ユーザーズ・ガイド

MERCURY™

Mercury WinRunner

ユーザーズ・ガイド

Version 8.2

MERCURY™

Mercury WinRunner ユーザーズ・ガイド , 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。

WRUG8.2JP/01

目次

第 10 部 : テストの作成 ー 基本

第 22 部 : テストの実行 ー 基本

第 35 部 : TSL を使ったプログラミング

Mercury WinRunner へようこそ

WinRunner へようこそ。WinRunner は，Mercury の企業向けテスト自動化ソリューションです。WinRunner では，作成したアプリケーションを対象とする洗練された自動テストをすばやく作成して実行できます。

注 :『Mercury WinRunner 基本機能ユーザーズ・ガイド』と**『Mercury WinRunner 上級機能ユーザーズ・ガイド』**は，印刷版においてのみ別々になっています。PDF およびコンテキスト・センシティブ・ヘルプでは 1 つに統合されています。

本書の使い方

本書では，ソフトウェアの自動テストの背景にある主要概念を説明します。また，テストの作成，デバッグ，実行の手順 , およびテスト工程において，検出した不具合を報告する方法についても説明します。

本書は，以下の部で構成されています。

第 1 部　テスト・プロセスの開始

WinRunner の概要と，テスト工程の主要な段階について説明します。

第 2 部　GUI マップについて

コンテキスト・センシティブ・テストの方法と，柔軟性が高く，再利用可能なテスト・スクリプトを作成するための GUI マップの重要性について説明します。

第 3 部　テストの作成 ― 基本

テスト・スクリプトの作成方法，チェックポイントの挿入方法，パラメータの割り当て方法について説明します。

第 4 部　テストの実行 ― 基本

WinRunner からテストを実行し，テスト結果を分析する方法について説明します。

第 5 部　基本設定

WinRunner の標準設定をグローバルにあるいはテストごとに変更する方法を説明します。

第 6 部　GUI マップを使った作業

GUI マップ・ファイルを結合し設定する方法を説明します。また，ビットマップを「*仮想オブジェクト*」として定義することで，ビットマップを GUI オブジェクトとして認識させる方法も説明します。

第 7 部　テストの作成 ― 上級

正規表現の使用方法，およびテストの実行中に生じる予期しないイベントの処理方法などについて説明します。

第 8 部　TSL を使ったプログラミング

変数，フロー制御ステートメント，配列，ユーザ定義関数および外部関数，WinRunner のビジュアル・プログラミング・ツール，そしてテスト実行中の対話的な入力を使って，テスト・スクリプトの機能を強化する方法について解説します。

第 9 部　テストの実行 ― 上級

バッチ・テストを実行する方法のほか，WinRunner とコマンドラインの両方からテストを実行する方法を説明します。

第 10 部　テストのデバッグ

テスト実行を制御し，テスト実行中にブレークポイントを使ったり変数を監視したりすることによってテスト・スクリプトの不具合を特定して切り分ける方法について説明します。

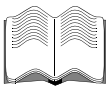
第 11 部　上級者向けの設定

WinRunner のユーザ・インタフェース，テスト・スクリプト・エディタ，および関数ジェネレータをカスタマイズする方法について説明します。また，特定の設定を初期化して WinRunner をテスト環境に適合させる方法についても説明します。

第 12 部　その他の Mercury 製品を使った作業

WinRunner を QuickTest Professional，Quality Center，Business Process Testing，および LoadRunner と統合する方法を説明します。

WinRunner の関連マニュアル



WinRunner には，この『ユーザーズ・ガイド』に加えて，以下の印刷版のマニュアルが付属します。

『WinRunner 基本機能 ユーザーズ・ガイド』：WinRunner を使用してアプリケーションの個別のテスト要件に対応するための手順を説明します。

『WinRunner 上級機能ユーザーズ・ガイド』：アプリケーションの個別のテスト要件に対応するために利用できるさらに高度な WinRunner 機能に関する情報を提供します。

『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 チュートリアル』：WinRunner の基本的な使い方，およびアプリケーションのテストを開始する方法について説明します。

『テスト・スクリプト言語リファレンス』：WinRunner テスト・スクリプト言語（TSL）と，TSL に含まれている関数について説明します。

『WinRunner カスタマイズ・ガイド』：アプリケーションごとの特別な要件に対応するために WinRunner をカスタマイズする方法を説明します。

オンライン・リソース

WinRunner には，以下のオンライン・リソースがあります。これらは，プログラム・グループまたは［ヘルプ］メニューから表示できます。

最初にお読みください：WinRunner に関する最新の情報を提供します。.

WinRunner ヘルプ：WinRunner で作業をしている中で疑問が生じたときに，現在のコンテキストに応じて即座に答を提示します。メニュー・コマンド，ダイアログ・ボックスを説明するほか，WinRunner で作業を行う方法を示します。

WinRunner クイック・プレビュー：WinRunner を初めて使うユーザに対して，WinRunner の主要な機能を説明する簡単なプレゼンテーションを提供します。

TSL オンライン・リファレンス：TSL（テスト・スクリプト言語）と，TSL に含まれている関数について説明し，その使用例を示します。

印刷用ドキュメント：ドキュメント一式を PDF 形式で表示します。オンライン文書は，インストール・パッケージに付属の Adobe Acrobat Reader を使って読んだり印刷したりできます。バージョン 5.0 以降を使用することをお勧めします。Adobe Acrobat Reader の最新版は，www.adobe.com からダウンロードできます。

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。

WinRunner の新機能：WinRunner の最新バージョンの新機能について説明します。

注：『Mercury WinRunner ユーザーズ・ガイド』のオンライン版は，単独ボリューム構成であるのに対し，印刷版と PDF 版は**『Mercury WinRunner 基本機能ユーザーズ・ガイド』**と**『Mercury WinRunner 上級機能ユーザーズ・ガイド』**の 2 冊からなります。

オンライン技術サポート：普段お使いの Web ブラウザで Mercury のカスタマ・サポートの Web サイトを開きます。この Web サイトの URL は，http://www.mercury.com/jp/resources/support/ です。

Mercury の 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 およびその製品とサービスについて最新の情報を提供します。ソフトウェアに新しいリリース，セミナー，トレード・ショー，カスタマ・サポート，トレーニングなどが含まれます。この Web サイトの URL は，http://www.mercury.com です。

表記規則

本書では，以下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。
➤	ブリット記号はオプションまたは特徴を示します。
＞	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**］	インタフェース要素の名前は，その要素を使用したアクションを実行する手順の説明で全角の大括弧に**太字**で示します（例：［**実行**］ボタンをクリックします）。
太字	メソッド名や関数名，書名，新機能は，**太字**で示します。
斜体字	*斜体字*のテキストは，変数名を示します。
Arial	使用例やユーザがそのまま入力しなければならない文字列は，Arial というフォントで示します。
[ ]	大括弧は，省略可能なパラメータを示します。
{ }	中括弧は，括弧内のいずれかの値をパラメータとして指定しなければならないことを示します。
...	構文内の省略記号は，同じ形式で項目をさらに組み入れることができることを意味します。
|	垂直バー（パイプ記号）は，バーで区切ったオプションのいずれか一方を指定しなければならないことを示します。

第 1 部

テスト・プロセスの開始

第 1 章

はじめに

クライアント / サーバ・ソフトウェア・ツールの昨今の向上により，アプリケーションはより早く構築できるようになり，機能性も向上しました。品質保証部門は，劇的に進歩したソフトウェアに対処しなければならなくなりましたが，テストの複雑性は増しました。コードの変更，改良，不具合修正を行うたびに，またはプラットフォーム・ポートごとに，アプリケーション全体を再テストして品質の高いリリースを実現する必要があります。この動的に進歩する環境では，手動テストを行っていたのではもはや間に合いません。

Mercury WinRunner は，エンタープライズ用の強力なテストの自動化ソリューションです。WinRunner は，テストの開発から実行まで，テスト工程の自動化を支援します。アプリケーションの機能性をテストする柔軟かつ再利用可能なテスト・スクリプトを作成できます。ソフトウェアのリリース前に，これらのテストを 1 晩実行することで，不具合を検出でき，より優れた品質のソフトウェアをリリースできます。

既存の WinRunner テストをスクリプト化コンポーネントに変換したり，新しいスクリプト化コンポーネントを作成したりすることもできます。スクリプト化コンポーネントは Mercury Quality Center の Business Process Testing の一部であり，アプリケーションのテストにキーワード駆動型の方法論を使用するものです。スクリプト化コンポーネントは，WinRunner で作成でき，ビジネス・プロセス・テストで使用できる，再利用可能なモジュール形式のスクリプトです。

本書の情報，例，画面キャプチャは，特に WinRunner テストで作業するものに的を絞っています。テストに関連する情報の多くは，テストと同様の機能を持つスクリプト化コンポーネントにも関連するものです。

Quality Center との統合およびスクリプト化コンポーネントを使った作業の方法については，を参照してください。詳細については，『**Business Process Testing ユーザーズ・ガイド**』を参照してください。

WinRunner テストのモード

WinRunner は，アプリケーションでの作業方法を記録することによって，テストの作成を簡単にします。アプリケーションで GUI（グラフィカル・ユーザ・インタフェース）オブジェクトをポイントしてクリックすると，WinRunner は C 言語に似たテスト・スクリプト言語（TSL）でテスト・スクリプトを自動的に生成します。手作業でプログラミングを行うことで，テスト・スクリプトをさらに強化できます。WinRunner には，記録済みのテストに素早く簡単に関数を追加できる関数ジェネレータが含まれています。

WinRunner には，テストを記録するためのモードが 2 つあります。コンテキスト・センシティブ・モードとアナログ・モードです。

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，テスト対象アプリケーションでのアクションを，選択した GUI オブジェクト（ウィンドウ，リスト，ボタンなど）の観点から記録します。画面上のオブジェクトの物理的位置は無視されます。テスト対象アプリケーションで操作を実行するたびに，選択したオブジェクトや実行したアクションを記述する TSL ステートメントがテスト・スクリプトに生成されます。

記録により，WinRunner は選択したオブジェクトの一意の記述を GUI マップ・ファイルに記述します。GUI マップ・ファイルは，テスト・スクリプトとは別に保守でき，同じ GUI マップ・ファイル（複数も可）を複数のテストで使用できます。アプリケーションのユーザ・インタフェースが変わった場合は，数百ものテストを更新しなくても，GUI マップだけを更新すればよいのです。これにより，アプリケーションの以降のバージョンでコンテキスト・センシティブなテスト・スクリプトを再利用できるようになりました。

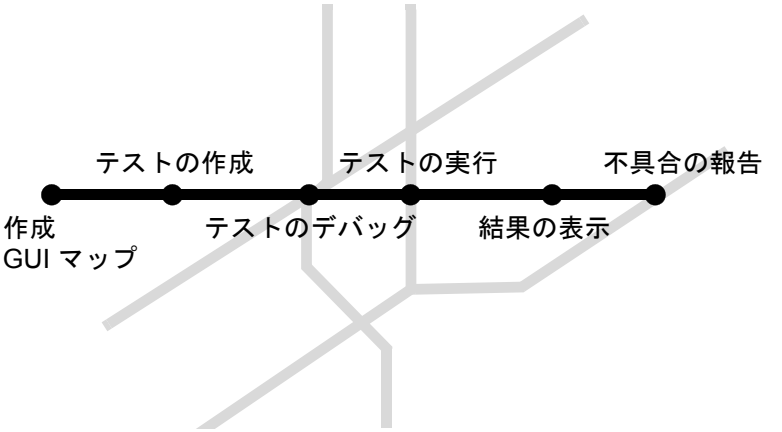
テストを実行するには，テスト・スクリプトを再生します。WinRunner は，アプリケーション上でマウス・ポインタを動かし，オブジェクトを選択し，キーボード入力を行って，ユーザをエミュレートします。WinRunner は，GUI マップでオブジェクトの物理的記述を読み取り，これらの記述に一致するオブジェクトをテスト対象アプリケーションで検索します。ウィンドウ内のオブジェクトの位置が変わっても，オブジェクトを見つけることができます。

アナログ

アナログ・モードは，マウス・クリック，キーボード入力，およびマウスが移動した正確な x，y 座標を記録します。テストを実行すると，WinRunner はマウスの軌跡をたどります。描画アプリケーションのテストなど，正確なマウスの動きが重要なテストではアナログ・モードを使用します。

WinRunner のテスト・プロセス

WinRunner によるテストには，次の主要な 6 つの段階があります。



GUI マップの作成

最初の段階は，GUI マップの作成です。これにより，WinRunner はテスト対象アプリケーションの GUI オブジェクトを認識できます。テスト・スクリプト・ウィザードを使用して，アプリケーションのユーザ・インタフェースを調査し，GUI マップに各 GUI オブジェクトの物理的記述を系統立てて追加できます。個々のオブジェクトを記録中にクリックすることで，GUI マップにこれらのオブジェクトの物理的記述を追加することもできます。

テスト特有の GUI マップ・モードで作業している場合は，この手順をスキップできます。詳細については，第 3 章「WinRunner の GUI オブジェクトの識別方法」を参照してください。

テストの作成

テスト・スクリプトは，記録，プログラミング，またはこれら両方の組み合わせによって作成できます。テストの記録中，テスト対象アプリケーションの反応を検査したい場所にチェックポイントを挿入します。GUI オブジェクト，ビットマップ，データベースをチェックするチェックポイントを挿入できます。この工程で，WinRunner はデータをキャプチャし，そのデータを**期待結果**（テスト対象アプリケーションの期待する応答）として保存します。

テストのデバッグ

テストがスムーズに実行されるかどうかを検査するには，テストをデバッグ・モードで実行します。ブレークポイントの設定，変数の監視，テストの実行方法の制御を行って，不具合を特定し，切り分けることができます。テスト結果はデバッグ・フォルダに保存されます。このフォルダはテストのデバッグが終了したら削除できます。

WinRunner はテストを実行すると，各スクリプト行に基本的な構文エラーや，**If**，**While**，**Swich**，**For** などのステートメントに欠落している要素がないか調べます。［**構文チェック**］オプション（［**ツール**］＞［**構文チェック**］）を使用して，テストを実行する前にこれらの構文エラーを検査します。

テストの実行

テストを**検証**モードで実行すると，アプリケーションをテストできます。テスト・スクリプト内でチェックポイントに遭遇すると，WinRunner はテスト対象のアプリケーションの現在のデータを，以前にキャプチャした期待データと比較します。不一致が検出されると，WinRunner はこれらを**実際の結果**としてキャプチャします。

注：検証モードは，テストの実行用で，コンポーネントには使用できません。コンポーネントでの作業中は，アプリケーションはコンポーネントが Quality Center のビジネス・プロセス・テストの一部として実行されている場合に検証されます。

結果の表示

結果を表示して，テストの成功または失敗を特定します。各テスト実行後に WinRunner は結果をレポート形式で表示します。レポートには，チェックポイント，エラー・メッセージ，システム・メッセージまたはユーザ・メッセージなど，実行中に発生したすべての主要なイベントの詳細が記録されます。

テストの実行中にチェックポイントで不一致が検出された場合，期待結果と実際の結果を［テスト結果］ウィンドウで参照できます。ビットマップの不一致の場合は，期待結果と実際の結果の差異だけを表示したビットマップを表示することもできます。

結果は，標準 WinRunner レポート・ビューまたは統一レポート・ビューで表示できます。WinRunner レポート・ビューは，Windows 形式のビューアにテスト・結果を表示します。統一レポート・ビューは，HTML 形式のビューア（QuickTest Professional のテスト結果に使用されるものと同一です）に結果を表示します。

不具合の報告

テスト対象アプリケーションの不具合のためにテストが失敗した場合は，［テスト］結果ウィンドウから直接不具合に関する情報をレポートできます。この情報は，不具合を修正されるまで追跡する品質保証マネージャによって管理されます。

テスト・スクリプトに **qcdb_add_defect** ステートメントを挿入して，テスト・スクリプトで定義した条件に基づいて不具合を Quality Center プロジェクトに追加することもできます。

サンプル・アプリケーション

本書の例の多くでは，WinRunner に用意されているサンプルのフライト予約アプリケーションを使用します。

サンプル・アプリケーションの起動

このアプリケーションは，［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**Sample Applications**］を選択し，フライト・アプリケーションのバージョン（Flight 4A または Flight 4B）を選択して起動します。

サンプル・アプリケーションのバージョン

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 は正しく動作するアプリケーションですが，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。本書の例では，Flight 4A または Flight 4B のいずれかを使用できます。

WinRunner が Visual Basic サポートと一緒にインストールされている場合は，Flight A および Flight B の Visual Basic バージョンも標準の Windows ベースのサンプル・アプリケーションと一緒にインストールされます。

ログイン

サンプルのフライト予約アプリケーションを起動すると，［ログイン］ボックスが開きます。アプリケーションを起動するには，ログインする必要があります。ログインするには，４ 文字以上の名前を入力します。パスワードは「mercury」です（大文字と小文字は区別しません）。

サンプル Web アプリケーション

WinRunner には，サンプルの Web 版フライト予約アプリケーションも含まれています。この Web サイトの URL は，**http://newtours.mercuryinteractive.com** です。このアプリケーションは，［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**Sample Applications**］＞［ **Mercury Tours site**］を選択して起動することもできます。

他の Mercury 製品との統合

WinRunner は，他の Mercury 製品と組み合わせて使用し，テスト工程のさまざまな段階（テスト計画，テスト開発，GUI および負荷テスト，不具合追跡，マルチ・ユーザ・システムのクライアントの負荷テストなど）に統合されたソリューションを提供できます。

Mercury QuickTest Professional

QuickTest Professional は，使いやすいながらも，包括的なアイコン形式の機能テスト・ツールで，動的な Windows ベースの Visual Basic，ActiveX，Web，およびマルチメディア・アプリケーションの機能テストと回帰テストを実行するために設計されました。.QuickTest の機能性を拡張して，Java，.NET，SAP，Siebel，PeopleSoft，Oracle などの最先端の開発環境を使用して作成されたアプリケーションのテストも行えます。

QuickTest Professional でテストを設計し，QuickTest テストから WinRunner テストを呼び出すことによって既存の WinRunner スクリプト・ライブラリを活用できます。また，QuickTest テストを WinRunner から呼び出すこともできます。

Mercury Quality Center

Quality Center（以前の TestDirector）はアプリケーションの品質管理製品です。品質保証担当者がテスト工程を計画したりまとめたりするのに役立ちます。Quality Center を使用すると，スクリプト化コンポーネントおよび手動 / 自動テストのデータベースの作成，テスト・サイクルの構築，テストの実行，不具合の報告および追跡が可能です。また，ソフトウェアのリリース前のテスト計画，テスト実行，不具合追跡の進行状況の確認に役立つ，レポートやグラフも作成できます。

WinRunner を使用する場合は，テストとスクリプト化コンポーネントを Quality Center データベースに直接保存するかどうかを選択できます。また，WinRunner でテストを実行してから Quality Center でテスト・サイクルの全体的な結果を検討できます。

WinRunner と，Business Process Testing サポートの組み込まれた Quality Center を組み合わせて，既存の WinRunner スクリプト・ライブラリへの投資を活用し，Business Process Testing フレームワークを使用することによって，テストの自動化プロセスを向上できます。

Mercury LoadRunner

LoadRunner は Mercury の自動テスト実行ソリューションです。LoadRunner を使用して，多くのユーザが 1 つのサーバ・アプリケーションで同時に作業する環境をエミュレートできます。LoadRunner では実際のユーザの代わりに仮想ユーザにテスト対象アプリケーションで自動テストを実行させます。仮想ユーザを複数のホスト・コンピュータで同時に有効にして，「負荷のかかった状態」でのアプリケーションのパフォーマンスをテストできます。

第 2 章

WinRunner の概要

本章では，WinRunner の起動方法と WinRunner のウィンドウについて説明します。

本章では，次の項目について説明します。

- WinRunner の起動方法
- WinRunner のメイン・ウィンドウ
- テスト・エディタ・ウィンドウ
- WinRunner コマンドの使用方法
- WinRunner アドインのロード

WinRunner の起動方法

WinRunner を初めて起動する際は，次の手順を実行します。

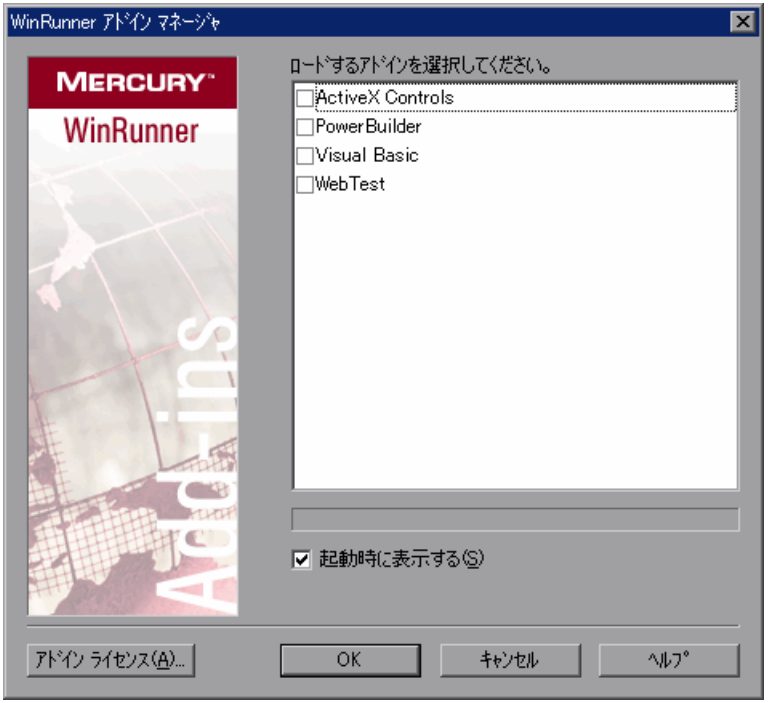
1 ［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**WinRunner**］を選択します。

Windows のタスクバーの状態表示領域に WinRunner の記録 / 実行エンジンのアイコンが現れます。このエンジンは，WinRunner とテスト対象アプリケーションの間の接続を確立して維持します。

2 標準設定では，WinRunner アドイン・マネージャ・ダイアログ・ボックスが開きます。

［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。現在の WinRunner セッション でロードするアドインを選択します。

一定の時間［アドイン マネージャ］ダイアログ・ボックスで変更を行わないと，ウィンドウが閉じ，前回の WinRunner セッションでロードしたアドインが自動的にロードされます。プログレス・バーに，ウィンドウが閉じるまでの残り時間が表示されます。



注：お使いのコンピュータで WinRunner の新しいバージョンを始めて起動すると，「WinRunner の新機能」ヘルプも表示されます。

アドイン・マネージャの詳細については，21 ページ「WinRunner アドインのロード」を参照してください。

3 ［WinRunner へようこそ］ウィンドウが開きます。［WinRunner へようこそ］ウィンドウで，新しい空のテストを開くには［**テストの新規作成**］を，保存済みのテストを選択して開くには［既存のテストを開く］を，普段お使いのブラウザで WinRunner の概要のプレゼンテーションを見るには［WinRunner のクイック プレビューを表示］をそれぞれ選択します。

