Visual COBOL チュートリアル

COBOL 開発: Eclipse - ネイティブ COBOL の単体テスト

1. 目的

本チュートリアルでは、ネイティブ COBOL プログラムに対するテスト作成、実行方法、および、テスト結果を表示させる方法の習得を目的と しています。

MFUnit は、Visual COBOL に搭載された xUnit 系の単体テストフレームワークです。xUnit はオブジェクト指向型の単体テストフレーム ワーク SUnit に起源を持つ JUnit や RUnit 等の単体テストフレームワークの総称です。MFUnit は xUnit の設計アーキテクチャーや 仕組みは取り入れつつも COBOL 開発者にとって扱いやすい手続き型の COBOL を対象とした単体テストフレームワークという設計思想 の下、開発されました。

MFUnit は COBOL 開発作業に以下の利点を提供します。

- テストを繰返し実行させることができるため、修正作業時などのテスト工数の削減が見込める
- Jenkins などの継続的インテグレーション (Continuous Integration) ツールと連携によりテストの自動化が行え、DevOps サイクルの導入が足がかりを作れる

2. 前提

- 本チュートリアルで使用したマシン OS : Windows Server 2019 Standard Edition
- Visual COBOL 9.0 for Eclipse Patch Update 3 がインストール済みであること

本資料は、ネイティブ COBOL に対する単体テストフレームワークの利用方法を記載したチュートリアルです。JVM COBOL の単体テ スト実現方法については、別チュートリアルを参照ください。

下記のリンクから事前にチュートリアル用のサンプルファイルをダウンロードして、任意のフォルダに解凍しておいてください。

サンプルプログラムのダウンロード

内容

- 1. 目的
- 2. 前提
- 3. チュートリアル手順の概要
- 3.1. IDE からの実行
 - 3.1.1. 前準備
 - 3.1.2. 基本的なテスト
 - 3.1.3. データ駆動型テスト
 - 3.1.4. 自己完結型テスト
- 3.2. コマンドラインからの実行

3. チュートリアル手順の概要

MFUnit は、単一処理の結果を判定する基本的なテストプログラムからデータ駆動型、自己完結型といった複数のテストタイプを用意しており、順に説明していきます。特定のテストタイプのみを実施したい場合においても、「3.1.1 前準備」は先に実施してください。

3.1. IDE からの実行

3.1.1. 前準備

3.1.1.1. Eclipse の起動

- 1) スタートメニューより、Visual COBOL for Eclipse を起動します。
- 2) ワークスペースを指定し、[起動(L)] ボタンをクリックします。

Eclipse ランチャー ディレクトリーをワークスペースとして選択 Eclipse は、ワークスペースディレクトリを使用して、環境設定と開発成果物を保存します。 ワークスペース(W): C*workspace_tut_native_mfunit 参照(B)... この選択をデフォルトとして使用し、今後この質問を表示しない(U) 擬近のワークスペース(R) 起動(L) キャンセル

3.1.1.2. **チュートリアルプロジェクトのインポート**

1) Eclipse IDE メニューより、[ファイル(F)] > [インポート(I)] を選択してください。



2) [一般] > [既存プロジェクトをワークスペースへ] を選択し、[次へ(N) >] ボタンをクリックします。

インポート	_		×
選択 アーカイブ・ファイルまたはディレクトリーから新規プロジェクトを作成します。		R	1
A with the state of the state o			
インホート・ワイサートの選択(<u>S</u>):			
フィルタ入力			
▶ → 一般			^
🔁 ファイル・システム			
<u>つォルダーまたはアーカイブ由来のプロ</u> ジェクト			
😂 既存プロジェクトをワークスペースへ			
> 🧁 AWS			
> 🥦 EJB			
> 🧁 Git			
> 🧁 Gradle			
> 🥭 J2EE			
> 🧀 Maven			
> Contro Focus			
> C Micro Focus 1 29- JI1 XY9/1-			~
L Comph			
? < 戻る(B) 次へ(N) > 終了	(<u>F</u>)	キャンセ	JV

3) 「ルート・ディレクトリーの選択(T)」欄に、チュートリアルプロジェクトへのパスを指定します。 [参照(R)] ボタンを押してサンプ ルファイルを展開したフォルダ内の AirportDemoMFUnit フォルダを指定してください。

インポート	_		×
プロジェクトのインポート 既存の Eclipse ブロジェクトを検索するディレクトリーを選択します。			
 ・ディレクトリーの選択(I): アーカイブ・ファイルの選択(<u>A</u>): 	~	参照(<u>R</u> 参照(<u>R</u>)
プロジェクト(<u>P</u>):			
	3	Fべて選択(<u>S</u>)
	選択	をすべて解	除(<u>D</u>)
		更新(<u>E</u>)	

4) 「プロジェクトをワークスペースにコピー(C)」項目にチェックを入れた上で、[終了(F)] ボタンをクリックします。

プロジェクトのインボート 既存の Eclipse プロジェクトを検索するディレクトリーを選択します。	
 ●ルート・ディレクトリーの選択(T): C:¥vc-elnativetestframework¥AirportD 	→ 参照(R)
○ アーカイブ・ファイルの選択(A):	~ 参照(R)
プロジェクト(P):	
AirportDemoTutNativeMFUnit(C:¥vc-elnativetestframework¥Ai	すべて選択(S)
	選択をすべて解除(D)
	更新(E)
< >>	
スフジョフ □ ネストしたプロジェクトを検索(H) ☑ プロジェクトをワークスペースにコピー(C) □ Close newly imported projects upon completion □ ワークスペースに既に存在するプロジェクトを隠す(i)	
ワーキング・セット	
□ ワーキング・セットにプロジェクトを追加(T)	新規(W)
ワーキング・セット(0):	∠ 選択(E)
(N) > 終了(F)	キャンセル

AirportDemoTutNativeMFUnit プロジェクトが作成されることを確認します。



補足)

COBOL 開発を行うためには、COBOL パースペクティブという画面レイアウトを使用します。異なるパースペクティブを開い ている場合、Eclipse IDE メニューの [ウィンドウ(W)] > [パースペクティブ(R)] > [パースペクティブを開く(O)] > [そ の他(O)] をクリックした上で、COBOL をクリックすることで、COBOL パースペクティブを開くことができます。 5) 単体テストを行なうために、出力形式を int 形式に変更します。以下の手順を実行してください。
 AirportDemoTutNativeMFUnit プロジェクト名を選択した状態で、マウスの右クリックにてコンテクストメニューを表示し、 [プロパティ(R)]を選択します。

