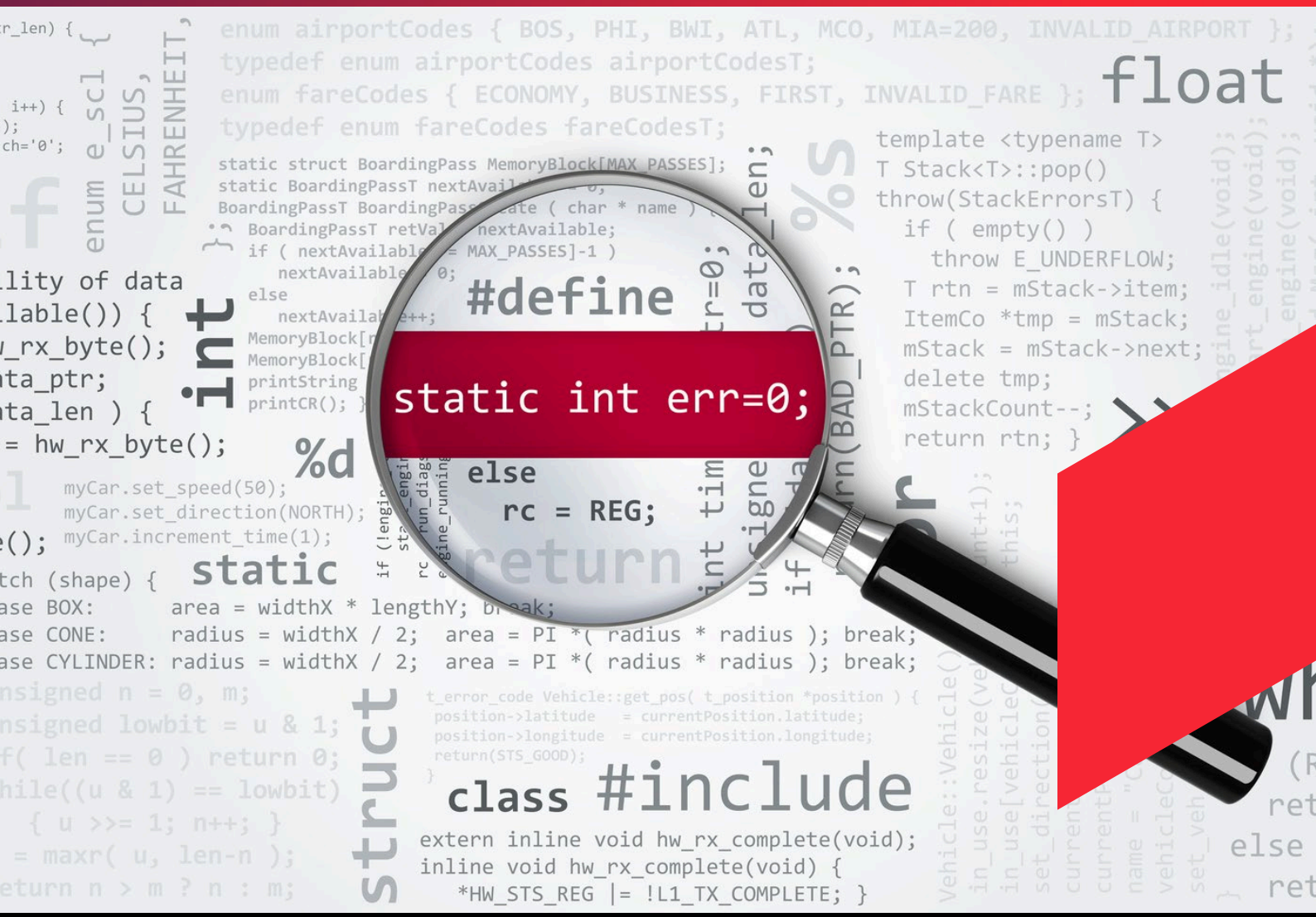
VECTOR >

はじめての VectorCAST

ビギナー向け資料「はじめてシリーズ」



目次

発行元：ベクター・ジャパン株式会社

はじめに

既刊の「はじめての単体テスト」では、「単体テストとは何か？」「なぜ単体テストが必要なのか？」「どのようにすれば効率的に単体テストを行うことができるのか？」といった観点から、近年の組み込みソフトウェア業界の現状や単体テストについての基本的な知識をわかりやすくご紹介しました。

本稿では、単体テストを効率的に実施していくためのツールとして、ベクターが開発、販売する単体テスト自動化ツール「VectorCAST（ベクターキャスト）」をご紹介させていただきます。

1. VectorCAST とは

本章では、VectorCAST の概要と機能について記載します。

1.1 VectorCAST 概要

VectorCAST（ベクターキャスト）は、C 言語、C++言語を対象とした、ソフトウェアの単体テスト自動化ツールです。統合開発環境（IDE）に付随するシミュレーターデバッガ、汎用的な開発環境である Visual Studio、またターゲットボード等、さまざまな環境で単体テストが実行可能です。

VectorCAST があれば、テストケースの自動生成～ テスト自動実行はもちろん、同時にコードカバレッジを計測することができ、テストレポートの生成、回帰テストもワンクリックで行うことができます。また、継続的インテグレーション（CI: Continuous Integration）のサポートツールである Jenkins と連携し、CI 環境も簡単に構築可能です。

ツールの信頼性においても、ドイツの認証機関 TÜV SÜD 社から、航空（DO-178C）、鉄道（EN 50128）、産業（IEC 61508）、自動車（ISO 26262）の各業界の機能安全規格に則ったツール認証を取得しており、ミッションクリティカルな製品開発にも安心してご利用いただくことができます。

VectorCAST は、テスト工数を削減するためにも、品質の高いソフトウェアを開発するためにも、必要不可欠なテスト自動化ツールです。

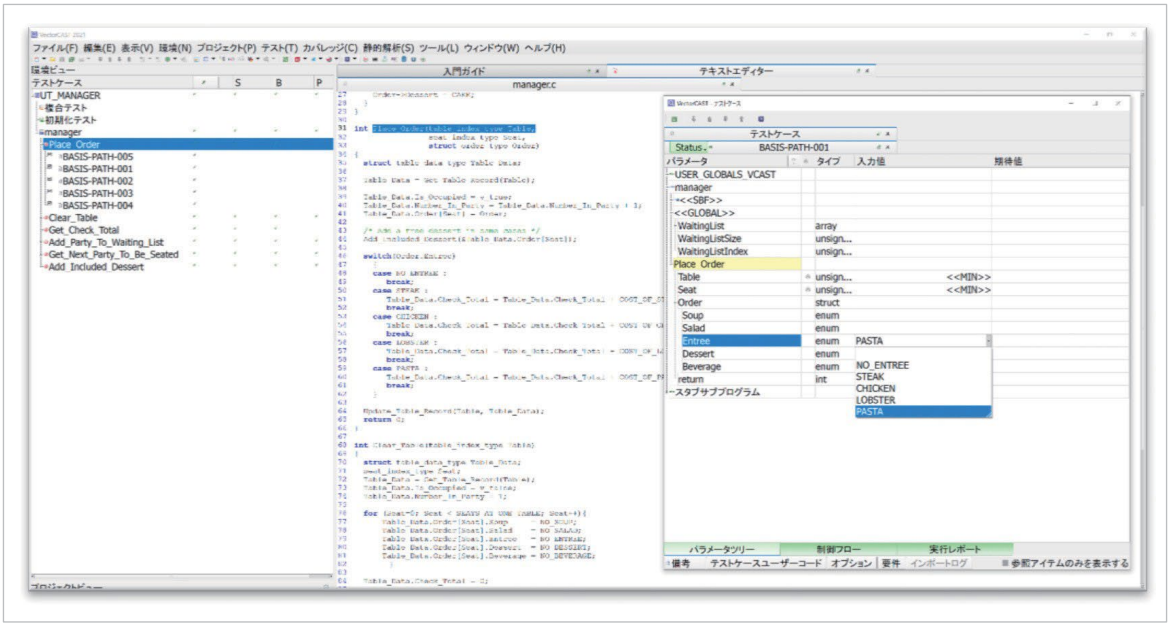


図 1: VectorCAST 画面

1.2 VectorCAST のテストの仕組み

VectorCAST で行うテストの仕組みは以下のようになっています。

① ソースコードを読み込んで解析を実施

② テスト環境を構築、GUI インターフェイス上でテストケースを作成

③ テスト実行

④ 実行結果の判定、コードカバレッジを計測

⑤ テストレポートを自動生成

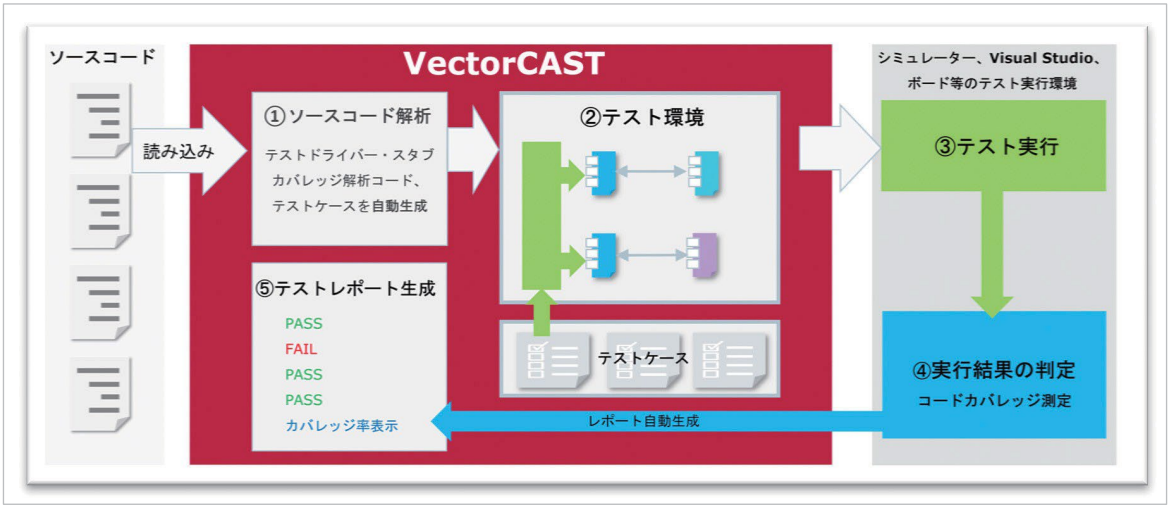


図 2:VectorCAST のテストの仕組み

1.2.1 ソースコード解析 - テスト環境構築

VectorCAST は、ソースコードを解析し、テスト対象関数を呼び出すためのドライバー、スタブ（テスト用に用意されたダミーの関数）を自動生成し、テスト環境を構築します。この際、手作業でのドライバーやスタブの編集は一切不要で、すべてツールで自動生成されます。

また、数関数程度の規模が小さいソースでもテスト環境を作ることができるので、ソースの変更時や、アジャイル開発、TDD（Test-Driven Development）の開発での適用もしやすくなっています。

1.2.2 テスト実行環境

VectorCAST は RSP（Runtime Support Package）という、VectorCAST とコンパイラやシミュレーターを繋ぐテンプレートを提供しています。そのため、数百を超えるコンパイラやマイコンの組み合わせに対応が可能です。

また、国内外の新規マイコンや、FPGA、DSP、独自の SoC 等を使用されている場合でも、ソースコードをビルド／実行できる環境さえあれば VectorCAST と連携が可能です。

図 3:幅広いコンパイラやマイコンの組み合わせに対応

1.2.3 実行結果の判定 - コードカバレッジ計測

VectorCAST は、テスト実行後に各種レポートを自動生成します。カバレッジレポートは、C0、C1、MC/DC、Function/Function Call のカバレッジ基準に標準対応しています。

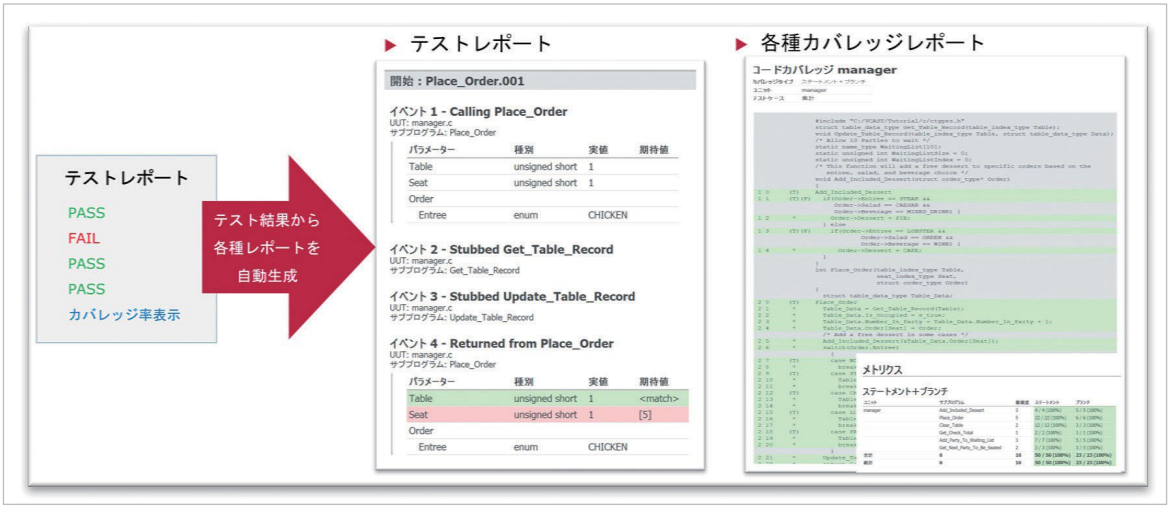


図 4:レポートの生成

1.3 VectorCAST の主な機能

1.3.1 変更ベーステスト機能

VectorCAST の「変更ベーステスト機能」を使用すると、ソースコードの変更があった際に自動でテストへの影響範囲を検出し、回帰テストを行うことが可能です。影響範囲のみに絞ってテストを行うので、回帰テストの効率化を行うことができます。

回帰テストとは、以前テストしたソフトウェアに対して、ソースコードの変更があった際に、変更していない部分に影響がないかを確認するテストです。第 9 章に詳細を記載しています。

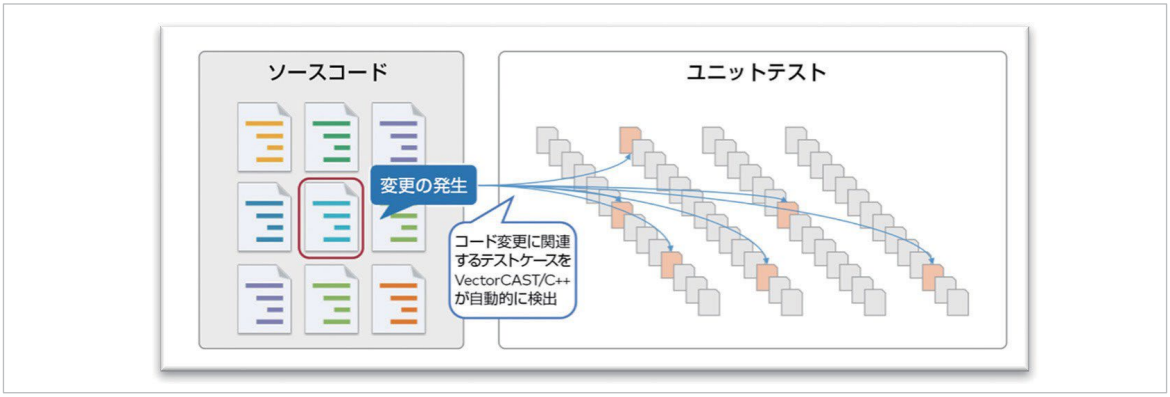


図 5：変更ベーステスト機能

1.3.2 Jenkins 連携機能

VectorCAST は Jenkins と連携し、CI 環境の実現が可能です。第 10 章に詳細を記載しています。

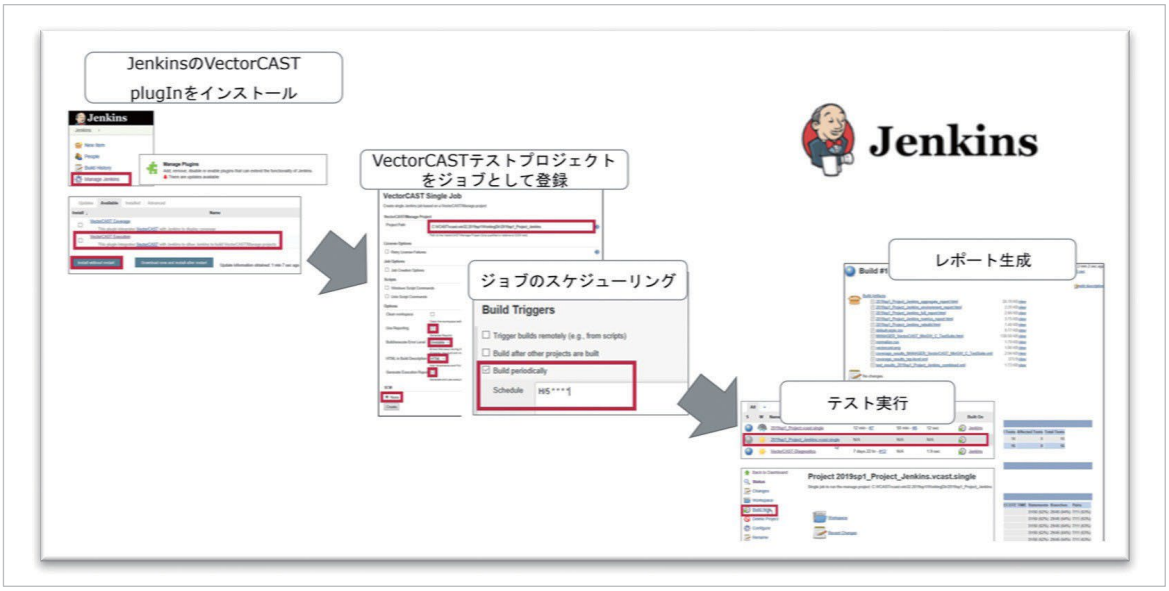


図 6：Jenkins 連携機能

1.3.3 CSV ファイルのインポート／エクスポート機能

VectorCAST は、CSV データファイルからテストデータをインポートしてテストケースを作成することができます。また、テスト実行後にテストパラメータの一覧を CSV ファイルへエクスポートすることができます。第 11 章に詳細を記載しています。

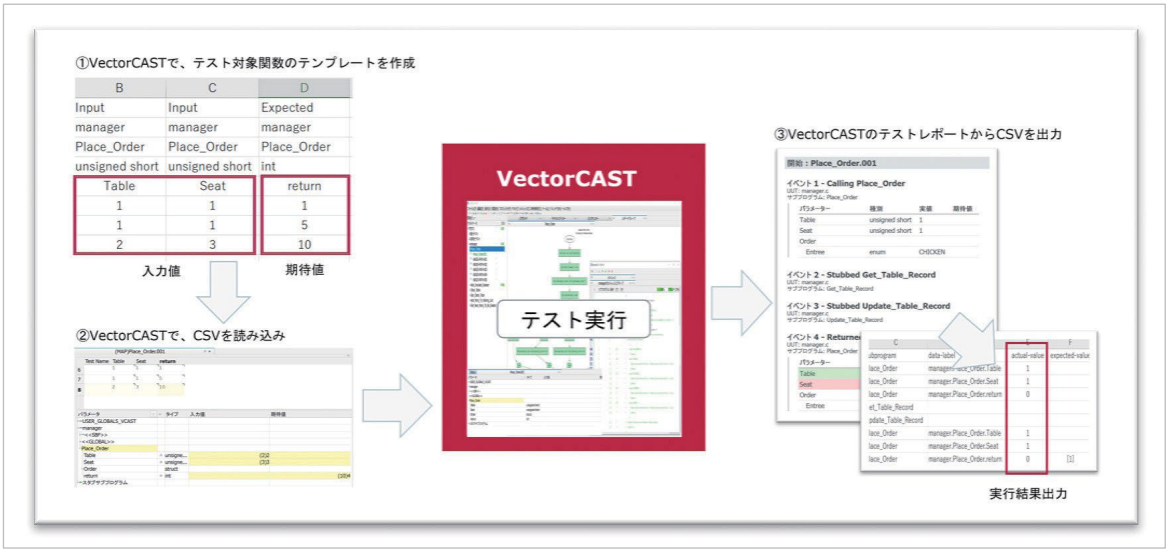


図 7:CSV ファイルのインポート／エクスポート機能

1.3.4 複合テスト機能

VectorCAST は単体テストだけではなく、複数の関数をシーケンシャルに実行することができます。

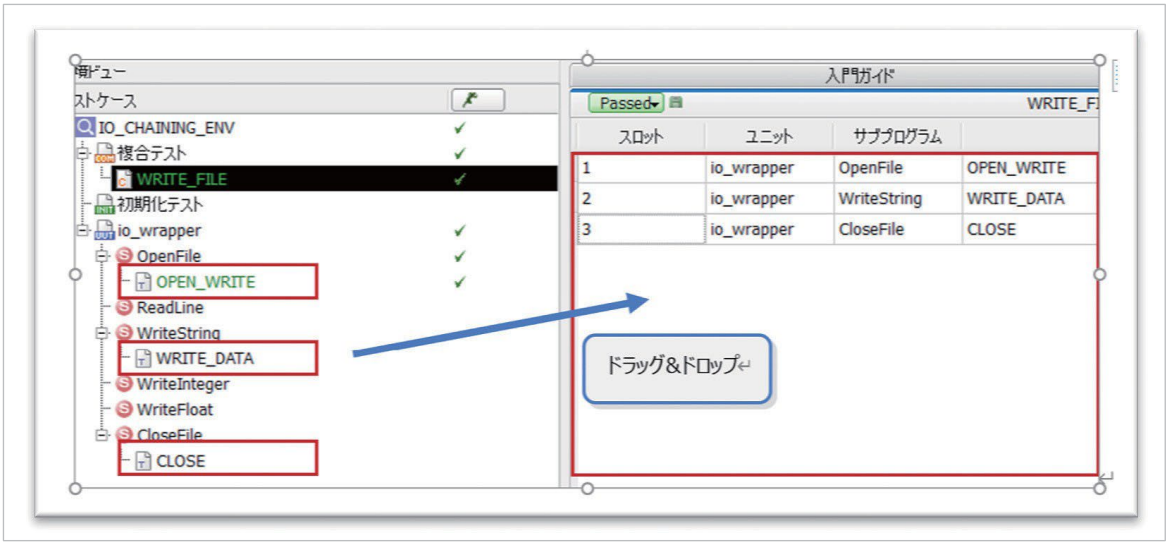


図 8:複合テスト機能

図 8 は、OPEN_WRITE、WRITE_DATA、CLOSE という一連の処理を、順番にテストする場合のイメージです。第 13 章に詳細を記載しています。

2. VectorCAST を使用する前に

本章では、VectorCAST を使用する際に必要なもの、また本稿で使用するソースコードについて記載します。

2.1 VectorCAST を使用する際に必要なもの

VectorCAST を使用する際は、VectorCAST 本体の他に以下が必要です。

① コンパイラ

② シミュレーター

③ VectorCAST ライセンスファイル

コンパイラ:

テスト環境のビルドに使用します。実際の開発環境で使用しているコンパイラ、もしくは VectorCAST と一緒にインストールされる MinGW コンパイラが使用できます。

シミュレーター:

テスト実行環境として、IDE などのシミュレーターが必要です。MinGW を使用する場合は、VectorCAST 内蔵シミュレーターを使用するため不要です。

VectorCAST ライセンスファイル:

評価をご希望のお客様へ評価版ライセンスをご提供しています。ライセンス形態はフローティングライセンスとなります。ライセンスに関してご要望、ご不明な点などございましたら巻末の「お問い合わせ先」よりご連絡ください。

2.2 テスト対象のソースコードについて

本稿では、説明に VectorCAST のチュートリアルとして用意されたソースコードを使用します。レストランでの入店待ち、注文状況に応じた価格決定などを行うプログラムとなっています。

本稿で扱う関数は以下のとおりです。ソースコードは巻末の参考資料に掲載しています。

① Add_Party_To_Waiting_List

関数定義:void Add_Party_To_Waiting_List(char* Name)

機能概要:入店待ちのお客様が入力した氏名を、待機リストの配列に追加する

② Add_Included_Dessert

関数定義:void Add_Included_Dessert(struct order_type* Order)
機能概要:お客様が注文したメニューに応じて、デザートを自動追加する

③ Place_Order

関数定義:int Place_Order(table_index_type Table, seat_index_type Seat,struct order_type Order)

機能概要:お客様が注文したメイン料理の金額を、テーブルの合計金額に加算する

④ Clear_Table

関数定義:int Clear_Table(table_index_type Table)
機能概要:テーブルの注文履歴、合計金額などを初期化する

3. テストプロジェクトの設定

本章では、VectorCAST 内で使用されるプロジェクトと環境設定について記載します。

3.1 プロジェクトとは

プロジェクトとは VectorCAST のテスト環境を集約したもので、複数の単体テスト環境とそれらに含まれるテストケースからなります。

プロジェクトの作成には、以下のメリットがあります。

- プロジェクトや単体テスト環境をコピーすることで、簡単に派生プロジェクトへ流用することができます。
- 変更ベーステスト機能や構成ベーステスト機能など、VectorCAST の機能を使うことができるようになります。

3.2 プロジェクトファイルの作成

まず、メニューバーから「ファイル」⇒「新規」⇒「VectorCAST プロジェクト」⇒「空のプロジェクト」を選択します。

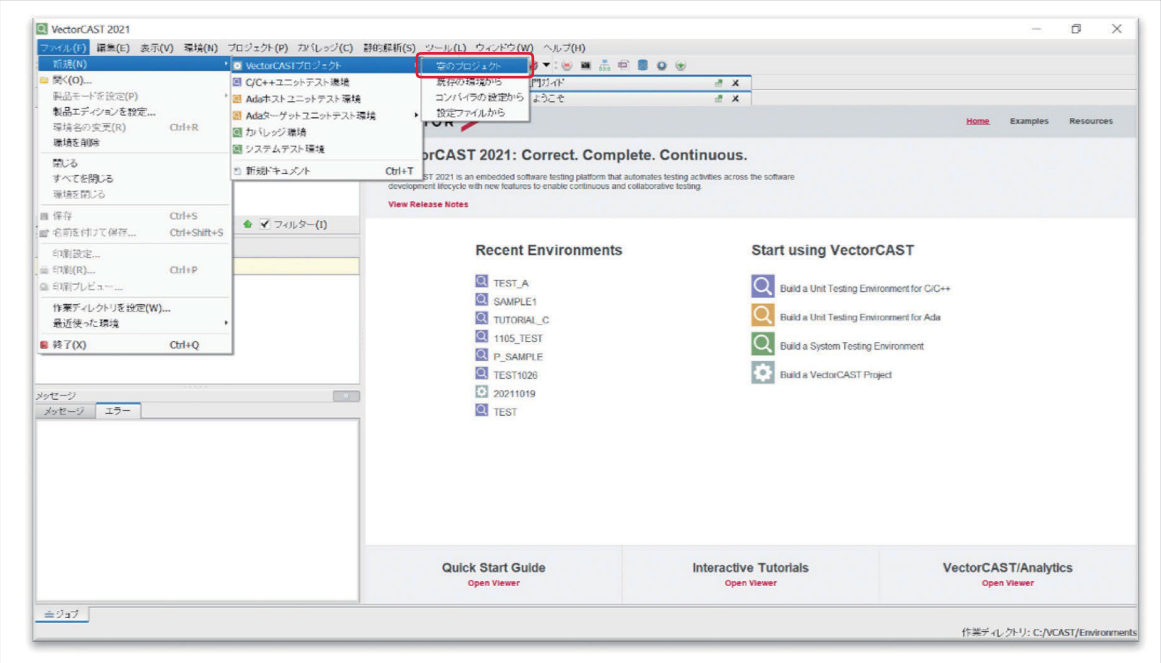


図 9：新規プロジェクト作成

「プロジェクト名」のフィールドに任意の名前（今回は Test _Campaign）と入力し、「コンパイラ」のドロップダウンメニューから「C/C++」⇒「VectorCAST MinGW[1]」⇒「C」を選択して、「作成」ボタンを選択します。

[1] 本稿では VectorCAST 内蔵のコンパイラ MinGW を使用しています。お客様の使用しているコンパイラを選択することも可能です。コンパイラ対応状況については「お問い合わせ先」よりご連絡ください。

図 10:プロジェクト名の設定とコンパイラ選択

メイン画面左上のプロジェクトビュー上に Test_Campaign のプロジェクトノードが展開されたら、Test_Campaign のプロジェクトのビルド設定を行います。

プロジェクトレベルで設定した内容は、その下に作成するテスト環境へ継承されます。そのため、テスト環境に共通な設定は、プロジェクトレベルで設定しておくことをお勧めします。

本稿では、ソースコードの検索ディレクトリと、コードカバレッジタイプの 2 つをプロジェクトレベルで設定しておきます。

まず、「Manage」ノードを展開し、ソースディレクトリの項目右部を選択します。

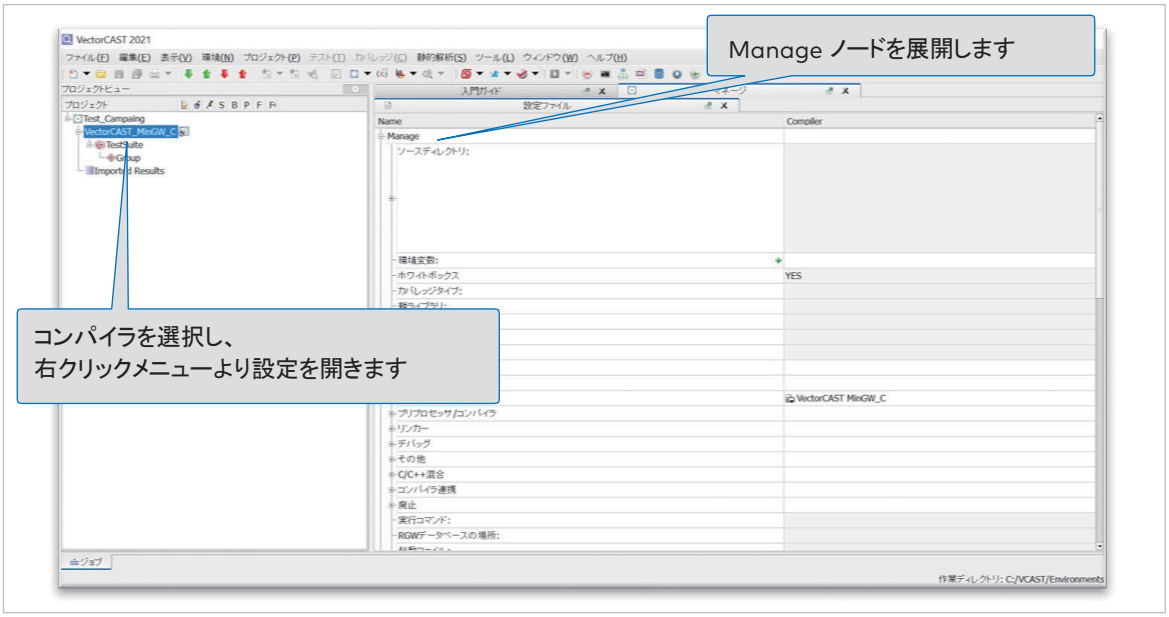


図 11:プロジェクトレベルの設定画面

ボタンを選択し、「検索ディレクトリを追加」ダイアログを開きます。ソースディレクトリに移動して「選択」ボタンをクリックすると、オプションにディレクトリパスが追加されます。

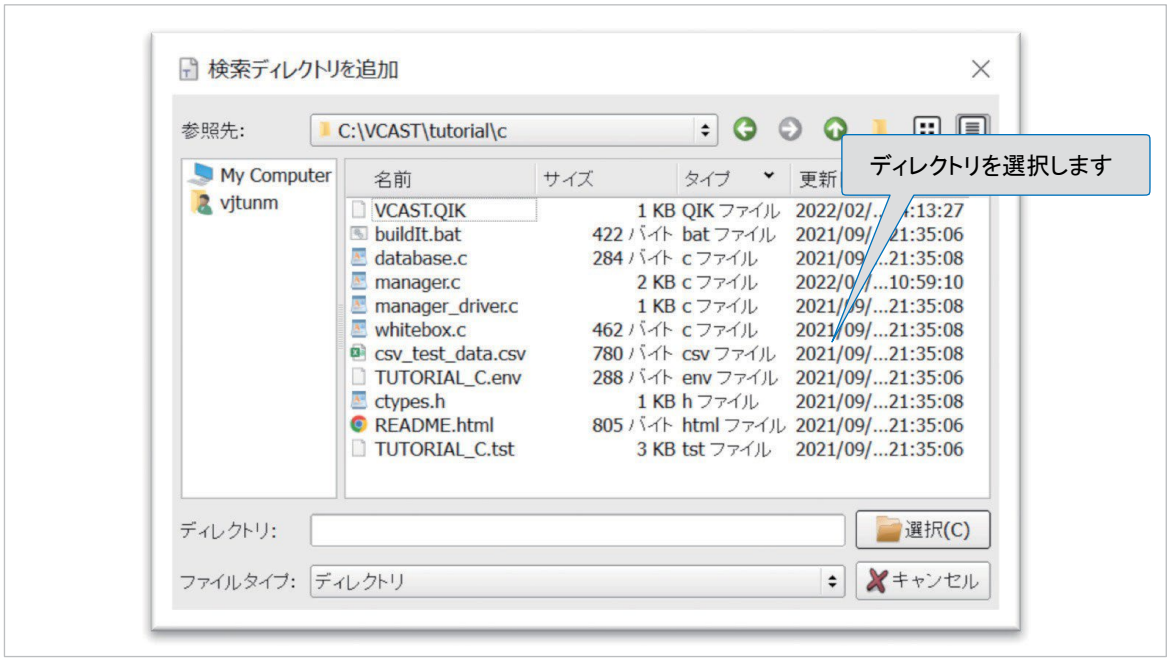


図 12：検索ディレクトリの追加

追加されたパスを右クリックし、コンテキストメニューから「テスト対象ソースの検索ディレクトリに設定」を選択して、変更を保存します。

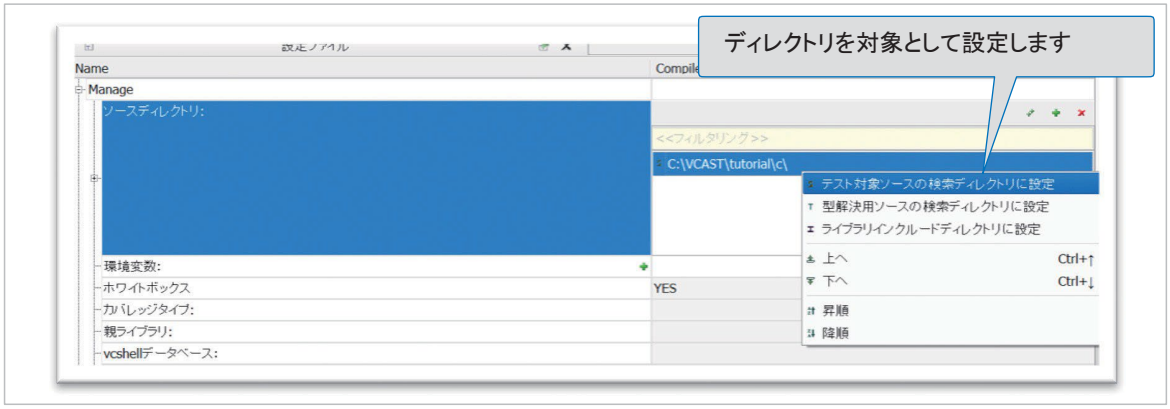


図 13：検索ディレクトリの追加

次に、「カバレッジタイプ」ドロップダウンメニューから「Statement+ MC/DC」を選択して、Test_Campaign プロジェクトのカバレッジタイプを更新し、変更を保存します。ここでは「Statement + MC/DC」を選択しましたが、必要なカバレッジに合わせて任意で選択してください。

4. 単体テストを実行しよう

本章では、単体テスト環境とテストケースを作成し、単体テストを実行する手順について記載します。

4.1 新しい環境の作成

第 3 章で作成したプロジェクトに、単体テスト環境を追加します。プロジェクトツリーでグループノードを右クリックし、コンテキストメニューから「ユニット(単体)テスト環境を作成」⇒「対話式」を選択します。

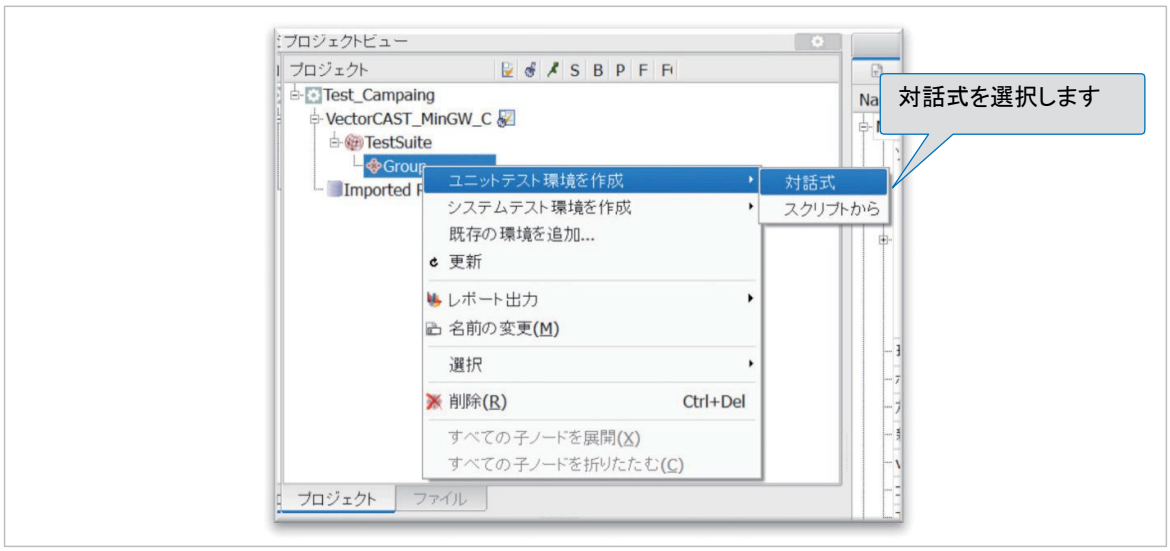


図 14:単体テスト環境作成

環境構築ウィザードが開いたら「1 コンパイラを選択」を選択し、図 15 のように VectorCAST MinGW_C コンパイラが選択されていることを確認してください。

