White Paper
Micro Focus Visual COBOL

Micro Focus Visual COBOL を活用した開発作業の効率化

Visual COBOL は Micro Focus が提供する最新版の COBOL 統合開発環境製品です。本製品は COBOL のコーディング・コンパイル機能やデバッグ機能のみならず、さまざまなシーンでの使用を想定した COBOL 専用の開発補助機能を豊富に搭載しています。これらはバッチ、シェル、CI ツール等からの利用も想定して設計されており、有効活用することで変化に強い高速な開発スタイルの実現も可能です。本書ではこれらの Visual COBOL に装備された主な開発補助機能を概説いたします。

目次

1. はじめに 3
1.1 Micro Focus Visual COBOL とは 3
1.2 Visual COBOL を活用した開発の自動化 3

2. 自動実行も可能、Visual Studio、Eclipse COBOLプロジェクトのビルド 3
2.1 MSBuild による Visual Studio プロジェクトのビルド 3
2.2 Ant による Eclipse プロジェクトのビルド 7

3. Test Coverage – COBOL 専用のカバレッジ分析ツール 12
3.1 Test Coverage の基本動作 12
3.2 IDE を通じた Test Coverage の操作 13

4. コード分析 – COBOL 専用の静的コード解析ツール 15
4.1 コード分析の基本動作 15
4.2 Enterprise Analyzer 上のクエリールールの活用 17
4.3 MSBuild, Ant による実行 18

5. mfunit - COBOL 専用の xUnit 系単体テストフレームワーク 19
5.1 mfunit の基本機能 19
5.2 JUnit 互換形式の出力フォーマットの活用 23

6. Profiler – COBOL 専用のパフォーマンス分析ツール 25
6.1 コマンドによる処理 25
6.2 IDE 上での処理 26

7. おわりに 28

1 はじめに

1.1 Micro Focus Visual COBOL とは

Visual COBOL は長年 COBOL 業界をリードしてきた Micro Focus が提供する最新版の COBOL 統合開発環境製品です。Windows, Linux, UNIX ターゲットのアプリケーションを最新テクノロジー環境にて様々なミドルウェア等と連携した COBOL アプリケーション開発を可能とします。Visual COBOL に装備された優れた開発ツール群を活用することで開発期間の短縮やコスト削減も望めます。

Windows ターゲットであれば Visual Studio や Eclipse IDE 上で COBOL アプリケーションを開発できます。Visual COBOL はこれらの IDE 向けに様々な COBOL 専用の開発支援機能を提供しています。例えば、COBOL 開発用に用意されたエディターを利用すれば、COBOL プログラムのコーディング・メンテナンスの生産性を著しく向上できます [1]。

Linux/UNIX をターゲットとする場合でも Windows をターゲットとする場合と同様に Eclipse IDE 上で Windows の開発と同じ感覚で開発が可能です [2]。

これらの環境上で開発した COBOL モジュールは Native アプリケーション、.NET アプリケーション、JVM アプリケーション、EJB、RESTful サービス、もしくは SOAP Web サービスとして運用が可能であり、目的に合ったアプリケーションを構築できます。

1.2 Visual COBOL を活用した開発の自動化

Visual COBOL は COBOL 開発における高機能なエディター、デバッガーを備えたコンパイラ製品に留まりません。Micro Focus が約 40 年に渡って培った COBOL アプリケーション開発ノウハウを反映させた COBOL 向けの開発支援機能も豊富に搭載しています。このうち本書では、以下の機能について概説いたします [3]。

> ビルド
> カバレッジ分析
> 静的コード解析
> 単体テスト
> パフォーマンス分析

これらの機能は様々なシーンでの使用を想定して設計されており、IDE 上での実行はもちろんのこと、Windows であればバッチや PowerShell、Linux/UNIX であればシェル経由で実行することも可能です。更には CI ツールによる実行を意識したインターフェースを用意したものもあり、これらの作業プロセスを自動化し継続的インテグレーションに組み入れるといった使い方も可能です。

2 自動実行も可能、Visual Studio、Eclipse COBOL プロジェクトのビルド

Visual COBOL は Visual Studio IDE 並びに Eclipse IDE 上で COBOL アプリケーションの開発を可能とします。C# や Java の開発と同様に IDE 上で COBOL アプリケーションのビルド命令を発行することも可能ですが、IDE の外でビルド命令を発行することも可能です。IDE を使用しないビルドについては従来よりコマンドラインインターフェースを提供していますが、ここでは IDE 上で作成したプロジェクトをビルドする方法を紹介いたします。各 IDE のプロジェクトとしてビルドするため、プロジェクトに設定したビルド命令、コンパイラ指令、プリコンパイル指令等をプロジェクトでビルド対象とするリソースに対して反映させます。

2.1 MSBuild による Visual Studio プロジェクトのビルド

MSBuild は Microsoft 社が提供するソースコードのコンパイルやパッケージング等の自動化を支援するビルドツールです。本ツールを使えば Visual Studio IDE がない環境 [4] であっても Visual Studio のソリューション並びにプロジェクトを Visual Studio IDE 上で設定したプロジェクト設定の内容に基づきビルドが出来るようになります。この MSBuild を活用した自動実行をサポートするツールは Microsoft Team Foundation Server を始め Jenkins のプラグイン等でも提供されており、コマンドラインのみならず様々な CI ツールより実行が可能です。Visual COBOL for Visual Studio がインストールされた環境であれば、Visual COBOL の COBOL プロジェクトについても自動ビルドが可能となります。ここでは Visual COBOL for Visual Studio 2015 で作成した COBOL のプロジェクトを Jenkins 経由でビルドするようすをご紹介します [5]。

1 エディターを活用した実際の開発イメージについては下記ページ中で公開しています動画「進化する COBOL エディター ～ Visual Studio 編～」及び「進化する COBOL エディター ～ Eclipse 編～」でご覧いただけます。 **http://www.microfocus.co.jp/resources/video/visualcobol/**

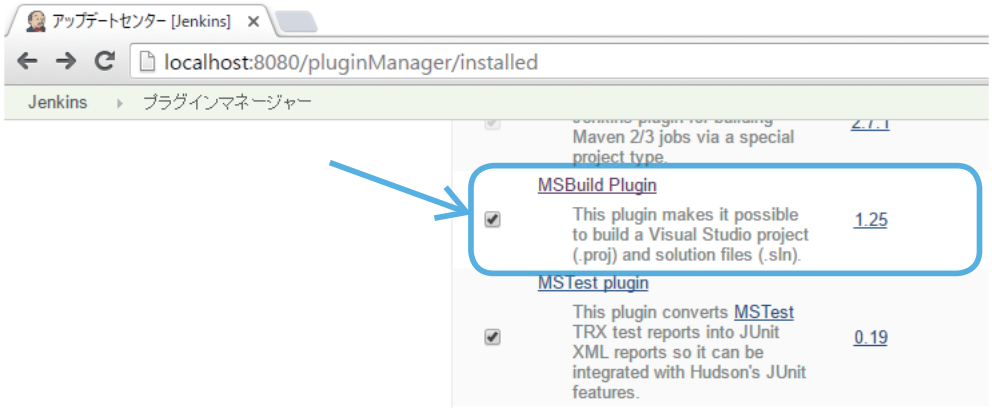
2 Linux/UNIX をターゲットとしたアプリケーションを Windows 上で Eclipse IDE を使って開発するようすを下記ページ中の「Eclipse IDE を使って UNIX/Linux ターゲットの COBOL アプリケーションをリモート開発」にてご覧いただけます。 **http://www.microfocus.co.jp/resources/video/visualcobol/**

3 本書では Visual COBOL 2.3J Update 1 for Windows を用いて動作イメージを紹介しています。異なるバージョンや OS 向けの製品で記載内容を試すのであればそれぞれに該当するマニュアルをご参照ください。

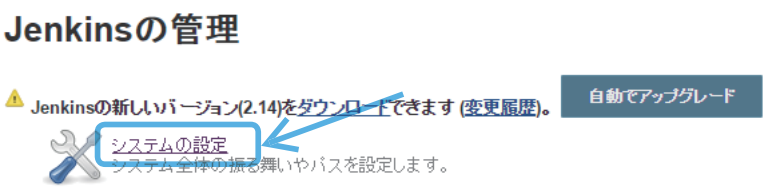
4 以前は、.NET Framework にバンドルされていたようですが、執筆時点の最新版については「Microsoft Build Tools 2015」にバンドルされているようです。「Microsoft Build Tools 2015」はMicrosoft 社のホームページより無償でダウンロードできるようです。

5 本書では Jenkins 1.654 を用いたイメージを掲載しています。別のバージョンをお使いになる場合、画面やプラグインのバージョン等、掲載と異なるものがあれば適宜置き換えてご覧ください。

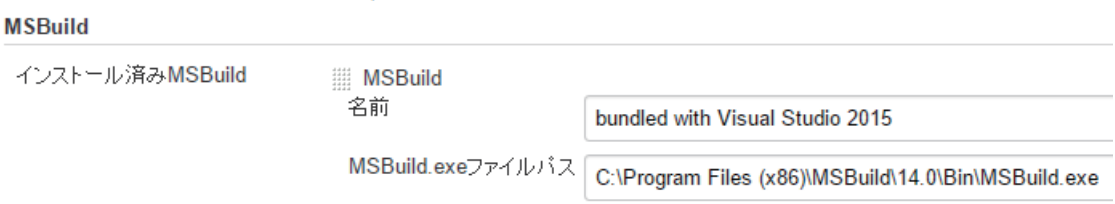
① Jenkins に MSBuild のプラグラインを追加



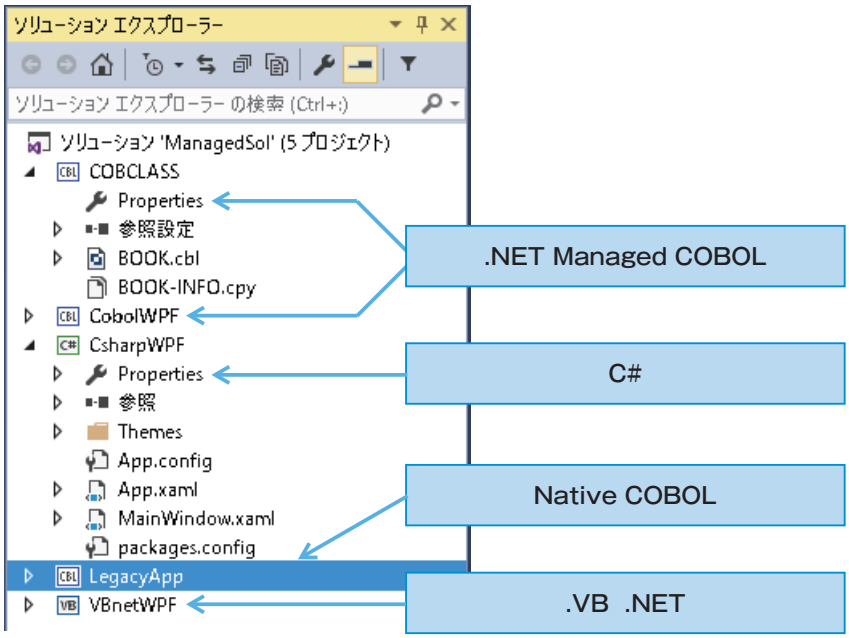
② 使用する MSBuild を設定
Jenkins の管理画面にて [システムの設定] をクリック



[MSBuild] 欄にて使用する MSBuild を指定。ここでは Visual Studio 2015 にバンドルされる MSBuild を指定しています。

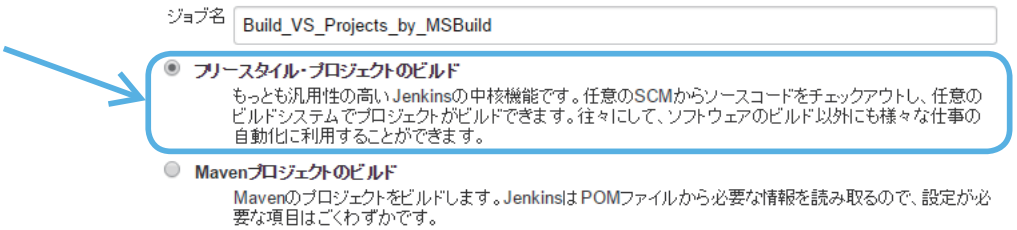


③ ビルド対象のアプリケーションを含む Visual Studio プロジェクトを用意
ここでは、Native COBOL、.NET Managed COBOL、C#、VB.NET のプロジェクトを含むソリューションを用意しました :



