OPTIMIZE

MERCURY WINRUNNER™

VERSION 9.2

チ ュ ー ト リ ア ル

MERCURY™

BUSINESS TECHNOLOGY OPTIMIZATION

Mercury WinRunner

チュートリアル

Version 8.2

MERCURY™

Mercury WinRunner チュートリアル，Version 8.2

Mercury Interactive Corporation
379 North Whisman Road
Mountain View, CA 94043
Tel: (650) 603-5200
Toll Free: (800) TEST-911
Customer Support: (877) TEST-HLP
Fax: (650) 603-5300

本書に関するご意見，ご要望は documentation@mercury.com まで電子メールにてお送りください。

WRTUT8.2JP/01

目次

WinRunner チュートリアルにようこそ

WinRunner チュートリアルへようこそ。本書では，WinRunner を使ったアプリケーション・テストの基本を自分のペースで学ぶことができます。このチュートリアルでは，自動テストの作成，実行，テスト結果の分析という工程を学んでいきます。

本チュートリアルの使用

このチュートリアルは 11 の短い練習で構成されています。各練習では，WinRunner プログラム・グループに含まれている，サンプルのフライト予約アプリケーション（Flight 1A および Flight 1B）を対象とするテストを作成して，実行します。

サンプルのフライト予約アプリケーションには，Flight 4A と Flight 4B という 2 つのバージョンがあります。Flight 4A は正しく動作する製品ですが，Flight 4B にはいくつかの「不具合」が組み込まれています。『WinRunner チュートリアル』では，これらのバージョンを組み合わせて使用し，開発工程をシミュレートして，アプリケーションの一方のバージョンのパフォーマンスともう一方のパフォーマンスを比較します。

チュートリアルを完了すれば，ここで学んだ技法を自分のアプリケーションに適用できます。

練習 1「WinRunner について」では，自動テストと手動テストの方法を比較します。ここで WinRunner のテスト工程を紹介しますので，WinRunner のユーザ・インタフェースに慣れてください。

練習 2「GUI マップの設定」では，WinRunner がアプリケーションの GUI（グラフィカル・ユーザ・インタフェース）オブジェクトを識別する方法と，GUI マップ・ファイルを編成するための 2 つのモードを説明します。

練習 3「テストの記録」では，テスト・スクリプトを記録する方法を示し，テスト・スクリプト言語（TSL）の基本を説明します。TSL は，スクリプト作成用に設計された Mercury のプログラミング言語で，C 言語に似ています。

練習 4「テストの同期化」では，テストの同期を取る方法について説明します。テストの同期を取ることで，入力に対するアプリケーションの応答が遅い場合でも，テストをうまく実行できます。

練習 5「GUI オブジェクトの検査」では，GUI オブジェクトを検査するテストの作成方法を示します。テストを使って，バージョンが違うサンプル・アプリケーションの GUI オブジェクトの振る舞いを比較できます。

練習 6「ビットマップの検査」では，アプリケーション内のビットマップを検査するテストの作成と実行の方法を示します。バージョンが違うサンプル・アプリケーションでテストを実行して，すべての違いをピクセルごとに検証します。

練習 7「TSL を使ったテストのプログラミング」では，ビジュアル・プログラミングを使って，記録済みのテスト・スクリプトに関数やロジックを追加する方法を説明します。

練習 8「データ駆動型テストの作成」では，データ・テーブルの複数のデータ・セットに対して同じテストを実行する方法を説明します。

練習 9「バッチ・テストの作成」では，これまでの練習で作成したテストを自動的に実行するバッチ・テストの作成方法を示します。

練習 10「テスト・スクリプトの保守」では，WinRunner が学習した GUI オブジェクトの記述を更新する方法を説明します。これにより，アプリケーションに変更を加えた場合でも，テスト・スクリプトを使い続けることができます。

練習 11「これ以降の進め方」では，自分のアプリケーションのテストを始める方法と，WinRunner に関するさらに詳しい情報の見つけ方を学びます。

表記規則

本書では次の表記規則に従います。

1，2，3	太字の数字は，操作手順を示します。
＞	大なり記号はメニュー・レベルを区切ります（例：［**ファイル**］＞［**開く**］）。
［**太字**］	全角の大括弧に太字は，インターフェイスの要素の名前（例：［**実行**］ボタン）やその他の強調する項目を示します。
太字	**太字**のテキストは，メソッド名または関数名を示します。また，メソッドまたは関数の引数，構文の記述中のファイル名，書名を示します。新しい用語を紹介する場合にも使用します。
＜＞	山括弧は，ユーザの環境次第で変わる可能性のある，ファイル・パスや URL アドレスの一部分を囲みます（例：**＜製品のインストール・フォルダ＞ ¥bin**）。
Arial	Arial のフォントはそのまま入力する例やテキストに使用されます。
Arial bold	**Arial bold** のフォントはそのまま入力しなければならない構文記述のテキストに使用されます。
SMALL CAPS	SMALL CAPS のフォントは，キーボードのキーを示します。
...	構文内の 3 つの点は，同じ形式で項目をさらに含めることができることを意味します。プログラム例での 3 つの点は，プログラム行が意図的に削除されていることを示します。
[ ]	半角の大括弧は，省略可能な引数を囲みます。
|	2 つの値のうちの 1 つを選択しなければならない場合，これらの値を垂直バーで区切ります。

練習 1

WinRunner について

この練習では次のことを学びます。

- 自動テストの利点について
- WinRunner のテスト工程について
- WinRunner のユーザ・インタフェースの概要

自動テストの利点について

ソフトウェアを手動でテストしたことがあれば，その欠点にお気付きでしょう。手動テストは時間のかかる退屈な作業である上，非常に多くの人的資源を必要とします。何よりも問題なのは，ソフトウェアがリリースされる直前になって各機能を手動で十分にテストすることが，時間的に不可能な場合がよくあるということです。このことが，重大なバグが検出されないままになっているのでないかという不安を抱かせるのです。

WinRunner を使って自動テストを行えば，これらの問題は解決され，テスト工程にかかる時間を劇的に短縮できます。アプリケーションのあらゆる側面を検査するテスト・スクリプトを作成して，アプリケーションの新しいビルドを対象に実行できます。WinRunner でテストを実行すると，WinRunner はアプリケーション上でマウス・カーソルを動かしたり，GUI オブジェクトをクリックしたり，キーボード入力をするなど，人間の振る舞いをシミュレートします。ただし，WinRunner はこれを人間ユーザよりも高速に実行します。

WinRunner を使えば，夜間にバッチ・テストを実行して時間を節約できます。

自動テストの利点	
速い	WinRunner は人間ユーザよりもはるかに高速でテストを実行します。
信頼できる	テストではまったく同じ操作を正確に繰り返し実行できるので，人為的な間違いをなくせます。
反復可能	同じ操作を繰り返し実行したときにソフトウェアがどのように反応するかをテストできます。
プログラム可能	アプリケーションから隠れた情報を抽出する，洗練されたテストをプログラムできます。
包括的	アプリケーションの全機能を網羅する一連のテストを構築できます。
再利用可能	ユーザ・インタフェースが変わっても，アプリケーションの異なるバージョンで同じテストを実行できます。

テスト工程について

WinRunner のテスト工程は，以下の 6 つの主要な段階から成り立っています。

1 **アプリケーションのオブジェクトを WinRunner に学習させる**

テストを実行するために，アプリケーションのオブジェクトを認識できるように，WinRunner に学習させる必要があります。選択する GUI マップ・モードによって，推奨されるオブジェクト学習法が異なります。以降の練習に，2 つの GUI マップ・モードの詳しい説明があります。

2 **アプリケーションの機能性をテストする付加的なテスト・スクリプトの作成**

アプリケーションでのアクションを記録すると，WinRunner が自動的にスクリプトを記述します。また，Mercury のテスト・スクリプト言語（TSL）を使って，直接プログラミングすることもできます。

3 **テストのデバッグ**

テストをデバッグして，テストが中断されることなくスムーズに動作することを確認します。

4 新しいバージョンのアプリケーションでのテストの実行

新しいバージョンのアプリケーションでテストを実行して，アプリケーションの振る舞いを検証します。

5 テスト結果の検証

テスト結果を検証し，アプリケーションの不具合を正確に特定します。

6 不具合の報告

WinRunner コンピュータに Quality Center がインストールされている場合は，不具合をデータベースに報告できます。Quality Center は，Mercury ソフトウェア・テスト管理ツールです。

［WinRunner］ウィンドウ

テストの作成を開始する前に，WinRunner のメイン・ウィンドウに慣れておきましょう。

注：WinRunner を開くには，画面表示設定を最低でも 256 色に設定する必要があります。

WinRunner を開始するには，次の手順を実行します。

［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**WinRunner**］を選択します。［WinRunner アドイン マネージャ］ダイアログ・ボックスが開きます。

注：初めて WinRunner を起動した時には，「WinRunner の新機能」ヘルプも表示されます。

［WinRunner アドイン マネージャ］ダイアログ・ボックスには，お使いのコンピュータで使用可能なアドインのリストが含まれます。このチュートリアルでは，すべてのアドイン・チェックボックスがクリアされていることを確認し，［**OK**］をクリックします。

［WinRunner へようこそ］ウィンドウが開きます。［WinRunner へようこそ］ウィンドウから，新規テストを作成するには［**テストの新規作成**］をクリック，既存のテストを開くには［**既存のテストを開く**］をクリック，普段お使いのブラウザで WinRunner の概要を参照するには［**WinRunner のクイック プレビューを表示**］をクリックします。

ヒント：これらのオプションの 1 つを初めて選択するときに，WinRunner のメイン画面が開き，次回 WinRunner を起動する際に［WinRunner へようこそ］ウィンドウを表示したくない場合は，［**起動時に表示**］チェック・ボックスをクリアします。

このチュートリアルでは，［**テストの新規作成**］をクリックして［WinRunner へようこそ］ウィンドウを閉じ，WinRunner での作業を開始します。

作成または実行する各テストは，WinRunner によってテスト・ウィンドウ上に表示されます。一度に複数のテストを開くことができます。

1 WinRunner ウィンドウには，開いているすべてのテストが表示されます。

2 各テストはそれぞれのテスト・ウィンドウ内に表示されます。このウィンドウで，テスト・スクリプトの記録，プログラミング，編集を行います。

3 標準ツールバーのボタンを使って，テストを素早く開始，実行，保存できます。

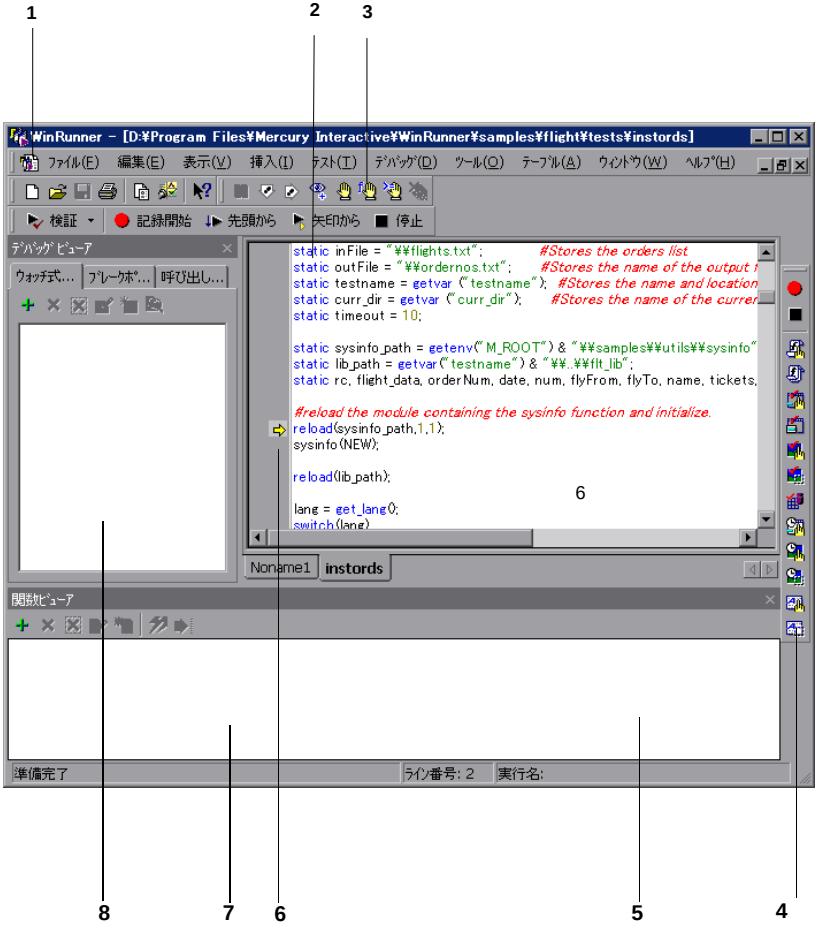
4 ユーザ定義ツールバーを使って，テスト作成ツールに簡単にアクセスできます。

5 ステータス・バーには，選択されたコマンドと現在のテスト実行に関する情報が表示されます。

6 実行矢印は，実行される次のスクリプト行を示します。

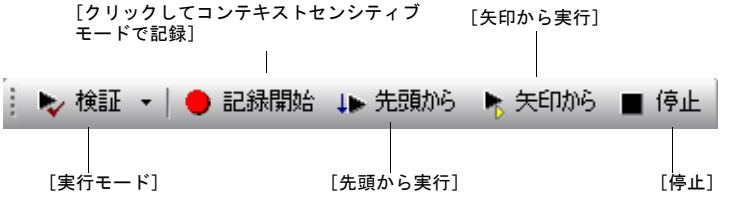
7 関数ビューアを使用して，テストにロードされている関数を参照，編集，使用できます。

8 ［デバッグ ビューア］を使用して，テストのウォッチ・リスト，ブレークポイント，呼び出しチェーンを参照できます。

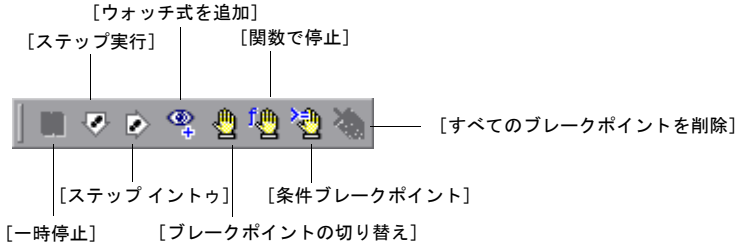


ファイル・ツールバーを使って，テストの開始，保存，テスト結果の表示など，よく使う機能に簡単にアクセスできます。

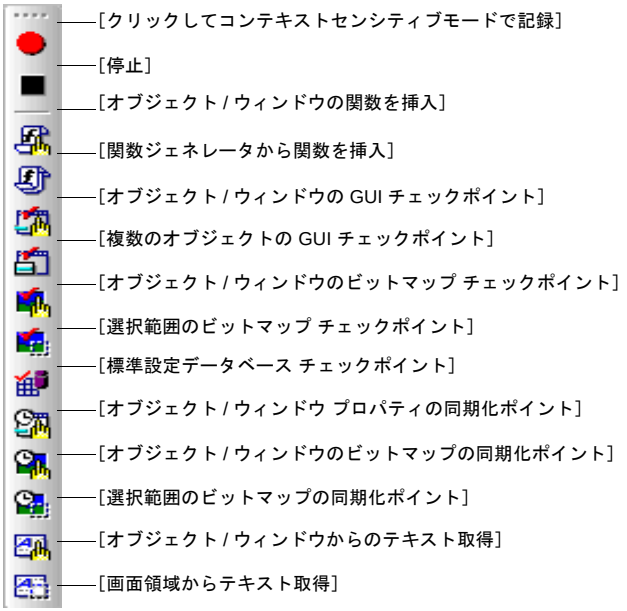
テスト・ツールバーを使用して，テストの実行および保守中に使用するボタンに簡単にアクセスできます。



デバッグ・ツールバーを使用して，テストのデバッグ中に使用するボタンに簡単にアクセスできます。



ユーザ定義ツールバーには，テスト・スクリプトを作成するときによく使うツールが表示されます。



テストを作成するときに，WinRunner ウィンドウを最小化して，ツールバーだけを使って作業ができます。ファイル・ツールバー，テスト・ツールバー，ユーザ定義ツールバーの各コマンドについては，以降の練習で詳しく説明します。

標準では，ユーザ定義ツールバーは非表示です。ユーザ・ツールバーを表示するには，［**表示**］＞［**ユーザ定義ツールバー**］を選択します。ユーザ定義ツールバーはカスタマイズできます。ボタンを追加または削除するには，［**表示**］＞［**ユーザ定義ツールバーのカスタマイズ**］メニュー・オプションを選択します。WinRunner を再び開いたときには，前回閉じたときと同じユーザ定義ツールバーが表示されます。

多くのコマンドは，「**ソフトキー**」を使って実行することもできます。ソフトキーはメニュー・コマンドを実行するためのショートカット・キーです。WinRunner プログラム・グループにある［Softkey Configuration］ユーティリティを使って，キーボードへのソフトキーの割り当てを構成できます。詳細については，『**WinRunner ユーザーズ・ガイド**』の「WinRunner の概要」の章を参照してください。

WinRunner のメイン・ウィンドウに慣れたところで，次の練習に進む前に，しばらくこれらのウィンドウのコンポーネントを操作してみましょう。

練習 2

GUI マップの設定

この練習では次のことを学びます。

- WinRunner がアプリケーション内の GUI オブジェクトを識別する方法
- GUI スパイを使って，オブジェクトのプロパティを表示する方法
- 2 つの GUI マップ・モード
- テスト・スクリプト・ウィザードを使って，GUI オブジェクトの記述を学習してテストを生成する方法

WinRunner の GUI オブジェクトの識別方法

GUI アプリケーションは，ウィンドウ，ボタン，リスト，メニューなどの GUI オブジェクトで構成されています。

WinRunner は，GUI オブジェクトの記述を学習するとき，オブジェクトの物理的なプロパティを参照します。各 GUI オブジェクトには，「**Class**」，「**Label**」，「**Width**」，「**Height**」，「**Handle**」，「**Enabled**」を始めとする多くの物理的なプロパティがあります。ただし，WinRunner が学習するのは，アプリケーション内の個々のオブジェクトを一意に区別する際に必要なプロパティだけです。プロパティの詳細については，『**WinRunner ユーザーズ・ガイド**』の「GUI マップの構成設定」の章を参照してください。

例えば，［**OK**］ボタンを例にとると，WinRunner はこのボタンが［開く］ウィンドウにあり，プッシュボタン・オブジェクト・クラスに属しており，「**OK**」というテキスト・ラベルを持っているボタンであると認識します。

GUI オブジェクトの偵察

WinRunner の GUI オブジェクトの識別方法を理解するために，サンプルのフライト予約アプリケーションでオブジェクトを検証してみましょう。

1 フライト予約アプリケーションを起動します。

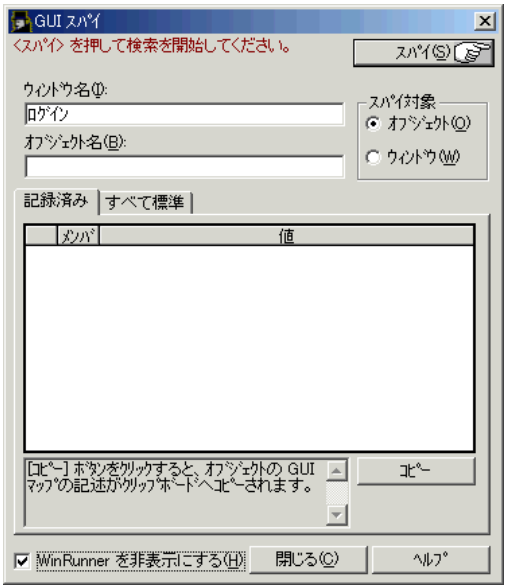
［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**Sample Applications**］＞［**Flight 4A**］を選択します。［ログイン］ウィンドウが開きます。

2 WinRunner を起動します。

［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**WinRunner**］を選択します。［WinRunner へようこそ］ウィンドウで，［**テストの新規作成** ］をクリックします。［WinRunner へようこそ］ウィンドウが表示されなければ，［**ファイル**］＞［**新規作成**］を選択します。

3［GUI スパイ］を開きます。このツールを使うと GUI オブジェクトのプロパティを「スパイ」できます。

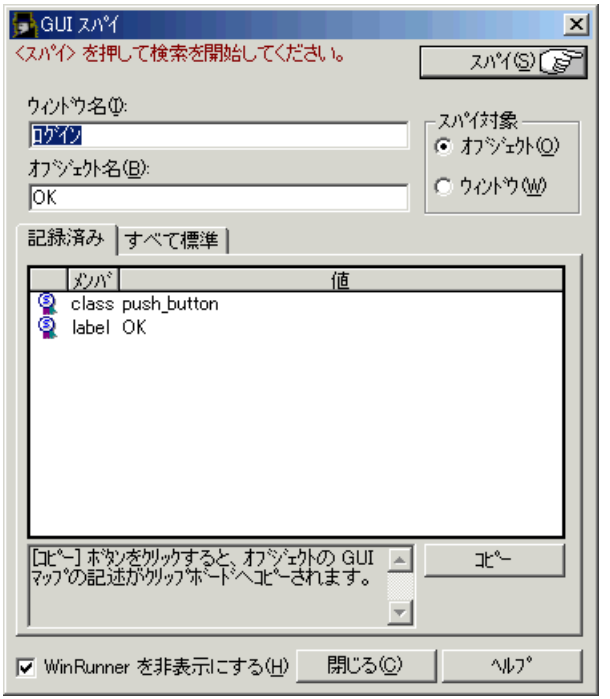
［**ツール**］＞［**GUI スパイ**］を選択します。［GUI スパイ］が開きます。［**WinRunner を非表示にする**］を選択してください。



4［OK］ボタンの一意の記述を提供するプロパティを表示します。

［GUI スパイ］で［**スパイ**］ボタンをクリックします。WinRunner ウィンドウが最小化されているため，［ログイン］ウィンドウが表示されているはずです。［ログイン］ウィンドウ内のオブジェクト上にポインタを移動します。ポインタで指したオブジェクトが点滅し，［GUI スパイ］にそのオブジェクトのプロパ

ティが表示されます。ポインタを［**OK**］ボタンの上に置き，**左 Ctrl** + **F3** を押します。これにより，［OK］ボタンの記述が［GUI スパイ］でフリーズします。



5 **［OK］ボタンのプロパティを検証します。**

［GUI スパイ］のダイアログ・ボックスの一番上には，オブジェクトが所属するウィンドウの名前とオブジェクトの論理名が表示されます。

［記録済み］タブには，記録されるプロパティの名前と値が表示されます。例えば，「**label** OK」は，ボタンが「**OK**」というテキスト・ラベルを持っていることを示し，「**class** push_button」はボタンがプッシュボタン・オブジェクト・クラスに属していることを示します。