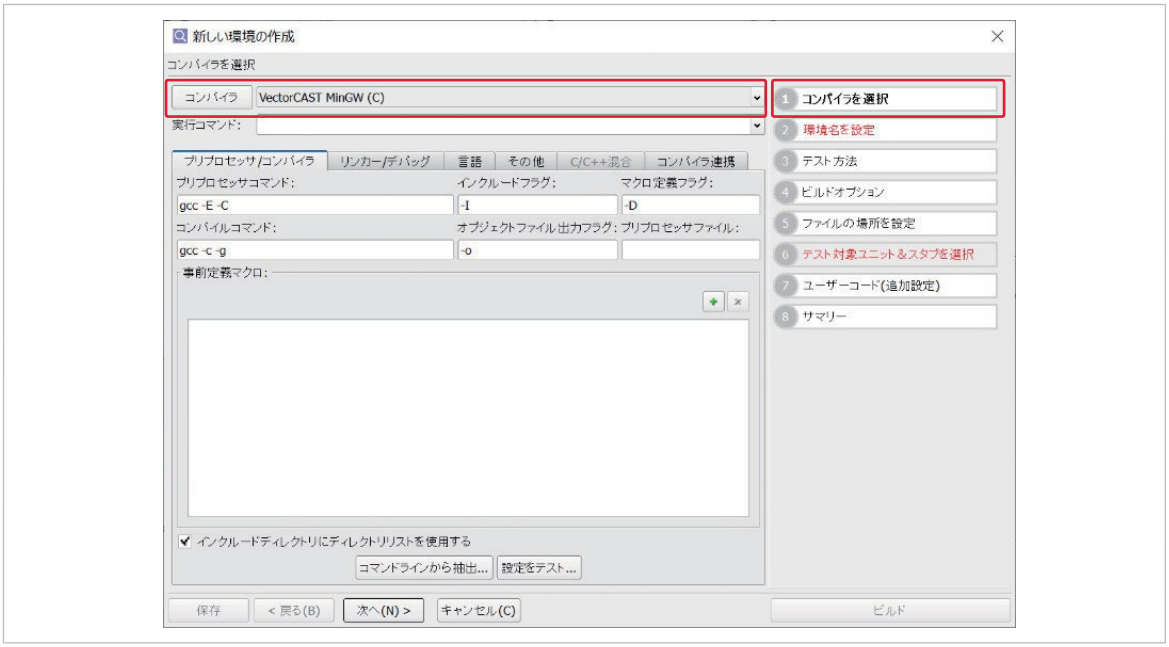


図 15：環境構築ウィザード Step1

「2 環境名を設定」に進んで、「環境名」に UT_MANAGER（任意で指定可能）と入力します。

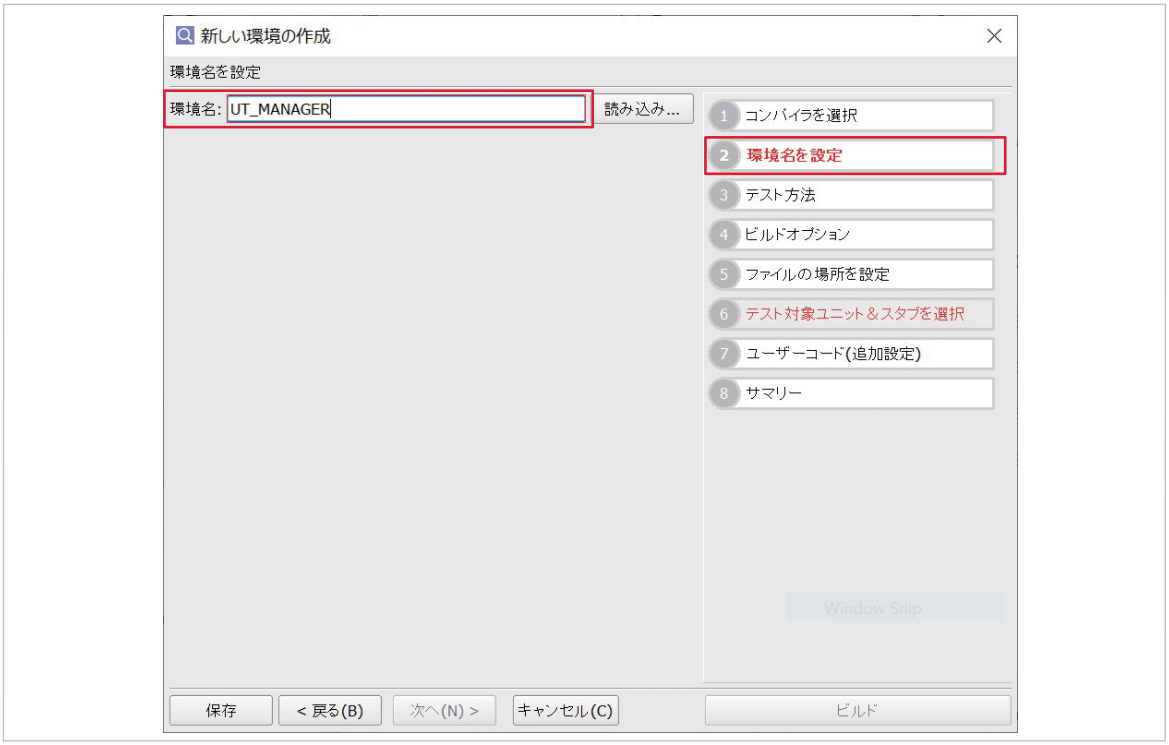


図 16：環境構築ウィザード Step2

「4 ビルドオプション」を選択し、Statement+ MC/DC カバレッジタイプが選択されていることを確認してください。

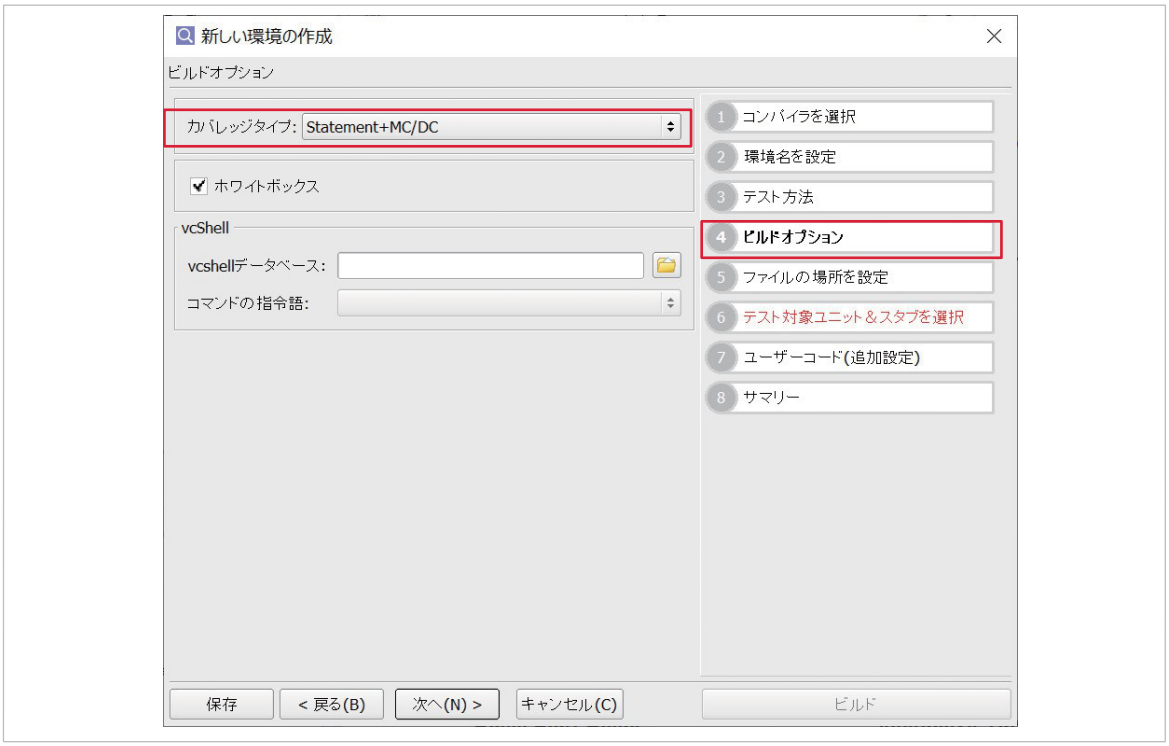


図 17：環境構築ウィザード Step4

「5 ファイルの場所を設定」を選択し、検索パスが継承されていることを確認してください。

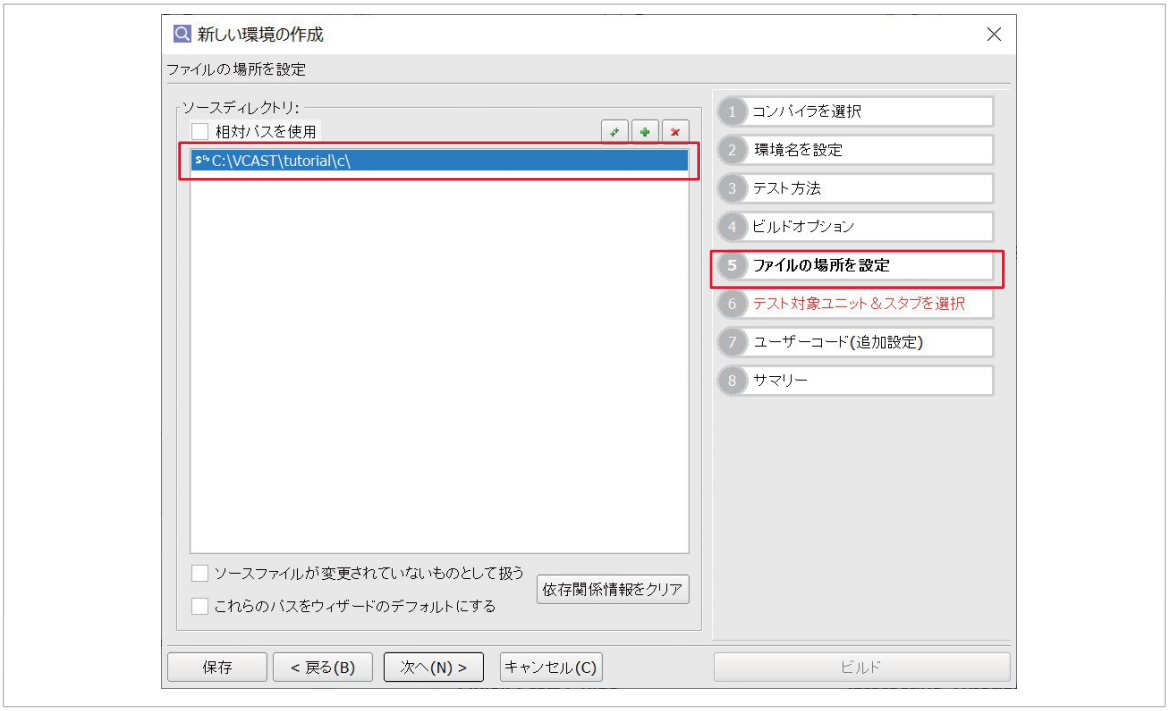


図 18：環境構築ウィザード Step5

「6 テスト対象ユニット&スタブを選択」を選択し、テスト対象のソースコードを選択し、「ビルド」ボタンを押下します。

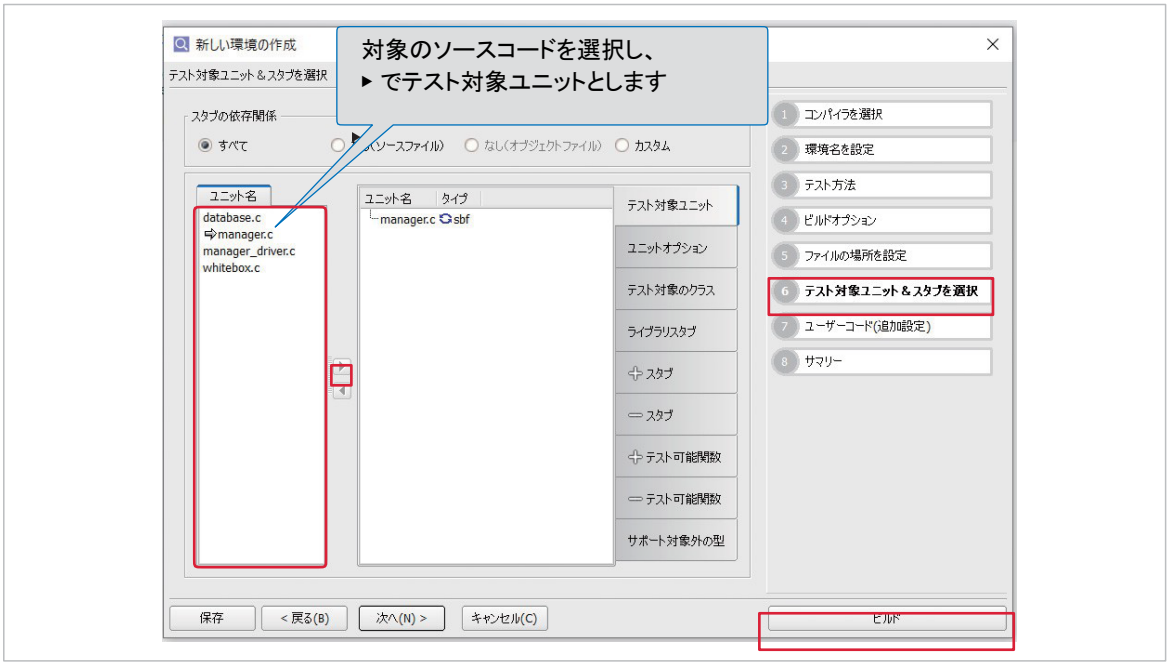


図 19：環境構築ウィザード Step6

ビルドが完了すると、環境ビューパネルで新しい環境が自動的に開き、テストケースを作成できるようになります。

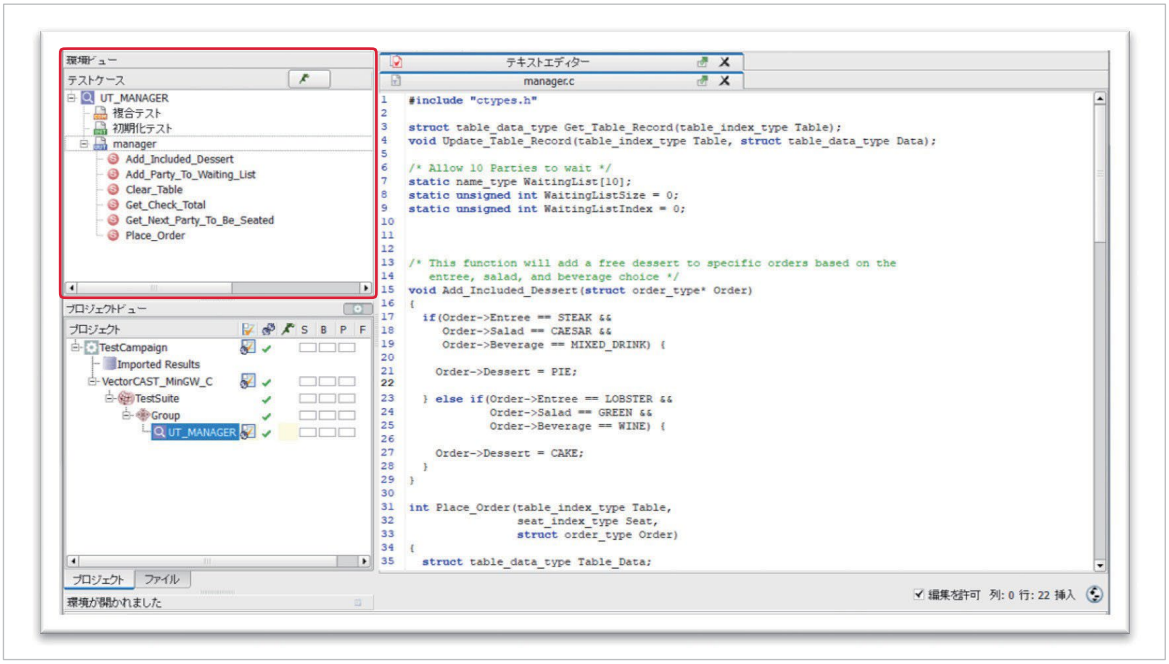


図 20：単体テスト環境

4.2 テストケースの作成

manager.c に定義されている Place_Order 関数を対象とし、テストケースを作成していきます。

Place_Order を右クリックし、コンテキストメニューから「テストケースを挿入」を選択します。すると、テストケース Place_Order.001 が作成されるので、入力値と期待値を図 22 に示した赤枠へ設定します。

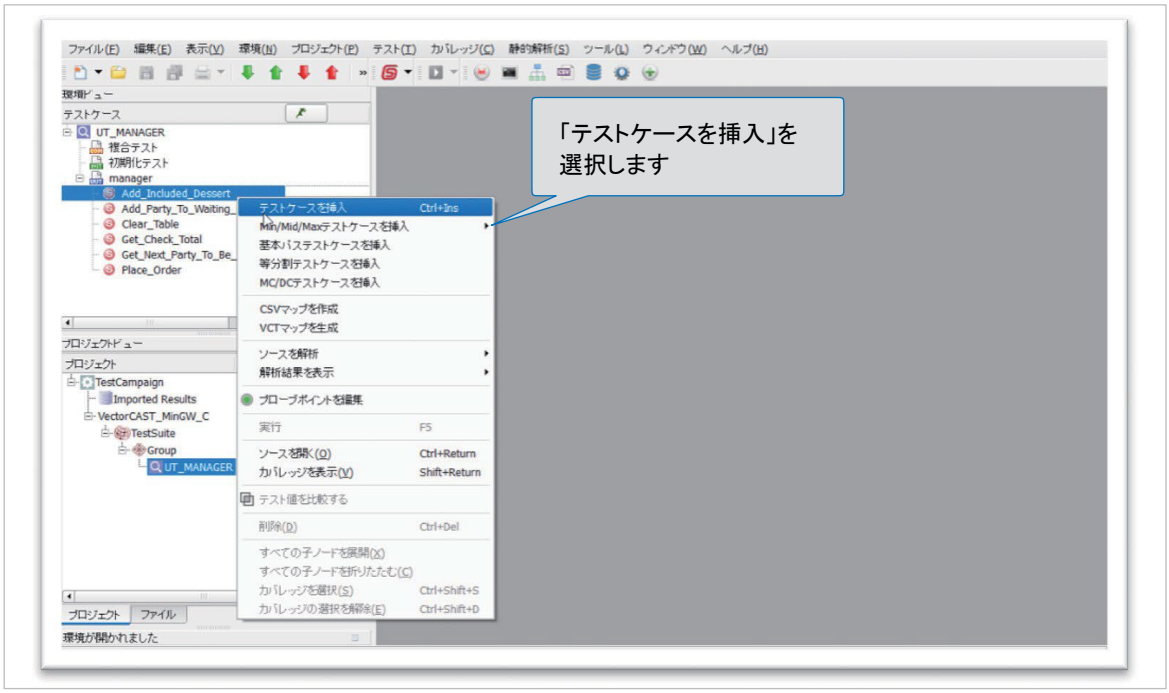


図 21:テストケース挿入

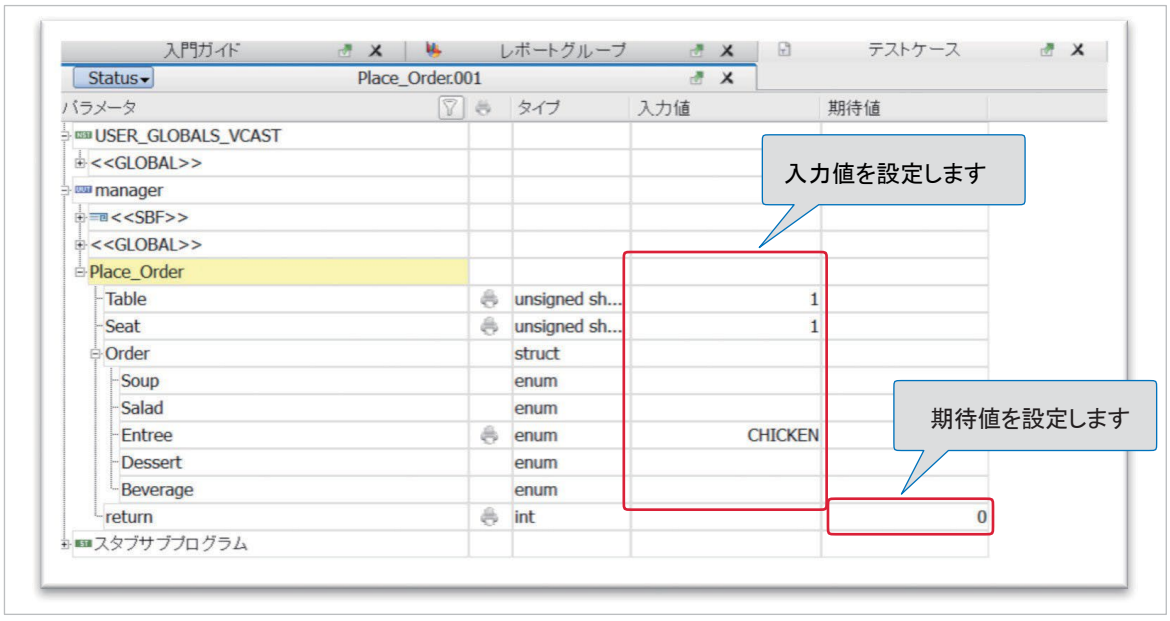


図 22:入力値と期待値の設定

4.3 テストケースの実行

テストケースの作成が完了したら、画面右上の  アイコンを押下してテストケースを実行します。

テストケースの実行後、実行結果が表示されます。入力した値が、実値として実行されていることを確認します。また、実行結果の期待値欄が"＜match＞"と表示されていることを確認します。

図 23:テスト結果

テスト実行の結果、カバレッジが網羅できた部分は緑色で表示されます。

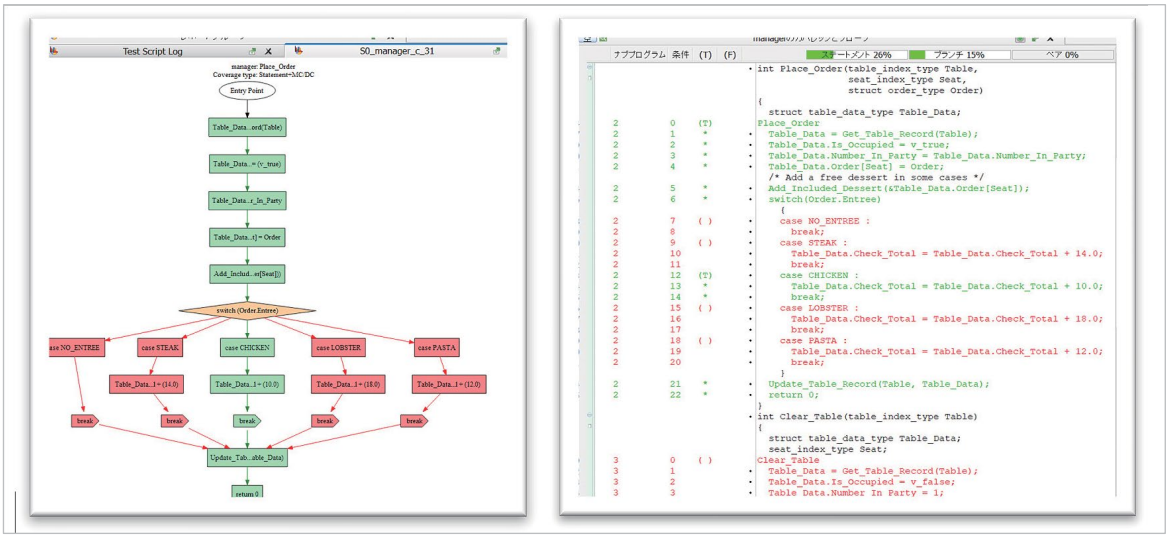


図 24:カバレッジ取得結果

5. テストケース自動生成機能を使おう

本章では、VectorCAST のテストケース自動生成機能について記載します。

5.1 VectorCAST が自動生成できるテストケースの種類

VectorCAST は、テスト対象となるソースを解析し、テストケースの自動生成を行うことが可能です。自動生成できるテストケースは、以下 4 種類です。

- Min/Mid/Max テストケース

  変数が持つ型の最小値、最大値、中間値が入力値に設定されたテストケースを自動生成します。

  例：unsigned char 型の場合、0、255、127 といった入力値を自動生成します。

- 基本パステストケース

  C0/C1 カバレッジを網羅するために必要なテストケースを自動生成します。

- 等分割テストケース

  ある変数の型の取り得る値を、N 数で割ったテストケースを自動生成します。

  例：unsigned char 型の変数の入力値を、データの分割数を 5 としてテストケース生成する場合、0、51、102、153、204、255 が入力値に設定されます。

- MC/DC テストケース

  MC/DC カバレッジを網羅するために必要なテストケースを自動生成します。

各カバレッジの定義については、別書「はじめての単体テスト」をご参照ください。

5.2 テストケースの自動生成方法

テストケースツリーで、テストケースを自動生成する対象（単体テスト環境、ソースファイル、関数）を右クリックし、コンテキストメニューから自動生成したいテストケースの種類を選択することで、自動生成が可能です。