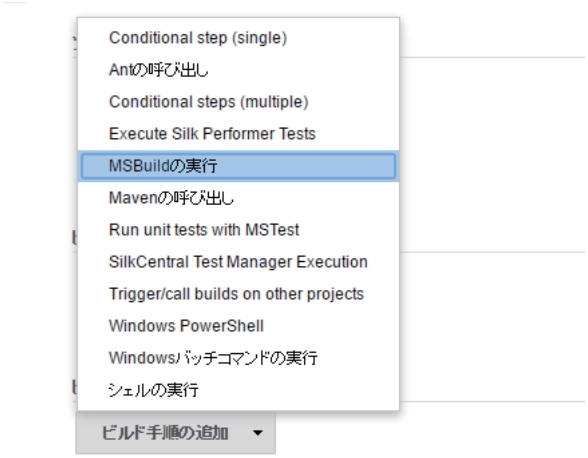
④ Jenkins にてビルド用のジョブを用意
[新規ジョブ作成] をクリック

[フリースタイル・プロジェクトのビルド] を選択の上、ジョブ名を命名

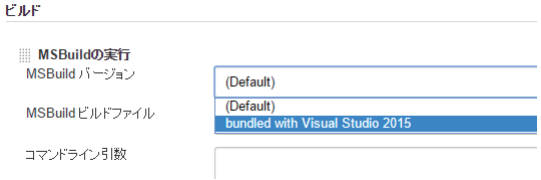


[OK] ボタンを押下

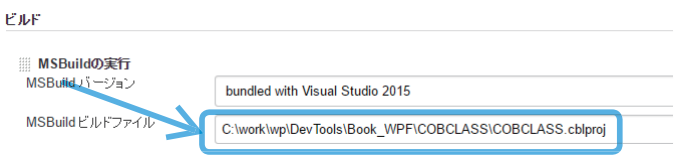
⑤ ジョブに MSBuild によるビルド命令を追加
[ビルド手順の追加] にて [MSBuild の実行] を選択



[MSBuild バージョン] 欄では②で用意した MSBuild 設定を選択



[MSBuild ビルドファイル] 欄に Visual Studio の COBOL プロジェクトフォルダ配下にあるプロジェクトファイル ＜プロジェクト名＞.cblproj を指定



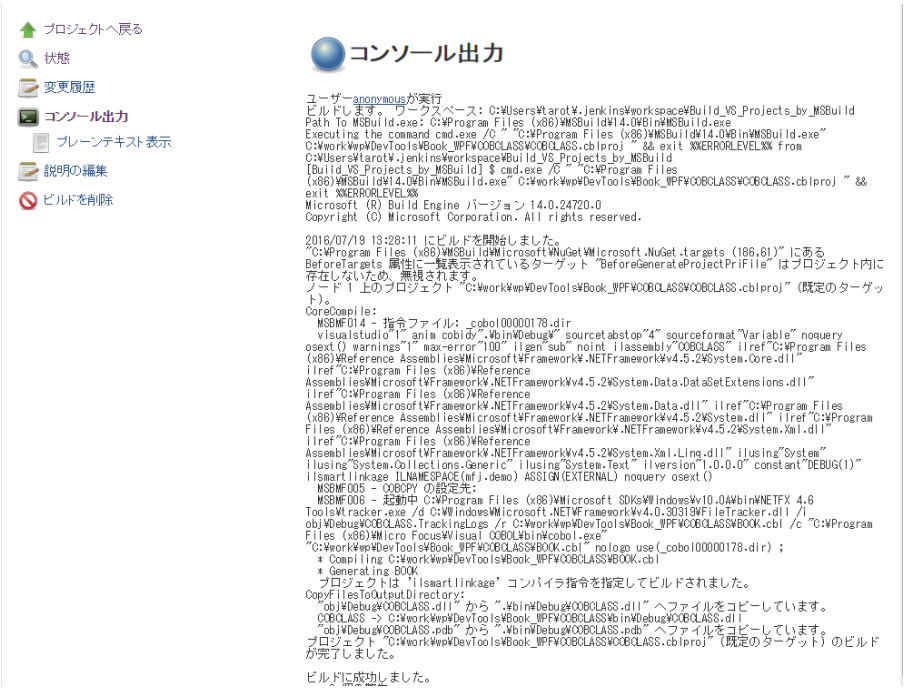
[保存] ボタンを押下

⑥ Visual Studio プロジェクトのビルドを実行

用意した Jenkins のプロジェクトにて [ビルド実行] をクリック

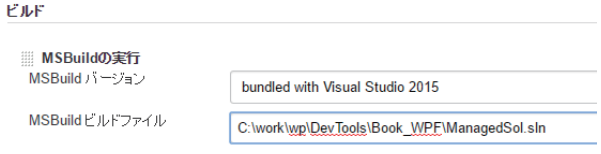
正常にビルドされると [ビルド履歴] 欄にて青いアイコンが表示されます。（失敗した場合は、赤いアイコンが表示されます。）

ビルド番号をクリックの上、[コンソール出力] をクリックするとビルド時のコンソール出力を確認できます：



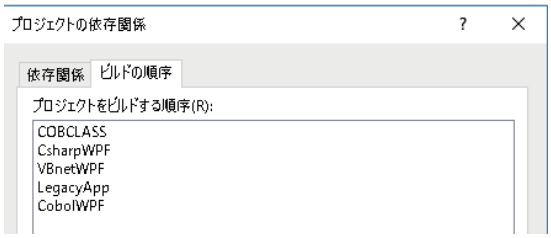
⑦ MSBuild のビルドターゲットにソリューションを指定し、COBOL や他の .NET 言語のプロジェクトをまとめてビルド

⑤で構成した Jenkins のビルド設定画面にて [MSBuild ビルドファイル] 欄にプロジェクトファイルではなく .sln の拡張子を持つソリューションファイルを指定：



再度 Jenkins から⑥の要領でビルドを実行

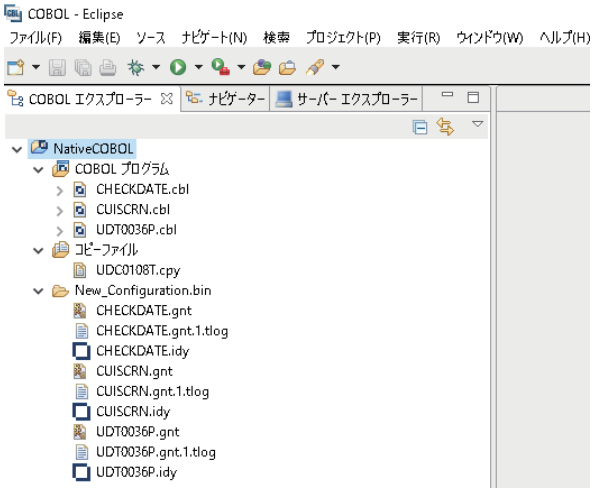
ソリューションに設定されたプロジェクト間の依存関係に基づいたビルド順序に沿ってソリューション配下のプロジェクトがビルドされます。
ソリューションに設定されたビルド順序のイメージ：



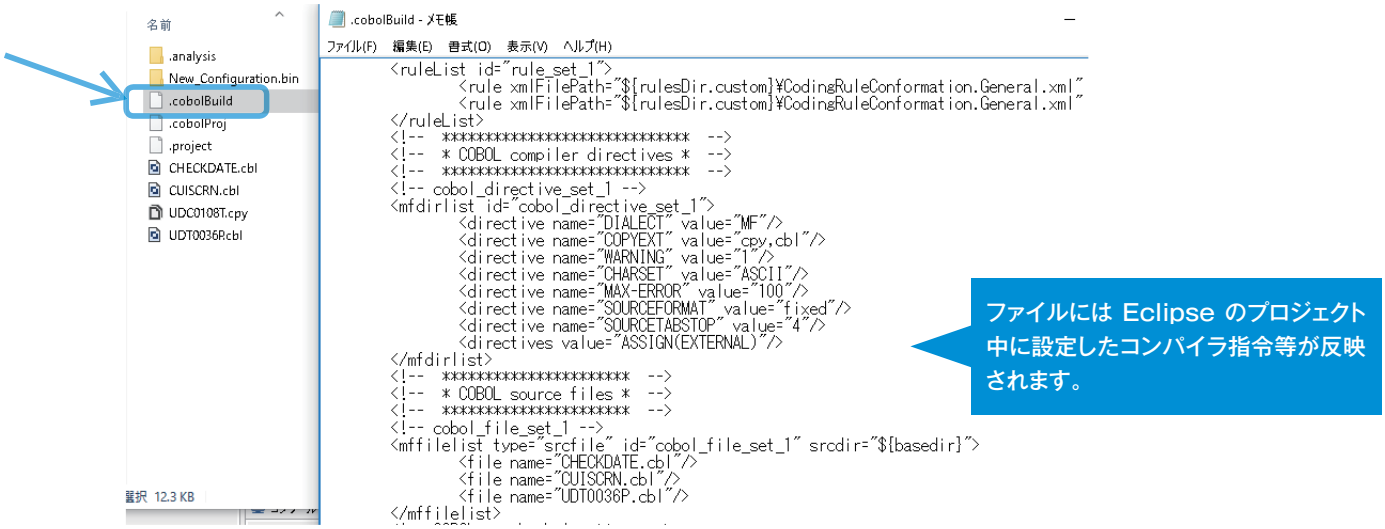
2.2 Ant による Eclipse プロジェクトのビルド

Ant は Apache オープンソースプロジェクトが提供する Java ベースのビルドツールです。Eclipse IDE 上で作成された Visual COBOL のプロジェクトはこの Ant を使ってプロジェクトをビルドします。Ant のビルドファイル .cobolBuild はプロジェクトファイル .cobolProj より自動生成されます。Java のプロジェクトのようにビルドファイルをプロジェクトからエクスポートして生成させる必要はありません。Visual COBOL は Eclipse 上で作成した COBOL プロジェクトをバッチやシェルを始めとする Ant を利用可能なツールから自動実行させることも想定して設計されています。前項で紹介した CI ツール Jenkins にも Ant によるビルドインターフェースが用意されています。ここでは、Visual COBOL for Eclipse のプロジェクトより自動生成された Ant のビルドファイルを Jenkins よりビルド実行させるイメージを紹介します。

① Visual COBOL for Eclipse のプロジェクトを用意

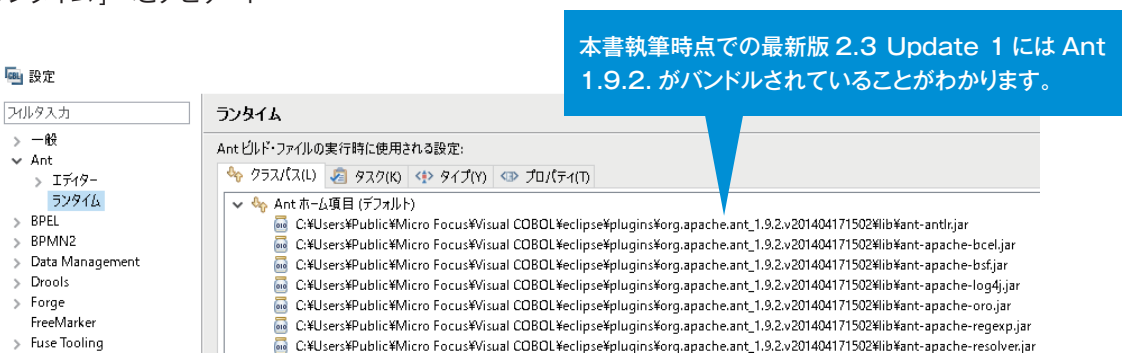


プロジェクトを作成すると自動で Ant ビルドファイル .cobolBuild が生成されます：



② Visual COBOL for Eclipse にバンドルされる Ant のバージョンを確認
Eclipse にて

[ウィンドウ] メニュー > [設定]
で表示される画面にて
[Ant] > [ランタイム] へとナビゲート

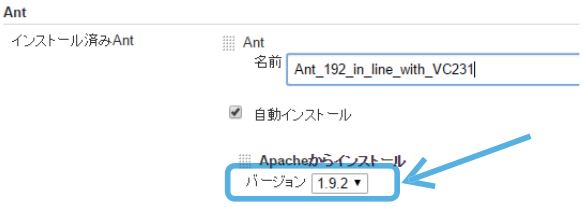


③ Jenkins に②で確認したバージョンの Ant を指定
[Jenkins の管理] をクリック

[システムの設定] をクリック

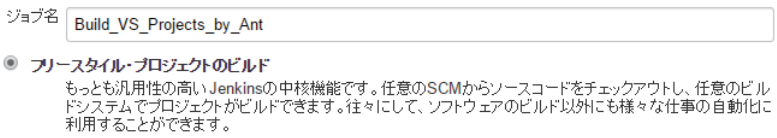
[Ant 追加] ボタンを押下 :