8	更新(F) プロジェクトを閉じる(S) Close Unrelated Project	F5
	リモートシステムビューで表示 検証(V)	
Q_	Coverage As	>
0	実行(R)	>
*	デバッグ(D)	>
	プロファイル(P)	>
	ローカル履歴から復元(Y)	
	チーム(E)	>
	比較対象(A)	>
	構成	>
	ソース(S)	>
	プロパティ(R)	Alt+Enter

ッリーメニューより、[Micro Focus] > [ビルド構成] > [リンク] を選択し、「ターゲットの種類」を"すべて INT/GNT ファ イル"に変更した上で、[適用して閉じる] ボタンをクリックします。

🧿 プロパティ: AirportDemoTutNa	tiveMFUnit		
	リンク	¢	▼ <> ▼ 8
> リソース Coverage V Micro Focus	New Configuration [使用中]	4	■成の管理
ビルドパス ビルド構成 > COBOI	フィルタテキストを入力		
イベント ディプロイ	設定 V Linkage	値	^
ビルド理査	出力の名前	AirportDemo	
	出力パス	New Configuration.bin	
プロジェクト設定	エントリポイント		
シュノーシュノーシュと	ターゲットの種類	すべて INT/GNT ファイル	
指令の確定。また時構成	ビット数	64 ピット	
> 夫1J时悔成 Designet Frances	.LBR にパッケージ化	いいえ	
Project Facets	サービスを COBOL アーカイブ (.car) ファイル	レにパッケージ化 いいえ	~
iaax iags > Validation WikiText タスス・リポジトリー ビルター プロジェクト・ネーチャー	ターゲットの種類 作成する出力ファイルの種類、および単一ファイル・ 定します(必須) デフォルト値:単一実行可能ファイル	を作成するか、ソースごとに1個のファイルを作成するか	を指 ~
プロジェクト参照 実行/デバッグ設定		デフォルトの復元(T)	適用(L)
?		適用して閉じる	キャンセル

注意)

上記画面では、「ビット数」に "64 bit" を指定しています。 "32 bit" 指定も可能ですが、その場合、以降の手順でも "32 bit" を選択していただく必要があります。

3.1.2. 基本的なテスト

この方式では、1つのテストを1つのテストブロックで記述していきます。

3.1.2.1. **MFUnit テストの作成**

1) Eclipse IDE メニューより、[ファイル(F)] > [新規(N)] > [COBOL ユニットテストプロジェクト] を選択します。

777 777	tut_native_mfunit - Eclipse <mark>ル(F)</mark> 編集(E) リファクタリング ナビゲート 新規(N) ファイルを聞く(.) ファイル・システムからプロジェクトを開く 最近のファイル	-(N) 検索 プロ: Alt+シフト+N > >		COBOL プロジェクト COBOL コピーファイル プロジェクト リモート COBOL プロジェクト リモート COBOL コピーファイル プロジェクト COBOL ユニット テスト プロジェクト COBOL JVM プロジェクト
n	閉じる(C) すべて閉じる(L)	Ctrl+W Ctrl+シフト+W	않	リモート COBOL ユニット テスト プロジェクト COBOL JVM ユニット テスト プロジェクト
	保存(S) 別名保存(A)	Ctrl+S	₩ 2	リモート COBOL JVM プロジェクト プロジェクト(R)

 プロジェクト名」に "AirportDemoTutNativeMFUnitTest" を入力し、実行環境に合わせたプロジェクトテンプレートを 選択した上で、[終了(F)] ボタンをクリックします。

🔵 COBOL ユニット テスト プロジェクトの新規作成	_		×
COBOL ユニット テスト プロジェクト ワークスペースに COBOL ユニット テスト プロジェクトを新規作成します。		k	
プロジェクト名(₽): AirportDemoTutNativeMFUnitTest プロジェクト テンプレートを選択 G Micro Focus テンプレート [32 ピット] G Micro Focus テンプレート [64 ピット]			
	テンプレート	の設定を構	載
□ テンプレートの参照			
場所:		参照	
ファイルシステムを選択: default \vee			
☑ デフォルト・ロケーションの使用(<u>D</u>)			
ロケーション(L): C¥workspace_tut_native_mfunit¥AirportDemoTut	NativeN	参照(<u>R</u>)	
ファイル・システムを選択(<u>Y</u>): <mark>デフォルト</mark> ~			
?	了(<u>F</u>)	キャンセ	!JV
()			

注意)

先行作業にて指定したビット数と同じテンプレートを選択してください。

AirportDemoTutNativeMFUnitTest プロジェクトが作成されていることを確認してください。



3) AirportDemoTutNativeMFUnitTest プロジェクトを選択した上で、Eclipse IDE メニューより、[ファイル(F)] > [新規

(N)] > [COBOL ユニットテスト] を選択します。

ファイ	「ル(F) 編集(E) リファクタリング ナビゲート	(N) 検索 プロシ	ェクト(F	P) 実行(R) ウィンドウ(W) ヘルプ(H)
	新規(N)	Alt+シフト+N >	😫	COBOL プロジェクト
	ファイルを開く(.)		📸	COBOL コピーファイル プロジェクト
	ファイル・システムからプロジェクトを開く		Ê	リモート COBOL プロジェクト
	最近のファイル	>	ŧġ	リモート COBOL コピーファイル プロジェクト
	閉じる(C)	Ctrl+W	Đ	COBOL ユニット テスト プロジェクト
	すべて閉じる(L)	Ctrl+シフト+W	썉	COBOL/Java 相互運用機能のプロジェクト
	(R 7= (C)	Challes	2	COBOL JVM プロジェクト
		Ctri+5	R	リモート COBOL ユニット テスト プロジェクト
	有前を刊りて本行(A)… オバア保存(E)	Ctol 1 Start 1 S	jet EU	COBOL JVM ユニット テスト プロジェクト
42		Cur+97P+3	e	リモート COBOL JVM プロジェクト
	前回休住した1人服に決9(1)			プロジェクト(R)
	移動(V)		(1)	フォルダー
	名前を変更(M)	F2	Ľ	ファイル
8	更新(F)	F5	đ	COBOL プログラム
	行区切り文字の変換(D)	>	B	COBOL コピーファイル
۵	印刷(P)	Ctrl+P	P	スタンドアロン ファイル
r - n	インポート(I)		Ľ	リモート スタンドアロン ファイル
4	エクスポート(O)		٢	Enterprise Server Common Web Administration 接続
	-7⊓ I(= ∠(P)	Alt. Enter	<u></u>	Directory Server 接続
) [//)] (h)	Ait+Enter	1	Enterprise Server
	ワークスペースの切り替え(W)	>	2	COBOL ユニットテスト
	再開		G	COBOL JVM クラス COBOL ユニット テストを作成します