例：manager.c に定義されているすべての関数に、基本パステストケースを挿入する場合

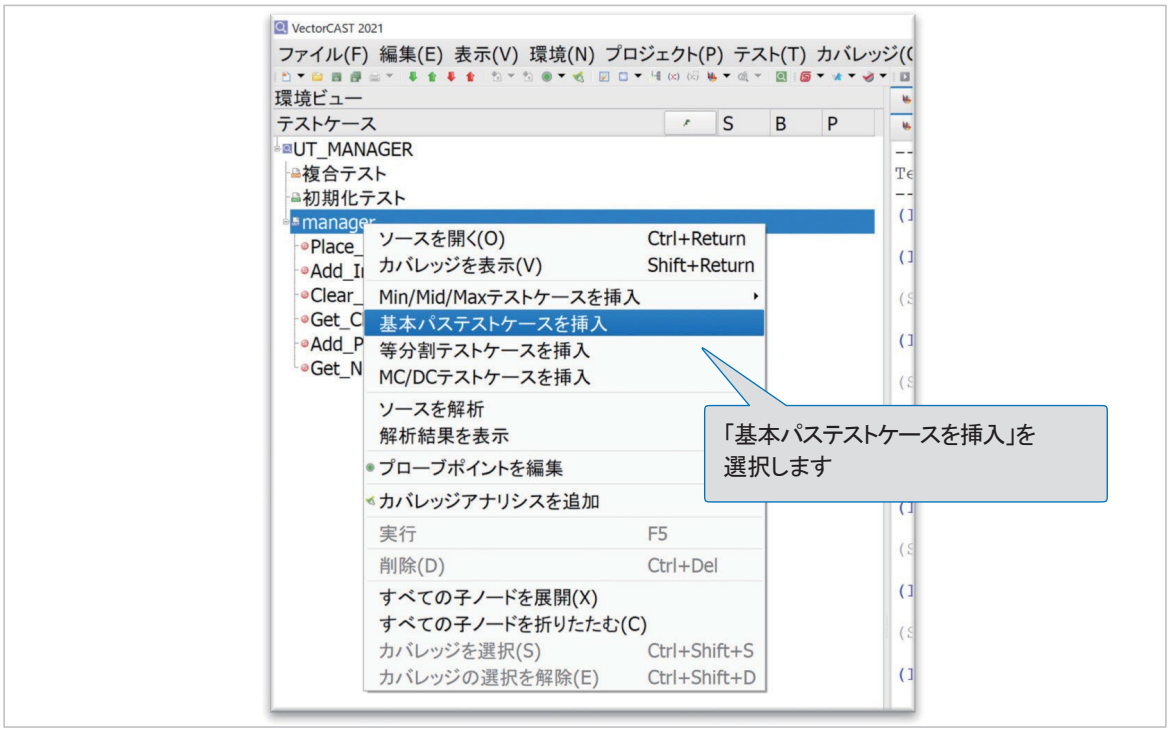


図 25：基本パステストケースを挿入

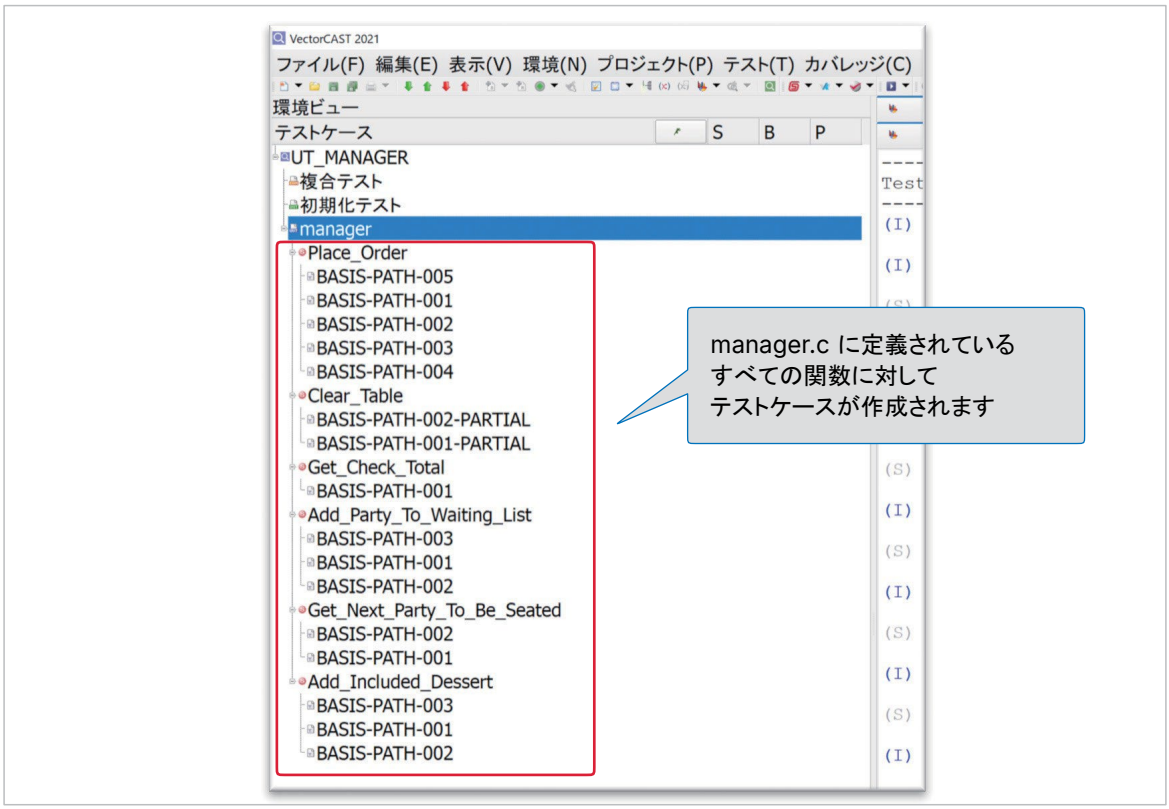


図 26：基本パステストケース挿入結果

テスト完了後は、テスト対象関数を右クリックし、コンテキストメニューから「カバレッジ表示」を選択することで、カバレッジの網羅状況を確認することが可能です。

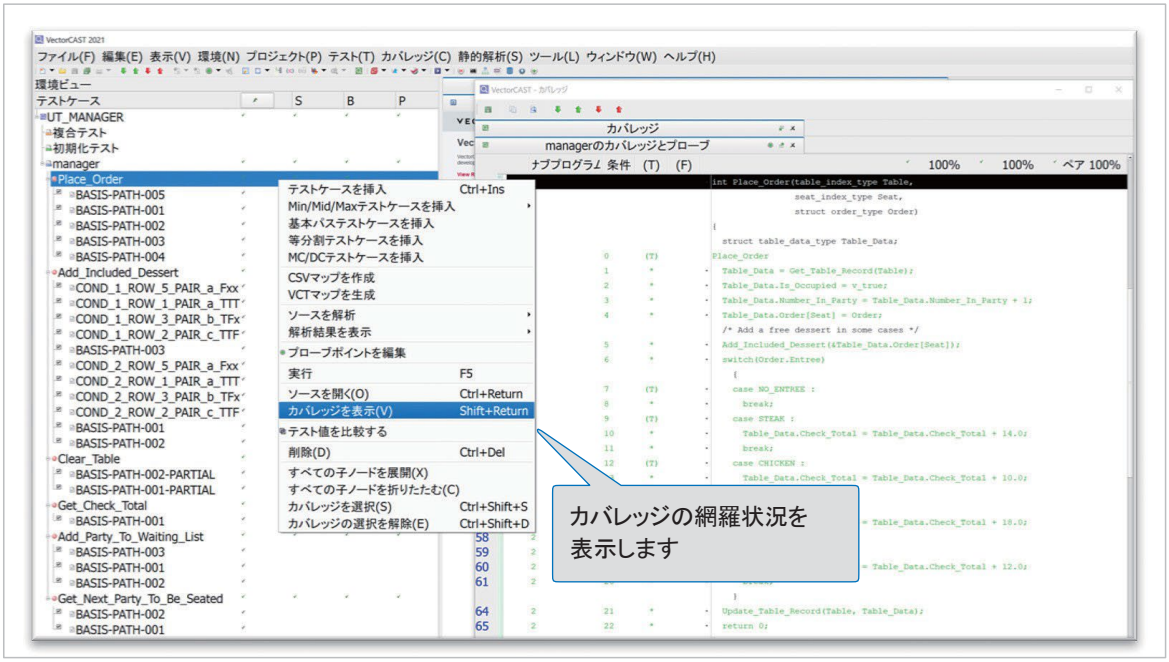


図 27:カバレッジ網羅状況の確認

また、「コントロールフローグラフ」アイコンを押下すると、フローチャートと、カバレッジの網羅状況をリンクして表示させることも可能です。

もし、コントロールフローグラフアイコンがツール上に表示されていない場合は、「ツール」 ⇒「カスタムツール」⇒「カスタムツールの編集」からコントロールフローグラフにチェックを入れることで、表示されるようになります。次ページの図 29 では属性の名前欄も赤枠で囲われていますが、入力は不要です。

図 28:コントロールフローグラフアイコン

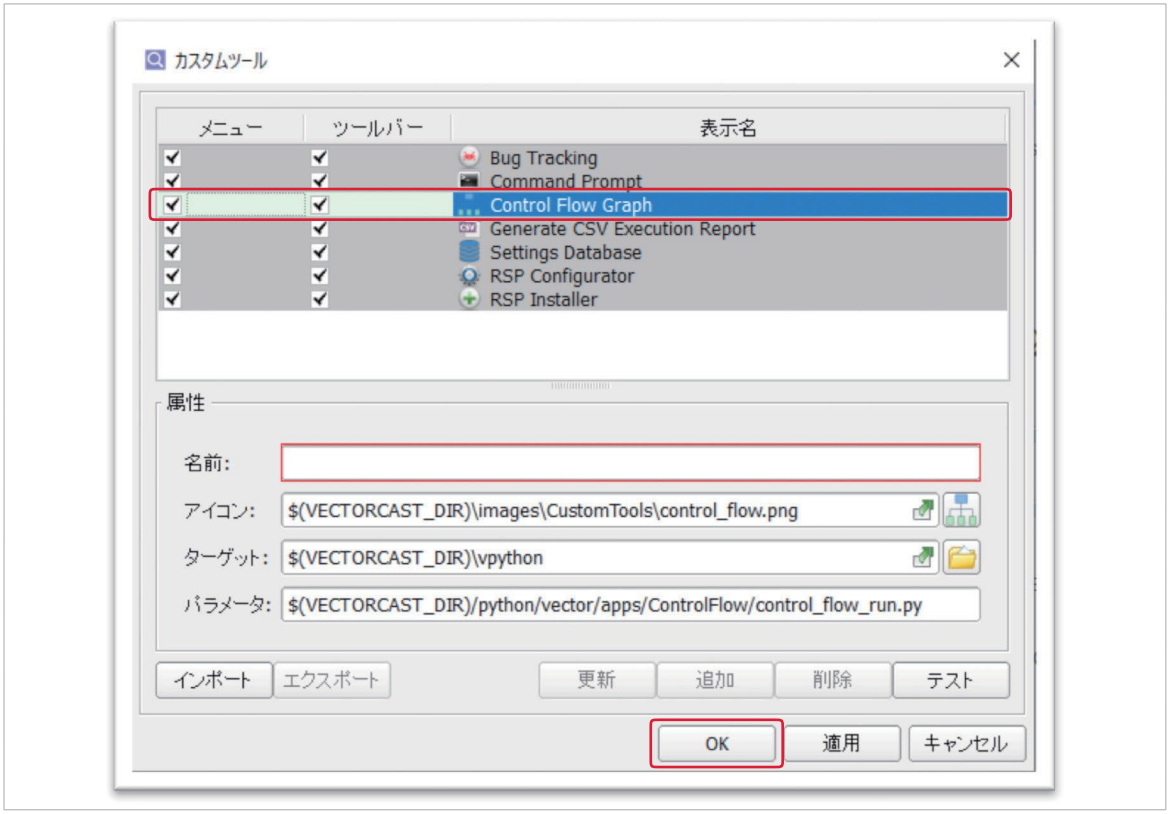


図 29:カスタムツール設定画面

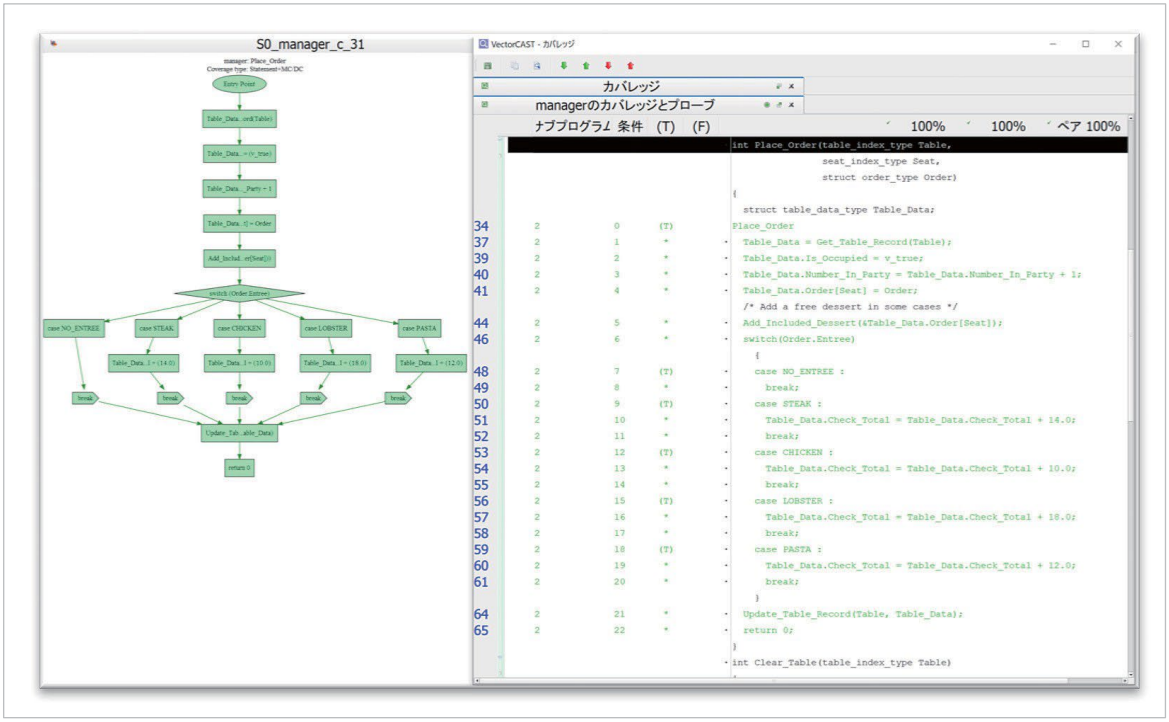


図 30:コントロールフローグラフとカバレッジビューアー

5.3 その他の便利な機能

① テストケースの入出力変数に対して、範囲で値を指定する方法

テストケースエディターに表示されている変数を右クリックし、コンテキストメニューから「プロパティ」を選択します。

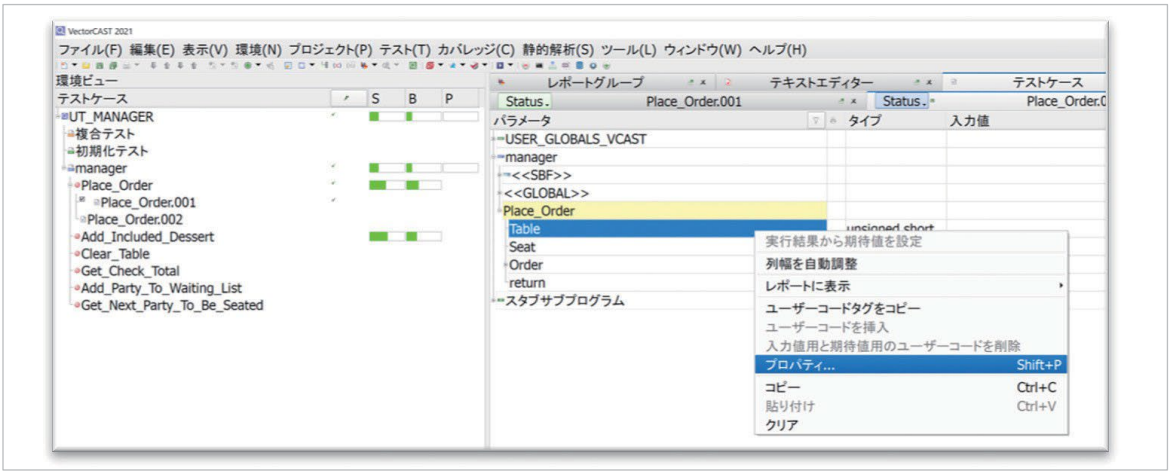


図 31:テストパラメータのプロパティを表示

プロパティ内の「値の範囲を設定」タブを選択し、任意の開始値と終了値、増分（デルタ値）を設定して入力値や期待値を設定します。

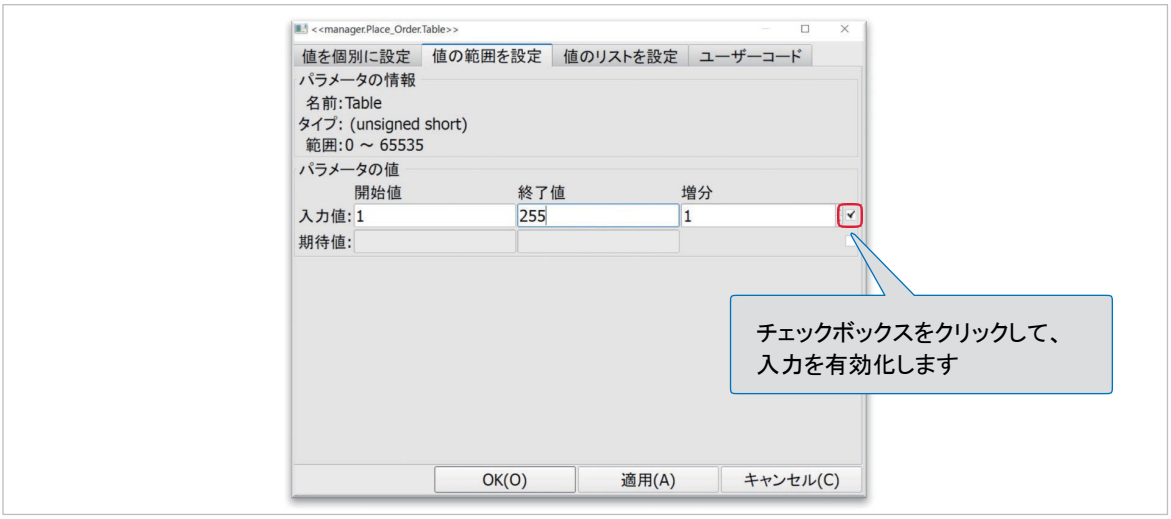


図 32:値の範囲を設定

パラメータに 1～255 までの値を順番に割り振ってテストを行う場合、通常の方法でテストケースを作成すると合計 255 個ものテストケースが必要です。しかし、値の範囲を設定してテストケースを作成すると、1 つのテストケースで 255 パターンのテストが実施できます。

② テストケースの入出力変数に対して、リストで値を指定する方法

プロパティ内の「値のリストを設定」タブを選択し、値を順番に追加して、リスト形式で入力値や期待値を設定します。

リストの値には、その値でテストを繰り返す回数を指定することができます。何も指定しない場合は、1 回と捉えられます。

図 33:値のリストを設定

図 33 の場合、ある変数の入力値を、1 が 1 回、5 が 2 回、10 が 3 回、20 が 2 回というパターンでテストするテストデータを、1 つのテストケースで作成することができます。

6. テストレポートを生成しよう

本章では、VectorCAST が生成可能な各種レポートについて記載します。

6.1 テストレポートの生成と表示

VectorCAST では、テスト実行後にレポートが自動生成されます。各レポートの要素がすべて盛り込まれている「詳細レポート」を参照いただくことを推奨します。

①テスト（T）⇒ 表示 ⇒ 詳細レポートを選択します。

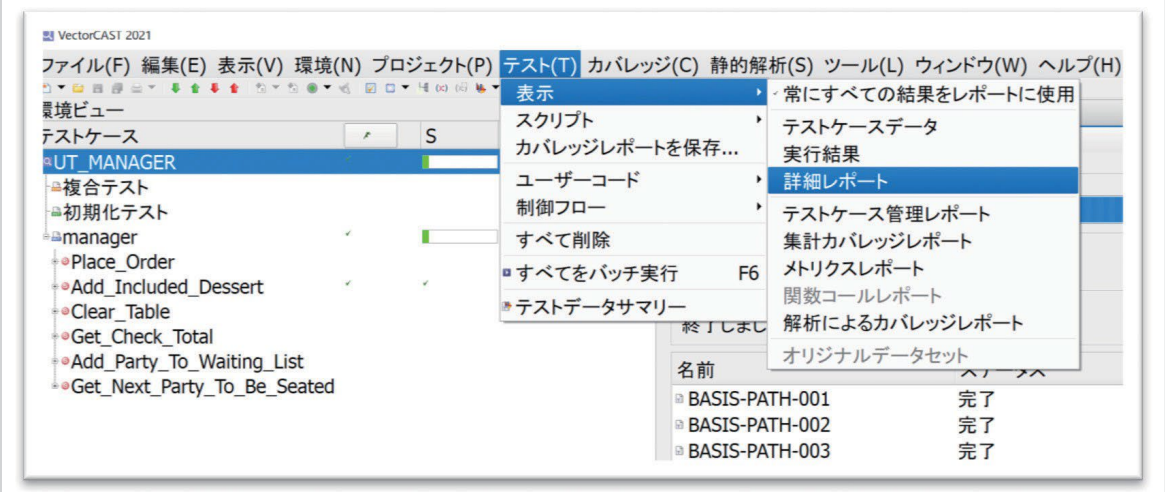


図 34：詳細レポートの表示

生成されたレポートは、クライアント PC 内のユニットテスト環境フォルダ内に保存されますので（図 34 の場合は UT_MANAGER フォルダに格納）、必要に応じてご利用ください。

6.2 レポート形式の変更

VectorCAST のテストレポートは、デフォルトは HTML 形式で生成されます。レポートの形式を変更する際には

① ツール（L）⇒ オプションを選択します。

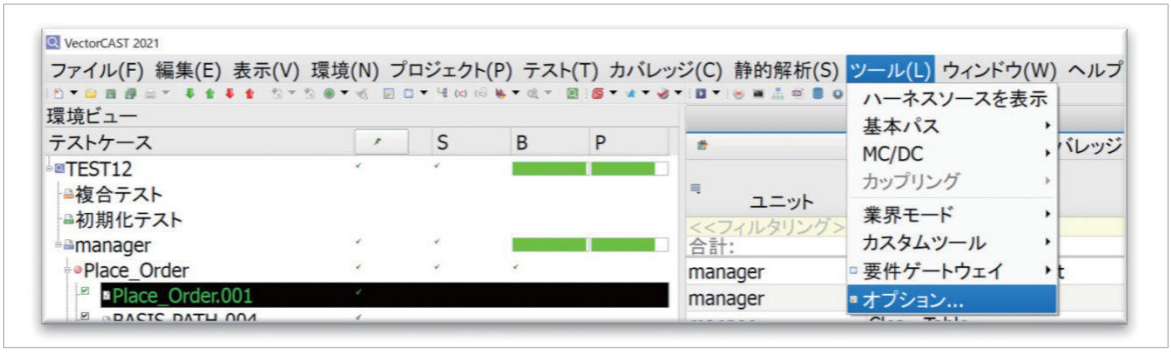


図 35：オプションメニューを選択

② 「内容」や「形式」タブからレポートの設定を変更可能です。

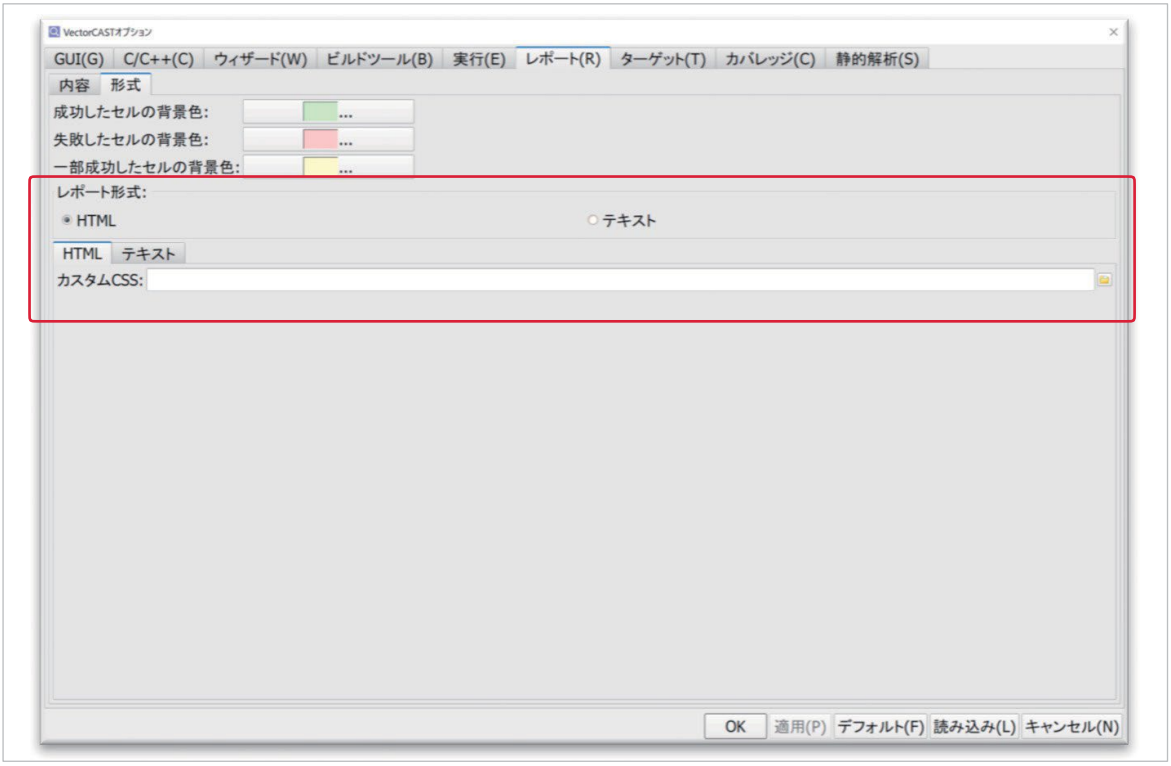


図 36:レポート形式の設定

7. 配列やポインタ変数をテストしよう

本章では、テスト対象関数の入力／出力値が配列またはポインタ変数の際に、VectorCAST のテストケースで必要な設定に関して説明します。

7.1 配列の取り扱い方法

ここでは、例に Add_Party_To_Waiting_List 関数を使用し、仕様より、入力値を、配列の要素数が「10」の「String 型」と想定します。VectorCAST では配列を入力／出力値に設定する際、テストパラメータ用にメモリを確保してから値を入力する必要があります。