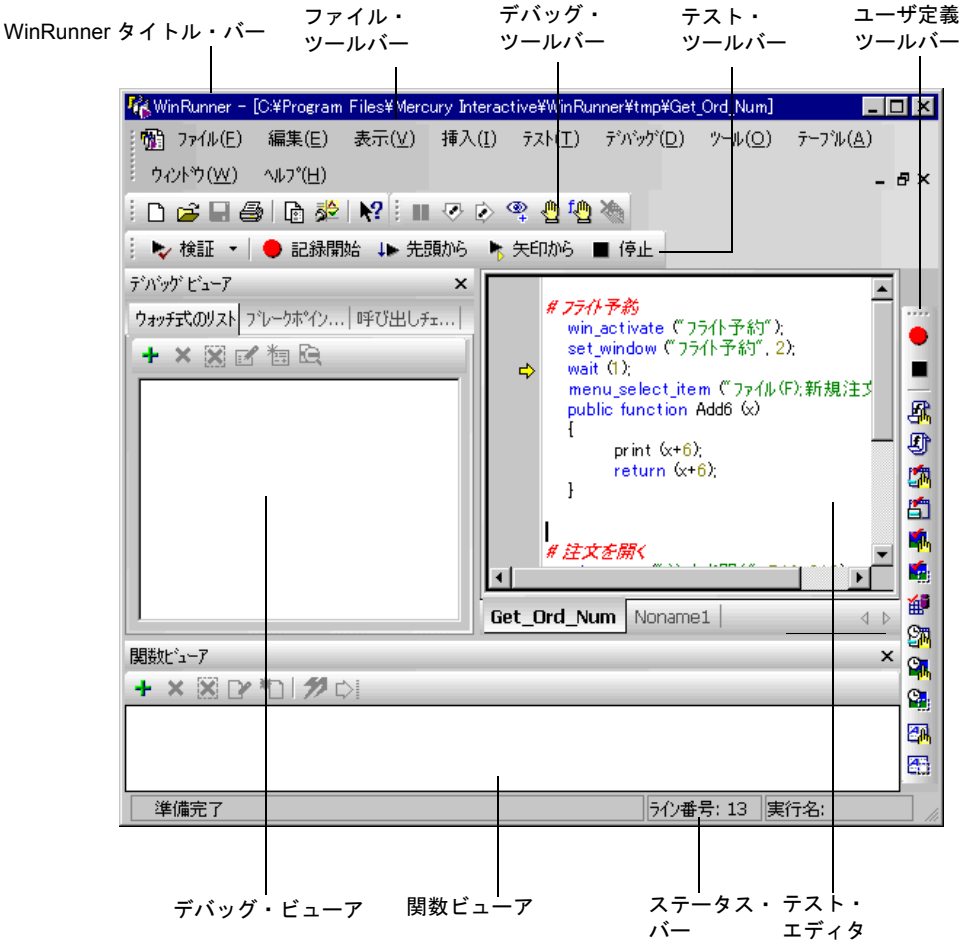
ヒント：次回 WinRunner を起動する際に［WinRunner へようこそ］ウィンドウを表示したくない場合は，［**起動時に表示**］チェック・ボックスをクリアします。このオプションは一度選択すると，後で WinRunner にようこそ画面を表示させることができます。［**ツール**］＞［**一般オプション**］を選択して，［**一般設定**］＞［**起動**］カテゴリを選択し，［**[ようこそ] 画面を起動時に表示する**］チェック・ボックスを選択します。

WinRunner のメイン・ウィンドウ

WinRunner ウィンドウの主な要素を以下に示します。

- **WinRunner タイトル・バー**：現在開いているテストの名前とパスが表示されます。
- **ファイル・ツールバー**：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。
- **デバッグ・ツールバー**：テストのデバッグ中に使用されるオプションにアクセスできます。
- **テスト・ツールバー**：テストの実行および保守中に使用されるオプションにアクセスできます。
- **ユーザ定義ツールバー**：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。標準設定では，ユーザ定義ツールバーは非表示です。ユーザ定義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。
- **ステータス・バー**：現在のコマンド，挿入ポイントの行番号，現在の結果フォルダの名前などの情報が表示されます。
- **テスト・エディタ**：テスト・スクリプトを表示します。
- **デバッグ・ビューア**：現在選択されているデバッグ・オプション（**ウォッチ式のリスト**，**ブレークポイント**，**呼び出しチェーン**）からデータを表示します。この表示枠は，［**デバッグ**］メニューで，選択されているすべてのデバッグ切り替えオプションをクリアすると閉じることができます。
- **関数ビューア**：テストから呼び出すことのできるロードされた関数を表示します。この表示枠は，［**ツール**］メニューで［**関数ビューア**］トグル・オプションをクリアすることによって閉じることができます。

WinRunner のメイン・ウィンドウの例を，次に示します。

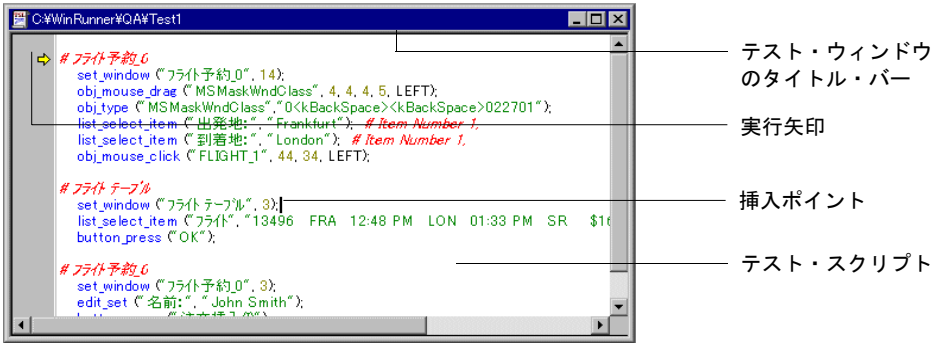


ヒント：標準設定では，各テストがテスト・エディタの異なるタブに表示されます。テストのタブが多く，エディタの底部に収まらない場合は，左矢印ボタンと右矢印ボタンを使用して，テストのタブを左または右にスクロールします。タブが表示されない場合は，［**一般オプション**］ダイアログ・ボックスの［**概観**］カテゴリで［**テスト タブを表示する**］オプションを選択して表示します。

テスト・エディタ・ウィンドウ

テスト・エディタ・ウィンドウで，WinRunner テストを作成し実行します。このウィンドウの主な要素を以下に示します。

- **テスト・ウィンドウのタイトル・バー**：開いているテストの名前を表示します。
- **テスト・スクリプト**：記録によって，またはテスト・スクリプト言語（TSL）でプログラミングして手作業で入力することによって生成されたステートメントが表示されます。
- **実行矢印**：テストの実行中に実行されているテスト・スクリプトの行または［**矢印から実行**］オプションを選択した場合に次の実行を開始する行を示します。
- **挿入ポイント**：テキストを挿入または編集する場所を示します。



WinRunner コマンドの使用方法

WinRunner コマンドはメニュー・バーまたはツールバーから選択できます。WinRunner コマンドには，ソフトキーを押して実行できるものもあります。

メニューでのコマンドの選択

すべての WinRunner コマンドを，メニュー・バーから選択できます。

ツールバーでのコマンドのクリック

ツールバーのボタンをクリックして，いくつかの WinRunner コマンドを実行できます。WinRunner には，**ファイル・ツールバー**，**テスト・ツールバー**，**デバッグ・ツールバー**，**アクション・ツールバー**の 4 つの組み込みツールバーがあります。**ユーザ定義ツールバー**は，最もよく使用するコマンドでカスタマイズできます。

フローティング・ツールバーの作成

ツールバーはフローティング・ツールバーに変更できます。また，ユーザ定義ツールバーがフローティング・ツールバーになっている場合は，WinRunner を最小化してもユーザ定義ツールバーのコマンドにはアクセスできます。したがって，テスト対象アプリケーションで自由に作業できます。

ツールバーのハンドルをダブルクリックして，これをフローティング・ツールバーに変更したり，フローティング・ツールバーをダブルクリックして，ツールバーに変更したりできます。ツールバー・ハンドルまたはタイトル・バーをドラッグして，固定ツールバーをフローティング・ツールバーに変更したり，フローティング・ツールバーを固定ツールバーに変更したりすることができます。

ファイル・ツールバー

ファイル・ツールバーには，テストを開く，テストの保存，テスト結果の表示，［ヘルプ］へのアクセスなどよく実行されるタスクに使用するコマンドのボタンが含まれます。ファイル・ツールバーの標準設定の位置は，WinRunner メニュー・バーの下です。

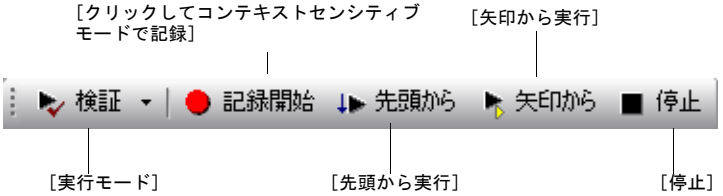
ファイル・ツールバーには次のボタンが表示されます。

ユーザ定義ツールバーの詳細については，第 8 章「テストの設計」を参照してください。

テスト・ツールバー

テスト・ツールバーには，テストの実行で使用されるコマンドのボタンが含まれます。WinRunner ファイル・ツールバーは標準設定では WinRunner メニュー・バーの下に固定されます。

テスト・ツールバーには次のボタンが表示されます。

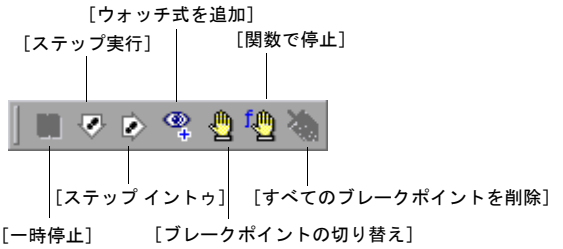


テスト・ツールバーの詳細については，第 19 章「テスト実行について」を参照してください。

デバッグ・ツールバー

デバッグ・ツールバー：テストのデバッグ中に使用されるコマンドのボタンが含まれます。デバッグ・ツールバーは標準設定では，WinRunner メニュー・バーの下，ファイル・ツールバーの右側に固定されます。

デバッグ・ツールバーには次のボタンが表示されます。

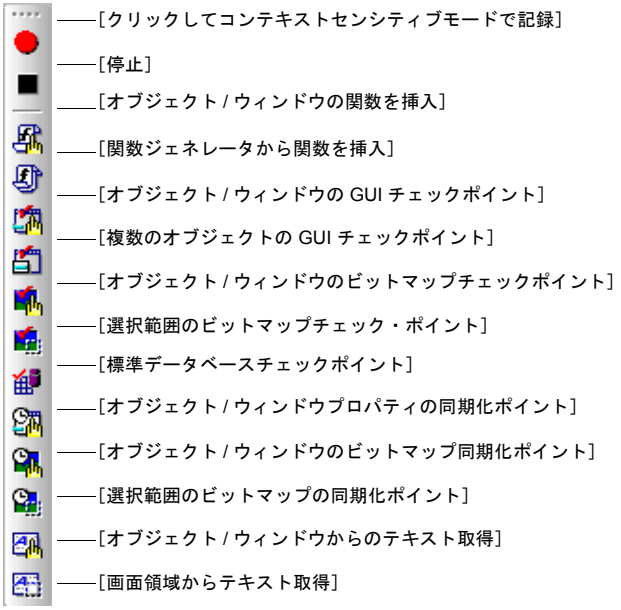


デバッグ・ツールバーの詳細については，第 19 章「テスト実行について」を参照してください。

ユーザ定義ツールバー

ユーザ定義ツールバーには，テストの作成中に使用されるコマンドのボタンが含まれます。標準設定では，ユーザ定義ツールバーは表示されません。ユーザ定義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を表示すると，標準設定では，WinRunner ウィンドウの右端に固定されます。

ユーザ定義ツールバーはカスタマイズできます。テスト対象アプリケーションによく使用するコマンドに簡単にアクセスできるボタンを追加または削除できます。標準設定では，ユーザ定義ツールバーには次のボタンが表示されます。



ユーザ定義ツールバーのカスタマイズについては，852 ページ「ユーザ定義ツールバーのカスタマイズ」を参照してください。

ソフトキーを使ったコマンドの有効化

ソフトキーを押すことによって，WinRunner コマンドのいくつかを実行できます。WinRunner は WinRunner ウィンドウが画面上でアクティブでなくても，あるいは最小化されていても，ソフトキーからの入力を読み取ります。

ソフトキーの割り当ては設定できます。テスト中のアプリケーションで，WinRunner にあらかじめ設定されている標準設定のソフトキーを使用している場合は，ソフトキー設定ユーティリティを使用して WinRunner のソフトキーを再定義できます。

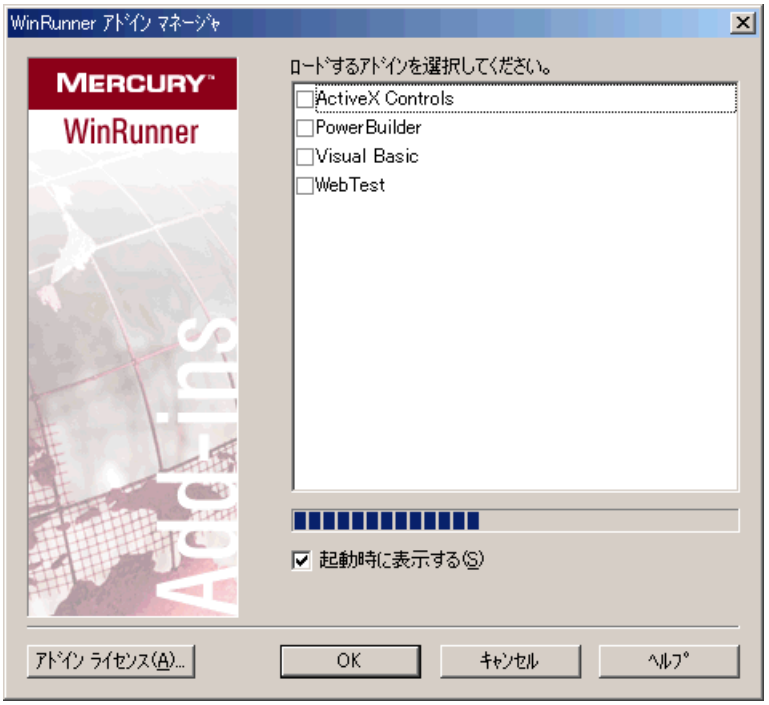
標準設定の WinRunner ソフトキーの設定の一覧と WinRunner ソフトキーの再定義に関する情報については，862 ページ「WinRunner ソフトキーの構成設定」を参照してください。

WinRunner アドインのロード

WinRunner のインストール時，あるいはインストールした後で，WebTest，Visual Basic，PowerBuilder，ActiveX などのコントロールなどのコア・アドインをインストールした場合，または外部アドインをインストールしている場合は，各 WinRunner セッションの最初でどのアドインをロードするかを指定できます。

標準設定では，WinRunner の起動時に［**アドイン マネージャ**］ダイアログ・ボックスが開きます。このダイアログ・ボックスにはインストールされている WinRunner アドインの一覧が表示されます。WinRunner の現在のセッションにロードするアドインを選択できます。一定時間変更を行わないと，ウィンドウが閉じ，選択したアドインが自動的にロードされます。

プログレス・バーに，ウィンドウが閉じるまでの時間が表示されます。



WinRunner を初めて起動したときには，アドインは選択されていません。次のセッションからは，前回のセッションで選択したアドインが標準設定となります。一覧に変更を行うと，タイマが停止するので，ダイアログ・ボックスを閉じて選択したアドインをロードするには［**OK**］をクリックする必要があります。

アドイン・マネージャには，お使いのコンピュータで使用されるアドインの一覧が表示されます。WinRunner のコア・インストールには，ActiveX コントロール，PowerBuilder，Visual Basic，WebTest アドインが含まれます。

外部 WinRunner アドインを購入することによって，多数の開発環境をサポートするよう WinRunner の機能性を拡張できます。

WinRunner の外部アドインをインストールすると，アドインはコア・アドインと一緒にアドイン・マネージャに表示されます。外部アドインをシート・ライセンスでインストールした場合は，特別な WinRunner アドイン・ライセンスもインストールしなくてはなりません。外部アドインをインストール後，初めて WinRunner を開くと，［アドイン マネージャ］にアドインが表示されますが，チェック・ボックスが無効となっており，アドイン名がグレーで表示されます。［**アドイン ライセンス**］ボタンをクリックしてアドインのライセンスをインストールします。詳細については，『**WinRunner インストール・ガイド**』を参照してください。

WinRunner の起動時に毎回［**アドイン マネージャ**］ダイアログ・ボックスを表示するかどうか，表示する場合はどのくらい長く表示するのかを決めることができます。［一般オプション］ダイアログ・ボックスの［**一般設定**］＞［**起動**］カテゴリの［**アドイン マネージャを起動時に表示する**］オプションを使用します。［一般オプション］ダイアログ・ボックスの使用方法については，第 22 章「グローバル・テスト・オプションの設定」を参照してください。これらのオプションは，**-addins** および **-addins_select_timeout** コマンドライン・オプションを使用して設定することもできます。コマンドライン・オプションを使った作業の詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

第 2 部

GUI マップについて

第 3 章

WinRunner の GUI オブジェクトの識別方法

本章では，コンテキスト・センシティブ・テストとは何かを紹介し，WinRunner がアプリケーションのグラフィカル・ユーザ・インタフェース（GUI）オブジェクトをどのように識別するかを解説します。

本章では，以下の項目について説明します。

- GUI オブジェクトの識別について
- テストが GUI オブジェクトを識別する方法
- 論理名
- GUI マップ
- ウィンドウのコンテキストの設定

GUI オブジェクトの識別について

コンテキスト・センシティブ・モードで作業する場合，アプリケーションの見え方に基づいてテストが行えます。つまり，ウィンドウ，メニュー，ボタン，リストといった GUI オブジェクトに基づいてテストが行えます。それぞれのオブジェクトには，オブジェクトのふるまいと外観を決める，あらかじめ定義されているプロパティがあります。WinRunner は，これらのプロパティを学習し，その情報に基づいて，テスト実行の際に GUI をオブジェクトを識別して探します。コンテキスト・センシティブ・モードでは，WinRunner は GUI オブジェクトの画面における物理的な位置を知っている必要はないのです。

GUI スパイを使うと，デスクトップに表示されている GUI オブジェクトのプロパティが表示されます。これにより，WinRunner が GUI オブジェクトをどのように識別するかわかります。GUI オブジェクトのプロパティの表示と WinRunner に学習させる方法の詳細については，第 4 章「GUI マップの基本概念について」を参照してください。

WinRunner は学習した情報を，**GUI マップ**に格納します。WinRunner は，テストを実行すると，GUI マップを使ってオブジェクトの位置を特定します。GUI マップに格納されているオブジェクトの記述を読み取り，それらと同じプロパティを持つテスト対象アプリケーションのオブジェクトを探します。アプリケーションに含まれているオブジェクトの全体像を把握するために，GUI マップを開いて表示できます。

GUI マップは実際は，1 つまたは複数の GUI マップ・ファイルで構成されています。GUI マップ・ファイルを構成するモードには，次の 2 つがあります。

- アプリケーション全体，またはアプリケーションのウィンドウごとに GUI マップ・ファイルを作成できます。複数のテストから 1 つの共通の GUI マップ・ファイルを参照できます。これは WinRunner の標準のモードです。WinRunner の上級ユーザにとっては，これが最も効率的な作業方法です。グローバル GUI マップ・ファイル・モードでの作業については，第 5 章「グローバル GUI マップ・ファイル・モードでの作業」を参照してください。
- WinRunner は，作成するテストごとに GUI マップ・ファイルを自動的に作成できます。そのため，GUI マップ・ファイルの作成，保存，ロードについて心配する必要がありません。WinRunner を使い慣れていない場合は，この方法が最も単純です。テスト特有の GUI マップ・ファイル・モードでの作業の詳細については，第 6 章「テスト特有の GUI マップ・ファイル・モードでの作業」を参照してください。

  テスト工程のどの段階でも，テスト特有の GUI マップ・ファイル・モードからグローバル GUI マップ・ファイル・モードに切り換えられます。詳細については，第 23 章「GUI マップ・ファイルのマージ」を参照してください。

  アプリケーションのユーザ・インタフェースが変わっても，以前に開発されたテストを使い続けることができます。GUI マップのオブジェクト記述を追加，削除，あるいは編集するだけで，WinRunner は，修正したアプリケーションでオブジェクトを探すことができます。詳細については，第 7 章「GUI マップの編集」を参照してください。

  WinRunner がオブジェクトの特定のクラスの識別するのに使用するプロパティを指定できます。WinRunner に，ユーザ定義オブジェクトを識別してそのオブジェクトを標準クラスにマップさせることもできます。詳細については，第 24 章「GUI マップの構成設定」を参照してください。

  また，ビットマップを仮想オブジェクトと定義することで，ウィンドウの任意のビットマップを GUI オブジェクトとして認識させることができます。詳細については，第 25 章「仮想オブジェクトの学習」を参照してください。

テストが GUI オブジェクトを識別する方法

テストは，「**テスト・スクリプト**」を記録またはプログラミングして作成します。テスト・スクリプトは，Mercury 独自のテスト・スクリプト言語（TSL）のステートメントで構成されています。各 TSL ステートメントは，テスト対象アプリケーションへのマウスまたはキーボード入力を表します。詳細については，第 8 章「テストの設計」を参照してください。

WinRunner は，各オブジェクトを識別するために，直観的な「**論理名**」を使用します。例えば，［印刷］ダイアログ・ボックスには「印刷」，［OK］ボタンには「OK」という名前を使います。論理名は「**物理的記述**」のニックネームのようなものです。物理的記述には，オブジェクトの一連の物理的なプロパティ，つまり属性が含まれています。例えば，［印刷］ダイアログ・ボックスは，「印刷」というラベルを持つウィンドウとして識別されます。論理名と物理的記述を利用して，各 GUI オブジェクトを一意に識別できます。

物理的記述

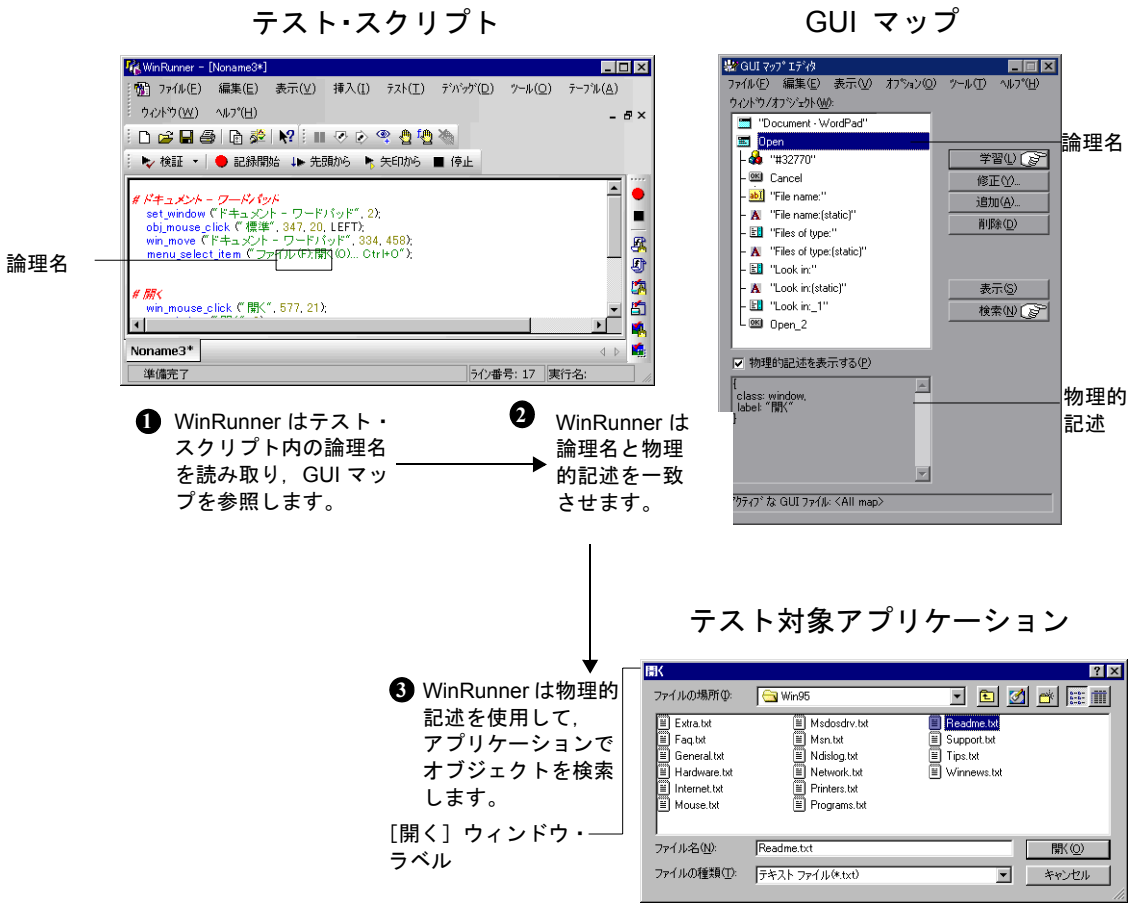
WinRunner は，オブジェクトの「**物理的記述**」に基づいて，テスト対象アプリケーションの各 GUI オブジェクトを識別します。物理的記述とは，一連の物理的プロパティとそれらの値のことです。これらの属性と値の対は，GUI マップの中で以下の形式で記録されます。

{*property1:value1, property2:value2, property3:value3, ...*}

例えば，［開く］ウィンドウの記述には 2 つの属性があります。class と label です。この場合，class プロパティは *window* という値を持ち，label プロパティは「**開く**」という値を持っています。

{class:window, label: 開く }

class プロパティは，オブジェクトの種類を示します。各オブジェクトは，その機能に応じて異なるクラスに属します。window，push button，list，radio button，menu などのクラスがあります。