このように，WinRunner はオブジェクトを一意に識別するのに，いくつかのプロパティしか必要としません。

6 **しばらく［ログイン］ウィンドウ内のほかの GUI オブジェクトのプロパティも調べてみましょう。**

［**スパイ**］ボタンをクリックし，［ログイン］ウィンドウのほかの GUI オブジェクトの上にポインタを移動します。

個々のオブジェクトのプロパティの拡張リストを見たい場合は，**左 Ctrl +F3** キーを押して現在の［スパイ］を停止し，［**すべて標準**］タブをクリックします。

7 **［GUI スパイ］を終了します。**

左 Ctrl + **F3** を押して，現在のスパイを停止し，［**閉じる**］をクリックします。

GUI マップ・モードの選択

アプリケーションの GUI を WinRunner に学習させる前に，［テスト特有の GUI マップ ファイル］モード，［グローバルな GUI マップ ファイル］モードのどちらで GUI マップ・ファイルを編成するかを検討する必要があります。

［テスト特有の GUI マップ ファイル］モード

［テスト特有の GUI マップ ファイル］モードでは，新しいテストを作成するたびに，GUI マップ・ファイルが自動的に作成されます。テストに対応する GUI マップ・ファイルが，テストを保存するたびに自動的に保存され，テストを開くたびに自動的にロードされます。

WinRunner の使い方やテストにまだ慣れていない場合は，［テスト特有の GUI マップ ファイル］モードで操作することをお勧めします。これは，テストに慣れていないユーザ向けの非常に簡単なモードであり，更新された GUI マップ・ファイルを確実に保存し，ロードすることができます。

［グローバルな GUI マップ ファイル］モード

［グローバルな GUI マップ ファイル］モードでは，複数のテストで構成されるテスト・グループに対して 1 つの GUI マップを使用できます。［グローバルな GUI マップ ファイル］モードで作業する場合，WinRunner がプロパティについて学習する情報を GUI マップ・ファイルに保存する必要があります。テストを実行するときに，適切な GUI マップ・ファイルをロードする必要があります。

WinRunner またはテストに慣れている場合は，［グローバルな GUI マップ ファイル］モードで操作するほうが，より効率的でしょう。

任意の GUI マップ・ファイル・モードの設定

標準では，WinRunner は［グローバルな GUI マップ ファイル］モードに設定されています。モードを［テスト特有の GUI マップ ファイル］モードに変更するには，［**ツール**］＞［**一般オプション**］を選択し，［**一般設定**］カテゴリで

［**テスト特有の GUI マップ ファイル**］を選択します。［**OK**］をクリックし，ダイアログ・ボックスを閉じます。

注：GUI マップ・ファイル・モードを変更した場合，変更を有効にするために WinRunner を再起動する必要があります。

テストの開始方法

以降の節は，［グローバルな GUI マップ ファイル］モードにのみ適用されます。［テスト特有の GUI マップ ファイル］モードで作業する場合は，前述の方法でモードの設定を変更し，練習 3 の 21 ページ「テストの記録」に進んでください。

［グローバルな GUI マップ ファイル］モードで操作する場合は，次の「テスト・スクリプト・ウィザードの使用」に進んでください。

重要なヒント：

1 つの GUI マップ・ファイルを選んだら，モードを変更せず，ほかの練習でも続けて使用することをお勧めします。両方のモードで練習したい場合は，それぞれのモードでチュートリアルの練習全部を通して繰り返してください。

［グローバルな GUI マップ ファイル］モードで作業する場合は，このチュートリアルのすべての練習を完了する前に WinRunner を終了して，WinRunner が閉じる前に GUI マップ・ファイルが保存されていることを確認してください。詳細については 21 ページを参照してください。WinRunner を再度開いたら，練習を続ける前に GUI マップ・ファイルがロードされていることを確認してください。詳細については 21 ページを参照してください。

テスト・スクリプト・ウィザードの使用

［グローバルな GUI マップ ファイル］モードを選択した場合，通常はテスト・スクリプト・ウィザードを使うことによって，テスト工程を最も簡単に素早く開始できます。

注：［テスト特有の GUI マップ ファイル］モードで作業しているときは，テスト・スクリプト・ウィザードは使用できません。

テスト・スクリプト・ウィザードは，体系的にアプリケーション内のウィンドウを開き，すべての GUI オブジェクトの記述を学習します。ウィザードはこの情報を GUI マップ・ファイルに格納します。WinRunner の学習工程を観察するために，フライト予約アプリケーションを対象にテスト・スクリプト・ウィザードを使ってみましょう。

注：端末エミュレータ，WebTest，または Java アドインがロードされていると，テスト・スクリプト・ウィザードは使用できません。したがって，これらのアドインを使用している場合は，この練習の以降の節はスキップするか，WinRunner を閉じてからこれらのアドインをロードせずに再度開いてください。

1 フライト予約アプリケーションにログインします。

［ログイン］ウィンドウがデスクトップにまだ開いていない場合は，［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**Sample Applications**］＞［**Flight 4A**］を選択します。

［**代理店名**］ボックスに自分の名前を，［**パスワード**］ボックスに「mercury」を入力し，［**OK**］ボタンを押します。フライト予約アプリケーションが開きます。

2 WinRunner を起動します。

WinRunner がまだ開いていない場合は，［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**WinRunner**］を選択します。［WinRunner へようこそ］ウィンドウが表示されたら，［**テストの新規作成**］ボタンをクリックします。［WinRunner へようこそ］ウィンドウが表示されなければ，［**ファイル**］＞［**新規作成**］をクリックします。新規テストのウィンドウが開きます。

3 テスト・スクリプト・ウィザードを起動します。

［**挿入**］＞［**スクリプト ウィザード**］を選択します。［スクリプト ウィザードへようこそ］画面内で［**次へ**］ボタンをクリックし，次の画面に進みます。

4 テストするアプリケーションを指します。

ボタンをクリックしてから**フライト予約**アプリケーションをクリックします。ウィザードの［ウィンドウ名］ボックスにアプリケーションのウィンドウ名が表示されます。［**次へ**］をクリックします。

5 すべてのチェック・ボックスがクリアされていることを確認します。

この練習のために，すべてのチェック・ボックスがクリアされていることを確認します。このウィザードでは，フライト予約アプリケーションの GUI のみを学習します。［**次へ**］をクリックします。

注：回帰テストは，不具合の修正の前後で同じテストを実行して，テスト工程の進捗を見る場合に実行します。回帰テストを行うことによって，期待するテスト結果と実際の結果を比較できます。

---

6 標準のナビゲーション・コントロールを使います。

ナビゲーション・コントロールは，ウィンドウを開くために使われる GUI オブジェクトを WinRunner に指定します。フライト予約アプリケーションは標準のナビゲーション・コントロール（.... , >>）を使うので，コントロールを追加する必要はありません。［**次へ**］をクリックします。

7 学習フローを［速習］に設定します。

学習フローの設定に応じて，WinRunner がアプリケーション全体を学習する方法が決まります。［**速習**］と［**包括学習**］の 2 つのモードがあります。包括学習モードでは，ウィザードが GUI オブジェクトの記述を学習する方法をカスタマイズできます。WinRunner を初めて使用するユーザは速習モードを使うことをお勧めします。

［**学習**］ボタンをクリックします。ウィザードはアプリケーション全体の学習を開始し，メニューをプルダウンしたり，ウィンドウを開いたりして，オブジェクト記述を学習します。この工程には数分間かかります。

インタフェースの要素が無効である旨を知らせるポップアップ・メッセージが表示された場合は，メッセージ・ボックスの［**続行**］ボタンを押します。

ウィザードがウィンドウを閉じることができない場合，ウィンドウを閉じる方法をたずねてきます。画面上の指示に従ってください。

8［アプリケーションの開始］画面で［いいえ］をそのまま受け入れます。

WinRunner を開始するたびに，WinRunner がフライト予約アプリケーションを自動的に開くように指定できます。標準の［**いいえ**］をそのまま受け入れます。［**次へ**］をクリックします。

9 GUI 情報と起動スクリプトを保存します。

ウィザードは GUI 情報を「**GUI マップ・ファイル**」に保存します。

ウィザードは起動スクリプトも作成します。WinRunner を開始するたびに，このスクリプトが自動的に実行されます。これは GUI マップ・ファイルをロードするコマンドを含んでいるので，WinRunner がアプリケーションをテストする準備ができます。

標準のパスとファイル名をそのまま使うか，別のパスとファイル名を指定します。次の標準設定の GUI マップ・ファイル名とパスを受け入れます。
＜ WinRunner インストール・フォルダ＞ ¥dat¥flight4a.gui 選択したディレクトリへの書き込み権限があることを確認してください。［**次へ**］ボタンをクリックします。

10 ［おめでとうございます！］画面の［OK］ボタンを押します。

WinRunner がアプリケーションについて学習した情報は，GUI マップ・ファイルに格納されます。

練習 3

テストの記録

この練習では次のことを学びます。

- コンテキスト・センシティブとアナログ記録モードについて
- コンテキスト・センシティブ・モードでのテスト・スクリプトの記録方法
- テスト・スクリプトの読み方
- アナログ・モードでのテスト・スクリプトの記録方法
- 記録済みテストの実行と結果の分析方法
- テストの記録のヒント

記録モードの選択

ユーザの操作を記録することによって，自動テストのスクリプトを迅速に作成できます。通常のようにオブジェクト上でマウスをクリックしたり，キーボードから入力をしたりして，アプリケーションを操作します。WinRunner はそれらの操作を記録し，Mercury のテスト・スクリプト言語である TSL でステートメントを生成します。これらのステートメントは，［WinRunner］テスト・ウィンドウにスクリプトとして表示されます。

テストの記録を開始する前に，テストの主要なステージを設計し，適切な記録モードを選択する必要があります。記録モードには，「コンテキスト・センシティブ」と「アナログ」の 2 つがあります。

コンテキスト・センシティブ

コンテキスト・センシティブ・モードは，アプリケーション内の GUI オブジェクトに基づいて，操作を記録します。WinRunner は，クリックされた個々のオ

ブジェクト（ウィンドウ，メニュー，リスト，ボタンなど）や，実行された操作の種類（押す，使用可能にする，移動する，選択するなど）を識別します。

例えば，フライト予約アプリケーションの［ログイン］ウィンドウの［**OK**］ボタンに対するマウスのクリックを記録すると，WinRunner はテスト・スクリプトに次のような TSL ステートメントを記録します。

```
button_press("OK");
```

スクリプトを実行すると，WinRunner はコマンドを読み取り，［**OK**］ボタンを探してこれを押します。

アナログ

アナログ・モードの場合，WinRunner はマウス・クリックやキーボード入力のほか，マウスの通った正確な座標を記録します。例えば，［ログイン］ウィンドウの［**OK**］ボタンをクリックすると，WinRunner は以下のようなステートメントを記録します。

記録されたステートメント	意味
move_locator_track (1);	マウスの通った跡 mtype
("<T110><kLeft>-");	左マウス・ボタン押下 mtype
("<kLeft>+");	左マウス・ボタン解放

テストを実行すると，WinRunner は画面上の絶対座標に基づいて，記録された動作を再現します。アプリケーションがデスクトップ上の違う位置にあったり，ユーザ・インタフェースが変わった場合，WinRunner はテストを正しく実行することができません。

注：例えば，絵を描画する操作を再生する場合など，テストにおいてマウスの正確な動作が重要になる場合だけ，アナログ・モードで記録することをお勧めします。

記録モードを選択するときは，次の点を考慮してください。

コンテキスト・センシティブ・モードを選択 ...	**アナログ・モードを選択 ...**
アプリケーションに GUI オブジェクトが含まれている。	アプリケーションが（描画領域のような）ビットマップ領域を含んでいる。
正確なマウスの動作が必要ではない。	正確なマウスの動作が必要。
アプリケーションの異なるバージョンでテストを再利用することを計画している。	

GUI オブジェクトとビットマップ領域の両方を含むアプリケーションをテストしている場合，記録中にモードを切り替えることができます。これについては，練習の中で説明します。

コンテキスト・センシティブ・テストの記録

この練習では，フライト予約アプリケーションで，注文を開くための工程をテストするスクリプトを作成します。コンテキスト・センシティブ・モードで記録してスクリプトを作成します。

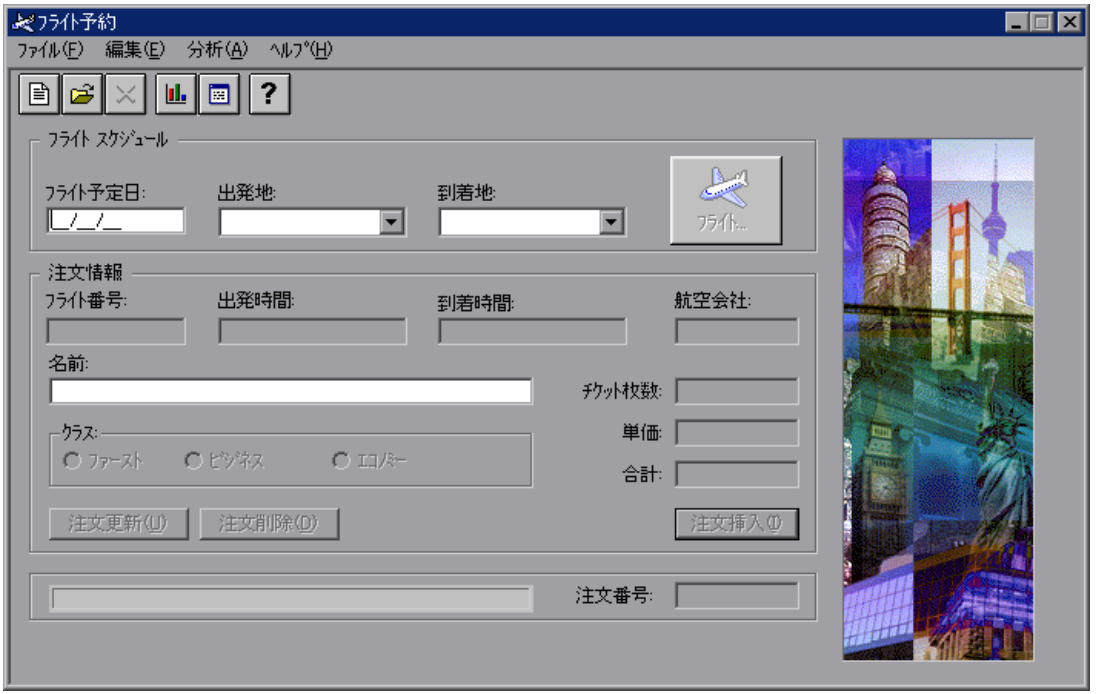
1 WinRunner を起動して，GUI マップをロードします。

WinRunner がまだ開いていない場合は，［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**WinRunner**］を選択します。［WinRunner へようこそ］ウィンドウが表示されたら，［**テストの新規作成**］ボタンをクリックします。［WinRunner へようこそ］ウィンドウが表示されなければ，［**ファイル**］＞［**新規作成**］をクリックします。新規テストのウィンドウが開きます。

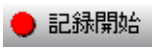
［グローバルな GUI マップ ファイル］モードで作業しているときは，GUI マップがロードされていることを確認してください。これを行うには，［**ツール**］＞［**GUI マップ エディタ**］を選択します。GUI マップ・エディタから［**表示**］＞［**GUI ファイル**］を選択し，**flight4a.GUI** が［**GUI ファイル**］リストに含まれていることを確認してください。GUI マップがロードされていない場合は，［GUI マップ エディタ］で［**ファイル**］＞［**開く**］を選択して，GUI マップを見つけて選択し，［**開く**］をクリックします。

2 フライト予約 アプリケーションを起動し，ログインします。

［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**Sample Applications**］＞［**Flight 4A**］を選択します。［ログイン］ウィンドウで，名前とパスワード「mercury」を入力し，［**OK**］ボタンを押します。入力する名前は 4 文字以上でなければなりません。フライト予約アプリケーションが開きます。



フライト予約 アプリケーションと WinRunner の全体が見えるように両方をデスクトップ上で配置し直します。



3 コンテキスト・センシティブ・モードで記録を開始します。

WinRunner で［**テスト**］＞［**記録 - コンテキスト センシティブ**］を選択するか，ツールバーの［**クリックしてコンテキストセンシティブモードで記録**］ボタンをクリックします。この時点から，WinRunner はすべてのマウス・クリックとキーボード入力を記録します。「REC」というテキストが，記録ボタンの上に青で表示されます。これは，コンテキスト・センシティブ・モードで記録していることを示します。ステータス・バーにも，現在使用している記録モードが表示されます。

4 注文番号 3 を開きます。

フライト予約アプリケーションで，［**ファイル**］＞［**注文を開く**］を選択します。［**注文を開く**］ダイアログ・ボックスで，［**注文番号**］チェック・ボックスをクリックします。隣のフィールドに「3」と入力し［**OK**］を押します。

操作に応じて WinRunner がテスト・スクリプトをテスト・ウィンドウに生成する様子を観察しましょう。

5 記録を停止します。

WinRunner で，［**テスト**］＞［**記録停止**］を選択するか，ツールバー上の［**停止**］ボタンをクリックします。

6 テストを保存します。

［**ファイル**］＞［**上書き保存**］を選択するか，ツールバーの［**保存**］ボタンをクリックします。テストを **lesson3** として保存し，ハード・ディスク内の適切な場所に格納します。

WinRunner が **lesson3** テストを，単独のファイルではなくフォルダとしてファイル・システムに保存することに注意してください。このディレクトリには，テスト・スクリプトとテストを実行したときに生成された結果が保存されます。

テスト・スクリプトについて

前の練習では，フライト予約アプリケーションにおけるフライト注文を開始する工程を記録しました。操作を進めていくにつれて，WinRunner は次に示すようなテスト・スクリプトを生成しました。

```
# フライト予約
    set_window (" フライト予約 ", 3);
    menu_select_item (" ファイル ; 注文を開く ");
# 注文を開く
    set_window (" 注文を開く ", 1);
    button_set (" 注文番号 ", ON);
    edit_set ("Edit_1", "3");
    button_press ("OK");
```

このように，記録された TSL ステートメントは選択したオブジェクトと実行したアクションを示します。例えば，メニュー項目を選択すると，WinRunner は **menu_select_item** ステートメントを生成します。

次の点を参考にすればテスト・スクリプトが理解しやすいでしょう。

- オブジェクトをクリックすると，WinRunner はオブジェクトに「**論理名**」を割り当てます。通常これはオブジェクトのテキスト・ラベルです。論理名があると，テスト・スクリプトが読みやすくなります。例えば，［**注文番号**］チェック・ボックスをクリックすると，WinRunner は次のステートメントを記録しました。

  button_set (" 注文番号 ", ON);

  「**注文番号**」はオブジェクトの論理名です。

- 標準では，新しいウィンドウで作業を始めるたびに，WinRunner はスクリプトを読みやすくするために，自動的にコメント行を追加します。例えば，［フライト予約］ウィンドウ上でクリックしたときに，WinRunner は次のコメント行を生成しました。

  # フライト予約

- 新しいウィンドウで作業を始めるたびに，WinRunner は **set_window** ステートメントを生成します。**set_window** ステートメントに続くステートメントは，そのウィンドウ内のオブジェクトに対する操作を実行します。例えば，［注文を開く］ダイアログ・ボックスを開いたときに，WinRunner は次のステートメントを生成しました。

  set_window (" 注文を開く ", 1);

- キーボードから入力をすると，WinRunner はテスト・スクリプト内に **type**，**obj_type**，**edit_set** などのステートメントを生成します。例えば，［**注文番号**］ボックスに **3** と入力すると，WinRunner は次のステートメントを生成しました。

  edit_set ("Edit", "3");

  WinRunner がキーボード入力を記録する様々な方法の詳細については，［**ヘルプ**］＞［**TSL オンライン リファレンス**］を選択します。

アナログ・モードでの記録

この練習では，ファックスを送信する工程をテストします。コンテキスト・センシティブ・モードで記録を開始して，ファックスに署名を加えるためにアナログ・モードに切り替えた後，再びコンテキスト・センシティブ・モードに切り替えます。

1 **lesson3 のテストで，カーソルをスクリプトの最終行の下に置きます。**

lesson3 テストに新しいテスト・セグメントを追加します。テストをまだ開いていない場合は，［**ファイル**］＞［**開く**］を選択して，テストを選択します。［**lesson3**］テスト・ウィンドウでは，カーソルをテストの最終行の下に置きます。

2 **コンテキスト・センシティブ・モードで記録を開始します。**

［**テスト**］＞［**記録 - コンテキスト センシティブ**］を選択するか，ツールバーの［**クリックしてコンテキストセンシティブモードで記録**］ボタンをクリックします。

3 **［FAX 注文］フォームを開き，ファックス番号を入力します。**

フライト予約 アプリケーションで，［**ファイル**］＞［**FAX 注文**］を選択します。［FAX 番号］ボックスに「4155551234」と入力します。

4 **［署名を注文とともに送信］チェック・ボックスを選択します。**

5 コンテキスト・センシティブ・モードでファックスに署名します。

マウスを使って［**代理店署名**］ボックスに名前を署名します。

WinRunner が署名をどのように記録するか観察してください。

6 署名をクリアします。

［**署名をクリア**］ボタンをクリックします。

7 デスクトップ上で［FAX 注文］ウィンドウを違う位置に移動します。

アナログ・モードに切り替える前に，作業中のウィンドウを再配置します。

8 アナログ・モードで再度ファックスに署名します。

キーボードで **F2** キーを押すか，［**クリックしてコンテキストセンシティブモードで記録**］ボタンを再度クリックしてアナログ・モードに切り替えます。［**クリックしてコンテキストセンシティブモードで記録**］ボタンの「記録」というテキストが赤で表示されることに注意してください。これでアナログ・モードになりました。［**代理店署名**］ボックスに名前を署名します。

WinRunner が署名をどのように記録するか観察してください。

9 コンテキスト・センシティブ・モードに再び切り替えて，ファックスを送信します。

F2 キーを押すか，［**クリックしてコンテキストセンシティブモードで記録**］ボタンをクリックして，コンテキスト・センシティブ・モードに再び切り替えます。［**送信**］をクリックします。アプリケーションがファックスの送信工程をシミュレートします。