【配列の「0」番目の要素へ入力／期待値を設定する】

① メモリを確保するため、<<Expand Indices: Size 10>>の入力値に、使用したい配列の要素番号を設定します。

たとえば、図 37 のように、要素番号「0」を設定すると、1 要素分（要素「0」）のメモリが確保できます。

② 入力値を設定します。（例：TOM）

③ 必要に応じて期待値を設定します。

④ テストを実行します。

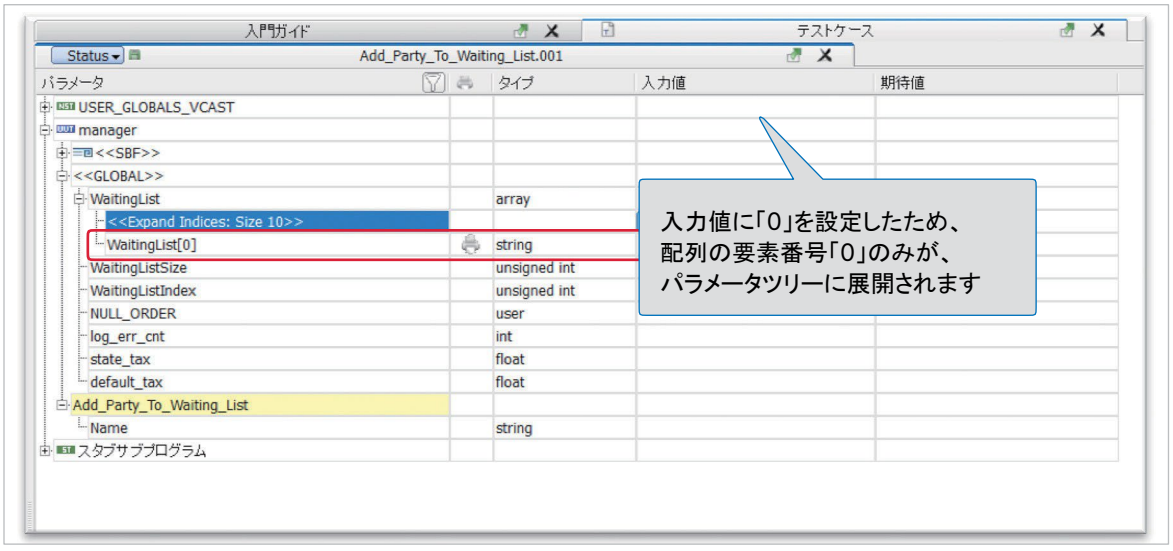


図 37：配列型変数のメモリ確保（1 要素分）

【配列の「0」番目から「9」番目の要素（すべての要素）へ入力／期待値を設定する】

① メモリを確保するため、<<Expand Indices: Size 10>>の入力値に、使用したい配列の要素番号を「..」を使って設定します。

たとえば、図 38 のように「0..9」と設定すると、10 要素分（要素「0」から[9]まで）のメモリが確保できます。

② 入力値を設定します（例：TOM）

③ 必要に応じて期待値を設定します。

④ テストを実行します。

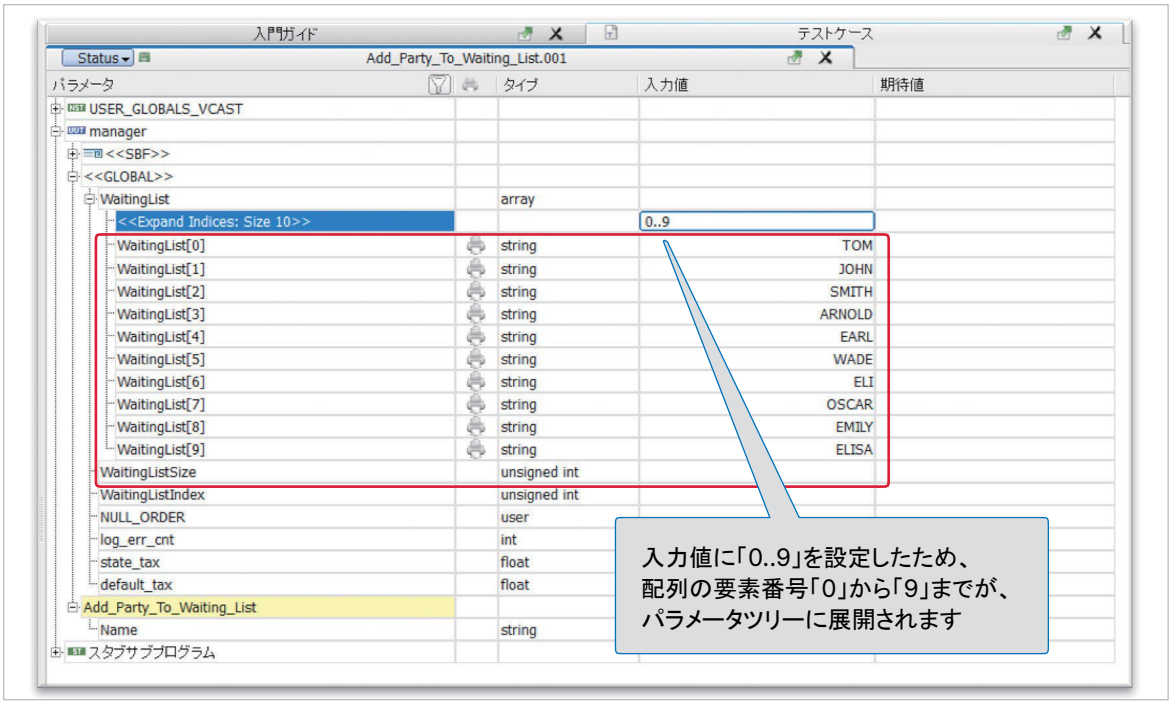


図 38：配列型変数のメモリ確保（10 要素分）

7.2 ポインタ変数の取り扱い方法

ここでは、例に Add_Included_Dessert 関数を使用し、当該関数の仕様より、入力値を「Struct *型」と想定します。VectorCAST ではポインタ変数を入力／出力値に設定する際、テストパラメータ用にメモリを確保してから値を入力する必要があります。

基本的な操作は、「int *」や「char *」型など、どのポインタ型も同様となります。

① 対象となる「パラメータ」の「入力値」／「出力値」欄をクリックし、上下矢印から必要変数分のメモリをポインタ変数に割り当てます。

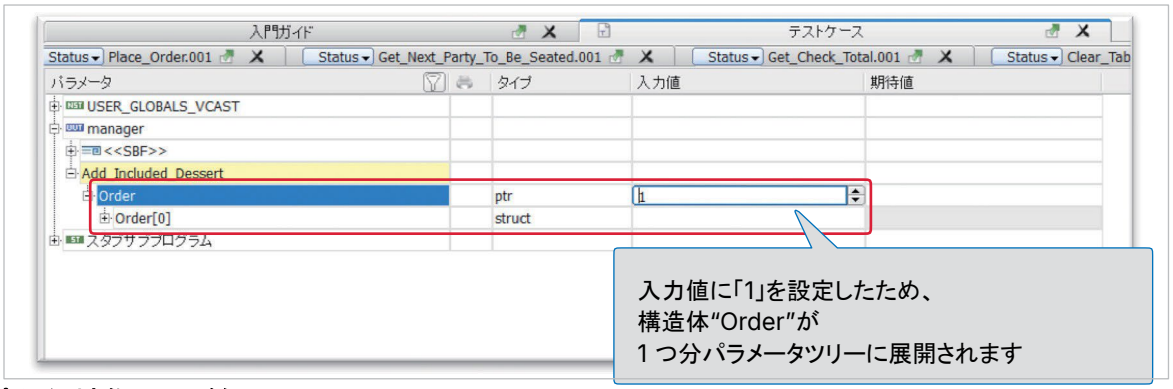


図 39:ポインタ型変数のメモリ確保

② パラメータ名（ここでは、"Order[0]"）の左側の「+」ボタンをクリックし、ツリー展開後に任意の値を設定します。値が enum の場合はプルダウンより値を選択できます。それ以外の値に関しては直接入力してください。

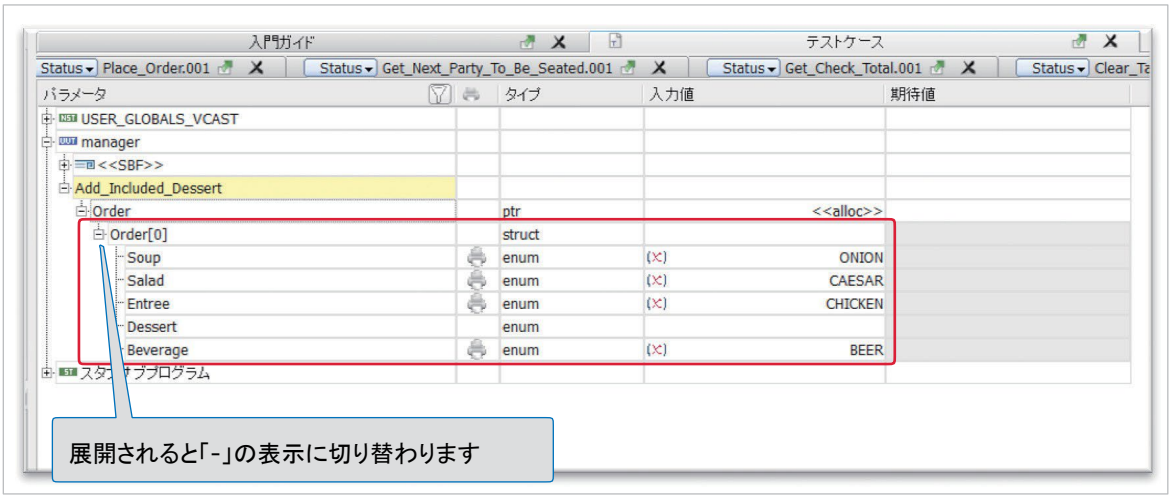


図 40:ポインタ型変数へのテスト値入力

③ 必要に応じて期待値を設定します。

④ テストを実行します。

8. スタブ機能を使おう

本章では、VectorCAST におけるスタブ機能について説明します。

8.1 スタブとは

スタブとは、テスト用に用意されたダミーの関数です。テストにおいて、動作する関数を呼び出すことができない、または呼びたくない場合に使われます。スタブは空関数であり、何も処理を行いませんが、テストケースでスタブの引数や戻り値などを設定して、テスト対象関数へ値を返したり、テスト対象関数から適切な引数でスタブが呼び出されているかを確認したりすることができます。

8.2 スタブの種類

VectorCAST では、スタブを以下の 2 種類に区別しています。

① SBF（スタブ・バイ・ファンクション）

テスト対象のソースファイルに定義されている関数は、SBF の一覧に配置されます。これらの関数は VectorCAST から実際の関数処理内容がわかるため、関数実体を呼び出して処理することも、スタブ化して処理することも可能です。デフォルトでは、関数実体を呼び出して処理する設定となっています。

② スタブサブプログラム

テスト対象のソースファイル以外に定義されている関数で、テスト対象関数から参照されている関数は、スタブサブプログラムの一覧に配置されます。これらは VectorCAST から実際の関数処理内容がわからないため、関数実体を呼び出して処理することはできず、必ずスタブ化して処理されます。

8.3 スタブの設定

ここでは、例に Place_Order 関数とその呼び出し先の関数である Clear_Table 関数を使用し、スタブの設定方法を説明します。

SBF とスタブサブプログラムで、値の設定方法に違いはありません。

VectorCAST では、スタブと実体のいずれで処理するかを切り替える際は、チェックボックスを使用します。スタブサブプログラムは必ずスタブとして処理されるため、切り替え用のチェックボックスがありません。

ここでは、SBF を例に値の設定方法を説明します。

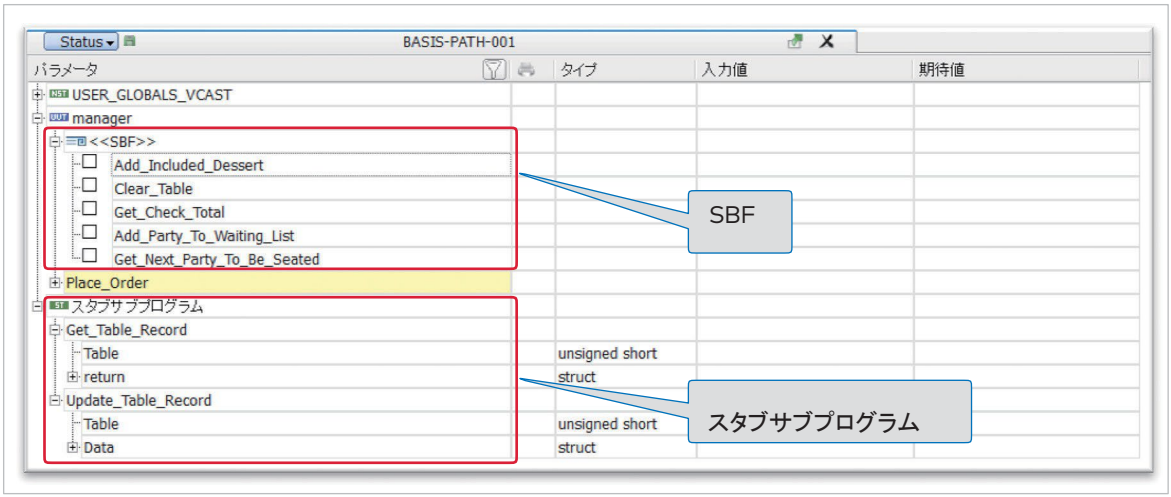


図 41:SBF とスタブサブプログラム

パラメータツリーの<<SBF>>左側の「+」ボタンをクリックすると、SBF が展開されます。図 41 の赤枠で囲っている 5 つの関数はいずれも、“manager.c”で定義されている関数です。関数名の左側のチェックボックスにチェックを入れると、その SBF はスタブとして扱われます。ここでは、Clear_Table 関数を SBF に設定します。

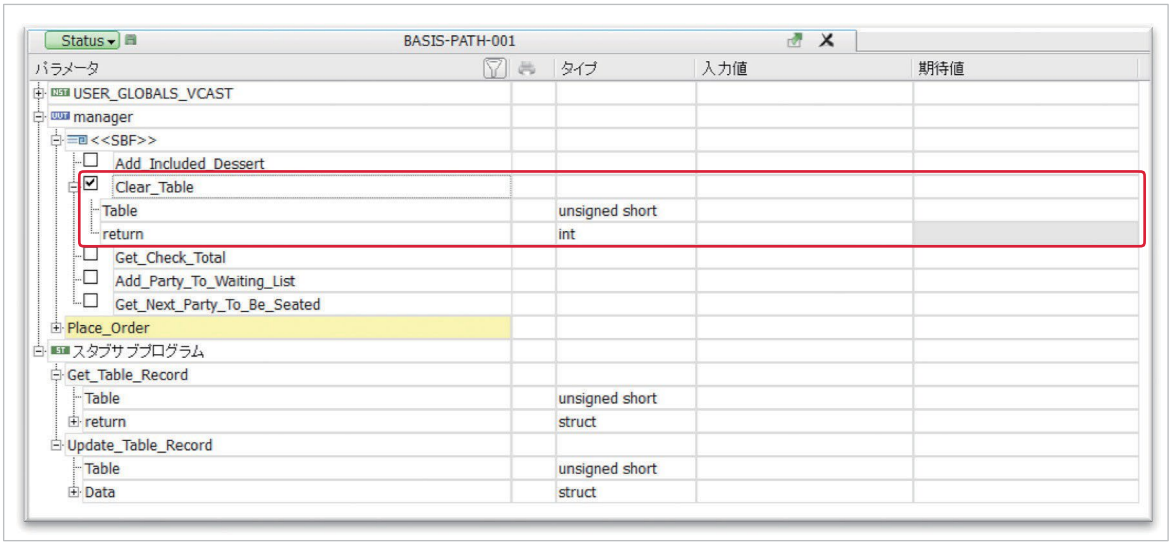


図 42:スタブ設定

チェックボックス横の「+」ボタンをクリックすると、SBF の入力値欄と期待値欄が展開されます。スタブからテスト対象の関数へ戻り値を渡したい場合は、テストケースでスタブの“入力値”に値を設定します。

スタブは基本的に内部処理を行わないため、スタブに引数として値を渡しても結果に影響することはありません。しかし、テストケースでスタブの“期待値”に値を設定することで、テスト対象の関数からスタブへ渡す引数が想定した値となっているかどうかを確認することが可能です。

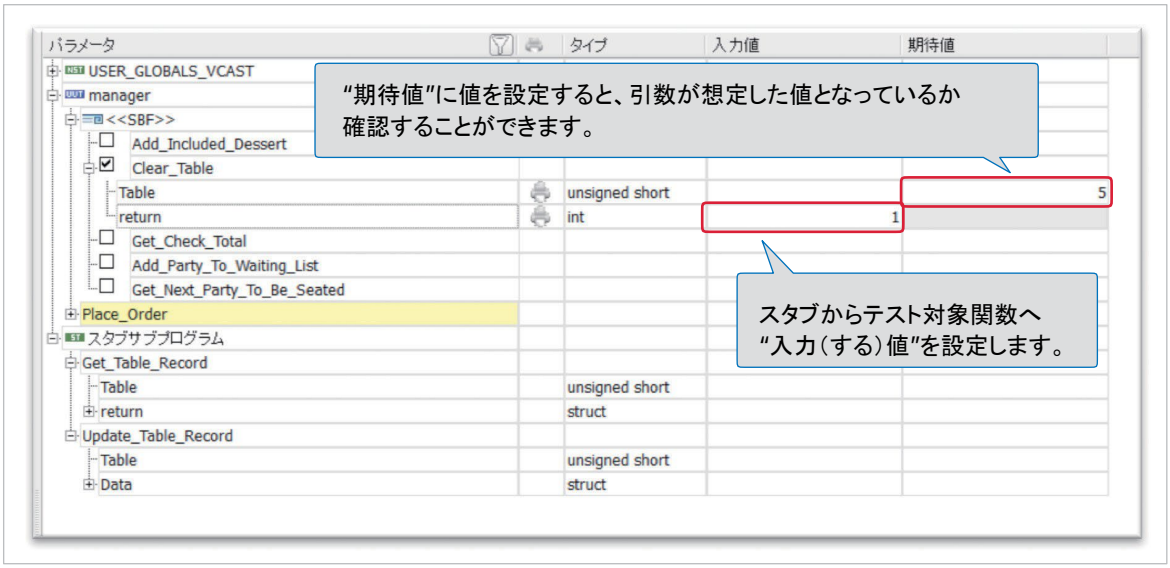


図 43：スタブの入力値と期待値

図 43 では、スタブ関数の戻り値、“return”に“1”を設定しているため、テスト対象関数 Place_Order の処理の中で、Clear_Table が呼ばれた際は、“1”を返す処理が行われます。

9. 変更ベーステスト機能を使おう

本章では、VectorCAST における変更ベーステスト機能の使用方法について説明します。

9.1 変更ベーステスト機能とは

VectorCAST の「変更ベーステスト機能」を使用すると、以前にテストしたソフトウェアに対してソースコードの変更があった際に、VectorCAST がその変更を検出して実施済テストに対する影響範囲を確認し、再実行が必要なテストケースを抽出して実行する機能です。変更ベーステスト機能を使うことで、効率的な回帰テストを行うことが可能です。

9.2 変更ベーステスト機能の使用方法

Add_Included_Dessert 関数を例に、変更ベーステスト機能を使って回帰テストを実施します。ここでは、前提条件として以下を想定しています。

- manager.c に含まれるすべての関数に対して、基本パステストケースを挿入済
- 挿入したすべてのテストケースを実行済

① ソースコードに変更を加えます。例として、Add_Included_Desser t 関数にある「Order->Dessert = PIE;」（図 44、21 行目）を「Order->Dessert = FRUIT;」に変更します。変更したらソースコードを保存してください。

```
15  void Add_Included_Dessert(struct order_type* Order)
16  {
17    if(Order->Entree == STEAK &&
18       Order->Salad == CAESAR &&
19       Order->Beverage == MIXED_DRINK) {
20
21       /* Order->Dessert = PIE; */
22       Order->Dessert = FRUIT;
```

図 44：ソースコード変更例

② テスト環境が開いているときは、VectorCAST はソースコードの変更を検出できません。テスト環境を閉じた状態にすると、VectorCAST がソースコードの変更を検出します。プロジェクトツリーでテスト環境名にマウスをホバーさせると、図 45 のように「！」アイコンで変更を通知されていることがわかります。

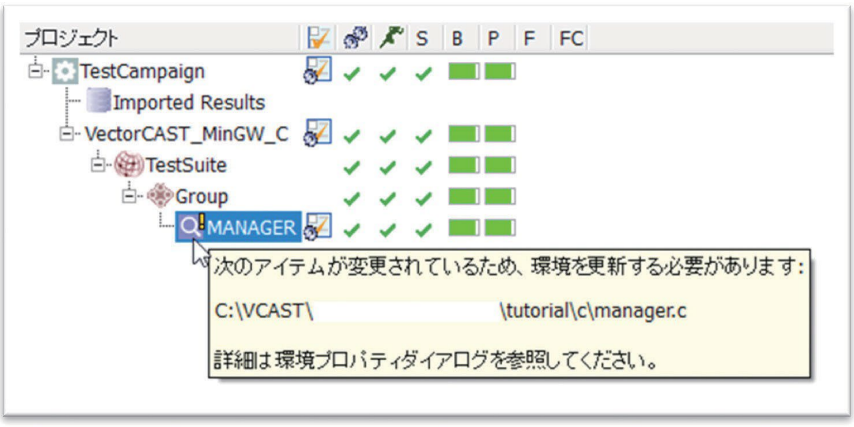


図 45：ソースコードの変更を検出

③ プロジェクトツリーでプロジェクト名を右クリック ⇒「ビルド／実行」⇒「インクリメンタル」を選択します。

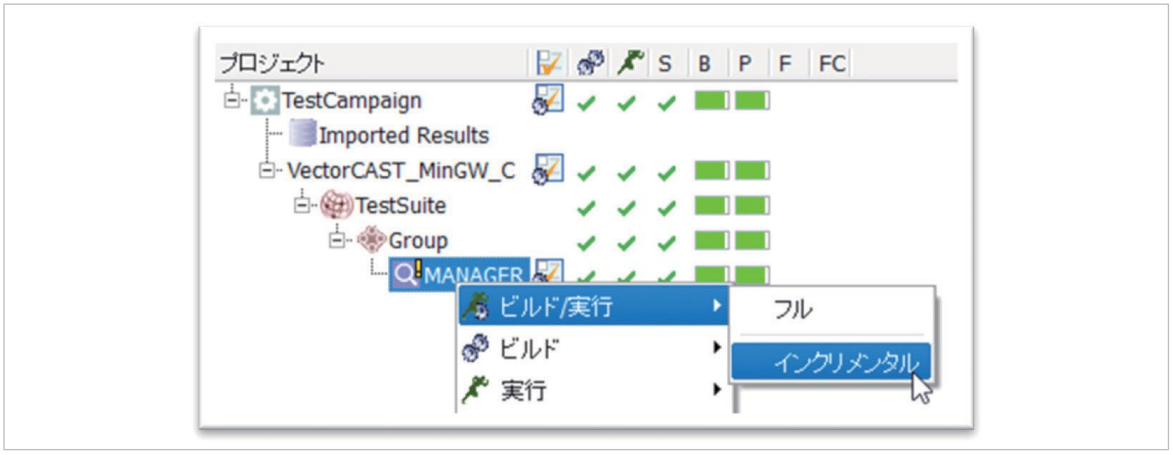


図 46：インクリメンタルリビルド実行

「インクリメンタル」を選択することで、すべてのテストケースではなく、①で行ったソースコードの変更が影響するテストケースのみ、ビルド／実行されます。

④ インクリメンタルリビルドを行うと、実行後にインクリメンタルリビルドレポートが開きます。このレポートでは、ソースコードの変更により影響を受けるテストの数や、リビルド後のステータスが確認できます。

