### Introduction

### What it does

This tool permits a series of tests to be written addressing specific tags or entry points within a project and act to verify that the return results are as expected for that code. The significance of this is that, when run routinely any time that the project is modified, it will act to indicate whether the intended function has been modified inadvertently or whether the modification has had unexpected effects on other functionality within the project. The set of unit tests for a project should run rapidly (usually within a matter of seconds) and with minimal disruption for developers. Another function of unit tests is that they indicate what the intended software was written to do. The latter may be especially useful when new developers start working with the software or a programmer returns to a project after a prolonged period.

The concept of Unit Testing was already in place before Kent Beck created a tool that he used in the language Smalltalk, and then was turned into the tool Junit for Java by Kent Beck and Erich Gamma. This tool for running specific tests on facets of a software project was subsequently referred to as xUnit, since NUnit was developed for .NET developers, DUnit for Delphi developers, etc. MUnit is the equivalent tool for M developers to use and was originally created in 2003.

### Using M-Unit

The M-Unit functionality is contained in the %ut and %ut1 routines. The code was originally written by Joel Ivey when he was working as a developer for the Department of Veteran Affairs. The code had input as suggestions by several other developers both inside and outside of the VA, including Kevin Meldrum and especially Sam Habiel who made significant contributions to the current status. Current development is being continued for OSHERA via VistA Expertise Network.

# %ut ;VEN-SMH/JLI - PRIMARY PROGRAM FOR M-UNIT TESTING ; 08/04/14 16:13

# ;;0.1;MASH UTILITIES;

; This routine and its companion, %ut1, provide the basic functionality for

- ; running unit tests on parts of M programs either at the command line level
- ; or via the M-Unit GUI application for windows operating systems.

From a user's perspective the basic start for unit tests from the command line is the entry point EN^%ut, the first argument is the name of the routine to be tested and is required, but the tag can take up to two additional arguments: a verbose indicator and a BREAK indicator, both of these require a non-zero value to activate them.

# D EN^%ut("ROUTINE\_NAME")

or

# D EN^%ut("ROUTINE\_NAME, VERBOSE, BREAK)

The command with a single argument will result in the unit tests being run and each successful test is shown by a period ('.') followed by specification of the number of tags entered, the number of tests run, the number of failures, and the number of errors encountered. Instead of the period for successes,

failures or errors are indicated by the tag and routine name for the specific test, a description of the test if provided, and a message concerning the failure if provided or the line and routine at which the error occurred. The verbose option will result in a listing of each test that is executed, which may make it more difficult to identify problems if they have occurred. The BREAK option will result in termination of the unit test as soon as a failure or error is encountered.

The code written in a unit test routine has specific entry points that should indicate a specific set of functionality being tested. The tag may have more than one test, but these should all focus on the same aspect being tested. Originally specification of the tags and a description of the functionality being tested by the tag testing were entered following an XTENT tag in the following manner.

XTENT ;

;;TEST1;Testing functionality for one feature ;;ANEW1;Testing another piece of functionality ;;ATHIRD;Testing still something else

More recently, an alternative method was added similar to the annotation used in C#, thanks to the suggestion of Kevin Meldrum. The indicator @TEST is specified as the first string following the semicolon on the same line as the tag, and a description can then be added following this indicator.

TEST4 ; @TEST another test for different functionality

Since there will frequently be multiple routines with tests created to test a specific project, these can be indicated in a manner similar to the original description of the entry tags, following a XTROU tag. The following could be used to link additional test routines to a ZZUXQA1 test routine.

XTROU ;

;;ZZUXQA2 ;;ZZUXQA3 ;;ZZUXQA4

The other routines can also reference these as well, or additional related test routines. Each routine would be included only once, no matter how many of the other routines reference it.

A test routine can use one of three types of calls for its tests, determining truth, equivalence, or simply indicating failure for the test. In each of these a final argument can be used to specify information about the specific test result.