各クラスには，一連の標準のプロパティがあります。WinRunner は，これらの属性を必ず学習します。すべての属性の詳細については，第 24 章「GUI マップの構成設定」を参照してください。

WinRunner は，オブジェクトの物理的記述を，必ずオブジェクトが現れるウィンドウのコンテキストにおいて学習します。これにより，各オブジェクトについて一意の物理的記述が作成されます。詳細については，本章の 31 ページ「ウィンドウのコンテキストの設定」を参照してください。

注：WinRunner は常にウィンドウのコンテキスト内でオブジェクトを識別しますが，ウィンドウの物理的記述はそれが含まれるオブジェクトには依存しません。

論理名

WinRunner は，テスト・スクリプトの中では，オブジェクトの完全な物理的記述を利用するわけではありません。代わりに，各オブジェクトに対して「**論理名**」という短く，分かりやすい名前を割り当てます。

オブジェクトに割り当てられる論理名は，そのオブジェクトのクラスによって決まります。ほとんどの場合，オブジェクトのラベルが論理名となります。例えば，ボタンの論理名は，［OK］や［キャンセル］などのラベルが論理名となります。また，ウィンドウの場合は，そのタイトル・バーのテキストが論理名となります。さらに，リストの場合は，リストの横または上に表示されるテキストが論理名となります。

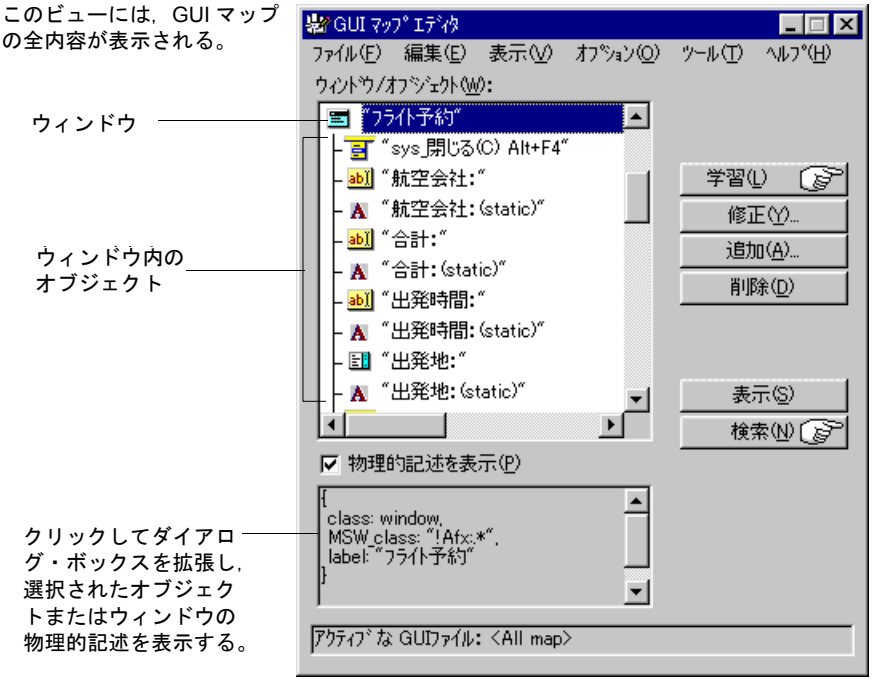
静的テキスト・オブジェクトの場合，テキストと「:(static)」という文字列をつなげたものとなります。例えば，「ファイル名 (N)」という静的テキストの論理名は，「ファイル名 (N):(static)」です。

場合によっては，同じウィンドウ内のいくつかの GUI オブジェクトに同じ論理名とロケーション・セレクタが割り当てられることがあります（例えば，LogicalName_1，LogicalName_2 など）。セレクタのプロパティは，オブジェクトに一意の名前を与えるためにあります。

GUI マップ

GUI マップの内容は，［**ツール**］＞［**GUI マップ エディタ**］を選択すればいつでも表示できます。GUI マップは，1 つまたは複数の GUI マップ・ファイルで構成されています。

［GUI マップ エディタ］では，GUI マップ全体の内容，または個々の GUI マップ・ファイルの内容を表示できます。GUI オブジェクトは，アプリケーションのウィンドウごとにグループ分けされます。



GUI マップ・ファイルには，GUI オブジェクトの論理名と物理的記述が含まれる。

GUI マップ・エディタの詳細については第 7 章「GUI マップの編集」を参照してください。

GUI マップ・ファイルを構成するモードには，次の 2 つがあります。

- **グローバル GUI マップ・ファイル・モード**：アプリケーション全体，またはアプリケーションのウィンドウごとに GUI マップ・ファイルを作成できます。異なる複数のテストから 1 つの共通の GUI マップ・ファイルを参照できます。詳細については，第 5 章「グローバル GUI マップ・ファイル・モードでの作業」を参照してください。
- **テスト特有の GUI マップ・ファイル・モード**：WinRunner は，作成する各テストに対応する GUI マップ・ファイルを自動的に作成します。詳細については，第 6 章「テスト特有の GUI マップ・ファイル・モードでの作業」を参照してください。

各モードの短所と長所については，42 ページ「使用する GUI マップ・ファイル・モードの決定」を参照してください。

ウィンドウのコンテキストの設定

WinRunner は，オブジェクトの学習と操作を，そのオブジェクトが現れるウィンドウのコンテキストにおいて行います。テストの記録の際，アクティブなウィンドウが変わって，GUI オブジェクトが操作されると，WinRunner はそのたびに，テスト・スクリプトに **set_window** ステートメントを自動的に挿入します。これにより，すべてのオブジェクトが，そのウィンドウのコンテキストにおいて識別されます。以下の例を見てください。

```
set_window (" 印刷 ", 12);
button_press ("OK");
```

set_window ステートメントは，［印刷］ウィンドウがアクティブなウィンドウであることを示します。［OK］ボタンは，このウィンドウのコンテキストで学習されます。

テストを手作業でプログラミングする場合は，アクティブなウィンドウが変わる時点で **set_window** ステートメントをユーザ自身が入力する必要があります。スクリプトを編集する際には，必要な **set_window** ステートメントを削除してしまわないよう注意してください。

第 4 章

GUI マップの基本概念について

本章では，WinRunner がアプリケーションのグラフィカル・ユーザ・インタフェース（GUI）を識別する方法と，GUI マップ・ファイルの使い方について説明します。

本章では，以下の項目について説明します。

- GUI マップの概要
- GUI オブジェクトのプロパティの表示
- WinRunner によるアプリケーションの GUI の学習
- GUI マップでのオブジェクトまたはウィンドウの検索
- GUI マップ・ファイルの使い方に関する一般的なガイドライン
- 使用する GUI マップ・ファイル・モードの決定

GUI マップの概要

WinRunner はテストの実行時に，アプリケーションでマウス・カーソルを移動したり，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ことによって，ユーザの動作をシミュレートします。実ユーザと同様に，WinRunner はアプリケーションの GUI を処理するために，それらを学習する必要があります。

このため，WinRunner はアプリケーションの GUI オブジェクトとそのプロパティを学習し，GUI マップにこれらのオブジェクトの物理的記述を保存します。デスクトップ上の GUI オブジェクトのプロパティを表示して，WinRunner の識別方法を確認するには，GUI スパイを使用します。

WinRunner は，次の方法でアプリケーションの GUI 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ウィンドウに含まれるすべての GUI オブジェクトのプロパティを学習する。
- アプリケーションで記録を行って，記録したすべての GUI オブジェクトのプロパティを学習する。
- GUI マップ・エディタを使って，各 GUI オブジェクト，ウィンドウ，ウィンドウ内のすべての GUI オブジェクトのプロパティを学習する。

  アプリケーションの GUI が，ソフトウェアの開発工程で変更された場合は，GUI マップ・エディタを使って各ウィンドウとオブジェクトを学習し，GUI マップを更新できます。

  WinRunner にアプリケーションの GUI を学習させる前に，GUI マップ・ファイルの編成方法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**］モードでは，新規作成した各テストに対して，新しい GUI マップ・ファイルが自動的に作成されます。
- ［**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。

  どちらのモードを使用したほうが良いかについては，本章の最後で説明します。

GUI オブジェクトのプロパティの表示

WinRunner は，GUI オブジェクトの記述を学習するときに，オブジェクトの物理プロパティを見ます。各 GUI オブジェクトには，「クラス」，「ラベル」，「幅」，「高さ」，「ハンドル」，「有効」など，多くのプロパティがあります。ただし，WinRunner はアプリケーション内の特定のオブジェクトをその他のオブジェクトと区別するために，選択されたプロパティのセットだけを学習します。

アプリケーションの GUI マップを作成する前，または GUI マップに GUI オブジェクトを追加する前に，GUI オブジェクトのプロパティを確認できます。GUI スパイを使えば，デスクトップ上の任意のオブジェクトのプロパティを確認できます。GUI スパイのポインタでオブジェクトを指すと，［GUI スパイ］ダイアログ・ボックスにプロパティとその値が表示されます。

オブジェクトのすべてのプロパティを表示したり，WinRunner が学習する対象として選択されたプロパティのセットだけを表示したりできます。

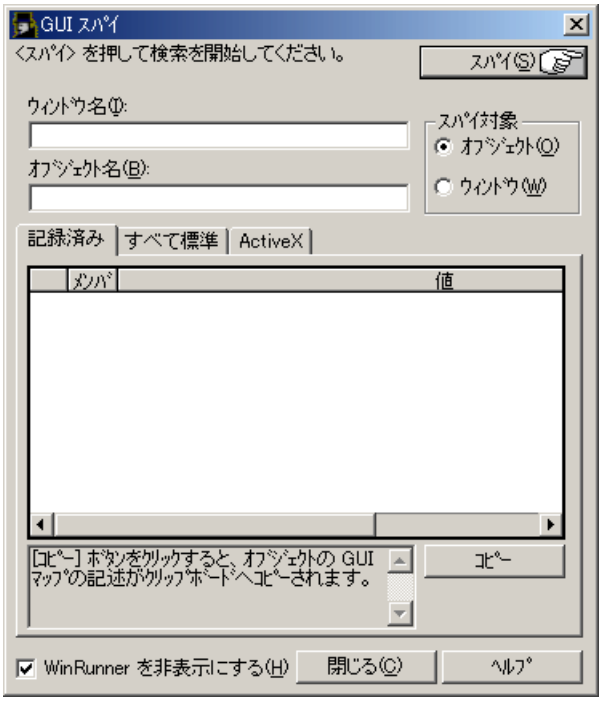
例えば，サンプルのフライト・アプリケーションの［ログイン］ウィンドウに含まれる［代理店名］編集ボックスを指した場合，［GUI スパイ］の［**すべて標準**］タブは，次のように表示されます。

ヒント：すべての内容を一度で確認できるように，［GUI スパイ］のサイズを変更できます。

注：［ActiveX］タブは，ActiveX アドインがインストールされ，ロードされた場合にのみ表示されます。

GUI オブジェクト・ウィンドウを探索するには，次の手順を実行します。

1 ［**ツール**］＞［**GUI スパイ**］を選択すると，［GUI スパイ］ダイアログ・ボックスが開きます。



標準設定では，［GUI スパイ］の［記録済み］タブが表示されます。このタブには，WinRunner が記録または学習した標準 GUI オブジェクトのプロパティが表示されます。

➤ 標準のウィンドウとオブジェクトのすべてのプロパティを表示するには，［**すべて標準**］タブをクリックします。

➤ ActiveX コントロールのすべてのプロパティとメソッドを表示するには，［**ActiveX**］タブをクリックします（ActiveX アドインがインストールされ，ロードされた場合のみ）。

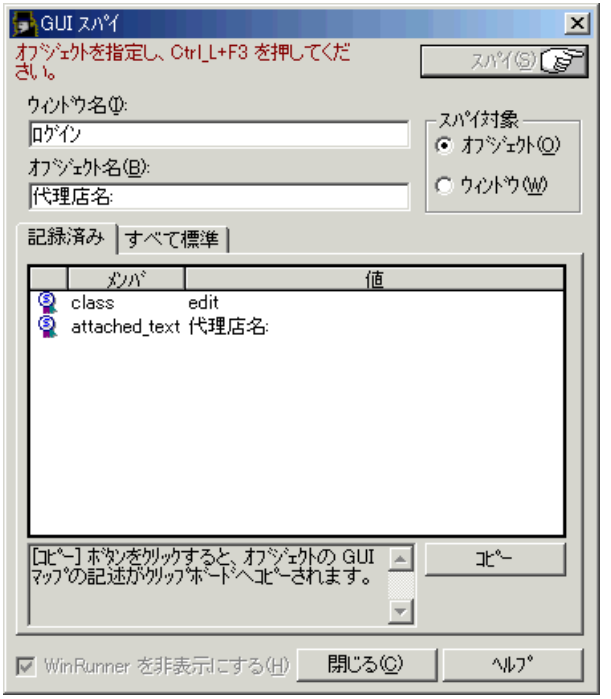
2 ［スパイ対象］ボックスで，［**オブジェクト**］または［**ウィンドウ**］を選択します。

3 オブジェクトを探索中に WinRunner 画面（GUI スパイではなく）を非表示にするには，［**WinRunner を非表示にする**］を選択します。

4 ［**スパイ**］をクリックし，画面上のオブジェクトを指します。オブジェクトが強調表示され，アクティブなウィンドウの名前，オブジェクトの名前と記述（プロパティとその値）が，対応するフィールドに表示されます。

その他のオブジェクトにポインタを移動すると，各オブジェクトが順番に強調表示され，その記述が記述表示枠に表示されます。

サンプルのフライト・アプリケーションの［ログイン］ウィンドウに含まれる［代理店名］編集ボックスを指した場合，［GUI スパイ］の［**記録済み**］タブは，次のように表示されます。



5 ［GUI スパイ］ダイアログ・ボックスのオブジェクトの記述をキャプチャするには，任意のオブジェクトを指し，STOP ショートカット・キーを押します（標準のショートカット・キーの組み合わせは左 Ctrl+F3 です）。

オブジェクトのスパイを開始する前に［**WinRunner を非表示にする**］を選択した場合は，STOP ソフトキーを押すと WinRunner 画面が再び表示されます。

➤ ［**記録済み**］タブと［**すべて標準**］タブで，オブジェクトの物理的記述をクリップボードにコピーするには，［**コピー**］ボタンをクリックします。

前の例で［コピー］をクリックすると，次の物理的記述がクリップボードに貼り付けられます。

```
{class: "edit", attached_text: " 代理店名 : "}
```

ヒント：CTRL + C を押して，選択した行からプロパティと値をクリップボードにコピーできます。

➤ ［**ActiveX**］タブで，記述が含まれているプロパティを強調表示すると，その記述が下部のグレイの表示枠に表示されます。この ActiveX コントロールのヘルプ・ファイルがインストールされている場合，［**項目ヘルプ**］をクリックすると，このヘルプ・ファイルが表示されます。

Visual Basic で記述されているサンプルのフライト・アプリケーションに含まれる［フライトテーブル］を指し，STOP ショートカット・キーを押し，「FixedAlignment」プロパティを強調表示させた場合，［GUI スパイ］の［**ActiveX**］タブは，次のように表示されます。

注：ActiveX プロパティ値が他のオブジェクトへのポインタ（参照）であり，そのオブジェクトにコントロール・ベンダによって標準設定として指定されたプロパティがある場合，GUI スパイはポインタの値ではなく標準設定のプロパティの値を表示します。ただし，ポインタ値を含むプロパティの **ActiveX_get_info** 関数を使用している場合は，**PropA.PropB** 形式でプロパティを指定する必要があります。

例えば，ActiveX list オブジェクトに，リスト項目を表す他のオブジェクトへのポインタである値を持つ **SelectedItem** プロパティがある場合，リスト項目の標準設定のプロパティがテキスト・プロパティであれば，GUI スパイはテキスト・プロパティの値（ABC など）を表示します。

ActiveX_get_info 関数を使用する場合，

ActiveX_get_info("LogName", "SelectedItem", RetVal)

は，Object Reference - 0x782e789f などのポインタ値を返します。

ActiveX_get_info("LogName", "SelectedItem.Text", RetVal)
は，ABC などのテキストのプロパティを返します。

---

6 ［GUI スパイ］を閉じるには，［**閉じる**］をクリックします。

WinRunner によるアプリケーションの GUI の学習

ユーザと同様に，WinRunner はアプリケーションの GUI を処理するために，それらを学習する必要があります。

- アプリケーションで記録を行って，記録したすべての GUI オブジェクトのプロパティを学習する。
- GUI マップ・エディタの［**学習**］ボタンをクリックして，各 GUI オブジェクト，ウィンドウ，ウィンドウ内のすべての GUI オブジェクトのプロパティを学習する。
- テスト・スクリプト・ウィザードを使用して，アプリケーションのすべての GUI オブジェクトのプロパティを学習する。

注 :［**テスト特有の 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・ウィザードは使用できません。また，テスト・スクリプト・ウィザードは，WebTest または特定のアドインがロードされている場合は使用できません。現在使用しているアドインでテスト・スクリプト・ウィザードをしようできるかどうかについては，お使いのアドインのマニュアルを参照してください。

［**グローバルな GUI マップ ファイル**］モードで作業する場合，上記の 3 つの方法を使用するだけでなく，最初にいくつかの管理ステップを実行する必要があります。例えば，永久 GUI マップ内のオブジェクトを保存し，テストの実行時にこのファイルがロードされたことを確認する必要があるとします。詳細については，第 5 章「グローバル GUI マップ・ファイル・モードでの作業」を参照してください。ただし，［**テスト特有の GUI マップ ファイル**］モードでは余計なステップを行う必要はありません。WiRunner は管理タスクを自動的に行います。

前述の方法で WinRunner にアプリケーションの GUI を学習させる場合の，その他の情報については，第 5 章「グローバル GUI マップ・ファイル・モードでの作業」および 第 6 章「テスト特有の GUI マップ・ファイル・モードでの作業」を参照してください。

GUI マップでのオブジェクトまたはウィンドウの検索

GUI オブジェクトまたはウィンドウを参照するテスト・スクリプトのステートメントにカーソルがあるときに，右クリックして［**GUI マップ内で検索**］を選択します。

WinRunner は指定されたオブジェクトまたはウィンドウを，GUI マップか GUI マップ・ファイル内，および開いているアプリケーション内で検索し，強調表示します。

- ウィンドウを含む GUI マップ・ファイルがロードされており，指定されたウィンドウが開いている場合，［GUI マップ エディタ］が開き，GUI マップとアプリケーションでそのウィンドウが強調表示されます。
- オブジェクトを含む GUI マップ・ファイルがロードされており，指定されたオブジェクトを含むウィンドウが開いている場合，［GUI マップ エディタ］が開き，GUI マップとアプリケーションでそのオブジェクトが強調表示されます。
- オブジェクトまたはウィンドウを含む GUI マップ・ファイルがロードされており，そのオブジェクトまたはウィンドウを含むアプリケーションが開いていない場合，［GUI マップ エディタ］が開き，GUI マップでそのオブジェクトまたはウィンドウが強調表示されます。

GUI マップ・ファイルの使い方に関する一般的なガイドライン

GUI マップ・ファイルを使用するときは，次のガイドラインに留意してください。

- 1 つの GUI マップ・ファイルに，同じ論理名を持つ 2 つのウィンドウを入れることはできません。
- GUI マップ・ファイル内の 1 つのウィンドウに，同じ論理名を持つ 2 つのオブジェクトを入れることはできません。
- ［GUI マップ エディタ］では，［**オプション**］＞［**フィルタ**］コマンドを使って［フィルタ］ダイアログ・ボックスを開き，GUI マップ内のオブジェクトを論理名，物理的記述，またはクラスでフィルタリングすることができます。詳細については，89 ページ「表示されるオブジェクトのフィルタ処理」を参照してください。

使用する GUI マップ・ファイル・モードの決定

テストを計画して作成するときは，GUI マップの使い方について検討する必要があります。各テストに対して 1 つの GUI マップ・ファイルを使用するか，複数のテストに対して共通の GUI マップ・ファイルを使用できます。

- WinRunner またはテストに慣れていない場合は，**テスト特有の GUI マップ・ファイル・**モードで作業することをお勧めします。このモードでは，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。テストを保存するたびに，そのテストに対応する GUI マップ・ファイルが自動的に保存され，テストを開くたびに，その GUI マップ・ファイルが自動的にロードされます。
- 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・**モードで作業するほうが効果的です。これは，WinRunner の標準モードです。

次の表に，各モードで作業した場合の利点と欠点をまとめます。

	[テスト特有の GUI マップ ファイル]	[グローバルな GUI マップ ファイル]
メソッド	WinRunner は，記録時にアプリケーションの GUI を学習し，各テストに対応する GUI マップ・ファイルにこの情報を自動的に保存します。テストを開くと，対応する GUI マップ・ファイルが自動的にロードされます。	記録する前に，[GUI マップ エディタ] の [学習] ボタンをクリックし，アプリケーションのウィンドウをクリックして，WinRunner にアプリケーションを学習させます。アプリケーション内のすべてのウィンドウに対して，この工程を繰り返します。各ウィンドウまたはウィンドウのセットごとに，個別の GUI マップ・ファイルを保存します。テストの実行時に GUI マップ・ファイルをロードします。アプリケーションが変更されたら，GUI マップ・ファイルを更新します。
利点	1. 各テストに対して，固有の GUI マップ・ファイルがあります。 2. テストまたは WinRunner に慣れていないユーザは，GUI マップ・ファイルの保存またはロードを忘れることがあるので，このモードを使用するほうが簡単です。 3. 各テストを簡単に保守および更新できます。	1. オブジェクトまたはウィンドウの記述が変更された場合，1 つの GUI マップ・ファイルを変更するだけで，そのファイルを参照しているすべてのテストを正しく実行できます。 2. 一連のテストを簡単かつ効率的に保守および更新できます。

	[テスト特有の GUI マップ ファイル]	[グローバルな GUI マップ ファイル]
欠点	アプリケーションの GUI が変更された場合，テストを正しく実行できるように，各テストの GUI マップを個別に更新する必要があります。	GUI マップ・ファイルを保存およびロードしなければなりません。また，起動テストなどのテストに GUI マップ・ファイルをロードするステートメントを追加しなければなりません。
推奨するメソッド	テストまたは WinRunner に慣れていない場合や，アプリケーションの GUI が変更されないと考えられる場合は，このメソッドを使用することをお勧めします。	テストまたは WinRunner に慣れている場合や，アプリケーションの GUI が変更される可能性がある場合は，このメソッドを使用することをお勧めします。

注：オブジェクトの論理名が，そのオブジェクトを説明するような名前でないことがあります。記録の前に［GUI マップ エディタ］を使ってアプリケーションを学習する場合，GUI マップのオブジェクトを強調表示させ，［修正］ボタンをクリックすることで，そのオブジェクト名を分かりやすい名前に修正できます。これによって，WinRunner がアプリケーションを記録したときに，新しい名前がテスト・スクリプトに表示されます。オブジェクトの論理名の修正の詳細については，77 ページ「論理名と物理的記述の修正」を参照してください。

［グローバルな GUI マップ ファイル］モードで作業する場合の，その他のガイドラインについては，63 ページ「グローバル GUI マップ・ファイル・モードで作業する場合のガイドライン」を参照してください。

第 5 章

グローバル GUI マップ・ファイル・モードでの作業

本章では，**グローバルな GUI マップ・ファイル**・モードで作業を行っている際に，GUI マップの情報を保存する方法について説明します。これは WinRunner に標準で設定されているモードです。より簡単な**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，この章をスキップして，第 6 章「テスト特有の GUI マップ・ファイル・モードでの作業」に進んでください。

本章では，以下の項目について説明します。

- ［グローバルな GUI マップ ファイル］モードについて
- テスト間での GUI マップ・ファイルの共有
- アプリケーションの GUI の学習
- GUI マップの保存
- GUI マップ・ファイルのロード
- グローバル GUI マップ・ファイル・モードで作業する場合のガイドライン

［グローバルな GUI マップ ファイル］モードについて

WinRunner で最も効果的に作業するには，テスト・スイート設計時に，テストをグループにまとめることです。グループ内の各テストは，アプリケーションの同一の GUI オブジェクトをテストするため，共通のリポジトリ内の GUI オブジェクトに関する情報を参照します。アプリケーションの GUI オブジェクトが変更されたら，GUI マップ・ファイルに関連する情報だけを更新します。このオブジェクトの情報を各テストで更新する必要はありません。上記の方法で作業する場合は，**グローバルな GUI マップ・ファイル**・モードを使用します。