4) [COBOL ユニット テストの新規作成] ウィンドウが表示されるので[プログラム ユニット テスト] 項目を選択し、[次へ(N)] ボタンをクリックします。



5) 「プログラムのユニットテストを作成する」項目を選択し、[参照] ボタンをクリックします。

ocobol ユニット デ	ストの新規作成		×
COBOL ユニットラ 😵 テストする有効なソ	-スト -スファイルを指定する必要があります	C	Ů
含まれるブロジェクト: 新規ファイル名:	AirportDemoTutNativeMFUnitTest TestProgram1.cbl	 , the	♥照
 プログラムのユニット テスト対象のプログラ 	、テストを作成する ム:	 参	照

6) AirportDemoTutNativeMFUnitTest プロジェクト内の「aircode.cbl」を選択した上で、[OK] ボタンをクリックします。

💿 ソースファイル	_		\times
ソースファイルを選択します			
 ✓ AirportDemoTutNativeMFUnit > ▷ .settings > ▷ New_Configuration.bin ○ aircode.cbl ○ main.cbl 			
	OK	キャンカ	L
		7770	V

7) テスト対象のプログラムに、さきほど選択した aircode.cbl が表示されていることを確認して、[終了(F)] ボタンをクリックしま す。

SCOBOL ユニット :	テストの新規作成				
COBOL ユニット ラ エディタで開くことがて	テスト きる COBOL ユニット テスト ファイルる	を新規作成します。			U
含まれるプロジェクト: 新規ファイル名:	AirportDemoTutNativeMFUnitT TestProgram1.cbl	Test			参照
● プログラムのユニッ テスト対象のプログラ	トテストを作成する j/」 AirportDemoTutNativeMFU	nit/aircode.cbl			参照
○ テンプレートからユ テンプレートを選択:	ニット テストを作成する :us テンプレート				
□ テンプレートの	参照			<u></u>	トの設定を構成
5時77: ファイルシフ	テムを選択: default 〜				容照 aa
?		< 戻る(B)	次へ(N) >	終了(F)	キャンセル
単体テストプロク	ブラムが作成されたことをで	確認します。			



8) テストプログラムを確認します。

AirportDemoTutNativeMFUnitTest プロジェクト内の「TestProgram1.cbl」をダブルクリックして、コードを表示します。

🖻 TestPro	gram1.cbl 🕱
🛛 т 🖻	estProgram1.cbl 🕨
	•••*A·1·B·••···2···••3···••
	01 i pic 99.
Θ	procedure division.
Θ	goback returning 0
	entry MFU-TC-PREFIX & TEST-TESTAIRCODE.
	call "AIRCODE" using
	by value lnk-function
	by value lnk-airport1
	by value lnk-airport2
	by value lnk-prefix-text
	by reference ink-rec
	by reference ink-distance-result by reference ink-matched-codes-array
	*> Verify the outputs here
	goback returning MFU-PASS-RETURN-CODE
	•
Θ	<pression configuration<="" pre="" testcase=""></pression>
	entry MFU-TC-SETUP-PREFIX & TEST-TESTAIRCODE.
	perform InitializeLinkageData
	*> Add any other test setup code here
	goback returning 0
	•

以下のコードが記述されていることが分かります。

- entry MFU-TC-PREFIX & TEST-TESTAIRCODE
- entry MFU-TC-SETUP-PREFIX & TEST-TESTAIRCODE

MFUnit では、テストを下記のように決められた手順で実行しています。

テスト名 test1 を実行する場合、以下の順序で動作します。

- ① entry MFU-TC-SETUP-PREFIX & "test1"
- ② entry MFU-TC-PREFIX & "test1"
- ③ entry MFU-TC-TEARDOWN-PREFIX & "test1"
- ④ 次のテストを実行・・・

MFU-TC-SETUP-PREFIX で始まる entry にて、テストの前処理を定義できます。前処理の代表例としては、ファイルを あらかじめオープンしておくなどが考えられます。一方、MFU-TC-TEARDOWN-PREFIX で始まる entry では、テスト実行 後の処理を定義できます。前処理でオープンしたファイルをクローズするような処理が該当します。前処理、後処理ともに省 略可能です。

自動生成されるテストプログラムはテンプレートであり、実際には、上記ルールに従い、テストを記述する必要があります。

9) 新規のテストケース(羽田・ロンドンヒースロー空間間の距離(km)のテスト)を追加した上で、実行を行ないます。 サンプルファイルを展開したフォルダ内の Basic¥TestProgram1.cblの内容で、TestProgram1.cblを上書きしてください。これは、テストケース "testDistance"を途中まで作成しています。前述した MFU-TC-SETUP-PREFIX, MFU-TC-PREFIX, MFU-TC-TEARDOWN-PREFIXの3 entryが追加されていますが、肝心な結果判定を記述していません。

結果判定処理を実装するため、82 行目に、以下のコードを追加します。

	if distance-km = wk-distance-km
	then
	goback returning MFU-PASS-RETURN-CODE
	else
	string "expected " wk-distance-km ", but " distance-km z"" into err-msg end-
string	
	call MFU-ASSERT-FAIL-Z using err-msg
	end-if.

期待値である wk-distance-km (9591) と一致しているかを判定した上で、成功・失敗を戻します。

参考)

```
テスト失敗時の記述方法として、成功時同様に、戻り値で返す方法もあります。その場合は、以下のような例になります。
if distance-km = wk-distance-km
then
goback returning MFU-PASS-RETURN-CODE
else
string "expected " wk-distance-km ", but " distance-km into err-msg end-string
display err-msg
goback returning MFU-FAIL-RETURN-CODE
end-if.
反り値 MFU-FAIL-RETURN-CODE を利用する場合、テスト結果を確認するためにエラー情報を、display などで出
力する必要があります。
```

3.1.2.2. MFUnit テストの実行

本テスト対象のプログラムは、環境変数で設定された空港情報が保存されたデータファイルを参照するため、手順内で設定を行います。

 「TestProgram1.cbl」を選択した状態で、マウスの右クリックでコンテクストメニューを表示し、[実行(R)] > [実行の構成 (N)]をクリックします。

0	実行(R)	>	CBL	1 COBOL アプリケーション	Alt+シフト+X,N
脊	デパッグ(D)	>	œ	2 COBOL ユニット テスト	Alt+シフト+X,F
	プロファイル(P)	>		実行の構成(N)	
	検証(V)	L	-	[mfurun.exe	Finished: SUC

