Micro Focus Visual COBOL チュートリアル

COBOL 開発：Visual Studio – 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 11
- Visual COBOL 9.0 for Visual Studio 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。 .NET 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

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. Visual Studio の起動

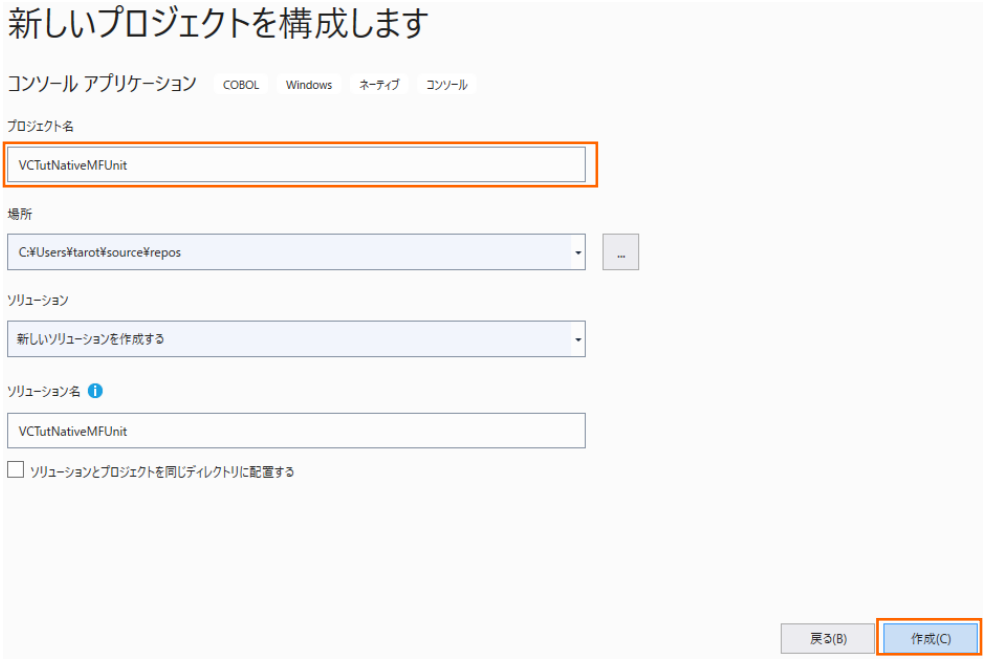
1) スタートメニューより、Visual Studio を起動します。

3.1.1.2. チュートリアルプロジェクトの作成

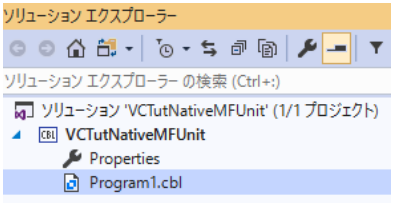
1) [新しいプロジェクトの作成] をクリックします。

2) 言語に “COBOL”、プロジェクトタイプに ”ネーティブ” を選択し、「コンソール アプリケーション」を選択した上で、[次へ(N)] ボタンをクリックします。

3) プロジェクト名に “VCTutNativeMFUnit” を入力し、[作成(C)] ボタンをクリックします。

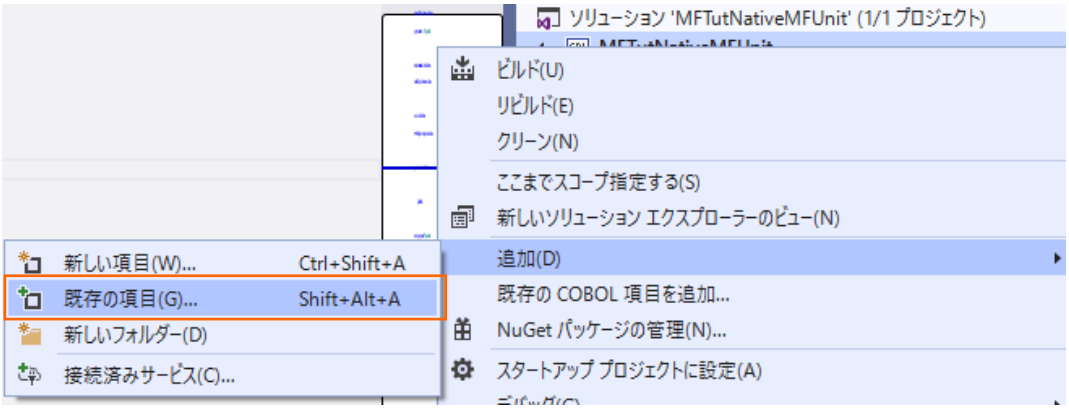


新規プロジェクトが作成されます。



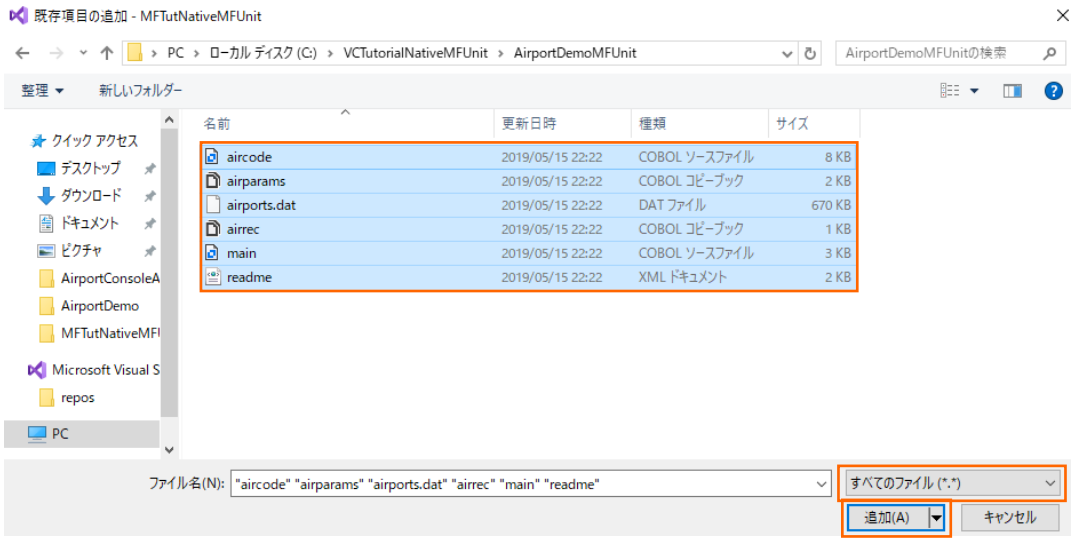
この Program1.cbl は不要なため、削除してください。

4) VCTutNativeMFUnit 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、[追加(D)] > [既存の項目(G)] を選択します。

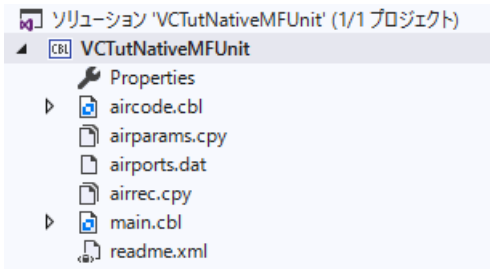


5) サンプルファイルを展開したフォルダ内の AirportDemoMFUnit フォルダ配下を選択し、”すべてのファイル(*.*)” を選択し

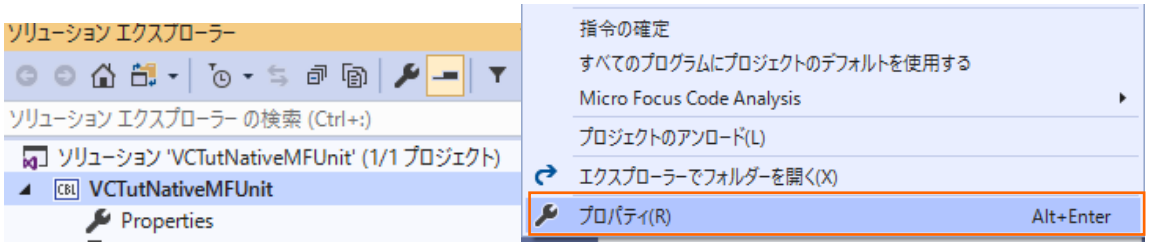
た結果、表示される全てのファイルを選択した上で、[追加(A)] ボタンをクリックします