あるテスト・グループのテストがウィンドウの複数のオブジェクトをテストしている間も，その中の特定の GUI オブジェクトを同じグループ内の別のテストがテストできます。そのため，記録だけでアプリケーションの GUI を WinRunner に学習させると，GUI マップ・ファイルは，ウィンドウのオブジェクトすべてに対する包括的なリストを含むことができなくなる場合があります。WinRunner でテストの記録を開始する前に，アプリケーションの GUI を包括的に学習させることをお勧めします。

WinRunner でアプリケーションの GUI を学習する方法は何通りかあります。通常は，テストを始める前に，テスト・スクリプト・ウィザードを使用して，アプリケーションのすべての GUI オブジェクトを一度に学習します。こうすることで，どのコンテキスト・センシティブ・テストでも利用できる，完全で体系的な基礎情報を持つことができます。GUI オブジェクトの記述は，GUI マップ・ファイルに保存されます。これらのファイルはテスト担当の全員で共有できるので，各ユーザが GUI を個別に学習し直す必要はありません。

ソフトウェア開発工程でアプリケーションの GUI が変更された場合は，［GUI マップ エディタ］を使用して，ウィンドウやオブジェクトを個別に学習して GUI マップを更新できます。記録の際にもオブジェクトを学習できます。テストの記録を開始すれば，WinRunner は，アプリケーションで使われた各 GUI オブジェクトのプロパティを学習します。この方法は簡単なため，ユーザは直ちにテスト・スクリプトの作成を始めることができます。

GUI マップ・ファイルはテストに依存しないので，テストを閉じるときに，自動的には保存されません。変更を保存したい場合は，GUI マップ・ファイルを保存してください。

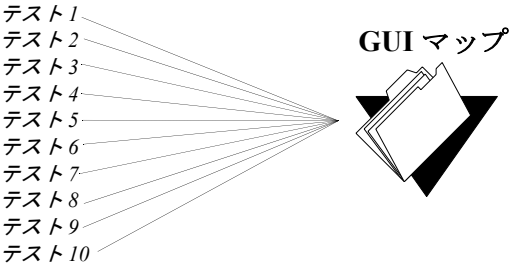
GUI マップ・ファイルはテストに依存しないので，テストを開くときも自動的にはロードされません。したがって，テストを実行する前に，適切な GUI マップ・ファイルをロードしておく必要があります。WinRunner は，これらのファイルを使って，テスト対象アプリケーション内のオブジェクトを識別します。起動テストの中に **GUI_load** ステートメントを入れておくと最も効率的です。WinRunner を起動すると，初期化テストが自動的に実行され，指定された GUI マップ・ファイルがロードされます。初期化テストの詳細については，第 45 章「特殊な構成の初期化」を参照してください。

別の方法として，個々のテストに **GUI_load** ステートメントを埋め込む方法と，GUI マップ・エディタを使って GUI マップ・ファイルを手作業でロードする方法があります。

注：グローバル GUI マップ・ファイル・モードで作業をしているときに，**テスト特有の GUI マップ・ファイル・**モードで作成された GUI オブジェクトを参照しているテストを呼び出すと，そのテストは正しく動作しない場合があります。

テスト間での GUI マップ・ファイルの共有

1 つの GUI マップ・ファイルを複数のテストで共有するよう，テスト・スイートを設計する場合，テスト対象アプリケーションのユーザ・インタフェースに対して行った変更を簡単に反映できます。テスト・スイート全体を編集しなくても，GUI マップの関連するオブジェクトの記述だけを更新すればよいのです。



例えば，［開く］ダイアログ・ボックスの［開く］ボタンを［OK］ボタンに変更するとします。この［開く］ボタンを使うすべてのテスト・スクリプトを編集する必要はありません。GUI マップの［開く］ボタンの物理的記述を下に示すように変更します。ボタンのラベルのプロパティ値が［開く］から［OK］に変わります。

［開く］ボタン：{class:push_button, label:OK}

WinRunner は，テスト実行中にテスト・スクリプト内の［開く］ダイアログ・ボックスの［開く］という論理名を見つけると，［OK］というラベルの付いたプッシュ・ボタンを探します。

テスト工程のどの段階でも，GUI マップ・エディタを使って，GUI オブジェクトの論理名や物理的記述を変更できます。また，実行ウィザードを使って，テスト実行中に GUI マップを更新することもできます。WinRunner がアプリケーション内のオブジェクトの場所を特定できないと，実行ウィザードが自動的に開きます。詳細については，第 7 章「GUI マップの編集」を参照してください。

注 : ［GUI マップ構成設定］ダイアログ・ボックスを使用して，特別なオブジェクトに対して WinRunner が学習する一連のプロパティを変更できます。GUI マップの構成設定の詳細については，第 24 章「GUI マップの構成設定」

アプリケーションの GUI の学習

WinRunner は，GUI マップ・ファイルに情報を追加するため，アプリケーションの GUI オブジェクトの情報を学習しなくてはなりません。WinRunner は，次の方法で GUI オブジェクトのプロパティに必要な情報を学習できます。

- 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ウィンドウのすべての GUI オブジェクトのプロパティを WinRunner に学習させます。
- アプリケーションで記録を行い，記録したすべての GUI オブジェクトのプロパティを WinRunner に学習させます。
- GUI マップ・エディタを使って，個々の GUI オブジェクト，ウィンドウ，あるいはウィンドウのすべての GUI オブジェクトのプロパティを WinRunner に学習させます。

テスト・スクリプト・ウィザードを使った GUI の学習

テストを開始する前に，テスト・スクリプト・ウィザードを使用して，アプリケーションのすべての GUI オブジェクトを一度に WinRunner に学習させることができます。これにより WinRunner に，すべてのコンテキスト・センシティブ・テストに対して，完全で体系的な基礎情報が提供されます。GUI オブジェクトの記述は，GUI マップ・ファイルに保存されます。これらのファイルはテストのユーザ全員で共有できるので，各ユーザが GUI を個別に学習し直す必要がありません。

注：テスト・スクリプト・ウィザードは，**グローバル GUI マップ・ファイル・**モード（本章で説明した標準のモードです）で作業しているときだけ使用できます。WinRunner の 6.02 以前のバージョンで作成されたテストはすべてこのモードで作成されています。

テスト特有の GUI マップ・ファイル・モードで作業する場合，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，WebTest またはその他の特定のアドインがロードされている場合は使用できません。使用しているアドインでテスト・スクリプト・ウィザードを使用できるかどうかについては，アドインのマニュアルを参照してください。

テスト・スクリプト・ウィザードを使えば，WinRunner でテスト対象アプリケーション内のすべてのウィンドウとオブジェクトを一度に学習できます。このウィザードは，アプリケーションの各ウィンドウを系統立てて開き，アプリケーションに含まれているすべての GUI オブジェクトのプロパティを学習します。この他にも，WinRunner は，個々のオブジェクトのプロパティを学習する方法をいくつか提供します。

WinRunner は学習が終わると，GUI マップ・ファイルにその情報を保存します。また，スタートアップ・スクリプトも作成します。スタートアップ・スクリプトには，GUI マップ・ファイルを読み込む **GUI_load** コマンドも含まれています。スタートアップ・テストの詳細については，第 45 章「特殊な構成の初期化」を参照してください。

テスト・スクリプト・ウィザードを使って WinRunner にアプリケーションを学習させるには，次の手順を実行します。

1 ［**挿入**］＞［**テスト スクリプト ウィザード**］を選択します。テスト・スクリプト・ウィザードの［ようこそ］画面が開きます。

［次へ］をクリックします。

注：テスト・スクリプト・ウィザードのオプションは，WinRunner の実行のみのバージョンを使用している場合，**テスト特有の GUI マップ・ファイル**・モードで作業している場合，また，WebTest アドインなど特定のアドインをロードしている場合は使用できなくなります。アドインをロードする際に，アドインがロードされてもテスト・スクリプト・ウィザードが使用できるかどうか，アドインのマニュアルを参照してください。

---

2 **アプリケーションを指定する**画面が開きます。

指差しポインタをクリックしてからアプリケーションをクリックし，スクリプト・ウィザードがアプリケーションを特定できるようにします。クリックしたウィンドウ名が［ウィンドウ名］ボックスに表示されます。［**次へ**］をクリックします。

3 テスト選択画面が開きます。

4 WinRunner で作成するテストの種類を選択します。スクリプト・ウィザードがアプリケーションを調べ終わると，WinRunner のウィンドウに選択したテストが表示されます。

テストは次の中から選択できます。

- ［**GUI 回帰テスト**］：アプリケーションの異なるバージョン間で，GUI オブジェクトの状態を比較できます。例えば，ボタンが有効か無効かを検査できます。

  GUI 回帰テストを作成するため，ウィザードはアプリケーションの各 GUI オブジェクトに関する標準情報をキャプチャします。アプリケーションでテストを実行すると ,WinRunner は GUI オブジェクトのキャプチャされた状態と現在の状態を比較して不一致をすべて報告します。

- ［**ビットマップ回帰テスト**］：アプリケーションの異なるバージョン間で，アプリケーションのビットマップ・イメージを比較できます。GUI オブジェクトが含まれないアプリケーションをテストする場合は，このテストを選択します。

  ビットマップ回帰テストを作成するため，ウィザードはアプリケーションの各ウィンドウのビットマップ・イメージをキャプチャします。テストを実行すると，WinRunner はキャプチャしたウィンドウのイメージと現在のウィンドウを比較して不一致をすべて報告します。

- ［**ユーザ インタフェース テスト**］：アプリケーションが Microsoft Windows の標準に対応できているか測定します。次の項目を検査します。

  - GUI オブジェクトがウィンドウ内で整列しているか
  - 定義されているテキストがすべて GUI オブジェクトに表示されているか
  - GUI オブジェクトのラベルが大文字で表示されているか
  - 各ラベルに下線付きの文字が含まれているか（ニーモニック）
  - 各ウィンドウに［OK］ボタン，［キャンセル］ボタン，システム・メニューが含まれているか

  このテストを実行すると，WinRunner はアプリケーションのインタフェースを探し，Microsoft Windows の標準に対応していない事項を報告します。

➤ [**テスト テンプレート**]：アプリケーションを操作する自動テストの基本となる枠組みを提供します。このテストは，各ウィンドウを開いたり閉じたりして記録を行いますが，ウィンドウを検査するコードを記録やプログラミングによって追加することができます。

ヒント：上で説明したテストを作成しなくても，スクリプト・ウィザードを使ってアプリケーションの GUI を学習することができます。

[**次へ**] をクリックします。

5 [ナビゲーションコントロールの定義] 画面が開きます。

アプリケーションのコントロール類を表わす文字を入力します。[**ウィンドウ毎に，確認のため一時停止する**] チェック・ボックスを選択して，どのオブジェクトが起動して追加のウィンドウを開くか確認するため，テスト・スクリプト・ウィザードにアプリケーションの各ウィンドウで一時停止させることができます。

[**次へ**] をクリックします。

6 [**学習フローの設定**] 画面が開きます。

学習工程を［**速習**］または［**包括学習**］から選択します。［**速習**］をクリックします。WinRunner が 1 回につき 1 つのウィンドウを系統的に学習を開始します。アプリケーションが複雑な場合，学習に数分かかることがあります。

7 ［**アプリケーションの開始**］画面が開きます。

［**はい**］または［**いいえ**］を選択して，WinRunner を呼び出すときはいつもこのアプリケーションを自動的に起動させるかどうか WinRunner に指示します。［**次へ**］をクリックします。

8 ［**ファイルの保存**］画面が開きます。

スタートアップ・スクリプトと GUI マップ・ファイルを格納する場所の完全パスとファイル名を入力するか，標準設定を受け入れます。［**次へ**］をクリックします。

9 ［**おめでとうございます**］画面が開きます。

［**OK**］をクリックして，テスト・スクリプト・ウィザードを閉じます。WinRunner が学習したアプリケーションを元に作成されたテストが，WinRunner のウィンドウに表示されます。

記録による GUI の学習

WinRunner が，アプリケーションでコンテキスト・センシティブ・モード（標準モード）で記録しながら，オブジェクトを学習することもできます。テストの記録を開始するだけで，WinRunner はアプリケーションで使用される各 GUI オブジェクトのプロパティを学習します。この方法で，初心者でも短時間で素早くテスト・スクリプトを作成できます。コンテキスト・センシティブ・モードでの記録については，第 8 章「テストの設計」を参照してください。

テストを記録する際，WinRunner はまず，ユーザが選択したオブジェクトが GUI マップに含まれていることを確認します。含まれていない場合，WinRunner はオブジェクトを学習します。

WinRunner は，学習した情報を一時 GUI マップ・ファイルに追加します。一時 GUI マップ・ファイルへの情報の保存は，WinRunner を終了する前に行ってください。GUI マップの保存については 57 ページ「GUI マップの保存」を参照してください。

ヒント：情報を一時 GUI マップ・ファイルに追加しない場合は，［一般オプション］ダイアログ・ボックスの［一般設定］カテゴリで，一時 GUI マップ・ファイルをロードしないように設定します。詳細については，第 22 章「グローバル・テスト・オプションの設定」を参照してください。

一般に，記録による学習という方法は，規模の小さい暫定的なテストの場合にだけ使用すべきです。アプリケーションの GUI をすべて学習する場合は，テスト・スクリプト・ウィザードか GUI マップ・エディタを使いましょう。

GUI マップ・エディタを使った GUI の学習

GUI マップ・エディタを使って，個々のオブジェクトやウィンドウ，あるいはウィンドウのすべてのオブジェクトを学習することができます。

GUI マップ・エディタを使って WinRunner に GUI オブジェクトを学習させるには，次の手順を実行します。

1 ［**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。

2 ［**学習**］をクリックします。マウス・ポインタが指差し型に変わります。指差し型を学習するオブジェクト上に配置し，マウスの左ボタンをクリックします。

- ウィンドウ内のオブジェクトをすべて学習するには，ウィンドウのタイトル・バーをクリックします。「ウィンドウ内のオブジェクトをすべて学習しますか？」というメッセージに対して［**はい**］（標準）をクリックします。
- ウィンドウだけを学習するには，ウィンドウのタイトル・バーをクリックします。「ウィンドウ内のオブジェクトをすべて学習しますか？」というメッセージに対して，［**いいえ**］をクリックします。
- オブジェクトを学習するには，そのオブジェクトをクリックします。

（操作をキャンセルするには，マウスを右クリックします。）

WinRunner は，一時 GUI マップ・ファイルに学習した情報を追加します。一時 GUI マップ・ファイルへの情報の保存は，WinRunner を終了する前に行わなくてはなりません。

GUI マップの保存

記録による GUI オブジェクトの学習を行った場合，オブジェクトの記述は仮 GUI マップ・ファイルに追加されます。仮ファイルは，常に開いているため，そこに含まれているオブジェクトはすべて WinRunner で認識できます。WinRunner を起動すると，最後にテストを終了したときの内容を含んでいる仮ファイルがロードされます。

新規の記録セッションで，重要な GUI 情報が上書きされないようにするには，仮 GUI マップ・ファイルを恒久的な GUI マップ・ファイルに保存してください。

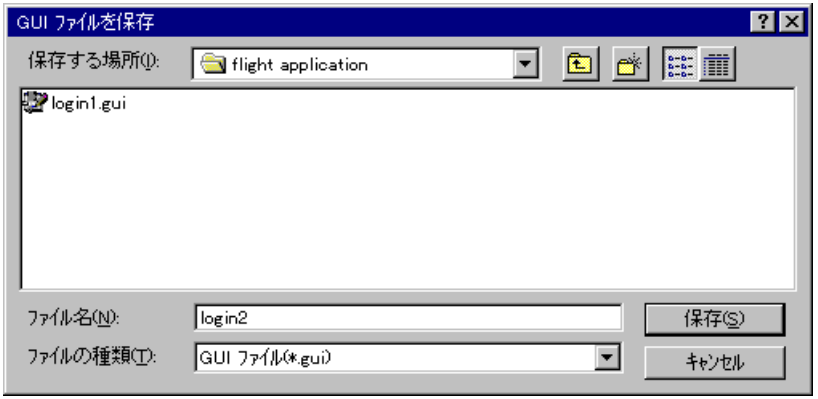
一時 GUI マップ・ファイルの内容を恒久 GUI マップ・ファイルに保存するには，次の手順を実行します。

1 ［**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。

2 ［**表示**］＞［**GUI ファイル**］を選択します。

3 ［GUI ファイル］リストに＜**仮**＞ファイルが表示されていることを確認してください。ファイル名の横にアスタリスク記号（*）が付いている場合，その GUI マップ・ファイルは変更されています。ファイルを保存すると，アスタリスク記号は消えます。

4 GUI マップ・エディタで［**ファイル**］＞［**名前を付けて保存**］を選択して［GUI ファイルを保存］ダイアログ・ボックスを開きます。



5 フォルダをクリックし，新しいファイル名を入力するか，既存のファイルを選択します。

6 [**保存**] をクリックします。保存した GUI マップ・ファイルがロードされ，GUI マップ・エディタに表示されます。

仮ファイルのオブジェクトを既存の GUI マップ・ファイルに移動することも可能です。詳細については，83 ページ「ファイル間でのオブジェクトのコピーと移動」を参照してください。

GUI マップ・ファイルの内容を Quality Center プロジェクトに保存するには，次の手順を実行します。

注 : Quality Center で作業している場合は，GUI マップ・ファイルだけを Quality Center のデータベースに保存できます。詳細については，第 48 章「テスト工程の管理」を参照してください。

---

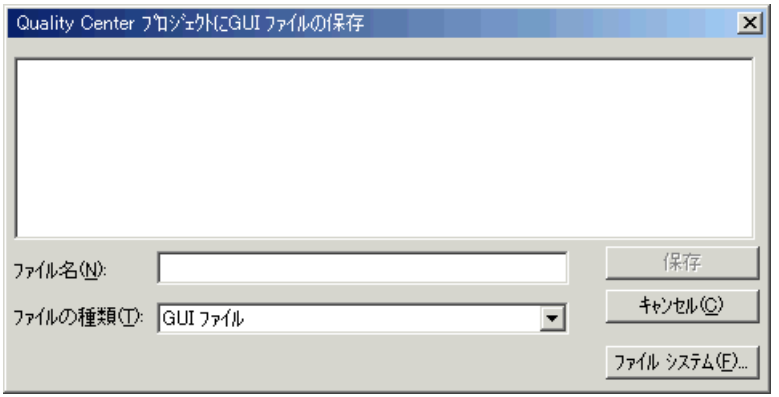
1 [**ツール**] > [**GUI マップ エディタ**] を選択して，GUI マップ・エディタを開きます。

2 [**表示**] > [**GUI ファイル**] を選択します。

3 [GUI ファイル] リストに<**仮**>ファイルが表示されていることを確認してください。ファイル名の横にアスタリスク記号（*）が付いている場合，その GUI マップ・ファイルは変更されています。ファイルを保存すると，アスタリスク記号は消えます。

4 GUI マップ・エディタの [**ファイル**] > [**名前を付けて保存**] を選択します。

［GUI ファイルを Quality Center プロジェクトに保存］ダイアログ・ボックスが開きます。



5 ［**ファイル名**］テキスト・ボックスに GUI マップ・ファイルの名前を入力します。後からでも GUI マップ・ファイルを簡単に識別できるような名前にします。

6 ［**保存**］をクリックして，GUI マップ・ファイルを Quality Center のデータベースに保存して，ダイアログ・ボックスを閉じます。

GUI マップ・ファイルのロード

WinRunner はアプリケーション内のオブジェクトを学習すると，その情報を GUI マップ・ファイルに格納します。WinRunner が GUI マップ・ファイルを使ってアプリケーション内のオブジェクトを見つけられるようにするには，GUI マップ・ファイルを GUI マップに**ロード**しなければなりません。テスト対象アプリケーションのテストを行う前に，該当する GUI マップ・ファイルをロードしてください。

GUI マップは，次のいずれかの方法でロードできます。

- **GUI_load** コマンドを使う方法。
- GUI マップ・エディタでロードする方法。

ロードした GUI マップ・ファイルは，GUI マップ・エディタに表示できます。ロード済みのファイルには，ファイル名の前に「L」という文字と数字が付きます。GUI マップ・ファイルを編集用に開く場合は，ロードしなくても開けます。

注：［**テスト特有の GUI マップ・ファイル**］**モード**で作業している場合は，GUI マップ・ファイルのロード，アンロード，保存を手作業で行わないでください。

GUI_load 関数を使った GUI マップ・ファイルのロード

GUI_load ステートメントは，ユーザが指定した任意の GUI マップ・ファイルをロードします。GUI マップには 1 つあるいは複数の GUI マップ・ファイルを含めることができますが，一度にロードできる GUI マップ・ファイルは 1 つだけです。複数のファイルをロードするには，各ファイルに対し個別のステートメントを使用します。**GUI_load** ステートメントは，テストの開始部に入れることもできますが，初期化テストに入れるほうがよいでしょう。このようにすると，WinRunner を起動するたびに，GUI マップ・ファイルが自動的にロードされます。詳細については，第 45 章「特殊な構成の初期化」を参照してください。

GUI_load を使ってファイルをロードするには，次の手順を実行します。

1 ［ファイル］＞［開く］を選んでファイルをロードするテストを開きます。

2 テスト・スクリプトに次の **GUI_load** ステートメントを入力するか，関数ジェネレータで **GUI_load** 関数をクリックして指定するか，ファイル・パスを入力します。

GUI_load (" *ファイルの完全パス名* ");

以下に例を示します。

```
GUI_load ("c:¥¥qa¥¥flights.gui")
```

関数ジェネレータの使い方については，第 34 章「関数の生成」を参照してください。

3 テストを実行してファイルをロードします。詳細については，第 19 章「テスト実行について」を参照してください。

注：GUI マップ・ファイルの編集だけを行なう場合，**GUI_open** 関数 を使えば，GUI マップ・ファイルをロードせずに開くことができます。開いている GUI マップ・ファイルは，**GUI_close** 関数で閉じることができます。GUI マップ・ファイルの編集については，第 7 章「GUI マップの編集」を参照してください。GUI マップ・ファイルをアンロードするには，**GUI_unload** 関数または **GUI_unload_all** 関数を使います。TSL 関数の使用法の詳細については，第 28 章「プログラミングによるテスト・スクリプトの 機能強化」を参照してください。個々の TSL 関数とその使用例については，「**TSL リファレンス**」を参照してください。

GUI マップ・エディタを使った GUI マップ・ファイルのロード

GUI マップ・エディタを使って，ファイル・システムまたは Quality Center プロジェクトから GUI マップ・ファイルを手作業でロードできます。

注：Quality Center データベースから GUI マップ・ファイルをロードできるのは，if you are connected to a Quality Center プロジェクトに接続されている場合のみです。詳細については，第 48 章「テスト工程の管理」を参照してください。

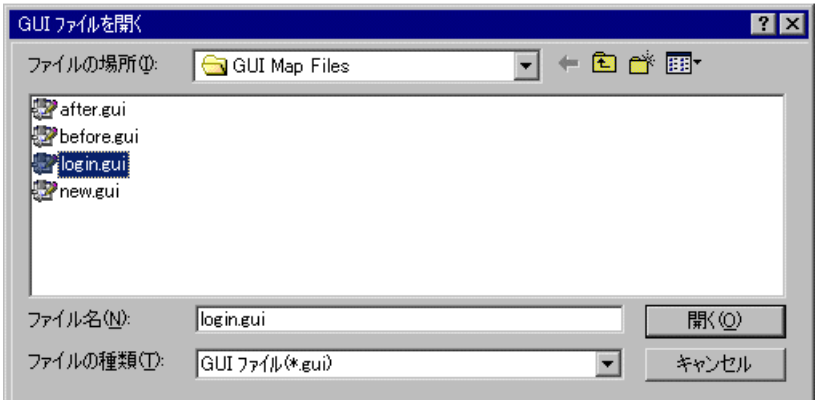
GUI マップ・エディタを使用して，ファイル・システムから GUI マップ・ファイルをロードするには，次の手順を実行します。

1 ［**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。

2 ［**表示**］＞［**GUI ファイル**］を選択します。

3 ［**ファイル**］＞［**開く**］を選択します。

4 ［**GUI ファイルを開く**］ダイアログ・ボックスで，GUI マップ・ファイルを選択します。



ファイルは，標準設定では GUI マップにロードされる点に注意してください。GUI マップ・ファイルを編集したいだけの場合，［**編集のためにのみ開く**］ボタンを選択します。GUI マップ・ファイルの編集については，第 7 章「GUI マップの編集」を参照してください。