[バージョン] 欄にて②で確認したバージョンを指定 :



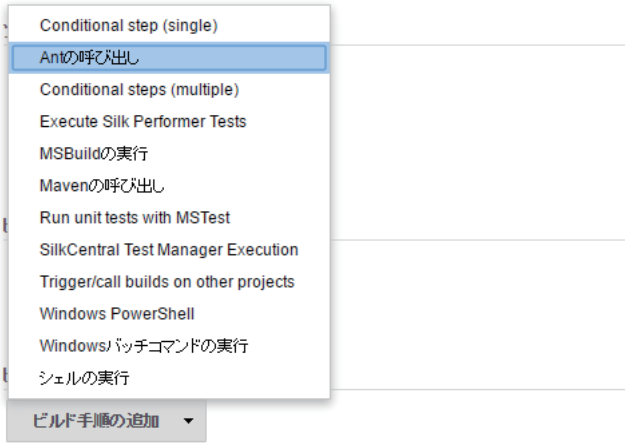
④ Jenkins にてビルド用のジョブを用意
[新規ジョブ作成] をクリック

ジョブ名を命名の上、[フリースタイル・プロジェクトのビルド] を選択



⑤ ジョブに Ant によるビルド命令を追加

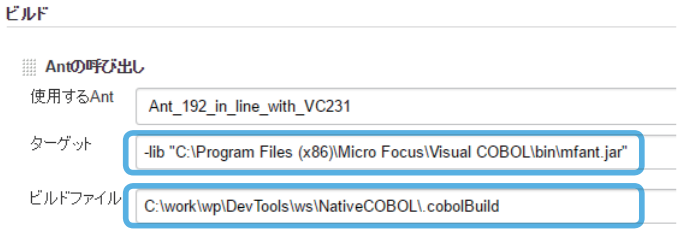
[ビルド手順の追加] にて [Ant の呼び出し] を選択



[使用する Ant] 欄にて③で用意した Ant を選択

[高度な設定] ボタンを押下

[ターゲット] 欄には -lib <Visual COBOL のインストールフォルダ >\bin\mfAnt.jar を、[ビルドファイル] 欄には Eclipse プロジェクトフォルダ配下にある Ant のビルドファイルを指定 :



[保存] ボタンを押下

⑥ Ant をインストール

用意した Jenkins のプロジェクトを一度実行させ、Ant をダウンロードさせます。

[ビルド実行] をクリック

Visual COBOL を使うための環境変数が指定されていないため、コンパイルできず、コンパイル処理の部分でエラーが返ります：

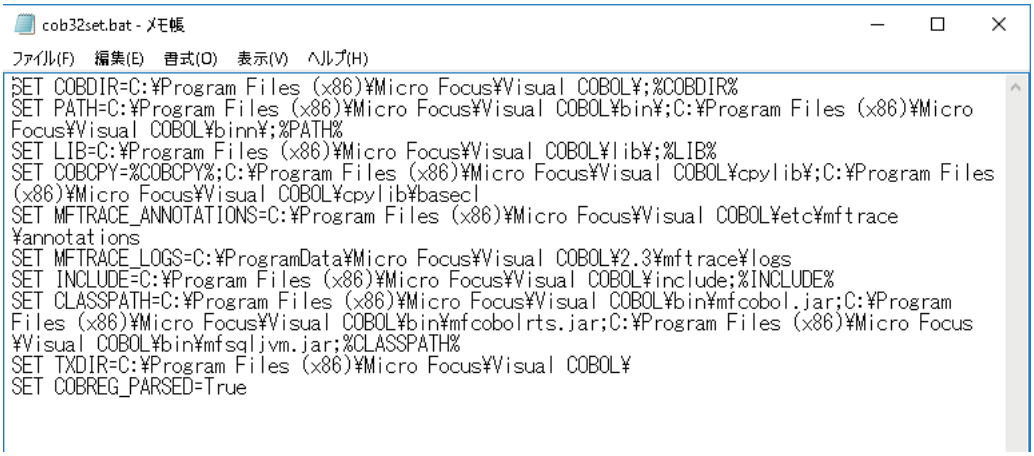
```
cobol.compile.cfg.New_Configuration:
    [cobol] Compiling CHECKDATE.cbl...
    [cobol]
    [cobol] An unexpected error was encountered when executing the COBOL compile task
    [cobol] Compilation complete with 2 errors, 0 warnings, 0 notices and an exit code of 12
    [cobol] Compiling CUISCRN.cbl...
    [cobol]
    [cobol] An unexpected error was encountered when executing the COBOL compile task
    [cobol] Compilation complete with 2 errors, 0 warnings, 0 notices and an exit code of 12
    [cobol] Compiling UDT0036P.cbl...
    [cobol]
    [cobol] An unexpected error was encountered when executing the COBOL compile task
    [cobol] Compilation complete with 2 errors, 0 warnings, 0 notices and an exit code of 12

cobol.link.cfg.New_Configuration:
```

⑦ Visual COBOL の利用に必要な環境変数を設定するバッチファイルを用意

Visual COBOL　コマンドプロンプトを開き、SET コマンドを実行

出力される環境変数をメモ

出力された環境を設定するバッチファイルを用意
バッチファイルの例：

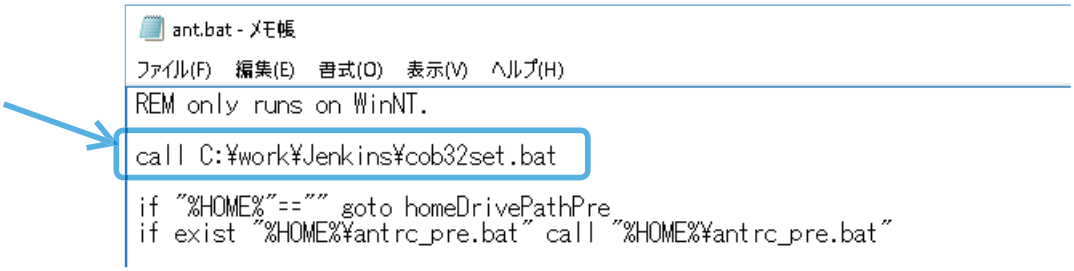


⑧ ant.bat に Visual COBOL を利用するための環境変数を指定

ant のインストールフォルダ配下の bin フォルダを開く（例えば、ログインユーザが「tarot」、③で指定した Ant の名前が「Ant_192_in_line_with_VC231」であれば
C:\Users\tarot\.jenkins\tools\hudson.tasks.Ant_AntInstallation\Ant_192_in_line_with_VC231
となります。）

フォルダ中に格納されている ant.bat をテキストエディタで開く

ant.bat ファイル中の先頭行に⑦で用意したバッチファイルを call する命令を埋め込み：



⑨ 再度 Jenkins より Eclipse プロジェクトをビルド
正しく機能することを確認します。
出力例：

```
cobol.cfg.New_Configuration:

cobol.compile.cfg.New_Configuration:
    [cobol] Compiling CHECKDATE.cbl...
    [cobol]
    [cobol] Micro Focus COBOL
    [cobol] Version 2.3.01142 Copyright (C) Micro Focus 1984-2016. All rights reserved.
    [cobol] * Checking complete with no errors - starting code generation
    [cobol] * Generating C:¥work¥wp¥DevTools¥ws¥NativeCOBOL¥New_Configuration.bin¥CHECKDATE
    [cobol] * Data:          408     Code:          736     Literals:          132
    [cobol] Compilation complete with no errors.
    [cobol] Compiling CUISCRN.cbl...
    [cobol]
    [cobol] Micro Focus COBOL
    [cobol] Version 2.3.01142 Copyright (C) Micro Focus 1984-2016. All rights reserved.
    [cobol] * Checking complete with no errors - starting code generation
    [cobol] * Generating C:¥work¥wp¥DevTools¥ws¥NativeCOBOL¥New_Configuration.bin¥CUISCRN
    [cobol] * Data:         1944     Code:         1864     Literals:         1392
    [cobol] Compilation complete with no errors.
    [cobol] Compiling UDT0036P.cbl...
    [cobol]
    [cobol] Micro Focus COBOL
    [cobol] Version 2.3.01142 Copyright (C) Micro Focus 1984-2016. All rights reserved.
    [cobol] * Checking complete with no errors - starting code generation
    [cobol] * Generating C:¥work¥wp¥DevTools¥ws¥NativeCOBOL¥New_Configuration.bin¥UDT0036P
    [cobol] * Data:          680     Code:         4216     Literals:          392
    [cobol] Compilation complete with no errors.

cobol.link.cfg.New_Configuration:
```

3 Test Coverage – COBOL 専用のカバレッジ分析ツール

ホワイトボックステストの実施にあたり、テスト網羅性の計測はテスト品質を評価する上で重要な指標となり得ます。Micro Focus は Visual COBOL よりも前の世代の製品から COBOL 専用のカバレッジ分析ツール「Test Coverage」を提供しています。本ツールを使えば COBOL ソース中の各文が各テストのポイントで通過した数を記録しレポート表示できます。

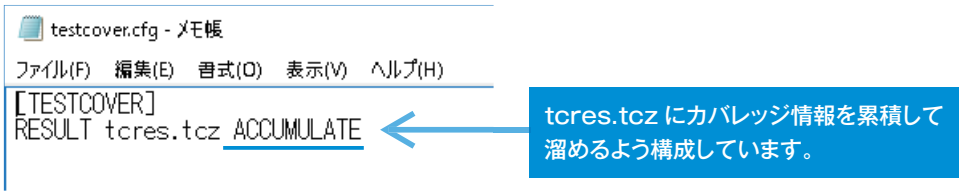
3.1 Test Coverage の基本動作

Test Coverage は TESTCOVER コンパイラ指令を指定してコンパイルされたモジュールに対してカバレッジを取得します。カバレッジの分析動作については TESTCOVER 環境変数でポイントする構成ファイルにて構成します。取得されたカバレッジ情報は .tcz の拡張子を持つバイナリファイルに溜められます。このファイルに書き出されたデータは tcreport コマンドを使ってテキストもしくは HTML 形式のレポートに出力します。この一連の処理については特に特別な GUI ツール等を介していないため、バッチやシェル等で自動化も可能です。つまり前章で紹介したような CI ツールから実行することも可能です。
以下に簡単なサンプルを用いた Test Coverage の実行例を記します：

① TESTCOVER コンパイラ指令を指定して対象のプログラムをコンパイル

```
C:¥work¥wp¥DevTools¥TestCover>cobol ussteel.cbl gnt TESTCOVER;
Micro Focus COBOL
Version 2.3.01142 Copyright (C) Micro Focus 1984-2016. All rights reserved.
* Checking complete with no errors - starting code generation
* Generating ussteel
* Data:        4448     Code:       25448     Literals:        4220

C:¥work¥wp¥DevTools¥TestCover>_
```