Truth is tested by the command

DO CHKTF^%ut(TorF,message)

where 'TorF' is a value to be tested for true (passing the test) or false (failing the test).

Equivalence is tested by the command

DO CHKEQ^%ut(expected, result, message)

where 'expected' is the value that is expected from the test, and 'result' is the value that was obtained and should be equal to 'expected' if the test is to pass.

Failure already determined is specified by the command

DO FAIL^%ut(message)

and is generally used when the processing has reached an area that it shouldn't be expected to reach given the circumstances, and 'message' then describes the situation.

The MUnit functionality is set up to capture information on errors, and to continue processing the remaining tests within the tag as well as additional tags.

There are four other tags that have meaning to the MUnit functionality - STARTUP, SETUP, TEARDOWN, and SHUTDOWN. Frequently, to provide specific data to use for testing, it may be necessary to add data which is totally temporary, either for all tests in one pass, or before each test is run.

The STARTUP tag specifies code that should be run once when the testing is starting up. At most only a single STARTUP tag should be present in a tests for a given project. Its companion is SHUTDOWN, which if present, will be run only after all of the tests have been completed. Again, there should only be a single SHUTDOWN tag in a project.

The SETUP tag specifies code that should be run before each test tag in a given routine is run, there could be similar SETUP tags in other routines as well. Its companion is TEARDOWN which, if present, will be run immediately after each test tag is processed.

It should be noted that care should be taken in using these four tags, since they may end up hiding significant functionality from testing or result in problems later if changes are made to the tests (which would then be converted into changes in the project related to the tests).

An additional tag (CHKLEAKS^%ut) is available for checking for variable leaks either as a part of a unit test, or it can be called outside of unit tests as well.

CHKLEAKS(%zuCODE,%zuLOC,%zuINPT) ; functionality to check for variable leaks on executing a section of code

- ; %zuCODE A string that specifies the code that is to be XECUTED and checked for leaks.
- ; this should be a complete piece of code
  - (e.g., "S X=\$\$NEW^XLFDT()" or "D EN^%ut(""ROUNAME"")")
- ; %zuLOC A string that is used to indicate the code tested for variable leaks

; %zuINPT - An optional variable which may be passed by reference. This may

- ; be used to pass any variable values, etc. into the code to be
- ; XECUTED. In this case, set the subscript to the variable name and the
  - value of the subscripted variable to the desired value of the subscript.
- e.g., (using NAME as my current namespace)
  - S CODE="S %zuINPT=\$\$ENTRY^ROUTINE(ZZVALUE1,ZZVALUE2)"
- S NAMELOC="ENTRY^ROUTINE leak test" (or simply "ENTRY^ROUTINE")
- S NAMEINPT("ZZVALUE1")=ZZVALUE1
- ; S NAMEINPT("ZZVALUE2")=ZZVALUE2



The GUI MUnit application provides a visually interactive rapid method for running unit tests on M code.

ĭ		M-Unit	- Testing Framev	vork (OSE	HRA)	- 🗆 ×
File	Help		_			
	Server:		Port:			
1.	MYSERVER.4	ADDRESS	9502	Sele	ect Server	Disconnect
2.	Primary Test R	outine:		Select Group	p	
3.	List		Connected			
4.	Run					Exit
	Tags:	Tests:	Errors:	Failed:	Elapsed	
	ALER DEBU	TS GGER S FOR UNIT	Look Up Utilit	y	✓ <u>D</u> K ★ <u>C</u> ancel	
		-sikuree/Errere			<u>ĵ∌i</u> Eind	
1.16	ssi mierarchy   r	direction and a second se				

Figure 1. Selection of an M-Unit test

After specifying the server address and port, the user can sign on or click the Select Group button to select a unit test from the M-UNIT TEST GROUP file (#17.9001) as shown here (Figure 1), or simply enter the name of a unit test routine in the Primary Test Routine field and click on List. This will bring up a list of the routines and tags in the unit test run (Figure 2).