10 記録を停止します。

［**テスト**］＞［**記録停止**］を選択するか，［**停止**］ボタンをクリックします。

11 テストを保存します。

［**ファイル**］＞［**上書き保存**］を選択するか，［**保存**］ボタンをクリックします。

12 ［グローバルな GUI マップファイル］モードで操作している場合，新しいオブジェクトを GUI マップに保存します。

練習 2 でテスト・スクリプト・ウィザードを実行したときに，ウィザードはアクセス可能なすべてのウィンドウとオブジェクトを学習しました。しかし，ファックス注文ダイアログ・ボックスは，25 ページの「コンテキスト・センシティブ・テストの記録」手順 4 で行ったように，注文をすでに開いている場合にのみ開きます。したがって，27 ページ の手順 3 でファックス注文ダイアログ・ボックスを開いたときに，WinRunner は新しいウィンドウと，そのウィンドウで記録したオブジェクトを仮の GUI マップに追加しました。仮の GUI マップは WinRunner を終了するたびに破棄されるため，テストが使用した GUI マップに，新しいウィンドウとオブジェクトを保存することが重要です。

［**ツール**］＞［**GUI マップ エディタ**］を選択します。［**表示**］＞［**GUI ファイル**］を選択します。［FAX 注文番号 3］ウィンドウが **L0 < 一時バッファ >**GUI マップ・ファイルに表示されます。［**ファイル**］＞［**上書き保存**］を選択します。［新規ウィンドウ］ダイアログ・ボックスが開きます。**flight4a.GUI** ファイルが［**ロードされている GUI ファイル**］ボックスに表示されます。［**OK**］をクリックします。［FAX 注文番号 3］ウィンドウと，そのウィンドウのすべてのオブジェクトが仮の GUI マップから **flight4a.GUI** マップ・ファイルへ移されます。［**ファイル**］＞［**終了**］を選択し，GUI マップ・エディタを終了します。

注 : ［テスト特有の GUI マップ ファイル］モード で操作している場合，新しいオブジェクトはテストを保存するときに自動的にファイルに追加され，保存されます。オブジェクトを手作業で GUI マップに保存することはお勧めしません。

テストの実行

記録済みのテスト・スクリプトを実行し，テスト結果を分析する準備ができました。WinRunner はテストを実行するために 3 つのモードを提供します。ツールバーでモードを選択します。

➤ アプリケーションの振る舞いを検査するテストを実行して，テスト結果を保存したい場合には，「**検証モード**」を使います。これは，この練習で使用するモードです。

➤ テスト・スクリプトが文法エラーなしでスムーズに動作することを確かめたい場合には，「**デバッグ・モード**」を使います。詳細については，練習 7 を参照してください。

➤ GUI チェックポイントまたはビットマップ・チェックポイントに新しい期待結果を作成したい場合には，「**更新モード**」を使います。詳細については，練習 5 と 6 を参照してください。

テストを実行するには，次のようにします。

1 **WinRunner とフライト予約アプリケーションがデスクトップ上で開いていることを確認します。**

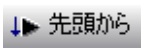
2 **WinRunner で［lesson3］テスト・ウィンドウがアクティブであることを確認します。**

［**lesson3**］テスト・ウィンドウのタイトル・バーをクリックします。まだテストを開いていない場合は，［**ファイル**］＞［**開く**］を選択します。

3 **フライト予約アプリケーションのメイン・ウィンドウがアクティブになっているか確認します。**

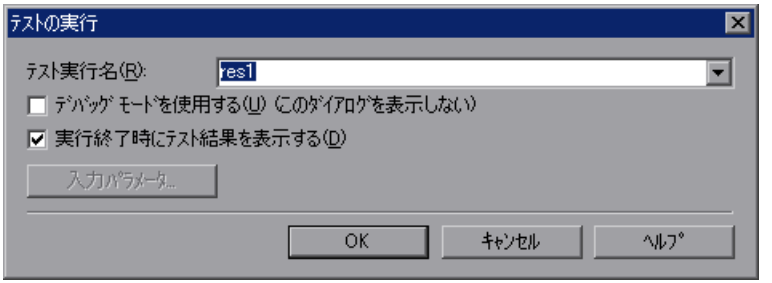
開いているダイアログ・ボックスがあれば閉じます。

4 **ツールバーで検証モードが選択されていることを確認します。**



5 **［先頭から実行］を選択します。**

［**テスト**］＞［**先頭から実行**］を選択するか，［**先頭から実行**］ボタンをクリックします。［テスト実行］ダイアログ・ボックスが開きます。



6 **テスト実行名を選択します。**

WinRunner がテスト結果を格納するフォルダの名前を指定します。標準のフォルダ名「**res1**」を使用します。結果フォルダはテストのフォルダ内に格納されます。

ダイアログ・ボックスの最下部に［**実行終了時にテスト結果を表示する**］チェック・ボックスが表示されます。このチェック・ボックスが選択されている場合，テストの実行が終了すると，WinRunner はテスト結果を自動的に表示します。このチェック・ボックスが選択されていることを確認します。

7 テストを実行します。

［テスト実行］ダイアログ・ボックスで，［**OK**］をクリックします。WinRunner がテストの実行を開始します。

WinRunner がフライト予約アプリケーションの個々のウィンドウをどのように開くか観察しましょう。

8 結果を見ます。

テスト実行が完了すると，テスト結果が［WinRunner テスト結果］ウィンドウに自動的に現れます。テスト結果の分析方法については，次の節で学習します。

テスト結果の分析

テストの実行が完了したら，直ちに［WinRunner テスト結果］ウィンドウでテスト結果を確認できます。

WinRunner テスト結果ビューについて

WinRunner には，2 種類のテスト結果ビューアが用意されています。

- **WinRunner レポート・ビュー**：Windows 形式のビューアです。これは，以前のバージョンの WinRunner でも使用可能でした。
- **統一レポート・ビュー**：HTML 形式のビューアです。このビューアは，QuickTest Professional で使用されるものと同じです。

標準設定では，WinRunner は WinRunner レポート・ビューを使用してテスト結果ウィンドウを開きますが，統一レポート・ビューで結果を表示するために必要なデータも生成します。

このチュートリアルでは，WinRunner レポート・ビューを使用するものと想定します。統一レポート・ビューを使用している場合は，テスト結果ウィンドウはこのチュートリアルで示す画像とは大幅に異なるものとなります。

WinRunner レポート・ビューを使用していることを確認してください。

1 ［**ツール**］＞［**一般オプション**］を選択して，［**実行**］カテゴリを選択します。

2 ［**WinRunner レポート ビュー**］を選択し，［**OK**］をクリックします。

テスト結果の表示

次のステップでは，テスト結果ウィンドウの開き方と表示方法について説明します。以降の練習で，テスト結果に表示される情報の分析も行って，テストまたはテスト対象アプリケーションの問題を特定します。

1 ［WinRunner テスト結果］ウィンドウが開いていて，テスト結果を表示していることを確認します。

［WinRunner テスト結果］ウィンドウが開いていない場合は，テスト・ウィンドウをクリックしてアクティブにしてから，［**ツール**］＞［**テスト結果**］を選択するか，［**テスト結果**］ボタンをクリックします。

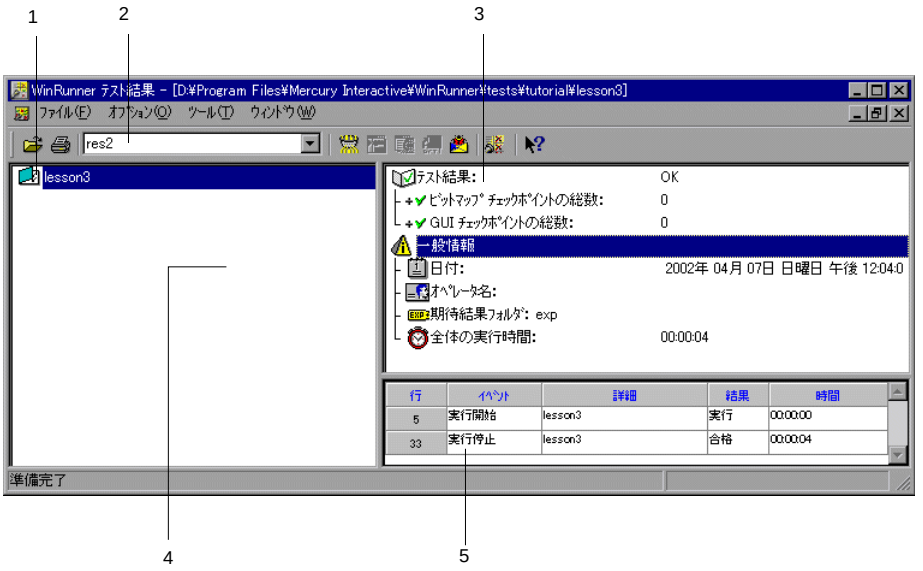
1 現在のテスト名を表示します。

2 現在の結果ディレクトリ名を表示します。

3 テスト実行の成否を表示します。

4 テスト実行についての一般情報（日付，オペレータ名，全体の実行時間）などを表示します。これらの詳細情報を表示するには，［一般情報］アイコンをダブルクリックします。

5 テスト・ログのセクションは，テスト実行中に発生した主なイベントを一覧表示します。また，各イベントが発生したテスト・スクリプト行も表示します。



2 結果を確認します。

3 ［テスト結果］ウィンドウを閉じます。

［WinRunner テスト結果］ウィンドウで，［**ファイル**］＞［**終了**］を選択します。

4 テストを閉じます。

WinRunner のメイン・ウィンドウで，［**ファイル**］＞［**閉じる**］を選択します。

5 フライト予約アプリケーションを閉じます。

［**ファイル**］＞［**終了**］を選択します。

記録に関するヒント

- 記録を開始する前に，テストに必要でないアプリケーションは閉じておきます。
- 始まった場所で終了するように，テストを作成します。例えば，テストがあるアプリケーションを開く場合，テストがテスト実行の最後にそのアプリケーションを閉じるようにします。これは，WinRunner が同じテストを繰り返し実行できることを保証します。
- アナログ・モードで記録する場合，結果が繰り返しのアクションになるならば，マウス・ボタンを押し続けないようにします。例えば，ウィンドウをスクロールする際に，マウス・ボタンを押し続けないようにします。ウィンドウをスクロールする場合は，スクロール・バーの矢印を繰り返しクリックします。これにより，WinRunner はテストを正確に実行できます。
- 記録セッションの途中でコンテキスト・センシティブ・モードからアナログ・モードに切り替えるときに，必ず現在のウィンドウをデスクトップ上の新しい位置に移動してください。これにより，テストを実行するときにテストのアナログ・モードの部分で，ウィンドウの正しい位置にマウス・ポインタを置くことができます。
- 記録中，非標準の GUI オブジェクトをクリックすると，WinRunner は汎用の **obj_mouse_click** ステートメントを，テスト・スクリプトに生成します。例えば，グラフ・オブジェクトをクリックすると，次のような行が記録されます。

  obj_mouse_click (GS_Drawing, 8, 53, LEFT);

  アプリケーションに標準の GUI オブジェクトのように振る舞う非標準の GUI オブジェクトが含まれている場合，このオブジェクトを標準のオブジェクト・クラスにマップすることで，テスト・スクリプトによりわかりやすいステートメントを記録することができます。詳細については，『**WinRunner ユーザーズ・ガイド**』の「GUI マップの構成設定」の章を参照してください。
- ［グローバルな GUI マップ ファイル］モードで操作している場合，事前に記述が学習されなかったオブジェクトをクリックすると，WinRunner はそのオブ

ジェクトの記述を学習し，仮の GUI マップ・ファイルに追加します。詳細については，『**WinRunner ユーザーズ・ガイド**』の「グローバルな GUI マップファイル・モードを使った作業」の章を参照してください。

- **F2** を押すことで，コンテキスト・センシティブ・モードとアナログ・モードを簡単に切り替えられます。
- ［グローバルな GUI マップ ファイル］モードで操作している場合，WinRunner を閉じる前に，新しいウィンドウまたはオブジェクトが仮の GUI マップに追加されているかどうかを必ず確認します。新しいオブジェクトが追加されている場合，テストに対応する適切な GUI マップ・ファイルにそれらを保存します。

練習 4

テストの同期化

この練習では次のことを学びます。

- いつテストを同期化するか
- 同期化の設定の変更方法
- 同期化の問題を特定する方法
- テストを同期化する方法
- テストの実行と同期化および結果の分析方法

いつ同期させるのか？

テストを実行するとき，アプリケーションがいつも同じ速さで入力に応答できるとは限りません。例えば，次のような処理には数秒間かかる可能性があります。

- データベースからの情報の取り出し
- ウィンドウのポップアップ
- 進捗表示バーが 100% に到達するまで
- ステータスを示すメッセージが表示されるまで

WinRunner は，一定の時間，アプリケーションが入力に応答するのを待機します。標準の設定の場合，最大 10 秒間待機します。アプリケーションがテストの実行中にゆっくりと応答する場合，WinRunner は標準の待機時間では不十分となり，テストの実行は思いがけず失敗してしまいます。

テストとアプリケーション間の同期について問題を見つけた場合は，次のどれかを実施します。

- WinRunner の標準の待機時間を増やします。これを行うには，［**ツール**］＞［**一般オプション**］を選択し，［**一般オプション**］ダイアログ・ボックスを開きます。［**実行**］カテゴリで［**設定**］サブカテゴリを選択します。［**チェックポイントと CS ステートメントのタイムアウト**］オプション の値を変更します。この方法はすべてのテストに影響するので，ほかの多くのコンテキスト・センシティブの操作が遅くなります。
- テスト・スクリプト内の問題が起こる正確な場所に，「**同期ポイント**」を挿入します。同期ポイントは WinRunner に，テスト実行を一時停止して，アプリケーションが特定の応答を返すまで待機させます。テストをアプリケーションと同期させるには，この方法をお勧めします。

次の練習では，次のことをします。

- フライト予約 アプリケーションで新規注文を開くテストを作成し，注文をデータベースに挿入する
- 同期設定を変更する
- 同期の問題を特定する
- テストを同期化する
- 同期テストを実行する

テストの作成

この最初の練習では，フライト予約アプリケーションで新規注文を開くテストを作成し，注文をデータベースに挿入します。

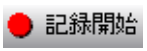
1 WinRunner を起動して，GUI マップをロードします。

WinRunner がまだ開いていない場合は，［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**WinRunner**］を選択します。［WinRunner へようこそ］ウィンドウが表示されたら，［**テストの新規作成**］ボタンをクリックします。［WinRunner へようこそ］ウィンドウが表示されなければ，［**ファイル**］＞［**新規作成**］をクリックします。新規テストのウィンドウが開きます。

［グローバルな GUI マップ ファイル］モードで作業しているときは，GUI マップがロードされていることを確認してください。これを行うには，［**ツール**］＞［**GUI マップ エディタ**］を選択します。GUI マップ・エディタから［**表示**］＞［**GUI ファイル**］を選択し，**flight4a.GUI** が［**GUI ファイル**］リストに含まれていることを確認してください。

2 フライト予約アプリケーションを起動し，ログインします。

［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**Sample Applications**］＞［**Flight 4A**］を選択します。［ログイン］ウィンドウで，名前とパスワード「mercury」を入力し，［**OK**］ボタンを押します。フライト予約アプリケーションと WinRunner の全体が見えるように，両方をデスクトップ上で配置し直します。



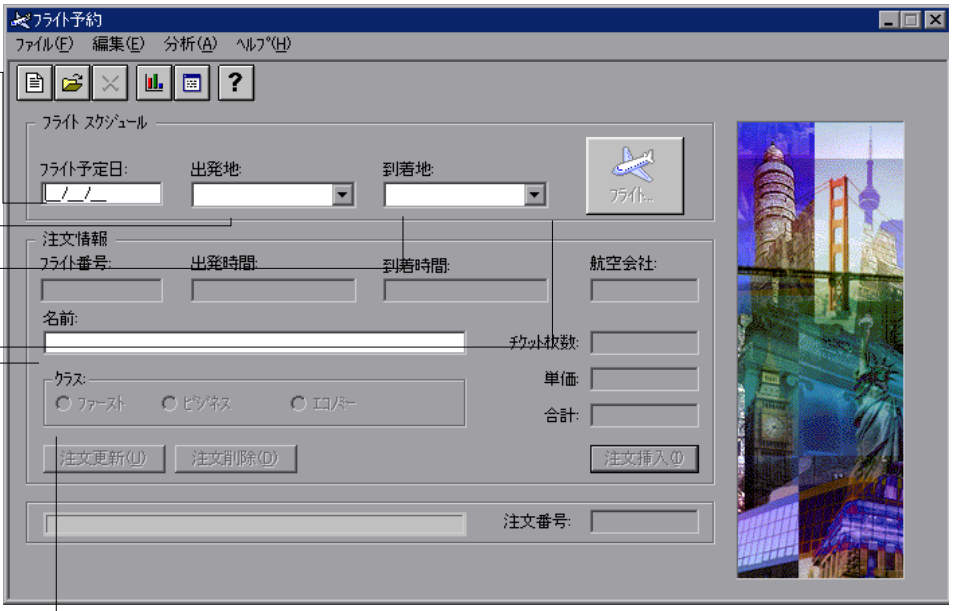
3 コンテキスト・センシティブ・モードで記録を開始します。

［**テスト**］＞［**記録 - コンテキスト センシティブ**］を選択するか，ツールバーの［**クリックしてコンテキストセンシティブモードで記録**］ボタンをクリックします。WinRunner がテストの記録を開始します。

4 新規注文を作成します。

フライト予約 アプリケーションで［**ファイル**］＞［**新規注文**］を選択します。

5 フライトと顧客情報を埋めます。



注：チュートリアルを完了できるよう，入力した日付がチュートリアルを完了する日付よりも先であることを確認してください。フライト予約アプリケーションを使用して予約できるのは将来のフライトだけなので，フライトの日付を過ぎてからこのチュートリアルのテスト・スクリプトを実行しようとしても正しく動きません。

---

6 注文をデータベースに挿入します。

[**注文挿入**] ボタンをクリックします。挿入が完了したら，ステータス・バーに［**挿入を完了しました ...**］と表示されます。

7 注文を削除します。

[**注文削除**] ボタンをクリックし，メッセージ・ウィンドウで［**はい**］をクリックして削除を確認します。

8 記録を停止します。

［**テスト**］＞［**記録停止**］を選択するか，［**停止**］ボタンをクリックします。

9 テストを保存します。

［**ファイル**］＞［**上書き保存**］を選択します。ハードディスクの適当な場所に，**lesson4** としてテストを保存します。

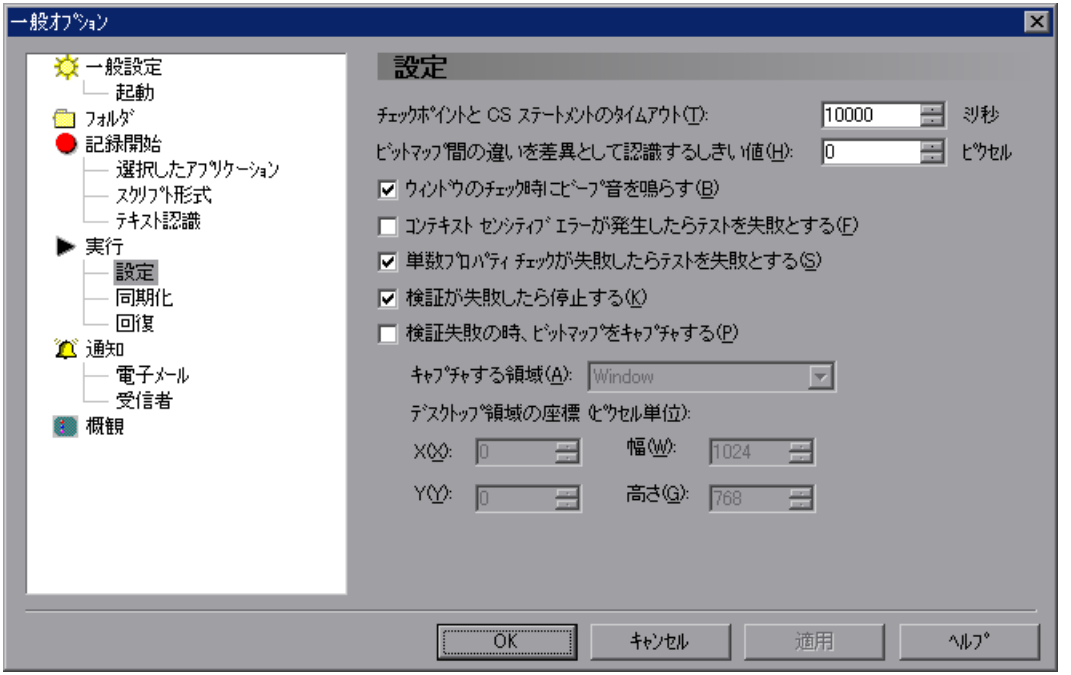
同期設定の変更

標準設定の場合，WinRunner はアプリケーションが入力に応答するのを 10 秒間待機します。次の練習では，同期の問題を特定して同期化ポイントを追加し，問題を解決します。記録したテストを実行して同期の問題を意図的に発生させるために，標準の同期設定を変更する必要があります。

1 ［一般オプション］ダイアログ・ボックスを開きます。

［**ツール**］＞［**一般オプション**］を選択します。

2 ［実行］カテゴリを選択し，［設定］サブカテゴリを選択します。



3 タイムアウトの値を 1000 ミリ秒（1 秒）に変更します。

［**チェックポイントと CS ステートメントのタイムアウト**］ボックスでタイムアウトの値を「**1000**」に変更します。

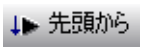
4 ［**OK**］をクリックしてダイアログ・ボックスを閉じます。

同期の問題の検出

lesson4 テストを実行する準備ができました。テストを実行して同期の問題が発生することを確認します。

1 **WinRunner** で［**lesson4**］テスト・ウィンドウがアクティブであることを確認します。

［**lesson4**］テスト・ウィンドウのタイトル・バーをクリックします。



2 ［先頭から実行］を選択します。

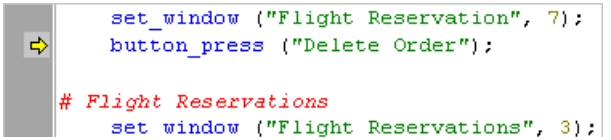
［**テスト**］＞［**先頭から実行**］を選択するか，［**先頭から実行**］ボタンをクリックします。［テスト実行］ダイアログ・ボックスが開きます。標準の名前「**res1**」を使います。［**実行終了時にテスト結果を表示**］チェック・ボックスが選択されていることを確認します。

3 テストを実行します。

［テスト実行］ダイアログ・ボックスで，［**OK**］をクリックします。WinRunner がテストを実行し始めます。**WinRunner が［削除］ボタンを押そうとすると何が起こるか注目してください。**

4 テストの実行を一時停止します。

WinRunner メッセージ・ウィンドウで［**一時停止**］をクリックします。［**注文削除**］ボタンはまだ無効なので，WinRunner はこのボタンのクリックに失敗します。このエラーは，WinRunner が注文挿入の操作が完了するのを待機しなかったために発生したものです。［**注文削除**］ボタンをクリックするコマンドの横で，実行矢印が一時停止したことに注意してください。



```
    set_window ("Flight Reservation", 7);
    button_press ("Delete Order");

# Flight Reservations
    set window ("Flight Reservations", 3);
```

