRunningApp if the app was running normally. The app will be in the same state as it was before this call.
 - **dwProcessId** specifies the process on which to begin interception
- **Return Value** - Returns **HEAT_SUCCESS** if operation succeeds, otherwise **HEAT_ALREADY_ATTACHED** or **HEAT_FAILED_OPEN_PROCESS**

DWORD attachToRunningApp(const PROCESS_INFORMATION& procInfo)

- **Description** - Attaches to an application using a PROCESS_INFORMATION structure filled in from a previous call to CreateProcess. This form should be used if the app is running normally. Use attachToApp if the app has just been created suspended and hasn't been resumed yet. The app will be in the same state as it was before this call.
 - **procInfo** specifies the process on which to begin interception
- **Return Value** - Returns **HEAT_SUCCESS** if operation succeeds, otherwise **HEAT_ALREADY_ATTACHED** or **HEAT_FAILED_OPEN_PROCESS**

DWORD attachToRunningApp(DWORD dwProcessId)

- **Description** - Attaches to an application using a process ID. This form should be used if the app is running normally. Use attachToApp if the app has just been created suspended and hasn't been resumed yet. The app will be in the same state as it was before this call.
 - **dwProcessId** specifies the process on which to begin interception
- **Return Value** - Returns **HEAT_SUCCESS** if operation succeeds, otherwise **HEAT_ALREADY_ATTACHED** or **HEAT_FAILED_OPEN_PROCESS**

DWORD detachFromApp()

- **Description** - Detaches from (stops interception of) the application that is currently being intercepted
- **Return Value** - Returns **HEAT_SUCCESS** if operation succeeds, otherwise **HEAT_NOT_READY**

DWORD deinitializeApp()

- **Description** - Terminates the process HEAT is currently attached to and removes all intercepts from the intercept list.

- **Return Value** - Returns **HEAT_SUCCESS** if operation succeeds, otherwise **HEAT_NOT_READY**

bool isReady()

- **Return Value** - Returns true if an application is ready to run and be intercepted, false otherwise

bool isRunning()

- **Return Value** - Returns true if the application is running

void setIsRunning(bool value)

- **Description** - allows thread procedure to set running state

void setIsReady(bool value)

- **Description** - Allows thread procedure to set ready state

void setIsInjected(bool value)

- **Description** - Allows thread procedure to set injected state

DWORD getProcessId()

- **Return Value** - Returns the process ID of the current process

HANDLE getProcessHandle()

- **Return Value** - Returns the handle of the current process

DWORD getThreadId()

- **Return Value** - Returns the ID of the current thread

DWORD getThreadsInCurrentProcess(DWORD *threadList)

- **Return Value** - Returns an array of thread IDs being used by the process

Intercept Function Control

```
DWORD interceptFunc(LPSTR pszFuncToIntercept,
                    LPSTR pszFuncDLL,
                    LPSTR pszHandlerFunc,
                    LPSTR pszHandlerDLL)
```

- **Description** - Intercepts **pszFuncToIntercept** from the DLL at **pszFuncDLL**, redirecting it to **pszHandlerFunc** from the DLL at **pszHandlerDLL**. This function can be called before forceInject. All intercepts that are in the interception list will automatically be enabled when forceInject is called.
 - **pszFuncDLL** specifies the location of the DLL containing **pszFuncToIntercept**
 - **pszHandlerDLL** specifies the location of the DLL containing **pszHandlerFunc**
- **Return Value** - Returns **HEAT_SUCCESS** if operation succeeds, otherwise **HEAT_INVALID_PARAMETERS**, **HEAT_ALREADY_INTERCEPTED** or **HEAT_FAILED_INTERCEPT**
- **Note:** intercepting overloaded functions requires using the mangled function name, for more information see the Intercepting Overloaded Functions with HEAT help topic

```
DWORD interceptFunc(PVOID pFuncToIntercept,
                    LPSTR pszHandlerFunc,
                    LPSTR pszHandlerDLL)
```

- **Description** - Intercepts function at address **pFuncToIntercept**, redirecting it to **pszHandlerFunc** from the DLL at **pszHandlerDLL**. This function can be called before forceInject. All intercepts that are in the interception list will automatically be enabled when forceInject is called.
 - **pszHandlerDLL** specifies the location of the DLL containing **pszHandlerFunc**
- **Return Value** - Returns **HEAT_SUCCESS** if operation succeeds, otherwise **HEAT_INVALID_PARAMETERS**, **HEAT_ALREADY_INTERCEPTED** or **HEAT_FAILED_INTERCEPT**
- **Note:** intercepting overloaded functions requires using the mangled function name, for more information see the Intercepting Overloaded Functions with HEAT help topic

```
DWORD unInterceptFunc(LPSTR pszFuncName,
                      LPSTR pszFuncDLL)
```

- **Description** - Stops intercepting calls from the Application Under Test to **pszFuncName**
 - **pszFuncDLL** specifies the location of the DLL containing **pszFuncName**
- **Return Value** - Returns **HEAT_SUCCESS** if operation succeeds, otherwise **HEAT_INVALID_PARAMETERS** or **HEAT_FAILED_UNINTERCEPT**

```
DWORD unInterceptFunc(PVOID pFuncToIntercept)
```

- **Description** - Stops intercepting calls from the Application Under Test to **pFuncToIntercept**
- **Return Value** - Returns **HEAT_SUCCESS** if operation succeeds, otherwise **HEAT_INVALID_PARAMETERS** or **HEAT_FAILED_UNINTERCEPT**

```
DWORD deInitOnProcessTerminate(DWORD pid);
```

- **Description** - Automatically unintercept all functions and detach from the process if the given process ID terminates. Commonly used to auto-detach if the tool using HEAT API crashes (by passing in the result of GetCurrentProcessId).

HEAT API Injection

DWORD forceInject()

- **Description** - Injects HEAT into the attached process. Must be called for interception to actually take place.
- **Return Value** - Returns **HEAT_SUCCESS** if operation succeeds, otherwise **HEAT_NOT_READY** or **HEAT_ALREADY_INJECTED**

DWORD delayedInject()

- **Description** - Injects HEAT into a process that hasn't been initialized yet. This is only useful if intercepting a process before completion of the CreateProcess call that created the process. Nearly all apps should use forceInject.
- **Return Value** - Returns **HEAT_SUCCESS** if operation succeeds, otherwise **HEAT_NOT_READY** or **HEAT_ALREADY_INJECTED**