② Test Coverage の構成ファイルを準備



準備したファイルは環境変数 TESTCOVER でポイントします：

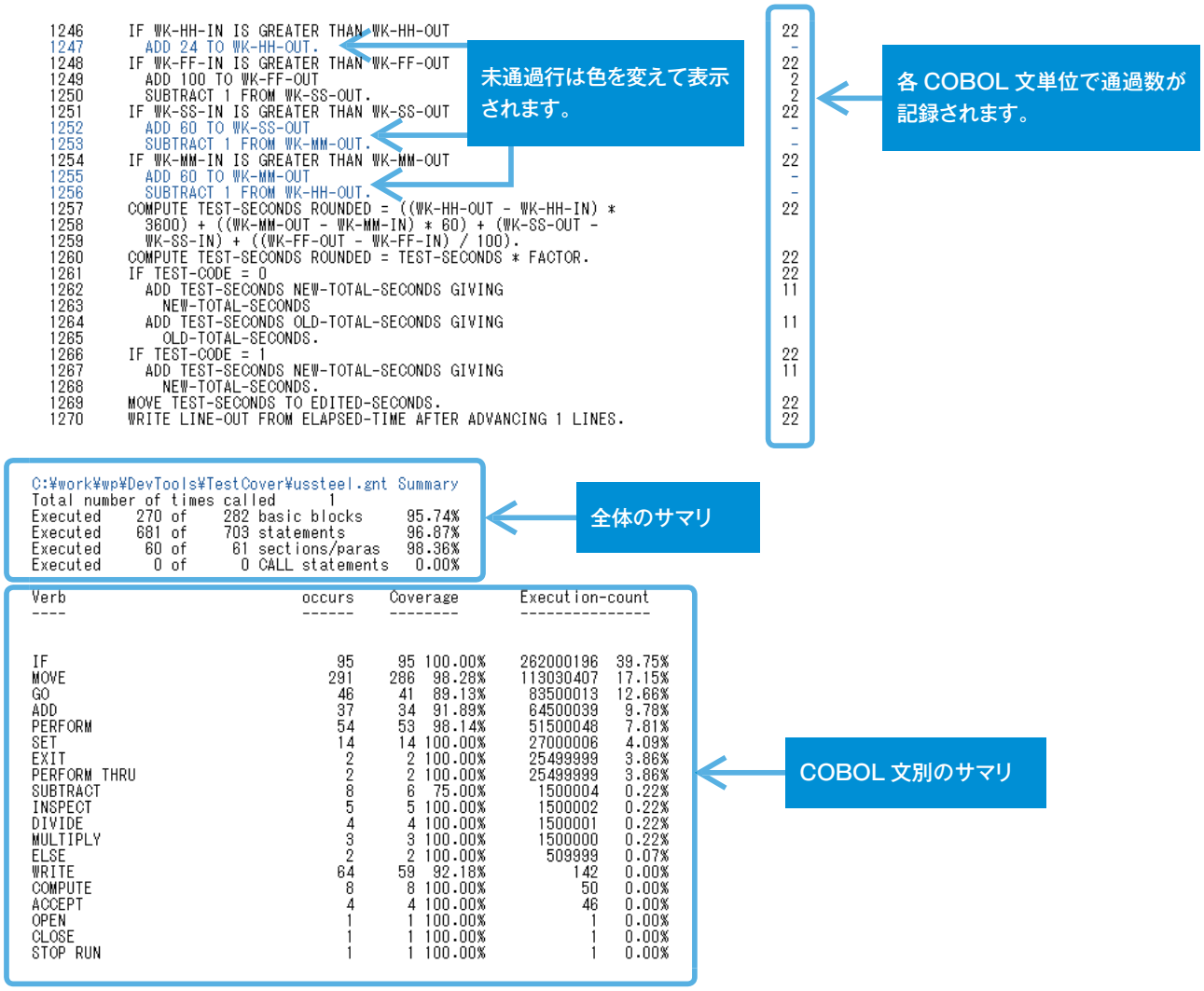
```
C:¥work¥wp¥DevTools¥TestCover>set TESTCOVER=C:¥work¥wp¥DevTools¥TestCover¥testcover.cfg

C:¥work¥wp¥DevTools¥TestCover>_
```

③ アプリケーションを実行
④ カバレッジ情報を HTML 形式のレポートに成形

```
C:¥work¥wp¥DevTools¥TestCover>tcreport tcres.tcz browse html verbs
Program ussteel.gnt processed normally
Listing of reports produced is in:
C:¥work¥wp¥DevTools¥TestCover¥TCIndex.htm

C:¥work¥wp¥DevTools¥TestCover>_
```

browse: レポート生成後、ブラウザを起動させるオプション
html: HTML 形式のレポートを生成させるオプション
verbs: COBOL 文のサマリをレポートに含めるオプション

生成されたレポートのイメージ：




```
1246      IF WK-HH-IN IS GREATER THAN WK-HH-OUT                                22
1247        ADD 24 TO WK-HH-OUT.                                               -
1248      IF WK-FF-IN IS GREATER THAN WK-FF-OUT                                22
1249        ADD 100 TO WK-FF-OUT                                               2
1250        SUBTRACT 1 FROM WK-SS-OUT.                                         2
1251      IF WK-SS-IN IS GREATER THAN WK-SS-OUT                                22
1252        ADD 60 TO WK-SS-OUT                                                -
1253        SUBTRACT 1 FROM WK-MM-OUT.                                         -
1254      IF WK-MM-IN IS GREATER THAN WK-MM-OUT                                22
1255        ADD 60 TO WK-MM-OUT                                                -
1256        SUBTRACT 1 FROM WK-HH-OUT.                                         -
1257      COMPUTE TEST-SECONDS ROUNDED = ((WK-HH-OUT - WK-HH-IN) *             22
1258        3600) + ((WK-MM-OUT - WK-MM-IN) * 60) + (WK-SS-OUT -
1259        WK-SS-IN) + ((WK-FF-OUT - WK-FF-IN) / 100).
1260      COMPUTE TEST-SECONDS ROUNDED = TEST-SECONDS * FACTOR.                22
1261      IF TEST-CODE = 0                                                     22
1262        ADD TEST-SECONDS NEW-TOTAL-SECONDS GIVING                          11
1263          NEW-TOTAL-SECONDS
1264        ADD TEST-SECONDS OLD-TOTAL-SECONDS GIVING                          11
1265          OLD-TOTAL-SECONDS.
1266      IF TEST-CODE = 1                                                     22
1267        ADD TEST-SECONDS NEW-TOTAL-SECONDS GIVING                          11
1268          NEW-TOTAL-SECONDS.
1269      MOVE TEST-SECONDS TO EDITED-SECONDS.                                 22
1270      WRITE LINE-OUT FROM ELAPSED-TIME AFTER ADVANCING 1 LINES.            22
```

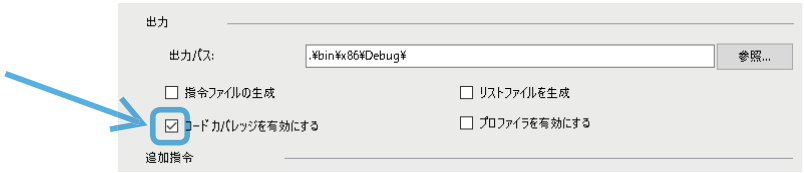
```
C:¥work¥wp¥DevTools¥TestCover¥ussteel.gnt Summary
Total number of times called      1
Executed    270 of    282 basic blocks     95.74%
Executed    681 of    703 statements       96.87%
Executed     60 of     61 sections/paras   98.36%
Executed      0 of      0 CALL statements   0.00%
```

Verb	occurs	Coverage	Execution-count	
IF	95	95 100.00%	262000196	39.75%
MOVE	291	286 98.28%	113030407	17.15%
GO	46	41 89.13%	83500013	12.66%
ADD	37	34 91.89%	64500039	9.78%
PERFORM	54	53 98.14%	51500048	7.81%
SET	14	14 100.00%	27000006	4.09%
EXIT	2	2 100.00%	25499999	3.86%
PERFORM THRU	2	2 100.00%	25499999	3.86%
SUBTRACT	8	6 75.00%	1500004	0.22%
INSPECT	5	5 100.00%	1500002	0.22%
DIVIDE	4	4 100.00%	1500001	0.22%
MULTIPLY	3	3 100.00%	1500000	0.22%
ELSE	2	2 100.00%	509999	0.07%
WRITE	64	59 92.18%	142	0.00%
COMPUTE	8	8 100.00%	50	0.00%
ACCEPT	4	4 100.00%	46	0.00%
OPEN	1	1 100.00%	1	0.00%
CLOSE	1	1 100.00%	1	0.00%
STOP RUN	1	1 100.00%	1	0.00%

3.2 IDE を通じた Test Coverage の操作

Visual COBOL を使う場合、Visual Studio や Eclipse IDE 上でも COBOL アプリケーションの開発が可能です。これらの IDE 上には COBOL 用に作りこまれたデバッガーが搭載されており、これらを使ってデバッグ実行して消化したテストのカバレッジを確認することも可能です。前項で紹介した方法でカバレッジを取得することももちろん可能ですが、Visual COBOL ではこれらの IDE 上で前項で紹介した設定内容を直感的な操作で構成し、カバレッジ結果を専用のビューで確認することも可能です [6]。以下は IDE 上における Test Coverage の操作イメージとなります [7]。

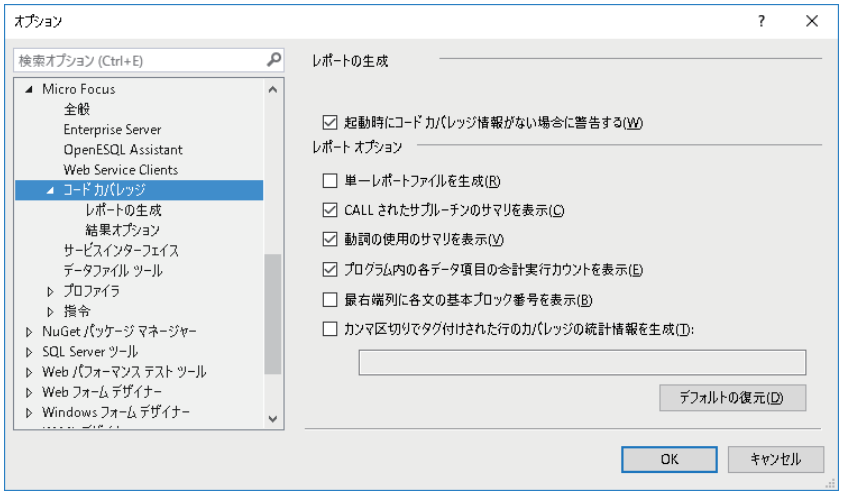
カバレッジの有効化／無効化の切り替え：



6 IDE 上での操作機能は Visual COBOL 2.3 より実装されています。

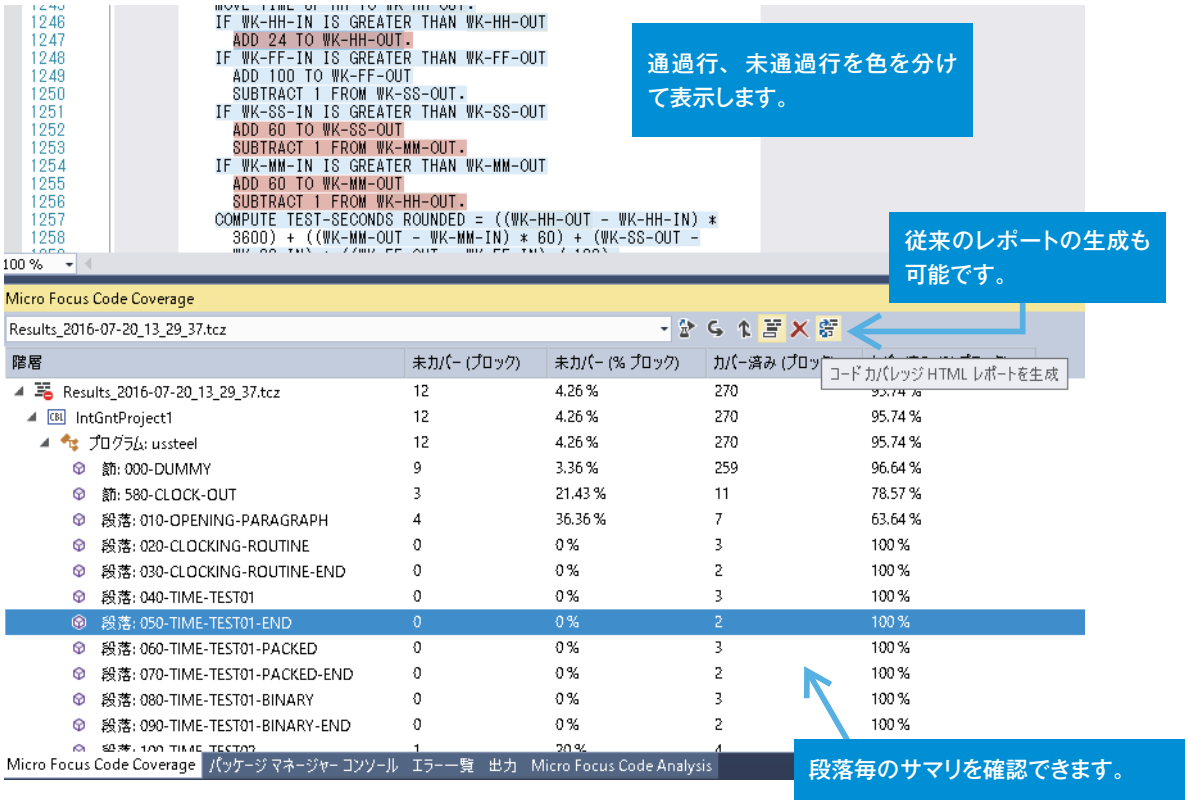
7 ここでは Visual Studio 2015 を使ったイメージを紹介してますが、Visual Studio の他のバージョン並びに Eclipse でも同様の操作で構成し、同様のビューを使って結果を閲覧することが可能です。

カバレッジ動作の構成：



Test Coverage にカバレッジを取得させつつアプリケーションを実行：

IDE 内でカバレッジの結果を確認：



4 コード分析 – COBOL 専用の静的コード解析ツール

プログラムのコンパイルが通れば、言語の規約に準じたコーディングができていることをある程度は担保できます。しかし、言語規約に従ったプログラムであってもプログラム中の問題が完全に排除されたことにはなりません。例えば、以下のような問題はこの場合でも混入し得ます:

> 潜在バグとなり得るようなコーディング
（例：未初期化の変数の利用、動作が規定されない命令の利用 等）

> 性能の観点上、非効率なコーディング
（例：ゾーン十進変数を使った算術演算、リトルエンディアン環境におけるビッグエンディアン変数の利用 等）

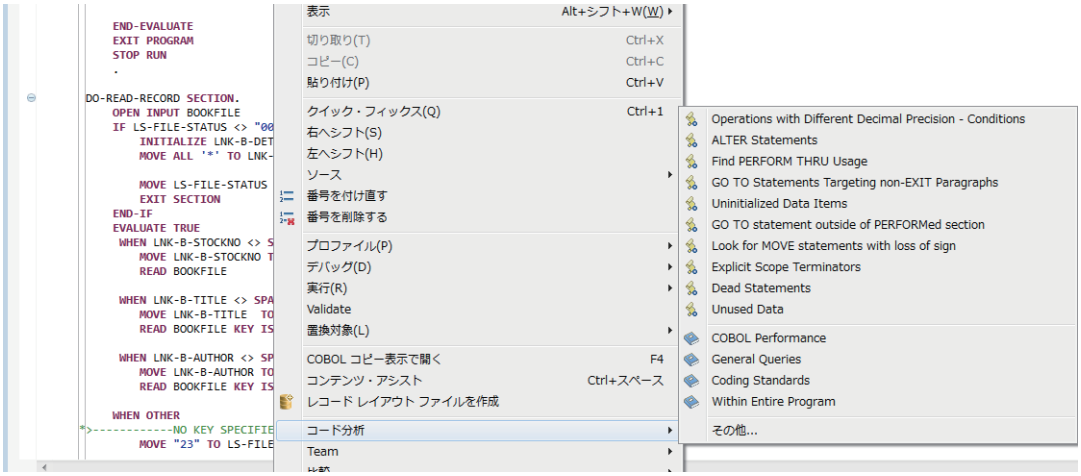
> ソースの可読性を低下させるようなコーディング
（例：デッドコード、GO TO 文の多用 等）

多くの場合、各開発プロジェクトでコーディング規準を設けてこのようなコーディングが行われないよう徹底します。このような管理作業はプログラムを実行せずに支援するツール「静的コード解析ツール」が Java や .NET 言語向けに多くのベンダーより提供されています。静的コード解析ツールはツールの性質上、一般的に対象の言語に合わせて作りこまれます。Micro Focus は長年積み上げてきた COBOL の知見を活かして COBOL 専用の静的コード解析機能を Visual COBOL に搭載しました。

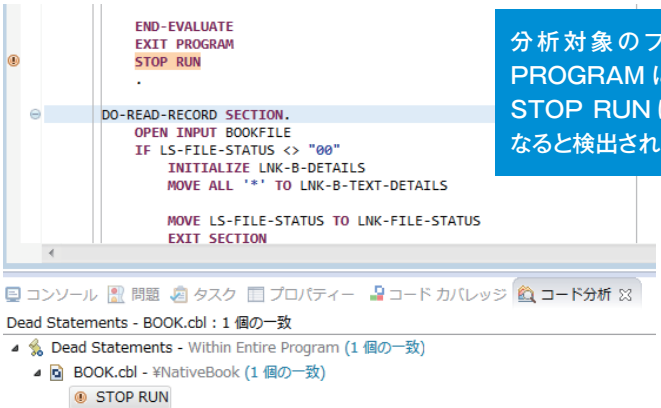
4.1 コード分析の基本動作

Visual COBOL が提供する静的コード解析機能「コード分析」は、Visual Studio IDE 並びに Eclipse IDE 上にて操作が可能です。Visual COBOL をインストール[8]しますと、エディターのコンテキストメニューからビルドインされたルールに基づいた分析を実行できます。

Eclipse IDE における分析処理の開始イメージ：



Dead Statements の実行例：

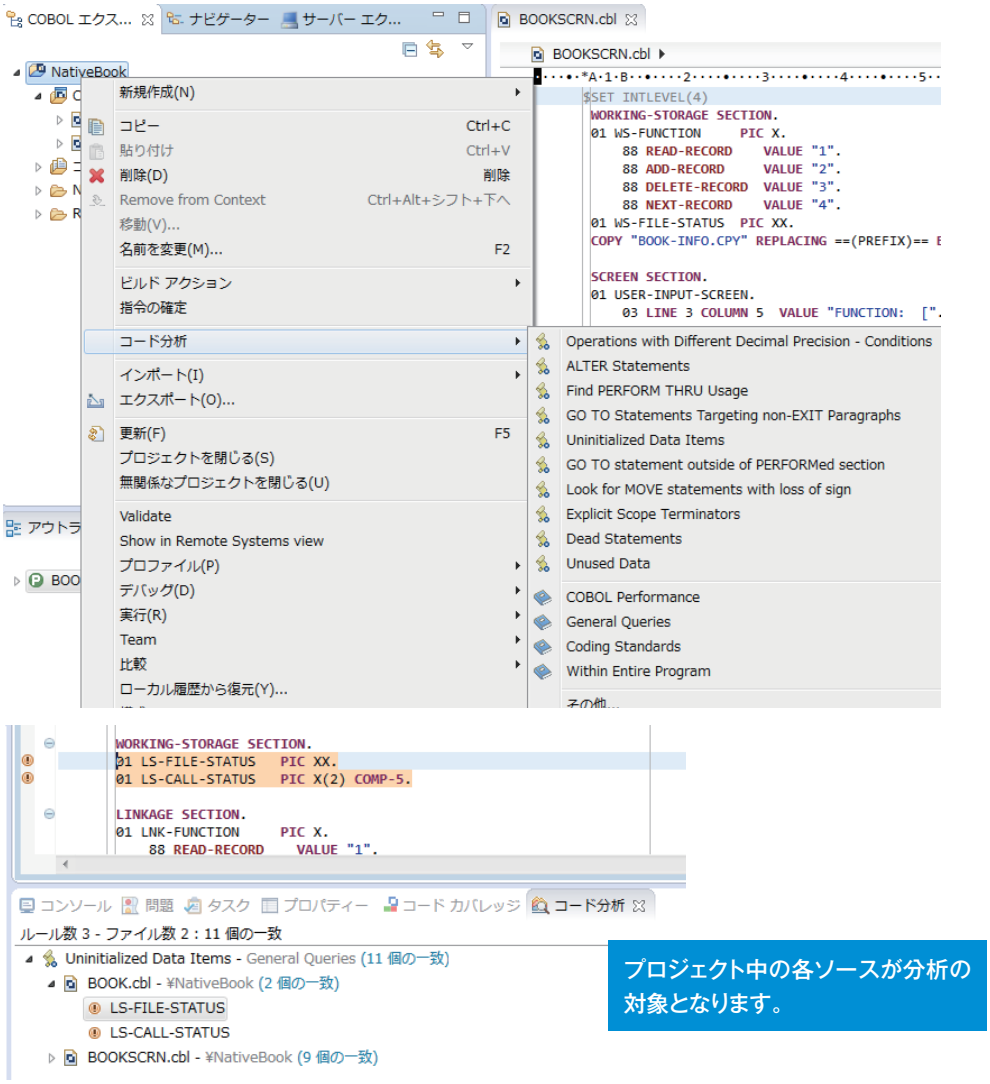


分析対象のプログラムはサブプログラムとなるため、EXIT PROGRAM にてプログラムは抜けます。そのため、次行の STOP RUN についてはフロー上未到達 (Dead Statement) になると検出されました。

8 コード分析機能は Visual COBOL 2.3 より実装された機能です。

分析対象は各ソース単位のみならず、Visual Studio のプロジェクトや Eclipse のプロジェクトへと対象を拡大することも可能です。

Eclipse のプロジェクト単位による分析処理イメージ：



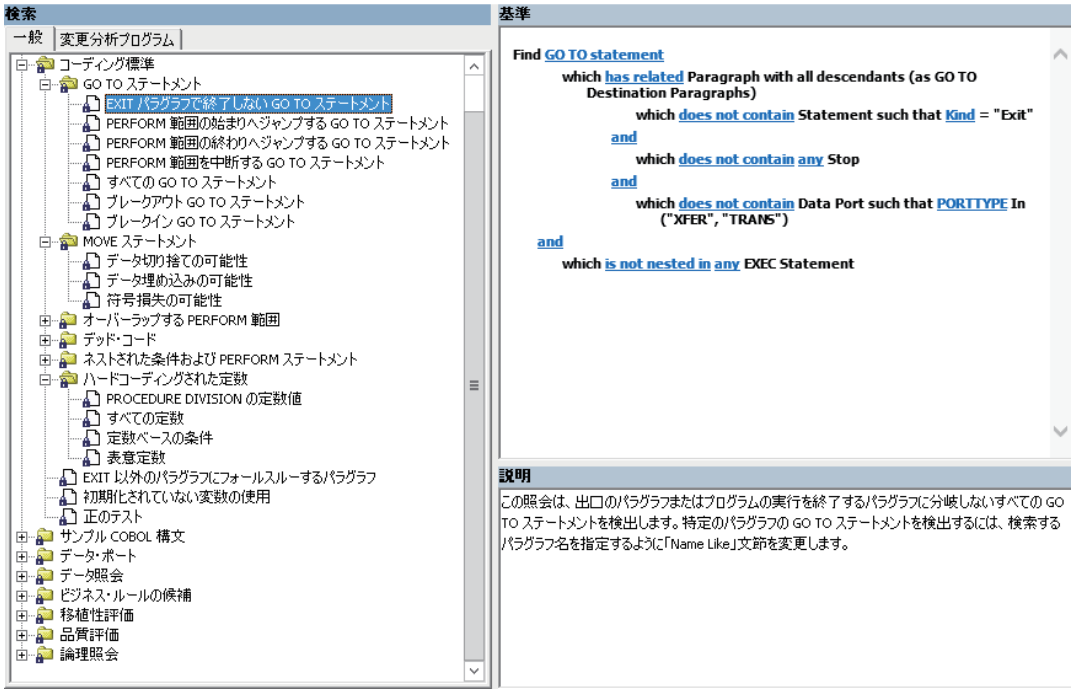
Visual COBOL が提供するコード分析機能はビルド実行後に自動で処理させる構成オプションを用意しています。これにより、コンパイルチェック並びに静的コード解析が済んだソースが常に得られるよう環境を整えることが可能です。

Eclipse プロジェクトにおけるプロジェクト設定イメージ：

4.2 Enterprise Analyzer 上のクエリールールの活用

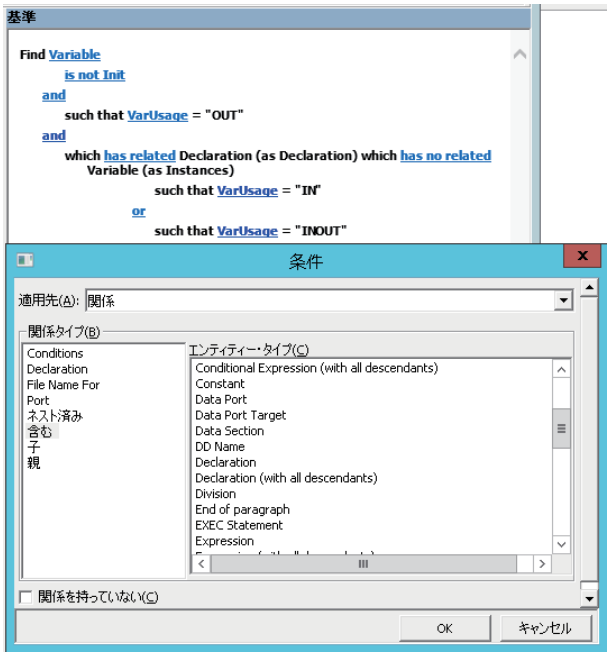
Visual COBOL にビルドインされたコード分析の解析ルールは一般的な開発プロジェクトで検討され得る基本的なルールが取り揃えられています。しかし、実際の開発現場で用意されるコーディング規準はこのルールだけに留まらないはずです。このような要件をも満たすための製品として Micro Focus は Enterprise Analyzer[9] という製品を開発・販売しています。この Enterprise Analyzer には Visual COBOL のコード分析機能よりも広い観点で解析ルールが取り揃えられています。

Enterprise Analyzer にビルドインされたクエリールールのイメージ：



Enterprise Analyzer にデフォルトで用意されていない観点で解析したい場合は、ルールをカスタマイズ作成することも可能です。

カスタマイズ作成画面イメージ：



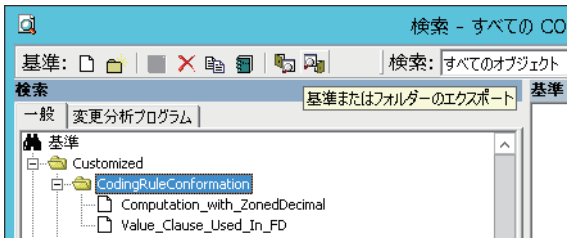
9 Enterprise Analyzer は COBOL ソースのコード解析以外にも様々な機能が搭載されています。本製品の詳細については下記ページをご覧ください。同ページではデータシートやデモ動画にて製品概要を紹介しています。
http://www.microfocus.co.jp/products/modernization/enterprise-analyzer/