2) 画面左側より「COBOL ユニットテスト」を選択した状態で、マウスの右クリックにてコンテクストメニューを表示し、[新規構成 (W)] を選択します。

💿 実行構成

構成の作成、管理、および実行

📑 🖻 🍫 🗎 🗶 🖻	7 -		このダイアログから	5起動設定を構成します:
7ብሥል እ			📑 - 選択した利	重類の構成を作成するには、「新規構成
COBOL アプリケーション	,	^	🖻 - 選択した利	重類の起動構成プロトタイプを作成する
COBOL ユニット テスト	-4	新規構成	t(W)	クスポートするには、「エクスポー
Debug Adapter Laur		シジョンロ		
Eclipse アノリケーション	P	オリカモノロ	(F)	ヒーするには、「複製」ホタンを担
Eclipse 7-9-9-1	\$0	エクスホー	ト(X)	除するには、「削除」ボタンを押

3) 「名前」に "AirtportDemoTutNativeMFUnitTest" を入力し、「環境」 タブを選択した後、[追加(A)] ボタンをクリッ クします。



4) 以下の情報を入力し、[OK] ボタンをクリックします。

変数: "dd_airports"

值: "..¥..¥AirportDemoTutNativeMFUnit¥airports.dat"

◎ 変響 環境:	^{炎を追加} 変数を追加または変更します		Ď
変数: <u>値</u> :	dd_airports ¥¥AirportDemoTutNativeMFUnit¥airports.dat		
?		ОК	キャンセル

5) さきほど追加した dd_airports 環境変数が表示されていることを確認して、[実行(R)] ボタンをクリックします。
 ● 素行構成 - □ ×

構成の作成、管理、および	実行		
	名前(<u>N</u>): AirportDemoTutNativeMf	FUnitTest	
7ብሥል እ	🕟 一般 🧤 ソース 🚾 環境 🔳] 共通(Q) 🔊 実行時) 🦅 デバッグシンボル) 👌 動的:	分析) 60 CTF)
Apache Tomcat AspectJ Load-Time Weavi	(注:ここで定義された変数は、任意の 環境スクリプト内の設定値を上書きし	現在の設定値、または任意の指定された ます。)	
ASPECT/Java Application COBOL JVM アプリケーション	変数	值	追加(<u>A</u>)
COBOL JVM ユニット テスト	dd_airports	¥¥AirportDemoTutNativeMFUnit¥ai	編集(F)
COBOL JVM リモート アプリク			小田 戸下 (上/…
			削除(<u>R</u>)
☞ 新規構成			
● Eclipse アプリケーション			
■ Eclipseデータ・ツール			
Gradie Project			
G Gulp	実行する環境スクリフト:		
HTTP プレビュ−	场///:		参照
∃ J2EE ブレビュー	ファイルがプロジェクト内に	ある場合、絶対バスは相対バスになります。	
J Java アプリッーション W Java アプレット	パラメータ:		
Ju JUnit			
🚏 JUnit プラグイン・テスト	│ □ 関連付けられたプロジェクトのビルド	環境から値を継承	
m2 Maven Build			
30 項目のうち 28 項目がフィルターに-		前回保管した状態に戻す(⊻) 適用(<u>Y</u>)
?		実行(<u>R</u>)	閉じる

6) [Micro Focus Unit Testing] ビューが自動的に表示され、2つのテストケースが緑色で表示されています。緑色は、テストが正常に終了したことを表しています。

📃 コンソール 🖹 問題 🧔 タスク 🔲 プロパティー	Table Results 🕢 Filter Definitions 🔄 Mic	ro Focus Unit Testing 🙁 📴 コードカバレッジ	
実行: 2/2	⊠ Iラ-: 0	◎ 失敗: 0	
AirportDemoTutNativeMFUnitTest (16 ms) DestProgram1.cbl MFUT_TESTAIRCODE (0 ms) MFUT_TESTAIRCADE (16 ms)	5)		7スト結果

 エラーケースを確認します。「TestProgram1.cbl」をエラーとなるように修正した上で、ツールバーより、[実行] アイコンの 矢印をクリックし、「AirportDemoTutNativeMFUnitTest」をクリックします。

なお、本例では、3.1.2 で作成したテストプログラム内に記載されていた期待値 9591 を 9592 に修正しています。

o TestProgram1.cbl ⊠	
TestProgram1.cbl	
····•*A·1·B··•···2····•3····•	4
entry MFU-TC-PREFIX & "testDist set get-distance to true move "HND" to lnk-airport1 move "LHR" to lnk-airport2 call "AIRCODE" using by value lnk-airg by velerence ln by reference ln by reference ln	ance" unction port1 irport2 refix-text ik-rec uk-distance-result uk-matched-codes-array
<pre>move 9592 to wk-distance-km display wk-distance-km " / if distance-km = wk-distance then goback returning MFU-P/ else string "expected " wk-c call MFU-ASSERT-FAIL-Z end-if.</pre>	" distance-km :e-km ASS-RETURN-CODE distance-km ", but " distance-km into err-msg end-string using err-msg

プロジェクト(P) 実行(R) ウィンドウ(W) ヘルプ(H)

= 17 式 🖂	苓	•	0 -	🌯 र 🍃 🗁 🛷 र 🤮 र 👬 र	*> <				
サーバー エクス		E	9	1 AirportDemoTutNativeMFUnitTest					
✓ □ ♀ ↓ ^a z	₽ @	1		実行(R)	>				
				実行の構成(N)					
				お気に入りの編成(V)					
			_	move 95921 to wk	-dist				

MFUT_TESTDISTANCE のテストで、エラーが発生したことが一覧から判断できます。

💷 コンソール 🖹 問題 🎾 タスク 🔲 プロパティー	Table Results 🥡 Filter Definitions	😨 Micro Focus Unit Testing 🙁 📴 コード カバレッジ	
実行: 2/2	⊠ Iラ-: 0	◎ 失敗: 1	
✓ ₩ AirportDemoTutNativeMFUnitTest (1 ms)			テスト結果
✓ ☑ TestProgram1.cbl			
HEIMFUT_TESTAIRCODE (0 ms)			

3.1.3. データ駆動型テスト

この方式では、複数のテストデータをデータファイルに設定しておくことで、データによって結果が異なるテストを効率よく行うことができます。

3.1.3.1. MFUnit テストの作成

1) Eclipse IDE メニューより、[ファイル(F)] > [新規(N)] > [COBOL ユニットテストプロジェクト] を選択します。



2) 「プロジェクト名」に "AirportDemoTutNativeMFUnitTest2" を入力し、実行環境に合わせたプロジェクトテンプレート を選択した上で、[終了(F)] ボタンをクリックします。