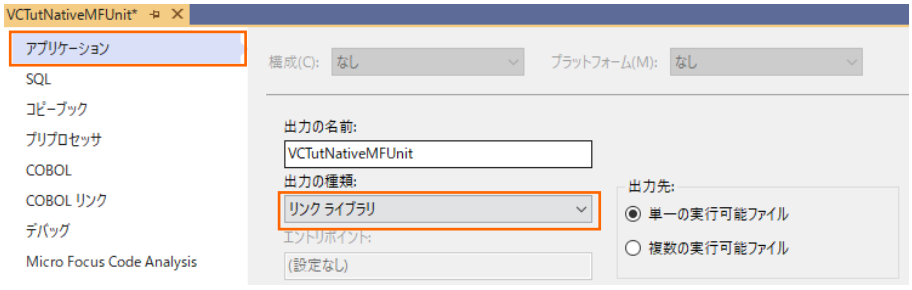
プロジェクトが、以下のようになります。



6) VCTutNativeMFUnit 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。



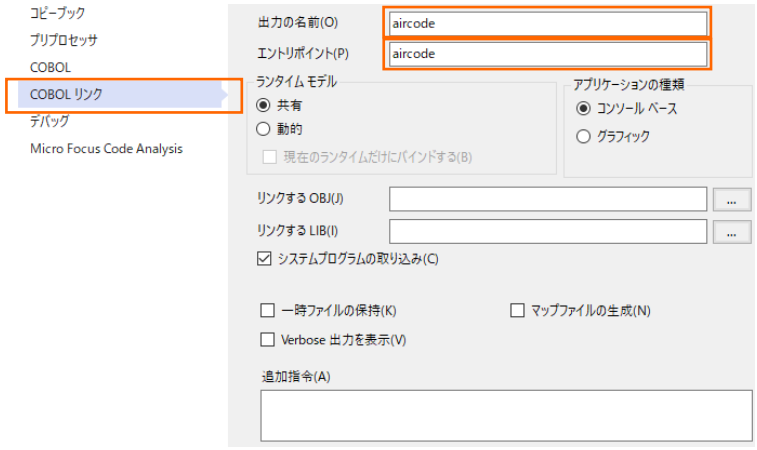
7) アプリケーションタブを選択し、「出力の種類」に “リンク ライブラリ” を選択し、保存します。



8) COBOL リンクタブを選択し、以下の入力を行った後、保存します。

出力の名前： “aircode”

エントリポイント： “aircode”

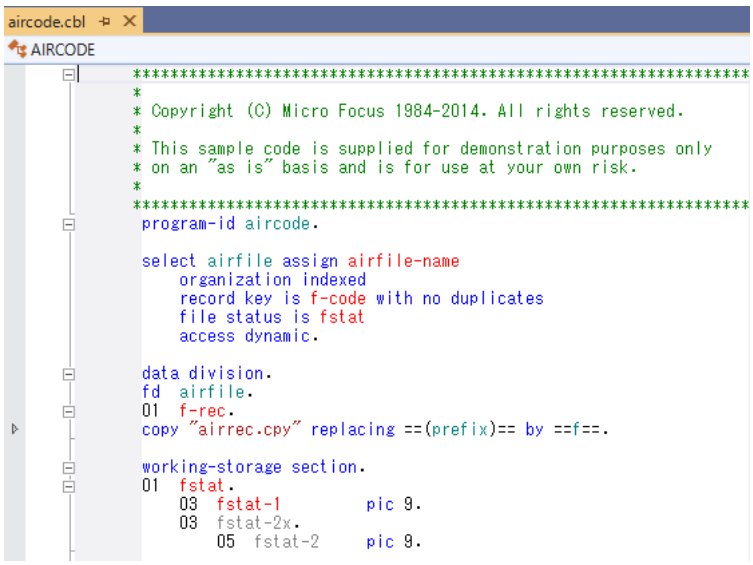


3.1.2. 基本的なテスト

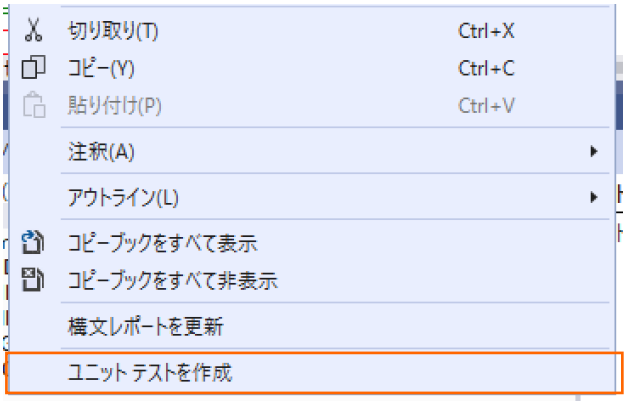
この方式では、１つのテストを１つのテストブロックで記述していきます。

3.1.2.1. MFUnit テストの作成

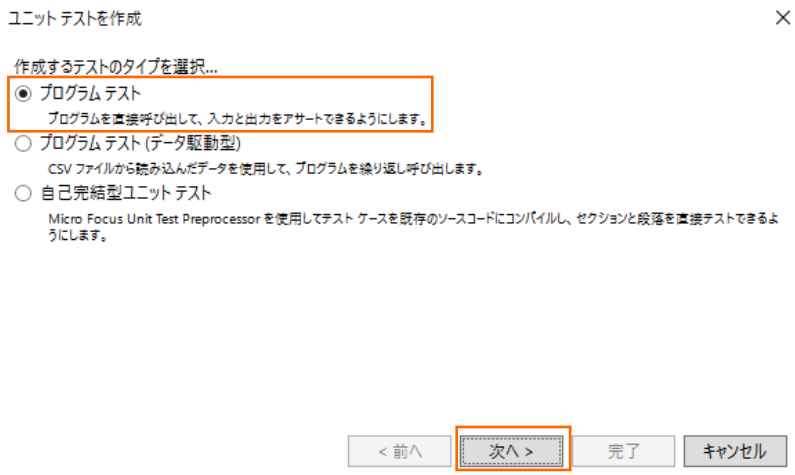
1) ソリューションエクスプローラーより、aircode.cbl をダブルクリックして、エディターで開きます。



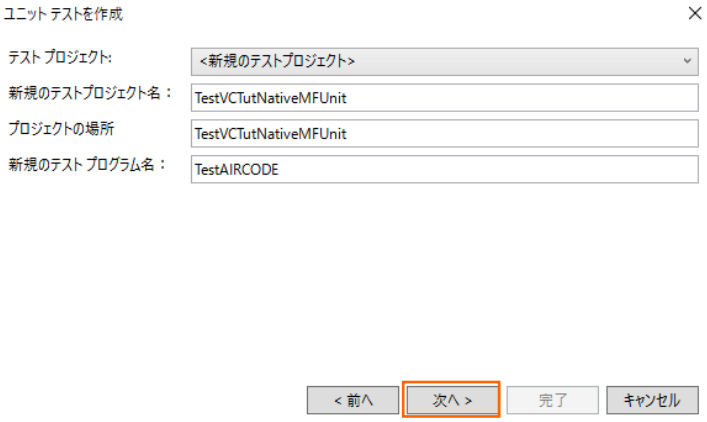
2) エディター上にて、マウスの右クリックにてコンテクストメニューを表示し、[ユニットテストを作成] を選択します。