テストの同期化

この練習では，同期ポイントを **lesson4** のテスト・スクリプトに挿入します。同期ポイントは，ステータス・バーの「**挿入を完了しました ...**」というメッセージのビットマップ・イメージをキャプチャします。後でテストを実行する際に，WinRunner は［**注文削除**］ボタンを押そうとする前に，「**挿入を完了しました ...**」というメッセージが現れるのを待機します。

1 デスクトップで，フライト予約ウィンドウが可視になっていることを確認します。

フライト予約アプリケーションと WinRunner の全体が見えるように，両方をデスクトップ上で配置し直します。

2 WinRunner で［lesson4］テスト・ウィンドウがアクティブであることを確認します。

［lesson4］テスト・ウィンドウのタイトル・バーをクリックします。

3 テストを同期させたい場所にカーソルを置きます。

```
button_press (" 注文挿入 ");
```

上記のステートメントの下に空白行を追加します。カーソルを空白行の先頭に置きます。

4 テストを同期化し，ステータス・バー上に「挿入を完了しました ...」というメッセージが現れるのを待機します。

［**挿入**］＞［**同期化ポイント**］＞［**オブジェクト / ウィンドウ ビットマップ**］を選択するか，ユーザ定義ツールバー上の［**オブジェクト / ウィンドウのビットマップの同期化ポイント**］ボタンをクリックします。

ポインタを使用して，［フライト予約］ウィンドウの「**挿入を完了しました** ...」というメッセージをクリックします。WinRunner は **obj_wait_bitmap** 同期ポイントをテスト・スクリプトに自動的に挿入します。このステートメントは WinRunner に，ステータス・バーに「**挿入を完了しました** ...」というメッセージが現れるのを 1 秒間待機するよう指示します。

5 スクリプト内で待機時間を 1 秒から 10 秒に手作業で変更します。

前の手順で挿入した 1 秒の待機時間では十分ではありません。次のステートメントを検索します。

```
obj_wait_bitmap(" 挿入を完了しました ...", "Img1", 1);
```

上記のステートメントの最後の 1 を 10 に変更し，10 秒待機させます。

6 テストを保存します。

［**ファイル**］＞［**上書き保存**］を選択するか，［**保存**］ボタンをクリックします。

7「グローバルな GUI マップ」モードで作業している場合は，ローカルの GUI マップから flight4a GUI マップに新規オブジェクトを追加します。

このテスト中に，フライト予約ウィンドウで新規オブジェクトに同期化ポイントを実行します（［**挿入を完了しました**］ビットマップ）。このオブジェクトを GUI マップに追加しなければなりません。

GUI マップに既に存在するウィンドウからオブジェクトを保存するには，［**ツール**］＞［**GUI マップ エディタ**］を選択します。［**表示**］＞［**GUI ファイル**］を選択します。新しいオブジェクトが **L0 <一時バッファ>** GUI マップ・ファイルに表示されます（表示されない場合は，［**表示**］＞［**オブジェクト ツリーの展開**］を選択します)。［**ファイル**］＞［**上書き保存**］を選択します。

既存ウィンドウから新しいオブジェクトがこのウィンドウを含む **flight4a.GUI** マップに追加されたことを示す WinRunner メッセージが表示されます。［**はい**］をクリックします。

8 修正された flight4a.GUI マップを保存します。

GUI ファイル・ボックスで，「**L1 flight4a.GUI**」を選択します。GUI マップ名の横にアスタリスクが付きます。これはファイルが変更され（［**挿入を完了しました ...**］を追加した)，まだ保存されていないことを意味します。［**ファイル**］＞［**上書き保存**］を選択して，GUI マップを保存します。

9［GUI マップ エディタ］を閉じます。

［**ファイル**］＞［**終了**］を選択し，GUI マップ・エディタを終了します。

参考：

同期化ポイントは，テスト・スクリプト内で **obj_wait_bitmap** ステートメントまたは **win_wait_bitmap** ステートメントとして表されます。次に例を示します。

```
obj_wait_bitmap(" 挿入を完了しました ...", "Img1", 10);
```

挿入を完了しました ... は，オブジェクトの論理名です。

Img1 は，オブジェクトのキャプチャされた画像を含んでいるファイルです。

10 は，WinRunner がアプリケーションに画像が現れるのを待機する時間（単位：秒）です。この時間は **timeout-msec** テスト・オプションで定義された標準の時間に追加されます（上の例では，WinRunner は合計 11 秒待機します)。

同期テストの実行

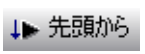
この練習では，同期させたテスト・スクリプトを実行し，テスト結果を検証します。

1 WinRunner で［lesson4］テスト・ウィンドウがアクティブであることを確認します。

［lesson4］テスト・ウィンドウのタイトル・バーをクリックします。

2 テスト・ツールバーで検証モードが選択されていることを確認します。

他のモードを選択するまで，検証実行モードが選択された状態になっています。



3［先頭から実行］を選択します。

［**テスト**］＞［**先頭から実行**］を選択するか，［**先頭から実行**］ボタンをクリックします。［テスト実行］ダイアログ・ボックスが開きます。標準の名前である「**res2**」を使います。［**実行終了時にテスト結果を表示**］チェック・ボックスが選択されていることを確認します。

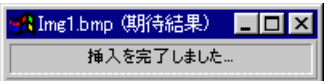
4 テストを実行します。

［テスト実行］ダイアログ・ボックスで，［**OK**］をクリックします。WinRunner はスクリプトの先頭行からテストの実行を開始します。

WinRunner が，「挿入を完了しました ...」というメッセージがステータス・バーに現れるのをどのように待機するか注目してください。

5 結果を確認します。

テストの実行が完了すると，テスト結果が［WinRunner テスト結果］ダイアログ・ボックスに表示されます。「**ビットマップ待機**」イベントがテスト・ログ・セクション内に緑色で表示されます。これは，同期が問題なく実行されたことを示します。このイベントをダブルクリックして，「**挿入が完了しました** ...」というメッセージを表示するステータス・バーのビットマップ・イメージを見ることができます。



6［テスト結果］ウィンドウを閉じます。

［**ファイル**］＞［**終了**］を選択します。

7 lesson4 のテストを閉じます。

WinRunner で［**ファイル**］＞［**閉じる**］を選択します。

8 フライト予約アプリケーションを閉じます。

［**ファイル**］＞［**終了**］を選択します。

9 タイムアウト値を 10000 ミリ秒（10 秒）に戻します。

［**ツール**］＞［**一般オプション**］を選択して，［一般オプション］ダイアログ・ボックスを開きます。［**実行**］カテゴリで［**設定**］サブカテゴリを選択します。［**チェックポイントと CS ステートメントのタイムアウト**］ボックスでタイムアウトの値を「**10000**」に変更します。［**OK**］をクリックし，ダイアログ・ボックスを閉じます。

同期の方法についてさらに学習するには，『**WinRunner ユーザーズ・ガイド**』の「テスト実行の同期化」の章を参照してください。

練習 5

GUI オブジェクトの検査

この練習では次のことを学びます。

- GUI オブジェクトの振る舞いの検査方法
- GUI オブジェクトを検査するテストの作成方法
- バージョンの異なるアプリケーションでのテストの実行と結果の検証
- GUI オブジェクトの検査のヒント

GUI オブジェクトを検査する方法

アプリケーションを使っているときには，GUI オブジェクトの動作を観察することでアプリケーションが正しく機能しているかどうかを見極めることができます。GUI オブジェクトが入力に対して期待どおりに応答しない場合は，おそらくアプリケーションのコードのどこかに不具合が存在します。

「GUI チェックポイント」を作成して，GUI オブジェクトを検査します。GUI チェックポイントは，オブジェクトのプロパティの振る舞いを検証します。例えば，以下のような検査が可能です。

編集ボックスの内容

ラジオ・ボタンがオンかオフか

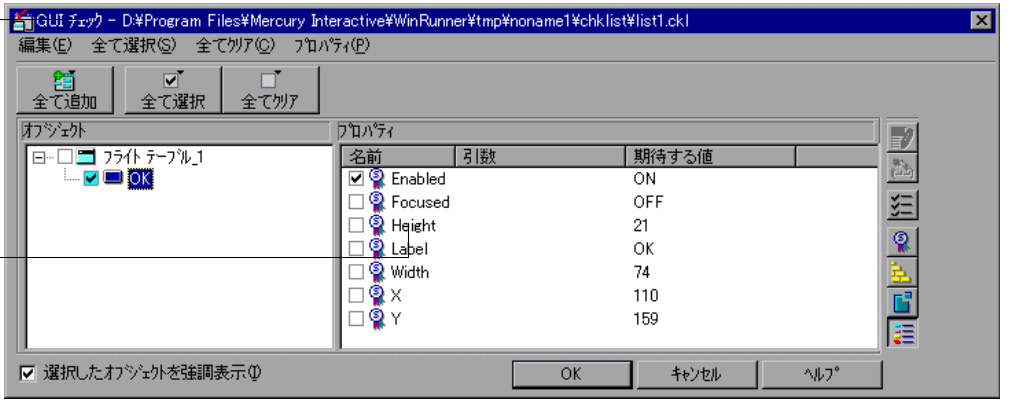
プッシュ・ボタンが有効か無効か

単独のオブジェクトを対象とする GUI チェックポイントを作成するには，はじめにアプリケーションの中のオブジェクトをポインタで指します。

- オブジェクトを**シングルクリック**すると，チェックリストと選択したオブジェクトの標準の検査がテスト・スクリプトに挿入されます。**チェックリスト**には GUI オブジェクトと検査する選択されたプロパティの情報が含まれます。
- オブジェクトを**ダブルクリック**すると［GUI チェック］ダイアログ・ボックスが開き，選択したオブジェクトが表示されます。検査したいプロパティを選択し［**OK**］をクリックして，オブジェクトに対するチェックリストをテスト・スクリプトに挿入します。

［**注文挿入**］プッシュ・ボタンをダブルクリックすると，このダイアログ・ボックスが開きます。

検査するプロパティを選択します。プッシュ・ボタンの標準の検査は「**有効**」になっています。



オブジェクトの標準のプロパティの検査を選択しても，特定のオブジェクトのプロパティの検査を選択しても，WinRunner は，選択されたプロパティの現在値をキャプチャして，この情報を「**期待結果**」として保存します。オブジェクトを検査している場合には **obj_check_gui** ステートメントをテスト・スクリプトに挿入し，ウィンドウを検査している場合には **win_check_gui** ステートメントを挿入します。

アプリケーションの新しいバージョンでこのテストを実行するときには，WinRunner はアプリケーション内のオブジェクトの期待される振る舞いと「**実際の**」振る舞いを比較します。

テスト・スクリプトへの GUI チェックポイントの追加

この練習では，既存の注文を開くときに，フライト予約アプリケーションの［注文を開く］ダイアログ・ボックスのオブジェクトが正しく機能するかどうかを検査します。

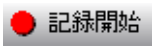
1 WinRunner を起動して，GUI マップをロードします。

WinRunner がまだ開いていない場合は，［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**WinRunner**］を選択します。［WinRunner へようこそ］ウィンドウが表示されたら，［**テストの新規作成**］ボタンをクリックします。［WinRunner へようこそ］ウィンドウが表示されなければ，［**ファイル**］＞［**新規作成**］をクリックします。新規テストのウィンドウが開きます。

［グローバルな GUI マップ ファイル］モードで作業しているときは，GUI マップがロードされていることを確認してください。これを行うには，［**ツール**］＞［**GUI マップ エディタ**］を選択します。GUI マップ・エディタから［**表示**］＞［**GUI ファイル**］を選択し，**flight4a.GUI** が［**GUI ファイル**］リストに含まれていることを確認してください。

2 フライト予約アプリケーションを開始し，ログインします。

［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**Sample Applications**］＞［**Flight 4A**］を選択します。［ログイン］ウィンドウで，名前とパスワード「mercury」を入力し，［**OK**］ボタンを押します。フライト予約アプリケーションと WinRunner の全体が見えるように，両方をデスクトップ上で配置し直します。

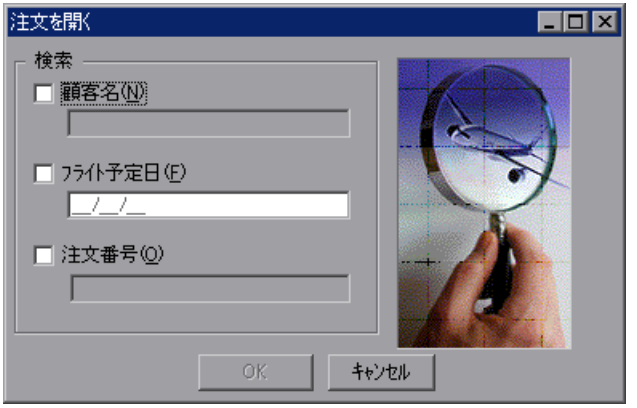


3 コンテキスト・センシティブ・モードで記録を開始します。

［**テスト**］＞［**記録 - コンテキストセンシティブ**］を選択するか，ツールバーの［**クリックしてコンテキストセンシティブモードで記録**］ボタンをクリックします。

4［注文を開く］ダイアログ・ボックスを開きます。

フライト予約アプリケーションで，［**ファイル**］＞［**注文を開く**］を選択します。

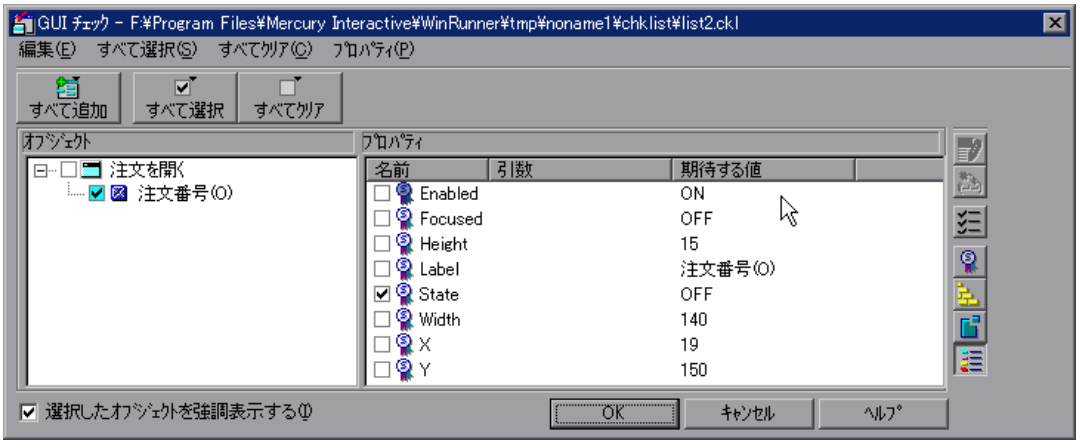


5［注文番号］チェック・ボックスに **GUI** チェックポイントを作成します。

WinRunner のメイン・ウィンドウで，［**挿入**］＞［**GUI チェックポイント**］＞［**オブジェクト / ウィンドウ**］を選択するか，ユーザ定義ツールバーの［**オブジェクト / ウィンドウの GUI チェックポイント**］ボタンをクリックします。

注：標準設定では，ユーザ定義ツールバーは非表示になっています。ユーザ定義ツールバーを開くには，［**表示**］＞［**ユーザ定義ツールバー**］を選択します。ユーザ定義ツールバーの詳細については，3 ページ「［WinRunner］ウィンドウ」を参照してください。

ポインタを使って，［**注文番号**］チェック・ボックスを「**ダブルクリック**」します。［GUI チェック］ダイアログ・ボックスが開き，指定可能な検査が表示されます。［注文番号］チェック・ボックスを**シングルクリック**しただけでは，このダイアログ・ボックスは開きません。標準の検査である「**State**」を使います。この検査はチェック・ボックスの現在の状況（オフ）をキャプチャし，期待結果として保存します。



［**GUI チェック**］ダイアログ・ボックスで［**OK**］をクリックして，テスト・スクリプトにチェックポイントを挿入します。チェックポイントは **obj_check_gui** ステートメントとして現れます。

6 注文番号に「4」を入力します。

［注文を開く］ウィンドウで，［**注文番号**］チェック・ボックスを選択し，［**注文番号**］テキスト・ボックスに「4」と入力します。

7 ［注文番号］チェック・ボックス用の他の GUI チェックポイントを作成します。

［**挿入**］＞［**GUI チェックポイント**］＞［**オブジェクト / ウィンドウ**］を選択するか，ユーザ定義ツールバー上の［**オブジェクト / ウィンドウの GUI チェックポイント**］ボタンをクリックします。

ポインタを使って，［**注文番号**］チェック・ボックスを「**シングルクリック**」します。WinRunner は，すぐに標準の検査である「**State**」を検査するテスト・スクリプトにチェックポイント（**obj_check_gui** ステートメント）を挿入します(オブジェクトの標準の検査使用する場合のみシングルクリック・オプションを使用します)。この検査はボタンの現在の状態（オン）をキャプチャし，期待結果として保存します。

注：チェックポイントでオブジェクトとプロパティを確認するには，［GUI チェック］ダイアログ・ボックスを開く必要があります。このダイアログ・ボックスとその他の GUI チェックポイント・ダイアログ・ボックスの詳細については，『**WinRunner ユーザーズ・ガイド**』の「GUI オブジェクトの検査」の章を参照してください。

8［顧客名］チェック・ボックス用の GUI チェックポイントを作成します。

［**挿入**］＞［**GUI チェックポイント**］＞［**オブジェクト / ウィンドウ**］を選択するか，ユーザ定義ツールバー上の［**オブジェクト / ウィンドウの GUI チェックポイント**］ボタンをクリックします。

ポインタを使用し，［**顧客名**］チェック・ボックスを「**ダブルクリック**」します。［**GUI チェック**］ダイアログ・ボックスが開き，指定可能な検査が表示されます。標準の検査である「**State**」を使用し，追加の検査として「**Enabled**」を選択します。**State** 検査はチェック・ボックスの現在の状況（オフ）をキャプチャし，**Enabled** 検査は現在の状態（オフ）をキャプチャします。

［**GUI チェック**］ダイアログ・ボックスで［**OK**］をクリックして，テスト・スクリプトにチェックポイントを挿入します。チェックポイントは **obj_check_gui** ステートメントとして現れます。

9 **［注文を開く］ダイアログ・ボックスで［OK］をクリックして，注文を開きます。**

10 **記録を停止します。**

［**テスト**］＞［**記録停止**］を選択するか，［**停止**］ボタンをクリックします。

11 **テストを保存します。**

［**ファイル**］＞［**上書き保存**］選択するか，［**保存**］ボタンをクリックします。ハードディスク上の任意の場所に「**lesson5**」と名前を付けてテストを保存します。

参考：

GUI チェックポイントは，テスト・スクリプト上に **obj_check_gui** と **win_check_gui** ステートメントとして現れます。次に例を示します。

obj_check_gui(" **注文番号** ", "list1.ckl", "gui1", 1);

注文番号は，オブジェクトの論理名です。

list1.ckl は，選択された検査を含んでいるチェックリストです。

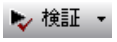
gui1 は，キャプチャされた GUI データを含んでいるファイルです。

1 は，検査をするために必要な時間（秒）です。この時間は，テスト・オプションの［**timeout_msec**］の値に加えられます。詳細については，練習 4 の 35 ページ「テストの同期化」を参照してください。

テストの実行

lesson5 のテストを実行し，テストがスムーズに動作することを検証する準備ができました。

1 **デスクトップで，フライト予約アプリケーションが開いていることを確認します。**



2 **WinRunner のテスト・ツールバーで，［検証］モードが選択されていることを確認します。**

3 **［先頭から実行］を選択します。**

［**テスト**］＞［**先頭から実行**］を選択するか，［**先頭から実行**］ボタンをクリックします。［**テスト実行**］ダイアログ・ボックスが開きます。標準の名前「**res1**」を使います。［**実行終了時にテスト結果を表示**］チェック・ボックスが選択されていることを確認します。

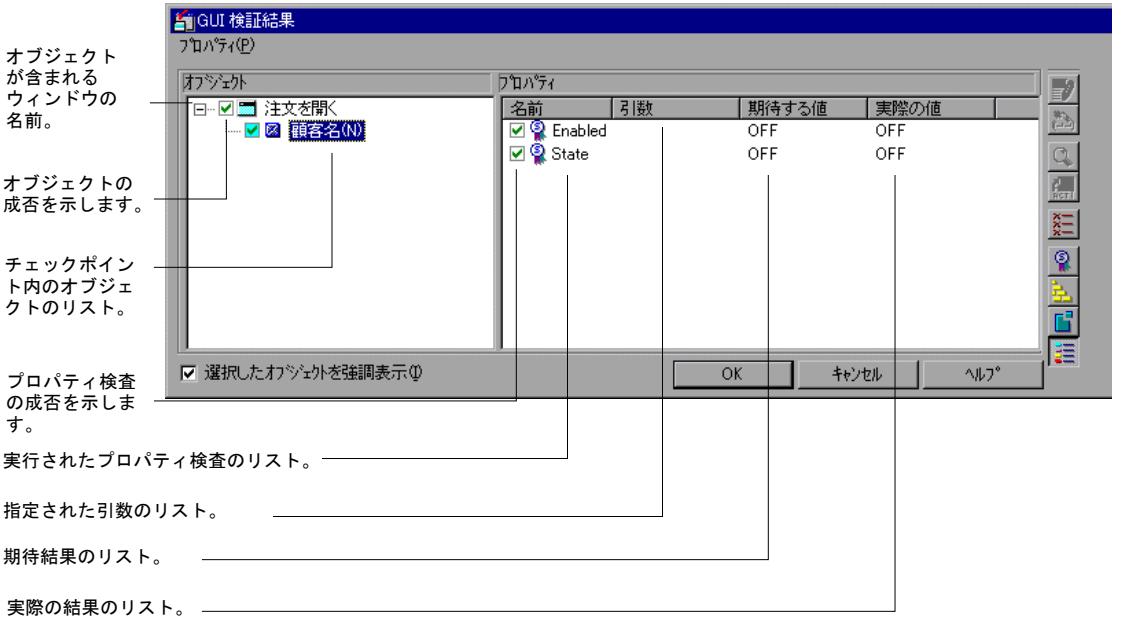
4 **テストを実行します。**

［**テストの実行**］ダイアログ・ボックス［**OK**］ボタンをクリックします。

5 **結果を確認します。**

テストの実行が完了すると，テスト結果が［WinRunner テスト結果］ダイアログ・ボックスに表示されます。テスト・ログ・セクション内に，すべての［**GUI 検査終了**］イベントが緑色（成功を示す）になっているはずです。

