

GXtest

Automated testing for
better applications



GeneXus™

abstracta

概要



GXtest について

- テストツールとGeneXus IDEの統合
- ナレッジベース内で一元管理されるテストオブジェクト
- エンドツーエンドのテストケース実行、CI/CD環境の実現



まず、GXtestとはどのようなものか。

GXtestはGeneXusのテストツールであり、GeneXus IDEに同梱されています。そのため、テストオブジェクトもナレッジベース内で一元管理されます。

GXtestの利用による最も重要な利点は、エンドツーエンドのテストを実行して

継続的インテグレーションと継続的デリバリープロセス（CI / CD）を実現できることです。

テストツールの必要性

1. 開発者が自分の実装をテストできる。
2. CI/CD環境を実現できる。
3. **Shift-left** の実現。
出来るだけ早い段階でバグを検出し、早期に解決する。
開発サイクルを高速に回転させる。
4. DevOpsカルチャーへの架け橋になる。

開発プロセスでテストツールを使用する必要があるのはなぜですか？

- 1：開発者自身が出来上がった処理を一番理解しているため、自分自身の実装をテストすべきです。
- 2：すべてのテストをパイプラインで実行されるように設計し、CI/CD環境を実現できる。
- 3：Shift-left を行うことで、早期に仕様ミス、バグの検出を行うことが可能です。
- 4：テスト自動化はdevopsのハーネスであり、継続的デリバリーのコアであり、サイクルを短縮および高速化する

テストの種類について

Unit Test

ソースコードの単位(Unit)を実行し、適切に実装されているか判断するためのテスト



ユニットテストは、任意のコードをテストする処理を定義します。

テストの種類について

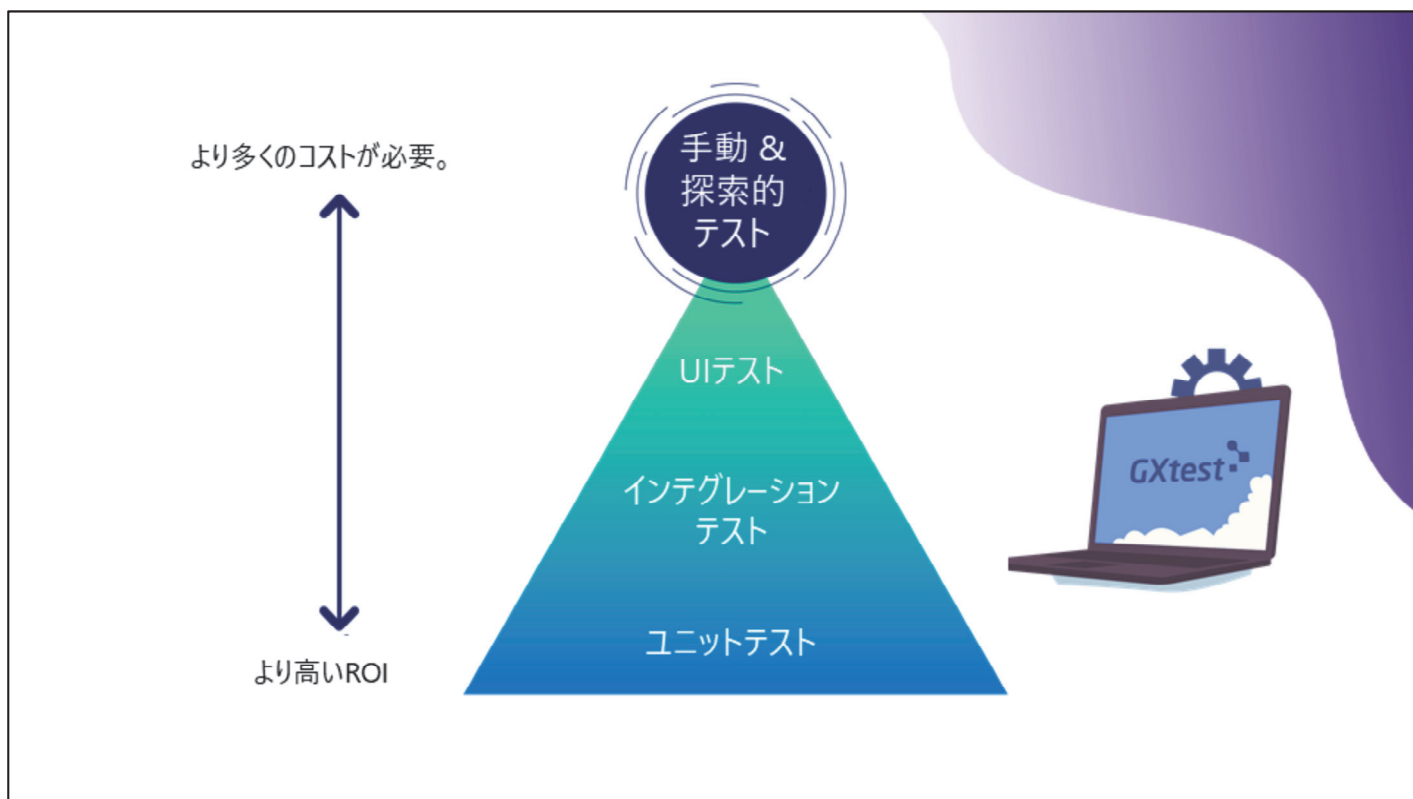


UI Test



アプリケーション上でのユーザーの操作をシュミレーションし、適切に動作するかテスト

UIテストは、実行したアプリケーション上でユーザーの操作をシュミレーションできます。
UIテストを自動化することでリグレッションテストや、コアビジネステストのために有用です。