HEAT API Example

```
.cpp file for Application Interception
#include <CHeatApp.h>

int main(void)
{
    CHeatApp *testApp = new CHeatApp();

    DWORD retVal = 0;

    retVal = testApp->initializeApp("D:\\WINDOWS",
"NOTEPAD.EXE");
    if (retVal != HEAT_SUCCESS)
        MessageBox (NULL, "not initialized", "Heat Test", 0);

    retVal = testApp->forceInject();
    if (retVal != HEAT_SUCCESS)
        MessageBox (NULL, "not injected", "Heat Test", 0);

    retVal = testApp->interceptFunc("GetStartupInfoA" ,
"kernel32.dll", "GetStartupInfoARemplacement",
"D:\\TestProjDll.dll");
    if (retVal != HEAT_SUCCESS)
        MessageBox (NULL, "not intercepted", "Heat Test", 0);

    retVal = testApp->interceptFunc("GetStartupInfoW" ,
"kernel32.dll", "GetStartupInfoWReplacement",
"D:\\TestProjDll.dll");
    if (retVal != HEAT_SUCCESS)
        MessageBox (NULL, "not intercepted", "Heat Test", 0);

    retVal = testApp->interceptFunc("GetProcessHeap" ,
"kernel32.dll", "GetProcessHeapReplacement",
"D:\\TestProjDll.dll");
    if (retVal != HEAT_SUCCESS)
        MessageBox (NULL, "not intercepted", "Heat Test", 0);

    testApp->runApp();

    return 0;
}
```

```
.cpp file for DLL
/*
    TestProjDll.cpp
*/
#include "TestProjDll.h"
```

```

BOOL APIENTRY DllMain( HANDLE hModule, DWORD ul_reason_for_call,
LPVOID lpReserved )
{
    switch (ul_reason_for_call)
    {
        case DLL_PROCESS_ATTACH:
        case DLL_THREAD_ATTACH:
        case DLL_THREAD_DETACH:
        case DLL_PROCESS_DETACH:
            break;
    }
    return TRUE;
}

TESTPROJDLL_API VOID GetStartupInfoAReplacement (LPSTARTUPINFOA
lpStartupInfo)
{
    //Modify any parameters that you want here

    //Call the real function
    GetStartupInfoA(lpStartupInfo);

    //Get the last error set by the function
    DWORD lastError = GetLastError();

    //You may modify the error code or return value here
    //lastError =
    //returnValue =

    //These should be the last two calls, no user code should
    go below them.
    SetLastError (lastError);
}

TESTPROJDLL_API VOID GetStartupInfoWReplacement (LPSTARTUPINFOW
lpStartupInfo)
{
    //Modify any parameters that you want here

    //Call the real function
    GetStartupInfoW(lpStartupInfo);

    //Get the last error set by the function
    DWORD lastError = GetLastError();

    //You may modify the error code or return value here
    //lastError =
    //returnValue =

    //These should be the last two calls, no user code should
    go below them.
    SetLastError (lastError);
}

```

```

}

TESTPROJDLL_API HANDLE GetProcessHeapReplacement ()
{
    //Modify any parameters that you want here

    //call the real function
    HANDLE returnValue = GetProcessHeap ();

    //Get the last error set by the function
    DWORD lastError = GetLastError();

    //You may modify the error code or return value here
    //lastError =
    //returnValue =

    //These should be the last two calls, no user code should
    go below them.
    SetLastError (lastError);
    return returnValue;
}

```

.h file for DLL

```

// All files within this DLL have to be compiled with the symbol
TESTPROJDLL_EXPORTS
// symbol defined on the command line. This symbol should not be
defined on any project
// that uses this DLL. This way any other project whose source
files include this file see
// IMPDLL_API functions as being imported from a DLL, whereas
this DLL sees symbols
// defined with this macro as being exported.

#ifndef TESTPROJDLL_EXPORTS
#define TESTPROJDLL_API __declspec(dllexport)
#else
#define TESTPROJDLL_API __declspec(dllimport)
#endif

#include <windows.h>

TESTPROJDLL_API VOID GetStartupInfoAReplacement (LPSTARTUPINFOA
lpStartupInfo);
TESTPROJDLL_API VOID GetStartupInfoWReplacement (LPSTARTUPINFOW
lpStartupInfo);
TESTPROJDLL_API HANDLE GetProcessHeapReplacement ();

```

Definitions file

```

LIBRARY TESTPROJDLL
EXPORTS
GetStartupInfoAReplacement
GetStartupInfoWReplacement

```

`GetProcAddress`

Intercepting Overloaded Functions with HEAT

To intercept overloaded functions with Holodeck's HEAT, you must use the mangled function names. To find the mangled function names you must use depends.exe provided with Visual Studio.net

Steps to find out the mangled names:

- 1) Start Depends.exe from the Visual Studio.NET Command Prompt (found under the Visual Studio.NET tools section of the Visual Studio in the Start Menu).
- 2) Open the DLL containing the function.
- 3) On the right side, the bottom list contains a list of functions that are exported from the DLL. Right click anywhere in this list and choose "Undecorate C++ functions"
- 4) Find the desired function in the list.
- 5) Right click in the list, and select "Undecorate C++ Functions" again to show the mangled names.
- 6) Right click on the desired function and choose "Copy Function Name". This places the mangled function name in the clipboard.
- 7) Paste the mangled name into your HEAT project.

Appendices

About fault injection

Copied from "How to Break Software" Appendix A by James A. Whittaker
Addison-Wesley Publishing; Book and CD-ROM edition (May 9, 2002)

Testing Exception and Error Cases Using Runtime Fault Injection

Fault injection deals with the insertion or simulation of faults in order to test the robustness and fault tolerance of a software application [8]. Such measures are generally performed on software that is mission critical, to the extent that failure could have significant negative ramifications. Actual injection of faults can be performed either at compile time, when additional code is inserted to force error conditions to evaluate to true, or at runtime during which faults are injected into the software's execution environment. This paper focuses on the latter type of fault injection and presents a new mechanism for inserting environment faults. In addition, insight is provided into fault selection based on an analysis of runtime behavior. This paper presents a methodology and tool for performing runtime fault injection, both of which are demonstrated on a commercial software product.

1. Introduction

For the purposes of our discussion of fault injection, code comes in two forms:

1. *Functional code* is code that accomplishes the mission of the software by implementing user requirements. In other words, it is the code that does the work necessary for users to fulfill their purpose in using the software.
2. *Error handling code* is code that keeps the functional code from failing. Examples include code to check inputs for validity and code that ensures stored data does not exceed its defined type and value range.

By its very definition, functional code is readily accessible through the software's interface(s) (be they graphical user interfaces or programming interfaces). In fact, testing functional code is a fairly well established discipline, though often imprecise [9]. On the other hand, exercising error-handling code is generally trickier and may require more extreme measures. Of course, some error conditions are easy to handle; for example, some conditions require only that certain input values be entered incorrectly to be satisfied. But other error handling code may require considerably complicated environmental circumstances to arise before it will execute [8].

Consider the case in which developers write code to guard against a full storage medium. The straightforward way to set up this anomaly is for testers to generate and maintain many large data files—files that are large enough to fill the capacity of the local storage device. Not only are such files hard to generate, but keeping them around means that the storage device can serve no other purpose (because it is full) than to house a single test case. And full media is only one case out of many faulty file system possibilities. We also need to consider file corruption, access privileges (read-only, etc.), file permissions and damage to the actual media, among other scenarios.

Indeed, the file system itself is only one possible part of the environment that developers write error code against. We must also consider memory calls, network APIs, databases, third party components and controls, etc. All of these environmental elements can fail in ways that an application must expect and guard against especially when its mission is compromised [9].