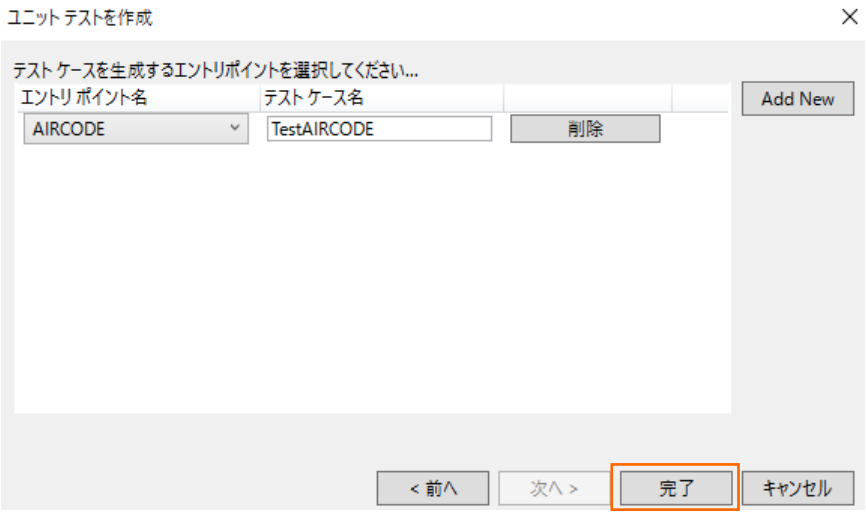
3) 「プログラムテスト」を選択し、[次へ >] ボタンをクリックします。



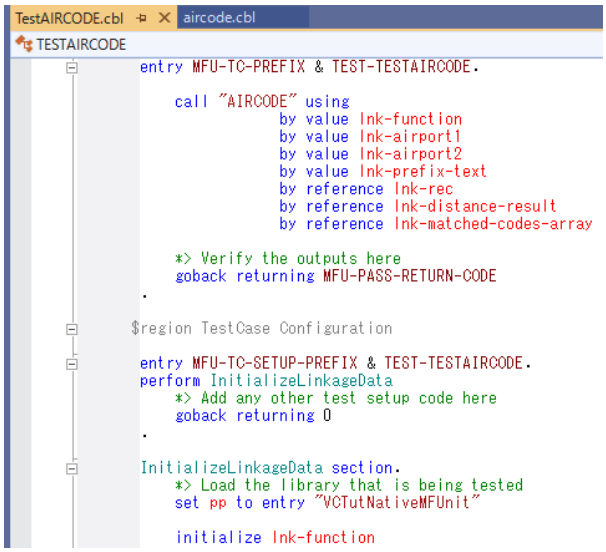
4) そのまま [次へ>] ボタンをクリックします。



5) そのまま [完了] ボタンをクリックします。



単体テストプログラムの雛型が作成されます。



6) 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¥TestAIRCODE.cbl で、現在の TestAIRCODE.cbl を上書き保存します。これは、テストケース “testDistance” を途中まで作成したものになります。プログラムを確認すると、MFU-TC-SETUP-PREFIX, MFU-TC-PREFIX, MFU-TC-TEARDOWN-PREFIX から始まる “testDistance” の 3 entry が定義されていることが分かります。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。

① entry MFU-TC-SETUP-PREFIX & “testDistance”

② entry MFU-TC-PREFIX & “testDistance”

③ entry MFU-TC-TEARDOWN-PREFIX & “testDistance”

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。

テスト本体である MFU-TC-PREFIX を確認すると、下記のように結果検証コードが実装されていません。

```
entry MFU-TC-PREFIX & "testDistance"
    set get-distance to true
    move "HND" to lnk-airport1
    move "LHR" to lnk-airport2
    call "AIRCODE" using by value lnk-function
                         by value lnk-airport1
                         by value lnk-airport2
                         by value lnk-prefix-text
                         by reference lnk-rec
                         by reference lnk-distance-result
                         by reference lnk-matched-codes-array

    move 9591 to wk-distance-km
    display wk-distance-km " / " distance-km.

$region TestCase Configuration

entry MFU-TC-SETUP-PREFIX & "testDistance".
```

このテストを実装するため、以下のコードを７８行目に挿入してください。

```
if wk-distance-km = 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.
```

補足）
テスト失敗時の記述方法として、成功時同様に、戻り値で返す方法は、以下の通りです。

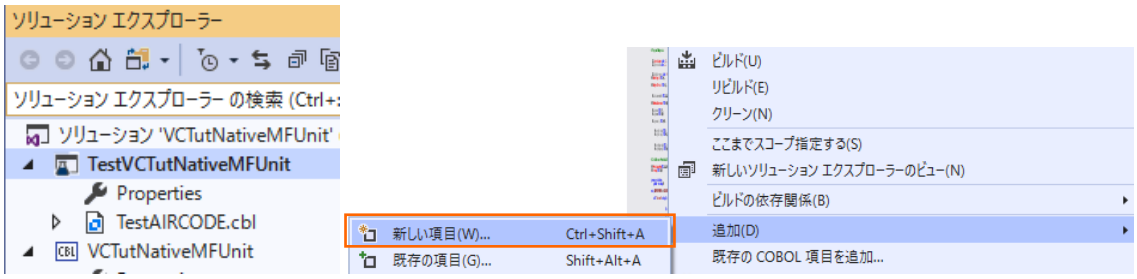
```
if wk-distance-km = distance-km
then
    goback returning MFU-PASS-RETURN-CODE
else
    string
        "expected "
        wk-distance-km
        ", but "
        distance-km
        into err-msg
    end-string
    display err-msg
    goback returning MFU-FAIL-RETURN-CODE
end-if.
```

戻り値 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。

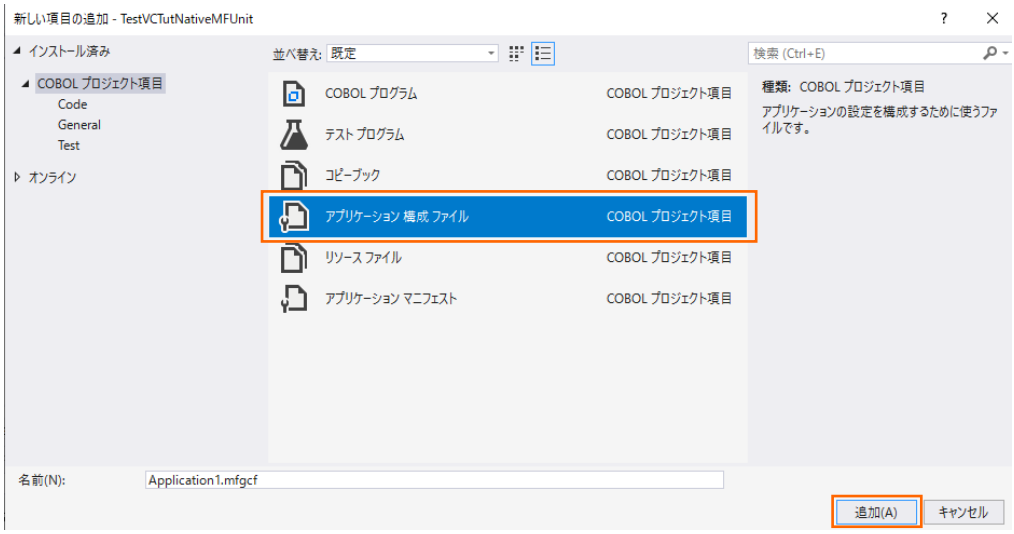
3.1.2.2. MFUnit テストの実行

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。

1) TestVCTutNativeMFUnit プロジェクトを選択し、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、[追加(D)] > [新規の項目(W)] を選択します。