COBOL ユニット テスト プロジェクト ワークスペースに COBOL ユニット テスト プロジェクトを新規作成します。
プロジェクト名(P): AirportDemoTutNativeMFUnitTest2
プロジェクトテンプレートを選択
○ Ext 1 1 1 1 1 1 2 ビット
□ テンプレートの参照
場所: 参照
ファイルシステムを選択: default ~
✓ デフォルト・ロケーションの使用(D)
ロケーション(L): C:¥workspace¥AirportDemoTutNativeMFUnitTest2 参照(R)
コッイル・システムを運用(ハ)・デフォルト
(?) < 厚3(B) 次へ(N) > 終了(F) キャンセル
注意)
先行作業にて指定したビット数と同じテンプレートを選択してください。

3) AirportDemoTutNativeMFUnitTest2 プロジェクトを選択した上で、Eclipse IDE メニューより、[ファイル(F)] > [新規

(N)] > [COBOL ユニットテスト] を選択します。

ファイ	Ίル(<u>E</u>)	編集(<u>E</u>)	ソース	リファクタリング	ナビゲート(<u>N</u>)	検索	æ	COBOL プロジェクト
	新規(N	l)			Alt+シフト	+N >	2	COBOL コピーファイル プロジェクト
	ファイル	を開く(.)					٠ ۲	リモート COBOL プロジェクト
	ファイル	・システム	からプロシ	ジェクトを開く			1	リモート COBOL コピーファイル プロジェクト
	最近の	ファイル				>	e	COBOL ユニット テスト プロジェクト
	閉じる(C)			Ctrl-	+W	2	COBOL JVM プロジェクト
	すべて鳥	引じる(L)			Ctrl+シフト-	+W		リモート COBOL ユニット テスト プロジェクト
	保存(5)	\			Ctr	2+1	Jo≎ LaU	COBOL JVM ユニット テスト プロジェクト
	別名保	/ 存(Δ)			cu		2	リモート COBOL JVM プロジェクト
	すべて信	R管(F)			Ctrl+シフト	+5	1	プロジェクト(R)
	前回保	管した状態	態に戻す	(T)		-	B	COBOL プログラム
	チクモトハノ	<u> </u>					BŶ	COBOL コピーファイル
-2	12回(1	/… 亦亩(M)				F2	_ ₽	スタンドアロン ファイル
<u>ା</u> କ	百新(F)	≪				F5	Ċ	フォルダー
©	行区切	/ り文字の:	<u> 変換(D)</u>			5	Ľ	ファイル
			~1~(0)					リモート スタンドアロン ファイル
۳	EIJ师J(P)			Ctrl	I+P	@	Enterprise Server Common Web Administration 接続
è	インポー	ŀ(I)					<u></u>	Directory Server 接続
4	エクスポ	¦-ト(O)						Enterprise Server
	プロパテ	-1(R)			Alt+En	nter	Ē₿	COBOL ユニット テスト

4) [COBOL ユニット テストの新規作成] ウィンドウが表示されるので [プログラム ユニット テスト] 項目を選択し、[次へ

(N)] ボタンをクリックします。

COBOL ユニット テスト 作成するテストのタイプを選択してください	
● フロクラム ユニット テスト プログラムを直接呼び出して、入力と出力をアサートできるようにします。CSV ファイルから読み込んだデータを使用して、プログラムを繰り返し実行できます。	
○ 自己完結型ユニット テスト Micro Focus Unit Test Preprocessor を使用してテスト ケースを既存のソースコードにコンパイルし、セクションと段落を直接テストできるようにします。	

5) 「プログラムのユニットテストを作成する」項目を選択し、[参照] ボタンをクリックします。

COBOL ユニット 終 テストする有効なソ	テスト Iースファイルを指定する必要があります	D U
含まれるプロジェクト:	AirportDemoTutNativeMFUnitTest2	参照
新規ファイル名:	TestProgram1.cbl]
◉ プログラムのユニッ	トテストを作成する	
テスト対象のプログ	54:	参照

6) AirportDemoTutNativeMFUnitTest プロジェクト内の「aircode.cbl」を選択した上で、[OK] ボタンをクリックします。

🔵 บ-スファイル	_		×
ソースファイルを選択します			
 AirportDemoTutNativeMFUnit Settings New_Configuration.bin aircode.cbl main.cbl 			
OK		キャンセ	ll I

7) テスト対象のプログラムに、さきほど選択した aircode.cbl が表示されていることを確認して、[終了(F)] ボタンをクリックしま す。

COBOL ユニット エディタで開くことがで	テスト できる COBOL ユニット テスト ファイルを新規作成	にします。			
含まれるプロジェクト:	AirportDemoTutNativeMFUnitTest2				参照
新規ファイル名:	TestProgram1.cbl				
⑦ プログラムのユニ	ットテストを作成する				
テスト対象のプログ	ラム: AirportDemo/aircode.cbl				参照
○ テンプレートからご	1ニットテストを作成する				
テンプレートを選択					
📄 Micro Fo	icus テンプレート				
				テンプレ-	・トの設定を構成
	参照				
場所:					参照
ファイルシ	ステムを選択: default ~				
?		< 戻る(B)	次へ(N) >	終了(F)	キャンセル

- 8) AirportDemoTutNativeMFUnitTest2 プロジェクト配下の TestProgram1.cbl を削除してください。
- AirportDemoTutNativeMFUnitTest2 プロジェクトを選択したうえで、Eclipse IDE メニューから [ファイル(F)] > [イ ンポート(I)] をクリックします。

ファイ	Ίル(<u>E</u>)	編集(<u>E</u>)	ソース	リファクタリング	ナビゲート(<u>N</u>)	検索
	新規(ファイ) ファイ) ティノ 最近((N) ルを開く(.) ル・システム: のファイル	からプロミ	ジェクトを開く	Alt+シフト	+N > >
	閉じる すべて	5(C) :閉じる(L)			Ctrl・ Ctrl+シフト・	+W +W
	保存(別名((S) 保存 (A)			Ctr	I+S
¢.	すべて 前回([保管(E) 保管した状態	態に戻す	(T)	Ctrl+シフト	`+S
r	移動(名前{	(V) を変更(M).				F2
88) 	更新(行区)	(F) 切り文字の)	変換(D)			F5
•	印刷((P)			Ctr	I+P
è	インボ	i−⊦(I)				
4	エクス	ポート(O)				