In this paper, we discuss injecting faults into an application's environment at runtime to effectively and accurately trigger failures in a manageable fashion. We begin by describing the

runtime injection mechanism and provide specific examples for Microsoft's Windows® operating system. Next, we discuss in detail the types of faults that can be injected and the situations in which testers should use specific failure scenarios. Finally, we illustrate the technique by outlining results from a case study performed by the authors on a commercial software product.

2. A Mechanism for Runtime Fault Injection

Source-based fault injection can be complicated to achieve but is easy to explain: source statements are modified so that specific faulty behavior is attained [1],[2],[6]. When faults are injected to trigger exceptions, source statements are actually added so that internal data can be set to values which cause exception conditions to evaluate to true [3].

But source-based fault injection requires access to source code and, in most cases, the cooperation of the original developers, which is not always a given [8]. Release pressure is one reason that developers refuse to write such code. Further, it is often the case that many testers have no access to the source code. Either they are outsourcers or the build culture at their company does not support such involvement.

Regardless of the organizational factors that complicate source-based fault injection, runtime fault injection holds the benefit that the faults are more realistic. By inserting faults into the environment instead of the application, the latter, free of any additional code that may introduce unwanted behavior, is forced to react in exactly the same way as if the failures were real and not triggered by testers.

Environment faults can be forced either by reproducing the causal scenario or by simulation. For example, consider the case of the ubiquitous network through which many applications communicate with other applications or services. To test fault tolerance of network applications, one might physically damage the network by, say, unplugging the cable or by sabotaging the network adapter. Further, one could cause a busy network by generating large amounts of bogus traffic (say by sending constant command line pings from a few dozen machines).

But these same results, and many others, can be achieved by simulating the exact same circumstances so that the fault affects the application under test but not the rest of the system. The key is to realize that any environmental fault will manifest as failed system calls made by the application. For example, an application sees a network outage as a series of failed calls made to the local socket API. The application sees low memory as failed calls to the kernel. The application sees file corruption as CRC errors raised by the function `CreateFile`, and so forth.

Ultimately, there is the reality of a failure and the reality of what an application actually sees when the failure occurs. It is this latter entity that we can recreate and it is at the system-application boundary that faults can be injected. These faults will affect only the system under test, allowing the machine to be useful for other purposes.

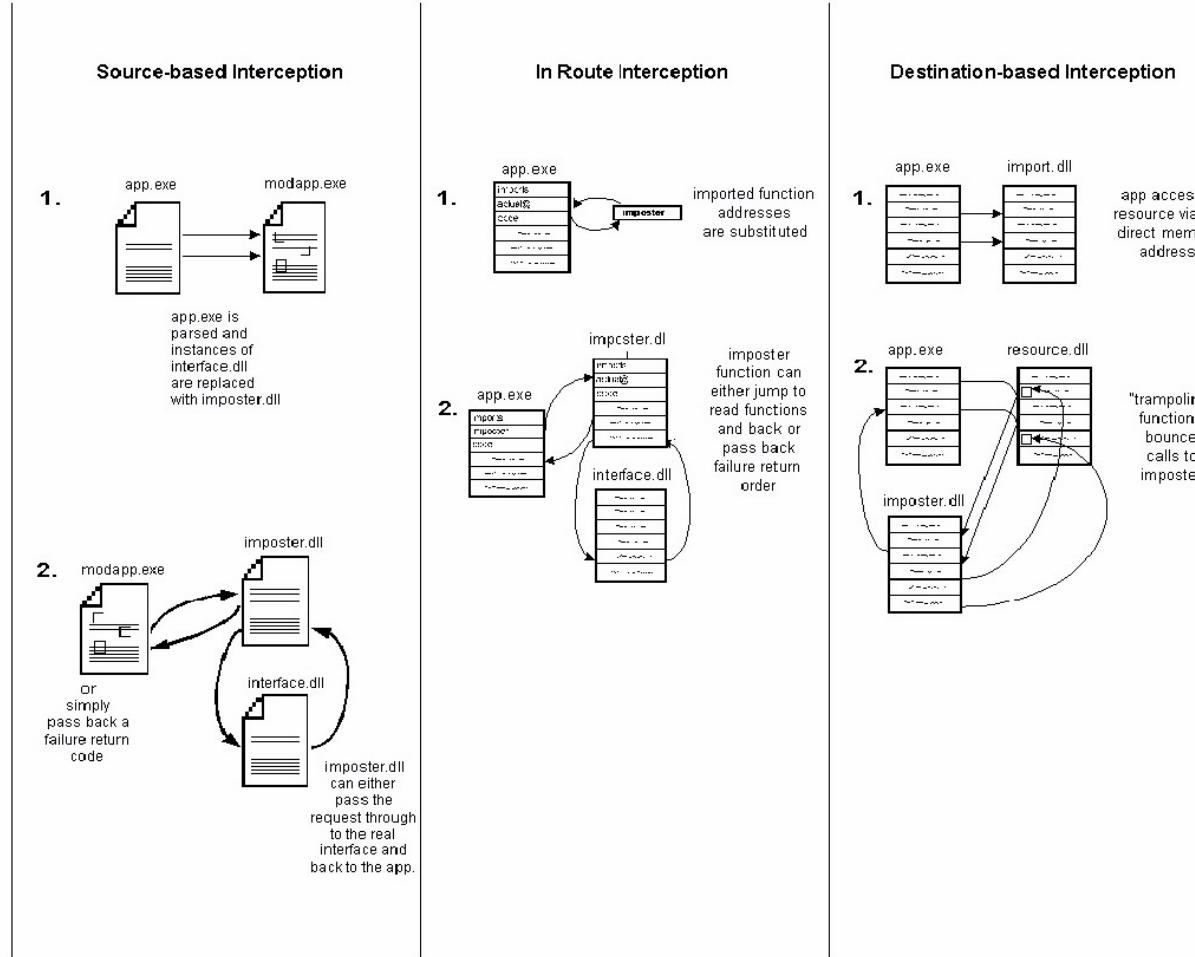
In order to understand how to interpret faults as failed system calls, we must first be able to capture system calls before they reach their destination. Then we must be able to record exactly how real faults manifest in the error codes and return values of these calls.

There are at least three ways to accomplish system call interception.

1. *Source-based interception* requires actual editing of a binary and replacing instances of the destination API with an imposter API. The imposter API then simply acts as a pass-through mechanism. For example, using a hex editor it is easy to search an executable for the string `kernel32.dll` and replace it with `mykernel32.dll`. We must then write imposter functions in `mykernel32.dll` with the same name as the functions in `kernel32.dll` that we want to fail. The imposter functions simply log the call and then call the real function in `kernel32.dll`. When `kernel32.dll` responds to `mykernel32.dll`, the imposter simply passes the error codes and return values back to the application.

2. *In route interception* can achieve the same effect as source-based interception without having to change an application's binary image on disk. Using techniques published in [7], one can modify addresses in function dispatch tables to divert calls to imposter functions. The

imposters then act as pass-through mechanisms as above. Of course, in route interception only works when calls are routed through a centralized function dispatch mechanism like import address tables. Since these tables are stored in memory, the application's binary does not have to be modified on disk.