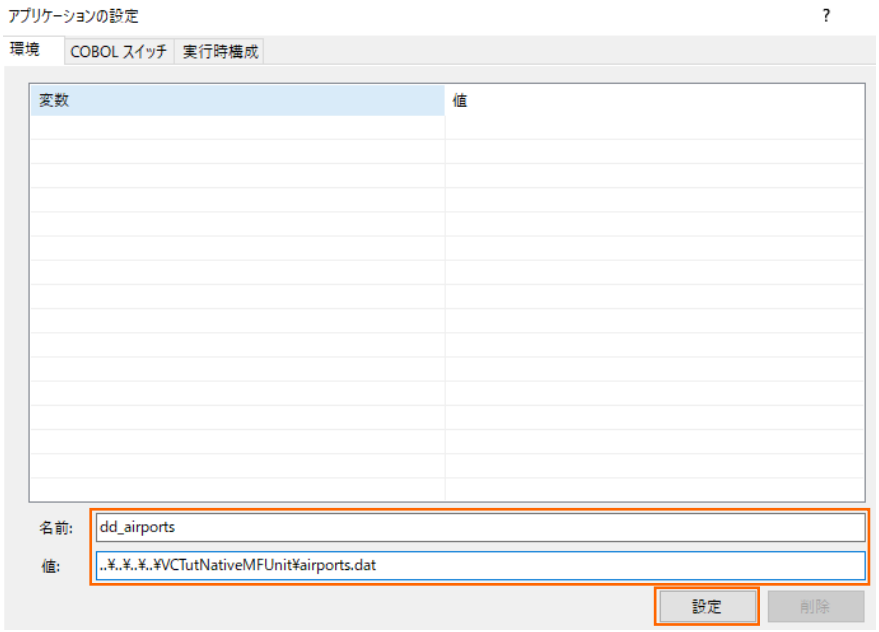
2) 「アプリケーション構成ファイル」を選択し、[追加(A)] ボタンをクリックします。



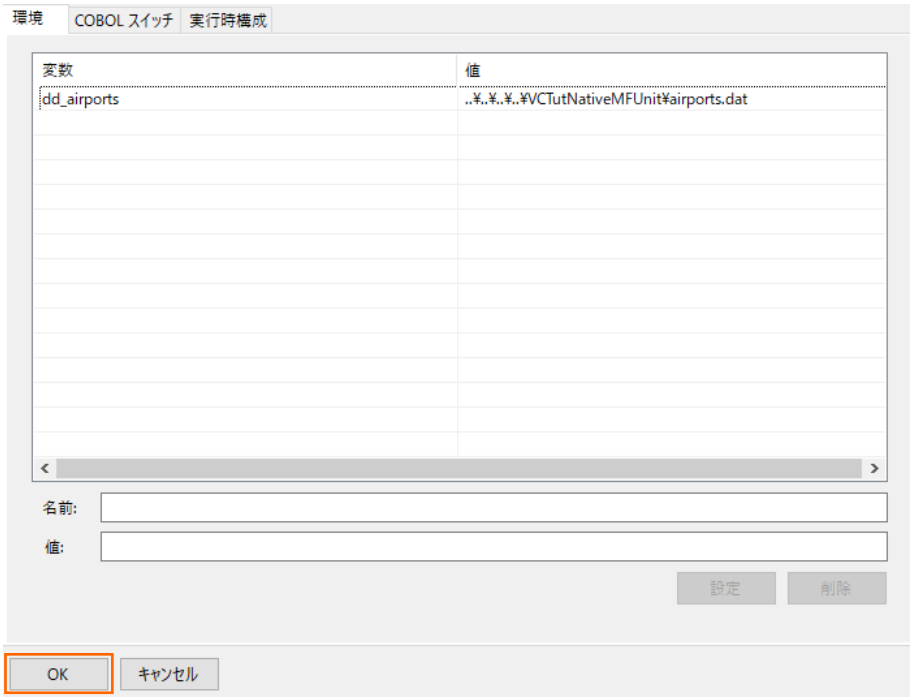
3) TestVCTutNativeMFUnit プロジェクト配下に追加された Application.mfgcf をクリックし、表示されたダイアログ上で、以下の入力を行い、[設定] ボタンをクリックします。

名前： “dd_airports”

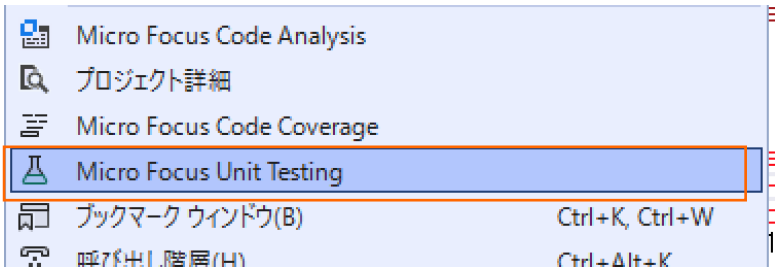
値： “..¥..¥..¥..¥VCTutNativeMFUnit¥airports.dat”



dd_airports 変数が追加されたことを確認して、[OK] ボタンをクリックします。



4) Visual Studio IDE メニューより、[表示(V)] > [Micro Focus Unit Testing] を選択します。



5) Micro Focus Unit Testing ビューより、[全て実行] ボタンをクリックします。

以下のように全て緑色のマークが設定されます。緑色は、テストに成功したことを示します。

6) エラーケースを確認します。「TestAIRCODE.cbl」をエラーとなるように修正した上で、再度、Micro Focus Unit Testing ビューより、[全て実行] ボタンをクリックします。

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。

```
move 9592 to wk-distance-km
display wk-distance-km " / " distance-km.
if wk-distance-km = distance-km
  then
    goback returning MFU-PASS-RETURN-CODE
else
    string
      "expected "
      wk-distance-km
      ", but "
      distance-km
      into err-msg
    end-string
    call MFU-ASSERT-FAIL-Z using err-msg
end-if.
```

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。

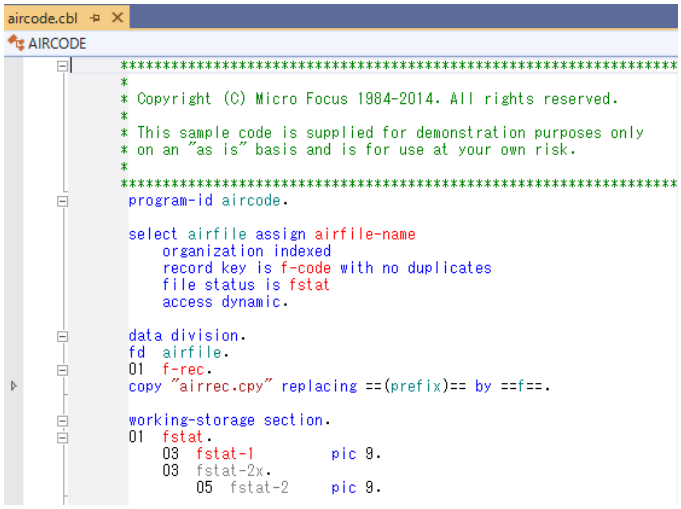

Micro Focus Unit Testing
コードカバレッジで実行
すべて実行
直前を実行
失敗を実行
選択を実行
TestVCTutNativeMFUnit (0 ms)
TestAIRCODE.cbl (0 ms)
MFUT_TESTAIRCODE (0 ms)
MFUT_TESTDISTANCE (0 ms)
テスト結果
サマリ: 1 1 0 0
MFUT_TESTDISTANCE - 出力
expected 9,592, but 9,591
システム出力
HND Tokyo Intl
Japan Lat:+035.552258
LHR Heathrow
United Kingdom Lat:+051.004775
9,592 / 9,591
Error opening airport data