［GUI 検査終了］イベントをダブルクリックして，その GUI チェックポイントの詳細結果を表示します。［GUI 検証結果］ウィンドウが開きます。［**顧客名**］を選択して以下のダイアログ・ボックスを表示します。



注：選択したプロパティで検査の引数を指定できます。詳細については，『**WinRunner ユーザーズ・ガイド**』の「GUI オブジェクトの検査」の章を参照してください。

---

6 結果を閉じます。

［**OK**］をクリックして，［GUI チェックポイント結果］ダイアログ・ボックスを閉じます。［**ファイル**］＞［**終了**］を選択し，［**テスト結果**］ウィンドウを閉じます。

7 フライト予約アプリケーションを閉じます。

［**ファイル**］＞［**終了**］を選択します。

新しいバージョンでのテスト実行

この練習では，フライト予約 アプリケーションの新しいバージョンで **lesson5** テストを実行し，GUI オブジェクトの振る舞いを検査します。

1 フライト予約アプリケーションの Flight 4B バージョンを開きます。

［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**Sample Applications**］＞［**Flight 4B**］を選択します。［ログイン］ウィンドウで，名前とパスワード「mercury」を入力して，［**OK**］をクリックします。フライト予約 アプリケーションと WinRunner の全体が見えるように両方をデスクトップ上で配置し直します。

2 lesson5 がアクティブなテストであることを確認します。

WinRunner の［**lesson5**］テスト・ウィンドウをクリックします。

3 ツールバーで検証モードが選択されていることを確認します。

4 ［先頭から実行］を選択します。

［**テスト**］＞［**先頭から実行**］を選択するか，［**先頭から実行**］ボタンをクリックします。［テスト実行］ダイアログ・ボックスが開きます。標準の名前「**res1**」を使います。［**実行終了時にテスト結果を表示**］チェック・ボックスが選択されていることを確認します。

5 テストを実行します。

［**OK**］をクリックします。WinRunner は，テストの実行を開始します。この工程は少々時間がかかることがあります。

GUI チェックポイントで不一致が検出された場合は，メッセージ・ウィンドウで［**続行**］をクリックしてください。

6 結果を確認します。

テストの実行が完了すると，テスト結果が［WinRunner テスト結果］ダイアログ・ボックスに表示されます。テスト・ログ・セクション内に，1 件の「**GUI 検査終了**」チェックポイント・ステートメントが赤で現れ，その「**結果**」フィールドに「**不一致**」と表示されます。これは，オブジェクトに対して実行された検査の 1 つまたは複数が失敗したことを示します。

赤色の［**GUI 検査終了**］イベントをダブルクリックして，失敗したテストの詳細結果を表示します。［GUI 検証結果］ウィンドウが開きます。［顧客名］を選択して以下のダイアログ・ボックスを表示します。

7 ［テスト結果］ウィンドウを閉じます。

［GUI チェックポイント］ダイアログ・ボックスの［**OK**］をクリックしてから，［**ファイル**］＞［**終了**］を選択して［WinRunner テスト結果］ウィンドウを閉じます。

8 **lesson5 のテストを閉じます。**

［**ファイル**］＞［**閉じる**］を選択します。

9 **フライト予約アプリケーションの Flight 4B バージョンを閉じます。**

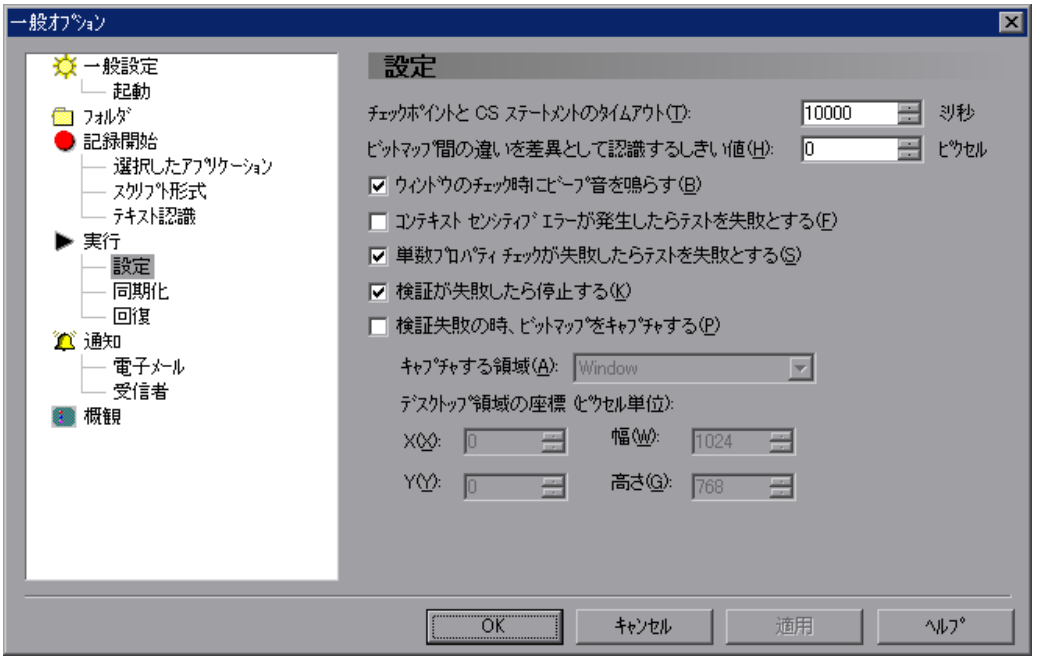
［**ファイル**］＞［**終了**］を選択します。

GUI チェックポイントに関するヒント

- ウィンドウ内の一部または全部のオブジェクトを検査する単独の GUI チェックポイントをテスト内に作ることができます。［**挿入**］＞［**GUI チェックポイント**］＞［**複数のオブジェクト**］を選択します（このメニュー・コマンドは，テストが開いている場合のみ使用できます）。GUI チェックポイント作成ダイアログ・ボックスが開くので，オブジェクトを指定して検査を選択します。チェックリストの作成を完了すると，WinRunner はテスト・スクリプトに win_check_gui ステートメントを挿入します。

- 夜間にテストを実行するときのために，WinRunner に GUI の不一致が検出されてもメッセージを表示しないように指示できます。［**ツール**］＞［**一般オプション**］を選択します。［一般オプション］ダイアログ・ボックスで，［実行］カテゴリから［設定］サブカテゴリを選択して，［**検証が失敗したら停止する**］チェック・ボックスをクリアします。これにより，テストは中断することなく実行されます。

  テスト実行オプションの詳細については，『**WinRunner ユーザーズ・ガイド**』の「グローバル・テスト・オプションの設定」の章，または「テスト・スクリプトからのテスト・オプションの設定］の章を参照してください。



➤ GUI チェックポイントのための新しい期待結果を作成したい場合には，テストを更新モードで実行します。WinRunner は，既存の期待 GUI データを，更新実行中にキャプチャされた新しいデータで上書きします。

GUI チェックポイントの詳細については，『**WinRunner ユーザーズ・ガイド**』の「GUI オブジェクトの検査」の章を参照してください。

練習 6

ビットマップの検査

この練習では次のことを学びます。

- アプリケーション内のビットマップ・イメージの検査方法
- ビットマップを検査するテストの作成方法
- アプリケーションのバージョン間でビットマップを比較するためのテストの実行方法
- 結果の分析方法
- ビットマップを検査する際のヒント

ビットマップの検査方法

アプリケーションに絵やグラフといったビットマップ領域が含まれている場合，こうした領域をビットマップ・チェックポイントを使って検査できます。ビットマップ・チェックポイントは，キャプチャされたビットマップ・イメージをピクセルごとに比較します。

ビットマップ・チェックポイントを作成するには，検査したい領域，ウィンドウまたはオブジェクトを指定します。例を次に示します。

WinRunner はビットマップ・イメージをキャプチャして，これを「**期待結果**」として保存します。オブジェクトをキャプチャする場合には **obj_check_bitmap** ステートメントをテスト・スクリプトに挿入し，領域やウィンドウをキャプチャする場合には **win_check_bitmap** ステートメントを挿入します。

アプリケーションの新しいバージョンを対象にテストを実行すると，WinRunner はアプリケーションにおいて，期待されたビットマップを実際のビットマップと比較します。何らかの違いが検出された場合には，［WinRunner テスト結果］ウィンドウでその違いをイメージで見ることができます。

テスト・スクリプトへのビットマップ・チェックポイントの追加

この練習では［FAX 注文］ダイアログ・ボックス内の［代理店署名］ボックスをテストします。ビットマップ・チェックポイントを使用して，ボックス内に署名できることを検査します。次に，別のビットマップ・チェックポイントを使用して，［**署名をクリア**］ボタンをクリックしたときにボックスがクリアされることを検査します。

1 WinRunner を起動し，GUI マップをロードします。

WinRunner がまだ開いていない場合は，［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**WinRunner**］を選択します。［WinRunner へようこそ］ウィンドウが表示されたら，［**テストの新規作成**］ボタンをクリックします。［WinRunner へようこそ］ウィンドウが表示されなければ，［**ファイル**］＞［**新規作成**］をクリックします。新規テストのウィンドウが開きます。

［グローバルな GUI マップ ファイル］モードで作業しているときは，GUI マップがロードされていることを確認してください。これを行うには，［**ツール**］＞［**GUI マップ エディタ**］を選択します。GUI マップ・エディタから［**表示**］＞［**GUI ファイル**］を選択し，**flight4a.GUI** が［**GUI ファイル**］リストに含まれていることを確認してください。

2 フライト予約 アプリケーションを起動し，ログインします。

［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**Sample Applications**］＞［**Flight 4A**］を選択します。［ログイン］ウィンドウで，名前とパスワード「mercury」を入力し，［**OK**］ボタンを押します。フライト予約アプリケーションと WinRunner の全体が見えるように，両方をデスクトップ上で配置し直します。

3 コンテキスト・センシティブ・モードで記録を開始します。

［**テスト**］＞［**記録 - コンテキストセンシティブ**］を選択するか，ツールバーの［**クリックしてコンテキストセンシティブモードで記録**］ボタンをクリックします。

4 注文番号 6 を開きます。

フライト予約 アプリケーションで，［**ファイル**］＞［**注文を開く**］を選択します。［注文を開く］ダイアログ・ボックスで［**注文番号**］チェック・ボックスを選択し，隣のボックスに「6」と入力します。［**OK**］をクリックして，注文を開きます。

5 ［FAX 注文］ダイアログ・ボックスを開きます。

［**ファイル**］＞［**FAX 注文**］を選択します。

6 ［FAX 番号］ボックスに 10 桁のファックス番号を入力します。

括弧やダッシュを入れる必要はありません。

7 ［FAX 注文］ダイアログ・ボックスを移動します。

［フライト予約］ウィンドウの上に重ならないように，ダイアログ・ボックスを配置します。

8 アナログ・モードに切り替えます。

キーボードの **F2** キーを押すか，［**クリックしてコンテキストセンシティブモードで記録**］ボタンをクリックして，アナログ・モードに切り替えます。

9 ［代理店署名］ボックスに名前を署名します。

10 再びコンテキスト・センシティブ・モードに戻します。

キーボードの **F2** キーを押すか，［**クリックしてコンテキストセンシティブモードで記録**］ボタンをクリックして，コンテキスト・センシティブ・モードに切り替えます。

11 署名を検査するビットマップ・チェックポイントを挿入します。

［**挿入**］＞［**ビットマップ チェックポイント**］＞［**オブジェクト / ウィンドウ**］を選択するか，ユーザ定義ツールバーの［**オブジェクト / ウィンドウのビットマップ チェックポイント**］ボタンをクリックします。

ポインタで［代理店署名］ボックスをクリックします。WinRunner はビットマップをキャプチャして，**obj_check_bitmap** ステートメントをテスト・スクリプトに挿入します。

12 ［署名をクリア］ボタンを押します。

［代理店署名］ボックスから署名がクリアされます。

13 ［代理店署名］ボックスを検査する別のビットマップ・チェックポイントを挿入します。

［**挿入**］＞［**ビットマップ チェックポイント**］＞［**オブジェクト / ウィンドウ**］を選択するか，ユーザ定義ツールバーの［**オブジェクト / ウィンドウのビットマップ チェックポイント**］ボタンをクリックします。

ポインタで［代理店署名］ボックスをクリックします。WinRunner はビットマップをキャプチャして，**obj_check_bitmap** ステートメントをテスト・スクリプトに挿入します。

14 ［FAX 注文］ダイアログ・ボックスの［キャンセル］ボタンを押します。

15 記録を停止します。

［**テスト**］＞［**記録停止**］を選択するか，［**停止**］ボタンをクリックします。

16 テストを保存します。

［**ファイル**］＞［**上書き保存**］選択するか，［**保存**］ボタンをクリックします。ハードディスク上の任意の場所に「**lesson6**」という名前でテストを保存します。

17「グローバルな GUI マップ」モードで作業している場合は，ローカルの GUI マップから flight4a GUI マップに新規オブジェクトを追加します。

GUI マップに既に存在するウィンドウからオブジェクトを保存するには，［**ツール**］＞［**GUI マップ エディタ**］を選択します。［**表示**］＞［**GUI ファイル**］を選択します。新しいオブジェクトが **L0 ＜一時バッファ＞** GUI マップ・ファイルに表示されます（表示されない場合は，［**表示**］＞［**オブジェクト ツリーの展開**］を選択します)。［**ファイル**］＞［**上書き保存**］を選択します。既存ウィンドウから新しいオブジェクトがこのウィンドウを含む **flight4a.GUI** マップに追加されたことを示す WinRunner メッセージが表示されます。［**はい**］をクリックします。

18 修正された flight4a.GUI マップを保存します。

GUI ファイル・ボックスで，「**L1 flight4a.GUI**」を選択します。GUI マップ名の横にアスタリスクが付きます。これは，ファイルが変更され，保存されていないことを意味します。［**ファイル**］＞［**上書き保存**］を選択して，GUI マップを保存します。

19［GUI マップ エディタ］を閉じます。

［**ファイル**］＞［**終了**］を選択し，GUI マップ・エディタを終了します。新規ウィンドウと新規オブジェクトの保存の詳細については，42 ページの手順 7 を参照してください。

参考：

ビットマップ・チェックポイントは，テスト・スクリプト上に **obj_check_bitmap** と **win_check_bitmap** ステートメントとして現れます。次に例を示します。

```
obj_check_bitmap("(static)", "Img1", 1);
```

static は，オブジェクトまたは領域の論理名です。

Img1 は，キャプチャされたビットマップを含んでいるファイルです。

1 は，検査をするために必要な時間（秒）です。この時間は，テスト・オプションの［**タイムアウト**］の値に加えられます。詳細については，練習 4 の 35 ページ「テストの同期化」を参照してください。

期待結果の表示

lesson6 のテストの期待結果を表示する準備ができました。

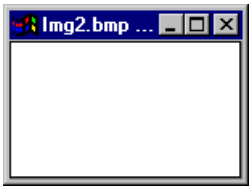
1［WinRunner テスト結果］ウィンドウを開きます。

［**ツール**］＞［**テスト結果**］を選択するか，［**テスト結果**］ボタンをクリックします。［WinRunner テスト結果］ウィンドウが開きます。

2 キャプチャされたビットマップを表示します。

テスト・ログ・セクションで，最初の「ビットマップのキャプチャ」イベントをダブルクリックし，［**表示**］ツールバー・ボタンをクリックします。

次に 2 番目の［ビットマップのキャプチャ］イベントをダブルクリックし，［**表示**］ツールバー・ボタンをクリックします。



3［テスト結果］ウィンドウを閉じます。

ビットマップを閉じて，［**ファイル**］＞［**終了**］を選択し，［WinRunner テスト結果］ウィンドウを閉じます。

新しいバージョンでのテスト実行

フライト予約アプリケーションの新しいバージョンでテストを実行する準備ができました。

1 Flight Reservation 4A を閉じます。

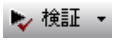
［**ファイル**］＞［**終了**］を選択します。

2 Flight Reservation 4B を閉じます。

［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**Sample Applications**］＞［**Flight 4B**］を選択します。［ログイン］ウィンドウで，名前とパスワード「mercury」を入力し，［OK］ボタンを押します。フライト予約 アプリケーションと WinRunner の全体が見えるように，両方をデスクトップ上で配置し直します。

3 lesson6 がアクティブなテストであることを確認します。

［**lesson6**］テスト・ウィンドウをクリックします。



4 テスト・ツールバーで検証モードが選択されていることを確認します。

5［先頭から実行］を選択します。

［**実行**］＞［**先頭から実行**］を選択するか，［**先頭から実行**］ボタンをクリックする。［**テスト実行**］ダイアログ・ボックスが開きます。標準の名前「**res1**」を使います。［**実行終了時にテスト結果を表示**］チェック・ボックスが選択されていることを確認します。

6 テストを実行します。

［**OK**］をクリックします。WinRunner は，テストの実行を開始します。

ビットマップ・チェックポイントで不一致が検出された場合は，メッセージ・ウィンドウで［**続行**］をクリックしてください。

7 結果を確認します。

テスト実行が完了すると，テスト結果が［WinRunner テスト結果］ウィンドウに現れます。

WinRunner が［**署名をクリア**］ボタンを押しても［代理店署名］フィールドがクリアされなかったために，テストが失敗しました。

失敗したビットマップ・チェックポイントをダブルクリックして，期待結果，実際の結果，およびビットマップの違いを表示します。

8［テスト結果］ウィンドウを閉じます。

［**ファイル**］＞［**終了**］を選択して，［WinRunner テスト結果］ウィンドウを閉じます。

9 **lesson6 のテストを閉じます。**

［**ファイル**］＞［**閉じる**］を選択します。

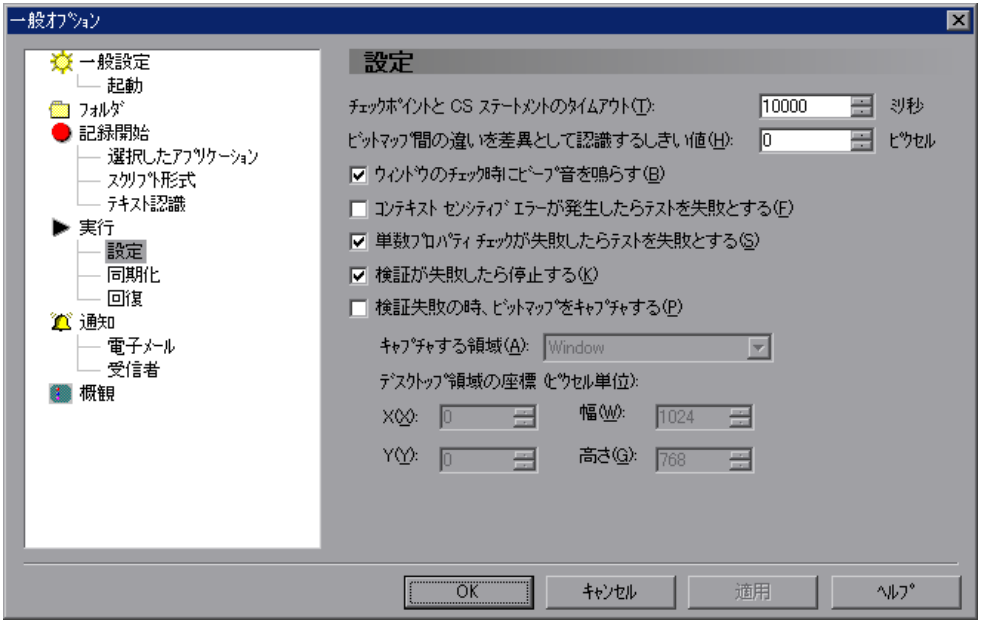
10 **フライト予約アプリケーションの Flight 4B バージョンを閉じます。**

［**ファイル**］＞［**終了**］を選択します。

ビットマップ・チェックポイントに関するヒント

- 領域をキャプチャするには，［**挿入**］＞［**ビットマップ チェックポイント**］＞［**画面領域**］を選択するか，ユーザ定義ツールバーの［**選択範囲のビットマップ チェックポイント**］ボタンをクリックします（このメニュー・コマンドは，テストが開いている場合のみ使用できます）。十字ポインタを使って，WinRunner にキャプチャさせたい領域をマークします。WinRunner は，テスト・スクリプトに win_check_bitmap ステートメントを挿入します。このステートメントには，領域の位置（x 座標と y 座標）とサイズ（幅と高さ）を定義するパラメータが含まれます。
- 夜間に無人テストを実行する場合は，WinRunner に，ビットマップの不一致が検出されてもメッセージを表示しないように指示できます。［**ツール**］＞［**一般オプション**］を選択します。［一般オプション］ダイアログ・ボックスで，［実行］カテゴリの［設定］サブカテゴリを選択します。［**検証が失敗したら停**

止する］チェック・ボックスをクリアします。これにより，テストを中断させることなく実行できます。



- ビットマップ・チェックポイントを含むテストを実行するときには，画面表示の設定がテスト・スクリプトを作成したときのディスプレイ・ドライバの設定と同じであることを確認してください。画面設定が違う場合，WinRunner はビットマップの不一致を報告することがあります。
- ビットマップ・チェックポイントのための新しい期待結果を作成したい場合には，テストを更新モードで実行します。WinRunner は，既存の期待ビットマップを，更新実行中にキャプチャされた新しい期待ビットマップで上書きします。

  ビットマップ・チェックポイントの詳細については，『**WinRunner ユーザーズ・ガイド**』の「ビットマップの検査」の章を参照してください。

練習 7

TSL を使ったテストのプログラミング

この練習では次のことを学びます。

- ビジュアル・プログラミングを使って，記録済みのテスト・スクリプトに関数を追加する方法
- 意思決定ロジックを記録されたテスト・スクリプトに追加する方法
- テスト・スクリプトをデバッグする方法
- テストを新しいバージョンのアプリケーションで実行できるようにし，その結果を分析する方法

TSL を使ってテストをプログラムする方法

テストを記録していると，WinRunner はユーザが GUI オブジェクトをクリックしたり，キーボードで入力したりするたびに，テスト・スクリプトに TSL のステートメントを生成します。記録済みの TSL 関数のほかに，TSL にはテストの性能と柔軟性を高める組み込み関数がたくさんあります。これらの関数は，「**関数ジェネレータ**」という WinRunner のビジュアル・プログラミング・ツールを使って，即座にテスト・スクリプトに追加することができます。［関数ジェネレータ］にある関数はすべて，『**TSL オンライン・リファレンス**』および『**TSL リファレンス・ガイド**』で説明しています。

関数ジェネレータを使って，次の 2 つの方法で TSL 関数を追加できます。

- GUI オブジェクトにポインタをあわせ，WinRunner に適当な関数を「提案」させます。そしてこの関数をテスト・スクリプトに挿入します。
- 関数をリストから選択できます。関数は種類別，アルファベット順の両方で表示されます。