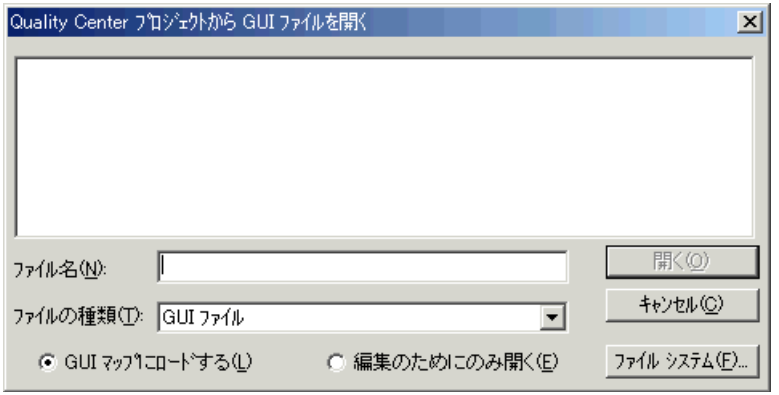
5 ［**開く**］をクリックします。GUI マップ・ファイルが GUI ファイル・リストに追加されます。ファイルがロードされると，「L」という文字と数字がファイル名の前に表示されます。

GUI マップ・エディタを使って Quality Center プロジェクトから GUI マップ・ファイルをロードするには，次の手順を実行します。

1 ［**ツール**］＞［**GUI マップ エディタ**］を選択して，［GUI マップ エディタ］を開きます。

2 ［**ファイル**］＞［**開く**］を選択します。

［Quality Center プロジェクトから GUI ファイルを開く］ダイアログ・ボックスが開きます。選択されたデータベースに保存されているすべての GUI マップ・ファイルが，ダイアログ・ボックスに表示されます。



3 選択されているデータベースの GUI マップ・ファイルの一覧から GUI マップ・ファイルを選択します。GUI マップ・ファイルの名前が，［**ファイル名**］テキスト・ボックスに現れます。

GUI マップ・エディタに GUI マップ・ファイルをロードするには，標準設定の［**GUI マップにロード**］が選択されていることを確認してください。GUI マップ・ファイルの編集だけしたい場合は，［**編集のためにのみ開く**］をクリックします。詳細は，第 7 章「GUI マップの編集」を参照します。

4 ［**開く**］をクリックして，［GUI マップ ファイル］を開きます。GUI マップ・ファイルが GUI ファイル・リストに追加されます。ファイルがロードされていると，ファイル名に「L」という文字が付きます。

グローバル GUI マップ・ファイル・モードで作業する場合のガイドライン

［**グローバル GUI マップ・ファイル**］**モード**で作業する場合は，以下のガイドラインに従ってください。

➤ パフォーマンスを向上させるには，アプリケーションのテスト用の GUI マップは大きいものではなく小さいものを使用してください。アプリケーションのユーザ・インタフェースを，ウィンドウごとまたは他の論理に基づいて個々の GUI マップ・ファイルに分割することもできます。

- オブジェクトの論理名が記述的でない場合もあります。アプリケーションを記録する前に，GUI マップ・エディタを使って学習する場合は，オブジェクトを強調表示して［修正］ボタンを押せば，GUI マップ内のオブジェクトの論理名前をわかりやすいものに変更できます。WinRunner でアプリケーションの記録を行う場合は，テスト・スクリプトに新しい名前が表示されます。GUI マップのオブジェクトの論理名を変更する前にテストを記録してしまったときは，テストを実行する前に，テスト・スクリプトのオブジェクトの論理名を必ず更新するようにしてください。オブジェクトの論理名を変更にする方法の詳細については，77 ページ「論理名と物理的記述の修正」を参照してください。
- WinRunner がアプリケーションの GUI について学習した情報を，一時 GUI マップに保存してはなりません。この情報は，WinRunner を終了すると自動的に保存されます。再使用しない一時的な小さなテストを作成するのでなければ，テストを終了する前に GUI マップを GUI マップ・エディタから保存してください（［**ファイル**］＞［**保存**］を選択）。

ヒント：一時 GUI マップ・ファイルをロードしないよう WinRunner に指示することができます。これは［一般オプション］ダイアログ・ボックスの［一般設定］カテゴリで設定します。このオプションの詳細については，第 22 章「グローバル・テスト・オプションの設定」を参照してください。

- WinRunner が，アプリケーションに対するユーザの操作を記録する方法で GUI を学習する場合は，操作が実行されたオブジェクトしか学習しません。つまり，アプリケーションのすべてのオブジェクトを学習するわけではありません。したがって，再使用する予定のない一時的で小さなテストを作成するのでなければ，記録を開始してからアプリケーションを WinRunner に学習させるよりも，記録を開始する前に GUI マップ・エディタの［学習］ボタンでアプリケーションの GUI を学習させるのが一番良い方法です。
- テスト担当者の中で，アプリケーションの GUI 変更時に GUI マップの更新を行う「GUI マップ管理者」を決めておくことをお勧めします。

GUI マップで作業する場合のガイドラインについては，41 ページ「GUI マップ・ファイルの使い方に関する一般的なガイドライン」を参照してください。

第 6 章

テスト特有の GUI マップ・ファイル・モードでの作業

本章では，［テスト特有の GUI マップ ファイル］モードでの作業方法について説明します。テストまたは WinRunner に慣れていない場合は，このモードを使用することをお勧めします。GUI マップ・ファイルの作成，保存，ロードの方法を理解する必要がないので，このモードは非常に簡単に使用できます。

本章では，以下の項目について説明します。

- ［テスト特有の GUI マップ ファイル］モードについて
- ［テスト特有の GUI マップ ファイル］モードの指定
- ［テスト特有の GUI マップ ファイル］モードでの作業
- ［テスト特有の GUI マップ ファイル］モードでの作業のガイドライン

［テスト特有の GUI マップ ファイル］モードについて

［**テスト特有の GUI マップ ファイル**］モードで作業する場合，WinRunner にアプリケーションの GUI を学習させたり，GUI マップ・ファイルを保存またはロードしたりする必要はありません（第 5 章「グローバル GUI マップ・ファイル・モードでの作業」を参照してください)。これらの処理は自動的に行われます。

［**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，新しい GUI マップ・ファイルが作成されます。テストを保存するたびに，そのテストの GUI マップ・ファイルが保存されます。テストを開いたときには，そのテストに関連付けられている GUI マップ・ファイルが自動的にロードされます。

このモードで作業する場合，次に示す WinRunner の機能が使用できません。

- テスト・スクリプト・ウィザードは使用できません。このウィザードの詳細については，第 5 章「グローバル GUI マップ・ファイル・モードでの作業」を参照してください。
- WinRunner の再起動時に（最後の）一時 GUI マップ・ファイルを再ロードするオプション (［一般オプション］ダイアログ・ボックスの［**一般設定**］カテゴリの［**一時 GUI マップ ファイルをロード**］チェック・ボックス）は，使用できません。このオプションの詳細については，第 22 章「グローバル・テスト・オプションの設定」を参照してください。
- コンパイル済みモジュールが GUI マップ・ファイルをロードしません。コンパイル済みモジュールが GUI オブジェクトを参照する場合，それらのオブジェクトは，そのモジュールをロードするテストでも参照されなければなりません。詳細については，第 31 章「テストでのユーザ定義関数の利用」を参照してください。
- ［**テスト特有の GUI マップ ファイル**］モードで作成され，呼び出されたテストが，GUI オブジェクトを参照する場合，そのテストは［**グローバルな GUI マップ ファイル**］モードで正しく実行できないことがあります。

  ［**テスト特有の GUI マップ ファイル**］モードで作業するには，［一般オプション］ダイアログ・ボックスの［**一般設定**］カテゴリで，このオプションを指定します。

  WinRunner に慣れてきたら，［**グローバルな GUI マップ ファイル**］モードで作業することを検討してください。［**テスト特有の GUI マップ ファイル**］モードから［**グローバルな GUI マップ ファイル**］モードに変更するには，各テストに関連付けられている GUI マップ・ファイルを，テスト・グループに共通の GUI マップ・ファイルにマージする必要があります。GUI マップ・ファイルをマージするには，GUI マップ・ファイル・マージ・ツールを使用します。GUI マップ・ファイルのマージと，［**グローバルな GUI マップ ファイル**］モードへの変更の詳細については，第 23 章「GUI マップ・ファイルのマージ」を参照してください。

［テスト特有の GUI マップ ファイル］モードの指定

［**テスト特有の GUI マップ ファイル** ］モードで作業するには，［一般オプション］ダイアログ・ボックスの［**一般設定**］カテゴリで，このオプションを指定する必要があります。

［テスト特有の GUI マップ ファイル］モードで作業するには，次の手順を実行します。

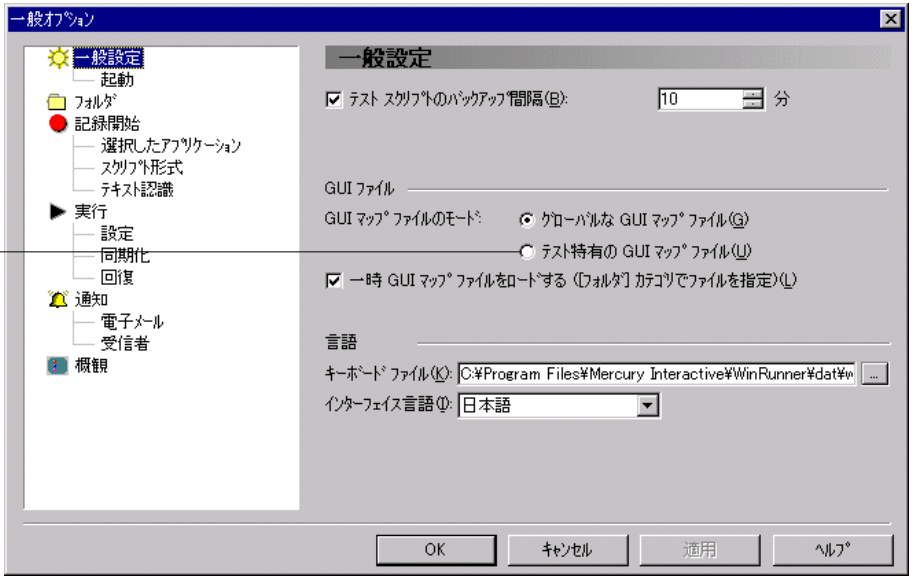
1 ［**ツール**］＞［**一般オプション**］を選択します。

［一般オプション］ダイアログ・ボックスが開きます。

2 ［**一般設定**］カテゴリをクリックします。

3 ［GUI マップ ファイル モード］セクションで，［**テスト特有の GUI マップ ファイル**］をクリックします。

［テスト特有の GUI マップ ファイル］モードの設定



4 ［**OK**］をクリックし，ダイアログ・ボックスを閉じます。

5 WinRunner を一旦閉じて再起動するまで，変更が反映されないという内容の警告ダイアログ・ボックスが表示されます。［**OK**］をクリックします。

［**一時 GUI マップ ファイルをロード**］オプションが，自動的に無効になります。

6 WinRunner を閉じるときに，設定の変更を保存するかどうかを尋ねるメッセージが表示されます。［**はい**］をクリックします。

注：この変更を有効にするには，WinRunner を再起動する必要があります。

［一般オプション］ダイアログ・ボックスの詳細については，第 22 章「グローバル・テスト・オプションの設定」を参照してください。

［テスト特有の GUI マップ ファイル］モードでの作業

新しいテストを作成するたびに，各テストに対して新しい GUI マップ・ファイルが自動的に作成されます。テストを保存するたびに，対応する GUI マップ・ファイルが保存されます。GUI マップ・ファイルはテストと同じフォルダに保存されます。テストを新しい場所に移動すると，そのテストに関連付けられている GUI マップ・ファイルも移動します。

WinRunner は，記録または［学習］機能を使用することによってアプリケーションの GUI を学習します。アプリケーションの GUI が変更された場合は，［GUI マップ エディタ］を使って，各テストの GUI マップ・ファイルを更新できます。GUI マップ・ファイルをロードまたは保存する必要はありません。

GUI マップ・ファイルを更新するには，次の手順を実行します。

1 GUI マップ・ファイルを更新するテストを開きます。

2 ［**ツール**］＞［**GUI マップ エディタ**］を選択し，［GUI マップ エディタ］を開きます。

3 第 7 章「GUI マップの編集」の説明に従って，開いている GUI マップ・ファイルを編集します。

注：GUI マップ・ファイルのオブジェクトの論理名を変更する場合，それに従ってテスト・スクリプトを更新する必要があります。詳細については，77 ページ「論理名と物理的記述の修正」を参照してください。

---

4 終了したら，［**ファイル**］＞［**終了**］を選択し，［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］モードでの作業のガイドライン

［**テスト特有の GUI マップ ファイル**］モードで作業するときは，次のガイドラインに注意してください。

- ［GUI マップ エディタ］から GUI マップ・ファイルに対して行った変更は保存しないでください。変更は，テストの保存時に自動的に保存されます。
- **GUI_load** ステートメントをテストに挿入しないでください。
- ［**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，GUI マップ・ファイルのロードとアンロードを手作業で行わないでください。テストを開いたときに，各テストの GUI マップ・ファイルが自動的にロードされます。
- グローバル GUI マップ・モードを使用する他のテストを呼び出さないでください。

GUI マップの使い方のガイドラインについては，41 ページ「GUI マップ・ファイルの使い方に関する一般的なガイドライン」を参照してください。

第 7 章

GUI マップの編集

本章では，GUI マップにおけるオブジェクトの記述を修正して，テストを長く使うための方法を解説します。

本章では，以下の項目について説明します。

- GUI マップの編集について
- GUI マップ・エディタ
- 実行ウィザード
- 物理的記述の正規表現の使用
- WinRunner のウィンドウ・ラベル変更の対処方法
- 物理的記述の正規表現の使用
- ファイル間でのオブジェクトのコピーと移動
- GUI マップ・ファイルでオブジェクトを検索する方法
- 複数の GUI マップ・ファイルでオブジェクトを検索する方法
- 手作業による GUI マップ・ファイルへのオブジェクトの追加
- GUI マップ・ファイルからのオブジェクトの削除
- GUI マップ・ファイルの全内容の削除
- 表示されるオブジェクトのフィルタ処理
- GUI マップへの変更の保存

GUI マップの編集について

WinRunner は，GUI マップを利用して，アプリケーションにおける GUI オブジェクトを識別し，探します。アプリケーションの GUI が変わった場合は，GUI マップのオブジェクト記述を更新して，既存のテストを使い続けられるようにします。

GUI マップを更新する方法は 2 つあります。

- テスト工程の任意の時点で，GUI マップ・エディタを実行する方法
- テストの実行中，実行ウィザードを使用する方法

実行ウィザードは，テスト実行時に WinRunner がテスト対象アプリケーションで特定のオブジェクトを見つけられないと自動的に開きます。このウィザードは，オブジェクトを識別して，GUI マップにおけるその記述を更新する手順を示してくれます。これにより，WinRunner は以降のテスト実行において，オブジェクトを探し出すことができます。

GUI マップ・エディタを使って作業中，次のことができます。

- GUI マップ・エディタを使って GUI マップを手作業で編集する
- オブジェクトの論理名や物理的記述の修正，新しい記述の追加，古い記述の削除などを行う
- GUI マップ・ファイルの記述を別の GUI マップ・ファイルに移動あるいはコピーする

GUI マップを更新する場合は，適切な GUI マップ・ファイルをあらかじめロードする必要があります。テスト・スクリプトで **GUI_load** ステートメントを使うか，GUI マップ・エディタで［ファイル］＞［開く］を選択して，ファイルを読み込むことができます。詳細については，第 4 章「GUI マップの基本概念について」を参照してください。

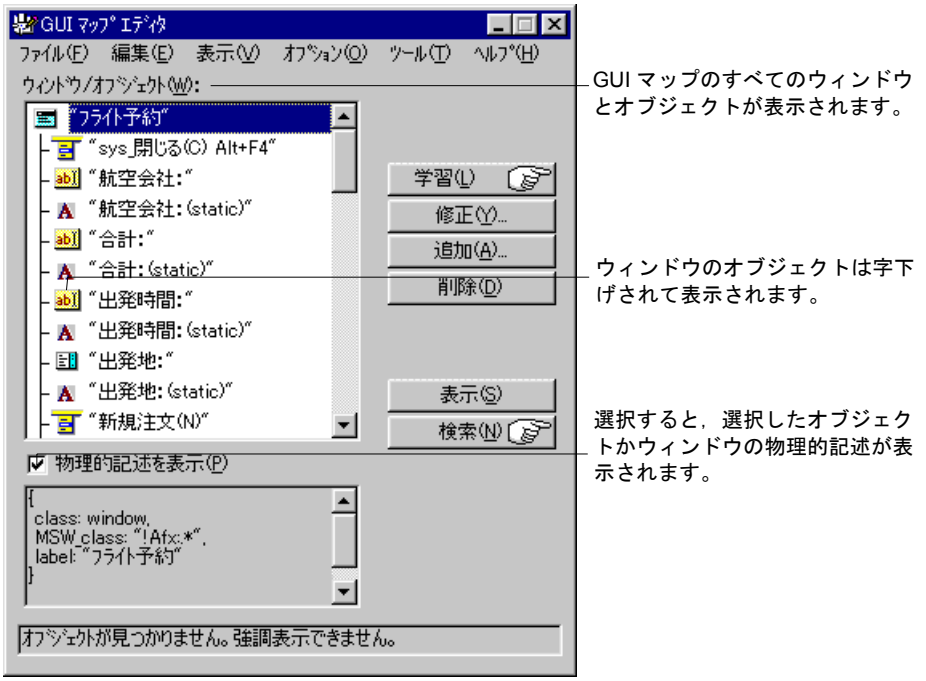
注：［テスト特有の GUI マップ ファイル］モードで作業している場合は，GUI マップ・ファイルのロード，アンロードを手作業で行わないでください。

GUI マップ・エディタ

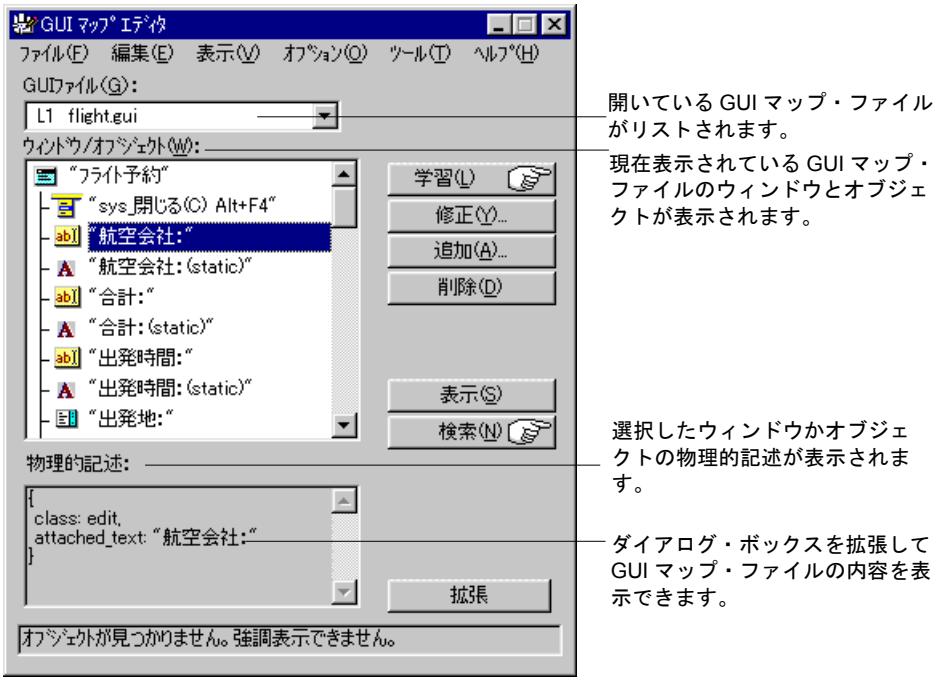
GUI マップ・エディタを使えば，いつでも GUI マップを編集できます。GUI マップ・エディタを開くには，［**ツール**］＞［**GUI マップ エディタ**］を選択します。

GUI マップ・エディタには 2 つのビューがあります。次のどちらかの内容を表示できます。

- GUI マップ全体
- 個々の GUI マップ・ファイル



特定の GUI マップ・ファイルの内容を表示するときに，GUI マップ・エディタを拡張して，同時に 2 つの GUI マップ・ファイルを表示できます。この機能を使えば，ファイル間の記述のコピーや移動が簡単に行えます。個々の GUI マップ・ファイルの内容を表示するには，［**表示**］＞［**GUI ファイル**］を選択します。



GUI マップ・エディタでは，オブジェクトが表示されるウィンドウ・アイコンごとにオブジェクトがまとめられて，ツリー構造で表示されます。ツリー内のウィンドウ名またはアイコンをダブルクリックすると，そのウィンドウに含まれているすべてのオブジェクトが表示されます。ツリーのすべてのオブジェクトを同時に見たい場合は，［**表示**］＞［**オブジェクトツリーの展開**］**を選択します。**ウィンドウだけを見たい場合は，［**表示**］＞［**オブジェクトツリーの折りたたみ**］を選択します。

GUI マップ全体を表示しているときに，［**物理的記述を表示する**］チェック・ボックスを選択すれば，［**ウィンドウ / オブジェクト**］リストで選択している任意のオブジェクトの物理的記述を表示できます。単独の GUI マップ・ファイルの内容を表示する場合，GUI マップ・エディタには物理的記述が自動的に表示されます。

GUI マップ・ファイルに［ワードパッド］ウィンドウがあるとします。［**物理的記述を表示する**］を選択して，ウィンドウのリストから［ワードパッド］ウィンドウ名またはアイコンをクリックすると，GUI マップ・エディタの真中の枠に以下の物理的記述が表示されます。

```
{
class: window,
label: " ドキュメント - ワードパッド ",
MSW_class: WordPadClass
}
```

注 :

GUI マップのオブジェクトの論理名を変更する場合，WinRunner が GUI マップで該当するオブジェクトを見つけられるよう，テスト・スクリプトのオブジェクトの論理名も変更しなければなりません。

プロパティの値に空白や特殊文字が含まれている場合，その値は二重引用符で囲まれます。複数のプロパティと値のセットは，カンマで区切ります。

実行ウィザード

実行ウィザードは，テストの実行の妨げとなる，アプリケーションの GUI における変更を検出します。テスト実行時に WinRunner がオブジェクトを見つけられないと，実行ウィザードが自動的に開きます。実行ウィザードは，アプリケーションのオブジェクトを指すようユーザに指示し，オブジェクトが見つからなかった原因を特定し，解決策を提示します。例えば，実行ウィザードから適切な GUI マップ・ファイルをロードするよう指示がでます。ほとんどの場合，GUI マップに自動的に新しい記述が追加されるか，既存の記述が修正されます。この手順を完了すると，テスト実行が継続されます（この更新作業により，以降のテストでは，WinRunner がオブジェクトを見つけられるようになります）。

例えば，アプリケーション内の［開く］ウィンドウで［ネットワーク］ボタンを押すようなテストを実行したとします。この部分のスクリプトには，次のように表示されます。

```
set_window (" 開く ");
button_press(" ネットワーク ");
```

［ネットワーク］ボタンが GUI マップにない場合，実行ウィザードが開き，問題点が表示されます。

ウィザードの指差しボタンをクリックし，［ネットワーク］ボタンを押すと，実行ウィザードから解決策が提示されます。

［**OK**］をクリックすると，［ネットワーク］オブジェクトの記述が GUI マップに自動的に追加され，WinRunner はテストを継続します。こうしておけば，次にテストを実行するときには，WinRunner は［ネットワーク］ボタンを特定できます。

実行ウィザードは，GUI マップではなく，テスト・スクリプトを編集する場合もあります。例えば，WinRunner がオブジェクトを見つけられなかった原因が，適切なウィンドウがアクティブでなかったためである場合，実行ウィザードは，テスト・スクリプトに **set_window** ステートメントを挿入します。

論理名と物理的記述の修正

GUI マップ・エディタを使って，GUI マップ・ファイル内の任意のオブジェクトの論理名や物理的記述を変更できます。

オブジェクトの論理名は，割り当てられている論理名がわかりにくい場合や，長すぎる場合には，変更すると便利です。例えば，WinRunner が静的テキスト・オブジェクトに対して「顧客アドレス : (static)」という論理名を割り当てたとします。この名前を「アドレス」に変えれば，テスト・スクリプトが読みやすくなります。

オブジェクトのプロパティ値が変わった場合には，物理的記述も変更しなければなりません。例えば，ボタンのラベルが「挿入」から「追加」に変わったとします。［挿入］ボタンの物理的記述の label プロパティの値は次のように変更できます。