4. requires inserting code into the function being called by the application. Unlike in route interception, which modifies memory addresses that are part of the application's code space, destination-based interception requires modifying the code space of the target function. In our implementation, we insert jump statements into the first few bytes of a function and copy those bytes to the imposter function. When the application makes the function call, the jump statement transfers control to the imposter function which will first execute its code and then transfer control back to the next executable memory location of the original function (i.e., past our inserted jump statement).

The above figure illustrates each of the three types of interception.

3. Fault Selection

We have employed two types of fault selection strategies and developed tools to help carry out each type. The first strategy consists of recording function calls made by an application and then systematically failing each call everywhere it is used in the application. We call this method systematic, call-based fault injection. For example, if we record that the kernel call `LocalLock` is used each time an application accesses a file, then we can cause `LocalLock` to fail and force the software through paths that have file opens, reads, writes, etc, so that the application sees the failure of `LocalLock`. Obviously, this is a time consuming and painstaking way to inject faults. The second strategy consists of staging a particular environment fault, recording the pattern of

failed function calls caused by the fault, and then simulating that pattern in other parts of the application. We call this method *pattern-based fault injection*. For example, we might stage an unresponsive network by unplugging the Ethernet cable and record that the application sees failed return code from any number of socket APIs. We can then fail these same APIs as a simulated substitute for physically unplugging the Ethernet cable.

3.1 Pattern-based Fault Injection

The ultimate question for fault injection is: **What faults should be injected?** The answers to this question are varied:

The faults should collectively cause all of the error code to be executed and exceptions to be tripped.

This is a typical developer-centric answer. As desirable as it is to execute all the source code of an application under test, this is usually unachievable given today's relatively short development cycles and aggressive deadlines. Other difficulties include access to source code and the use of sophisticated code coverage tools. Again, in practice, not all testing teams have access to source code, development teams, or the required tools that would enable them to stage code-based fault injection.

Only the faults that can be readily staged in the testing lab should be selected.

This, on the other hand, is a typical tester-centric answer. It may be hard for some to imagine that testers are required to consider and run scenarios that cannot be accomplished outside the testing lab. However, testing labs are often not representative of the setup and environment of the real world. Users typically have more data, more machines, bigger networks, more software, and a wider variety of hardware, peripherals, and drivers than can be in a lab. Users, therefore, are a great source of realistic scenarios that may not have been anticipated by developers and that may cause unexpected failures.

Only faults that may realistically occur in the field need to be injected.

Users expect that software will work well in their uncontrolled, generally unpredictable environments. However, since such environments are difficult or impossible to stage in the testing lab, we have developed a tool to help simulate some of the more common faulty scenarios. We call the general principle the *Hostile Environment Application Tester* and the tool "Canned HEAT. The purpose of Canned HEAT is to stage some realistic problems in the environment in an easy-to-use way.

Canned HEAT works on a simple principle. Every faulty environment causes digital symptoms that the application recognizes and that we can recreate. Take a network that has gone down for example. This can be caused by an unplugged cable, a misconfigured adaptor, faulty network software, or network congestion. The application only recognizes the symptom that certain API calls, say to the network port, are failing. That is, instead of working as expected, they are returning failure codes to the application. Therefore, any number of actual faults may end up producing the same symptoms. Canned HEAT is a tool that reproduces these symptoms so that the application runs as if an actual failure has occurred.

3.1.1 Memory Faults

The amount of memory that an application uses varies according to the task it is performing. Some tasks require very little memory and are unlikely to cause memory to be depleted. Other tasks consume vast amounts of memory. Such tasks along with other applications simultaneously running may deprive the application of the amount of memory necessary for normal operation. In order to determine those features that are memory intensive, Canned HEAT is equipped with a monitor that keeps track of an application's use of memory. Once a list of these features is gathered, the first set of tests consists of continually lowering the available memory threshold to determine where (or if) the application begins to falter.

The next step is to run scenarios that will test the application's reactions to varying memory conditions. Canned HEAT can randomly vary the amount of memory available to an application.

The intent is to simulate the real world scenario in which background applications access memory at sporadic times. The application is then run through its paces, concentrating on the memory-intensive features identified earlier.

The final test to perform is fault injection, and, with Canned HEAT, this is as simple as running the application and selecting a fault's check box at any time.

Consider the following example using Microsoft® Internet Explorer®.

Step 1. Use the application and determine which features are memory intensive.

This step can actually be performed while the application is being tested under ordinary circumstances. Simply launch the application under Canned HEAT and pay close attention to the memory monitor while you are using the application. Make a note of which features use the most memory. Obviously, disk-intensive operations like reading and writing files will cause memory to be used, but also, loading rich images, processing large files and performing any computationally intense function will require memory usage.

Step 2. Determine the application's lower bound threshold of tolerance to low memory.

It is now the time to see how the application fares with restricted memory resources. This can be accomplished by forcing the features to be exercised and simultaneously restricting the application's access to memory resources using Canned HEAT.

Canned HEAT has a convenient slider bar under the memory tab for this purpose. By simply sliding the bar to the left, the available main memory is decreased (see figure below).



Note the slider bar is all the way to the right, allowing the application access to all available memory. However, if we move it to the left and continue to use Internet Explorer's memory intensive features, we note that around 35MB, things begin to slow down tremendously. Further, if we take away all but about 15MB, Internet Explorer ceases to work at all.

Step 3. Run Canned HEAT's scenarios that randomly vary available memory.

Once this is determined and reported to development, the next set of tests to run concerns the application's ability to perform well under tremendously varying memory conditions.

Selecting the varying memory scenario will make the memory control slider unavailable, meaning that Canned HEAT has assumed control of deciding when and how much memory will be available to any given request made by the application under test.

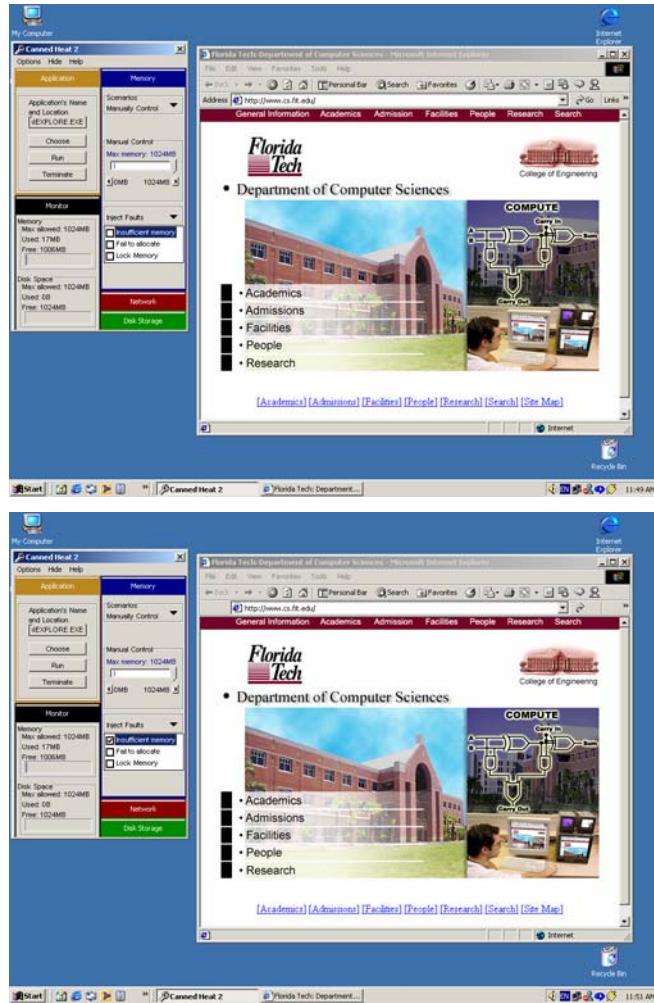
Using this scenario will often crash applications even when the human user is not working with them. This is because any given memory call may result in an artificial failure being injected by