3.1.3. データ駆動型テスト

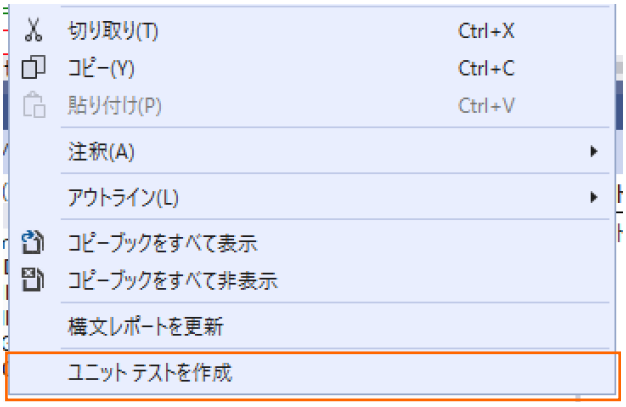
この方式では、複数のテストデータをデータファイルに設定しておくことで、データによって結果が異なるテストを効率よく行うことができます。

3.1.3.1. MFUnit テストの作成

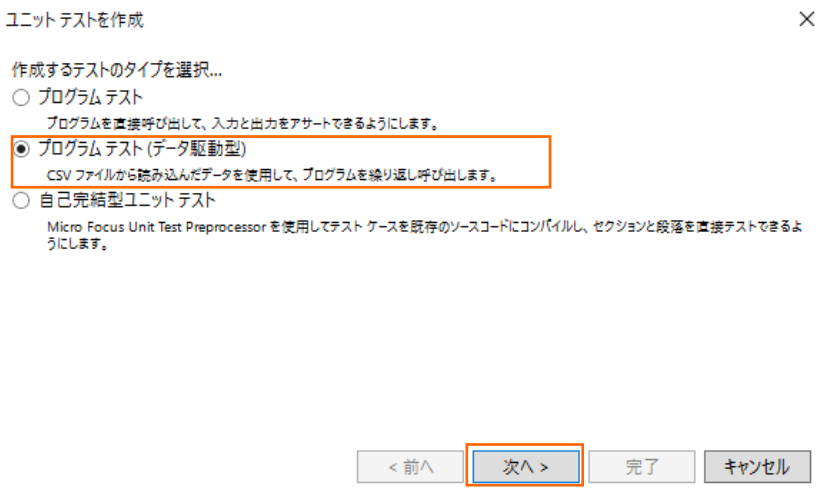
1) ソリューションエクスプローラーより、VCTutNativeMFUnit プロジェクト配下の aircode.cbl をダブルクリックして、エディターで開きます。



2) エディター上にて、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、[ユニットテストを作成] を選択します。



3) 「プログラムテスト（データ駆動型）」を選択肢、[次へ>]ボタンをクリックします。

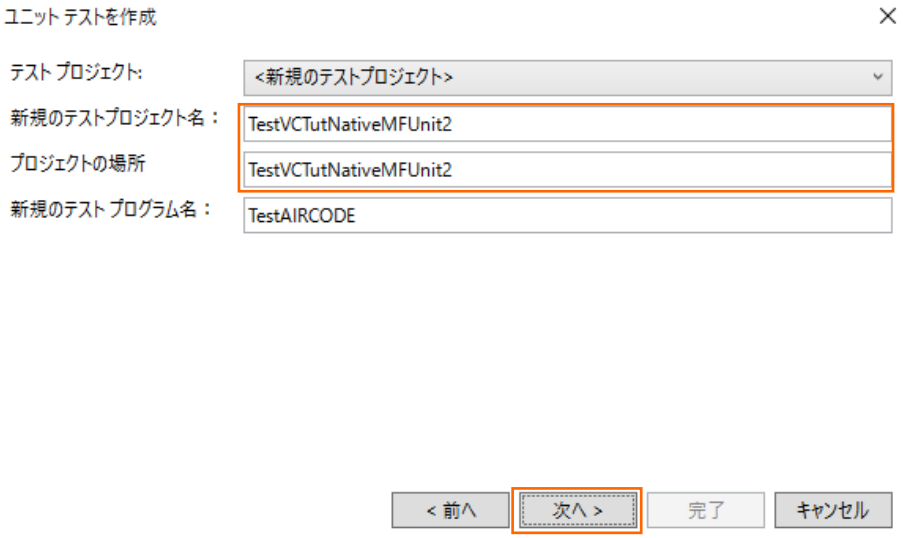


4) 以下の入力を行った後、[次へ>]ボタンをクリックします。

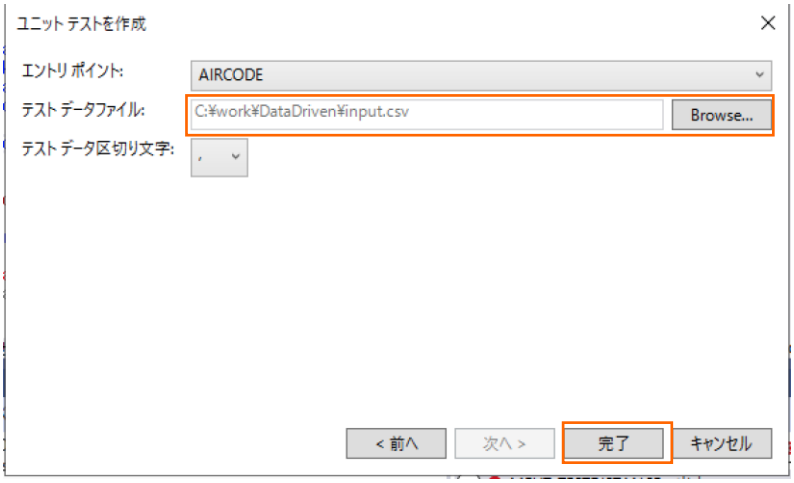
テストプロジェクト： “<新規のテストプロジェクト>” を選択

新規のテストプロジェクト名： “TestVCTutNativeMFUnit2”

プロジェクト場所： “TestVCTutNativeMFUnit2”



5) [Browse...] ボタンをクリックし、サンプルファイルを展開したフォルダ内の DataDriven¥input.csv を選択したうえで、[完了] ボタンをクリックします。



6) テストのひな型が作成されます。サンプルファイルを展開したフォルダ内の DataDriven¥TestAIRCODE.cbl で中身を上書きしてください。

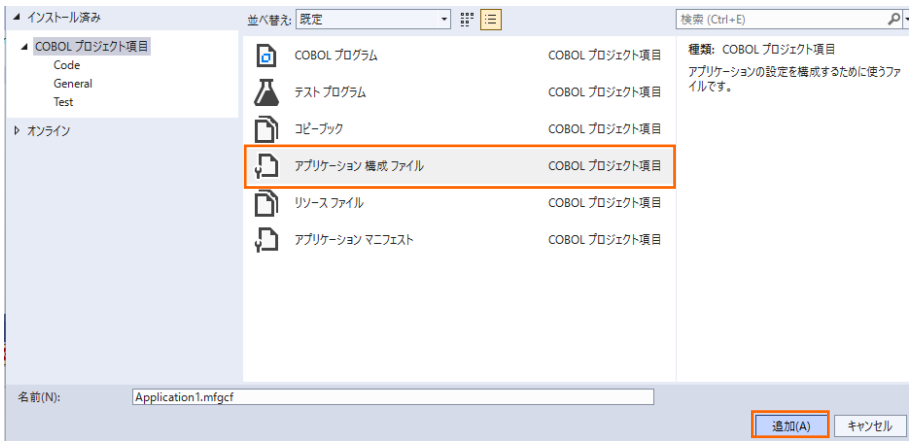
行数	コード説明
46 - 48	01 mfu-dd-airport1 is MFU-DD-VALUE external. 01 mfu-dd-airport2 is MFU-DD-VALUE external. 01 mfu-dd-distance-km is MFU-DD-VALUE external. テストデータファイル “TestAIRCODE.csv” から取得する各テストデータの値が格納されます。
53	entry MFU-TC-PREFIX & TEST-TESTAIRCODE. 基本的な単体テストプログラムと同様の entry 句の構成ですが、テストデータ 1 行ごとに呼び出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、72 行目では goback での戻り値に MFU-FAIL-RETURN-CODE を設定しています。
78	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