テスト自動化ピラミッド

GXtestは、このテスト範囲全体をカバーすることを目的としています。
それらが何であるか見てみましょう、

開発者にできるだけ早くフィードバックを提供するために実行するクイックテストである単体テストを用意します。
さらに、ブラウザ上のアプリケーションでのユーザーインタラクションをシミュレートするインターフェイステストがあります。

したがって、テスト自動化スキームを指すと、まず、単体テストの強固な基盤が得られます。
さらに一連のインターフェイステストを実施し、自動化できないテストとして手動テストと探索的テストの実施を検討します。

コスト的な面で見てみましょう。
基本的にテストの自動化はユニットテストに対して、高いROIを示すことが知られています。
なぜならユニットテストは、極めて短調なテスト項目を高速に繰り返すため、システムティックに実行することが簡単であり、
それにより人為的なミスを排除することが可能です。自動化すれば、テストケースを1000回することも苦ではありません。

UIテストは、単体テストよりは効果が出にくいものとなりますが、それでも自動化のメリットは大きいものです。

利用方法






Unit Test

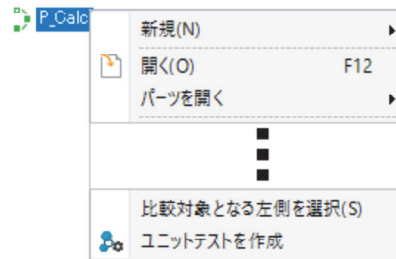


Unit Test

対象のオブジェクトタイプ

-  プロシージャ
-  データプロバイダー
-  ビジネスコンポーネント

作成方法

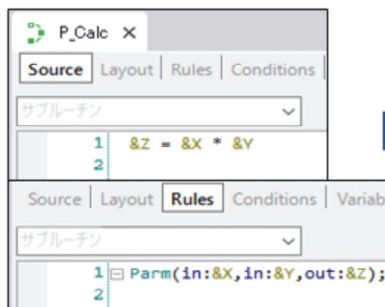


Unit Testは画面を持たない内部処理オブジェクトを対象に利用可能です。
アプリケーションを実行せずにこれらのオブジェクトの動作検証が可能です。
メンテナンスが簡単で、すぐに実行することができます。

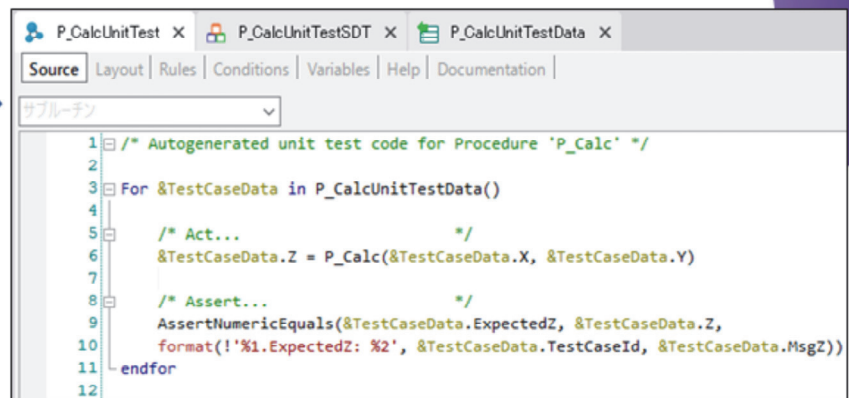
対象オブジェクトの右クリックメニューから簡単に既定のフォーマットに基づくテストオブジェクトを作成できます。

Unit Test

対象オブジェクト



自動生成されたテスト

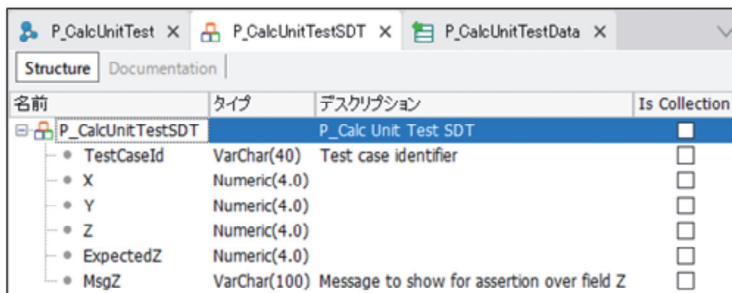


「ユニットテストを作成」を実行した場合、GXtestは対象のオブジェクトを繰り返しテストできるようにUnit Testオブジェクトの作成と、そのオブジェクトを実行するために必要なオブジェクトを作成します。