［挿入］ボタン：{class:push_button, label: 追加 }

これにより，WinRunner はテストの実行中にテスト・スクリプト内で「挿入」という論理名に遭遇すると，「追加」というラベルを持つボタンを探します。

GUI マップ・ファイルでオブジェクトの論理名または物理的記述を変更するには，次の手順を実行します。

1 ［**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。

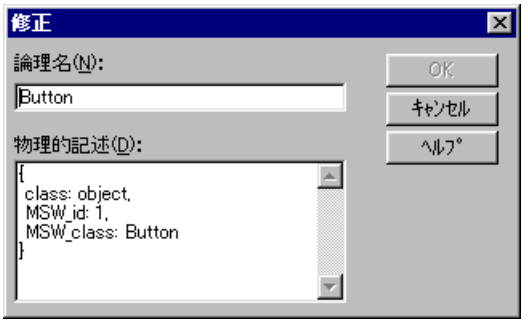
2 ［**表示**］＞［**GUI ファイル**］を選択します。

3 適切な GUI マップ・ファイルがロードされない場合は，［**ファイル**］＞［**開く**］を選択してファイルを開きます。

4 ウィンドウに属するオブジェクトを見るには，［**ウィンドウ / オブジェクト**］フィールドでウィンドウの名前をダブルクリックします。ウィンドウ内のオブジェクトは，字下げされて表示されます。

5 変更するオブジェクトまたはウィンドウの名前を選択します。

6 [**修正**] ボタンをクリックします。[修正] ダイアログ・ボックスが表示されます。



7 必要に応じて論理名あるいは物理的記述を編集して，[**OK**] ボタンをクリックします。変更は，ただちに GUI マップ・ファイルに反映されます。

物理的記述へのコメントの追加

オブジェクトの物理的記述を変更する際，物理的記述がより分かりやすいようコメントを追加できます。例えば，オブジェクトを認識しやすくするコメントを追加する場合，次のようにコメントを書くことができます。

```
{
 class: object,
 MSW_class: html_text_link,
 html_name: here,
 comment: " ホーム・ページへのリンク "
}
```

注：他のプロパティと同様，コメントのプロパティの値にスペースや特殊文字が含まれている場合はその値を引用符で囲まなくてはなりません。

WinRunner のウィンドウ・ラベル変更の対処方法

Windows では，よくラベルが変わります。例えば，テキスト・アプリケーションのメイン・ウィンドウのタイトル・バーには，アプリケーション名と一緒にファイル名を表示するものもあります。

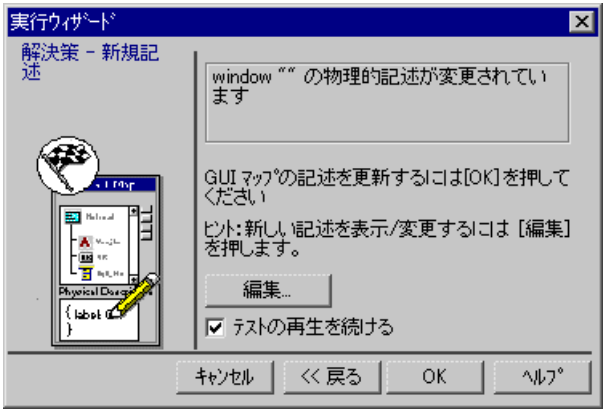
学習したラベルが変わったために WinRunner がウィンドウを認識できない場合，実行ウィザードが開き，不明ウィンドウを特定するようにユーザに求めます。ウィンドウを特定すると，WinRunner はラベルが変更されたことを認識し，その変更に合わせてウィンドウの物理的記述を変更します。

例えば，Microsoft Word のメイン・ウィンドウを対象にしたテストを記録したとします。WinRunner は以下の物理的記述を学習します。

```
{
class: window,
label: " 文書 11 - Microsoft Word"
MSW_class: OpusApp
}
```

Microsoft Word で文書 12 が開いているときにテストを実行したとします。WinRunner がウィンドウを見つけられないと，実行ウィザードが開きます。

指差しボタンをクリックして Microsoft Word の該当するウィンドウをクリックし，WinRunner に学習させます。WinRunner の GUI マップ内のウィンドウ記述を更新するように指示されます。



［編集］をクリックすると，WinRunner がウィンドウの物理的記述を変更して正規表現を含むようになったのがわかります。

```
{
class: window,
label: "!.* - Microsoft Word",
MSW_class: OpusApp
}
```

(テスト実行を継続するには［OK］をクリックします)

「文書 - Microsoft Word」ウィンドウ・タイトルの前にどういった名前が表示されても，WinRunner はこれらの正規表現を使って Microsoft Word ウィンドウを認識できます。

物理的記述の正規表現の使用

WinRunner は，オブジェクトの物理的記述に正規表現を使うために，2 つの「隠し」プロパティを使用します。これらのプロパティは，regexp_ label と regexp_ MSW_ class です。

regexp_ label プロパティは，ウィンドウだけに使用されます。このプロパティは，ウィンドウのラベル記述に正規表現を挿入するために「裏で」処理をします。

regexp_ MSW_ class プロパティは，オブジェクトの MSW_ class に正規表現を挿入します。これは，すべてのタイプのウィンドウと object クラスのオブジェクトに対して必須です。

正規表現の追加

regexp_ label と regexp_ MSW_ class プロパティを，必要なクラスの GUI 構成に追加できます。アプリケーションのオブジェクトのラベルか MSW クラスの一方に安全に無視できる共通の文字がある場合は，この方法で正規表現を追加できます。

正規表現の削除

ウィンドウの物理的記述の正規表現を削除することもできます。アプリケーションのすべてのウィンドウのラベルが「AAA Wingnuts ー」で始まっているとします。

WinRunner に個々のウィンドウを区別させるためには，アプリケーションのウィンドウ内の学習した必須のプロパティのリストの regexp_ label プロパティを label プロパティに置き換えることができます。詳細については，第 24 章「GUI マップの構成設定」を参照してください。

正規表現の詳細については，第 27 章「正規表現の使い方」を参照してください。

ファイル間でのオブジェクトのコピーと移動

GUI マップ・ファイル間で，GUI オブジェクトの記述をコピーあるいは移動することで，GUI マップ・ファイルを更新できます。編集だけのために開いた GUI ファイル，つまりロードしていないファイルから，オブジェクトをコピーすることも可能です。

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場合は，手作業で GUI マップ・ファイルを開いたり，ファイル間でオブジェクトをコピーまたは移動したりしないでください。

---

2 つの GUI マップ・ファイル間でオブジェクトをコピーまたは移動するには，次の手順を実行します。

1 ［**ツール**］＞［**GUI マップ エディタ**］を選択して，GUI マップ・エディタを開きます。

2 ［**表示**］＞［**GUI ファイル**］を選択します。

3 GUI マップ・エディタで［**拡張**］をクリックします。ダイアログ・ボックスが拡張し，2 つの GUI マップ・ファイルを並べて表示できるようになります。

4 ［**GUI ファイル**］リストでファイル名を選択して，ダイアログ・ボックスの両側に異なる GUI マップ・ファイルを表示します。

5 一方のファイルで，コピーまたは移動するオブジェクトを選択します。複数のオブジェクトを選択するには，Shift または Control キーを使います。GUI マップ・ファイルのすべてのオブジェクトを選択するには，［**編集**］＞［**すべて選択**］を選びます。

6 ［**コピー**］または［**移動**］ボタンをクリックします。

7 GUI マップ・エディタを元のサイズに戻すには，［**閉じる**］ボタンをクリックします。

注：ロードされている GUI マップ・ファイルから一時 GUI マップ・ファイルに新しいウィンドウを追加すると，一時 GUI マップ・ファイルを保存するときに，［新規ウィンドウ］ダイアログ・ボックスが開きます。新規ウィンドウを，ロードした GUI マップ・ファイルに追加するのか，新しい GUI マップ・ファイルに保存するのかたずねるメッセージが表示されます。詳細については，コンテキスト・センシティブ・ヘルプを参照してください。

GUI マップ・ファイルでオブジェクトを検索する方法

GUI マップ・ファイルにある特定のオブジェクトの記述を探すのは簡単です。テスト対象アプリケーションでそのオブジェクトをポイントするだけです。

GUI マップ・ファイルでアプリケーションのオブジェクトを検索するには，次の手順を実行します。

1 ［**ツール**］＞［**GUI マップ エディタ**］を選択して，GUI マップ・エディタを開きます。

2 ［**表示**］＞［**GUI ファイル**］を選択します。

3 ［**ファイル**］＞［**開く**］を選択して，GUI マップ・ファイルをロードします。

4 ［**検索**］を押します。マウス・ポインタが指差し型に変わります。

5 アプリケーションでオブジェクトをクリックします。オブジェクトが GUI マップで強調表示されます。

GUI マップ・ファイルでテスト・スクリプトのオブジェクトを検索するには，次の手順を実行します。

1 既存のテストを開き，すべての関連 GUI マップがロードされていることを確認します。

2 オブジェクトを含む行を右クリックし，［**GUI マップ内で検索**］を選択します。［GUI マップ エディタ］ダイアログ・ボックスが関連するオブジェクトが強調表示された状態で開きます。

テスト・スクリプトとテスト・スクリプト言語の詳細については，第 28 章「プログラミングによるテスト・スクリプトの 機能強化」を参照してください。

複数の GUI マップ・ファイルでオブジェクトを検索する方法

1 つのオブジェクトが複数の GUI マップ・ファイルで記述されている場合，GUI マップ・エディタの［トレース］ボタンを使って，すべてのオブジェクトを記述を探すことができます。これは，WinRunner にオブジェクトの新しい記述を学習させて，GUI マップ・ファイルの古い記述を探し出して削除したい場合に便利です。

複数の GUI マップ・ファイルでオブジェクトを探すには，次の手順を実行します。

1 ［**ツール**］＞［**GUI マップ エディタ**］を選択して，GUI マップ・エディタを開きます。

2 ［**表示**］＞［**GUI ファイル**］を選択します。

3 ［**ファイル**］＞［**開く**］**を選択して**，オブジェクト記述が含まれている可能性のあるすべての GUI マップ・ファイルを開きます。

開きたい GUI マップ・ファイルを選んで，［**編集のためにのみ開く**］を選択します。［**OK**］をクリックします。

4 ［**GUI ファイル**］ボックスで GUI マップ・ファイルを表示して，オブジェクトの最新の記述が含まれているファイルの内容を表示します。

5 ［**ウィンドウ / オブジェクト**］ボックスでオブジェクトを選択します。

6 ［**拡張**］ボタンをクリックして［GUI マップ エディタ］ダイアログ・ボックスを拡張します。

7 ［**トレース**］ボタンをクリックします。オブジェクトが見つかった GUI マップ・ファイルがダイアログ・ボックスのもう一方の側に表示され，オブジェクトが強調表示されます。

手作業による GUI マップ・ファイルへのオブジェクトの追加

別のオブジェクトの記述をコピーした後に編集を行うことで，GUI マップ・ファイルに手作業でオブジェクトを追加できます。

GUI マップ・ファイルに手作業でオブジェクトを追加するには，次の手順を実行します。

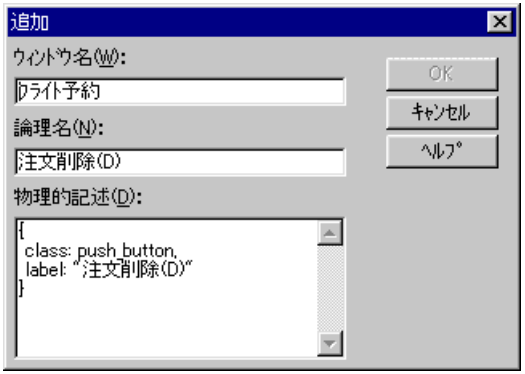
1 ［**ツール**］＞［**GUI マップ エディタ**］を選択して，GUI マップ・エディタを開きます。

2 ［**表示**］＞［**GUI ファイル**］を選択します。

3 GUI マップ・エディタで［**ファイル**］＞［**開く**］を選択して，適切な GUI マップ・ファイルを開きます。

4 編集の基礎とするオブジェクトを選択します。

5 ［**追加**］ボタンをクリックして，［**追加**］ダイアログ・ボックスを開きます。



6 適切なフィールドを編集して，［**OK**］を押します。オブジェクトが GUI マップ・ファイルに追加されます。

GUI マップ・ファイルからのオブジェクトの削除

不要になったオブジェクト記述は，GUI マップ・ファイルから削除できます。

GUI マップ・ファイルからオブジェクトを削除するには，次の手順を実行します。

1 ［**ツール**］＞［**GUI マップ エディタ**］を選択して，GUI マップ・エディタを開きます。

2 ［**表示**］＞［**GUI ファイル**］を選択します。

3 GUI マップ・エディタで［**ファイル**］＞［**開く**］を選択して，適切な GUI マップ・ファイルを開きます。

4 削除するオブジェクトを選択します。複数のオブジェクトを削除したい場合は，Shift または Control キーを使って選択します。

5 ［**削除**］をクリックします。

6 ［**ファイル**］＞［**上書き保存**］を選択して，GUI マップ・ファイルの変更を保存します。

GUI マップ・ファイルからすべてのオブジェクトを削除するには，次の手順を実行します。

1 ［**ツール**］＞［**GUI マップ エディタ**］を選択して，GUI マップ・エディタを開きます。

2 ［**表示**］＞［**GUI ファイル**］を選択します。

3 GUI マップ・エディタで［**ファイル**］＞［**開く**］を選択して，適切な GUI マップ・ファイルを開きます。

4 ［**編集**］＞［**すべてクリア**］を選択します。

GUI マップ・ファイルの全内容の削除

仮 GUI マップ・ファイルや他の任意の GUI マップ・ファイルの全内容をすばやく削除できます。

GUI マップ・ファイルの全内容を削除するには，次の手順を実行します。

1 ［**ツール**］＞［**GUI マップ エディタ**］を選択して，GUI マップ・エディタを開きます。

2 ［**表示**］ > ［**GUI ファイル**］を選択します。

3 適切な GUI マップ・ファイルを開きます。

4 ［GUI ファイル］リストの先頭にある GUI マップ・ファイルを表示します。

5 ［**編集**］ > ［**すべてクリア**］を選びます。

表示されるオブジェクトのフィルタ処理

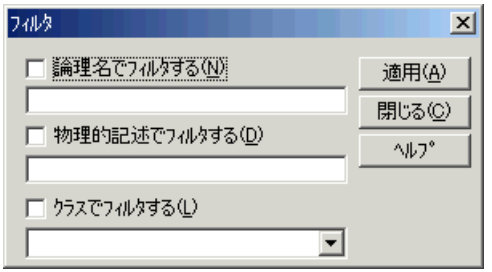
以下のフィルタを使って，GUI マップ・エディタに表示されるオブジェクトをフィルタ処理できます。

- 「**論理名**」には，指定した論理名（「Open」など）あるいは部分文字列（「Op」など）を持つオブジェクトだけが表示されます。
- 「**物理的記述**」には，指定した物理的記述に一致したオブジェクトだけが表示されます。物理的記述に属するいずれかの部分文字列を使用します（例えば，「w」と指定すると物理的記述に「w」を含むすべてのオブジェクトだけが表示されます）。
- 「**クラス**」には，「プッシュ・ボタンすべて」というように，指定したクラスのオブジェクトだけが表示されます。

フィルタを適用するには，次の手順を実行します。

1 ［**ツール**］ > ［**GUI マップ エディタ**］を選択して，GUI マップ・エディタを開きます。

2 ［**オプション**］ > ［**フィルタ**］を選択して，［フィルタ］ダイアログ・ボックスを開きます。



3 チェック・ボックスをクリックして使用するフィルタを選択し，適切な情報を入力します。

4［**適用**］ボタンを押します。GUI マップ・エディタには，指定したフィルタの条件に合うオブジェクトが表示されます。

GUI マップへの変更の保存

GUI マップに含まれているオブジェクトの論理名や物理的記述を編集したり，GUI マップ・ファイルのオブジェクトやウィンドウを変更したりした場合，テスト・セッションを完了して WinRunner を終了する前に，GUI マップ・エディタで変更を保存しなければなりません。

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場合は，GUI マップ・ファイルへの変更を手作業で保存しないでください。変更はテストが自動的に保存します。

GUI マップに対する変更を保存するには，以下の 2 つの方法があります。

➤ GUI マップ・エディタで［**ファイル**］＞［**上書き保存**］を選択して，適切な GUI マップ・ファイルに変更を保存します。

➤［**ファイル**］＞［**名前を付けて保存**］を選択して，新しい GUI マップ・ファイルに変更を保存します。

注：ロードされている GUI マップ・ファイルから一時 GUI マップ・ファイルに新しいウィンドウを追加すると，一時 GUI マップ・ファイルを保存するときに，［新規ウィンドウ］ダイアログ・ボックスが開きます。新規ウィンドウを，ロードした GUI マップ・ファイルに追加するのか，新しい GUI マップ・ファイルに保存するのかたずねるメッセージが表示されます。詳細については，**WinRunner** のオンライン・ヘルプを参照してください。

第 3 部

テストの作成 ― 基本

第 8 章

テストの設計

記録とプログラミングのいずれか，またはその両方を行って，自動テストをすばやくデザインできます。

本章では，以下の項目について説明します。

- テストの作成について
- WinRunner のテスト・ウィンドウについて
- テストの計画
- コンテキスト・センシティブ記録モードを使用したテストの作成
- アナログ記録モードを使ったテストの作成
- テストの記録のガイドライン
- テストへのチェックポイントの追加
- データ駆動型テストを使った作業
- テストへの同期化ポイントの追加
- トランザクションの測定
- ソフトキーを使用したテスト作成コマンドのアクティブ化
- テストのプログラミング
- テストの編集
- テスト・ファイルの管理

テストの作成について

テストは，記録とプログラミングの両方で作成することができます。通常は，基本的な「**テスト・スクリプト**」を記録するところから始めます。記録の際には，操作を行うたびに Mercury Interactive 社のテスト・スクリプト言語（TSL）でステートメントが生成されます。これらのステートメントは，テスト・ウィンドウにテスト・スクリプトとして表示されます。記録を終了したら，TSL 関数やプログラミング要素を追加入力するか，WinRunner のビジュアル・プログラミング・ツールである関数ジェネレータまたは関数ビューアを使って，記録したテスト・スクリプトを強化できます。

テストを記録するためのモードは 2 つあります。

- 「**コンテキスト・センシティブ**」モードでは，グラフィカル・ユーザ・インタフェース（GUI）オブジェクトが識別され，アプリケーションに対する操作が記録されます。
- 「**アナログ**」モードでは，キーボード入力，マウス・クリック，および画面上のマウス・ポインタの軌跡を示す正確な x 座標と y 座標が記録されます。

テスト・スクリプトには，同期化ポイントだけでなく，GUI，ビットマップ，テキスト，データベースのチェックポイントを追加できます。チェックポイントを使用すれば，アプリケーションの以前のバージョンでの動作と現在のバージョンでの動作を比較し，アプリケーションを検査できます。また，同期化ポイントを使用することで，テスト実行の際に生じる可能性のある，タイミングやウィンドウの表示位置の問題を解決できます。

内部テーブルに格納されたデータによって駆動する，データ駆動型テストを作成できます。

注：WinRunner の記録と編集の操作の多くはマウスを使用して実行されます。Section 508 に準拠し，WinRunner は，Windows Accessibility Options ユーティリティの **MouseKeys** オプションを使用した操作も認識します。また，WinRunner ソフトキーを使用して多くの操作が行えます。詳細については，862 ページ「WinRunner ソフトキーの構成設定」を参照してください。

テスト・スクリプトを作成するための大まかな手順は，次のとおりです。

1 テストの対象となる機能を決めます。テスト・スクリプトで必要なチェックポイントと同期化ポイントを決めます。

2 ［テストのプロパティ］ダイアログ・ボックスで，テストの概要情報を書きます。

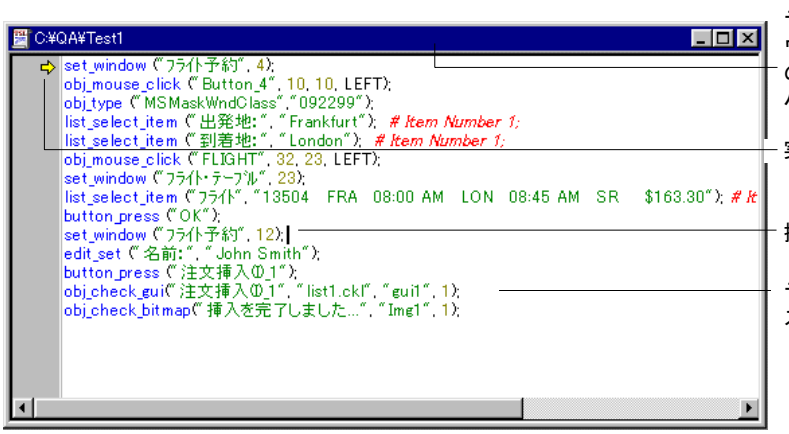
3 記録モード（「**コンテキスト・センシティブ**」または「**アナログ**」）を選択して，アプリケーションに対するテストを記録します。

4 テストに名前を付けて，それをファイル・システムか Quality Center プロジェクトに保存します。

WinRunner のテスト・ウィンドウについて

テスト・ウィンドウで WinRunner テストを開発して実行します。テスト・ウィンドウには次のものが含まれています。

- 「**テスト・ウィンドウのタイトル・バー**」。開いているテストの名前を表示します。
- 「**テスト・スクリプト**」。Mercury Interactive 社のテスト・スクリプト言語（TSL）で記録またはプログラミングすることによって生成されるステートメントで構成されます。
- 「**実行矢印**」。実行中のテスト・スクリプトの行または［矢印から実行］オプションを使用してテスト実行を開始する行を示します（マーカを移動するには，行の左横のウィンドウ・マージンでマウスをクリックします）。
- 「**挿入ポイント**」。テキストを挿入または編集する場所を示します。



テストの計画

テストは，記録またはプログラミングする前に十分計画を立てる必要があります。検討すべき点をいくつか次に示します。

- テストしようとしている機能を特定します。複数のタスクを実行する長いテストよりも，アプリケーションの特定の機能を検査する短い特化されたテストを設計するほうがよいでしょう。
- テストの一部またはすべてを記録する場合は，テストのどの部分にアナログ記録モードを使用して，どの部分にコンテキスト・センシティブ記録モードを使用するか決定します。詳細については，97 ページ「コンテキスト・センシティブ記録モードを使用したテストの作成」と 102 ページ「アナログ記録モードを使ったテストの作成」を参照してください。
- テストで使用するチェックポイントと同期化ポイントの種類を決定します。詳細については，106 ページ「テストへのチェックポイントの追加」と 107 ページ「テストへの同期化ポイントの追加」を参照してください。
- 記録済みのテスト・スクリプトに追加するプログラミング要素の種類（ループ，配列，ユーザ定義関数など）を決定します。詳細については，113 ページ「テストのプログラミング」を参照してください。

コンテキスト・センシティブ記録モードを使用したテストの作成

「**コンテキスト・センシティブ**」モードでは，アプリケーションに対して行った操作が，GUI オブジェクトに基づいて記録されます。記録を行うと，WinRunner はクリックされた各 GUI オブジェクト（ウィンドウ，ボタン，リストなど）と，行った操作（ドラッグ，クリック，選択など）を識別します。

例えば，［開く］ダイアログ・ボックスで［開く］ボタンをクリックすると，WinRunner は以下を記録します。

```
button_press (" 開く ");
```

このテストを実行すると，テスト・スクリプトに記録されている WinRunner は［開く］ダイアログ・ボックスと［開く］ボタンを探します。この後でテストを実行したときに，［開く］ダイアログ・ボックスの中でボタンが別の場所にあったとしても，WinRunner はボタンを見つけだすことができます。

アプリケーションのユーザ・インタフェースに対する操作をテストする場合は，コンテキスト・センシティブ・モードを使います。例えば，WinRunner は，GUI に対する操作（ボタンのクリック，メニューやリストの選択など）を実行し，GUI オブジェクトの状態を調べて，操作の結果（チェック・ボックスの状態，テキスト・ボックスの内容，リストで選択されている項目など）を検査できます。

コンテキスト・センシティブ・テストでは，GUI マップと GUI マップ・ファイルを使う点に注意してください。記録を始める前に，本書の「GUI マップについて」（23 ページ～）には必ず目を通しておきましょう。