カスタマイズ作成されたルールを含め Enterprise Analyzer 上のルールは Visual COBOL と共有することが可能です。即ち、例えば下記のようなシナリオの実現も可能です：

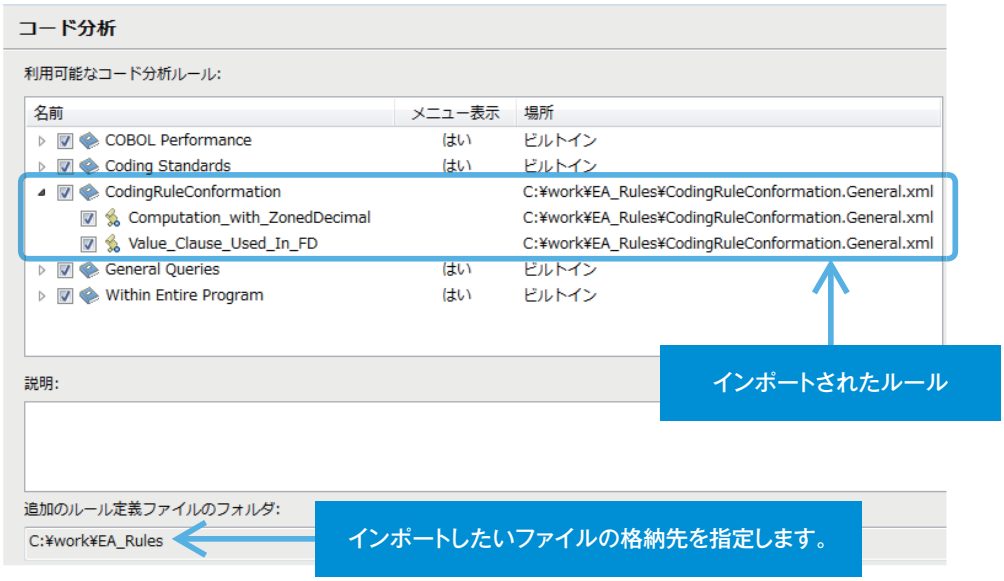
① 開発プロジェクトリーダーが Enterprise Analyzer を所有し、コーディング規準に基づいたルールを用意
② Enterprise Analyzer で用意されたルールをエクスポート
③ エクスポートしたルールを Visual COBOL のライセンスを持った開発メンバへ配布
④ 開発メンバは Enterprise Analyzer のルールを自身の開発環境に取り込みコーディング規準に従いコーディング

以下に Enterprise Analyzer 上でカスタマイズ作成したルールを Visual COBOL に取り込む手順を記します。

カスタマイズ作成したルールをエクスポート：



エクスポートされたルールを Visual COBOL for Eclipse で取り込み：



4.3 MSBuild, Ant による実行

Visual COBOL のコード分析機能は各プログラムソースに対して分析用の一時ファイルをディスク上に作成の上、分析処理します。そのため、分析対象のソースのボリュームによっては大幅な処理時間を要する可能性も否めません。このような状況下でビルドの度に分析処理を流すことは非効率であり、生産性の低下を招く恐れがあります。Visual COBOL は先の章で紹介したビルドツール MSBuild 及び Ant 経由でコード分析処理を発行するインターフェースを備えています。即ち、CI ツールによるコード分析処理の発行も可能なため、ビルドが済んだ時点でコード分析処理を自動でキックさせる仕組みを用意することも可能です。

以下に Jenkins 経由でコード分析を処理させる設定イメージを紹介します。

Ant を使って Eclipse COBOL プロジェクトをコード分析させる設定例 [10]：

10 Ant 経由で命令を発行する場合、Eclipse プロジェクトに設定されたルールに基づいて分析をします。MSBuild を使う場合は、Visual Studio に設定されたルールもしくは MSBuild にオプションとして渡すルールを実行するか選択可能です。

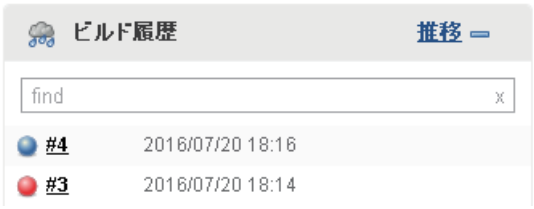
ルールに基づいたクエリで問題が検出された場合の出力例：

```
perform.analysis.cfg.New_Configuration:
 [analysis] Micro Focus Enterprise Analyzer Command - Line Tool, Version 3.6.1
 [analysis] Copyright (C) Micro Focus 2015, All Rights Reserved.
 [analysis] Analysis started.
 [analysis] Analyzing CHECKDATE ...
 [analysis] Analyzing CUISCRN ...
 [analysis] Analyzing UDT0036P ...
 [analysis] C:¥work¥wp¥DevTools¥ws¥NativeCOBOL¥UDT0036P.cbl(223,28,223,42): error MFCA0001:
CodingRuleConformation: Computation_with_ZonedDecimal: WK-DEPRECIATION
 [analysis] C:¥work¥wp¥DevTools¥ws¥NativeCOBOL¥UDT0036P.cbl(226,28,226,42): error MFCA0001:
CodingRuleConformation: Computation_with_ZonedDecimal: WK-DEPRECIATION
 [analysis] C:¥work¥wp¥DevTools¥ws¥NativeCOBOL¥UDT0036P.cbl(227,37,227,51): error MFCA0001:
CodingRuleConformation: Computation_with_ZonedDecimal: WK-DEPRECIATION
 [analysis] C:¥work¥wp¥DevTools¥ws¥NativeCOBOL¥UDT0036P.cbl(231,28,231,42): error MFCA0001:
CodingRuleConformation: Computation_with_ZonedDecimal: WK-DEPRECIATION
 [analysis] C:¥work¥wp¥DevTools¥ws¥NativeCOBOL¥UDT0036P.cbl(232,37,232,51): error MFCA0001:
CodingRuleConformation: Computation_with_ZonedDecimal: WK-DEPRECIATION
 [analysis] Analysis Complete.
 [analysis]
 [analysis] Done: 00:00:05
 [analysis] Finished: FAILURE
   [delete] Deleting directory C:¥work¥wp¥DevTools¥ws¥NativeCOBOL¥.analysis¥data

BUILD FAILED
C:¥work¥wp¥DevTools¥ws¥NativeCOBOL¥.cobolBuild:242:
C:¥work¥wp¥DevTools¥ws¥NativeCOBOL¥.cobolBuild:249: The following error occurred while
executing this line:
C:¥work¥wp¥DevTools¥ws¥NativeCOBOL¥.cobolBuild:145: Analysis failure

Total time: 9 seconds
Build step 'Antの呼び出し' marked build as failure
Finished: FAILURE
```

ルールに基づいたクエリで問題が検出されなかった場合の出力例：

```
perform.analysis.cfg.New_Configuration:
 [analysis] Micro Focus Enterprise Analyzer Command - Line Tool, Version 3.6.1
 [analysis] Copyright (C) Micro Focus 2015, All Rights Reserved.
 [analysis] Analysis started.
 [analysis] Analyzing CHECKDATE ...
 [analysis] Analyzing CUISCRN ...
 [analysis] Analyzing UDT0036P ...
 [analysis] Analysis Complete.
 [analysis]
 [analysis] Done: 00:00:06
 [analysis] Finished: SUCCESS
   [delete] Deleting directory C:¥work¥wp¥DevTools¥ws¥NativeCOBOL¥.analysis¥data

BUILD SUCCESSFUL
Total time: 9 seconds
Finished: SUCCESS
```

上の結果と Jenkins のビルド結果が紐付きます：



5 mfunit - COBOL 専用の xUnit 系単体テストフレームワーク

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 [11]。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。
本章ではこの mfunit の基本動作について紹介いたします。

5.1 mfunit の基本機能

mfunit では以下のような手順でテストを進めます。

11 本機能は本書執筆時点では Technology Preview 扱いとなっており、サポート対象外の機能です。お試しになる際は、マニュアルをご参照いただき最新のステータスをご確認の上ご自身の判断でご利用ください。尚、本機能は Visual COBOL 2.3 より実装された機能です。

① Test Fixture の作成
Test Fixture(テストを実行、成功させるために必要な状態や前提条件の集合) を COBOL でコーディングして用意します。COBOL プログラム中に複数の Test Fixture を含めることも可能です。

② Test Fixture をコンパイル
Windows 版であればダイナミックリンクライブラリ (.dll) に、Linux/UNIX 版であれば呼び出し可能な共有オブジェクト (.so) にコンパイルします。

③ テストを実行
Windows 版であれば mfurun、Linux/UNIX 版であれば cobmfurun32/ cobmfurun64 が mfunit の Test Runner として用意されています [12]。

④ 結果を確認
結果は標準出力に出力されます。加えて、テキストファイルや JUnit 互換のある XML 形式で出力することも可能です。

ここでは簡便化のために用意した電卓の機能を処理するサブモジュールを mfunit を使って Windows 環境上で幾つか単体テストするようすを紹介します。

```
DATA DIVISION.
LINKAGE SECTION.
01 LNK-PARAM.
  03 LNK-OPERATOR          PIC 9.
  03 LNK-TERM1             PIC S9(4) COMP-5.
  03 LNK-TERM2             PIC S9(4) COMP-5.
  03 LNK-RES               PIC S9(6).
PROCEDURE DIVISION USING LNK-PARAM.
MAIN-PARA.
    EVALUATE LNK-OPERATOR
     WHEN 1
       PERFORM ADDITION-PARA
     WHEN 2
       PERFORM SUBTRACTION-PARA
     WHEN 3
       PERFORM MULTIPLICATION-PARA
     WHEN 4
       PERFORM DIVISION-PARA
    END-EVALUATE.
    GOBACK.
ADDITION-PARA.
   COMPUTE LNK-RES = LNK-TERM1 + LNK-TERM2
   EXIT.
SUBTRACTION-PARA.
   COMPUTE LNK-RES = LNK-TERM1 - LNK-TERM2
   EXIT.
MULTIPLICATION-PARA.
   COMPUTE LNK-RES = LNK-TERM1 * LNK-TERM2
   EXIT.
DIVISION-PARA.
   COMPUTE LNK-RES = LNK-TERM1 / LNK-TERM2
   EXIT.
```

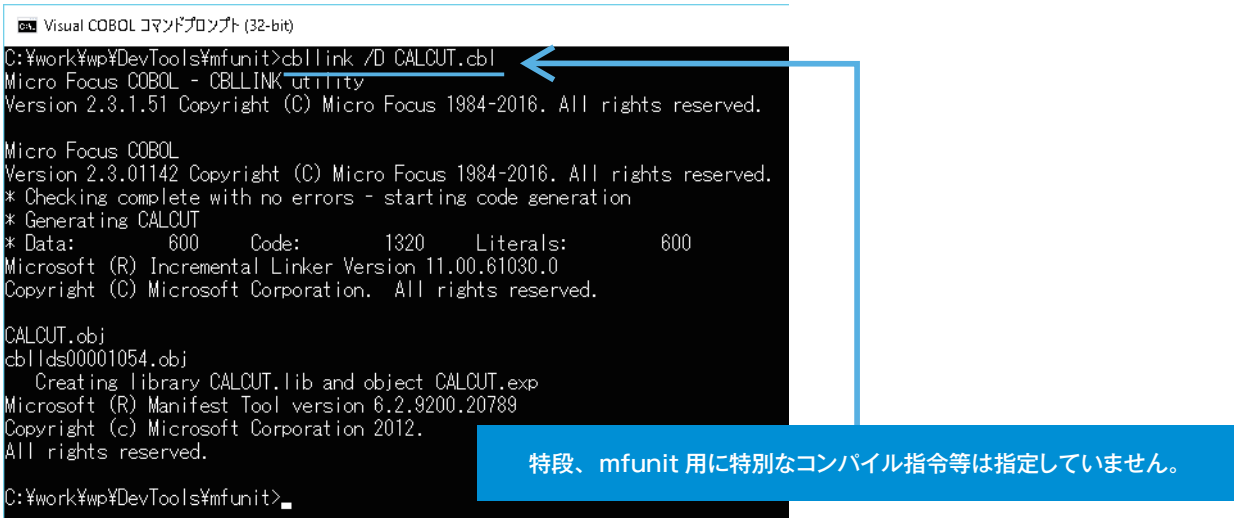
12 .NET 用の mfurunil 等 native 以外のアプリケーションがターゲットの場合はそれぞれを対象とした Test Runner が用意されています。詳細についてはマニュアルをご参照ください。