インクリメンタルリビルド

影響を受ける環境

環境	リビルドステータス	影響を受けないテスト	影響を受けるテスト	総テスト
VectorCAST_MinGW_C / TestSuite / MANAGER	Incremental Build Succeeded	8	8	16
合計	1 / 1 (100%)	8	8	16

図 47:インクリメンタルリビルド結果

詳細な情報を確認するためには、環境を開いて「テスト」⇒「表示」⇒「テストケース管理レポート」を選択します。テストケース管理レポートでは、環境内にあるすべてのテストケースのビルド状況を確認することができます。これにより、視覚的に実行状況を把握することが可能です。

テストケース管理

ユニット	サブプログラム	テストケース	実行日時	成功／失敗
manager	Add_Included_Dessert	BASIS-PATH-001	13 JUN 2022 9:48:22 AM	成功
		BASIS-PATH-002	13 JUN 2022 9:48:22 AM	成功
		BASIS-PATH-003	13 JUN 2022 9:48:22 AM	成功
	Place_Order	BASIS-PATH-001	13 JUN 2022 9:48:22 AM	成功
		BASIS-PATH-002	13 JUN 2022 9:48:22 AM	成功
		BASIS-PATH-003	13 JUN 2022 9:48:22 AM	成功
		BASIS-PATH-004	13 JUN 2022 9:48:22 AM	成功
		BASIS-PATH-005	13 JUN 2022 9:48:22 AM	成功
	Clear_Table	BASIS-PATH-001-PARTIAL	10 JUN 2022 6:17:27 PM	成功
		BASIS-PATH-002-PARTIAL	10 JUN 2022 6:17:27 PM	成功
	Get_Check_Total	BASIS-PATH-001	10 JUN 2022 6:17:27 PM	成功
	Add_Party_To_Waiting_List	BASIS-PATH-001	10 JUN 2022 6:17:27 PM	成功
		BASIS-PATH-002	10 JUN 2022 6:17:27 PM	成功
		BASIS-PATH-003	10 JUN 2022 6:17:27 PM	成功
	Get_Next_Party_To_Be_Seated	BASIS-PATH-001	10 JUN 2022 6:17:27 PM	成功
		BASIS-PATH-002	10 JUN 2022 6:17:27 PM	成功
合計	6	16		成功 16 / 16

図 48:テストケース管理レポート

図 48 を見ると、Add_Included_Dessert 関数と Place_Order 関数のテストが同日時に実施されています。これは VectorCAST が、Add_Included_Dessert 関数のソースコード変更が他関数（Place_Order）のテストケースにも影響していると判断して、テストを再実行したことを意味しています。

このように、VectorCAST の変更ベーステスト機能を用いると、影響を受けるテストケースを自動的に判別し、効率良くテストを実行することが可能です。

注意点として、テストケースに期待値が入っている場合は実行結果が失敗になることがあります。その際はテストケースの再設定を行ってください。

また、テストケースへの影響範囲は関数単位で抽出されます。実際には 1 テストケースにしか影響しない場合でも、そのテストケースが含まれる関数全体を回帰テスト対象として抽出します。

10. CI（Jenkins 連携）機能を使おう

本章では、VectorCAST における CI（Jenkins 連携）機能について説明します。

10.1 CI 機能を使用するメリット

CI とは Continuous Integration の略であり、継続的インテグレーションとも呼ばれています。これは、新しいコードを頻繁にマージ（修正／追加部のテスト完了後にメインのブランチへ登録）していくものです。通常、コードをマージするまでの時間が長くなるにつれてコードの変化量が多くなり、テストで問題が発生した場合には、その原因となるコードの特定と分離が難しくなります。コードの変化量を小さくして頻繁にマージをすることで、ソフトウェアの品質を保つとともに、問題の早期発見が可能となり、対応コストを大幅に削減することが可能です。

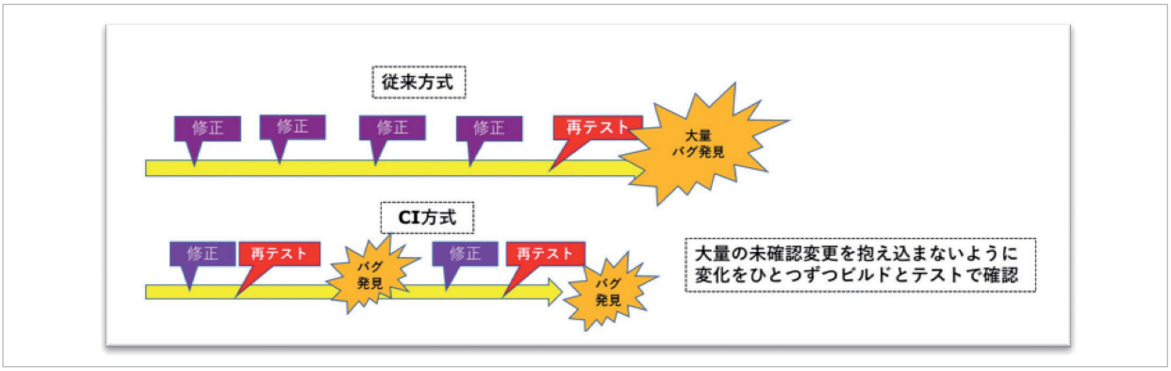


図 49：従来方式のテストのリスク

特に組み込みシステムでは、ビルド自動化のシステム構築に手間が掛かったり、単体テストを自動化することが難しかったりといった問題がありました。CI 方式のテストを実現するためには、以下 3 点を自動化する必要があります。

1. コード変更で影響を受けるテストケースを判別
2. それらのテストケースをコード変更に対して実施
3. テスト結果のレポートを作成し、コードカバレッジを更新

図 50 は、最初のテスト（テスト環境構築も含む）には時間がかかりますが、一度テストケースを作成し、継続的にテストを行う環境を構築することでテスト時間が大きく短縮可能となることを示しています。VectorCAST を使用すると、継続的なテストにおいて、マニュアル（手作業）でのテストに比べて、テスト工数を削減することが可能です。

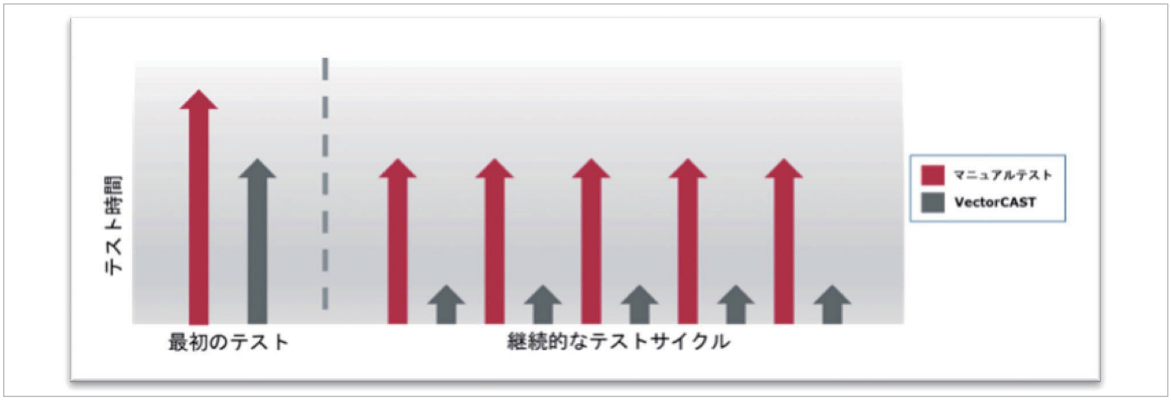


図 50：VectorCAST によるテスト時間削減効果

10.2 VectorCAST における Jenkins 連携機能

Jenkins とは、CI を実現するためのフリーツールの 1 つです。昨今では、さまざまな開発現場で使用されています。VectorCAST では、Jenkins と連携可能なプラグインを提供しており、変更ベーステスト機能と組み合わせて使用することで、より効率的な CI 環境の実現をサポートします。

図 51：Jenkins 運用イメージ

Jenkins 連携機能の詳細な使用方法に関しては、アドバンストマニュアルに記載しています。ベクターの営業窓口までお問い合わせください。

11. CSV ファイルからテストケースを生成しよう

本章では、VectorCAST における CSV ファイルからテストケースを生成する方法について説明します。

11.1 CSV ファイルからテストケースを生成するメリット

すでに CSV ファイルのデータをお持ちの場合や、テストケースが多数ある場合には、CSV ファイルからのテストケース作成が有効です。

11.2 CSV ファイルの作成

ここでは、例に Place_Order 関数を使用します。

まず、VectorCAST がデータ読み込み可能な CSV テンプレートを作成し、そこに任意のデータを設定した「CSV ファイル」を作成します。

① テスト環境を開き、テストケースツリーからテスト対象の関数を選択します。その後、右クリック⇒「CSV マップを作成」を選択します。

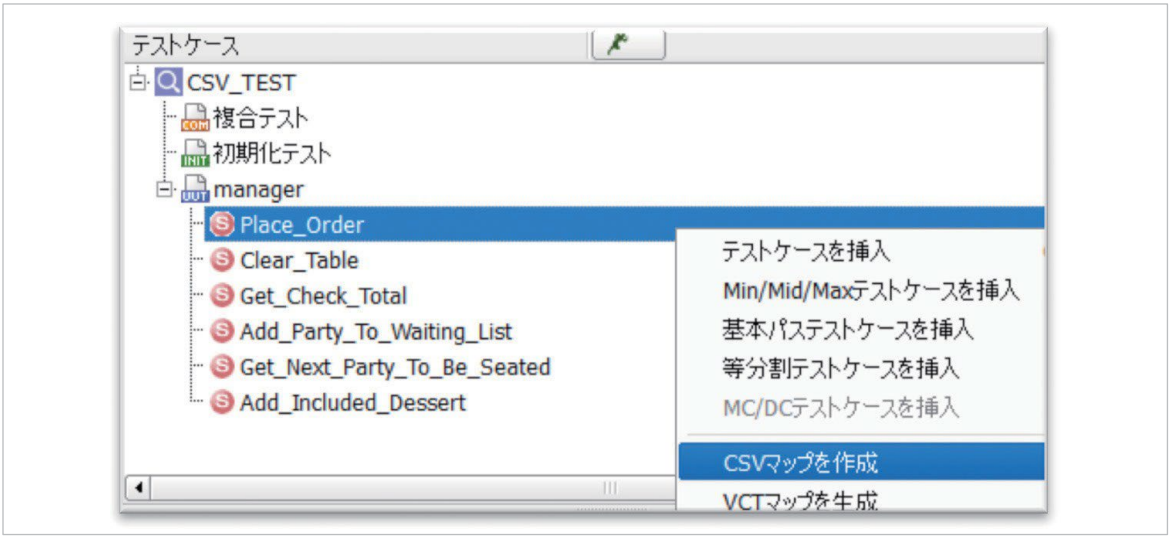


図 52:CSV マップを作成

② CSV マップが表示されます。「テンプレートを作成」を選択し、保存先を指定します。

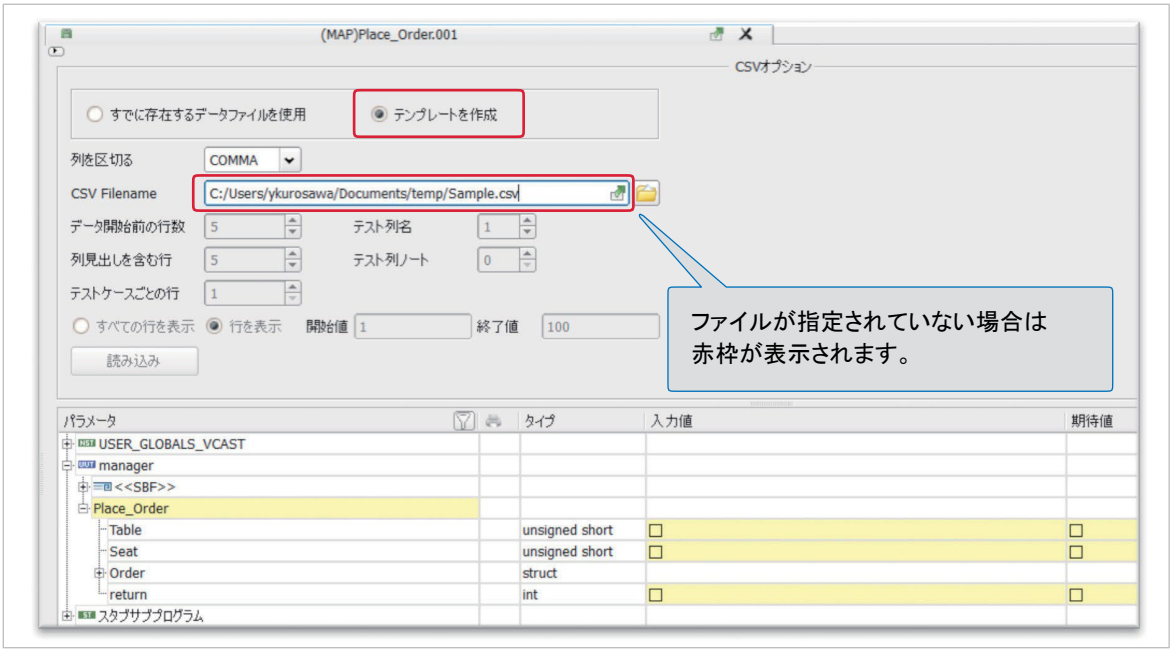


図 53:テンプレートを作成

③ 次にテンプレートを構成する入力値と期待値を選択します。

入力値として選択する場合は入力値欄のチェックボックスを、期待値として選択する場合は期待値欄のチェックボックスにチェックを入れます。また、チェックボックスの隣に表示されている数字はテンプレートの列番号となります。(1 列目はテストケース名となるため、パラメータは 2 列目以降に配置されます)

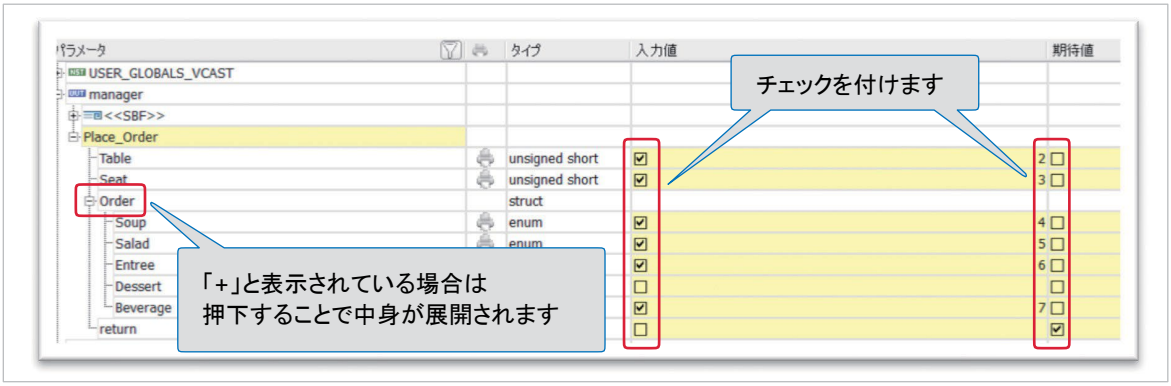


図 54:テンプレートを構成する入力値と期待値の選択

④　選択完了後にテストケースツリーに表示されている CSV マップを選択し、右クリック ⇒「CSV テンプレートを作成」を選択します。

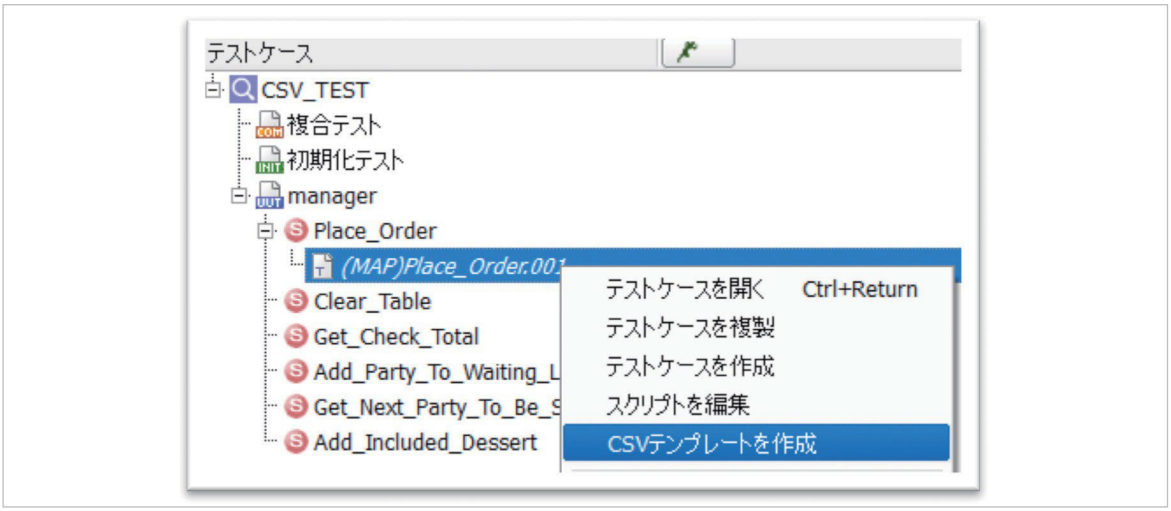


図 55:CSV テンプレートを作成

⑤　保存された CSV テンプレートを開き、テンプレートが作成されていることを確認します。

	A	B	C	D	E	F	G	H
1	Test Name	Input	Input	Input	Input	Input	Input	Expected
2	Test Name	manager	manager	manager	manager	manager	manager	manager
3	Test Name	Place_Order	Place_Order	Place_Order	Place_Order	Place_Order	Place_Order	Place_Order
4	Test Name	unsigned short	unsigned short	enum soups	enum salads	enum entrees	enum beverages	int
5	Test Name	Table	Seat	Order.Soup	Order.Salad	Order.Entree	Order.Beverage	return
6								
7								

図 56:CSV テンプレート例

CSV テンプレートは図 56 のように作成されます。1 行目はパラメータの種類、2 行目はソースファイル名、3 行目は対象の関数名、4　行目はパラメータの型、5　行目はパラメータ名を表示しています。設定した内容が正しく反映されていることを確認します。

次にテストケースのデータを入力します。図 57 は設定例です。

	A	B	C	D	E	F	G	H
1	Test Name	Input	Input	Input	Input	Input	Input	Expected
2	Test Name	manager	manager	manager	manager	manager	manager	manager
3	Test Name	Place_Order	Place_Order	Place_Order	Place_Order	Place_Order	Place_Order	Place_Order
4	Test Name	unsigned short	unsigned short	enum soups	enum salads	enum entrees	enum beverages	int
5	Test Name	Table	Seat	Order.Soup	Order.Salad	Order.Entree	Order.Beverage	return
6	Case1	1	1	NO_SOUP	NO_SALADA	STEAK	WINE	0
7	Case2	1	2	ONION	CEASAR	CHICKEN	BEER	0
8	Case3	2	1	CHOWDER	GREEN	LOBSTER	SODA	0

図 57:CSV テンプレート テストケースのデータ入力例

これで、CSV ファイルの作成は完了です。次は実際にテストケースを作成します。

11.3 CSV ファイルからのテストケース生成

前項に続き、例に Place_Order 関数を使用して CSV ファイルからテストケースを生成します。

① テスト環境を開き、テストケースツリーからテスト対象の関数を選択します。その後、右クリック ⇒「CSV マップを作成」を選択します。11.2 で作成した CSV マップがある場合は、そちらを使用できます。

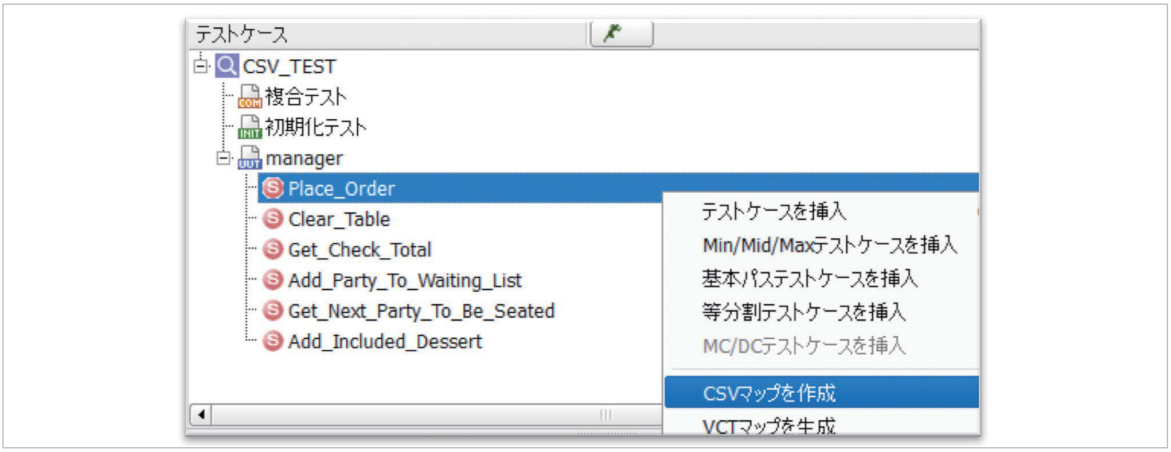


図 58：CSV マップを作成

② 「すでに存在するデータファイルを使用」を選択し、データを取り込みたい CSV ファイルの格納先を指定します。その他、表示されている項目に任意の値を設定し、「読み込み」を押下します。

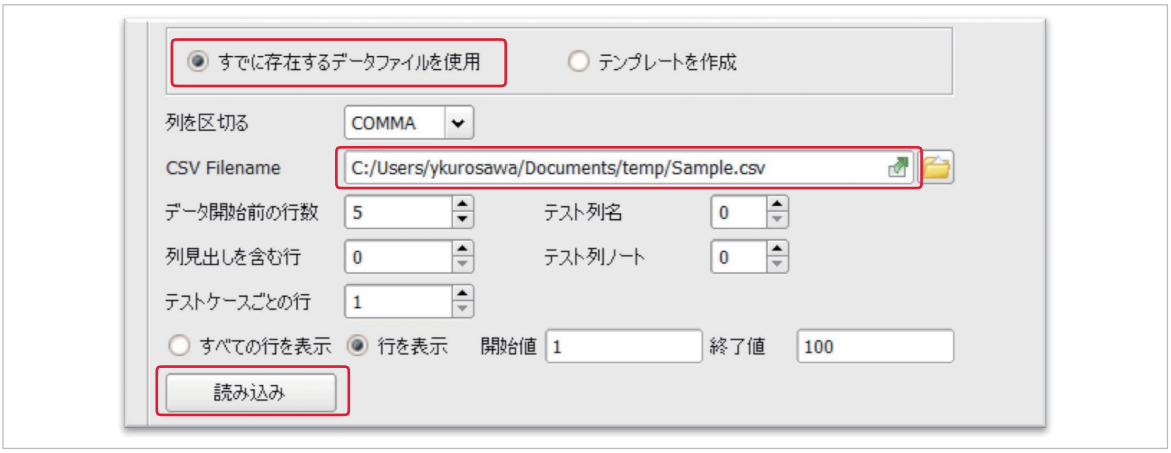


図 59：データファイルを読み込み設定

③　CSV ファイルが読み込まれると、図 60 のように表示されます。読み込み内容を確認します。

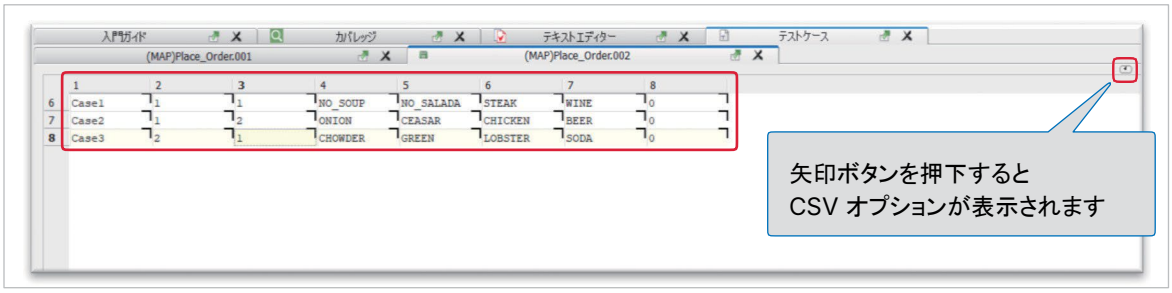


図 60:CSV ファイル読み込み結果

④　読み込んだテストケースのデータを、設定したいテストケースの入力値や出力値にマッピングします。どれでも構わないので、1 つのテストケースのすべてのデータを、対応するテストケースの入力値や出力値の欄へドラッグ &ドロップします。データが設定可能なパラメータは、図 61 のように黄色のハイライトで表示されています。