Unit Testオブジェクト内ではデータプロバイダーから出力されるテストデータを含むSDTコレクションに基づき繰り返しテストを実行します。想定される出力値と一致するかをAssertと呼ばれるGXtestが用意する検証用関数にて確認します。この確認の値もデータプロバイダーより取得します。

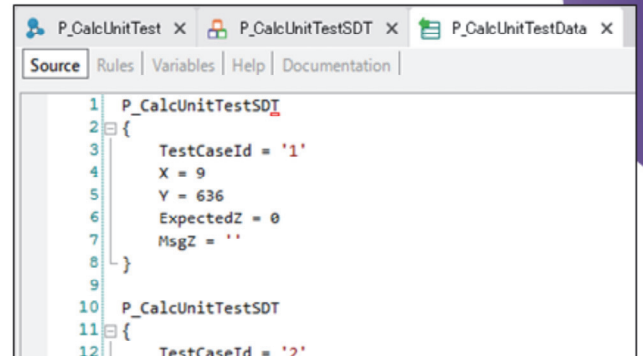
Unit Test

その他の自動生成オブジェクト



The screenshot shows the 'Structure' view of a development environment. It displays a tree view of the 'P_CalcUnitTestSDT' object. The tree has a root node 'P_CalcUnitTestSDT' which contains several child nodes: 'TestCaseId', 'X', 'Y', 'Z', 'ExpectedZ', and 'MsgZ'. Each node has a 'Type' and a 'Description' column. The 'Is Collection' column has checkboxes for each node.

名前	タイプ	DESCRIPTION	Is Collection
P_CalcUnitTestSDT		P_Calc Unit Test SDT	<input type="checkbox"/>
• TestCaseId	VarChar(40)	Test case identifier	<input type="checkbox"/>
• X	Numeric(4.0)		<input type="checkbox"/>
• Y	Numeric(4.0)		<input type="checkbox"/>
• Z	Numeric(4.0)		<input type="checkbox"/>
• ExpectedZ	Numeric(4.0)		<input type="checkbox"/>
• MsgZ	VarChar(100)	Message to show for assertion over field Z	<input type="checkbox"/>



The screenshot shows the 'Source' view of a development environment. It displays the source code for the 'P_CalcUnitTestSDT' object. The code is written in a language that looks like SQL or a similar declarative language. It defines the 'P_CalcUnitTestSDT' object and its properties: 'TestCaseId', 'X', 'Y', 'ExpectedZ', and 'MsgZ'. The code is as follows:

```
1 P_CalcUnitTestSDT
2 {
3     TestCaseId = '1'
4     X = 9
5     Y = 636
6     ExpectedZ = 0
7     MsgZ = ''
8 }
9
10 P_CalcUnitTestSDT
11 {
12     TestCaseId = '2'
```

前のページで少し触れましたが、同時に作成されるオブジェクトとしてSDTとデータプロバイダーがあります。

SDTオブジェクトには、テストID、対象のプロシーチャーで定義された引数、戻り値(Parmルールの定義)に基づく値、想定される戻り値、想定される戻り値に関する表示メッセージが格納できるアイテムが定義されています。
必要に応じてカスタマイズできます(例：データ桁数を変更する)

データプロバイダーオブジェクトにはテストデータのセットが既定で5件定義されています。
この自動生成される件数はKBの設定であらかじめ指定可能です。
また、この自動生成データでは、想定される戻り値および表示メッセージは空の値となるため、テストを実施する前に指定する必要があります。

Unit Test

テスト結果

The screenshot shows a window titled 'テスト結果' (Test Results). It displays the execution details for a unit test named 'P_CalcUnitTest'. The test was executed on 2022-08-09 at 15:51:44, took 168 ms, and passed successfully, indicated by a green checkmark. The test results are summarized in a table with columns for '期待値' (Expected Value), '取得値' (Actual Value), and '情報' (Information). The table shows five test cases, all of which passed. The first four test cases are highlighted with green checkmarks, and the fifth is highlighted with a red X. The '期待値' column contains the values 6, 40, 45, 3000, and 99980001. The '取得値' column contains the values 6, 40, 45, 3000, and -27935. The '情報' column contains the following descriptions: '1.ExpectedZ: 計算結果が異なります', '2.ExpectedZ: 10*4の結果を想定しています', '3.ExpectedZ: 5*9の結果を想定しています', '4.ExpectedZ: 50*60の結果を想定しています', and '5.ExpectedZ: 9999*9999の結果を想定しています'. A button labeled '期待値として設定' (Set as Expected Value) is located at the bottom right of the table.