ファイル・システム ローカル・ファイル・システムオ	からリソースをインポートしま	t ø.				
次のディレクトリーから(<u>Y</u>):	C:¥vc-elnativetestfra	mework¥DataDrive	n		~	参照(<u>R</u>)
🔳 😕 DataDriven			✓ ₱inpu ✓ @ TestF	t.csv rogram1.cbl		
タイプをフィルター(<u>T</u>)	すべて選択(<u>S</u>)	選択をすべて解除	<u>D</u>)			
オブション □ 警告を出さずに既存! □ トップ・レベルのフォルタ 拡張 >>(<u>A</u>)						
び ロジェクトは以下の	のようになります。		< 庆의(<u>B</u>)	次八(<u>N</u>) >	於「(<u>F</u>)	キャンセル
 ✓ Correct AirportDemo ✓ Ø COBOL JI > Ø TestPro > Ø New_Conf ▲ input.csv 	TutNativeMFUnitTe ログラム gram1.cbl iguration.bin	est2				

12) TestProgram1.cbl を開き、プログラムを確認します。

行数	コード説明			
10 - 12	01 mfu-dd-airport1 is MFU-DD-VALUE external.			
	01 mfu-dd-airport2 is MFU-DD-VALUE external.			
	01 mfu-dd-distance-km is MFU-DD-VALUE external.			
	テストデータファイル "input.csv" から取得する各テストデータの値が格納されます。			
53	entry MFU-TC-PREFIX & TEST-TESTAIRCODE.			
	基本的な単体テストプログラムと同様の entry 句の構成ですが、テストデータ1 行ごとに			
	呼び出されるようになります。			
	また、基本的な単体テストプログラムでは、テスト失敗時には MFU-ASSERT-FAIL-Z を			
	使用してエラー情報を戻していましたが、データ駆動型テストではこちらは使用できません。			
	このため、70 行目では goback での戻り値に MFU-FAIL-RETURN-CODE を設定			
	しています。			
77	entry MFU-TC-METADATA-SETUP-PREFIX & TEST-TESTAIRCODE.			
	テストデータファイルを読み込みます。			

3.1.3.2. MFUnit テストの実行

本テスト対象のプログラムは、環境変数で設定された空港情報が保存されたデータファイルを参照するため、手順内で設定を行い ます。

 AirportDemoTutNativeMFUnitTest2 プロジェクト配下の「TestProgram1.cbl」を選択した状態で、マウスの右クリ ックでコンテクストメニューを表示し、[実行(R)] > [実行の構成(N)] をクリックします。

0	Coverage As	>	H		
0	実行(R)	>	CBL	1 COBOL アプリケーション	Alt+シフト+X,N
夺	デバッグ(D)	>	œ'n	2 COBOL ユニット テスト	Alt+シフト+X,F
	プロファイル(P)	>		実行の構成(N)	
\checkmark	使祉				

2) 画面左側より「COBOL ユニットテスト」を選択した状態で、マウスの右クリックにてコンテクストメニューを表示し、[新規構成(W)]を選択します。

構成の作成、管理、および実行



3) 「名前」に "AirtportDemoTutNativeMFUnitTest2" を入力し、「環境」 タブを選択した後、[追加(A)] ボタンをク リックします。



4) 以下の情報を入力し、[OK]ボタンをクリックします。

変数: "dd_airports"

值: "..¥..¥AirportDemoTutNativeMFUnit¥airports.dat"

変数 <mark>: dd_airports</mark>			
值:¥¥Airport	Demo¥airports.dat	 	
?		ОК	キャンセル

5) さきほど追加した dd_airports 環境変数が表示されていることを確認して、[適用(Y)] ボタンをクリックした後、[実行(R)] ボタンをクリックします。

名前(N): AirportDemoTutNativeMFUnitTest2					
🗟 一般 🤤 ソース 🍓 環境 🔲 共通(C) 👂 実行時 💱 デバッグシンボル 🔬 動的分析 🥸 CTF 🥑 コンテナー					
(注: ここで定義された変数は、任意の現在の設定値、または任意の指定された 環境スクリプト内の設定値を上書きします。)					
変数 値 dd_airports¥.¥AirportDemoTutNativeMFUnit¥airports.dat 編集(E) 利除(R)					
実行する環境スクリプト: 場所: 参照					
ファイルがプロジェクト内にある場合、絶対パスは相対パスになります。 パラメータ: □ 関連付けられたプロジェクトのビルド環境から値を継承					
	前回保管した状態に戻す(V) 適用(V)				
	実行(R) 閉じる				

実行後に [Micro Focus Unit Testing] ビューを表示すると、以下のように1つテストが失敗します。



テストデータファイルの 4 行目 LTN, LGW の距離がエラーとなっています。これは、テストデータファイルの 820 が誤りであり、 正しい値は 82 です。テストデータファイルの 820 を 82 に修正したうえで、テストを再実行すると、以下のように全て成功しま す。

実行: 4/4 ロ ゴラー: 0 ロ 失敗: 0
✓ ₩ AirportDemoTutNativeMFUnitTest2 (2 ms)
✓ I TestProgram1.cbl
✓ 归 MFUT_TESTAIRCODE (2 ms)
🚰 Row #1 (LHR,HND,9591)
🗁 Row #2 (LHR,NRT,9592)
🚰 Row #3 (LTN,LGW,82)
P Row #4 (STN, LCY, 47)

3.1.4. 自己完結型テスト

この方式では、テスト対象となるプログラム内にテストコードをコンパイル時に埋め込むことで、より粒度の小さなテストを行えます。

3.1.4.1. **MFUnit テストの作成**

1) Eclipse IDE メニューより、[ファイル(F)] > [新規(N)] > [COBOL ユニットテストプロジェクト] を選択します。

🔵 tut_native_mfunit - Eclipse				
771	ル(F) 編集(E) リファクタリング ナビゲート(N) 検索	プロジ	1	COBOL コピーファイル プロジェクト
新規(N) Alt+シフト+N > ファイルを開く(.)		堂! (111)	リモート COBOL プロジェクト リモート COBOL コピーファイル プロジェクト	
	□ファイル・システムからプロジェクトを開く 最近のファイル >			COBOL ユニット テスト プロジェクト
	閉じる(C) Ctr	l+W		リモート COBOL ユニット テスト プロジェクト
	すべ(閉じる(L) (trl+シノ) 保存(S) (trl+シノ)	+W rl+S		COBOL JVM ユニット テスト ブロジェクト リモート COBOL JVM プロジェクト
	別名保存(A)		1	プロジェクト(R)