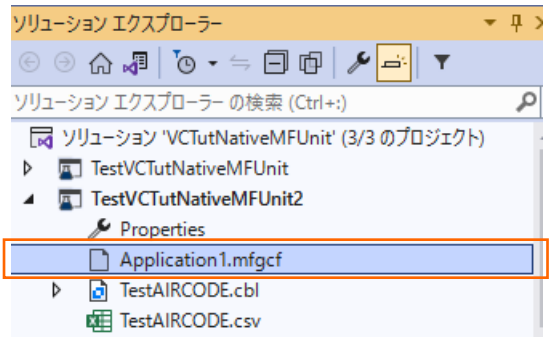
本テスト対象のプログラムは、環境変数で設定された空港情報が保存されたデータファイルを参照するため、手順内で設定を行います。

1) TestVCTutNativeMFUnit2 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、[追加(D)] > [新しい項目(W)] を選択します。

2) 「アプリケーション構成ファイル」を選択し、[追加(A)] ボタンをクリックします。



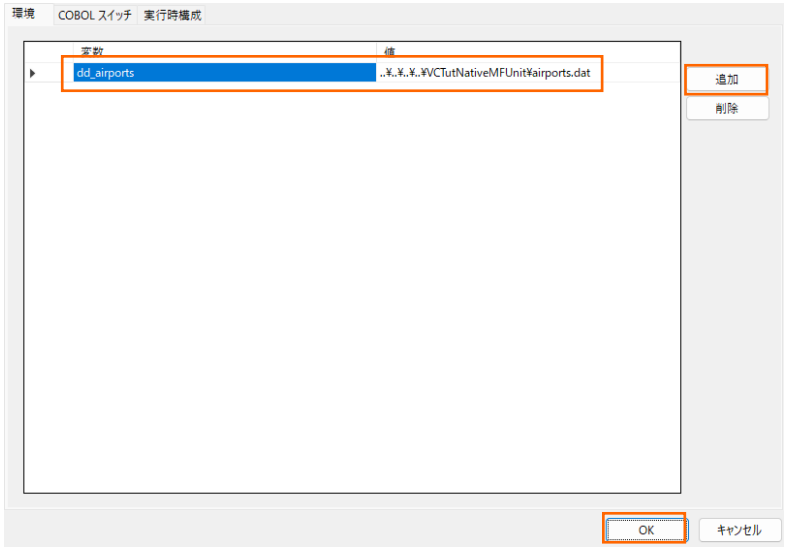
3) 追加された「Application1.mfgcf」をダブルクリックします。



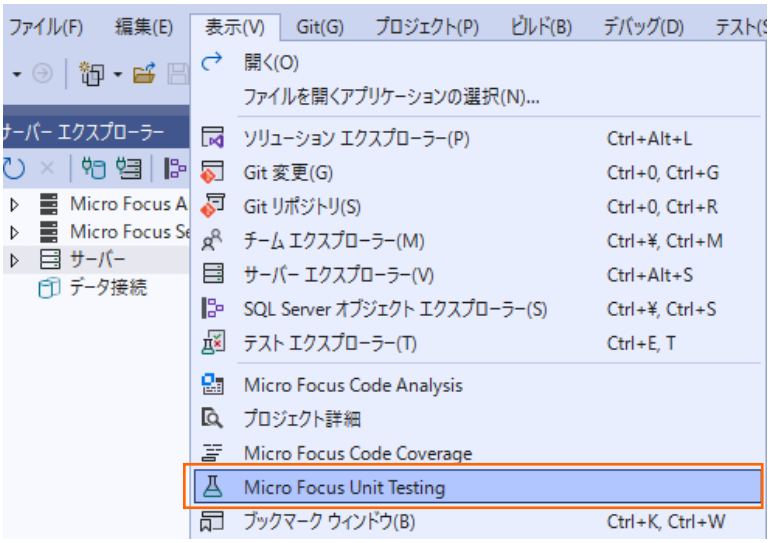
4) [追加] ボタンをクリックしたうえで、以下の入力を行い、[OK] ボタンをクリックします。

変数： “dd_airports”

値： “..¥..¥..¥..¥VCTutNativeMFUnit¥airports.dat”



5) Visual Studio のメニューより、[表示(V)] > [Micro Focus Unit Testing] を選択します。



6) Micro Focus Unit Testing ビューより、[すべて実行] をクリックします。

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。

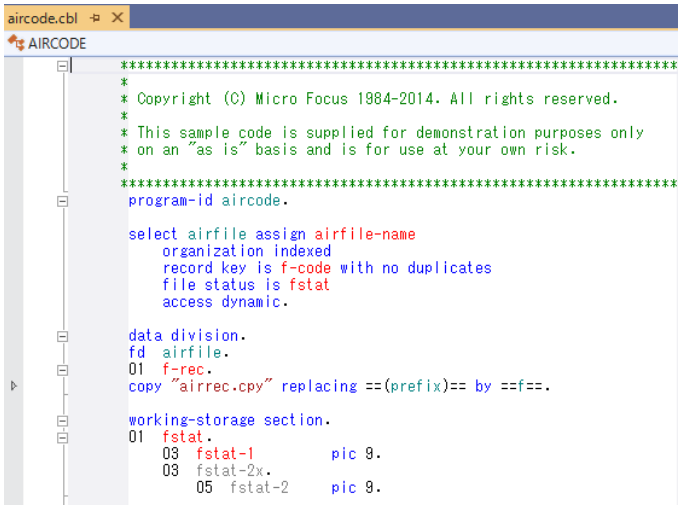
テストデータの 4 行目 LTN, LGW の距離がエラーとなっています。これは、テストデータファイルの 820 が誤りであり、正しい値は 82 です。テストデータファイルの 820 を 82 に修正したうえで、テストを再実行すると、以下のように全て成功します。

3.1.4. 自己完結型テスト

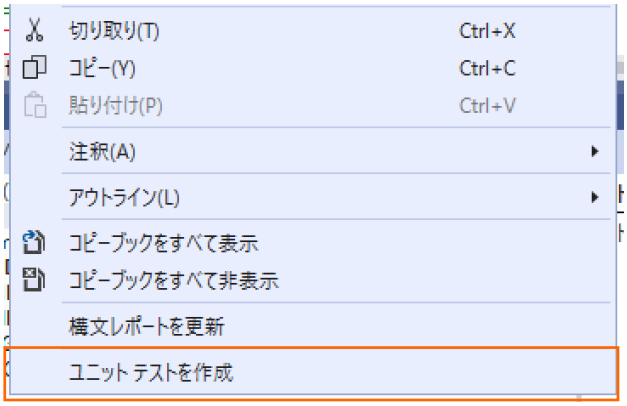
この方式では、テスト対象となるプログラム内にテストコードをコンパイル時に埋め込むことで、より粒度の小さなテストを行えます。