期待値	取得値	情報
6	6	1.ExpectedZ: 計算結果が異なります
40	40	2.ExpectedZ: 10*4の結果を想定しています
45	45	3.ExpectedZ: 5*9の結果を想定しています
3000	3000	4.ExpectedZ: 50*60の結果を想定しています
99980001	-27935	5.ExpectedZ: 9999*9999の結果を想定しています

テストを実行した結果は専用のウィンドウで表示されます。
テストデータとして(データプロバイダーで)指定した期待値が取得値と一致する場合には成功として緑色のチェックアイコンが表示されます。

もし、期待値と取得値が異なる場合にはエラーとなり、赤いバツアイコンが表示されます。

例えば、桁あふれが発生するようなテストケースを用意した場合です。
(SDTの期待値アイテムのデータ桁数を変更しています)

Unit Test

モッキング

The screenshot displays two windows from a software development environment. The left window, titled 'Product', shows a table structure with three columns: '名前' (Name), 'タイプ' (Type), and 'DESCRIPTION' (Description). The table contains three rows: 'Product' (Type: Product, Description: 製品), 'ProductId' (Type: Numeric(4.0), Description: 製品番号), and 'ProductName' (Type: VarChar(40), Description: 製品名称). The right window, titled 'P_Discount', shows a script editor with a 'Source' tab selected. The script contains a 'For each Product' loop that calculates the discounted price based on the discount percentage. The 'Rules' tab is also visible, showing a 'Parm' function call.

名前	タイプ	DESCRIPTION
Product	Product	製品
ProductId	Numeric(4.0)	製品番号
ProductName	VarChar(40)	製品名称
ProductPrice	Numeric(4.0)	製品価格

```
1 For each Product
2   where ProductId = &InProductId
3   ProductPrice = ProductPrice * (1 - (&DiscountPercentage/100))
4   &OutProductPrice = ProductPrice
5 Endfor
```

```
1 Parm(in:&InProductId,in:&DiscountPercentage,out:&OutProductPrice);
2
```

ユニットテストの機能の1つとしてモッキングが用意されています。
これはデータベースの状態をテスト用に保存しておくための機能です。

なぜこのような機能が必要となるのでしょうか？

データベースの値は様々なテストを行うために日々変化する可能性があります。

この時、あるテストケースが特定のテーブルを参照している場合、突然テスト結果が失敗になる可能性があります。

これはテストデータが変わってしまったため、事前に定義されていたテストデータと一致しなくなったためです。

また、テスト内でデータの操作を行い、元に戻さないために次の実行が失敗することも想定されます。

このようなケースでモッキングを活用できます。

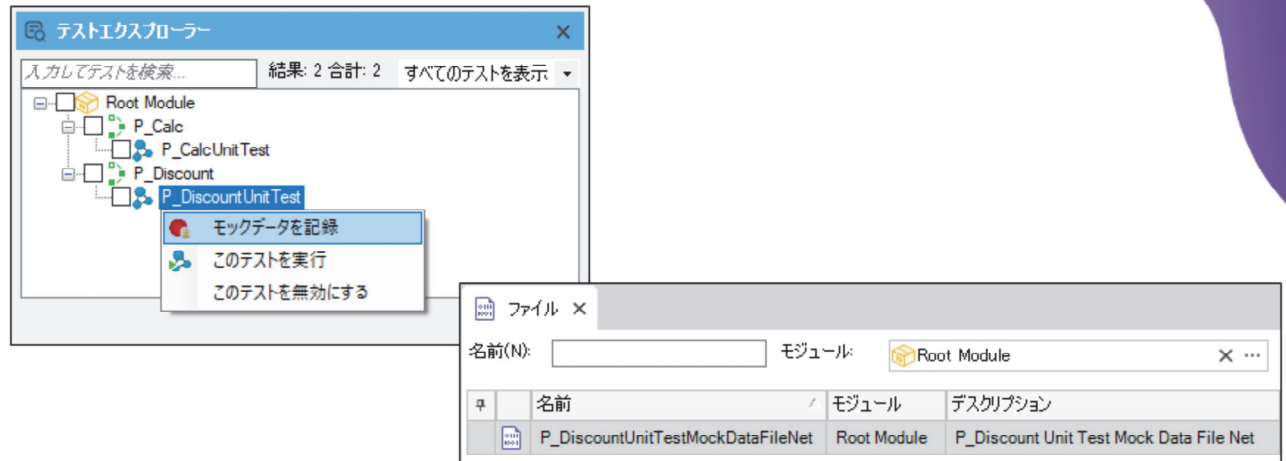
例えばこのスライドにある定義のようにあるトランザクションの価格項目を割引価格へ更新するプロシージャがあり、

このプロシージャをテストするユニットテストがある場合です。

登録されている価格が変わってしまうと、事前に定義したテストデータと一致せず、失敗することが考えられます。

Unit Test

モッキング



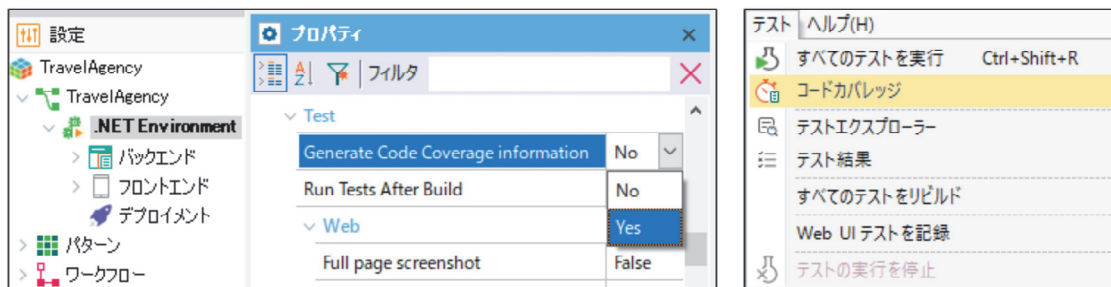
モッキングを行うためには、テストエクスプローラーにて「モックデータを記録」をクリックします。