テスト・スクリプトをさらに強化するために，ロジックを追加します。条件ステートメント，ループ，算術演算子などのプログラミング要素をテスト・ウィンドウに直接入力するだけで済みます。

次の練習では，以下のようなテストを作成します。

- 注文を開く
- ［FAX 注文］ダイアログ・ボックスを開く
- 合計が，チケットの注文枚数に単価をかけた数字と等しくなることを確認する
- 合計が正しいか，正しくないかを報告する

基本的なテスト・スクリプトの記録

フライト予約 アプリケーションで注文を開く工程を記録し，［FAX 注文］ダイアログ・ボックスを開いて開始します。

1 WinRunner を起動し，GUI マップをロードします。

WinRunner が開いていない場合は，［**スタート**］メニューで［**プログラム**］＞［**WinRunner**］＞［**WinRunner**］を選択します。［WinRunner へようこそ］ウィンドウが表示されたら，［**テストの新規作成**］ボタンをクリックします。［WinRunner へようこそ］ウィンドウが表示されなければ，［**ファイル**］＞［**新規作成**］をクリックします。新規テストのウィンドウが開きます。

［グローバルな GUI マップ ファイル］モードで作業しているときは，GUI マップがロードされていることを確認してください。これを行うには，［**ツール**］＞［**GUI マップ エディタ**］を選択します。GUI マップ・エディタから［**表示**］＞［**GUI ファイル**］を選択し，**flight4a.GUI** が［**GUI ファイル**］リストに含まれていることを確認してください。

2 フライト予約 アプリケーションを起動し，ログインします。

［**スタート**］メニューで［**プログラム**］＞［**WinRunner**］＞［**Sample Applications**］＞［**Flight 4A**］を選択します。［ログイン］ウィンドウで，名前とパスワード「**mercury**」を入力し，［**OK**］ボタンを押します。フライト予約 アプリケーションと WinRunner の全体が見えるように，両方をデスクトップ上で配置し直します。

3 コンテキスト・センシティブ・モードで記録を開始します。

［**テスト**］＞［**記録 - コンテキストセンシティブ**］を選択するか，ツールバーの［**クリックしてコンテキストセンシティブモードで記録**］ボタンをクリックします。

4 注文番号 3 を開きます。

フライト予約アプリケーションで，［**ファイル**］＞［**注文を開く**］を選択します。［注文を開く］ダイアログ・ボックスで［**注文番号**］チェック・ボックスを選択し，隣のボックスに「3」と入力します。［**OK**］をクリックして，注文を開きます。

5 ［FAX 注文］ダイアログ・ボックスを開きます。

［**ファイル**］＞［**FAX 注文**］を選択します。

6 ［キャンセル］をクリックして，ダイアログ・ボックスを閉じます。

7 記録を停止します。

［**テスト**］＞［**記録停止**］を選択するか，［**停止**］ボタンをクリックします。

8 テストを保存します。

［**ファイル**］＞［**上書き保存**］選択するか，［**保存**］ボタンをクリックします。ハードディスク上の任意の場所に「**lesson7**」という名前を付けてテストを保存します。

9 「グローバルな GUI マップ」モードで作業している場合は，ローカルの GUI マップから flight4a GUI マップに新規オブジェクトを追加します。

GUI マップに既に存在するウィンドウからオブジェクトを保存するには，［**ツール**］＞［**GUI マップ エディタ**］を選択します。［**表示**］＞［**GUI ファイル**］を選択します。新しいオブジェクトが **L0 <一時バッファ>** GUI マップ・ファイルに表示されます。(表示されない場合は，［**表示**］＞［**オブジェクト ツリーの展開**］を選択します)。［**ファイル**］＞［**上書き保存**］を選択します。既存ウィンドウから新しいオブジェクトがこのウィンドウを含む **flight4a.GUI** マップに追加されたことを示す WinRunner メッセージが表示されます。［**はい**］をクリックします。

10 修正された flight4a.GUI マップを保存します。

GUI ファイル・ボックスで，「**L1 flight4a.GUI**」を選択します。GUI マップ名の横にアスタリスクが付きます。これは，ファイルが変更され，保存されていないことを意味します。［**ファイル**］＞［**上書き保存**］を選択して，GUI マップを保存します。

11 ［GUI マップ エディタ］を閉じます。

［**ファイル**］＞［**終了**］を選択し，GUI マップ・エディタを終了します。

新規ウィンドウと新規オブジェクトの保存の詳細については，29 ページの手順 12 と 42 ページの手順 7 を参照してください。

関数ジェネレータによる関数の挿入

さて，［FAX 注文］ダイアログ・ボックスで［**チケット枚数**］，［**単価**］，［**合計**］のフィールドを問い合わせるテスト・スクリプトに，関数を追加する準備ができました。

1 button_press(" キャンセル "); ステートメントの上に空白行を挿入し，この行の先頭にカーソルを合わせます。

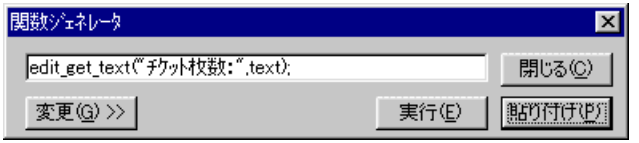
2 ［FAX 注文］ダイアログ・ボックスを開きます。

フライト予約アプリケーションで［**ファイル**］＞［**FAX 注文**］を選択します。

3 ［チケット枚数］フィールドを問い合わせます。

［**挿入**］＞［**関数**］＞［**オブジェクト / ウィンドウ**］を選択するか，ユーザ定義ツールバーの［**オブジェクト / ウィンドウの関数を挿入**］ボタンをクリックします。ポインタを使って，［**チケット枚数**］ボックスをクリックします。

［関数ジェネレータ］が開き，**edit_get_text** 関数を提案します。



この関数は，［**チケット枚数**］フィールドのテキストを読み取り，これを変数に割り当てます。標準の変数名は text です。ボックスに入力して，変数名 **text** を **tickets** に変更します。

edit_get_text("# チケット枚数 :",tickets);

［**貼り付け**］をクリックして，関数をテスト・スクリプトに追加します。

4 ［単価］ボックスを問い合わせます。

［**挿入**］＞［**関数**］＞［**オブジェクト / ウィンドウ**］を選択するか，ユーザ定義ツールバーの［**オブジェクト / ウィンドウの関数を挿入**］ボタンをクリックします。ポインタを使って［**単価**］フィールドをクリックします。

［関数ジェネレータ］が開き，**edit_get_text** 関数を提案します。変数名 **text** を，**price** に変更します。

```
edit_get_text(" 単価 :",price);
```

［**貼り付け**］をクリックして，関数をテスト・スクリプトに追加します。

5 ［合計］フィールドを問い合わせます。

［**挿入**］＞［**関数**］＞［**オブジェクト / ウィンドウ**］を選択するか，ユーザ定義ツールバーの［**オブジェクト / ウィンドウの関数を挿入**］ボタンをクリックします。ポインタを使って，［**合計**］ボックスでクリックします。

［関数ジェネレータ］が開き，**edit_get_text** 関数を提案します。変数名 **text** を，**total** に変更します。

```
edit_get_text(" 合計 :",total);
```

［**貼り付け**］をクリックして，関数をテスト・スクリプトに追加します。

6 ［FAX 注文］ダイアログ・ボックスを閉じます。

［**キャンセル**］をクリックして，フライト予約アプリケーションのダイアログ・ボックスを閉じます。

7 テストを保存します。

［**ファイル**］＞［**上書き保存**］選択するか，［**保存**］ボタンをクリックします。

8 「グローバルな GUI マップ」モードで作業している場合は，ローカルの GUI マップから flight4a GUI マップに新規オブジェクトを追加します。

GUI マップに既に存在するウィンドウから新しいオブジェクトを追加するには，［**ツール**］＞［**GUI マップ エディタ**］を選択します。［**表示**］＞［**GUI ファイル**］を選択します。新しいオブジェクトが **L0 ＜一時バッファ＞** GUI マップ・ファイルに表示されます。(表示されない場合は，［**表示**］＞［**オブジェクト ツリーの展開**］を選択します)。［**ファイル**］＞［**上書き保存**］を選択します。既存ウィンドウから新しいオブジェクトがこのウィンドウを含む **flight4a.GUI** マップに追加されたことを示す WinRunner メッセージが表示されます。［**はい**］をクリックします。

9 修正された flight4a.GUI マップを保存します。

GUI ファイル・ボックスで，「**L1 flight4a.GUI**」を選択します。GUI マップ名の横にアスタリスクが付きます。これは，ファイルが変更され，保存されていないことを意味します。［**ファイル**］＞［**上書き保存**］を選択して，GUI マップを保存します。

10 ［GUI マップ エディタ］を閉じます。

［**ファイル**］＞［**終了**］を選択し，GUI マップ・エディタを終了します。

新規ウィンドウと新規オブジェクトの保存の詳細については，29 ページの手順 12 と 42 ページの手順 7 を参照してください。

テスト・スクリプトへのロジックの追加

この練習では，if/else ステートメントを使って，意思決定ロジックをテスト・スクリプトにプログラムします。こうすることで，テストで以下のことが可能になります。

- 合計が，チケットの注文枚数に単価をかけたものと等しくなることを確認します。
- 合計が正しいか，正しくないかを報告します。

1 カーソルを lesson7 のスクリプトの最後にある edit_get_text ステートメントの下に合わせます。

2 次のステートメントと全く同じものを，テスト・スクリプトに追加します。

2 行目と 4 行目の最初の部分に挿入されているタブやスペースは省略可能です。

```
if (tickets*price == total)
                tl_step ("total", 0, "Total is correct.");
else
                tl_step ("total", 1, "Total is incorrect.");
```

これらのステートメントを普通の言葉で言えば，「チケット（**tickets**）に単価（**price**）をかけたものが合計（**total**）と等しければ，合計が正しいと報告しなさい。そうでない場合は，合計が正しくないと報告しなさい。」という意味です。**tl_step** 関数の詳細については，77 ページ [tl_step について」を参照してください。

3 スクリプトのこのセクションの動作を説明するコメントを追加します。

前のステップで追加した if ステートメントの上に空白行を挿入し，この行の先頭にカーソルを合わせます。［**編集**］＞［**コメント**］を選択します。「**#**」記号の後に，「**チケットの合計金額が正しく計算されたことを確認する**」と入力します。

4 テストを保存します。

［**ファイル**］＞［**上書き保存**］選択するか，［**保存**］ボタンをクリックします。

新規ウィンドウと新規オブジェクトの保存の詳細については，29 ページの手順 12 と 42 ページの手順 7 を参照してください。

参考：

関数ジェネレータを使用して，**tl_step** ステートメントを素早くテスト・スクリプトに挿入します。［**挿入**］＞［**関数**］＞［**関数ジェネレータから**］を選択するか，ユーザ・ツールバーで［**関数ジェネレータから関数を挿入**］を選択します。

tl_step について

多くの場合テストを実行すると，WinRunner は成功か失敗かというテスト全体の結果をレポートします。**tl_step** ステートメントをテスト・スクリプトに追加すると，そのテストのある特定の処理が成功したのか，失敗したのかを判断して，メッセージをレポートに送信することができます。

次に例を示します。

```
tl_step ("total", 1, "Total is incorrect.");
```

total は，この処理に割り当てる名前です。

1 の場合，WinRunner は処理が失敗したと報告します。**0** の場合，WinRunner は処理が成功したと報告します。

「**Total is incorrect**」というのがレポートに送信されるメッセージです。テストの結果として意味をなす，任意のメッセージを記述することができます。

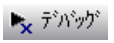
tl_step 関数の詳細については，WinRunner で「**TSL オンライン・リファレンス**」を参照してください。

テスト・スクリプトのデバッグ

テストをプログラミング要素で強化したら，テストをスムーズに，構文やロジックに誤りがなく実行できるかどうかを確認する必要があります。WinRunner では，この手順を素早くて簡単なものにするためにデバッグ・ツールを提供します。

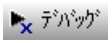
次のことができます。

- テストを［ステップ］コマンドを使って，1 行ごと実行できます。
- テスト・スクリプトの指定された行あるいは関数で，実行を停止できるブレークポイントを定義できます。
- ［ウォッチ リスト］を使って，変数と式の値を監視します。



テスト・スクリプトをデバッグするときには，テストを［デバッグ］モードで実行する必要があります（テストを［デバッグ］モードで実行するには，標準ツールバーの［実行モード］リストから［デバッグ］を選択します）。テスト結果は，**debug** ディレクトリに保存されます。［デバッグ］モードでテストを実行するたびに，WinRunner は前回のデバッグの結果を上書きします。

この練習では，［ステップ］コマンドを使ってテストの実行をコントロールします。エラー・メッセージが表示されたら，テスト・スクリプトを検証し，問題を修正します。



1 ［デバッグ］モードを，標準ツールバーの実行モード・リストから選択します。

［デバッグ］モードは，別のモードを選択するまで有効です。

2 実行マーカ「→」を，テスト・スクリプトの最初の行の横に配置します。

テスト・スクリプトの最初の行の脇にある左側のマージン内でクリックします。

3 ［デバッグ］＞［ステップ］を選択するか，［ステップ実行］ボタンをクリックして，テスト・スクリプトの最初の行を実行します。

WinRunner はテストの最初の行を実行します。

4 テスト全体を，［ステップ実行］ボタンを使って 1 行ずつ実行します。

［**ステップ実行**］ボタンをクリックして，テスト・スクリプトの各行を実行します。テストの実行中，オブジェクトをクリックするときに，マウス・ポインタがフライト・アプリケーションに移動することがあります。

5 ［停止］をクリックします。

［**停止**］ボタンをクリックして，WinRunner に［デバッグ］テスト実行が終了したことを通知します。

6 ［WinRunner テスト結果］ウィンドウでテスト結果を検討します。

テストを［デバッグ］モードで実行すると，テスト結果は自動的に開きません。［**ツール**］＞［**テスト結果**］を選択するか，［**テスト結果**］ボタンをクリックします。［WinRunner テスト結果］ウィンドウは［デバッグ］テスト実行の結果を表示します。

7 ［テスト結果］ウィンドウを閉じます。

［**ファイル**］＞［**終了**］を選択します。

8 ［フライト予約］アプリケーションを終了します。

［**ファイル**］＞［**終了**］を選択します。

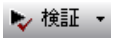
テスト・スクリプトのデバッグについての詳細は，『**WinRunner ユーザーズ・ガイド**』の第 6 部「テストのデバッグ」を参照してください。

新しいバージョンでのテスト実行

テスト・スクリプトをデバッグしたら，新しいバージョンのフライト予約 アプリケーションでこれを実行できます。

1 フライト予約アプリケーションの Flight 4B バージョンを開きます。

［**スタート**］メニューで［**プログラム**］＞［**WinRunner**］＞［**Sample Applications**］＞［**Flight 4B**］を選択します。［ログイン］ウィンドウで，名前とパスワード「mercury」を入力し，［**OK**］ボタンを押します。フライト予約アプリケーションと WinRunner の全体が見えるように，両方をデスクトップ上で配置し直します。



2 標準ツールバーの［実行モード］リストから［検証］を選択します。

検証モードは，別のモードを選択するまで有効です。

3 ［先頭から実行］を選択します。

［**テスト**］＞［**先頭から実行**］を選択するか，［**先頭から実行**］ボタンをクリックします。［テスト実行］ダイアログ・ボックスが開きます。標準の名前「**res1**」を使います。［**実行終了時にテスト結果を表示**］チェック・ボックスが選択されていることを確認します。

4 テストを実行します。

［テスト実行］ダイアログ・ボックスで，［**OK**］をクリックします。WinRunner は，テストの実行を開始します。

5 結果を見ます。

テスト実行が完了すると，テスト結果が［WinRunner テスト結果］ウィンドウに現れます。

チケット枚数を単価でかけたものが合計と等しくなりました。このため，**tl_step** ステートメントは，「**成功**」を報告します。

テスト・ログで **tl_step** ステートメントをダブルクリックし，詳細を表示します。

「**Total is correct**」というメッセージは，テスト・スクリプトに書いたメッセージと同じものです。［**OK**］をクリックして，メッセージを閉じます。

6 結果を閉じます。

［**ファイル**］＞［**終了**］を選択して，［WinRunner テスト結果］ウィンドウを閉じます。

7 lesson7 のテストを閉じます。

［**ファイル**］＞［**閉じる**］を選択します。

8 フライト予約アプリケーションの Flight 4B バージョンを閉じます。

［**ファイル**］＞［**終了**］を選択します。

練習 8

データ駆動型テストの作成

この練習では次のことを学びます。

- データ駆動テスト・ウィザードを使って，データ駆動型テストを作成する方法
- テストをデータ駆動型テストに変換し，テストにデータを追加する方法
- テストの反復ごとに異なる GUI オブジェクト名に対して，正規表現を使う方法
- 要件に合わせて結果情報をカスタマイズする方法
- 複数回テストを反復し，結果を分析する方法
- データ駆動型テストの作成のヒント

データ駆動型テストの作成方法

テストを正しくデバックし，実行できたら，複数のデータ・セットを使って同じテストを実行した場合の結果を確認する必要があるかもしれません。この作業を行うには，テストをデータ駆動型テストに変換し，テストするデータ・セットを使って，対応するデータ・テーブルを作成します。

次の手順で，テストをデータ駆動型テストに変換します。

- データ・テーブルを開いたり閉じたりするステートメントをスクリプトに追加します。
- テストがデータ・テーブルからデータを読み込んで，各データ・セットを適用する間ループするように，テストにステートメントと関数を追加します。
- 記録されているステートメントおよびチェックポイント・ステートメントに含まれる固定値を，パラメータに置換します。これを，テストを「**パラメータ化する**」といいます。

データ駆動テスト・ウィザードを使うか，スクリプトを手作業で変更することによって，テストをデータ駆動型テストに変換できます。

データ駆動型テストを実行すると，WinRunner はテスト内のパラメータ化されている部分を，データ・テーブル内の各データ・セットに対して 1 回実行します（これを 1 回の**反復**と呼びます）。次に，WinRunner は 1 つの［WinRunner テスト結果］ウィンドウにすべての反復の結果を表示します。

練習 7 では，予約したチケットの合計額が正しいことを確認するために，特定のフライトの注文を開き，［FAX 注文］ダイアログ・ボックスからチケットの枚数，単価，合計額を読み取るテストを作成しました。練習 8 では，チケットの枚数や単価が変わっても，アプリケーションが合計額を正しく計算できることを確認するために，複数のフライトの注文に対して同じ検査を行うテストを作成します。

データ駆動型テストへのテストの変換

まず，練習 7 で作成したテストを開き，データ駆動テスト・ウィザードを使ってテストをパラメータ化します。

1 lesson7 テストから，GUI マップをロードます。

WinRunner がまだ開いていない場合は，［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**WinRunner**］を選択します。［WinRunner へようこそ］ウィンドウが表示されたら，［**既存のテストを開く**］ボタンをクリックします。または，［**ファイル**］＞［**開く**］を選択して，練習 7 で作成したテストを選択します。**lesson7** テストが開きます。

［**ファイル**］＞［**テストとして保存**］を選択して，ハードディスク上の任意の場所に「**lesson8**」という名前でテストを保存します。

［グローバルな GUI マップ ファイル］モードで作業しているときは，GUI マップがロードされていることを確認してください。これを行うには，［**ツール**］＞［**GUI マップ エディタ**］を選択します。GUI マップ・エディタから［**表示**］＞［**GUI ファイル**］を選択し，**flight4a.GUI** が［**GUI ファイル**］リストに含まれていることを確認してください。

2 データ駆動型ウィザードを実行します。

［**テーブル**］＞［**データ駆動テストウィザード**］を選択します。データ駆動テスト・ウィザードの最初のウィンドウが開きます。［**次へ**］をクリックして，パラメータ化の工程を開始します。

3 テストで使用するデータ・テーブルを作成します。

［**新規 / 既存の Excel テーブルを使用**］ボックスに「lesson8」と入力します。データ駆動テスト・ウィザードは，この名前で Microsoft Excel のテーブルを作成し，それをテスト・フォルダに保存します。

4 テーブルの変数名を割り当てます。

テーブルの標準の変数名「**table**」を使用します。

データ駆動型テストのはじめに，使用する Microsoft Excel データ・テーブルをテーブルの変数値として割り当てます。スクリプトでは，テーブルの変数名だけを使います。これによって，スクリプトに変更を加えずに，後で簡単に別のデータ・テーブルをスクリプトに割り当てることができます。

5 グローバルに適用されるパラメータ化オプションを選択します。

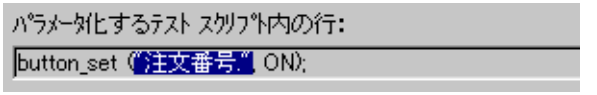
［**ステートメントを追加し，データ駆動テストを作成**］を選択します。これによって，テーブルの変数名を定義したり，データ・テーブルを開閉したり，データ・テーブル内の各行に対してループで適切なスクリプトを選択したりする TSL ステートメントが，テストに追加されます。

［**テストをパラメータ化**］を選び，［**行ごと**］オプションを選択します。［テストをパラメータ化］を選択した場合，WinRunner は，記録されているステートメントと選択されているチェックポイントに含まれる固定値を検出し，それらの固定値をパラメータに置換します。［行ごと］オプションを選択すると，ウィザードは，選択されているテストの中でパラメータ化が可能な各行の画面を開くので，その行をパラメータ化するかどうかを選択できます。

［**次へ**］をクリックします。

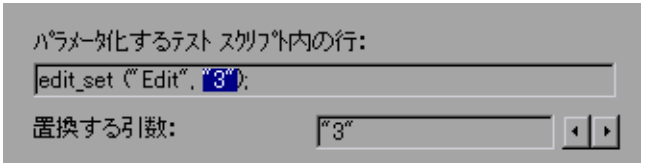
6 パラメータ化するデータを選択します。

各行を示す最初の画面が開きます。この画面は，［**注文番号**］ラジオ・ボタンを示しています。



このテストでは，各反復で異なる FAX 注文を開き，［**注文番号**］ラジオ・ボタンを毎回選択する必要があります。したがって，このスクリプト行に対しては，［**このデータは置換しない**］を選択し，［**次へ**］をクリックします。

各行を示す次の画面は，［**注文番号**］ボックスを示します。これは，各反復で変更したいフィールドです。値「**3**」が強調表示されます。［**置換する引数**］ボックスにも値「3」が表示され，パラメータ化の対象として選択された値であることが示されます。



［**選択された値と置換するデータの取得先**］の下にある［**新規の列**］を選択し，横のボックスに「Order_Num」と入力します。これによって，**lesson8.xls**