ile	Help				
	Ticib				
	Server:	Port:			
1	MYSERVER.ADDRESS	9502	Select S	erver	Disconnect
2.	Selected Test Group: TESTS F	OR UNIT TEST R	Clear Group		
З.	List	Connected			
4	Bun			[	E <u>x</u> it
ч.					
	Tags: 28 Tests:	Errors:	Failed:	Elapsed	
	2				
	BADCHKEQ - CHKEQ 3 BADCHKEQ - CHKEQ 3 BADCHKTF - CHKTF sh BADERROR - throws an CALLFAIL - called FAIL LEAKSOK - check leak NVLDARG1 - check leak ISUTEST - check INUTE Sutt6 -	style test miclearor2 nould fail on un equa ould fail on false val error on purpose o test it should be ok s with leak lid arg in CHKEQ ST inside unit test	l value ue		ļ
	CHKCMDLN - check con	nmand line processir	na of %utt5		
	CHKGUI - check GUI pro	cessing of %utt5	reference of the		
	CKGUISET - check list of	tests returned by G	UISET		
	NEWSTYLE - test return	of valid new style or	@TEST indicator	8	
		-			~
	Charles and the second s				

Figure 2. List of Unit tests selected for running

Clicking the Run button will run the unit tests, resulting in a bar which is green if all tests pass or red if any failures or errors are encountered (Figure 3).



Figure 3. The unit tests run with failures

If failures or errors are encountered, clicking on the Failures/Errors tab at the bottom of the listing opens a display of specific information on the problems.