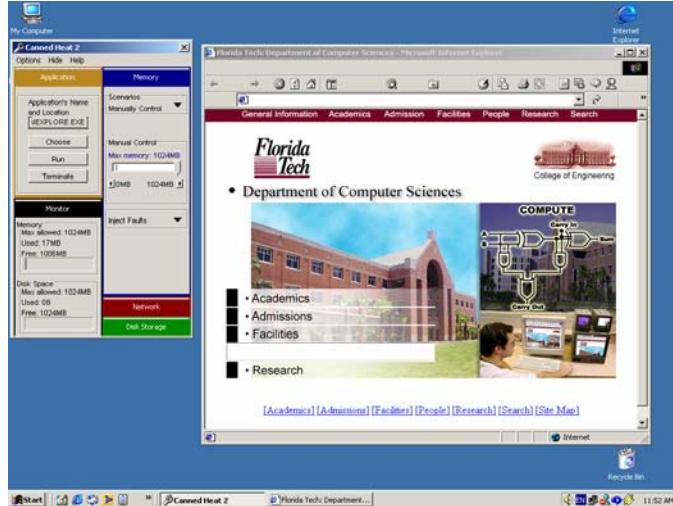
Canned HEAT. Such is the case with Internet Explorer as shown below.



Step 4. Inject faults at runtime during memory use.

Finally, the last set of tests involves injecting specific faults. Whereas the last two steps simply fail memory calls according to the amount of available memory, our tool allows individual faults to be injected regardless of the amount of memory available. The following series of figures shows an example of this in Internet Explorer. We use Canned HEAT to simulate an "insufficient memory" fault and watch as IE's controls simply disappear due to lack of adequate memory resources. Eventually, IE will hang.





3.1.2 Network Faults

We test network faults in the same four-stage process we have just demonstrated for memory faults. First, we will use the application and determine which features cause network activity. Second, we will use Canned HEAT's slider bar to slow the network until our application is unacceptably slow or until it crashes or hangs. Third, we will run scenarios that will vary the network speed over time, concentrating on those features that cause the most network activity. Fourth, we will inject specific faults, one at a time, and monitor the application's resulting behavior.

Let's consider an example.

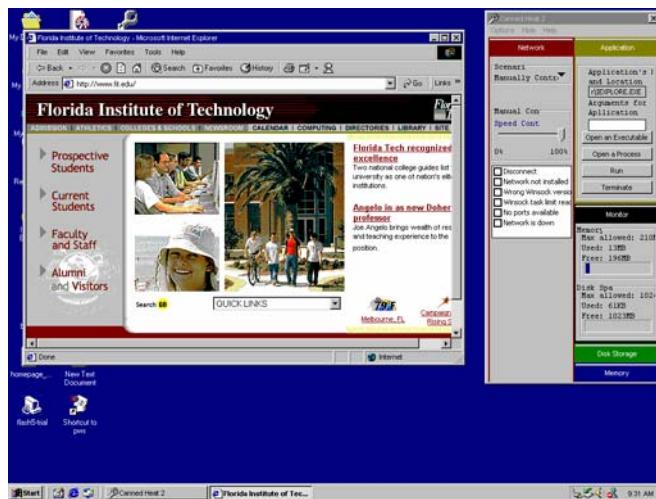
Step 1. Use the application and determine when it hits the network port.

The browser's most intense use of the network port occurs during file downloads and when web pages are served to it.

Step 2. Determine the application's lower bound threshold of tolerance to a slow network.

Once you have determined when the application hits the network port, it is interesting to find out the behavior of the application by reducing network speed. This can be achieved by using the application while manually reducing the network speed using Canned HEAT's network speed slider bar.

Canned HEAT allows the user to manually control the network speed the same way it does for available memory. Using the slider bar, the user can easily adjust the network speed to the desired percentage of the full capacity of the machine, on which the application is being tested. (see figure below)



The next two screenshots demonstrate the use of the network slider bar. The first one shows a perfectly loaded page while the network speed is 33% of its maximum. This shows that the loading of a page similar to this one does not require more than 33% of network speed.



This next screenshot though, shows an incomplete page (some pictures and menu titles are missing) with a network speed around 30%.



We thus determined the network speed threshold for which Internet Explorer can load accurately a page similar to the Florida Tech homepage, with respect to the number of graphics, animation etc. If we further lower the network speed, IE will not be able to load a page anymore.

Step 3. Run Canned HEAT's scenarios that randomly vary network speed.

The next step is to run tests using scenarios. Canned HEAT is programmed to simulate random network failures while the application is running. Examples of such failures are disabled network connection, unresponsive network port or failure of socket API's.

The following screenshot demonstrates the use of the Canned HEAT's random failures scenario. The network slider bar is unavailable when running random scenarios, as was the memory slider bar.



Step 4. Inject faults at runtime during network use.

The last tests to perform consist of injecting faults at runtime. The previous step demonstrated the use of the random failures scenarios. What we want to accomplish here is to study the behavior of the application when inserting specific faults.

Canned HEAT allows for inserting a number of faults including the "network is down" fault.

When this fault is inserted, we can watch Internet Explorer's reaction to a network that has become unresponsive.



3.2 Systematic Call-based Fault Injection

Canned HEAT is an easy to use tool to inject coarse-grained faults into an application's environment at runtime. It is not suitable for use when a more fine-grained, surgical approach is needed.

Canned HEAT works by failing sets of API calls either all of the time, most of the time or some of the time. However, the tester is given no ability to be more choosey than Canned HEAT's interface will allow. Sometimes, testers may want to fail a specific call only once. Or they may want to fail a call only in very specific contexts. In other words, they need a tool to observe APIs being called and have the ability to intervene on a call-by-call basis. This type of fault injection is called *observe-and-fail*.

There are many companies that have in house tools to do such fault injection and they seldom release their tools to the general market. I will attempt to explain how to use these tools using a prototype we have developed at Florida Tech called Holodeck. Any *Star Trek®* fan will immediately recognize the Holodeck as the virtual reality grid where holograms are indistinguishable from real people. Holodeck is our code name for a tool that makes faked software environments indistinguishable from real environments from the software's point of view.