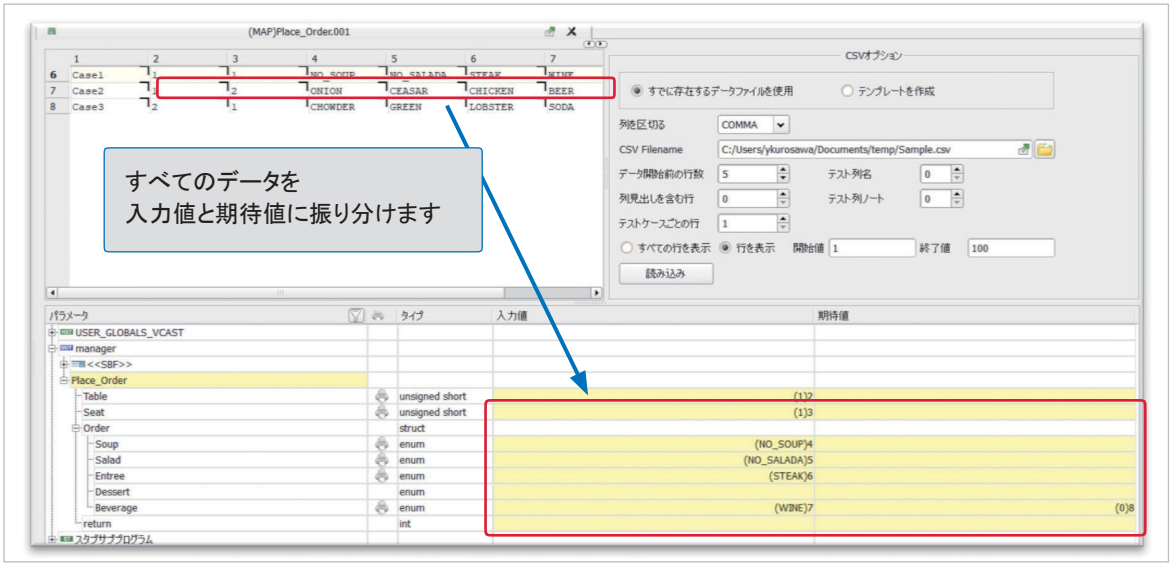


図 61:読み込んだデータからテストパラメータへの振り分け

⑤　テストケースツリーに表示されている CSV マップを選択し、右クリック⇒「テストケースを作成」を選択します。

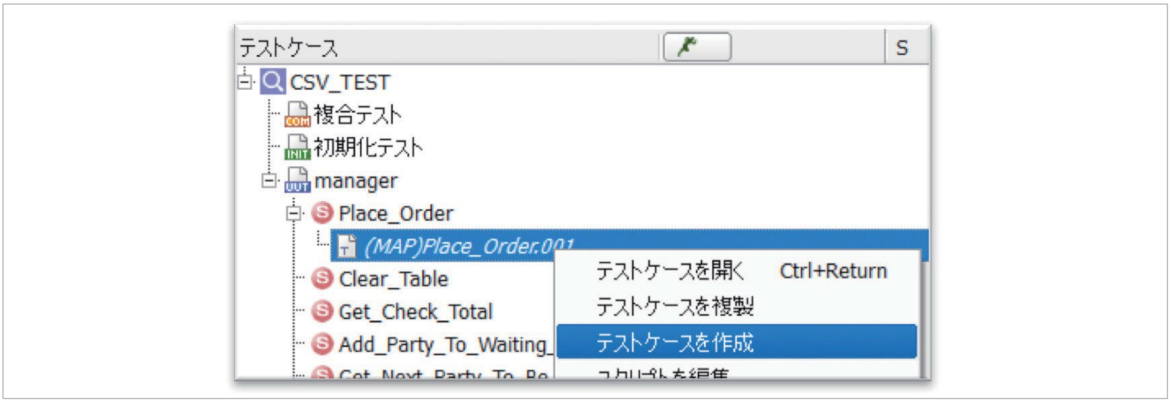


図 62:CSV マップからテストケースを作成

テストケースが正しく生成されたことを確認し、テストを実行します。

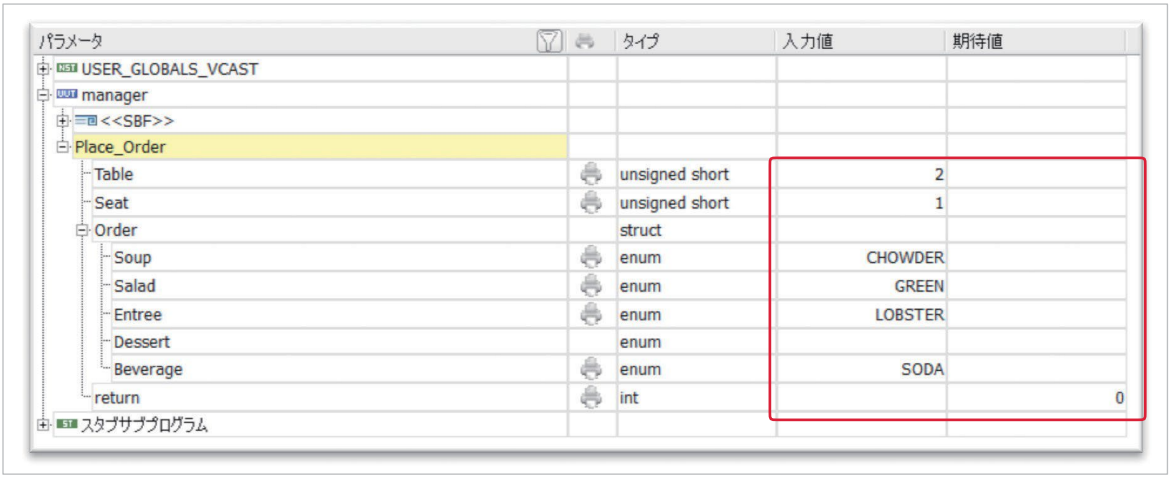


図 63:テストケース作成結果

12. プローブポイントを使おう

本章では、VectorCAST におけるプローブポイントの使用方法について説明します。

12.1 プローブポイントとは

プローブポイントは、オリジナルのソースコードには変更を加えず、任意のコードを追加することで、テストやカバレッジをパスするための手段として用いられます。特に、ローカル変数の値を変更したい場合や、強制的に分岐を変更したい場合などに有効です。また、printf 関数などを処理の中に追加することができるので、簡易的なデバッグの手段としても使用可能です。

12.2 プローブポイントの挿入

ここでは、例に Place _Order 関数を使用し、プローブポイントの挿入方法について説明します。

ある処理の値を強制的に変更するようなケースを例とします。注意点は、プローブポイントは環境内のテストケースすべてに適用される点です。テストケースごとに設定したい場合は、後述のユーザーコードと組み合わせる必要があります。

① テスト環境を開き、上部の「プローブポイントを編集」アイコンをクリックします。

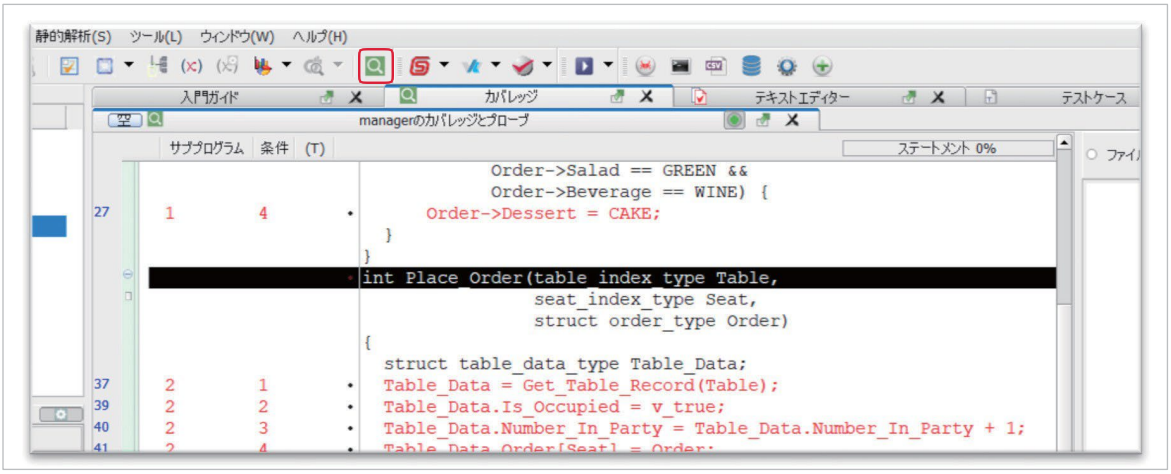


図 64：プローブポイント

② プローブポイントの編集画面より、プローブポイントを挿入したい箇所を黒点の中から選択します。黒点の表示が緑色のプローブポイントマークに変化することを確認してください。

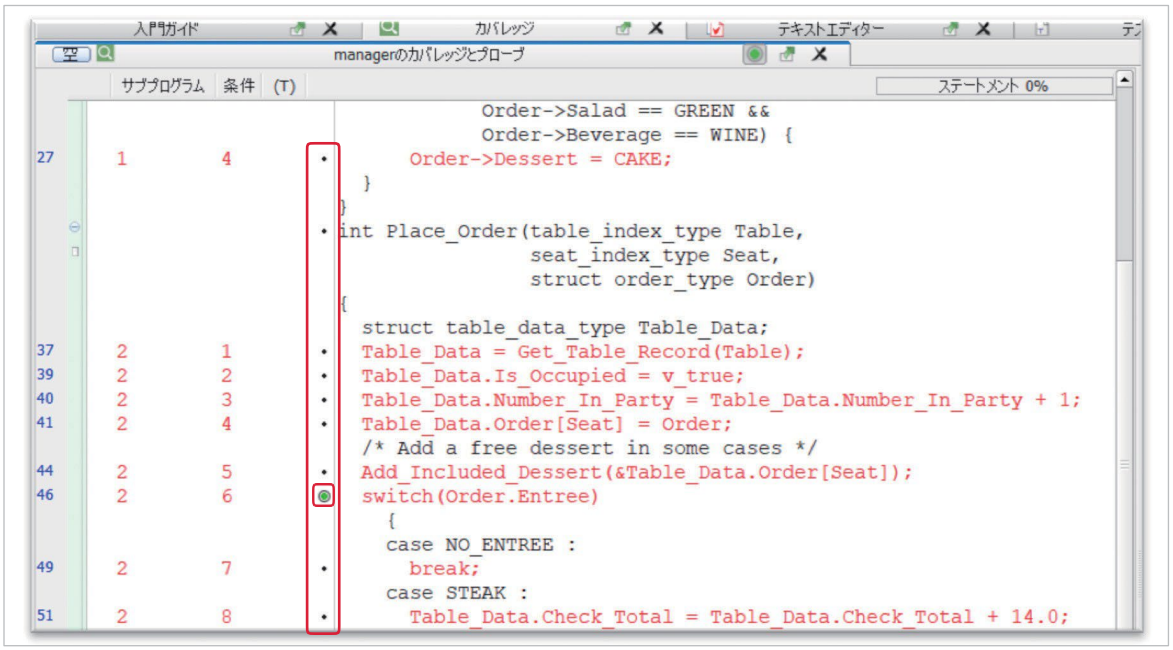


図 65:プローブポイント挿入箇所を選択

③ VectorCAST の右側に表示されているプローブポイントの記載欄へ、追加したいコードを記載します。例として、「Order.Entree」の内容を任意に変えるようなコードを使用しています。

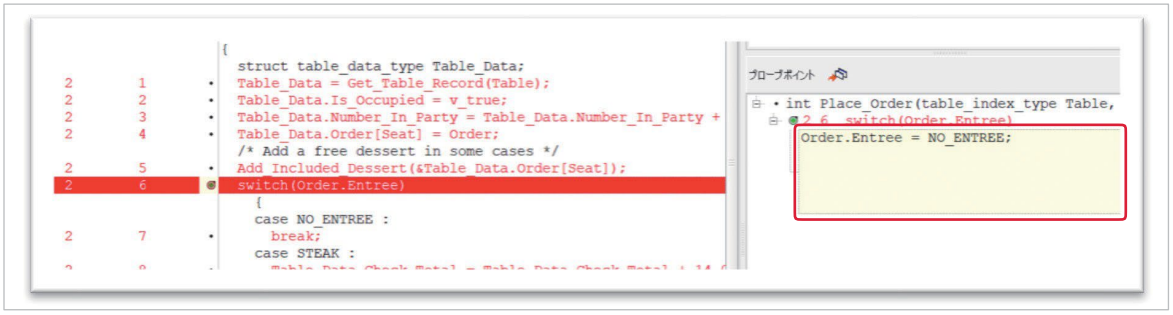


図 66:プローブポイントを編集

図 66 のように設定することで、「switch(Order.Entree)」のソースコードの前に「Order.Entree = NO_ENTREE」が挿入されるため、この switch-case 文は必ず「NO_ENTREE」として処理が行われます。

④ コードを入力後、「テストコンパイル」アイコンを押下し、コンパイルが成功することを確認します。コンパイルが成功しない場合は、エラーメッセージが表示されるので、内容を確認の上、コードの見直しを行ってください。

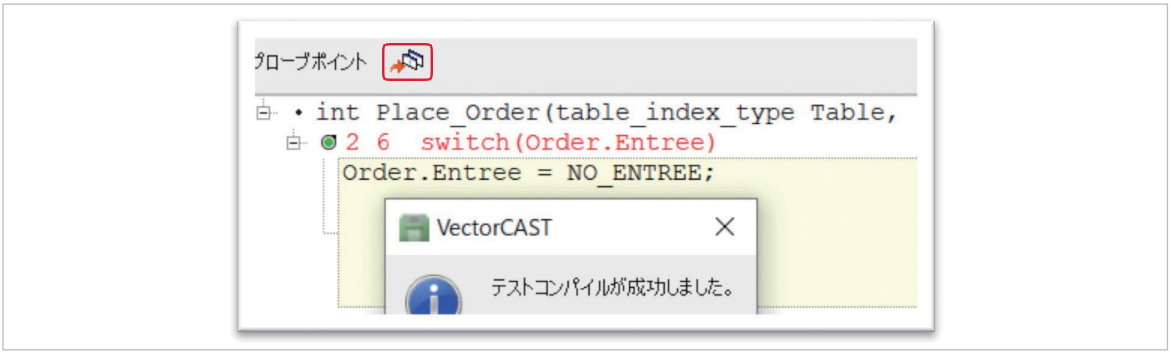


図 67:テストコンパイルの実施

⑤ テストコンパイル完了後、プローブポイントの保存と、環境のリビルドを行います。

プローブポイント編集画面上部の「未保存」と表示された部分を押下し、「適用済み」と表示が変更されることを確認してください。その後、環境をリビルドし、テストを実行します。

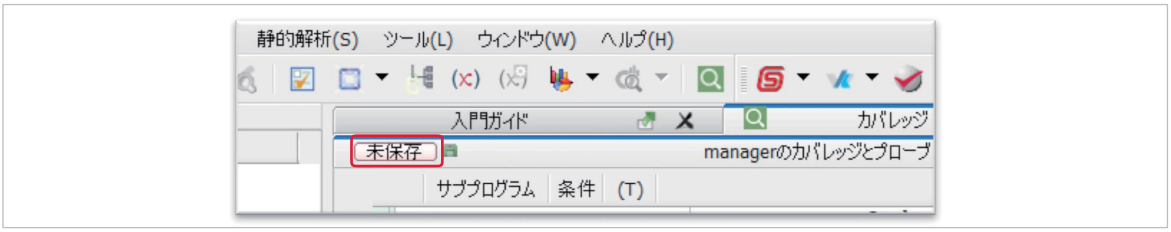


図 68:編集内容の保存

プローブポイントの保存、および環境のリビルドを実施していないとプローブポイントは反映されないため、ご注意ください。

12.3 プローブポイントとユーザーコード

テストケースごとにプローブポイントを設定したい場合は、ユーザーコードと組み合わせて使用します。ユーザーコードとは、ユーザーが任意のテスト用コードをテスト対象のソースコードに追加できる、VectorCAST の機能です。ここでは、デフォルトで定義されているユーザーコードを使用し、テストケースごとに任意のプローブポイントを設定する方法を説明します。

① 「環境」⇒「ユーザーコード」⇒「編集」と進み、「ユーザーグローバル」を展開します。使用するユーザーグローバル（VECTORCAST_INT1）が予め定義されていることを確認します。

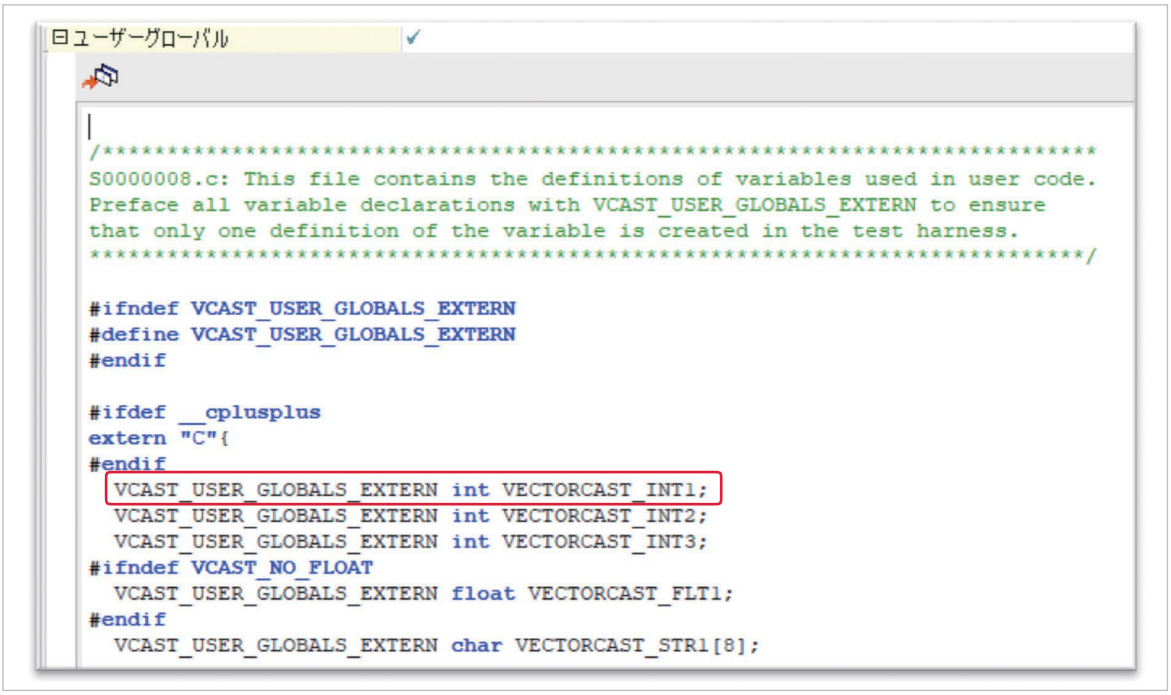


図 69：ユーザーグローバル

次にテストケースからパラメータツリーを開き、「USER_GLOBALS_VCAST」⇒「<<GLOBAL>>」を展開し、ユーザーグローバル（VECTORCAST_INT1）が表示されていることを確認します。

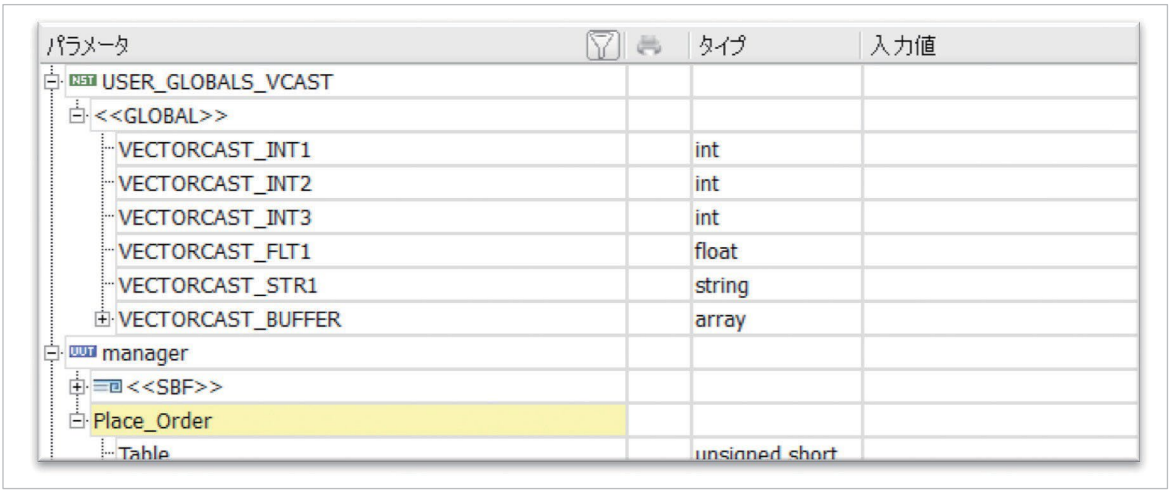


図 70：ユーザーグローバルの確認

② プローブポイント設定画面で、以下のように入力します

```
extern int VECTORCAST_INT1;

if(VECTORCAST_INT1 == 1){

        Order.Entree = NO_ENTREE;

    }else if(VECTORCAST_INT1 == 2){

        Order.Entree = CHICKEN;

}
```