-	Help					
	Server:	Port			1	
1.	MYSERVER.ADDRE	9502	Se	lect Server	Disconnec	
2.	Selected Test Group: TESTS FOF		UNIT TEST R Clear Group			
3.	List		Connected			
4.	Run					Exit
	Tags: <b>28</b> Te	sts: <b>31</b>	Errors: 1	Failed: 7	Ela	apsed 0.023
Ţ	AG^ROUTINE	Error Type	Message			
T	AG^ROUTINE	Error Type FAILURE	Message This is an inten	tional failure.	2	
T T T	'AG^ROUTINE '5^%utt1 '5^%utt1	Error Type FAILURE FAILURE	Message This is an inten Intentionally thre	iional failure. owing a failu	re	
T T T E	AG^ROUTINE 15^%utt1 15^%utt1 BADCHKEQ^%utt5	Error Type FAILURE FAILURE FAILURE	Message This is an inten Intentionally thr <4> vs <3> - SE	ional failure. owing a failu T UNEQUA	re L ON PUR	POSE - SHOULD F/
T T E E	AG^ROUTINE 15^%utt1 15^%utt1 DADCHKEQ^%utt5 DADCHKTF^%utt5	Error Type FAILURE FAILURE FAILURE FAILURE	Message This is an intent Intentionally thr <4> vs <3> - SE SET FALSE (0)	ional failure. owing a failu T UNEQUA ON PURPC	re L ON PUR ISE - SHOL	POSE - SHOULD F/ JLD FAIL
T T E E	AG^ROUTINE 55%utt1 55%utt1 ADCHKEQ^%utt5 ADCHKTF^%utt5 ADERROR^%utt5	Error Type FAILURE FAILURE FAILURE FAILURE ERROR	Message This is an intent Intentionally thr <4> vs <3> - SE SET FALSE (0) <undefined></undefined>	ional failure. owing a failu T UNEQUA ON PURPC BADERROI	re L ON PUR ISE - SHOL R+6^%utt5	POSE - SHOULD F/ JLD FAIL *Q
T T E E C	AG^ROUTINE 57%utt1 57%utt1 ADCHKEQ7%utt5 ADCHKTF7%utt5 ADERROR^%utt5 CALLFAIL7%utt5	Error Type FAILURE FAILURE FAILURE FAILURE ERROR FAILURE	Message This is an intent Intentionally three (4> vs <3> - SE SET FALSE (0) (UNDEFINED> Called FAIL to t	iional failure. owing a failu ET UNEQUA ON PURPC BADERROI est it	re IL ON PUR ISE - SHOL R+6^%utt5	POSE - SHOULD F/ JLD FAIL *Q
T T E E C	AG^ROUTINE 57%utt1 57%utt1 ADCHKEQ^%utt5 ADCHKTF^%utt5 ADDERROR^%utt5 CALLFAIL^%utt5 EAKSBAD^%utt5	Error Type FAILURE FAILURE FAILURE FAILURE ERROR FAILURE FAILURE	Message This is an intent Intentionally three (4> vs <3> - SE SET FALSE (0) (UNDEFINED> Called FAIL to t LEAKSBAD TE	ional failure. owing a failu ET UNEQUA ON PURPC BADERROI est it ST - X NOT	re IL ON PUR ISE - SHOL SPECIFIEI	ipose - Should F; Jld Fail *Q D Variable Leak:
T T E E C L	AG'ROUTINE 55%utt1 55%utt1 ADCHKEQ^%utt5 ADCHKTF%utt5 ADCHROR^%utt5 ZALFAIL%utt5 EAKSBAD^%utt5 IVLDARG1%utt5	Error Type FAILURE FAILURE FAILURE FAILURE ERROR FAILURE FAILURE FAILURE	Message This is an inten Intentionally thu <4> vs <3> SE SET FALSE (0) <undefined> Called FALL to t LEAKSBAD TE NO VALUES IN</undefined>	tional failure, pwing a failu T UNEQUA ON PURPC BADERROI est it ST -X NOT IPUT TO CH	re NL ON PUR ISE - SHOL R+6^%utt5 SPECIFIEI IKEQ^%ut	IPOSE - SHOULD F/ JLD FAIL "Q D VARIABLE LEAK: - no evaluation poss
T T E E C L	AG^ROUTINE 57%utt1 57%utt1 MADCHKEQ^%utt5 MADERROR^%utt5 MADERROR^%utt5 ZALFAIL <sup>®</sup> xutt5 ZALFAIL <sup>®</sup> xutt5 ZALFAIL <sup>®</sup> xutt5 WLDARG1 <sup>®</sup> %utt5	Error Type FAILURE FAILURE FAILURE FAILURE FAILURE FAILURE FAILURE FAILURE	Message This is an intent Intentionally thr (4> vs (3> - St SET FALSE (0) (UNDEFINED> Called FAIL to t LEAKSBAD TE NO VALUES IN	tional failure. owing a failu ET UNEQUA ON PURPO BADERROI est it ST -XNOT IPUT TO CH	re LE ON PUR ISE - SHOL R+6^%utt5 SPECIFIEI IKEQ^%ut	POSE - SHOULD F/ JLD FAIL *Q D VARIABLE LEAK: - no evaluation poss
T T E E C L N	AG^ROUTINE 57%ut1 57%ut1 ADCHKEQ^%ut5 ADCHKTF%ut5 ADCHKTF%ut5 ZALLFAIL^%ut5 ZALLFAIL^%ut5 ZALLFAIL^%ut5 IVLDARG1%ut5	Error Type FAILURE FAILURE FAILURE FAILURE ERROR FAILURE FAILURE FAILURE	Message This is an intent Intentionally thr <4> vs <3> - SE SET FALSE (0) <undefined> Called FAIL to t LEAKSBAD TE NO VALUES IN</undefined>	ional failure. owing a failu IT UNEQUA ON PURPC BADERROI BADERROI est it ST - X NOT IPUT TO CH	re IL ON PUR ISE - SHOL R+6^%utt5 SPECIFIE[ IKEQ^%ut	POSE - SHOULD F/ JLD FAIL "Q VARIABLE LEAK: - no evaluation poss

Figure 4. Specifics on failed tests or errors

In the case shown (Figure 4), all of the failures are intentional. Usually, failures and/or errors are not intentional and the user can then edit the routine, and save the changes, then simply click on the Run button again to see the effect of the changes.