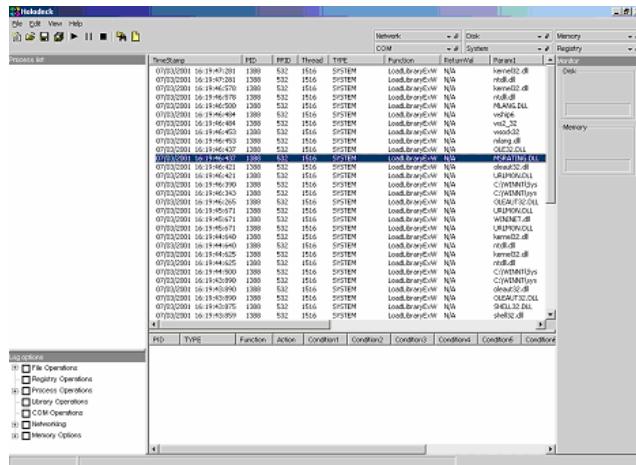
Here's how it works:

Similar to Canned HEAT, Holodeck is able to intercept API calls. Holodeck logs these calls so that testers can observe an application's activity and decide where to inject faults. Holodeck is equipped with filters that allow testers to narrow their search to very specific types of APIs.

Consider the following example (which represents a nice security exploit against the world's favorite web browser).

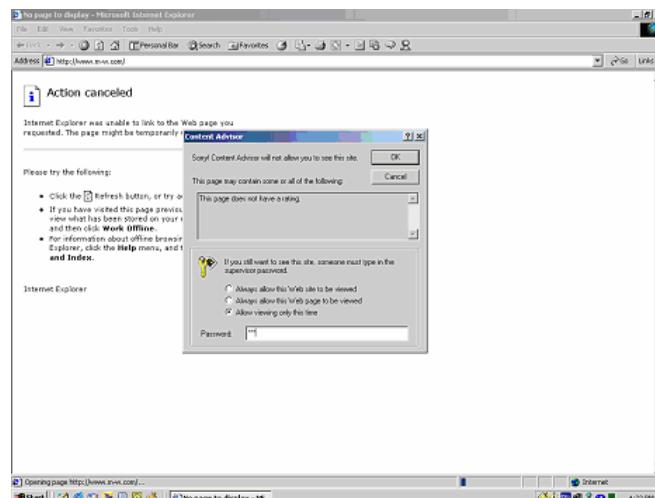
We begin the example by first using the target application and observing the system calls it makes (using Holodeck to view them). Most of these calls are mundane from a testing point of view but some are not and these can alert astute testers to possible attacks.

One such call is `LoadLibraryExW`. This call causes external code libraries to be loaded for use. One particularly suspicious such library is `MSRATING.DLL` as shown below.



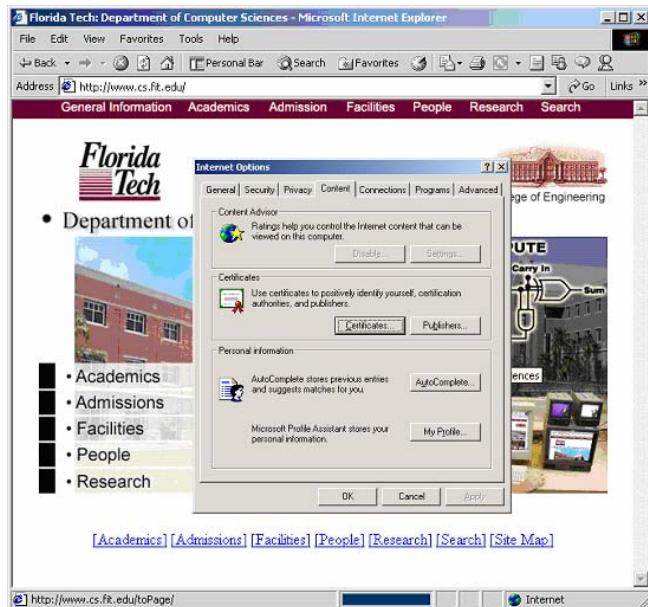
As a tester we are now alerted to the fact that this DLL is providing services to our application under test.

The desired behavior of the browser's rating system is to allow, say, a parent, to set up passwords for sites so that, say, their children cannot access them. When the browser is pointed to such a site, it will prompt for the password as shown below.



Now that the target system call has been identified, we can use Holodeck to inject a fault in the same manner as we used Canned HEAT. In this case, we will simply return a value indicating that the file MSRATING.DLL cannot be opened.

But failing the call to `LoadLibraryExW` causes the feature to be disabled, allowing anyone to surf anywhere they want. Note in the screen shot below, the blocked web site loads and the rating options is unavailable, as indicated by the inaccessibility of its icons.



4. Conclusions

Triggering exceptions can be very difficult at runtime. Creating scenarios that cause exception handlers to execute often involves a faulty environment that is not easy to stage in a laboratory. Thus, software is released without ever executing some exceptions or error handling code. Since user environments represent more diverse usage than is easily reproduced in testing labs, these exceptions are more likely in the field. This predicament is risky for software publishers who must release untested exception handlers, particularly publishers who release mission or safety critical applications.

Runtime software fault injection allows faulty environments to be simulated in the testing laboratory. Performed judiciously, software testers can increase coverage of error handling code and gain more confidence in their software's ability to perform robustly in an unstable environment.

The tool and methodology presented in this paper allow runtime fault injection to be performed without access to or modification of source code. By exposing system interfaces to interrogation, testers can reason about behaviors that may lead to exception handlers being executed. By modifying system-call return values and error codes dynamically, faults can be simulated so that the exact environment fault is presented to the application under test in a realistic manner. This mechanism is completely general, allowing most any type of stressed environment to be accurately simulated in a laboratory environment. As a result, the benefits can range from increased code coverage to a higher degree of confidence in the robustness of the application when it is deployed.

5. Acknowledgements

This work was supported in part by separate grants from Microsoft Research and Rational Software Corporation. We thank the remaining members of the HEAT and Canned HEAT development teams, which include Rahul Chaturvedi, Andres De Vivanco, Aditya Kakrania, Terry Lentz and John Brown. In addition, many thanks go to the testers at Microsoft and Rational for their insights into useful ways to fault inject. Special appreciation goes to Harry Robinson of Microsoft and Sam Guckenheimer of Rational for their input into Canned HEAT's user interface and fault selection methods.

6. References

- [1] Agrawal, H., et al, Design of mutant operators for the C programming language,

Technical Report SERC-TR-41-P, Software Engineering Research Center, Purdue University, West Lafayette, IN, (March 1989).

- [2] Bowser, J., Reference manual for Ada mutant operators, Technical Report GIT-SERC-88/02, Department of Computer Science, Georgia Institute of Technology, Atlanta, (February 1988).
- [3] Friedman, M. and Voas, J., Software assessment: reliability, safety, and testability, Wiley, (1995).
- [4] Ghosh, A. and Schmid, M., An approach to testing COTS software for robustness to operating system exceptions and errors. In Proceedings of 10th Int'l Symposium on. Software Reliability Eng., (Los Alamitos, CA, 1999) IEEE Computer Society Press, 166-174.
- [5] Houlihan, P. Targeted software fault insertion, Proceedings of STAR EAST 2001 (Software Testing Analysis and Review), (Orlando FL, 2001), Software Quality Engineering.
- [6] King, K. and Offut, A.J. A Fortran language system for mutation-based software testing, Software Practice and Experience, 21 7, (July 1991), 685-718.
- [7] Richter, J. Programming applications for Microsoft windows, Microsoft Press, (1997).
- [8] Voas, J. and McGraw, G. Software fault injection: Inoculating programs against errors, Wiley, NY, (1998).
- [9] Whittaker, J. What is software testing. And why is it so hard. IEEE Software, 17, 1, (2000), 70-79.
- [10] Whittaker, J. Software's invisible users. IEEE Software, 18, 3, (2001) 84-88.