① Test Fixture を用意

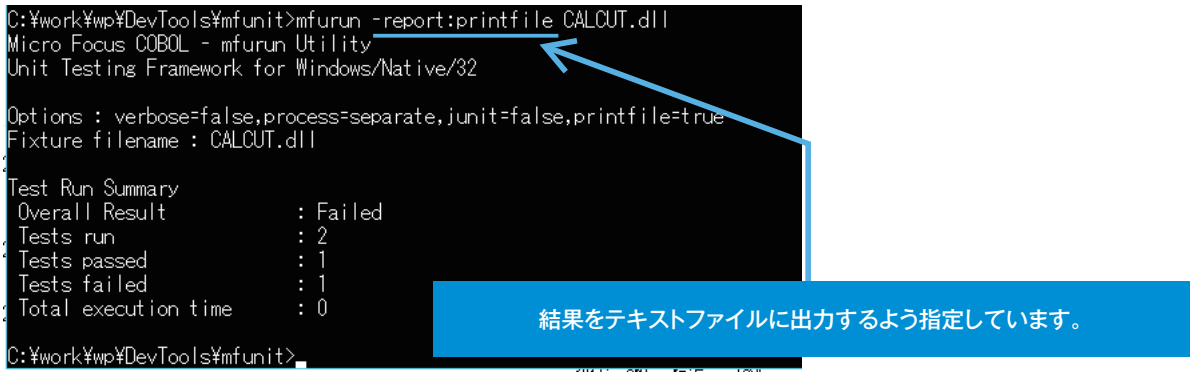
```
DATA DIVISION.
WORKING-STORAGE SECTION.
COPY "MFUNIT.CPY".
01 WS-FAIL-MSG.
  03 FILLER       PIC X(20) VALUE "EXPECTED VALUE WAS: ".
  03 WK-EXPECTED-VALUE       PIC X(06).
  03 FILLER       PIC X(23) VALUE ". BUT WHAT WE GOT WAS: ".
  03 WK-ACTUAL-VALUE         PIC X(06).
01 WK-PARAM.
  03 WK-OPERATOR             PIC 9 VALUE 1.
  03 WK-TERM1                PIC S9(4) COMP-5 VALUE 10.
  03 WK-TERM2                PIC S9(4) COMP-5 VALUE 2.
  03 WK-RES                  PIC S9(6) VALUE 0.
01 EXPECTED-VAL1             PIC S9(6) VALUE 12.
01 EXPECTED-VAL2             PIC S9(6) VALUE 7.
PROCEDURE DIVISION.
ENTRY MFU-TC-PREFIX & "PTN1".
   CALL "COBCALC" USING WK-PARAM.
   IF WK-RES NOT = EXPECTED-VAL1 THEN
     MOVE EXPECTED-VAL1       TO WK-EXPECTED-VALUE
     MOVE WK-RES              TO WK-ACTUAL-VALUE
     CALL "MFU_ASSERT_FAIL" USING
       BY REFERENCE WS-FAIL-MSG
       BY VALUE 65
     END-CALL
   END-IF.
   GOBACK.

ENTRY MFU-TC-PREFIX & "PTN2".
   MOVE 2 TO WK-OPERATOR.
   CALL "COBCALC" USING WK-PARAM.
   IF WK-RES NOT = EXPECTED-VAL2 THEN
     MOVE EXPECTED-VAL2       TO WK-EXPECTED-VALUE
     MOVE WK-RES              TO WK-ACTUAL-VALUE
     CALL "MFU_ASSERT_FAIL" USING
       BY REFERENCE WS-FAIL-MSG
       BY VALUE 65
     END-CALL
   END-IF.
   GOBACK.
```

ここでは、2つのテストケースを用意しました。テストケースは「MFUT_」から始まるエントリポイントで定義します[13]。1番目のテストケースは「10 + 2」をテスト対象のモジュールに処理させています。このテストケースでは mfunit 用の Assert ライブラリ「MFU_ASSERT_FAIL」にこの処理の結果は「12」になるはずだと期待値を指定しています。2番目のテストケースでは「10 - 2」を処理させ、敢えてテストが失敗するよう期待値には「7」を指定しています。

② ダイナミックリンクライブラリに Test Fixture をコンパイル

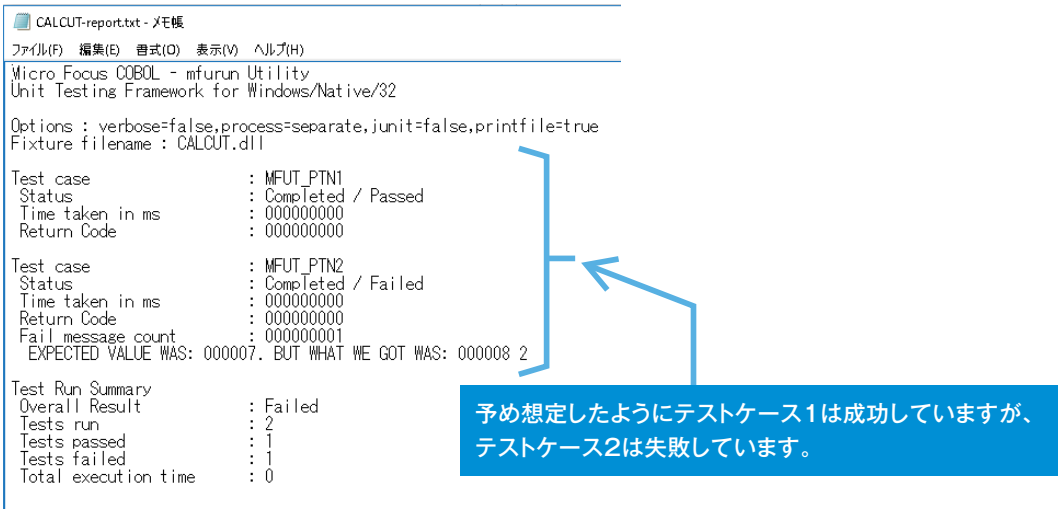


③ テストを実行



④ 結果を確認

テキストファイル形式で出力された結果を確認します：



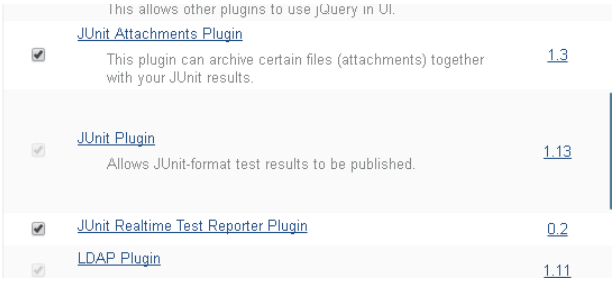
13 本例ではテストケース用のプリフィックスを Visual COBOL が提供する mfunit 用のコピーファイル MFUNIT.CPY 中の定数を通じて利用しています。

5.2 JUnit 互換形式の出力フォーマットの活用

mfunit は前項で紹介したようにテキスト形式で結果を出力することも可能ですが、JUnit 互換の XML 形式で結果を出力することも可能です。更に mfunit は先の章でも紹介した CI ツール Jenkins での使用を想定した結果を出力するオプションも用意しています。加えて Micro Focus Silk Central のようなテスト管理ツールとも連携させればより効率的にチーム内でテストを管理できます。
本項ではこれらのオプションを活用し Jenkins 経由で mfunit を実行し、結果を Jenkins 上で確認するようすを紹介します。

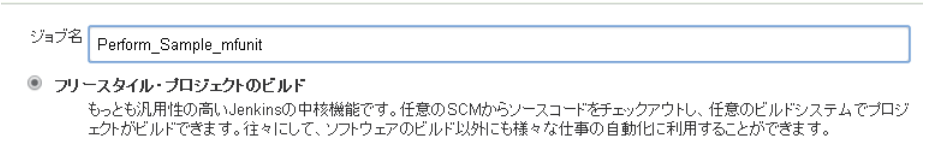
① JUnit 用のプラグインを Jenkins に追加

追加後のイメージ：



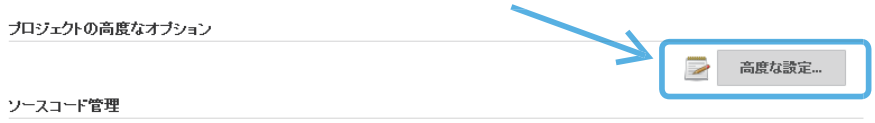
② mfunit を実行する Jenkins のプロジェクトを作成

フリースタイルのプロジェクトを用意：

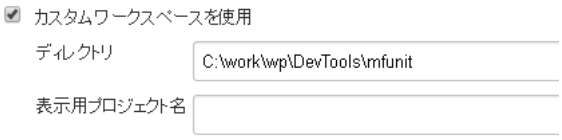


③ mfunit の Test Fixture が格納されたフォルダをプロジェクトディレクトリに指定

[プロジェクトの高度なオプション] 欄にて [高度な設定] ボタンを押下：



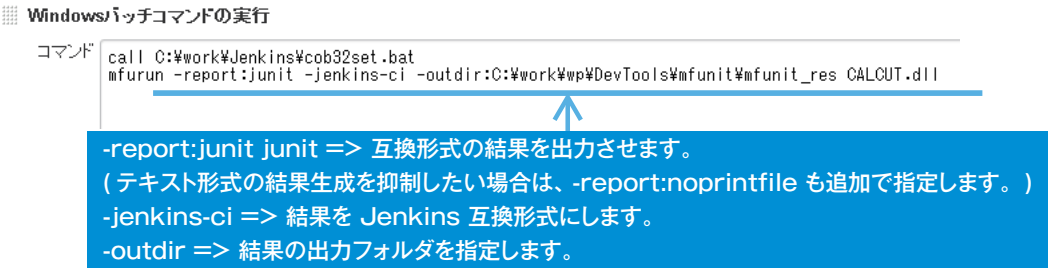
[カスタムワークスペースを使用] にチェックを入れ、[ディレクトリ] 欄に Test Fixture が格納されたフォルダを指定：



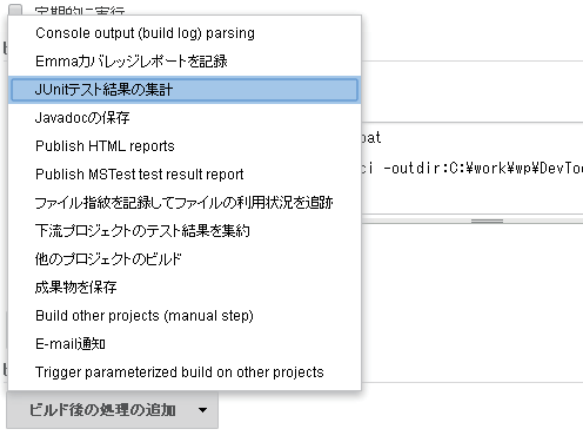
④ mfunit のテスト実行コマンドを追加

[ビルド手順の追加] にて [Windows バッチコマンドの実行] を選択：

mfunit のテストを実行するコマンドを追加：



⑤ テスト結果を JUnit 用のプラグラインを使って集計するよう構成
[ビルド後の処理の追加] にて [JUnit テスト結果の集計] を選択：



[テスト結果 XML] 欄に③で指定したディレクトリから mfurun の outdir で指定したディレクトリまでの相対パス及び「*.xml」を指定

[保存] ボタンを押下

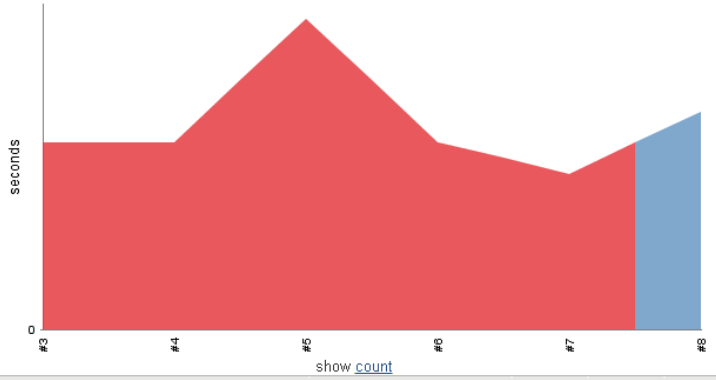
⑥ ビルドを実行
Jenkins のプロジェクトの処理実行後、履歴欄に表示される該当の番号をクリックすると「テスト結果」へのリンクが表示されます：