テーブルに「**Order_Num**」カラムが作成され，カラムの最初の行に値「**3**」が入力されます。

［**次へ**］をクリックし，次に［**終了**］をクリックします。テストがパラメータ化されます。

参考：

パラメータ化されたテストでは，次の要素が追加または変更されます。

table = 行は，テーブルの変数を定義します。

ddt_open ステートメントはテーブルを開きます。以降の行は，データ駆動型テストが正常に開くことを確認します。

ddt_get_row_count ステートメントは，テーブル内の行数を検査します。つまり，実行するテストの中でパラメータ化されているセクションの反復回数を検査します。

for ステートメントは反復のループを設定します。

ddt_set_row ステートメントは，各反復で使うテーブルの行をテストに指示します。

edit_set ステートメントでは，値「**3**」が **ddt_val** ステートメントに置換されます。

ddt_close ステートメントはテーブルを閉じます。

データ・テーブルへのデータの追加

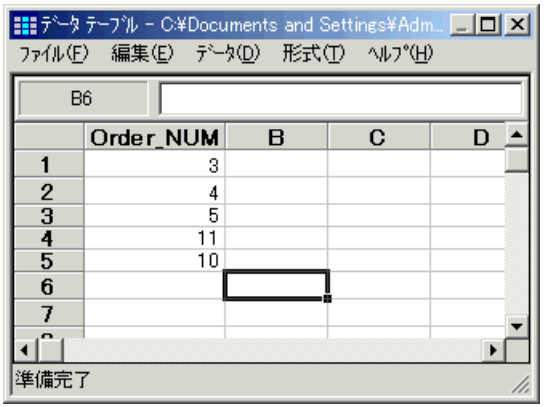
テストのパラメータ化が済んだので，パラメータ化されたテストで使うデータを追加できます。

1 **データ・テーブルを開きます。**

［**テーブル**］＞［**データ テーブル**］を選択します。**lesson8.xls** テーブルが開きます。「**Order_Num**」というカラムがあり，そのカラムの最初の行には，値「**3**」が含まれています。

2 テーブルにデータを追加します。

Order_Num カラムの行 **2**，**3**，**4**，および **5** に，それぞれ 1，6，8，10 の値を入力します。



3 テーブルを保存し，閉じます。

空のセルをクリックし，データ・テーブルのメニューから［**ファイル**］＞［**上書き保存**］を選択します。次に［**ファイル**］＞［**閉じる**］を選択して，テーブルを閉じます。

4 テストを保存します。

［**ファイル**］＞［**上書き保存**］選択するか，［**保存**］ボタンをクリックします。

正規表現によるスクリプトの変更

以上で，テストはほぼ完成しました。テストを実行する前に，テスト全体に目を通して，データ駆動型テストで競合が生じる可能性のある要素があるかどうかを確認します。データ駆動テスト・ウィザードは，選択されているチェックポイントおよび記録されているステートメントに含まれるすべての固定値を検出しますが，外部入力によっても変更される可能性のあるオブジェクト・ラベルなどの項目は検査しません。

フライト予約アプリケーションでは，［FAX 注文］ウィンドウの名前が，FAX 注文番号を反映したものに変更されます。テストをそのまま実行した場合，フライト予約アプリケーションが「FAX 注文番号 1」というラベルの付いたウィンドウを開きますが，スクリプトは「FAX 注文番号 3」というラベルの付いた

ウィンドウをアクティブにするように指示するので，テストは 2 回目の反復で失敗します。WinRunner は「FAX 注文番号 4」というラベルの付いたウィンドウを検出できません。

この問題を解決するには，正規表現を使います。正規表現は，WinRunner が様々な名前またはタイトルのオブジェクトを識別できるように，複雑な検索語句を指定する文字列です。

この練習では，WinRunner が［FAX 注文］ウィンドウのラベルの変化を無視できるように，物理的記述で正規表現を使います。

1 flight1a.gui GUI マップ・ファイルで［FAX 注文］ウィンドウを見つけます。

［**ツール**］＞［**GUI マップ エディタ**］を選択します。［**表示**］＞［**GUI ファイル**］を選択します。［**GUI ファイル**］ボックスから **flight4a.GUI** を選択します。［FAX 注文番号 3］ウィンドウ・アイコンを選択します。

2 ウィンドウのラベルを正規表現に変更します。

［**物理的記述**］ボックスで，**label** プロパティ行のダブルクォーテーションの直後に「**!**」を追加し，これが正規表現であることを示します。行末にあるピリオド，スペース，数値「**3**」を削除し，後に続くテキストが変化することを示す「*****」を代わりに入力します。

3［修正］ダイアログ・ボックスを閉じます。

［**OK**］をクリックして［修正］ウィンドウを閉じます。

4 **GUI マップを保存し（[グローバルな GUI マップ ファイル］モードで操作している場合のみ），［GUI マップ エディタ］を閉じます。**

［グローバルな GUI マップ ファイル］モードで操作している場合は，［**ファイル**］＞［**上書き保存**］を選択して変更を保存してから，［**ファイル**］＞［**閉じる**］を選択して［GUI マップ エディタ］を閉じます。

［**テスト特有の GUI マップ ファイル**］モードで操作している場合は，［**ファイル**］＞［**閉じる**］を選択して，［GUI マップ エディタ］を終了します。

結果の情報のカスタマイズ

以上でテストを実行する準備が整いましたが，テストを実行した後に，各反復の結果を解釈するのは難しいかもしれません。各反復に固有の情報をスクリプトのレポート・ステートメントに追加すれば，各結果の根拠となるデータを確認できます。

1 **tl_step ステートメントを変更します。**

スクリプト内の最初の **tl_step** ステートメントを見つけます。「 "total is correct"." 」という語句を削除し，「"Correct. "tickets" tickets at $"price" cost $"total"."」に置き換えます。

```
tl_step("total",0, "Correct. "tickets" tickets at $"price" cost $"total".");
```

同じ方法で，間違った結果をレポートする次の **tl_step** ステートメントを変更します。次に例を示します。

```
tl_step("total", 1, "Error! "tickets" tickets at $"price" does not equal $"total". ");
```

これで，結果を見たときに，各反復で使われているデータがわかるようになります。

2 **テストを保存します。**

［**ファイル**］＞［**上書き保存**］選択するか，［**保存**］ボタンをクリックします。

テストの実行と結果の分析

データ駆動型テストは，WinRunner でその他のテストと同じように実行します。テストの実行が完了したら，すべての反復の結果が 1 つの［テスト結果］ウィンドウに示されます。

1 フライト 4A のフライト予約アプリケーションがデスクトップに表示されていることを確認します。

2 WinRunner のテスト・ツールバーで，［検証］モードが選択されていることを確認します。

3 ［先頭から実行］を選択します。

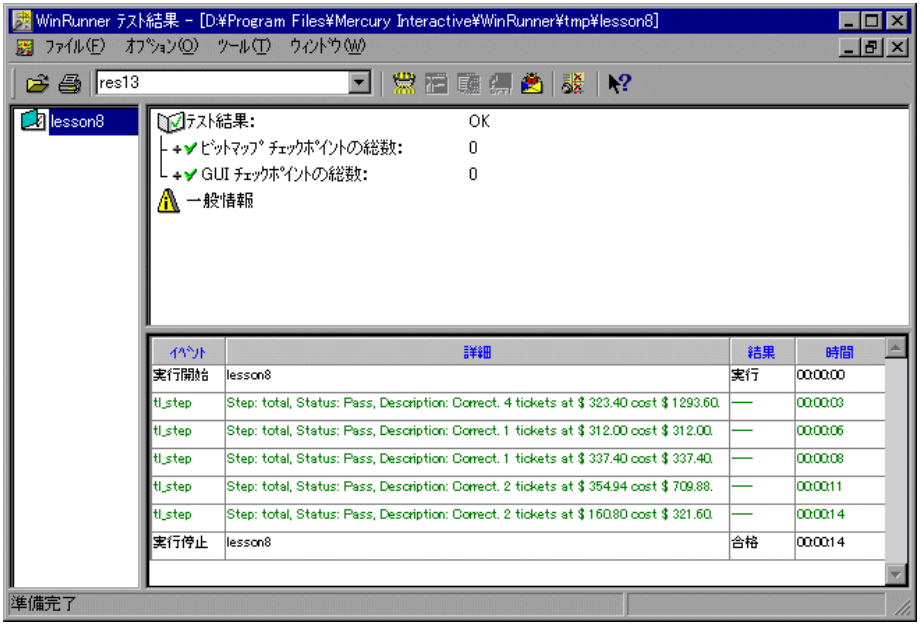
［**実行**］＞［**先頭から実行**］を選択するか，［**先頭から実行**］ボタンをクリックします。［**テスト実行**］ダイアログ・ボックスが開きます。標準のテスト実行名「res1」を適用します。［**実行終了時にテスト結果を表示**］チェック・ボックスが選択されていることを確認します。

4 テストを実行します。

［**テストの実行**］ダイアログ・ボックス［**OK**］ボタンをクリックします。テストでは，スクリプト内のパラメータ化されているセクションを，データ・テーブル内の各行に対して 1 回ずつ，全部で 5 回実行します。

5 結果を確認します。

テストの実行が完了すると，テスト結果が［WinRunner テスト結果］ウィンドウに表示されます。



tl_step イベントが 5 回分表示されます。各反復の詳細には，実際のチケットの枚数，単価，確認された合計額が表示されます。

6 結果を閉じます。

［**ファイル**］＞［**終了**］を選択し，［**テスト結果**］ウィンドウを閉じます。

7 フライト予約アプリケーションを閉じます。

［**ファイル**］＞［**終了**］を選択します。

8 lesson8 のテストを閉じます。

［**ファイル**］＞［**閉じる**］を選択します。

データ駆動型テストのヒント

- テスト・スクリプトの一部，またはテスト・スクリプト内のループのみをパラメータ化できます。また，1 つのデータ駆動型テストには，パラメータ化したループを複数含められます。
- **default.xls** データ・テーブル以外のデータ・テーブルを開いたり，保存したりできます。したがって，1 つのテスト・スクリプトで複数の異なるデータ・テーブルを使用できます。
- GUI チェックポイント，ビットマップ・チェックポイント，ビットマップ同期ポイントを含むステートメント，および定数をパラメータ化できます。
- データ・テーブルは，セルへの式の挿入など，Excel のスプレッドシートと同じ方法で使用できます。
- データ駆動型テストを実行する前に，スクリプト全体に目を通して，テストで競合が生じる可能性のある要素がないかどうかを確認します。このような競合のほとんどは，次の 2 つの方法で解決できます。
  - 正規表現を使って，WinRunner が物理的記述の一部に基づいてオブジェクトを認識できるようにします。正規表現の詳細については，『**WinRunner ユーザーズ・ガイド**』の「正規表現の使い方」の章を参照してください。
  - ［GUI マップの構成設定］ダイアログ・ボックスを使って，WinRunner が問題の起こりそうなオブジェクトを認識するために使う物理的なプロパティを変更します。GUI マップの構成設定の詳細については，『**WinRunner ユーザーズ・ガイド**』の「GUI マップの構成設定」の章を参照してください。
- TSL ステートメントを使って，テストの実行中にアクティブな行を変更することや，非アクティブな行からデータを読み取ることができます。詳細については，『**WinRunner ユーザーズ・ガイド**』の「データ駆動型テストでの TSL 関数の使用」の章を参照してください。
- テストを実行するときにデータ・テーブル・ビューアを開く必要がありません。

データ駆動型テストの詳細については，『**WinRunner ユーザーズ・ガイド**』の「データ駆動型テストの作成」の章を参照してください。

練習 9

バッチ・テストの作成

この練習では次のことを学びます。

- バッチ・テストを使って，テスト・セットを無人で実行する方法
- バッチ・テストを作成する方法
- バッチ・テストを実行し，結果を分析する方法

バッチ・テストとは？

例えば，アプリケーション製品を改訂した後で，その製品を対象とした以前のテスト・スクリプトを実行したいとします。その場合，各テストを個別に実行する代わりにバッチ・テストを使って，任意の数のテストを実行することができます。昼食に出掛けても戻ったときには，すべてのテストの結果を画面で見ることができます。

バッチ・テストは通常のテスト・スクリプトのように見え，そのように動作しますが，次の 2 つの主な違いがあります。

- 他のテストを開く **call** ステートメントを含みます。次に例を示します。

```
call "c:¥¥qa¥¥flights¥¥lesson9"();
```

テストの実行中，WinRunner は call ステートメントを解釈し，「呼び出し先」のテストを開いて，実行します。呼び出し先のテストが完了すると，WinRunner はバッチ・テストに戻り，実行を継続します。

- ［一般オプション］ダイアログ・ボックス（［**ツール**］＞［**一般オプション**］）の［実行］タブで，［**バッチ モードで実行**］オプションを選択してから，テストを実行します。このオプションは WinRunner に，テストを中断するメッセージを抑制するよう指示します。例えば，WinRunner がビットマップの不一致を検出しても，テスト実行を休止するかどうか尋ねてこなくなります。

バッチ・テスト実行の結果を見直すときに，バッチ・テスト全体の結果（成功または失敗）を見ることができます。また，そのバッチ・テストに呼び出された各テストの結果を見ることもできます。

バッチ・テストのプログラミング

この練習では，次のバッチ・テストを作成します

- これまでの練習（**lesson5**，**lesson6**，**lesson7**）で作成したテストを呼び出すバッチ・テスト
- フライト予約アプリケーションが，反復実行の「ストレス」をどのように扱うかを検査するために，それぞれの呼び出し先テストを 3 回ずつ実行するバッチ・テスト

1 WinRunner を起動し，GUI マップをロードします。

WinRunner がまだ開いていない場合は，［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**WinRunner**］を選択します。［WinRunner へようこそ］ウィンドウが表示されたら，［**テストの新規作成**］ボタンをクリックします。［WinRunner へようこそ］ウィンドウが表示されなければ，［**ファイル**］＞［**新規作成**］をクリックします。新規テストのウィンドウが開きます。

［グローバルな GUI マップ ファイル］モードで作業しているときは，GUI マップがロードされていることを確認してください。これを行うには，［**ツール**］＞［**GUI マップ エディタ**］を選択します。GUI マップ・エディタから［**表示**］＞［**GUI ファイル**］を選択し，**flight4a.GUI** が［**GUI ファイル**］リストに含まれていることを確認してください。

2 lesson5，lesson6，lesson7 を呼び出すテスト・スクリプトで call ステートメントをプログラムします。

call ステートメントを新規テスト・ウィンドウに入力します。ステートメントは次のようになります。

```
call "c:¥¥qa¥¥flights¥¥lesson5"();
call "c:¥¥qa¥¥flights¥¥lesson6"();
call "c:¥¥qa¥¥flights¥¥lesson7"();
```

テスト・スクリプトでは，c:¥¥qa¥¥flights を実際にテストが格納されているディレクトリ・パスに置き換えます。

注：パスを入力するときは，ディレクトリ名を 2 つのバックスラッシュで区切ります。

---

3 各テストを 3 回呼び出すループを定義します。

call ステートメントの周りにループを追加します。テスト・スクリプトは次のようになります。

```
for (i=0; i<3; i++)
{
call "c:¥¥qa¥¥flights¥¥lesson5"();
call "c:¥¥qa¥¥flights¥¥lesson6"();
call "c:¥¥qa¥¥flights¥¥lesson7"();
}
```

これは普通の言葉で言えば，「lesson5，lesson6，lesson7 を実行したら，折り返して，各テストを再度実行しなさい。この処理を各テストが 3 回実行されるまで反復しなさい」という意味です。中括弧（{ }）は，どのステートメントがループに含まれているかを定義します。

4 ［一般オプション］ダイアログ・ボックスで［バッチ モードで実行］オプションを選択します。

［**ツール**］＞［**一般オプション**］を選択します。［一般オプション］ダイアログ・ボックスで［実行］カテゴリをクリックします。［**バッチモードで実行する**］チェック・ボックスを選択して有効にします。［**OK**］をクリックして［一般オプション］ダイアログ・ボックスを閉じます。

5 バッチ・テストを保存します。

［**ファイル**］＞［**上書き保存**］選択するか，［**保存**］ボタンをクリックします。テストに **batch** という名前を付けます。

Version 4B でのバッチ・テストの実行

フライト予約アプリケーションを検査するためにバッチ・テストを実行する準備ができました。テストを実行すると，WinRunner は各テストの期待結果とアプリケーションの実際の結果とを比較します。これまでの練習でテストを作成したときに保存した期待結果を使います。

1 フライト予約アプリケーションの Flight 4B バージョンを開き，ログインします。

［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**Sample Applications**］＞［**Flight 4B**］を選択します。［**ログイン**］ウィンドウで，名前とパスワードに「mercury」を入力し，［**OK**］をクリックします。フライト予約アプリケーションと WinRunner の全体が見えるように，両方をデスクトップ上で配置し直します。

2 WinRunner のテスト・ツールバーで，［検証］モードが選択されていることを確認します。

3 ［先頭から実行］を選択します。

［**テスト**］＞［**先頭から実行**］を選択するか，［**先頭から実行**］ボタンをクリックします。［テスト実行］ダイアログ・ボックスが開きます。標準の名前「**res1**」を使います。［**実行終了時にテスト結果を表示**］チェック・ボックスが選択されていることを確認します。

4 テストを実行します。

［テスト実行］ダイアログ・ボックスで，［**OK**］をクリックします。WinRunner は，テストの実行を開始します。テストには 9 つの異なるテストの実行が含まれているため，多少時間がかかります。

WinRunner が呼び出し先の各テストを開いて実行し，折り返して，テストを再度実行する様子を観察します（全部で 3 回）。

バッチ・テスト結果の分析

バッチ・テスト実行が完了したら，［WinRunner テスト結果］ウィンドウで結果を分析できます。［WinRunner テスト結果］ウィンドウがバッチ・テスト全体の結果（成功または失敗）と，呼び出し先のテストのそれぞれの結果を表示します。呼び出し先テストのいずれかが失敗したら，バッチ・テストは失敗です。

1 ［WinRunner テスト結果］ウィンドウを開いて，バッチ・テストの **res1** の結果を表示します。

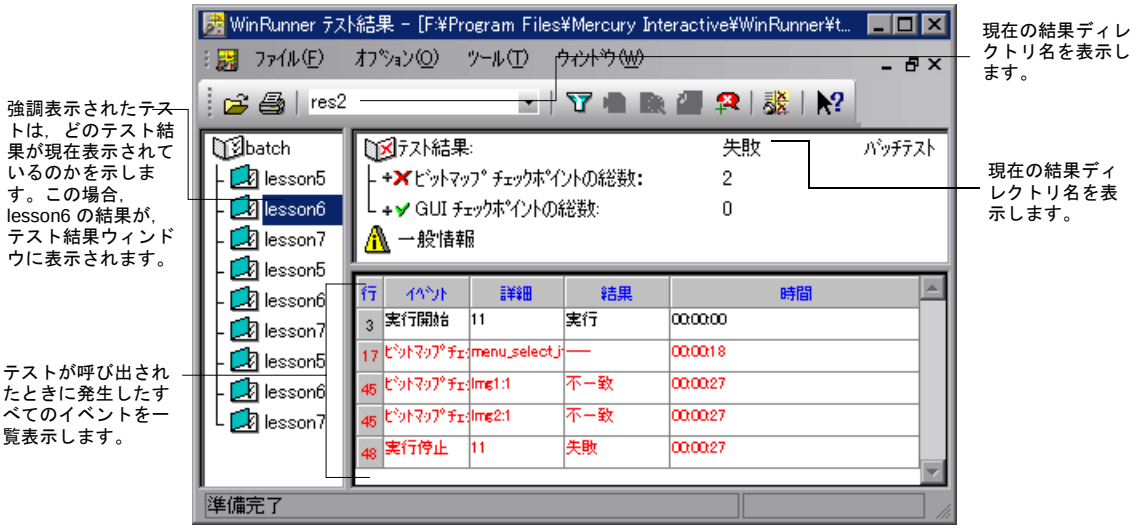
［WinRunner テスト結果］ウィンドウが現在開いていない場合は，バッチ・テスト・ウィンドウをクリックして，［**ツール**］＞［**テスト結果**］を開くか，［**テスト結果**］ボタンをクリックします。

2 バッチ・テストの結果が表示されます。

呼び出し先テストが 1 つまたは複数失敗したため，バッチ・テストは失敗しました。これまでの練習で見てきたように，Flight 4B バージョンには不具合があります。

3 呼び出し先テストの結果を表示します。

呼び出し先のテストの結果を表示するには，テスト・ツリーでテスト名をクリックします。



lesson6 はビットマップ・チェックポイントを使って，WinRunner が［**署名をクリア**］ボタンをクリックした後で，［FAX 注文］ダイアログ・ボックスの［**代理店署名**］ボックスをクリアすることを検査します。［**代理店署名**］ボックスがクリアされないので，ビットマップ・チェックポイントは，不一致を検出しました。失敗したイベントをクリックすると，期待結果，実際の結果，それらの相違点を表示できます。

4 ［テスト結果］ウィンドウを閉じます。

［**ファイル**］＞［**終了**］を選択します。

5 バッチ・テストを閉じます。

［**ファイル**］＞［**閉じる**］を選択して，開いているテストを閉じます。つまり，バッチ・テストとバッチが呼び出した 3 つのテストを閉じます。または，［**ファイル**］＞［**すべて閉じる**］を選択します。

6 ［一般オプション］ダイアログ・ボックスで［バッチ モードで実行する］オプションの選択を解除します。

バッチ・テストの実行が終了したら，［バッチモードで実行］オプションをクリアします。［**ツール**］＞［**一般オプション**］を選択します。［一般オプショ

ン］ダイアログ・ボックスで［実行］カテゴリをクリックします。［**バッチモードで実行**］チェック・ボックスをクリックして解除し，［**OK**］をクリックします。

7 **フライト予約アプリケーションの Flight 4B バージョンを閉じます。**

［**ファイル**］＞［**終了**］を選択します。

バッチ・テストのヒント

➤ 検索パスを定義して，WinRunner に，特定のディレクトリ内で呼び出し先のテストを検索するように指示できます。［**ツール**］＞［**一般オプション**］を選択します。［一般オプション］ダイアログ・ボックスで［フォルダ］カテゴリをクリックします。［**呼び出すテストの検索パス**］ボックスで，テストが格納されているパスを定義します。これで，**call** ステートメントにテスト名を指定するだけで済みます。次に例を示します。

```
call "lesson6"();
```

呼び出し先テストの検索パスの定義についての詳細は，『**WinRunner ユーザーズ・ガイド**』の「グローバル・テスト・オプションの設定」の章を参照してください。

パラメータ値をバッチ・テストから呼び出し先のテストに引き渡せます。パラメータ値は call ステートメントの括弧内に指定します。

```
call test_name ([parameter1, parameter2, ...]);
```