この操作により、現在のデータベースの状態がこのユニットテスト用に記録され、ファイルが生成されます。

今後、このテストを実行する際、プロシージャァが読み込むデータはデータベースの値ではなく、このファイル内に記録された値となります。

Unit Test

カバレッジ

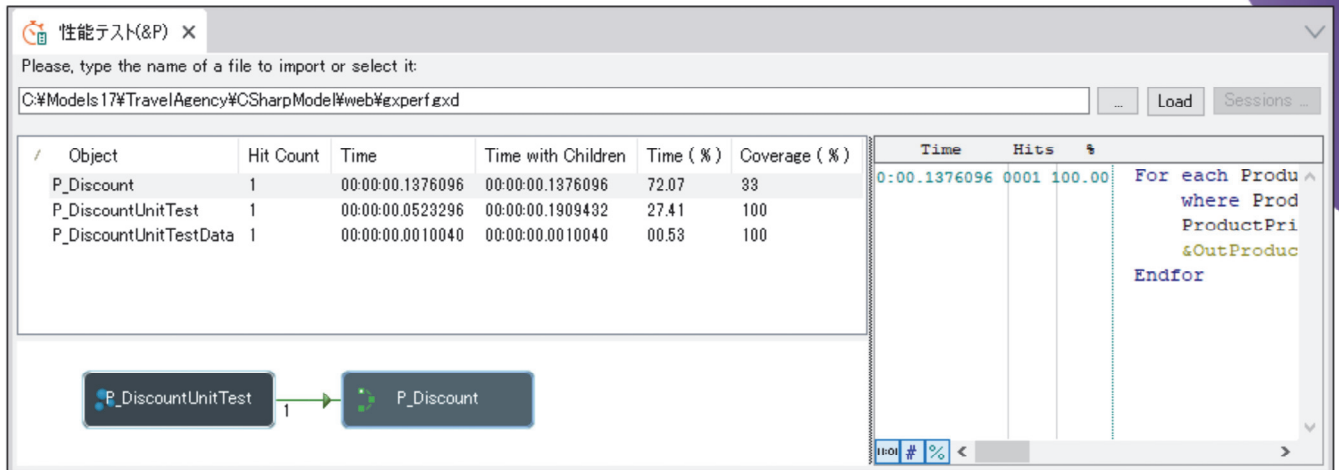


テストが対象のオブジェクトのコードのうち、何割のコードをテストできているかをカバレッジとして表示可能です。
例えば条件分岐が多数あるプロシーチャーをテストする場合に、すべての分岐で意図した結果となるのかを確認することが可能です。

この機能を利用するためには、まずはプロパティで機能を有効にする必要があります。
有効になった場合、ツールバーのメニューに「コードカバレッジ」というメニューが追加されます。

Unit Test

カバレッジ





実行したテストによって対象のオブジェクトのコードを何割検証することができたかを確認するための画面が用意されています。対象のオブジェクトのコードも表示されるため、どの行が実行されているのか、されていないのかを用意に確認できます。

UI Test

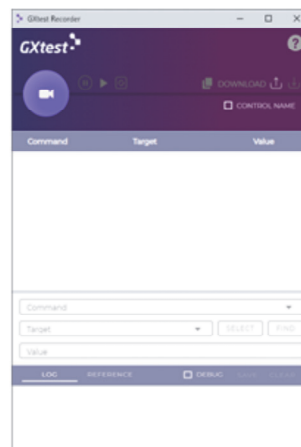


UI Test

対象アプリケーション

- Web
(Web UI Test) 
- ネイティブモバイル
(UI Test) 

作成方法



GXtestでは、Webアプリケーションとネイティブモバイルアプリケーション両方に対し、UIテストを実行することが可能です。

UIテストでは、機能をテストするためにユーザーとアプリケーションのやり取りを趣味レートし、UIおよびデータベース上のシステムの出力や結果を検証します。

そして、GeneXus上で定義を行うため、テスト対象のコントロールについては、GeneXus上のコントロール名で指定することも可能です。

作成する際には、2つの方法が用意されています。

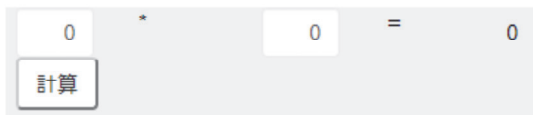
1つはすべてのテストコードを手書きする方法です。

もう1つは実行したアプリケーション上での操作の記録です。

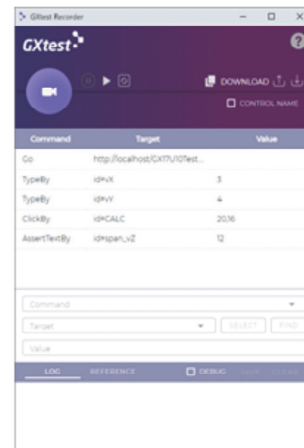
この場合、GXtestが提供しているGoogle Chromeのアドオン「GXtest Recorder」を利用できます。