2) 「プロジェクト名」に "AirportDemoTutNativeMFUnitTest3" を入力し、実行環境に合わせたプロジェクトテンプレート

を選択した上で、[終了(F)] ボタンをクリックします。

COBOL ユニット テスト プロジェクト ワークスペースに COBOL ユニット テスト プロジェクトを新規作成します。
プロジェクト名(P): AirportDemoTutNativeMFUnitTest3
ノビジェクト テジノレートを選択 ② Micro Focus テンプレート [32 ピット] ② Micro Focus テンプレート [64 ピット]
<u>テンプレートの設定を構成</u> □ テンプレートの参照
場所: 参照
ファイルシステムを選択: default ~
☑ デフォルト・ロケーションの使用(型)
ロケーション(L): C¥workspace¥AirportDemoTutNativeMFUnitTest3 参照(R)
ファイル・システムを選択(<u>')</u> : <mark>デフォルト</mark> ~
② 終了(E) キャンセル
注意) 先行作業にて指定したビット数と同じテンプレートを選択してください。

 AirportDemoTutNativeMFUnitTest3 プロジェクトを選択した上で、Eclipse IDE メニューより、[ファイル(F)] > [新 規(N)] > [COBOL ユニットテスト] を選択します。

2	ァイル(E) 編集(E) ソース リファクタリング	ナビゲート(<u>N</u>) 検索	COBOL プロジェクト
	新元(N) ファイルを開く(.) ファイル・システィーかにプロジェクトを用く	Alt+VJP+N >	 COBOL コピーファイル プロジェクト リモート COBOL プロジェクト
	ま近のファイル	>	 ・ リモート COBOL コピーファイル プロジェクト ・ ・ ・
	閉じる(C) すべて閉じる(L)	Ctrl+W Ctrl+シフト+W	 COBOL JVM プロジェクト (論) リモート COBOL ユニットテスト プロジェクト
	 保存(S) 別名保存(A) すべて保管(F) 	Ctrl+S	 〇〇日のビットテストプロジェクト ジェクト COBOL JVM プロジェクト プロジェクト(R)
	前回保管した状態に戻す(T)		COBOLプログラム
- 49 98	*8勘(V) 3 名前を変更(M)) 更新(F) 行区切り文字の変換(D)	F2 F5 >	COBUL 12 - J74 // COBUL 12 - J74 // COBUL 12 - J74 //
đ) 印刷(P)	Ctrl+P	 リモート スタントアロン ファイル Enterprise Server Common Web Administration 接続
	a インポート(I) a エクスポート(O)		➢ Directory Server 接続 ☆ Enterprise Server
	プロパティ(R)	Alt+Enter	다. COBOL 그二ット テスト

4) [自己完結型ユニットテスト] を選択して[次へ(N)]ボタンをクリックします。

COBOL ユニットテスト 作成するテストのタイプを選択してください
○ プログラム ユニット テスト プログラムを直接呼び出して、入力と出力をアサートできるようにします。CSV ファイルから読み込んだデータを使用して、プログラムを繰り返し実行できます。
 自己完結型ユニットテスト Micro Focus Unit Test Preprocessor を使用してテストケースを既存のソースコードにコンパイルし、セクションと段落を直接テストできるようにします。
(?) (N) > 終了(F) キャンセル

5) テスト対象のプログラム欄の右にある[参照...]ボタンをクリックしたうえで、AirportDemoTutNativeMFUnit プロジェクト配下の aircode.cbl を選択し、[終了(F)]ボタンをクリックします。

COBOL ユニット テスト 自己完結型ユニット テスト用 COBOL ファイルを新現作成します。						
含まれるプロジェクト:	AirportDemoTutNativeMFUnitTest3				参照	
テスト対象のプログラム:	AirportDemoTutNativeMFUnit/aircode.c	:bl			参照	
?		< 戻る(<u>B</u>)	次へ(<u>N</u>) >	終了(E)	キャンセル	

以下のようなコピーブックファイルが作成されます。

B N	/IFUPD_aircode.cpy	у 🛛
	MFUPD_aircod	de.cpy
	•••••*A•1•B•	••••••4••••
6	entry go	"MFUT_TESTAIRCODE". oback

自己完結型のテストでは、このコピーブックがテスト対象プログラム(本手順では aircode.cbl)内にコンパイル時に埋め込まれます。このため、テストコードに必要なデータ項目は LINKAGE SECTION で定義された項目ではなく、WORKING-STORAGE SECTION 内で定義された実体のある項目で記述されている必要があります。

サンプルファイルを展開したフォルダ内の Self¥MFUPD_aircode.cpy の内容で、MFUPD_aircode.cpy を上書きしてく ださい。



この例では、open-airfile と close-airfile section のテストを行っています。open-airfile のテストコードを確認すると、 fstat-1 項目の値を直接参照して処理結果を評価していることが確認できます。一方、close-airfile は特に判定材料が ないため、常にテストは成功します。

3.1.4.2. MFUnit テストの実行

本テスト対象のプログラムは、環境変数で設定された空港情報が保存されたデータファイルを参照するため、手順内で設定を行い ます。

 AirportDemoTutNativeMFUnitTest3 プロジェクト配下の「aircode.cbl」を選択した状態で、マウスの右クリックでコ ンテクストメニューを表示し、[実行®] > [実行の構成(N)] をクリックします。

Q.	Coverage As	>	\vdash		
0	実行(R)	>	CBL	1 COBOL アプリケーション	Alt+シフト+X,N
*	デパッグ(D)	>	œ	2 COBOL ユニット テスト	Alt+シフト+X,F
	プロファイル(P) 検証	>		実行 の構成(N)	

2) 画面左側より「COBOL ユニットテスト」を選択した状態で、マウスの右クリックにてコンテクストメニューを表示し、[新規構成(W)] を選択します。

構成の作成、管理、および実行



3) 「名前」に "AirtportDemoTutNativeMFUnitTest3" を入力し、「環境」 タブを選択した後、[追加(A)] ボタンをクリックします。

名前(<u>N</u>): AirportDemoTutNativeMFUr	nitTest3					
🗔 一般 🤤 ソース 🚾 環境 🔲 共調	≜(C) 👂 実行時 💱 デバッグシンボル 🛃 動的分析 🗳 CTF	💣 בעד				
(注: ここで定義された変数は、任意の現在の設定値、または任意の指定された 環境スクリプト内の設定値を上書きします。)						
変数	値	追加(A)				
		編集(<u>E</u>)				
		削除(<u>R</u>)				