3.1.4.1. MFUnit テストの作成

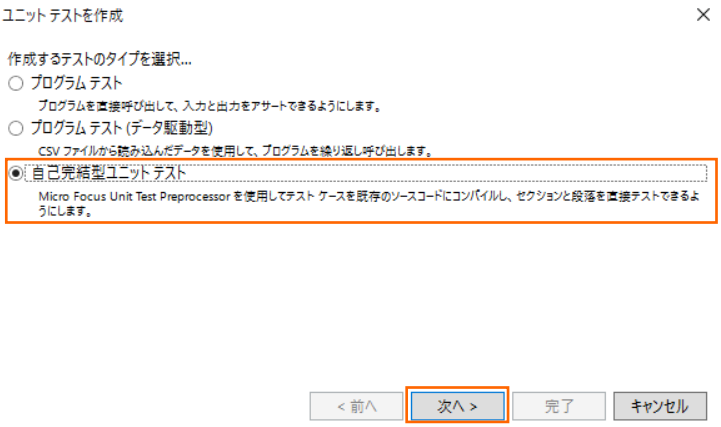
1) ソリューションエクスプローラーより、VCTutNativeMFUnit プロジェクト配下の aircode.cbl をダブルクリックして、エディターで開きます。



2) エディター上にて、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、[ユニットテストを作成] を選択します。



3) 「自己完結型テスト」を選択肢、[次へ>]ボタンをクリックします。

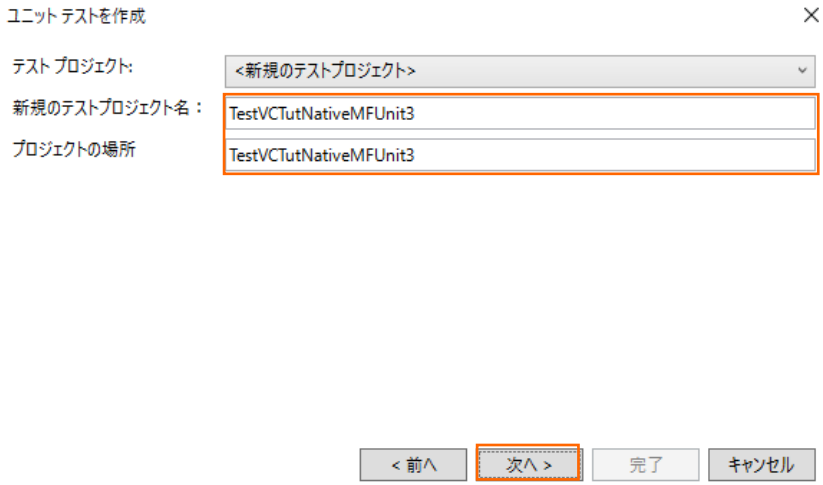


4) 以下の入力を行ったうえで、[次へ>] ボタンをクリックします。

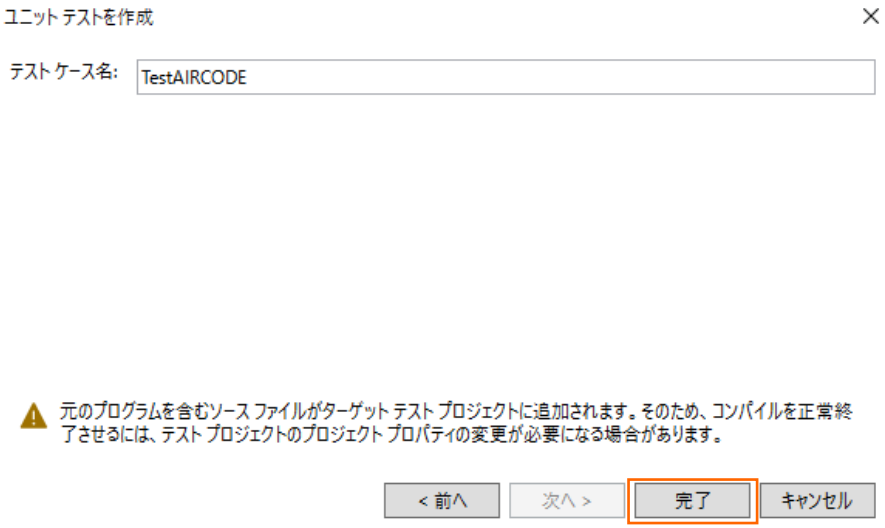
テストプロジェクト： “<新規のテストプロジェクト>”を選択

新規のテストプロジェクト名： “TestVCTutNativeMFUnit3”

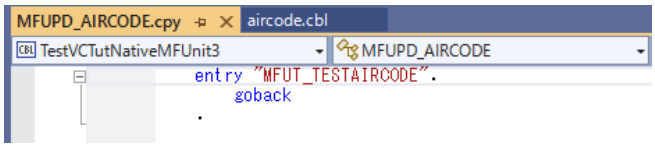
プロジェクトの場所： “TestVCTutNativeMFUnit3”



5) そのまま [完了] ボタンをクリックします。



以下のようなコピーブックファイルが作成されます。



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。

```
entry "MFUT_TESTAIRCODE".
      perform open-airfile
      if fstat-1 not = 0
            goback returning -1
      end-if
      goback.
entry "MFUT_TESTAIRCODE2".
      perform close-airfile
      goback.
```

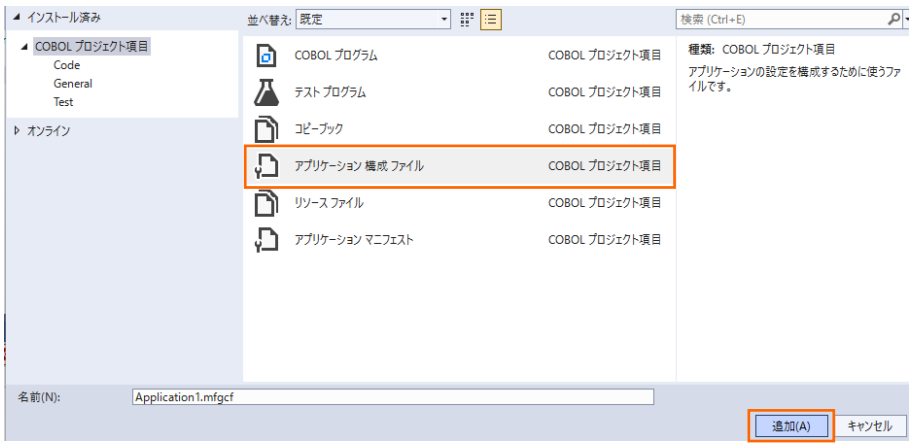
この例では、open-airfile と close-airfile section のテストを行っています。open-airfile のテストコードを確認すると、fstat-1 項目の値を直接参照して処理結果を評価していることが確認できます。一方、close-airfile は特に判定材料がないため、常にテストは成功します。

3.1.4.2. MFUnit テストの実行

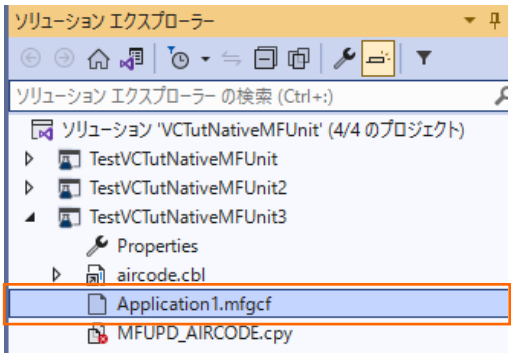
本テスト対象のプログラムは、環境変数で設定された空港情報が保存されたデータファイルを参照するため、手順内で設定を行います。

1) TestVCTutNativeMFUnit3 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、[追加(D)] > [新しい項目(W)] を選択します。