UI Test (Web)

対象の画面例



作成方法



あるアプリケーションでは、2つの入力欄に値を入力すると、計算(掛け算)が行われる画面があります。

この画面の動作をテストするために、Web UI Testを作成します。

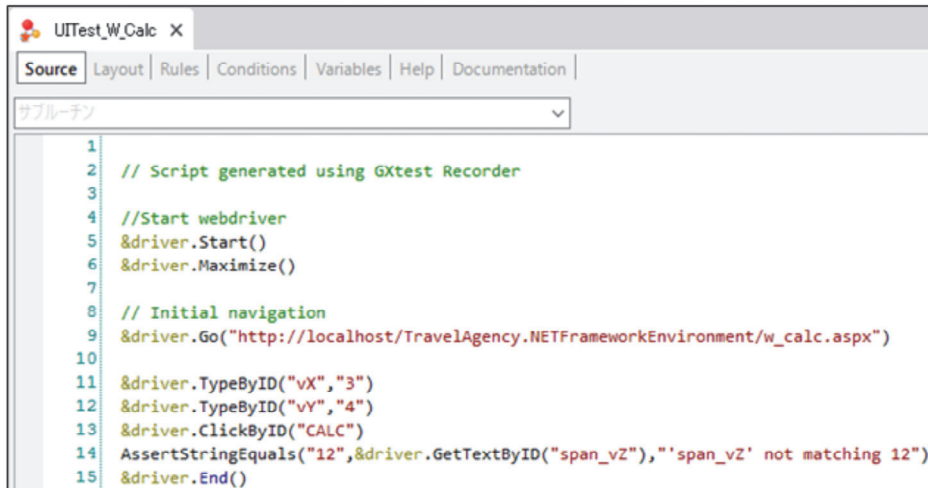
GXtest Recorderを起動し、操作の記録を開始すると、画面上での操作が記録されていきます。

X(画面左の入力欄)に値を入力し、Y(画面右の入力欄)に値を入力し、「計算」ボタンを押下。

計算結果として表示されるZ(画面右側の数値表示領域)の値を検証するテストケースを記録しました。

UI Test (Web)

生成されるオブジェクト



```
1 // Script generated using GXtest Recorder
2
3
4 //Start webdriver
5 &driver.Start()
6 &driver.Maximize()
7
8 // Initial navigation
9 &driver.Go("http://localhost/TravelAgency.NETFrameworkEnvironment/w_calc.aspx")
10
11 &driver.TypeByID("vX","3")
12 &driver.TypeByID("vY","4")
13 &driver.ClickByID("CALC")
14 AssertStringEquals("12",&driver.GetTextByID("span_vZ"),"'span_vZ' not matching 12")
15 &driver.End()
```

記録したテストケースをWeb UI Testオブジェクトとして生成した場合、オブジェクトはこのような定義を含むものとなります。

先ほど記録した通りの操作がコマンドとして生成され、値の代入や、ボタンの押下、値の検証がテストとなります。

右クリックすることでテストを実行することができ、ブラウザ上でテストが実行されることを確認できます。

ブラウザは既定でChromeが起動するようになっていますが、これはKB単位および各UIテスト単位で指定可能です。

UI Test (Web)

テスト結果

The screenshot displays the 'Web UI Test Results' window. The top section shows the test name 'UITest_W_Calc' and its status as 'Success'. Below this, a table lists the steps of the test, including 'Maximize()', 'Go(http://localhost/...)', 'TypeByID(vX3)', 'TypeByID(vY4)', 'ClickByID(CALC)', 'GetTextByID(span_vZ): '12'', and 'AssertStringEquals(12, 12, ... 'span_vZ' not matching 12)'. The table also includes columns for 'Info' and 'Duration (ms)'. The bottom right corner has a button labeled '期待値として設定'.

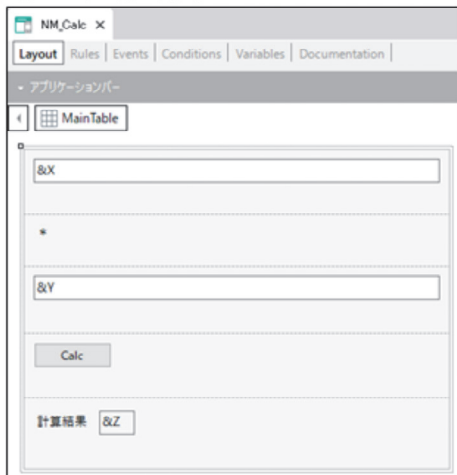
手順	情報	経過 (ms)
Maximize()		133
Go(http://localhost/...)		3231
TypeByID(vX3)		1167
TypeByID(vY4)		1154
ClickByID(CALC)		603
GetTextByID(span_vZ): '12'		24
AssertStringEquals(12, 12, ... 'span_vZ' not matching 12)		