4) 以下の情報を入力し、[OK]ボタンをクリックします。

変数: "dd_airports"

值: "..¥..¥AirportDemoTutNativeMFUnit¥airports.dat"

環境変数を追加または変更します

変数: dd_airports		
值:¥¥AirportDemo¥airports.dat		
?	OK ++7	セル

5) さきほど追加した dd_airports 環境変数が表示されていることを確認して、[適用(Y)] ボタンをクリックした後、[実行(R)] ボタンをクリックします。

名前(N): AirportDemoTutNativeMFUnitTest3	
🕟 一般 🧤 ソース 🌄 環境 🔲 共通(C) 👂 実行時 💺	
(注:ここで定義された変数は、任意の現在の設定値、または任 環境スクリプト内の設定値を上書きします。)	言の指定された
变数	
dd_airports	¥¥AirportDemoTutNativeMFUnit¥airports.dat
	漏集(<u>E</u>)
	<u> </u> 削除(<u>R</u>)
実行する環境スクリプト:	
	参照
ファイルかフロジェクト内にある場合、絶対ハスは相	対ハスになります。
パラメータ:	
関連付けられたノロシェットのビルト環境がら値を継承	
	前回保管した状態に戻す(V) 適用(Y)
	実行(<u>R</u>) 閉じる

実行後に [Micro Focus Unit Testing] ビューを表示すると、以下のように2つのテストがともに成功していることが確認 できます。

א-עעב 📃 🖡	🛛 問題 <u>द</u> Micro Focus Unit Testing 🖇	3
実行: 2/2	⊠ Iラ-: 0	⊠ 失敗: 0
V Airport V I airce	DemoTutNativeMFUnitTest3 (2 ms) ode.cbl MFUT_TESTAIRCODE (2 ms) MFUT_TESTAIRCODE2 (0 ms)	

3.2. コマンドラインからの実行

MFUnit によるテストは、Eclipse 上の画面からではなく、コマンドライン上からも行なうことができます。従来のスタイルでのテスト 作業の効率化を図ることができ、Jenkins などの CI ツールと連携する事で、テストの自動実行を行なえるため、品質担保や作 業工数の削減が見込めます。

どのテスト方式でも実行方法は同じであるため、ここでは 3.1.2 で作成したテストプログラムをコマンドラインから実行する方法について学びます。

1) スタートメニューより、Micro Focus Visual COBOL 配下の Visual COBOL コマンドプロンプト をクリックします。



注意)

3.1.2 にて選択したビットと同様のプロンプトを使用してください。

2) 作業フォルダを作成し、作成したフォルダに移動します。

C:¥>mkdir VCCommandTutorial		
C:¥>cd VCCommandTutorial		
C:¥VCCommandTutorial>		

- 3) 3.1 で利用したワークスペースフォルダを ECLIPSE_WORKSPACE に指定した上で、下記コマンドを実行します。
 - set ECLIPSE_WORKSPACE=c:\u00e4workspace_tut_native_mfunit
 - cobol %ECLIPSE_WORKSPACE%¥AirportDemoTutNativeMFUnit¥aircode.cbl;
 - cobol %ECLIPSE_WORKSPACE%¥AirportDemoTutNativeMFUnitTest¥TestProgram1.cbl sourceformat(variable);
 - cbllink -D aircode.obj
 - cbllink -D TestProgram1.obj

C:¥VCCommandTutorial>set ECLIPSE_WORKSPACE=c:¥workspace_tut_native_mfunit C:¥VCCommandTutorial>cobol %ECLIPSE_WORKSPACE%¥AirportDemoTutNativeMFUnit¥airc ode.cbl;

(コマンド実行中の出力内容を省略)

C:¥VCCommandTutorial>cobol %ECLIPSE_WORKSPACE%¥AirportDemoTutNativeMFUnitTest¥

TestProgram1.cbl sourceformat(variable); (コマンド実行中の出力内容を省略) C:¥VCCommandTutorial>cbllink -D aircode.obj (コマンド実行中の出力内容を省略) C:¥VCCommandTutorial>cbllink -D TestProgram1.obj (コマンド実行中の出力内容を省略)

以下のファイルが作成されていることを確認します。

C:¥VCComma	andTutoria	l>dir				
ドライブ C のボ	ぎリューム ラベ	ルがありませ	せん。			
ボリューム シリフ	アル番号は 7	8B6-9FC	DC です			
C:¥VCComm	nandTutoria	al のディレ	クトリ			
2023/11/14	16:12	<dir></dir>				
2023/11/14	16:12	<dir></dir>				
2023/11/14	16:12		17,408	aircode.dll		
2023/11/14	16:12		10,159	aircode.obj		
2023/11/14	16:12		15,872	TestProgram1.dll		
2023/11/14	16:12		8,164	TestProgram1.obj		
	4 個のファ	ィル		51,603 バイト		
	2 個のデ	ィレクトリ	27,802,	177,536 バイトの空き領域	式	

- 4) プロンプト上で下記コマンドを実行し、MFUnit を実行します。
 - set dd_airports=%ECLIPSE_WORKSPACE%¥AirportDemoTutNativeMFUnit¥airports.dat
 - mfurun -report:junit -outdir:result TestProgram1.dll

C:¥VCCommandTutorial>set	
dd_airports=%ECLIPSE_WORKSPACE%¥AirportDemoTutNativeMFUnit¥airports.dat	
C:¥VCCommandTutorial>mfurun -report:junit -outdir:result TestProgram1.dll	
Micro Focus COBOL - mfurun Utility	
Unit Testing Framework for Windows/Native/64	
Fixture : TestProgram1	
Test Run Summary	
Overall Result	Passed
Tests run	2
Tests passed	2
Tests failed	0
Total execution time	0

outdir オプションにより、出力結果を result フォルダに保存されていることを確認してください。



WHAT'S NEXT

• 本チュートリアルで学習した技術の詳細については製品マニュアルをご参照ください。

免責事項

ここで紹介したソースコードは、機能説明のためのサンプルであり、製品の一部ではございません。ソースコードが実際に動作するか、御社業務に適合するかなどに関しまして、一切の保証はございません。 ソースコード、説明、その他すべてについて、無謬性は保障されません。 ここで紹介するソースコードの一部、もしくは全部について、弊社に断りなく、御社の内部に組み込み、そのままご利用頂いても構いません。 本ソースコードの一部もしくは全部を二次的著作物に対して引用する場合、著作権法の精神に基づき、適切な扱いを行ってください。