これは、ユーザーグローバル（VECTORCAST_INT1）の値に応じて、Order.Entree へ代入する値を変えるコードとなっています。

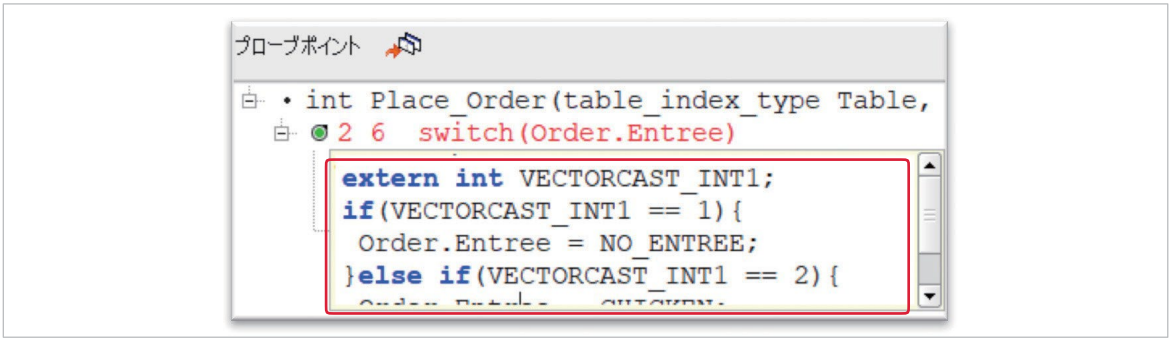


図 71：ユーザーコードでプローブポイントをコントロール

③ テストコンパイル、プローブポイントの保存を行い、環境をリビルドします。ビルド完了後に、パラメータツリーの「VECTORCAST_INT1」へ、任意の値を設定します。図 72 のテストケース（Place_Order.001）には「1」、図 73 のテストケース（Place_Order.002）には「2」を設定します。

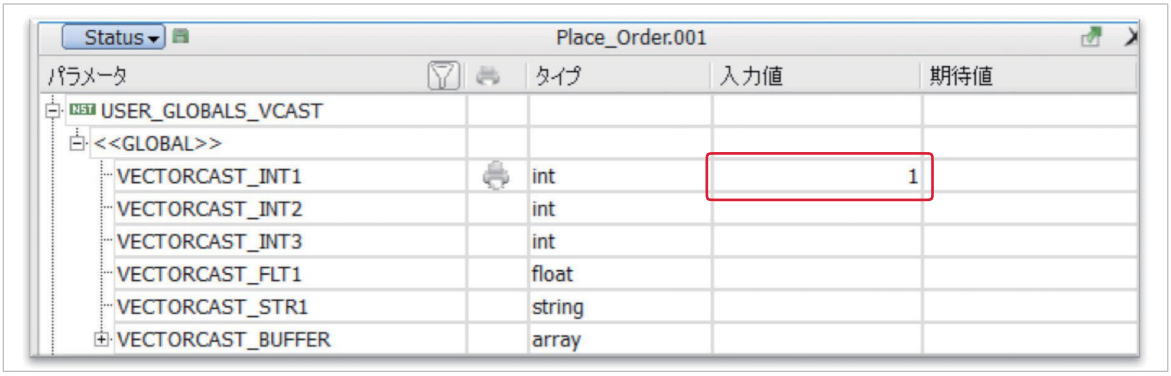


図 72：ユーザーグローバルでフラグ（1）を設定

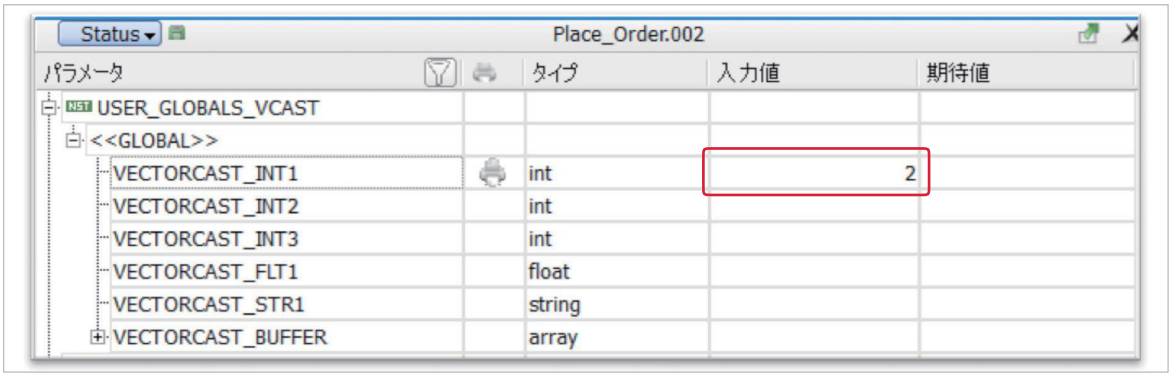


図 73:ユーザーグローバルでフラグ(2)を設定

設定したら、各テストケースを実行します。図 74 のように、テストケースごとに分岐が異なっていることが確認できます。

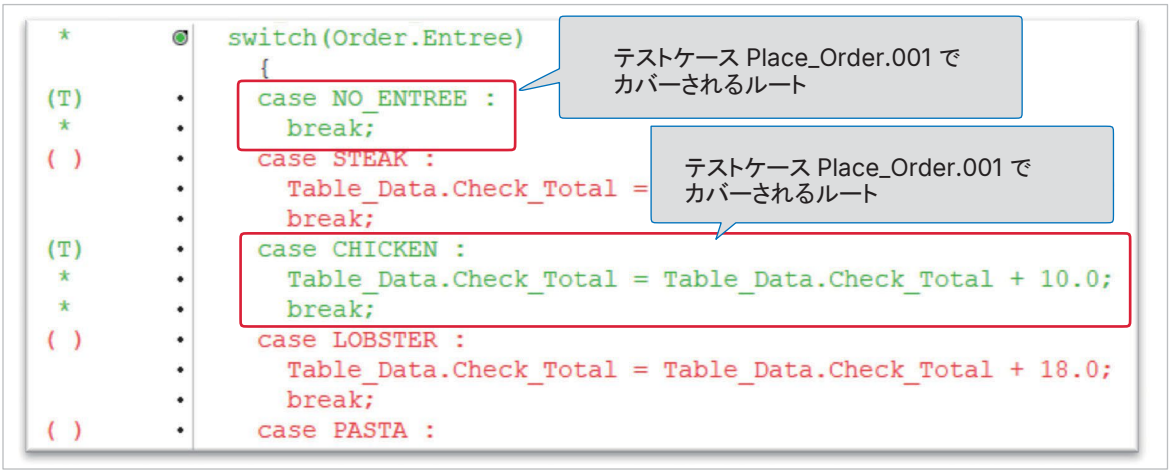


図 74:カバレッジ結果の確認

以上のように、処理の流れを強制的に変更したい場合や、ローカル変数の値を変更したい場合には、プローブポイントが有効な手段となります。またユーザーコードは、前述のようなケースで使用することが可能です。その他のユーザーコードの使用方法に関しては、アドバンストマニュアルに記載しています。詳細は、ベクターの営業窓口までお問い合わせください。

13. 結合テストを実施しよう

本章では、VectorCAST における結合テストについて説明します。

13.1 VectorCAST における結合テスト

ここまでの章では、ある 1 つの関数に着目し、その関数の入力に対する出力が正しいか、カバレッジを正しく取得できているか、といった観点でテストを実施してきました。

テストの中には、たとえばファイルへの書き込み処理のように、「ファイルのオープン」⇒「ファイルへの書き込み」⇒「ファイルのクローズ」という処理が順番に実施されなければ意味を成さないテストもあります。また、この一連の処理の中で、ファイルポインタはグローバル変数として保持される必要があります。

VectorCAST では、このようにグローバル変数を保持しつつ、複数のテストケースを順番に実施したい場合、結合テスト機能を使って実現します。

13.2 結合テストの実施方法

先に述べたように、ファイルへの書き込み時は「ファイルのオープン」⇒「ファイルへの書き込み」⇒「ファイルのクローズ」といった処理が行われます。これらをそれぞれ、OpenFile、WriteString、CloseFile の 3 つの関数に機能分割して実装したとします。

結合テストを作成する前に、必要となる入力値と期待値を設定したテストケースを用意します。図 75 では、3 つの関数に対して OPEN_WRITE、WRITE_DATA、CLOSE というテストケースを用意しています。

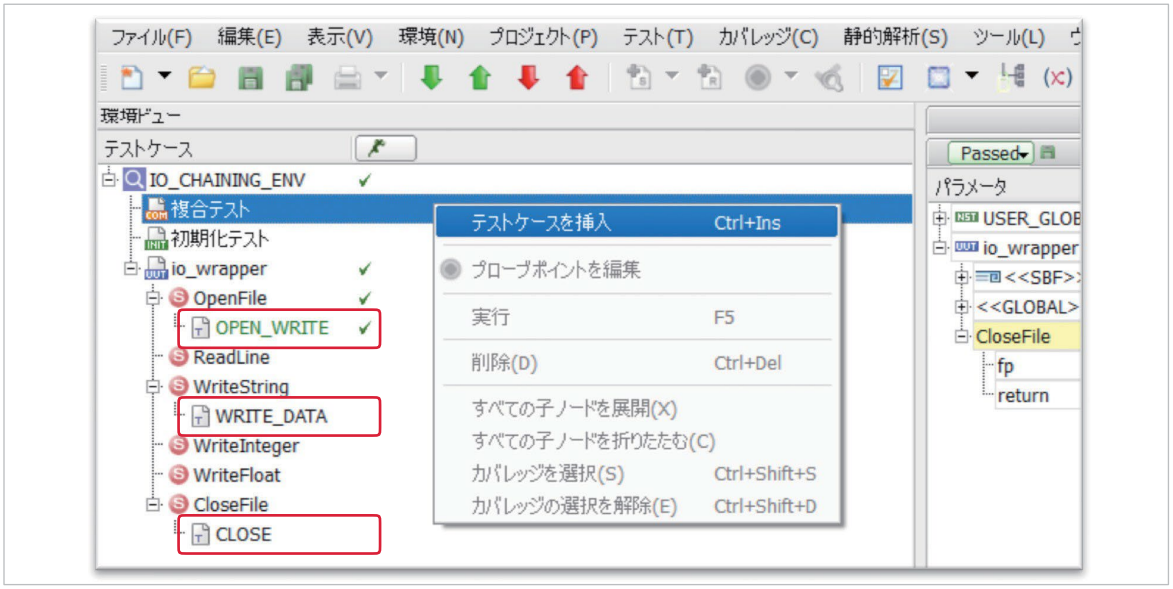


図 75:結合テストケースを挿入

次に、テストケースツリーの“複合テスト”ノードを右クリックし、コンテキストメニューから「テストケースを挿入」を選択します。テストケースには任意の名前を付けることができます。画面右側に複合ツリーというタブが表示されるので、テストケースを実施したい順番に複合ツリータブへドラッグ&ドロップします。スロット列が、テストケースの実施順を表しています。

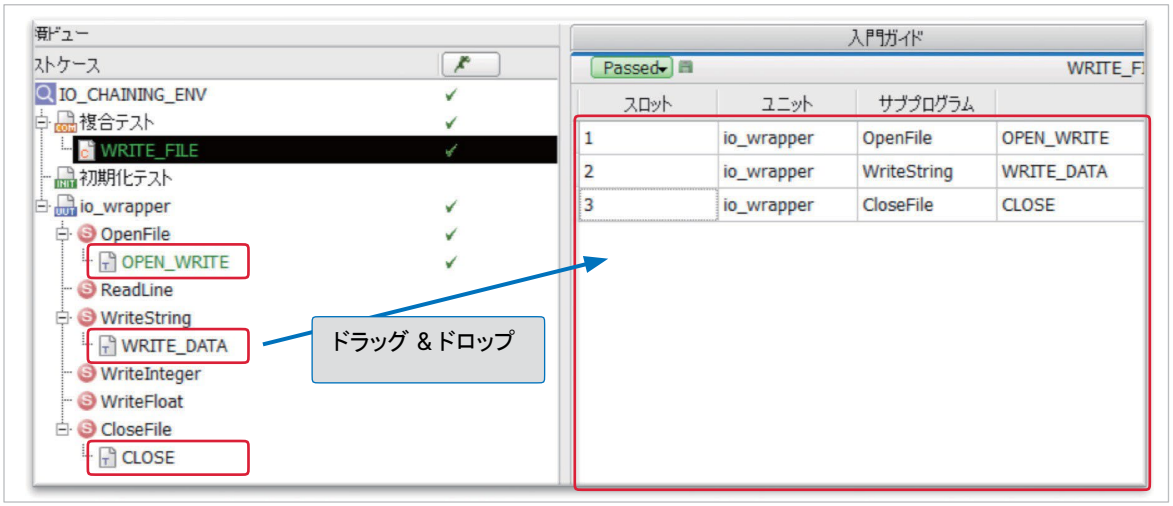


図 76:結合テストケースを編集

テストケースの配置が完了したら、テストを実行します。

テストケースが正常に実行されたら、緑色のハイライトで表示されます。また、詳細な実行結果は実行レポートに表示されます。なお、OPEN_WRITE で作成したファイルポインタは、グローバル変数として WRITE_DATA、CLOSE でも引き継がれて使用されています。

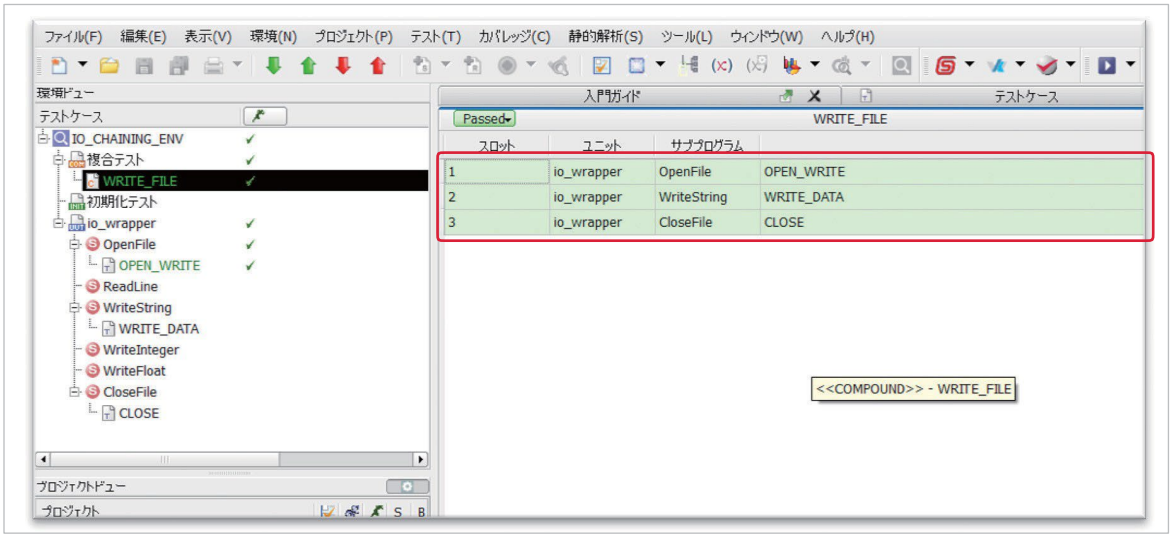


図 77:結合テストケースを実施

このように、複合テスト機能を使用することで、簡単に複数の関数をシーケンシャルにテストすることが可能です。

14. あとがき

本稿『はじめての VectorCAST』が VectorCAST についてのすべてを網羅しているわけではありませんが、VectorCAST の基本的な機能はご理解いただけたのではないかと思います。VectorCAST について、さらに詳細をお知りになりたい方は、ベクターWeb サイト「ベクター・ジャパン Member Area」に「アドバンストマニュアル」など各種ドキュメントをご用意しておりますので、是非、ご覧ください。ツールに関するご相談、お問い合わせは、お気軽にベクター・ジャパンまでお寄せください。

以上、本稿が皆様の開発業務の一助となれば幸いです。

> はじめての単体テスト
> **www.vector.com/jp/ja/know-how/vj-beginners/unittest-jp/**

> VectorCAST 紹介ページ「世界で選ばれる単体テストツール VectorCAST」はこちら
> **www.vector.com/jp/ja/products/products-a-z/software/vectorcast/vj-unit-testing-solution/**

> ベクター・ジャパン Member Area
> **www.vector.com/jp/ja/support-downloads/vj-memberarea/**

15. 参考資料

manager.c

```
#include "ctypes.h"

struct table_data_type Get_Table_Record(table_index_type Table);
void Update_Table_Record(table_index_type Table, struct table_data_type Data);

/* Allow 10 Parties to wait */
static name_type WaitingList[10];
static unsigned int WaitingListSize = 0;
static unsigned int WaitingListIndex = 0;



/* This function will add a free dessert to specific orders based on the
   entree, salad, and beverage choice */
void Add_Included_Dessert(struct order_type* Order)
{
  if(Order->Entree == STEAK &&
     Order->Salad == CAESAR &&
     Order->Beverage == MIXED_DRINK) {

    Order->Dessert = PIE;

  } else if(Order->Entree == LOBSTER &&
            Order->Salad == GREEN &&
            Order->Beverage == WINE) {

    Order->Dessert = CAKE;
  }
}
```

```
31 int Place_Order(table_index_type Table,
32                 seat_index_type Seat,
33                 struct order_type Order)
34 {
35   struct table_data_type Table_Data;
36
37   Table_Data = Get_Table_Record(Table);
38
39   Table_Data.Is_Occupied = v_true;
40   Table_Data.Number_In_Party = Table_Data.Number_In_Party;
41   Table_Data.Order[Seat] = Order;
42
43   /* Add a free dessert in some cases */
44   Add_Included_Dessert(&Table_Data.Order[Seat]);
45
46   switch(Order.Entree)
47     {
48     case NO_ENTREE :
49        break;
50     case STEAK :
51        Table_Data.Check_Total = Table_Data.Check_Total + COST_OF_STEAK;
52        break;
53     case CHICKEN :
54        Table_Data.Check_Total = Table_Data.Check_Total + COST_OF_CHICKEN;
55        break;
56     case LOBSTER :
57        Table_Data.Check_Total = Table_Data.Check_Total + COST_OF_LOBSTER;
58        break;
59     case PASTA :
60        Table_Data.Check_Total = Table_Data.Check_Total + COST_OF_PASTA;
61        break;
62     }
63
64   Update_Table_Record(Table, Table_Data);
65   return 0;
66 }
```

```
67
68  int Clear_Table(table_index_type Table)
69  {
70    struct table_data_type Table_Data;
71    seat_index_type Seat;
72    Table_Data = Get_Table_Record(Table);
73    Table_Data.Is_Occupied = v_false;
74    Table_Data.Number_In_Party = 1;
75
76    for (Seat=0; Seat < SEATS_AT_ONE_TABLE; Seat++){
77        Table_Data.Order[Seat].Soup     = NO_SOUP;
78        Table_Data.Order[Seat].Salad    = NO_SALAD;
79        Table_Data.Order[Seat].Entree   = NO_ENTREE;
80        Table_Data.Order[Seat].Dessert  = NO_DESSERT;
81        Table_Data.Order[Seat].Beverage = NO_BEVERAGE;
82         }
83
84    Table_Data.Check_Total = 0;
85
86    Update_Table_Record(Table, Table_Data);
87    return 0;
88  }
89
90  FLOAT Get_Check_Total(table_index_type Table)
91  {
92    struct table_data_type Table_Data;
93    Table_Data = Get_Table_Record(Table);
94    return (Table_Data.Check_Total);
95  }
96
97  void Add_Party_To_Waiting_List(char* Name)
98  {
99    int i = 0;
100   if(WaitingListSize > 9)
101     WaitingListSize = 0;
102
103   while(Name && *Name) {
104     WaitingList[WaitingListSize][i++] = *Name;
105     Name++;
106   }
107   WaitingList[WaitingListSize++][i] = 0;
108 }
109
110 char* Get_Next_Party_To_Be_Seated(void)
111 {
112   if(WaitingListIndex > 9)
113     WaitingListIndex = 0;
114   return WaitingList[WaitingListIndex++];
115 }
116
```

ctypes.h

```
1  #ifndef _TUTORIAL_TYPES_H_
2  #define _TUTORIAL_TYPES_H_
3
4  #define SEATS_AT_ONE_TABLE  4
5  #define NUMBER_OF_TABLES    6
6
7  #ifdef VCAST_NO_FLOAT
8  typedef int FLOAT;
9  #define COST_OF_STEAK   14
10 #define COST_OF_CHICKEN 10
11 #define COST_OF_LOBSTER 18
12 #define COST_OF_PASTA   12
13 #else
14 typedef float FLOAT;
15 #define COST_OF_STEAK   14.0
16 #define COST_OF_CHICKEN 10.0
17 #define COST_OF_LOBSTER 18.0
18 #define COST_OF_PASTA   12.0
19 #endif
20
21 enum boolean   { v_false, v_true };
22 enum soups     { NO_SOUP, ONION, CHOWDER };
23 enum salads    { NO_SALAD, CAESAR, GREEN };
24 enum entrees   { NO_ENTREE, STEAK, CHICKEN, LOBSTER, PASTA };
25 enum desserts  { NO_DESSERT, CAKE, PIE, FRUIT };
26 enum beverages { NO_BEVERAGE, WINE, BEER, MIXED_DRINK, SODA };
27
28 struct order_type
29 {
30   enum soups     Soup;
31   enum salads    Salad;
32   enum entrees   Entree;
33   enum desserts  Dessert;
34   enum beverages Beverage;
35 };
36
37 typedef unsigned short seat_index_type;
38 typedef unsigned short table_index_type;
39
40 struct table_data_type
41 {
42   enum boolean      Is_Occupied;
43   seat_index_type   Number_In_Party;
44   char              Designator;
45   char              Wait_Person[10];
46   struct order_type Order[SEATS_AT_ONE_TABLE];
47   FLOAT             Check_Total;
48 };
49
50 typedef char name_type[32];
51
52 #endif /* _TUTORIAL_TYPES_H_ */
53
```

16. お問い合わせ窓口

技術関連のお問い合わせは、ベクターサポートまでお寄せください。

✓ サポートリクエスト（Web サイト経由のサポート問い合わせフォーム）
www.vector.com/jp/ja/support-downloads/support/

【お問い合わせ時のお願い】

お問い合わせの際、サポート開始前に製品のバージョン（例：CANape バージョン xx.x など）、シリアルナンバーなどをお伺いいたします。お手数ではございますが、事前にご確認のうえ、お問い合わせくださいますようお願い申し上げます。

【サポート対応について】

夜間／休日に頂いたお問い合わせは翌営業日の対応となります。また、内容によりご回答までにお時間を頂く場合がございます。なお、不具合が発生した際にご使用されていた関連プログラム一式のコピーを、検証のためご提供いただくようお願いする場合がございます。何卒ご理解ご協力のほど、よろしくお願い申し上げます。

✓ **お見積もり／保守契約関連のお問い合わせ**

営業部　TEL：（東京）03-4586-1808　E-mail：sales@jp.vector.com

VECTOR >

MORE INFORMATION
Visit our website for:

> News
> Products
> Demo software
> Support
> Training and workshops
> Contact addresses

vector.com