Unit testと同様にテストとして定義された各コマンドの成功/失敗が結果として表示されます。

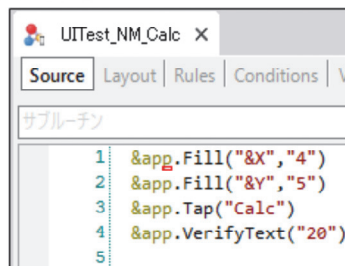
GeneXusのUIテストでは、入力欄の位置などでコマンドを定義せず、コントロール名で定義しているため、画面のデザインを変更し、項目の位置が変更となった場合でもこのままテストを再度実行することができます。

UI Test（ネイティブモバイル）

対象の画面例



作成例



現在のGXtestでは、スマートデバイス上で実行するネイティブモバイルアプリケーションのUIテストにも対応しています。
ネイティブモバイルアプリケーションとして生成されるメインオブジェクトを紐づけ、画面の操作をコマンドで指示します。
Web同様に画面上のコントロールの操作や値の検証を定義することが可能です。

テストケースを作成する場合には、必要なコマンドをGeneXus上で手動で記述する必要があります。
テスト対象となるアプリケーションは、UI Testオブジェクトのプロパティで指定しているため、
Web UI Testの場合のようにアプリケーションのURL指定などはありません。

また、作成についてWeb UI Testのような操作の記録機能は実装されていません。

UI Test（ネイティブモバイル）

実行結果

The screenshot shows the 'テスト結果' (Test Results) window. It displays the execution details for a test named 'UITest_NM_Calc'. The test was executed on 2022-08-30 at 11:57:53 and took 1 second to complete. The test passed, as indicated by the green checkmark. The execution details table shows the following steps:

手順	情報	経過 (ms)
fill(X, 4)		811
fill(Y, 5)		71
tap(Calc)		408
verifyText(20, true)		33

At the bottom right of the window, there is a button labeled '期待値として設定' (Set as Expected Value).

テストを実行した際の結果については、これまでのタイプと同様にテスト結果としてGeneXusのIDE上に表示されます。

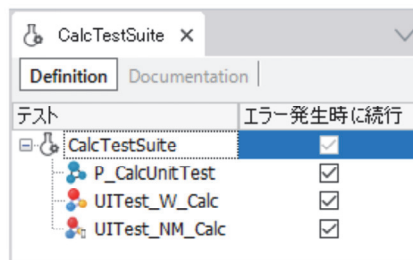
すべてのテスト結果はHTMLとして保存することができ、テストのエビデンスとしてご利用いただけます。

その他機能



Test Suites

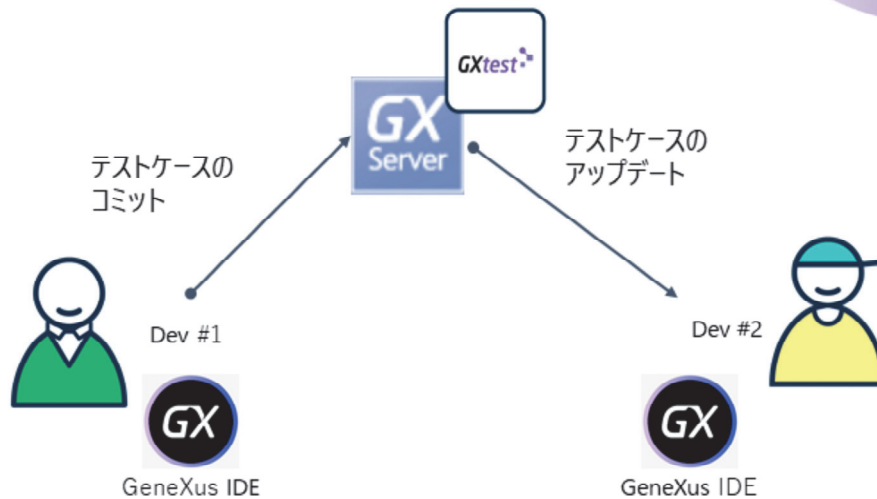
Test Suitesとは？



ナレッジベース内に作成されたすべてのテストケースは「テストエクスプローラー」ウィンドウから複数選択し、一度に実行の指示を行えます。しかし、この時の実行される順序は選択されたテストオブジェクトの名前をアルファベット順で並び替えたものとなります。

そのため、いくつかのテストケースで共通するテスト操作を1つのテストケースとして作成しても、適切に利用できません。このようなケースで活躍するのがこの「Test Suites」機能(オブジェクト)です。任意のテストケースを任意の順番で実行するように指定することができます。