To select a new unit test, the user would click on the Clear Group button, then again either select another group or as shown in Figure 5, entering the name of a unit test routine (ZZUXQA1 and related routines are not included with the M-Unit Test code and is shown only as an example) and clicking on the List button.

ø		M-Unit - Te	esting Framev	vork (OSEHRA	()	- 🗆 🗙
<u>F</u> ile	<u>H</u> elp					
	Server:		Port			
1.	MYSERVER.AD	DRESS	9502	Select Se	rver	Disconnect
2.	Primary Test Rou	tine ZZUXQA1		Select Group		
3.	List		Connected			
4.	Run					Exit
	Tags:	Tests:	Errors:	Failed:	Elapsed	

Figure 5. Specification of unit tests by routine name

Again, clicking the Run button will run the unit tests (Figure 6). This figure shows the desired result, a green bar meaning that all tests passed.

ø	M-Unit - Testing Framework (OSEHRA) 🛛 🗕 🗖 🗙
<u>F</u> ile	Help         Port:           MYSERVER.ADDRESS         9502         Select Server         Disconnect
2.	Primary Test Routine ZZUXQA1 Select Group
3.	List Connected
4.	Run
	Tags: 21 Tests: 127 Errors: 0 Failed: 0 Elapsed 7.213
T	SENDSURD - send alert to surrogate     ZUXQA4     AUDXREF1 - Check AUD cross-ref for simple send alert     SET3DEL1 - Setup 3 surrogates, remove middle one     CHNGSURD - Change surrogates between alerts     SENDREMV - Send to surrogate and then remove     SENDRMV2 - Send to user and surrogate then remove     SENDRMV2 - Send to user and surrogate then remove     SENDRMV2 - Send to user and surrogate then remove     SENDRMV2 - Send to user and surrogate then remove     SENDRMV2 - Send to user and surrogate then remove     SENDRMV2 - Send to user and surrogate then remove     SENDRMV2 - Send to user and surrogate then remove     SENDRMV2 - Send to user and surrogate then remove     SENDRMV2 - Send to user and surrogate then remove     SENDRMV2 - Send to user and surrogate(s) in date range     ZUXQA5 -     MULTEDIT - Create 2 suros, delete first, then reset first     DCYCLIC - Check for cyclic relationships with specific dates     MANEDIT - handle expired manual or FM edit on top zero node     MANUST - list generated after manual of FM edit     CYCLIC - problem with sequential or alternating surrogates     v est Hierarchy     Failures/Errors

Figure 6. Result from the second group of unit tests

The results of both of these groups of tests (%utt1 and ZZUXQA1 and their related routines) run at the command line using are shown in Figure 7.

Cache TRM:73200 (CACHEWEB)	- 0 ×
Eile Edit Help VISTA>D EN^%ut("%utt1")	^
İ5∽Xutt1 - Error count check - This is an intentional failure.	
İ5^%utt1 - Error count check - Intentionally throwing a failure	
BADCHKEQ^%utt5 - CHKEQ should fail on unequal value - <4> vs <3> - SET ON PURPOSE - SHOULD FAIL	r unequal
BADCHKTF^%utt5 - CHKTF should fail on false value - SET FALSE (0) ON F SHOULD FAIL	PURPOSE -
BADERROR^%utt5 - throws an error on purpose - Error: <undefined>BADERF 5 *Q</undefined>	Rr+6^%utt
CALLFAIL^%utt5 - called FAIL to test it - Called FAIL to test it	
LEAKSBAD^%utt5 - check leaks with leak - LEAKSBAD TEST - X NOT SPECIFIE E LEAK: X	ED VARIABL
NVLDARG1^%utt5 - check invalid arg in CHKEQ - NO VALUES INPUT TO CHKEQ^ evaluation possible 	`%ut - no
Ran 5 Routines, 26 Entry Tags Checked 29 tests, with 7 failures and encountered 1 error. VISTA>	~
VISTA>D EN^%ut("ZZUXQA1")	
Ran 5 Routines, 21 Entry Tags Checked 127 tests, with 0 failures and encountered 0 errors. VISTA>	~