How to Break Software

[Purchase this book](#)

Book Info

A practical tutorial on how to actually do testing by presenting numerous 'attacks' you can perform to test your software for bugs. The testing techniques are as flexible as conventional testing is rigid. Softcover.

From the Back Cover

Practical tutorial on how to actually do testing by presenting numerous "attacks" you can perform to test your software for bugs.

* Practical approach has little or no theory, but shows real ways effectively test software—accessible to beginners and seasoned testers.

* The author is well known and respected as an industry consultant and speaker.

* Uses market leading, and immediately identifiable, software applications as examples to show bugs and techniques. How to Break Software is a departure from conventional testing in which testers prepare a written test plan and then use it as a script when testing the software. The testing techniques in this book are as flexible as conventional testing is rigid. And flexibility is needed in software projects in which requirements can change, bugs can become features and schedule pressures often force plans to be reassessed. Software testing is not such an exact science that one can determine what to test in advance and then execute the plan and be done with it. Instead of a plan, intelligence, insight, experience and a "nose for where the bugs are hiding" should guide testers. This book helps testers develop this insight. The techniques presented in this book not only allow testers to go off-script, they encourage them to do so. Don't blindly follow a document that may be out of date and that was written before the product was even testable. Instead, use your head! Open your eyes! Think a little, test a little and then think a little more. This book does teach planning, but in an "on-the-fly while you are testing" way. It also encourages automation with many repetitive and complex tasks that require good tools (one such tool is shipped with this book on the companion CD). However, tools are never used as a replacement for intelligence. Testers do the thinking and use tools to collect data and help them explore applications more efficiently and effectively.

James A. Whittaker is a well-known speaker and consultant, as well as seasoned professor.

- **Paperback:** 208 pages
- **Publisher:** Addison-Wesley Publishing; Book and CD-ROM edition (May 9, 2002)
- **ISBN:** 0201796198

How to Break Software Security

Purchase this Book

Description

How to Break Software Security describes the general problem of software security in a practical perspective from a software tester's point of view. It defines prescriptive techniques (attacks that testers can use on their own software) that are designed to ferret out security vulnerabilities in software applications. The book's style is easy to read and provides readers with the techniques and advice to hunt down security bugs and see that they're destroyed before the software is released.

Accompanying the book is a CD-ROM containing Holodeck lite, which tests for security vulnerabilities. There are also a number of bug-finding tools, freeware, and an easy-to-use port scanner included on the CD-ROM.

- **Paperback:** 208 pages
- **Publisher:** Addison-Wesley Publishing; 1st edition (May 16, 2003)
- **ISBN:** 0321194330

Windows 2000 and attaching the Holodeck Debugger

Windows 2000 is incapable of detaching a debugger already attached to an application. Thus if the Holodeck Debugger option is enabled, it isn't possible for Holodeck to detach from the application. For more information please see the related help topic.

This has the following effects:

- 1) The menu item for attaching/detaching the Holodeck debugger to the application during the session is not present under Windows 2000, as this is not possible under Windows 2000.
- 2) If the attach debugger option is enabled, through the New Project Wizard the application will terminate when the project is closed.
- 3) If attaching to a service, the attach debugger is disabled by default, attaching to a service in Windows 2000 may potentially be dangerous as the service will be terminated forcefully.

This does not affect Windows XP or Longhorn.

API Call Categories

placeholder for other page

Error Codes

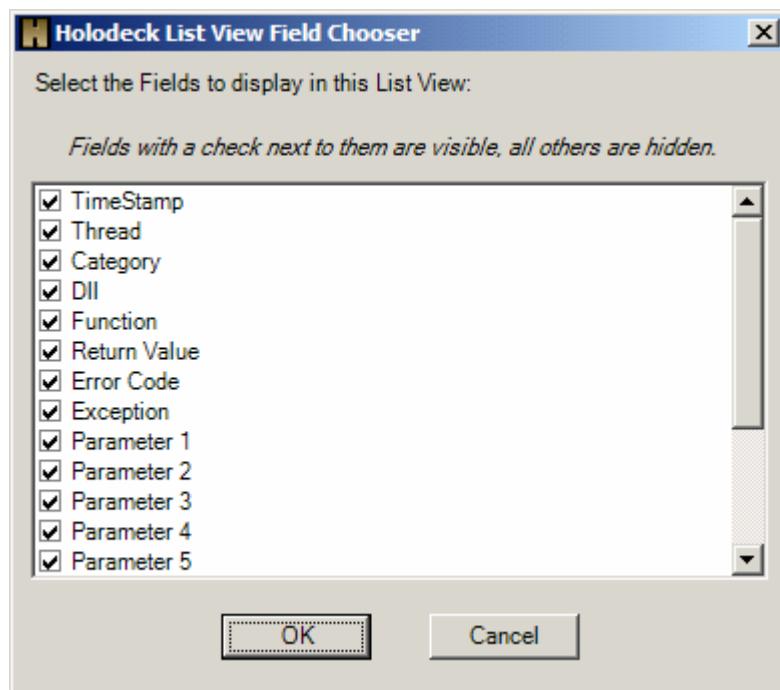
placeholder for other page

Exception Codes

placeholder for other page

Field Chooser

The Field Chooser allows you to select which columns are visible in the currently displayed pane with selectable columns. To access the Field Chooser open the pane you wish to change and click View > "Filed Chooser"



HEAT Application Execution

DWORD runApp();

- **Description** - Begins execution of the process

DWORD pauseApp();

- **Description** - Suspends execution of the process

DWORD setPauseState(bool value);

- **Description** - For externally setting the pause state of the application

DWORD resumeApp();

- **Description** - Continues execution of the process

DWORD terminateApp();

- **Description** - Terminates execution of the process

HEAT Error Codes

Success	0
Invalid Parameters	1
Already Attached	2
Failed to Create Process	3
Not Ready	4
Already Running	5
Failed to Resume Process	6
Failed to Terminate Process	7
Already Paused	8
Already Injected	9
Failed to Inject DLL	1 0
Already Intercepted	1 1
Failed to Intercept	1 2
Not Injected	1 3
Failed to Pause Process	1 4
Failed to Unintercept	1 5
Failed to Open Process	1 6
Failed to Create Event	1 7
Failed to Set Event	1 8
Failed to Send Deinitialize	1 9
HEAT Not Implemented	- 1

Index

A

About fault injection	440
Add a New Intercept Library	193
Add Holodeck Intercepts.....	342
Adding a second process to a project	106
Adding and Removing a Resource Fault.....	260
Adding custom faults	245
Adding New Intercepts Introduction.....	187
API Call Categories.....	456
API Log Summary Table	282
Application and Service Control.....	399
Application and Service Launch	393
Application Menu.....	45
Attach to a Service	218
Attach to an Application	216