次の例にテスト・スクリプトと GUI マップ間の関係，および論理名と物理的記述の関係を示します。［ファイル］メニューの［印刷］コマンドを選択して［印刷］ダイアログ・ボックスを開き，［**OK**］ボタンを押して Readme ファイルを印刷するテストを記録するとします。テスト・スクリプトは次のようになります。

```
# ［Readme.doc- ワードパッド］ウィンドウをアクティブにする。
win_activate ("Readme.doc - ワードパッド ");

# ［Readme.doc- ワードパッド］ウィンドウに入力を受け取るよう指示する。
set_window ("Readme.doc - ワードパッド ", 10);

# ［ファイル］＞［印刷］を選択する。
menu_select_item (" ファイル ; 印刷 ... Ctrl+P");

# ［印刷］ウィンドウに入力を受け取るよう指示する。
set_window (" 印刷 ", 10);

# ［OK］ボタンをクリックする。
button_press ("OK");
```

WinRunner は，関連する各オブジェクトの実際の記述（プロパティと値のリスト）を学習し，その記述を GUI マップに書き込みます。

GUI マップを開き，オブジェクトを強調表示すると，物理的記述を表示できます。次の例では，Readme.doc ウィンドウが GUI マップで強調表示されています。

WinRunner は，GUI マップ内の他のウィンドウおよびオブジェクトに対して次のような記述を書き込みます。

［**ファイル**］メニュー : {class:menu_item, label: ファイル , parent:None}
［**印刷**］コマンド : {class: menu_item, label: " 印刷 ... Ctrl+P", parent: ファイル }
［**印刷**］ウィンドウ : {class:window, label: 印刷 }
［**OK**］ボタン : {class:push_button, label:OK}

（これらの記述を見るには，GUI マップでウィンドウまたはオブジェクトを強調表示すると，GUI マップの下部に物理的記述が表示されます。）

WinRunner は，各オブジェクトに論理名を割り当てます。WinRunner はテストを実行すると，テスト・スクリプト内の各オブジェクトの論理名を読み取り，GUI マップの物理的記述を参照します。次に WinRunner は，この記述を使ってテスト対象アプリケーションから該当するオブジェクトを見つけます。

テストをコンテキスト・センシティブ・モードでテストするには，次の手順を実行します。

1 ［**テスト**］＞［**記録 – コンテキスト センシティブ**］を選択するか，［**記録 – コンテキスト センシティブ**］ボタンをクリックします。

［**記録開始**］ボタンの上に明るい青の背景に濃い青で Rec という文字が表示され，コンテキスト・センシティブ記録セッションがアクティブであることを示します。

2 キーボードとマウスを使用して，計画どおりにテストを実行します。

必要に応じて，ユーザ定義ツールバーまたは［挿入］メニューから適切なコマンド（GUI チェックポイント，ビットマップ・チェックポイント・データベース・チェックポイント，同期化ポイント）を選択して，チェックポイントと同期化ポイントを挿入します。

3 記録を停止するには，［**テスト**］＞［**記録停止**］を選択するか，［**停止**］をクリックします。

コンテキスト・センシティブな記録についての一般的な問題

この節では，コンテキスト・センシティブ・テストの作成中に生じる可能性のある一般的な問題について説明します。

WinRunner がオブジェクトに対し正しい TSL ステートメントを記録しない

オブジェクトを記録しても WinRunner がそのオブジェクト・クラスに適切な TSL ステートメントを記録せず，その代わりに **obj_mouse** ステートメントを記録します。これは，WinRunner がそのオブジェクトがどのクラスに属しているか認識できないために生じます。WinRunner はこうしたオブジェクトに汎用の「オブジェクト」クラスを割り当てます。

この問題が生じる原因と解決法には以下のものがあります。

考えられる原因	解決法
オブジェクト用のアドイン・サポートがロードされていない。	希望のオブジェクト用のアドイン・サポートをインストールしてロードする必要があります。例えば，HTML オブジェクトの場合は Web アドインをロードします。アドイン・サポートのロードについては，21 ページ「WinRunner アドインのロード」を参照してください。
オブジェクトがユーザ定義オブジェクトである。	ユーザ定義オブジェクトが標準オブジェクトと似ている場合は，ユーザ定義クラスを標準クラスにマッピングできます。詳細については，第 24 章「GUI マップの構成設定」を参照してください。
	ユーザ定義 GUI オブジェクト・クラスを追加できます。ユーザ定義 GUI オブジェクト・クラスを作成し，ユーザ定義オブジェクトを検査する方法の詳細については，『**WinRunner カスタマイズ・ガイド**』を参照してください。ユーザ定義オブジェクトに GUI 検査を作成することもできます。GUI オブジェクトの検査については，第 5 章「グローバル GUI マップ・ファイル・モードでの作業」を参照してください。
	ユーザ定義の記録関数と実行関数を作成できます。オブジェクトが変わった場合，テスト・スクリプト全体を更新するのではなく，関数を修正するだけで済みます。ユーザ定義の記録関数と実行関数については，『**WinRunner カスタマイズ・ガイド**』を参照してください。

WinRunner がアプリケーションの HTML ページからテキストを読み取れない

この問題の原因と解決法には以下のものがあります。

考えられる原因	解決法
WebTest アドインがロードされていない。	Web オブジェクトのアドイン・サポートをインストールしてロードする必要があります。アドイン・サポートのロードについての情報は，21 ページ「WinRunner アドインのロード」を参照してください。
WinRunner がテキストを HTML フレームまたはテーブルに含まれていると認識できない。	［**挿入**］＞［**テキストの取得**］＞［**指定範囲から（Web のみ）**］コマンドを使って，HTML ページからテキストを取得します。フレームに対しては，WinRunner は **web_frame_get_text** ステートメントを挿入します。他の GUI オブジェクト・クラスに対しては，WinRunner は **web_obj_get_text** ステートメントを挿入します。
	［**挿入**］＞［**テキストの取得**］＞［**Web テキストチェックポイント**］コマンドを使って，指定したテキスト文字列が HTML ページにあるかどうかを調べます。フレームに対しては，WinRunner は **web_frame_text_exists** ステートメントを挿入します。他の GUI オブジェクトクラスに対しては，WinRunner は **web_obj_text_exists** ステートメントを挿入します。

詳細については，第 10 章「Web オブジェクトでの作業」か「**TSL リファレンス**」を参照してください。コンテキスト・センシティブ・テストに関する問題を解決するための情報については，WinRunner のコンテキスト・センシティブ・ヘルプを参照してください。

アナログ記録モードを使ったテストの作成

「**アナログ**」モードでは，キーボード入力，マウス・クリック，マウスの正確な軌跡が記録されます。たとえば，アプリケーションの［ファイル］メニューから［開く］コマンドを選択した場合，WinRunner は画面上でのマウス・ポインタの動きを記録します。WinRunner でそのテストを実行すると，マウス・ポインタが記録された座標をたどります。

上記のメニュー選択の様子は，テスト・スクリプトに次のように記録されます。

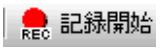
```
# マウスの軌跡
move_locator_track (1);

# マウスの左ボタンを押す。
mtype ("<T110><kLeft>-");

# マウスの軌跡
move_locator_track (2);

# マウスの左ボタンを放す。
mtype ("<kLeft>+");
```

描画アプリケーションのテストなど，テストにおいてマウスの正確な動きが重要な意味を持っている場合は，アナログ・モードを使います。コンテキスト・センシティブ・モードで記録しているときでも，適切なメニュー項目を選択したり，記録セッション中に［記録］ボタンをクリックしたり，F2 ショートカット・キーを使用したりすることによって，アナログ・モードに切り替えたり，アナログ・モードからコンテキスト・センシティブ・モードに切り替えたりすることができます

アナログ・モードを使用してテストを記録するには，次の手順を実行します。

1 WinRunner ウィンドウと テスト対象アプリケーションを両方とも見えるように配置します。

2 ［**テスト**］＞［**記録 – アナログ**］を選択します。または，［**記録 – コンテキスト センシティブ**］ボタンをクリックして，コンテキスト・センシティブ・モードで記録を開始します。記録セッション中にアナログ・モードに切り替えるには，再度［**記録開始**］ボタンまたは F2 を押します。



［**記録開始**］ボタンの上に白の背景に赤で Rec という文字が表示され，アナログ記録セッションがアクティブであることを示します。

3 キーボードとマウスを使用して，アプリケーションで必要な操作を実行します。

注：アナログ記録セッションでは，WinRunner ウィンドウまたは WinRunner ダイアログ・ボックスで実行されるものを含めすべてのマウス操作が記録されます。したがって，アナログ記録セッション中にチェックポイントや同期化ポイントを挿入したり，他の WinRunner メニューやツールバー・オプションを選択してはいけません。

4 記録を停止するには，［**テスト**］＞［**記録停止**］を選択するか，［**停止**］をクリックします。コンテキスト・センシティブ記録モードに戻すには，F2 を押すか，［**記録開始**］ツールバー・ボタンをクリックします。

テストの記録のガイドライン

テストの記録を行う際には，次のガイドラインを検討してください。

- 記録を開始する前に，テストで必要とされないアプリケーションをすべて閉じます。
- **invoke_application** ステートメントを使用するか，［**テストのプロパティ**］ダイアログ・ボックスの［**実行**］タブで起動アプリケーションを設定することで，テスト対象のアプリケーションを開くようにします。

  TSL 関数を使用した作業の詳細については，第 28 章「プログラミングによるテスト・スクリプトの 機能強化」を参照してください。**invoke_application** 関数の詳細とその使用例については，「**TSL リファレンス**」を参照してください。起動アプリケーションの詳細については，510 ページ「起動アプリケーションおよび起動関数の定義」. を参照してください。
- ウィンドウ内のオブジェクトを対象とした記録を開始する前に，ウィンドウのタイトルバーをクリックして **win_activate** ステートメントを記録してください。これによって，ウィンドウがアクティブになります。TSL 関数を使用した作業の詳細については，第 28 章「プログラミングによるテスト・スクリプトの 機能強化」. を参照してください。**win_activate** 関数の詳細とその使用例については，「**TSL リファレンス**」を参照してください。
- テストは，実行後に「後始末」を行うように作成してください。テストが完了したら，環境はテスト前の状態に戻っていなければなりません（例えば，テストの開始時にアプリケーションのウィンドウが閉じていたのであれば，テストの終了時に，ウィンドウを最小化してアイコンにするのではなく，ウィンドウを閉じるようにします）。
- テストの記録時には，WinRunner を最小化し，ユーザ定義ツールバーを浮動ツールバーに変更することができます。こうすることで，必要なメニュー・コマンドへのアクセスを確保しながら，全画面表示のアプリケーションを対象に記録が行えます。WinRunner を最小化して浮動ユーザ定義ツールバーを使用して作業をするには，WinRunner ウィンドウにドッキングされているユーザ定義ツールバーのドッキングを解除し，記録を開始し，WinRunner を最小化します。ユーザ定義ツールバーは，他のすべてのアプリケーションの手前に表示されま

す。ユーザ定義ツールバーは，テストの作成時に最もよく使うメニュー・コマンドを登録することでカスタマイズできます。詳細については，第 42 章「WinRunner のユーザ・インタフェースの カスタマイズ」を 参照してください。

- 記録時に，テスト対象アプリケーションのウィンドウ内で移動するときは，Tab キーではなく，マウス・クリックを使用して移動するようにします。
- アナログ・モードでの記録時，チェックポイントを挿入するには，WinRunner のメニューやツールバーではなく，ソフトキーを使用するようにします。
- アナログ・モードでの記録時，先行入力は避けてください。例えば，ウィンドウを開くときは，ウィンドウの描画が完全に終わるまで待ってから続行します。また，マウス・ボタンを押したままにすると（例えば，スクロール・バーを使用して画面表示を移動する場合など）反復アクションとなる場合には，押したままにしないようにしてください。押したままにすると，正確に再現することが難しい，時間に厳密な操作が開始されることがあります。その代わりに，同じことを達成するために，クリックを複数回に分けて個別に行うようにします。
- WinRunner では，RTL スタイルのウィンドウ・プロパティを持つアプリケーションを対象としたテストの記録と実行がサポートされています。RTL スタイルのウィンドウ・プロパティとは，右から左に向かう順序のメニューとタイプ入力，左側のスクロール・バー，GUI オブジェクトの右上隅の付属テキストなどのプロパティを指します。WinRunner では，タイプ入力時に，CTRL キーと SHIFT キーを同時に押すか，ALT キーと SHIFT キーを同時に押すことで，言語とタイプ入力の方向を変更できます。付属テキストに関する標準の設定では，RTL スタイルのウィンドウを持つアプリケーションを対象としたテストの記録と実行がサポートされています。付属テキストのオプションの詳細については，第 22 章「グローバル・テスト・オプションの設定」および第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。
- WinRunner では，ドロップダウンおよびメニューのようなツールバーを持つアプリケーションを対象としたテストの記録と実行をサポートしています。メニューに似たツールバーは，メニューとまったく同じように見えますが，そのクラスが異なるため，WinRunner では異なるように記録されます。ドロップダウンまたはメニューに似たツールバーの項目が選択されると，WinRunner によって **toolbar_select_item** ステートメントが記録されます（この関数は，メニューのメニュー・コマンドの選択を記録する **menu_select_item** 関数に似ています）。詳細については，「**TSL リファレンス**」を参照してください。

- テスト・フォルダまたはテスト・スクリプト・ファイルがファイル・システムの中で読み取り専用に設定されている場合，テスト・スクリプトまたは期待結果フォルダに変更を加えるような WinRunner 操作は実行できません。

テストへのチェックポイントの追加

チェックポイントを使って，テスト対象アプリケーションの現在の動作を，以前のバージョンの動作と比較することができます。

テスト・スクリプトには，4 種類のチェックポイントを追加できます。

- GUI チェックポイントは，GUI オブジェクトの情報を検証します。例えば，ボタンが使用可能かどうか，リストのどの項目が選択されているかなどを検査できます。詳細については，第 9 章「GUI オブジェクトの検査」を参照してください。
- ビットマップ・チェックポイントは，アプリケーションのウィンドウまたは特定の領域の「スナップショット」をキャプチャして，それを以前のバージョンでキャプチャしたイメージと比較します。詳細については，第 15 章「ビットマップの検査」を参照してください。
- テキスト・チェックポイントは，GUI オブジェクトまたはビットマップのテキストをロードして，その内容を検証します。詳細については，第 16 章「テキストの検査」を参照してください。
- データベース・チェックポイントは，データベースに作成したクエリに基づく結果セットの行とカラムの内容と数を検査します。詳細については，第 14 章「データベースの検査」を参照してください。

データ駆動型テストを使った作業

- アプリケーションをテストするときには，複数のセットのデータに対して同じ操作を実行したらどうなるか検査したいことがあります。10 回実行するループを持つ「**データ駆動型**」テストを作成することができます。ループを実行するごとに，テストは異なるデータ・セットによって駆動されます。WinRunner でテストを駆動するデータを使用するには，駆動するテスト・スクリプトにデータをリンクしなければなりません。これを，テストを「**パラメータ化**」するといいます。データは「**データ・テーブル**」に格納されます。これらの操作は手動で行うことができます。あるいは，データ駆動テスト・ウィザードを使用してテストをパラメータ化してデータ・テーブルにデータを格納することもでき

ます。詳細については，第 17 章「データ駆動型テストの作成」を参照してください。

テストへの同期化ポイントの追加

同期化ポイントを使って，テストとアプリケーションの間で生じる可能性のあるタイミングの問題を解決できます。例えば，データベース・アプリケーションを開くテストを作成した場合，同期化ポイントを追加して，データベースの記録が完全に画面に表示されるまで，テストを待機させることができます。

アナログ・テストの場合に，同期化ポイントを使って，WinRunner がウィンドウを特定の位置に確実に再配置させることができます。アナログ・テストを実行すると，マウス・ポインタは正確な座標をたどります。ウィンドウを再配置するとマウス・ポインタがウィンドウ内の正しい要素を見失わないようにすることができます。詳細については，第 18 章「テスト実行の同期化」を参照してください。

トランザクションの測定

トランザクションを定義することで，テストの特定セクションの実行所要時間を測定することができます。トランザクションは，測定の対象としたいビジネス・プロセスを表します。テスト内にトランザクションを定義するには，テストの該当するセクションを **start_transaction** ステートメントと **end_transaction** ステートメントではさみます。例えば，特定のフライトの座席が予約できるまでの所要時間や，クライアントの端末に確認情報が表示されるまでの所要時間を計測するトランザクションを定義できます。

各トランザクションは，**declare_transaction** ステートメントを使用して，テストの中で，それぞれに対応する **start_transaction** ステートメントの前の任意の場所に宣言しておく必要があります。すべてのトランザクションをテストの先頭で宣言することができます。あるいは，それぞれのトランザクションを，対応する **start_transaction** ステートメントの直前で宣言することもできます。

テストの実行時には，**start_transaction** ステートメントは，時間測定の開始を知らせます。時間の測定は，**end_transaction** ステートメントに遭遇するまで続けられます。テスト・レポートに，トランザクションの実行に要した時間が表示されます。

トランザクションの計画時には，次の点を考慮してください。

- テストに追加できるトランザクションの数には上限がありません。
- トランザクション終了の前に同期化ポイントを挿入することをお勧めします。
- トランザクションを入れ子にすることも可能ですが，それぞれの **start_transaction** ステートメントは，対応する **end_transaction** ステートメントと対応付けられている必要があります。

注：

特定のトランザクションについて **end_transaction** ステートメントが存在しない場合，テスト結果にはトランザクション時間は報告されません。

start_transaction に指定した名前が，対応する **end_transaction** の前に複数回使用されている場合には，テスト実行が繰り返しの **start_transaction** ステートメントに達したときに計測が再スタートします（0 にリセット）。

---

declare_transaction，**start_transaction**，および **end_transaction** の各ステートメントは手作業で入力できるほか，［**挿入**］＞［**トランザクション**］オプションを選択してこれらのステートメントを挿入します。

［挿入］＞［トランザクション］オプションを使用してトランザクション・ステートメントを入力するには，次の手順を実行します。

1 隣り合う行に **declare_transaction** ステートメントと **start_transaction** ステートメントを挿入したい場合には，手順 4 に進みます。

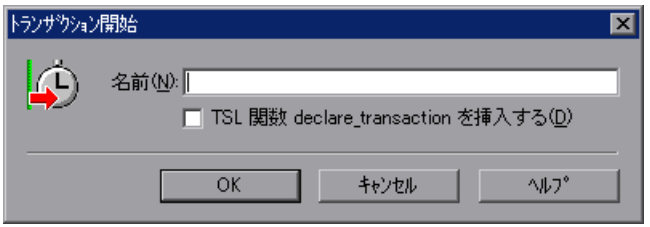
declare_transaction ステートメントを **start_transaction** ステートメントの 3 行以上手前に挿入したい場合には，トランザクションを宣言する位置にカーソルを置きます。

2 ［**挿入**］＞［**トランザクション**］＞［**トランザクションを宣言**］を選択します。［トランザクションの宣言］ダイアログ・ボックスが開きます。

3 トランザクションに与える名前を入力し，［**OK**］をクリックします。**declare_transaction** ステートメントがテストに追加されます。

4 トランザクションの測定を開始する行の先頭にカーソルを置きます。

5 ［**挿入**］＞［**トランザクション**］＞［**トランザクション開始**］を選択します。［トランザクション開始］ダイアログ・ボックスが開きます。



6 トランザクションに与える名前を入力します。

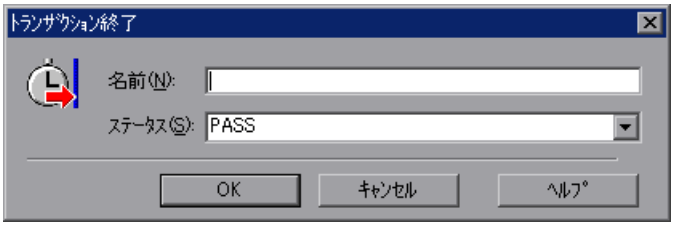
テストにすでに **declare_transaction** ステートメントを挿入している場合，**start_transaction** の名前は，**declare_transaction** ステートメントで指定した名前と同じでなければなりません。トランザクション名の大文字小文字は区別されます。

7 このトランザクションに対する **declare_transaction** ステートメントをまだ入力しておらず，その宣言を **start_transaction** ステートメントの直前の行に挿入したい場合には，［**TSL 関数 declare_transaction を挿入する**］チェックボックスを選択します。

8 ［**OK**］をクリックします。**start_transaction**（および，該当する場合には，**declare_transaction**）ステートメントがテストに追加されます。

9 トランザクション測定の終了を示す行の下にカーソルを置きます。

10 ［**挿入**］＞［**トランザクション**］＞［**トランザクション終了**］を選択します。［トランザクション終了］ダイアログ・ボックスが開きます。



11 終了するトランザクションの名前を入力します。指定するトランザクション名は，**declare_transaction** ステートメントおよび **start_transaction** ステートメントで使用している名前と同じでなければなりません。トランザクション名の大文字小文字は区別されます。

12 トランザクションに割り当てる合否ステータスを選択します。

13 ［**OK**］をクリックします。

declare_transaction，**start_transaction**，および **end_transaction** ステートメントを手作業で挿入する方法の詳細については，「**TSL リファレンス**」を参照してください。

ソフトキーを使用したテスト作成コマンドのアクティブ化

ソフトキーを使用して，WinRunner のコマンドのいくつかをアクティブにできます。WinRunner は WinRunner ウィンドウが画面上でアクティブでなくても，あるいは最小化されていても，ソフトキーからの入力を読み取ります。ソフトキーは自分で設定できます。詳細については，第 42 章「WinRunner のユーザ・インタフェースの カスタマイズ」を参照してください。

以下の表に，テスト作成用のソフトキーの標準の設定を示します。

コマンド	標準のソフトキーの組み合わせ	機能
RECORD	F2	テストの記録を開始します。記録中，このソフトキーはコンテキスト・センシティブ・モードとアナログ・モードの間でトグルします。
CHECK GUI FOR SINGLE PROPERTY	右 Alt + F12	GUI オブジェクトの単数のプロパティを検査します。
CHECK GUI FOR OBJECT/WINDOW	右 Ctrl+ F12	1 つのオブジェクトかウィンドウに GUI チェックポイントを作成します。
CHECK GUI FOR MULTIPLE OBJECTS	F12	［GUI チェックポイント作成］ダイアログ・ボックスを開きます。
CHECK BITMAP OF OBJECT/WINDOW	左 Ctrl+ F12	オブジェクトまたはウィンドウのビットマップをキャプチャします。
CHECK BITMAP OF SCREEN AREA	左 Alt+ F12	領域ビットマップをキャプチャします。
CHECK DATABASE (DEFAULT)	右 Ctrl+ F9	データベースの全内容に対する検査を作成します。
CHECK DATABASE (CUSTOM)	右 Alt+ F9	カラム，行，データベースの特定の情報の数を検査します。
RUNTIME RECORD CHECK	右 Alt + F10	実行時レコード・チェックポイント・ウィザードが開きます。
SYNCHRONIZE OBJECT/WINDOW PROPERTY	右 Ctrl + F10	WinRunner にオブジェクトまたはウィンドウが期待値を取得するのを待機するよう命令します。
SYNCHRONIZE BITMAP OF OBJECT/WINDOW	左 Ctrl+ F11	WinRunner に特定のオブジェクトまたはウィンドウ・ビットマップが現れるまで待機するよう命令します。
SYNCHRONIZE BITMAP OF SCREEN AREA	左 Alt+ F11	WinRunner に特定の領域ビットマップが現れるまで待機するよう命令します。

コマンド	標準のソフトキーの組み合わせ	機能
GET TEXT FROM OBJECT/WINDOW	F11	オブジェクトまたはウィンドウ内のテキストをキャプチャします。
GET TEXT FROM SCREEN AREA	右 Alt+ F11	指定された領域でテキストをキャプチャします。
INSERT FUNCTION FOR OBJECT/WINDOW	F8	GUI オブジェクトに TSL 関数を挿入します。
INSERT FUNCTION FROM FUNCTION GENERATOR	F7	［関数ジェネレータ］ダイアログ・ボックスを開きます。
CALL QUICKTEST TEST	左 Ctrl + q	QuickTest テストへの呼び出しを挿入します。
DECLARE TRANSACTION	左 Ctrl + 4	**declare_transaction** ステートメントを挿入します。
START TRANSACTION	左 Ctrl + 5	**start_transaction** ステートメントを挿入します。
END TRANSACTION	左 Ctrl + 6	**end_transaction** ステートメントを挿入します。
DATA TABLE	左 Ctrl + 8	既存のデータ・テーブルを開くまたは新しいデータ・テーブルを作成します。
PARAMETERIZE DATA	左 Ctrl + 9	［データのパラメータ化］ダイアログ・ボックスを開きます。
DATA DRIVER WIZARD	左 Ctrl + 0	データ駆動テスト・ウィザードを開きます。
STOP	左 Ctrl+ F3	テストの記録を停止します。