［一般オプション］ダイアログ・ボックスの［実行］カテゴリで，無人でバッチ・テストが実行されるように，［**バッチモードで実行する**］オプションを選択しなければなりません。このオプションを選択しないと，エラー発生時にエラー・メッセージが表示され，テスト実行が停止します。

バッチ・テストの作成の詳細については，『**WinRunner ユーザーズ・ガイド**』の「テストの呼び出し」の章と，「バッチ・テストの実行」の章を参照してください。

練習 10

テスト・スクリプトの保守

この練習では次のことを学びます。

- アプリケーションのユーザ・インタフェースが変更された後，GUI マップを使って既存のテスト・スクリプトを継続して使用できるようにする方法
- 既存のオブジェクト記述の編集方法と新しい記述の GUI マップへの追加方法
- ［実行ウィザード］を使って，GUI マップを自動的に更新する方法

ユーザ・インタフェースが変更されるとどうなるか

次のような場面を思い浮かべてください。数週間を費やして，アプリケーションのすべての機能をカバーする自動テスト・セットを作成しました。アプリケーション開発者は，改良されたユーザ・インタフェースで新しいバージョンを構築します。いくつかのオブジェクトを変更し，新規のオブジェクトを追加し，他のオブジェクトを削除します。この新しいバージョンを，既存のテストを使ってテストするにはどのようにしたらよいでしょうか。

WinRunner は簡単なソリューションを提供します。テスト・スクリプトを 1 つ 1 つ手作業で編集せずに，「**GUI マップ**」を更新することができます。GUI マップには，アプリケーションのオブジェクトの記述が含まれます。

GUI マップのオブジェクト記述は，以下で構成されます。

- 「**論理名**」（オブジェクトを表す短い直感的な名前）。これは，テスト・スクリプトに現れる名前です。次に例を示します。

  button_press (" **注文挿入** ");

  「**注文挿入**」がオブジェクトの論理名です。

- 「**物理的記述**」（オブジェクトを一意に識別するプロパティのリスト）。次に例を示します。

  {
  class:push_button
  label: " **注文挿入** "
  }

  このボタンは，push_button オブジェクト・クラスに属し，「**注文挿入**」というラベルを持っています。

  テストを実行すると，WinRunner はテスト・スクリプトに書いてあるオブジェクト論理名を読み取って，その物理的記述を GUI マップで参照します。その後 WinRunner は，この記述を使って，テスト対象のアプリケーションでオブジェクトを検索します。

  アプリケーションのオブジェクトに変更が加えられた場合は，テスト実行時に WinRunner がオブジェクトを見つけられるように，GUI マップの物理的記述を更新しなくてはなりません。

  次の練習では，以下のことをします。

  - GUI マップのオブジェクト記述を編集する
  - GUI マップにオブジェクトを追加する
  - ［実行ウィザード］を使って，ユーザ・インタフェースの変更を自動的に検出し，GUI マップを更新する

GUI マップでのオブジェクト記述の編集

新しいバージョンのフライト予約アプリケーションで，［**注文挿入**］ボタンが［**挿入**］ボタンに変更されていると想定します。［**注文挿入**］ボタンを使うテストを継続して実行できるようにするには，GUI マップでボタンの物理的記述でラベルを編集しなくてはなりません。物理的記述は，正規表現を使って変更できます。詳細については，このチュートリアルの 89 ページ「正規表現によるスクリプトの変更」および『**WinRunner ユーザーズ・ガイド**』の「正規表現の使い方」の章を参照してください。

1 WinRunner を起動し，テストを開いて GUI マップをロードします。

If WinRunner がまだ開いていない場合は，［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**WinRunner**］を選択します。［WinRunner へようこそ］ウィンドウが表示されたら，［**テストの新規作成**］ボタンをクリックします。［WinRunner へようこそ］ウィンドウが表示されなければ，［**ファイル**］＞［**新規作成**］をクリックします。新規テストのウィンドウが開きます。

［テスト特有の GUI マップ ファイル］モード で操作している場合は，**lesson4** テストを開きます。

［グローバルな GUI マップ ファイル］モードで作業しているときは，GUI マップがロードされていることを確認してください。これを行うには，［**ツール**］＞［**GUI マップ エディタ**］を選択します。GUI マップ・エディタから［**表示**］＞［**GUI ファイル**］を選択し，**flight4a.GUI** が［**GUI ファイル**］リストに含まれていることを確認してください。

2 ［GUI マップ エディタ］を開きます。

［**ツール**］＞［**GUI マップ エディタ**］を選択します。［GUI マップ エディタ］が開きます。［**表示**］＞［**GUI マップ**］を選択します。［ウィンドウ / オブジェクト］リストに，GUI マップの現在の内容が表示されます（［テスト特有の GUI マップ ファイル］モードでの作業中は，［GUI マップ エディタ］には次に示すよりも少ないオブジェクトが含まれます）。

［GUI マップ エディタ］は，ツリーにオブジェクト名を表示します。それぞれの名前の前にはオブジェクトの種類を表すアイコンが付いています。オブジェクトはそれぞれが属しているウィンドウごとにグループ化されています。ウィンドウのアイコンをダブルクリックして，オブジェクトの表示を閉じたり拡張したりできます。

3 ツリーで［注文挿入］ボタンを見つけます。

［GUI マップ エディタ］で，［**表示**］＞［**オブジェクト ツリーの折りたたみ**］を選択して，ウィンドウのタイトルだけを表示します（［テスト特有の GUI

マップ ファイル］モードでの作業中は，［GUI マップ エディタ］には次に示すよりも少ないオブジェクトが含まれます)。

［フライト予約］ウィンドウをダブルクリックして，オブジェクトを表示します。［**注文挿入**］ボタンが見つかるまで，アルファベット順のオブジェクト・リストを下方へスクロールします。

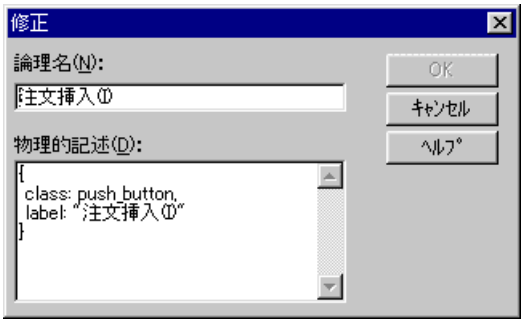
4［注文挿入］ボタンの物理的記述を表示します。

ツリーで［**注文挿入**］ボタンを見つけます（[テスト特有の GUI マップ ファイル］モードでの作業中は，［GUI マップ エディタ］には次に示すよりも少ないオブジェクトが含まれます)。

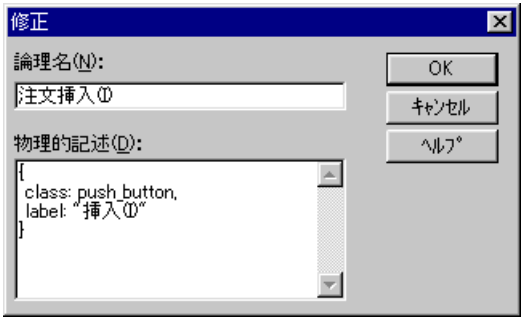
オブジェクトの物理的記述は，［GUI マップ エディタ］の下部の枠内に表示されます。

5［注文挿入］ボタンの物理的記述を変更します。

［**修正**］ボタンをクリックするか，［**注文挿入**］ボタンをダブルクリックします。［修正］ダイアログ・ボックスが開き，ボタンの論理名と物理的記述が表示されます。



［**物理的記述**］ボックスで，**label** プロパティを［**注文挿入**］から［**挿入**］に変更します。



［**OK**］をクリックして変更を適用し，ダイアログ・ボックスを閉じます。

6［**GUI マップ エディタ**］を閉じます。

GUI マップ エディタで，［**ファイル**］＞［**上書き保存**］を選択して変更を保存し，［**ファイル**］＞［**終了**］を選択します。［テストごとの GUI マップ エディタ］モードで作業中は，［GUI マップ エディタ］で［**ファイル**］＞［**終了**］を選択し，WinRunner で［**ファイル**］＞［**終了**］を選択します。

次回から論理名「**注文挿入**」を含むテストを実行すると，WinRunner はフライト予約 ウィンドウで［**挿入**］ボタンを探します。

［テスト特有の GUI マップ ファイル］モードで作業中は，戻って **lesson9** テストの手順 1 から 6 を実行します。

グローバルな GUI マップ・モードで作業する利点は，1 ヵ所でオブジェクトを変更すれば，同じ GUI マップ・ファイルを使用しているテストにすべてこの変更が適用される点です。テスト・モードごとに GUI マップ・ファイルを使い分けている場合は，変更したオブジェクトを含む各テストの GUI マップ・ファイルをそれぞれ更新しなければなりません。

GUI マップへの GUI オブジェクトの追加

注 : ［テスト特有の GUI マップ ファイル］モードでの作業中は，この練習はスキップしてください。新しいオブジェクトは，テストを保存する際に自動的に GUI マップに保存されるためです。

アプリケーションに新しいオブジェクトが含まれる場合，テスト・スクリプト・ウィザードを再度実行しなくても，そのオブジェクトを GUI マップに追加できます。［GUI マップ エディタ］で，［学習］ボタンを押すだけで，オブジェクトの記述を学習できます。ウィンドウ内の 1 つまたはすべてのオブジェクトの記述を学習できます。

この練習では，フライト予約アプリケーションの［ログイン］ウィンドウのオブジェクトを GUI マップに追加します。

1 フライト予約アプリケーションの［ログイン］ウィンドウを開きます。

［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**Sample Applications**］＞［**Flight 4A**］を選択します。

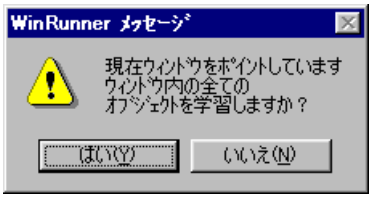
2 ［GUI マップ エディタ］を開きます。

WinRunner で［**ツール**］＞［**GUI マップ エディタ**］を選択します。［GUI マップ エディタ］が開きます。［**表示**］＞［**GUI ファイル**］を選択します。

3 ［ログイン］ウィンドウですべてのオブジェクトを学習します。

［**学習**］ボタンをクリックします。ポインタを使って，［ログイン］ウィンドウのタイトル・バーをクリックします。

ウィンドウのすべてのオブジェクトを学習したいかどうかを尋ねるメッセージが現れます。［**はい**］をクリックします。



WinRunner が［ログイン］ウィンドウの各オブジェクトの記述を学習し，それを仮の GUI マップに追加するのを観察してください。

4 **新しいオブジェクトを GUI マップに保存します。**

GUI マップ・エディタで［**ファイル**］＞［**上書き保存**］を選択します。［**はい**］か［**OK**］をクリックし，新規オブジェクトまたは新規ウィンドウを GUI マップに追加します。［**ファイル**］＞［**終了**］を選択し，GUI マップ・エディタを終了します。

新規ウィンドウと新規オブジェクトの保存の詳細については，42 ページの手順 7 を参照してください。

5 **［ログイン］ウィンドウを閉じます。**

［**キャンセル**］をクリックします。

実行ウィザードを使った GUI マップの更新

注：［テスト特有の GUI マップ ファイル］モードでの作業中は，この練習はスキップしてください。新しいオブジェクトは，テストを保存する際に自動的に GUI マップに保存されるためです。

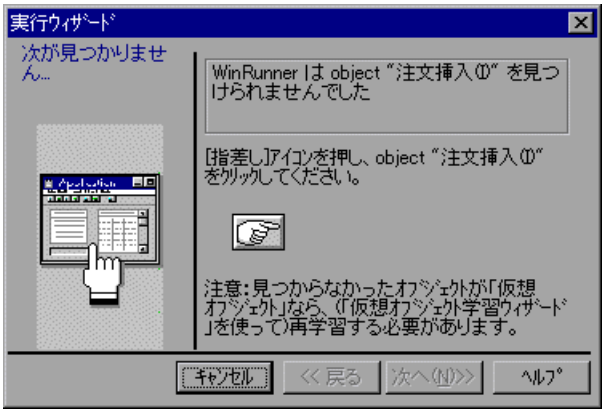
テストの実行中に WinRunner がテスト・スクリプトに記述されているオブジェクトを見つけられないと，［実行ウィザード］が開きます。［実行ウィザード］は GUI マップの更新の手助けをします。これでテストがスムーズに実行できます。このウィザードは，アプリケーションの不明なオブジェクトを指し示すよう求めてきます。そして，そのオブジェクトを見つけ出せなかった原因を判定

して，解決策を示します。ほとんどの場合，［実行ウィザード］は GUI マップのオブジェクト記述を自動的に変更し，新しいオブジェクト記述を追加します。

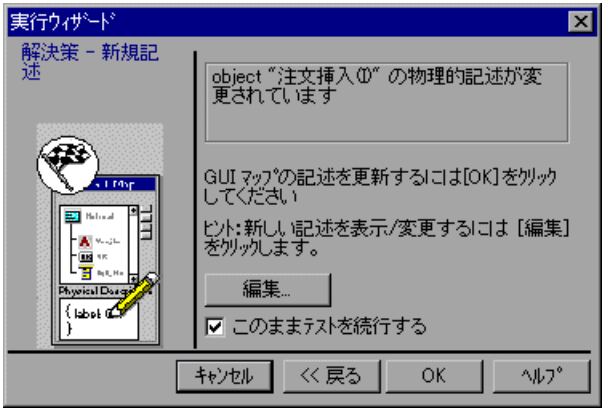
例えば，［フライト予約］ウィンドウで，［**注文挿入**］ボタンを押すテストを実行すると想定します。

```
button_press (" 注文挿入 ");
```

［**注文挿入**］ボタンが［**挿入**］ボタンに変更された場合，［実行ウィザード］がテスト実行中に開き，問題を解説します。



ウィザードで手のボタンをクリックし，フライト予約プログラムの［**挿入**］ボタンをクリックします。［実行ウィザード］は解決策を示します。



［**OK**］をクリックすると，WinRunner は GUI マップでオブジェクトの物理的記述を自動的に変更します。その後テスト実行を再開します。

実行ウィザードの動き方を見るには，次の手順を実行します。

1 ［GUI マップ エディタ］を開きます。

［**ツール**］＞［**GUI マップ エディタ**］を選択します。［**表示**］＞［**GUI ファイル**］を選択します。［**GUI ファイル**］ボックスから **flight4a.GUI** を選択します。

2 ［GUI マップ エディタ］のツリーから，「出発地」リスト・オブジェクトを削除します。

「**出発地**」オブジェクトが，［フライト予約］ウィンドウに一覧表示されます。このオブジェクトを選択し，［GUI マップ エディタ］で［**削除**］ボタンをクリックします。GUI マップ・ファイルを保存し［GUI マップ エディタ］を閉じます。

3 フライト予約 4A を開きます。

［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**Sample Applications**>］＞［**Flight 4A**］を選択します。［ログイン］ウィンドウで，名前とパスワード「mercury」を入力し，［**OK**］ボタンを押します。フライト予約アプリケーションと WinRunner の全体が見えるように，両方をデスクトップ上で配置し直します。

4 WinRunner で，lesson4 テストを開いて実行します。

WinRunner がこのステートメントに到達すると，何が起こるかを確認します。

```
list_select_item ("Fly From:", "Los Angeles");
```

5 ［実行ウィザード］の指示に従います。

［実行ウィザード］は「**出発地**」オブジェクトを指すよう求めてきます。オブジェクトを指すとオブジェクト記述が GUI マップに追加されます。WinRunner はその後テストの実行を続行します。オブジェクトのラベルを「**挿入**」に変更したため［**注文の挿入**］ボタンも指すよう求めてきます。［実行ウィザード］の指示に従って［注文の挿入］オブジェクトの詳細を修正してください。

6 「出発地」オブジェクトを GUI マップで保存します。

WinRunner がテスト実行を完了したら，［GUI マップ エディタ］に戻り，「出発地」オブジェクト記述を探します。「グローバルな GUI マップ」モードで作業している場合は，［実行ウィザード］は一時 GUI マップにオブジェクトを追加します。［**ファイル**］＞［**上書き保存**］をクリックしてオブジェクトを **flight1.GUI** マップ・ファイルに追加します。

7 GUI マップを閉じます。

［GUI マップ エディタ］で，［**ファイル**］＞［**終了**］を選択します。

8 フライト予約アプリケーションを閉じます。

［**ファイル**］＞［**終了**］を選択します。

練習 11

これ以降の進め方

練習 1 から練習 10 までの練習を完了したので，これらを通して学んだ WinRunner の考え方や技法を，自分のアプリケーションに適用する準備ができたことになります。

この練習では次のことを学びます。

- アプリケーションのテストを開始する方法
- WinRunner の追加情報を検索する場所について

テストの開始方法

アプリケーションのテストを開始するには，まず使用する GUI マップ・モードを決定します。

このチュートリアルを通して［テスト特有の GUI マップ ファイル］モードを使い，このモードで作業を続けたい場合，テストの記録を直ちに開始できます。

［グローバルな GUI マップ ファイル］モードを使用する場合は，テスト・スクリプト・ウィザードを使って，アプリケーションに含まれるすべてのオブジェクトの記述を学習します。ただし，その前に，サンプル・アプリケーションのオブジェクトの記述を GUI マップから削除してください。

［グローバルな GUI マップ ファイル］モードでテストを開始するには，次の手順を実行します。

1 **デスクトップで，WinRunner とテストしたいアプリケーション以外のすべてのアプリケーションを閉じます。**

2 **flight4a の GUI マップを閉じます。**

［**ツール**］＞［**GUI マップ エディタ**］を選択します。［**表示**］＞［**GUI ファイル**］を選択します。**flight4a.gui** マップ・ファイルを表示していることを確認します。［**ファイル**］＞［**閉じる**］を選択し，**flight4a.gui** マップ・ファイルを閉じます。［**ファイル**］＞［**終了**］を選択して，［GUI マップ エディタ］を閉じます。

3 **起動スクリプトから，flight4a の GUI マップを削除します。**

WinRunner で，［**ファイル**］＞［**テストを開く**］を選択します。**＜WinRunner インストール・パス＞ ¥dat** を参照します。**myinit** ファイルを選択します。次の行を削除します。

GUI_load(" ＜パス＞ ¥¥dat¥¥flight4a.GUI");

［**ファイル**］＞［**上書き保存**］を選択し，起動スクリプトを保存します。

4 **アプリケーションでテスト・スクリプト・ウィザードを実行します。［包括学習］モードでオブジェクト記述を学習します。**

テスト・スクリプト・ウィザードを使って，アプリケーションの各オブジェクトの記述を学習します。［**挿入**］＞［**テスト スクリプト ウィザード**］を選択し，画面の指示に従います。

ウィザードが学習フローを選択するように指示してきたら，［**包括学習**］を選択します。このモードは，WinRunner がオブジェクト記述を学習する方法をコントロールできるようにします。包括学習モードでは，論理名をカスタマイズしたり，ユーザ定義オブジェクトを標準オブジェクト・クラスにマップしたりできます。

学習工程が完了したら，ウィザードは GUI マップ・ファイルと起動スクリプトを作成します。グループでテスト作業をしている場合は，この情報を共有ネットワーク・ドライブに格納します。

ウィザードの使用中にヘルプを見る必要が生じたら，該当する画面で［**ヘルプ**］ボタンを押します。

5 テストを作成します。

ウィザードが終了したら，WinRunner で［グローバルな GUI マップ ファイル］モードを使ってテストの作成を開始できます。記録，プログラミング，またはその両方によって，自動テスト・スクリプトを作成します。

［テスト特有の GUI マップ ファイル］モードでテストを開始するには，次の手順を実行します。

1 デスクトップで，WinRunner とテストしたいアプリケーション以外のすべてのアプリケーションを閉じます。

2 テストを作成します。

作成するテストの主な段階を検討し，適切な記録モードを選択します。記録，プログラミング，またはその両方によって，自動テスト・スクリプトを作成します。

追加情報を入手する方法

WinRunner と TSL の詳細については，WinRunner 付属のユーザーズ・ガイドやオンライン・リソースを参照してください。

関連マニュアル

このチュートリアルのほかにも，WinRunner には次の関連マニュアルが付属しています。

『**WinRunner ユーザーズ・ガイド**』: WinRunner を使ってアプリケーションをテストする方法をステップごとに示しています。このチュートリアルで取り上げていない多くの便利なテストについて解説しています。

『**WinRunner インストール・ガイド**』: 単独のコンピュータまたはネットワークに接続されているコンピュータに WinRunner をインストールする方法を説明します。

『**WinRunner カスタマイズ・ガイド**』: アプリケーションの特殊なテスト要件に合わせて WinRunner をカスタマイズする方法を説明します。

『**TSL リファレンス・ガイド**』: TSL（テスト・スクリプト言語）および TSL に含まれる関数を説明します。

オンライン・リソース

WinRunner には次のオンライン・リソースがあります。

最初にお読みください：WinRunner に関する最新のニュースと情報を提供します（［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**Read Me**］＞）。

WinRunner の新機能：お使いのバージョンの WinRunner の新機能を説明します。

オンライン文書：全マニュアルを PDF 形式で提供します。オンライン文書は，Adobe Acrobat Reader を使用して参照および印刷できます。バージョン 5.0 以降をお勧めします。Adobe Acrobat Reader は www.adobe.co.jp からダウンロードできます。WinRunner のオンライン・マニュアルのアップデートについては，Mercury のカスタマー・サポート Web サイトをご覧ください。

WinRunner コンテキスト・センシティブ・ヘルプ：WinRunner で作業する際に生じる質問について直ちにお答えします。メニュー・コマンドとダイアログ・ボックスについて説明し，WinRunner のタスクを実行する方法を示します。Mercury カスタマー・サポート Web サイトで WinRunner ヘルプ・ファイルの最新版をご確認ください。

TSL オンライン・リファレンス：テスト・スクリプト言語（TSL）や含まれる関数，そして関数の使用法についての解説を行います。Mercury カスタマー・サポート Web サイトで「**TSL オンライン・リファレンス**」の最新版をご確認ください。

サンプル・テスト：ユーティリティとサンプル・テストが説明付きで含まれています。Mercury カスタマー・サポート Web サイトで WinRunner サンプル・ファイルの最新版をご確認ください。

WinRunner クイック・プレビュー：普段お使いの Web ブラウザで WinRunner の概要を開きます。

オンライン・カスタマー・サポート：普段お使いの Web ブラウザで，Mercury のカスタマー・サポート Web サイトを開きます。

サポート情報：Mercury のホームページ，カスタマー・サポート Web サイト，各国の Mercury のオフィスの一覧を紹介します。

Mercury Web サイト：普段お使いの Web ブラウザで，Mercury のホームページを開きます。このサイトでは，Mercury とその製品，およびサービスに関する最新情報を提供します。新しいソフトウェアのリリース，セミナー，展示会，カスタマー・サポート，トレーニングなどの情報も含まれています。