2) 「アプリケーション構成ファイル」を選択し、[追加(A)] ボタンをクリックします。



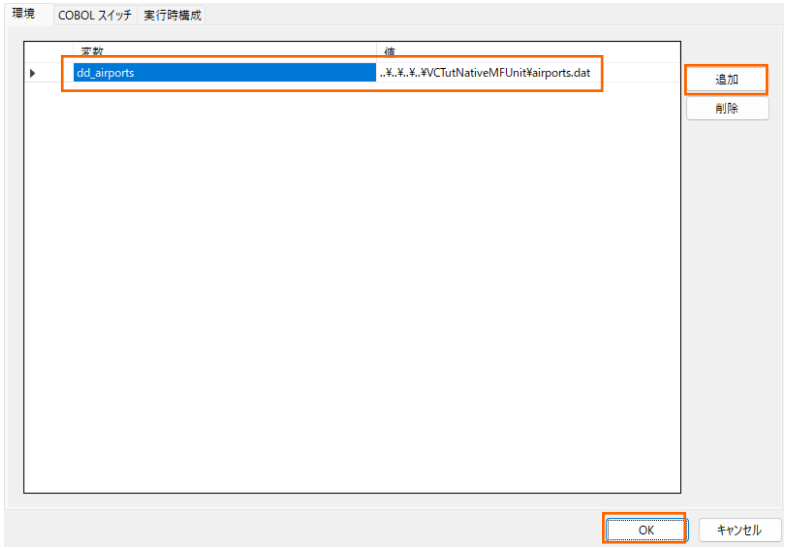
3) 追加された「Application1.mfgcf」をダブルクリックします。



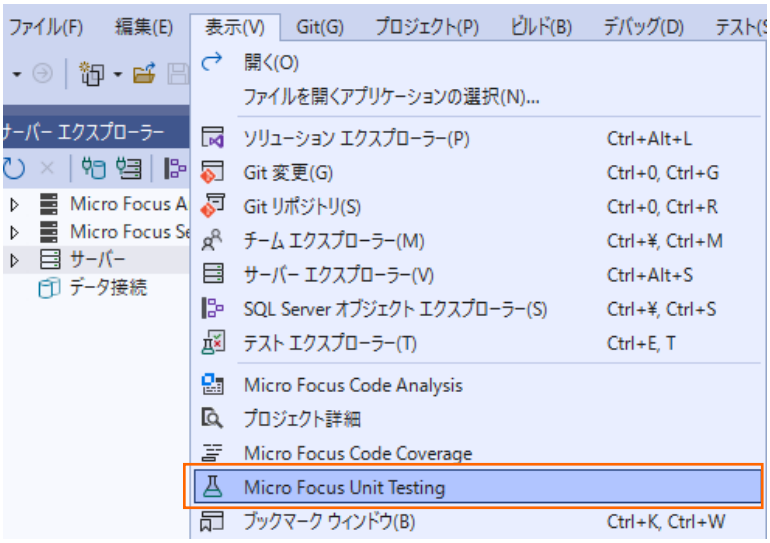
4) [追加] ボタンをクリックしたうえで、以下の入力を行い、[OK] ボタンをクリックします。

変数： “dd_airports”

値： “..¥..¥..¥..¥VCTutNativeMFUnit¥airports.dat”



5) Visual Studio のメニューより、[表示(V)] > [Micro Focus Unit Testing] を選択します。



6) Micro Focus Unit Testing ビューより、[すべて実行] をクリックします。

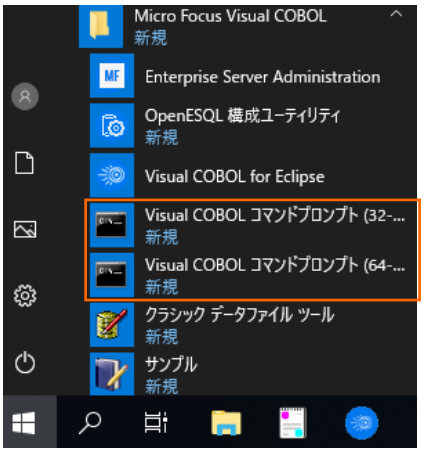
実行後に [Micro Focus Unit Testing] ビューを表示すると、以下のように 2 つのテストがともに成功していることが確認できます。

3.2. コマンドラインからの実行

MFUnit によるテストは、Visual Studio 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、実行環境に合わせた [Visual COBOL コマンドプロンプト] をクリックします。



1) 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 VC90CommandTutorial
C:¥>cd VC90CommandTutorial
C:¥VC90CommandTutorial>
```

2) 3.1 で作成したソリューションが保存されているフォルダを VS_SOLUTION_PATH に指定した上で、下記コマンドを実行します。

- set VS_SOLUTION_PATH=c:¥vs_solution_path
- cobol %VS_SOLUTION_PATH%¥VCTutNativeMFUnit¥aircode.cbl;
- cobol %VS_SOLUTION_PATH%¥TestVCTutNativeMFUnit¥TestAIRCODE.cbl sourceformat(variable);
- cbllink -D aircode.obj
- cbllink -D TestAIRCODE.obj

注意）
VS_SOLUTION_PATH は、各環境に合わせて修正してください。

```
C:¥VC90CommandTutorial> set VS_SOLUTION_PATH=c:¥vs_solution_path
C:¥VC90CommandTutorial>cobol %VS_SOLUTION_PATH%¥VCTutNativeMFUnit¥aircode.cbl;
(コマンド実行中の出力内容を省略)
C:¥VC90CommandTutorial>cobol %VS_SOLUTION_PATH%¥TestVCTutNativeMFUnit¥TestAIRC
```

```
ODE.cbl sourceformat(variable);
(コマンド実行中の出力内容を省略)
C:¥VC90CommandTutorial>cbllink -D aircode.obj
(コマンド実行中の出力内容を省略)
C:¥VC90CommandTutorial>cbllink -D TestAIRCODE.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。

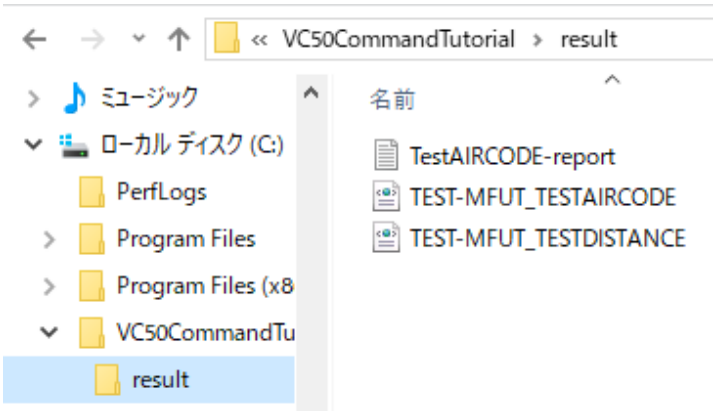
```
C:¥VC90CommandTutorial>dir
 ドライブ C のボリューム ラベルがありません。
 ボリューム シリアル番号は 969A-9F98 です
 C:¥VC90CommandTutorial のディレクトリ
2023/09/27   11:33     <DIR>            .
2023/09/27   11:33     <DIR>            ..
2023/09/27   11:33              17,408 aircode.dll
2023/09/27   11:22              10,121 aircode.obj
2023/09/27   11:33              15,360 TestAIRCODE.dll
2023/09/27   11:32               7,503 TestAIRCODE.obj
               4 個のファイル                 50,392 バイト
               2 個のディレクトリ   37,157,203,968 バイトの空き領域
```

3) 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=%VS_SOLUTION_PATH%¥VCTutNativeMFUnit¥airports.dat
- mfurun -report:junit -outdir:result TestAIRCODE.dll

```
C:¥VC90CommandTutorial>set
dd_airports=%VS_SOLUTION_PATH%¥VCTutNativeMFUnit¥airports.dat
C:¥VC90CommandTutorial>mfurun -report:junit -outdir:result TestAIRCODE.dll
Micro Focus COBOL - mfurun Utility
Unit Testing Framework for Windows/Native/64
Fixture : TestAIRCODE
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。