Figure 7. Command line unit tests for %utt1

The results of the single %utt1 unit test routine (and its related routines) run with the VERBOSE option, that some people prefer, specified permits the individual tests and their results to be seen, but makes the results more difficult to interpret (Figure 8).

	Cache TRM:73200 (CACHEWEB)	- 🗆 🗙
<u>Eile Edit H</u> elp VISTA>D EN^	%ut("%utt1",1)	^
T1 Make T2 Make T3 Make T4 - Entry T5 - Error T5^%utt1 -	sure Start-up Ran sure Set-up runs sure Teardown runs point using XTMENT count check Error count check - This is an intentional failure.	[ 0K ] [ 0K ] [ 0K ] [ 0K ]
T5^%utt1 -	Error count check - Intentionally throwing a failure	IFOT 1
T6 - Succee T7 - Make s T8 - If IO pal device. T11 - An @T T12 - An XT ts	d Entry Point ure we write to principal even though we are on another devic starts with another device, write to that device as if it's t EST Entry point in Another Routine invoked through XTROU offs ENT offset entry point in Another Routine invoked through XTR	IFHILI [OK] e[OK] he prici [OK] ets.[OK] OU offse [OK]
MAIN Te NEWSTYLE - OLDSTYLE - OLDSTYLI - BADCHKEQ - BADCHKEQ ^%u ON PURPOSE	st coverage calculations	LOK] [OK] [OK] [OK] UNEQUAL
BADCHKTF -	CHKTF should fail on false value	IFHILI V
BADCHKTF^%u	tt5 – CHKTF should fail on false value – SET FALSE (0) ON PU	RPOSE - ^
SHUULD FHIL		[FAIL]
BADERROR - BADERROR <sup>*</sup> %u <sup>-</sup> 5 ×0	throws an error on purpose tt5 - throws an error on purpose - Error: <undefined>BADERRO</undefined>	R+6^%utt
		[FAIL]
CALLFAIL <sup>~</sup> %u	tt5 - called FAIL to test it - Called FAIL to test it	[FAIL]
LEAKSBAD - ( LEAKSBAD^%u E LEAK: X	tt5 - check leaks with leak - LEAKSBAD TEST - X NOT SPECIFIED	VARIABL
NVLDARG1 - d NVLDARG1^%u evaluation p	check invalid arg in CHKEQ tt5 - check invalid arg in CHKEQ - NO VALUES INPUT TO CHKEQ^% possible	[FAIL] ut - no
ISUTEST - cl CHKCMDLN - c CHKGUI - ch CKGUISET - c NEWSTYLE -	heck ISUTEST inside unit test check command line processing of %utt5 eck GUI processing of %utt5 check list of tests returned by GUISET	LFAILJ [OK] [OK] [OK] [OK]
Ran 5 Routi Checked 29	nes, 26 Entry Tags tests, with 7 failures and encountered 1 error.	

Figure 8. Command line unit tests for %utt1 with VERBOSE option

On-going/Future plans for M-Unit functionality:

As a unique program in the realm of M[UMPS] code testing but following in the footsteps of other well established unit test frameworks, the M-Unit software will continue to move forward and improve (as the @TEST indicator was added based on changes in NUnit and JUnit). M-Unit will likely branch out and expand the types of checks that are available, matching the functions of other established test beds.

#### Summary

M-Unit provides a tool which can assist in writing and modifying routines in M projects with an aim to minimizing flaws in development and in the ongoing life of the software.