B

Begin Code Coverage Test Generation	170
Begin testing	181

C

Changing which functions are logged	271
Code Coverage testing from the command-line	359
Comparing logs	161
Compile the TestDLL Replacement Library	196
Conclusion	127, 140
Constructors and Deconstructors	392
Constructors and Destructors of HeatAPI.....	428
Corrupted Files Pane	52
Corrupting Files	120
Create a Custom Test Project.....	367
Create a Custom Test Project Wizard	366
Create a File Corruption Fault using Find and Replace	318
Create a File Corruption Fault using Random Corruption	315
Create a File Corruption Fault with Regular Expressions.....	321
Create a File in Use fault for Notepad.exe	109
Create a Holodeck project file.....	179
Create a Memory Fault for BrokenApp.exe	110
Create a memory limit for Notepad.exe	111
Create a New Resource Fault Wizard	263
Create a New Test Project Introduction	202
Create a Regular Expression and Expansion String	135
Create a Test dll	188
Create the Custom Test Project.....	203
Create the Test Application.....	190
Creating a Disk Space limit.....	238
Creating a fault using Find and Replace Corruption	303
Creating a fault using Random Corruption	299

Creating a fault using Regular Expressions.....	307
Creating a Memory Space limit	239
Creating a Network Bandwidth limit	240
Creating a recorded session.....	339
Creating a Report for easy viewing.....	159
Creating a test.....	224
Creating a Test Based on Stack Matching.....	234
Creating Code Coverage Generation.....	293
Creating Stress Test Generation.....	291
Custom Faults	257
Custom Log Filtering and Sorting.....	269
Custom Test Project Summary.....	370

D

Dealing with Multiple Processes.....	330
Debugging Functions	398
Default Logging	221
Disk Faults	252
Dynamic Help Pane.....	54

E

Error Codes	457
Error Codes Summary Table.....	285
EULA	26
Exception Codes.....	458
Exception Pane.....	55
Exceptions Overview	335
Exporting logs	160
Exporting Logs.....	272
Exporting Resource Logs.....	274

F

Failing the ReadLine function.....	145
Fault Control	411
Fault Injection	433
Faults Overview	242
Faults Pane	57
Faults Summary Table	283
Field Chooser	459
File Corruption Details Pane	58
File Corruption Faults Pane.....	59
File Corruption Overview	313
File Menu	42
Filtering the log result	158
First Example - Injecting Faults	415
Fourth Example - Modifying the AUT Execution Environment Network Limits.....	421
Further Exploration	200, 209

G

Getting started guide	10
-----------------------------	----

H

HEAT API Example	434
HEAT Application Execution	460
HEAT Error Codes.....	461

HeatAPI Method List	426
Help Menu.....	50
Help Pane	61
Holodeck Intercepts Manager	344
Holodeck support center.....	16
Holodeck Tutorials.....	11
HolodeckLib Application Execution and Control.....	376
HolodeckLib Constructors and Destructors	375
HolodeckLib Fault Control	381
HolodeckLib Interception Control	382
HolodeckLib Limit Control.....	380
HolodeckLib Log Operations.....	379
HolodeckLib Method List	373
HoloScript Method List.....	385
How to Break Software	453
How to Break Software Security.....	454
How to compare log files	182
How to use this guide.....	18

I

Installing Holodeck	22
Intercept Function Control	431
Intercepting Overloaded Functions with HEAT.....	438
Interception Information and Control.....	429
Introduction.....	91, 96, 105, 119, 129, 142, 156, 178
Introduction to Automatic Test Generation.....	290
Introduction to Automating Holodeck	353
Introduction to Holodeck in depth.....	211
Introduction to recoding and replaying	338
Introduction to using Holodeck as a command-line Utility	357
Investigate Logs	143
Investigate Network Packets	130
Investigate Resources.....	97
Investigating the Failure.....	101

L

Launch an Application	215
Limits Control	409
Limits overview	236
Limits Pane	62
Load project	157
Log Categories	268
Log Menu.....	47
Log Pane	63, 64
Logging Control	405
Logs Overview	267

M

Making sense of the difference	163
Managing and Printing Reports.....	278
Memory Faults	253
Menus overview	40
Mini-dump Overview.....	336
Modifying a test	233
Modifying Testing Settings.....	296

Moving, detaching and hiding panels.....	39
N	
Network Corruption Faults Pane.....	65
Network Faults	254
Network Logs.....	275
Network Logs Pane.....	67
Network Message Details Pane.....	69
New Project Options	219
New Project Wizard	213
O	
Open Network Corruption Wizard	132
Other resources	19
P	
Printing Logs.....	273
Process Chaining.....	396
Process/Library Faults.....	256
Project Pane.....	71, 73
Project toolbar	37
Properties Pane.....	74
R	
Registry Faults	255
Removing a fault.....	243
Replacement Library Results	350
Replay the Recorded Session	174
Replaying a recorded session	340
Reports Overview.....	277
Resource Dependencies Summary Table	286
Resource Fault Overview	259
Resource Faults Pane	75
Resource Pane	77, 78
Resource Usage Summary Table	287
Restart BrokenApp with Resource Faults	99
Return Values Summary Table	288
Review the Definitions of the Functions	347
Review Your Choices	349
Running Holodeck in user mode	24
Running Holodeck silently.....	364
S	
Saving the Project	89
Scheduled Test Wizard	226
Second Example - Examining AUT Execution Environment	417
Select Regular Expressions as the Method of Corruption	134
Select the functions to add to the Holodeck Database.....	346
Select Which Data Should be Corrupted.....	133
Selecting Libraries and Methods.....	368
Session Menu	44
Set Insufficient Memory	92
Set the Resource Fault.....	98
Set up the project.....	167
Setting per process items.....	108

Setting up your first project.....	84
Setup the Test Harness	180
Spawning Another Process from the BrokenApp	112
Specifying the Output Options.....	369
Start a Service	217
Start Recording the Session.....	168
Step by Step for HEAT API.....	425
Stop Test Generation.....	173
Stress testing from the command-line.....	361
Summary	94, 103, 116, 154, 165, 175, 185
System Requirements.....	21

T

Test Pane.....	79, 80
Tests	223
Tests Control	408
Tests Summary Table.....	284
The Create a New Project Wizard.....	85
Third Example - Modifying the AUT Execution Environment Disk and Memory Limits	419
To find text in the log files	270
Tools Menu	48
Types of faults.....	250

U

Use the Custom Test Project to Test Notepad.....	207
Using Corrupted Files to test file processing	124
Using HEAT	424
Using Holodeck with other Automated Test Frameworks	355
Using HolodeckLib	372
Using HoloScript.....	384
Using Recorded Sessions from the command-line.....	363
Using regular expressions	325
Using replacement strings	327
Using the Network Corruption Fault Wizard	298

V

Verify Insufficient Memory	93
Verify ReadLine Fault is Set.....	152
Verify the Fault is set	138
View Logs and Create a Test for the Test dll	197
View Menu.....	49
Viewing the details of the corrupted file	125

W

What's Contained in a Report	279
What's new in this version.....	30
Windows 2000 and attaching the Holodeck Debugger	455
Windows and panels overview.....	35
Working with Multi-threaded applications	332
Working with Pivot Tables	280