GXServerを用いたテストケース共有

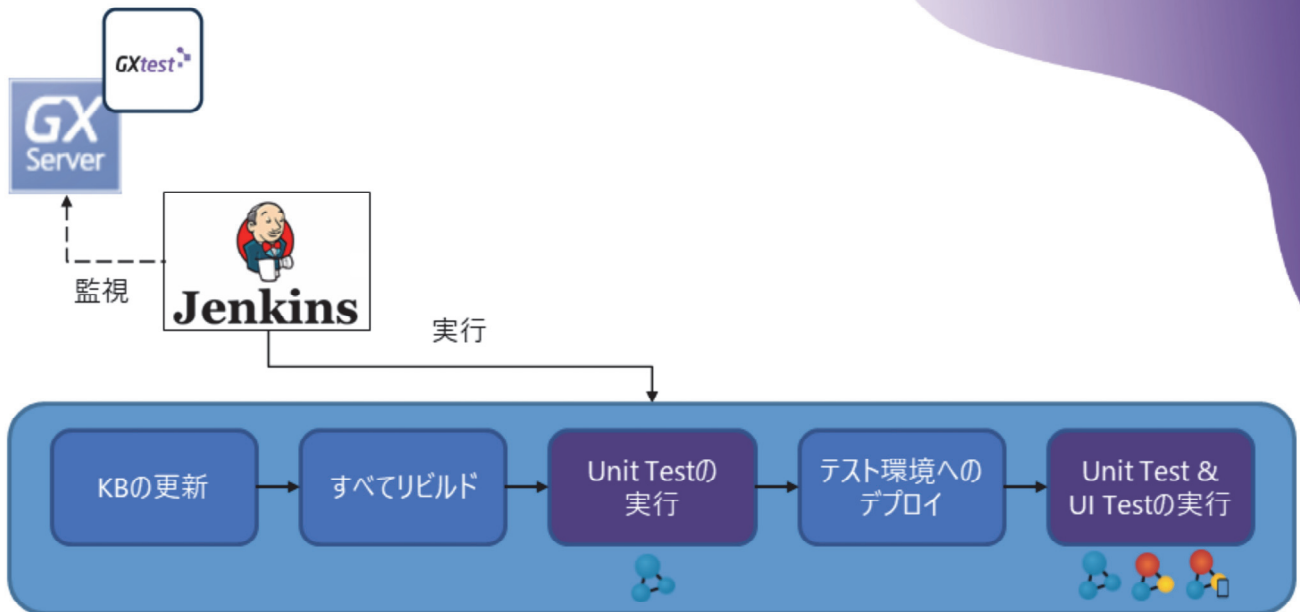


GXtestで作成するテストオブジェクトはナレッジベース内の1オブジェクトであることはお伝えしていました。

これは、GeneXus Serverを利用した場合、テストケースをチーム内で共有できることを意味します。

誰かが作成したテストケースはチーム内で共有することができ、この機能を利用し、最終的にリリース前の様々なテストにメンバーが作成したテストを利用することができます。

継続的インテグレーション（CI）



GeneXus Serverは、CI/CDを実現するために「パイプライン」という機能が搭載されています。

この機能を利用することで、継続的インテグレーション、継続的デリバリーつまり一般的に言われるDevOpsを実現できます。

GXtestによって作成されたテストケースもこの継続的インテグレーションに含めることが可能です。

図にあるようにパイプラインの中でテストケースを実行するプロセスを構成し、自動的にテストも完了させることが可能です。

利用を開始するには？

- ✓ テスト手法と役割分担の明確化
(開発者とテスターのGXServerへのコミット)
- ✓ アプリケーションの解析とテストの作成
(初期 : ユニットテストとスモークUIテスト
更新時 : 回帰UIテスト)
- ✓ CIパイプラインにテストプロセスを追加
(トランク版/安定板)



GXtestのご利用を開始する際に必ず検討が必要となる点を挙げさせていただきました。

ライセンスモデル



GXtestは、一部の機能は無償で利用可能ですが、すべての機能を利用するには、ライセンスが必要となります。
機能ごとのライセンス要否についてご説明させていただきます。

ライセンスモデル

	機能	ライセンス
Unit Test	Unit Testの作成/実行	不要
	カバレッジ	不要
	モッキング	必要
UI Test	UI Test (Web) の作成/実行	不要
	GXtest Recorderの利用	不要
	UI Test (ネイティブモバイル) の作成/実行	不要
その他	Test Suites	不要
	テストオブジェクトのインポート/エクスポート	必要
	GXServerへのコミット/ GXServerからの更新	必要
	コマンドラインからのテスト実行	必要
	Jenkinsとの連携	必要

比較



他のテスト自動化製品との比較について。
比較資料より抜粋

他製品との比較

機能	製品	Katalon	Selenium	GXtest
テスト可能なアプリケーション		Windows デスクトップアプリ Webアプリ Mobileアプリ API	Webアプリ	Webアプリ Mobileアプリ API
テスト機能		UIテスト APIテスト	UIテスト	Unitテスト UIテスト APIテスト
テストを実装する言語		Java/Groovy	Java, C#, Perl, Python, JavaScript, Ruby, PHP	GeneXus
習熟の容易さ		中	高	低 (GeneXus開発者)
テスト分析・解析機能		Katalon TestOps	なし	テストカバレッジ
GeneXusの待機時間		手動	手動	自動

表に記載の内容については、GXtest開発元にて検証し、記録したドキュメントのローカライズ版より抜粋したものととなります。
すべての比較内容については資料をご確認ください。

Thank you!

GXtest 

abstracta 