クリックするとテスト結果を専用のビューで確認できます：

「テスト履歴」をクリックするとこれまでのテストの履歴を確認できます。

下図は、テストケースを修正し一回実行した後に表示したイメージです：

テスト結果の履歴



show count

ビルド	テストの説明	テスト所要時間	失敗	スキップ	合計
Perform_Sample_mfunit #8		7 ms	0	0	2
Perform_Sample_mfunit #7		5 ms	1	0	2
Perform_Sample_mfunit #6		6 ms	1	0	2
Perform_Sample_mfunit #5		10 ms	1	0	2
Perform_Sample_mfunit #4		6 ms	1	0	2
Perform_Sample_mfunit #3		6 ms	1	0	2

6 Profiler - COBOL 専用のパフォーマンス分析ツール

Profiler は Visual COBOL が提供する COBOL 専用のパフォーマンス分析ツールです。COBOL の節や段落単位で処理に要した時間、節や段落へのエントリ回数、エントリ毎の平均処理時間等の統計情報を取得します。プログラムのパフォーマンスが許容基準を超える場合に、ボトルネック検出するのに非常に有用なツールです。

6.1 コマンドによる処理

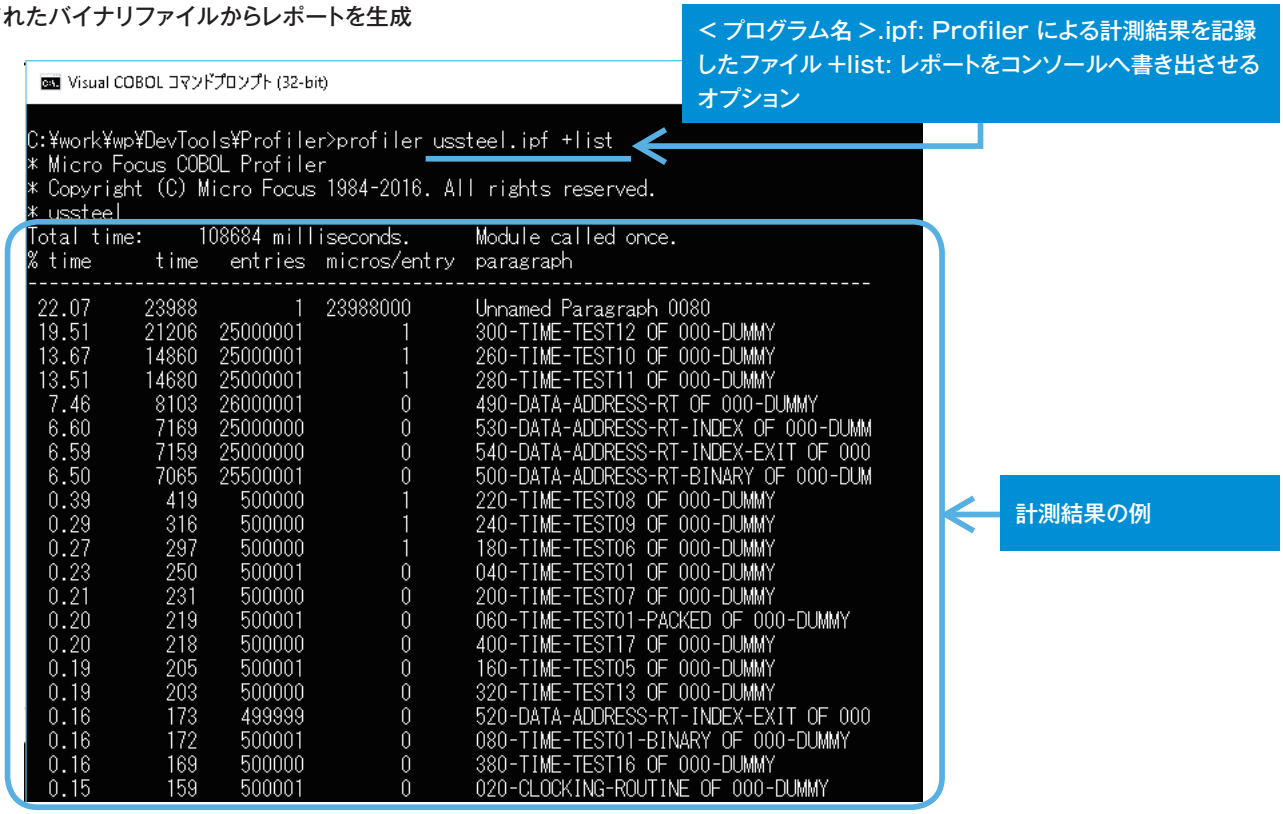
PROFILE コンパイラ指令を指定してコンパイルしたモジュールは Profiler による分析の対象となります。Profiler は分析対象のモジュールが実行されると計測情報をバイナリファイルに書き溜めます。計測情報をテキスト形式のレポートに成形するには profiler コマンドを利用します。Test Coverage と同様に本ツールについてもバッチやシェル等による自動化も可能です。
以下に簡単なサンプルを用いた Test Coverage の実行例を記します：

① PROFILE コンパイラ指令を指定してプログラムをコンパイル

```
C:¥work¥wp¥DevTools¥Profiler>cobol ussteel.cbl gnt PROFILE;
Micro Focus COBOL
Version 2.3.01142 Copyright (C) Micro Focus 1984-2016. All rights reserved.
* Checking complete with no errors - starting code generation
* Generating ussteel
* Data:        4424     Code:       22040     Literals:        4204

C:¥work¥wp¥DevTools¥Profiler>_
```

② アプリケーションを実行

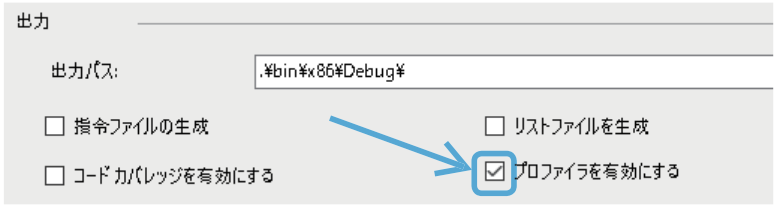
③ 生成されたバイナリファイルからレポートを生成



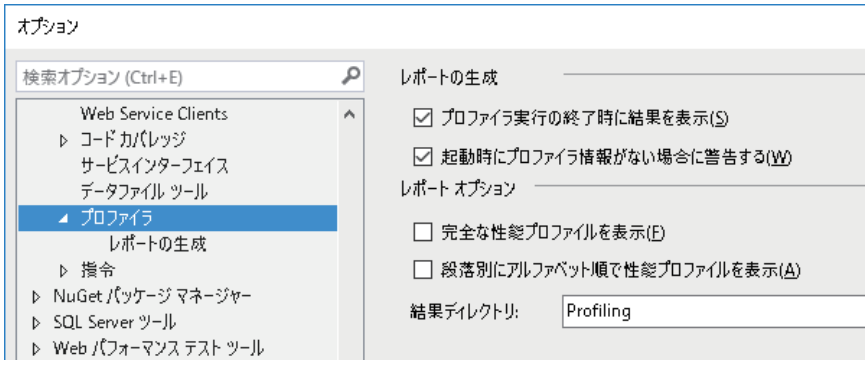
6.2 IDE 上での処理

Profiler は Visual Studio 並びに Eclipse IDE 上でも利用が可能です [14]。この場合、前項で紹介したようなコンパイラ指令指定や profiler コマンドによるレポート生成作業は不要です。
以下に本機能の動作イメージを記します。

Visual Studio 2015 における設定イメージ：

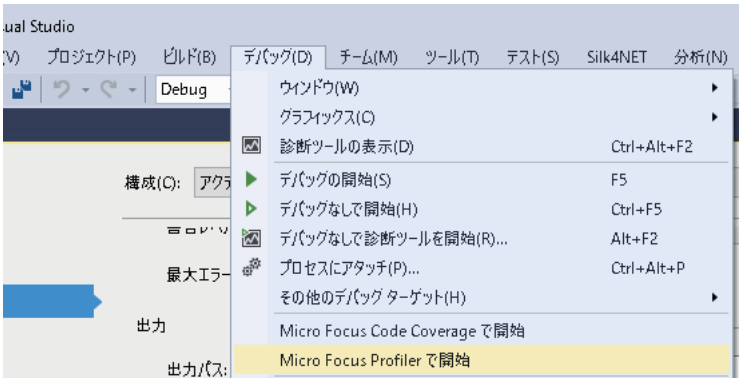


Profiler の挙動も IDE で構成可能です：

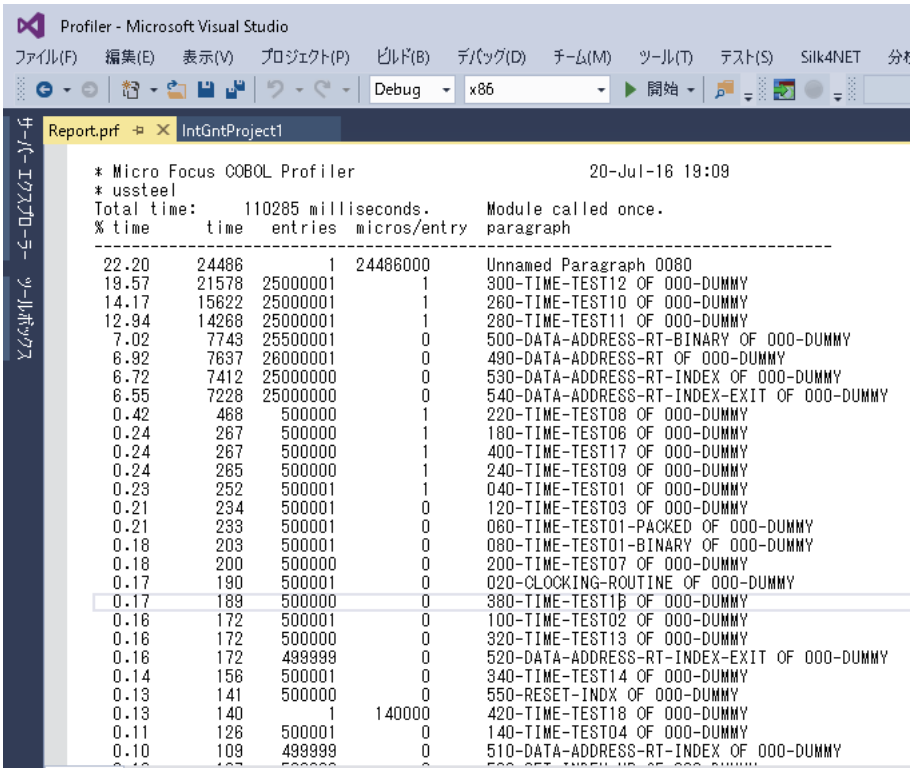


13 IDE 上における操作機能は Visual COBOL 2.3 より実装されています。

Profiler による計測をさせつつアプリケーションを実行 :



アプリケーションの処理が完了すると IDE 内にレポートが出力されます :



7 おわりに

近年、ビジネス環境を取り巻く変化のスピードは高速化の一途を辿っています。このように先の見通しが立てにくい環境にあって、開発期間中における不測の要件変更に対しても柔軟に追従することが求められる時代に突入しているとも捉えられます。実際、30 年以上も前からこのような不確実性に強く、迅速に反応できる開発モデルが様々な企業や団体によって考案されてはいましたが、近年一層注目されるようになっています。ただし、このようなアジャイル開発モデルは従来モバイルや IoT 等を活用して人との関係を構築する SoE(System of Engagement) には向いているが、システムの安定性やデータ保全性が強く求められる SoR(System of Record) には向かないと評価されてきたようです。COBOL で開発される基幹システムは SoR に分類されることが多く、このようなモデルが採用されることはあまりありませんでした。しかし、近年ではこのような開発モデルをエンタープライズ目的でも利用できるよう拡張したエンタープライズアジャイルといったモデルも考案されるようになり、採用企業も増えてきているようです。

本書で紹介した機能を活用した継続的インテグレーションはこのようなエンタープライズアジャイルを実現する第一歩となり得ます。本書ではそれぞれのツールを単体で処理する部分を紹介しましたが、開発方針に則りそれぞれの処理をパイプラインでつなぎ継続的インテグレーションのシナリオに基づいた自動処理を組み上げることが可能です。構成管理ツールと組み合わせソースを安全に管理しつつチェックインなどのタイミングで上述の処理をキックさせるなど、より保全性の高い仕組みを築き上げることも可能です。更には要件管理ツールやデプロイツール等も組み合わせ、デリバリー、運用部分が一体になって協調する仕組みへと拡げていくことも可能です。COBOL アプリケーションの開発ツールとしての利用はもちろんのこと、このような環境変化に強い開発モデル構築への切り口としても是非 Micro Focus Visual COBOL をお役立てください。