テストのプログラミング

プログラミングを行って，テスト・スクリプトをまるごと作成したり，記録したテストを強化したりできます。WinRunner には，関数ジェネレータというビジュアル・プログラミング・ツールがあります。これを使って，TSL 関数をテスト・スクリプトに誤りなく簡単に追加できます。アプリケーションのオブジェクトをポイントするか，リストから関数を選択するだけで，関数コールを生成できます。詳細については，第 34 章「関数の生成」を参照してください。

また，変数，フロー制御ステートメント，配列，ユーザ定義関数など，一般的なプログラミング機能をテスト・スクリプトに追加できます。これらの要素は，テスト・スクリプトに直接入力することができます。プログラミングを行ってテスト・スクリプトを作成する方法の詳細については，「TSL を使ったプログラミング」を参照してください。

テストの編集

テスト・スクリプトを変更するには，［編集］メニューのコマンドか，それに対応するツールバーのボタンを使用します。以下のコマンドを使用できます。

編集コマンド	説明
［元に戻す］	直前の編集操作を取り消します。
［やり直し］	元に戻した操作を再度実行します。
［切り取り］	テスト・スクリプトで選択されているテキストを削除し，それをクリップボードに置きます。
［コピー］	選択されているテキストをコピーし，それをクリップボードに置きます。
［貼り付け］	クリップボードのテキストを挿入ポイントに貼り付けます。
［削除］	選択されているテキストを削除します。
［すべて選択］	アクティブなテスト・ウィンドウの全テキストを選択します。
［コメント］	テキストで選択した行を，行頭に # 記号を付けてコメントにします。コメントになったテキストは，赤のイタリック体で表示されます。

編集コマンド	説明
［コメント解除］	選択されたテキストのコメント行を，行頭の # 記号を削除して実行コードにします。テキストは黒の標準文字で表示されます。
［インデントを増加］	テキストで選択した行をタブ・ストップ 1 つ分右に移動します。タブ・ストップのサイズは［編集オプション］ダイアログ・ボックスで変更できます。詳細は，第 41 章「テスト・スクリプト・エディタのカスタマイズ」を参照してください。
［インデントを減少］	テキストで選択した行のタブ・ストップ 1 つ分左に移動します。タブ・ストップのサイズは［編集オプション］ダイアログ・ボックスで変更できます。詳細は，第 41 章「テスト・スクリプト・エディタのカスタマイズ」を参照してください。
［検索］	アクティブなテスト・ウィンドウで指定された文字（文字列）を検索します。
［次を検索］	指定された文字（文字列）をファイルの下方向に検索します。
［前を検索］	指定された文字（文字列）をファイルの上方向に検索します。
［置換］	指定された文字（文字列）を検索し，それを新しい文字（文字列）で置き換えます。
［移動先行番号指定］	挿入ポイントを，テスト・スクリプトの指定された行に移動します。

テスト・ファイルの管理

［**ファイル**］メニューのコマンドを使って，テスト・ファイルを開いたり，作成，保存，印刷を行ったりできます。

新規テストの作成

［**ファイル**］＞［**新規作成**］を選択するか［**新規作成**］をクリックします。「*Noname*」という文字列に数字が付加されたタイトル（例えば，「*Noname7*」）を持つ新しいウィンドウが開きます。これで，テスト・スクリプトの記録またはプログラミングが行えます。

注：新規スクリプト化コンポーネントを作成するには，上記の手順に従って，テストを作成し，ドキュメントをスクリプト化コンポーネントとして保存する必要があります。

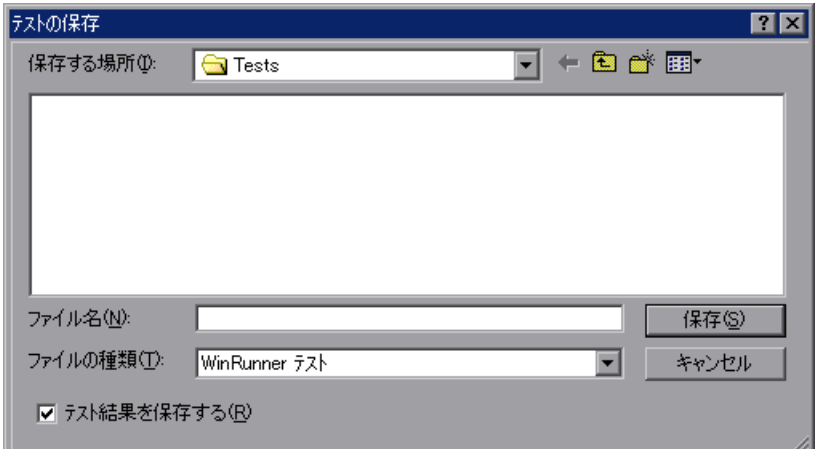
テストの保存

テストを保存するには，以下の方法があります。

- 以前に保存したテストへの変更を保存するには，［**ファイル**］＞［**上書き保存**］を選択するか，ツールバーで［**保存**］をクリックします。
- 新規テストをファイル・システムまたは Quality Center に保存するには，［**ファイル**］＞［**テストとして保存**］を選択するか，ツールバーで［**保存**］をクリックします。
- 開いている複数のファイルを一度に保存するには，［**ファイル**］＞［**すべて保存**］を選択します。
- 新規テスト・スクリプトをスクリプト化コンポーネントとして Quality Center に保存するには，[**ファイル**] ＞ [**スクリプト化コンポーネントとして保存**] コマンドを選択するか，［**保存**］をクリックします。

ファイル・システムにテストを保存するには，次の手順を実行します。

1 [**ファイル**] メニューで [**保存**] または [**テストとして保存**] コマンドを選択するか，ツールバーで [**保存**] をクリックします。[テストを保存] ダイアログ・ボックスが開きます。



2 [**保存する場所**] ボックスで，テストを保存したい場所をクリックします。

3 [**ファイル名**] ボックスにテストの名前を入力します。

4 [**テスト結果を保存する**] チェック・ボックスを選択するかクリアして，既存のテスト結果をテストと一緒に保存するかどうかを指定します。

このボックスをクリアすると，テスト結果ファイルはテストと一緒に保存されず，テスト結果ファイルを後から表示することができなくなります。テスト結果を後から分析する必要がない場合や，既存のテストを別名で保存しており，テスト結果を必要としない場合は，ディスク領域の節約のために，[**テスト結果を保存する**] チェック・ボックスをクリアします。

注：標準設定では，このオプションは新規テストを保存する場合（[**保存**]）に選択され，既存のテストを新しい名前で保存する場合（[**テストとして保存**]）にはクリアします。

---

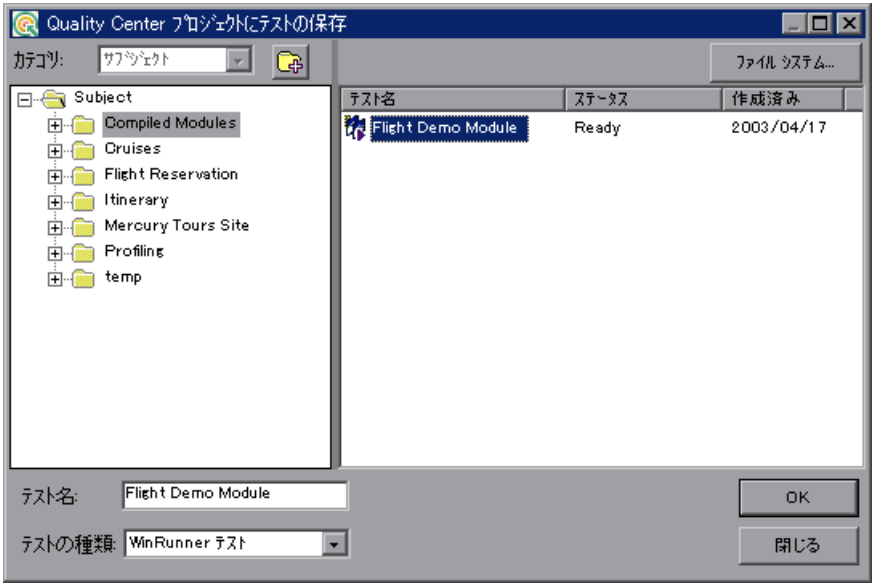
5 [**保存**] をクリックして，テストを保存します。

Quality Center プロジェクトにテストを保存するには，次の手順を実行します。

注：Quality Center データベースにテストを保存できるのは，Quality Center プロジェクトに接続されている場合のみです。詳細については，第 48 章「テスト工程の管理」を参照してください。

---

1 Quality Center プロジェクトに接続したら，［**ファイル**］＞［**テストとして保存**］を選択します。［Quality Center プロジェクトにテストの保存］ダイアログ・ボックスが開きます。



Quality Center テスト計画モジュールのテスト計画ツリーが表示されます。

［**Quality Center プロジェクトにテストの保存**］ダイアログ・ボックスは，WinRunner が Quality Center プロジェクトに接続されているときのみ開きます。

2 テスト計画ツリーで該当するサブジェクト・フォルダを選択するか，［**新規フォルダ**］ボタンをクリックして新規フォルダを作成します。サブジェクト・ツリーを展開するには，閉じているフォルダのアイコンをダブルクリックします。ツリーを閉じるには，開いているフォルダのアイコンをダブルクリックします。

3 ［**テスト名**］テキスト・ボックスにテストの名前を入力します。テストが識別しやすいように，分かりやすい名前を使用します。

4 ［**OK**］をクリックしてテストを保存し，ダイアログ・ボックスを閉じます。

注：［**ファイル システム**］ボタンをクリックして，［テストを保存］ダイアログ・ボックスを開き，ファイル・システムにテストを保存できます。

次回 Quality Center を開始するか，テスト計画ツリーをテスト計画モジュールで更新すると，新規テストがツリーに表示されます。詳細については，『**Mercury Quality Center ユーザーズ・ガイド**』を参照してください。

Quality Center プロジェクトへのテストの保存の詳細については，第 48 章「テスト工程の管理」を参照してください。

既存のテストを開く

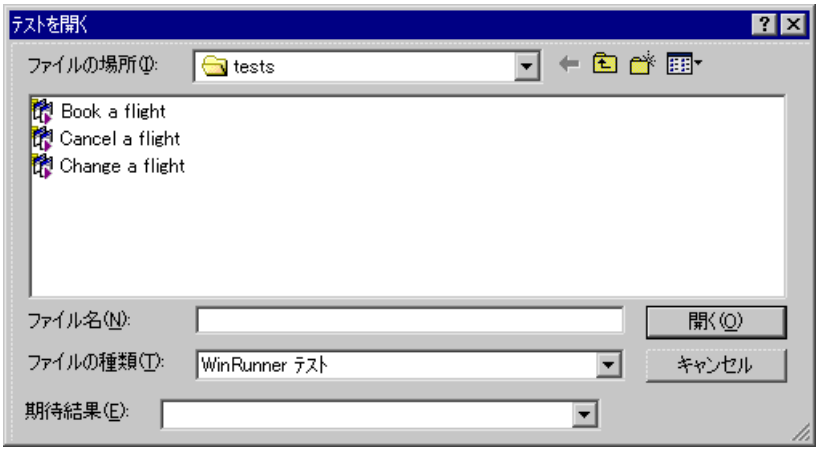
ファイル・システムまたは Quality Center プロジェクトから既存のテストを開くことができます。

Quality Center プロジェクトからスクリプト化コンポーネントを開くこともできます。詳細については，121 ページ「既存のスクリプト化コンポーネントを開く」を参照してください。

注：100 以上のテストを一度に開くことはできません。

ファイル・システムからテストを開くには，次の手順を実行します。

1 ［**ファイル**］＞［**テストを開く**］を選択するか，［**開く**］をクリックして，［**テストを開く**］ダイアログ・ボックスを開きます。



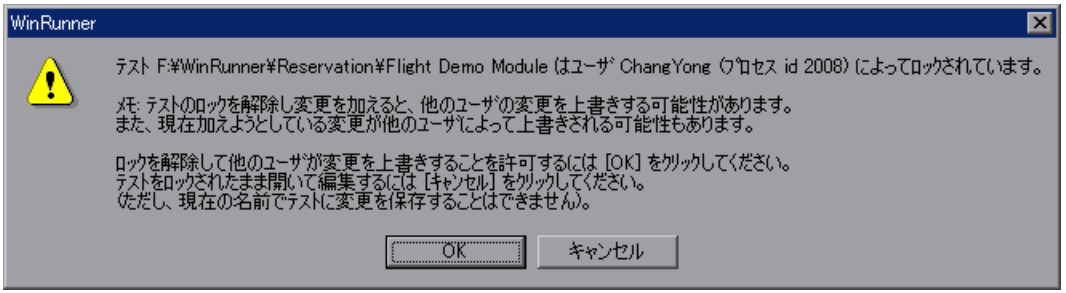
2 **ディレクトリ名**のボックスで，開きたいテストの場所をクリックします。

3 ［**ファイル名**］ボックスで，開きたいテストの名前をクリックします。

4 テストに複数の期待結果がある場合，使用するフォルダを［**期待結果**］リストで選択します。標準のフォルダは，「*exp*」です。

5 ［**開く**］をクリックすると，テストが開きます。

他の WinRunner ユーザによって開かれているテストを開くよう選択すると，次のようなメッセージが表示されます。



［**キャンセル**］をクリックして，テストをロックされた編集可能なテストとして開きます。テストを編集し実行できますが，現在の名前で保存することはできません。

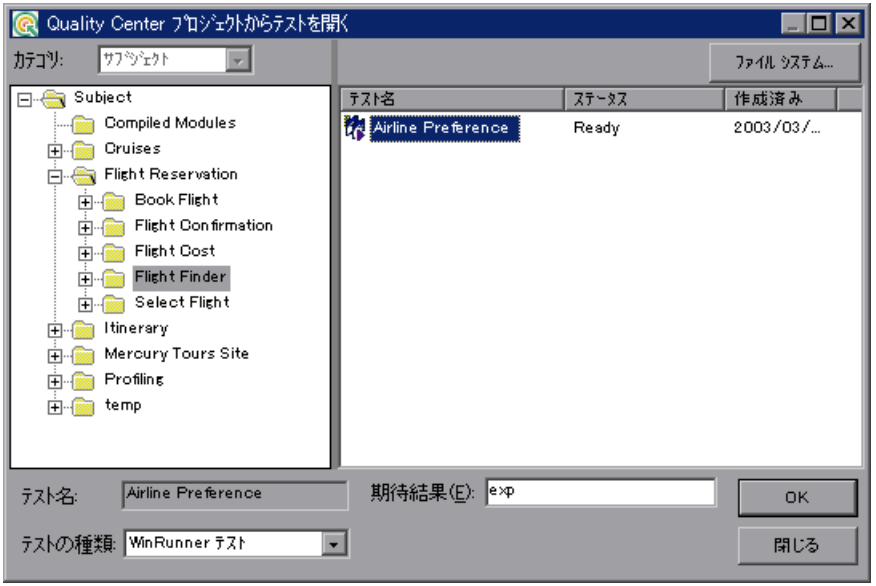
他のユーザの作業を妨げないことが分かっている場合のみ［**OK**］をクリックしてテストのロックを解除します。

Quality Center プロジェクトのテストを開くには，次の手順を実行します。

注：Quality Center データベースからテストを開けるのは，Quality Center プロジェクトに接続されている場合のみ 詳細については，第 48 章「テスト工程の管理」を参照してください。

---

1 ［**ファイル**］＞［**テストを開く**］を選択するか，［**開く**］をクリックします。Quality Center プロジェクトに接続されていれば，［**Quality Center プロジェクトからテストを開く**］ダイアログ・ボックスが開き，テスト計画ツリーを表示します。



［**Quality Center データベースからテストを開く**］ダイアログ・ボックスは，WinRunner が Quality Center プロジェクトに接続されているときのみ開きます。

2 テスト計画ツリーで該当するサブジェクトをクリックします。ツリーを広げて下位のレベルを表示するには，閉じているフォルダをダブルクリックします。ツリーを閉じるには，開いているフォルダをダブルクリックします。

サブジェクトを選択すると，そのサブジェクトに関するテストが［テスト名］リストに表示されます。

3 ［**テスト名**］リストでテストを選択します。テストが読み取り専用の［**テスト名**］ボックスに表示されます。

4 必要があれば，［**期待結果**］ボックスにテストの期待結果フォルダを入力します（または標準のディレクトリが使用されます）。

5 ［**OK**］をクリックすると，テストが開きます。テストが［WinRunner］ウィンドウで開きます。テスト・ウィンドウのタイトル・バーにサブジェクトの完全パスが表示されている点に注目してください。

注 : ［ファイルシステム］ボタンをクリックして，［テストを開く］ダイアログ・ボックスを開き，ファイル・システムからテストを開きます。

Quality Center プロジェクトでテストを開く方法の詳細については，第 48 章「テスト工程の管理」を参照してください。

既存のスクリプト化コンポーネントを開く

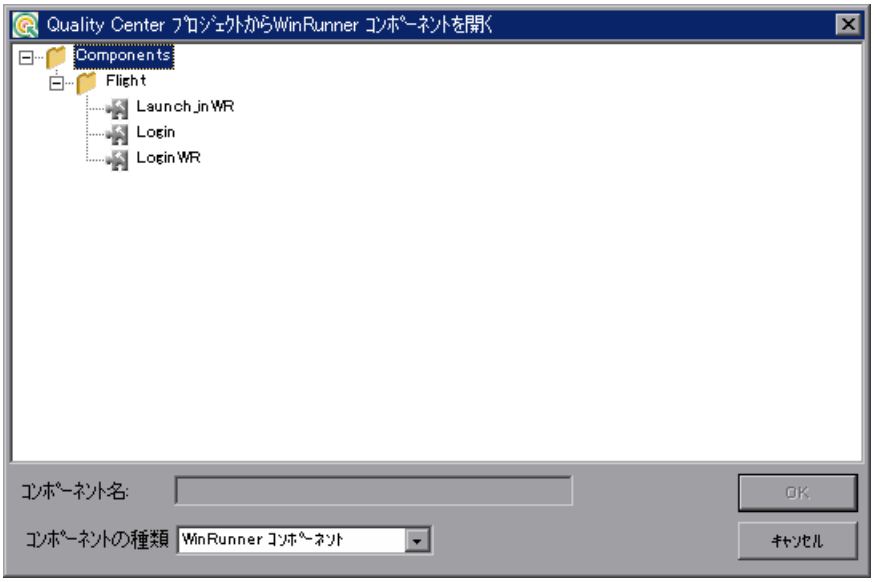
WinRunner スクリプト化コンポーネントは，ビジネス・プロセス・テスティング・サポート付きの Quality Center でのビジネス・プロセス・テストに含めることができます。ただし，WinRunner スクリプト化コンポーネントは Quality Center で編集はできません。既存の WinRunner スクリプト化コンポーネントは，WinRunner で必要に応じて表示 / 編集できます。

Quality Center プロジェクトからスクリプト化コンポーネントを開くには，次の手順を実行します。

注：Quality Center データベースからスクリプト化コンポーネントを開くには，Quality Center プロジェクトに接続している必要があります。詳細については，第 48 章「テスト工程の管理」を参照してください。

---

1 Quality Center プロジェクトに接続したら，［**ファイル**］＞［**スクリプト化コンポーネントを開く**］を選択するか，CTRL+H を押します。［Quality Center プロジェクトから WinRunner コンポーネントを開く］ダイアログ・ボックスが開き，コンポーネント・ツリーが表示されます。



注：［ファイル］メニューの［スクリプト化コンポーネントを開く］オプションは，Business Process Testing がサポートされている Quality Center に接続している場合にのみ表示されます。

---

2 関係するコンポーネントをコンポーネント・ツリーの中で選択します。ツリーを展開して下位レベルを表示するには，閉じているフォルダをダブルクリックします。ツリーを折りたたむには，開いているフォルダをダブルクリックします。スクリプト化コンポーネントが読み取り専用の［**コンポーネント名**］ボックスに表示されます。

3 ［**OK**］をクリックしてスクリプト化コンポーネントを開きます。コンポーネントは，WinRunner 内のウィンドウに開きます。WinRunner のタイトル・バーには，スクリプト化コンポーネントのサブジェクト・パス全体が表示されます。

4 必要に応じてコンポーネントを表示または編集します。

Quality Center プロジェクトのスクリプト化コンポーネントを開くことの詳細については，第 48 章「テスト工程の管理」

WinRunner テストの圧縮と解凍

［**Zip ファイルにエクスポート**］オプションを使用して，WinRunner テストを圧縮して配布しやすくできます。このオプションを選択すると，データ・テーブル，テスト結果，GUI ファイルを含め，テスト・フォルダに保存されているすべてのファイルが圧縮されます。テスト・フォルダ以外の場所に格納されている外部ファイルは圧縮されません。

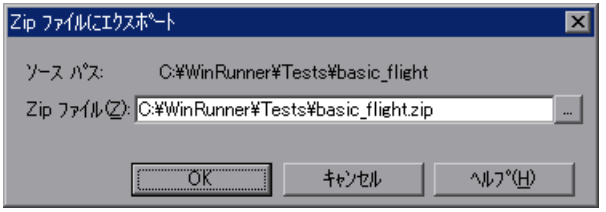
［**Zip ファイルからインポート**］オプションを使用して，［**Zip ファイルにエクスポート**］オプションを使用して圧縮された任意のテストからファイルを取り出せます。このオプションを使用して，別のユーティリティを使用して圧縮したテストからファイルを取り出すことはできません。

テストを圧縮するには，次の手順を実行します。

1 圧縮するテストを開きます。

2 開いているテストに未保存の変更がある場合は，テストを保存します。

3 [**ファイル**] > [**Zip ファイルにエクスポート**] を選択します。[Zip ファイルにエクスポート] ダイアログ・ボックスが開き，テストのソース・パスと推奨する圧縮ファイル名が表示されます。

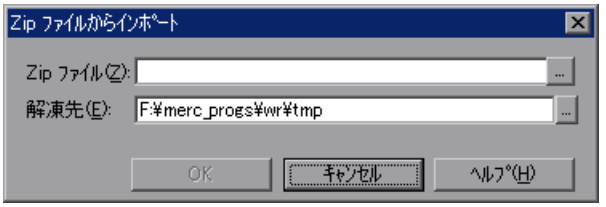


4 標準設定の圧縮ファイル名を使用するか，新しいファイル名を指定します。

5 [**OK**] をクリックします。テスト圧縮のプログレス・バーがダイアログ・ボックスに表示されます。圧縮プロセスが終了すると，ダイアログ・ボックスが閉じます。

圧縮したテストを解凍するには，次の手順を実行します。

1 [**ファイル**] > [**Zip ファイルからインポート**] を選択します。[Zip ファイルからインポート] ダイアログ・ボックスが開きます。



2 解凍する圧縮テストの場所を入力するか参照します。

3 テストを解凍する標準設定の場所を使用するか，新しい場所を指定します。

4 [**OK**] をクリックします。テスト解凍のプログレス・バーがダイアログ・ボックスに表示されます。解凍プロセスが終了すると，ダイアログ・ボックスが閉じ，解凍されたテストが WinRunner ウィンドウに表示されます。

テストの印刷

テスト・スクリプトを印刷するには，[**ファイル**] > [**印刷**] を選択します。[印刷] ダイアログ・ボックスが表示されます。

- 印刷オプションを選択します。
- ［**OK**］をクリックすると，印刷が開始されます。

テストを閉じる

- 現在のテストを閉じるには，［**ファイル**］＞［**閉じる**］を選択します。
- 開いている複数のテストを一度に閉じるには，［**ファイル**］＞［**すべて閉じる**］を選択します。

第 9 章

GUI オブジェクトの検査

テスト・スクリプトに GUI チェックポイントを追加することで，アプリケーションの動作を異なるバージョン間で比較できます。

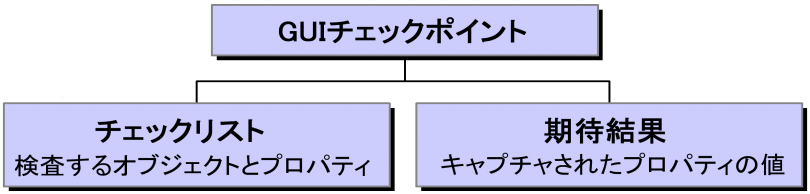
本章では，以下の項目について説明します。

- GUI オブジェクトの検査について
- 単数のプロパティ値の検査
- 単数のオブジェクトの検査
- ウィンドウ内の複数のオブジェクトの検査
- ウィンドウ内のすべてのオブジェクトの検査
- GUI チェックポイント・ステートメントについて
- GUI チェックポイントでの既存の GUI チェックリストの使用
- GUI チェックリストの変更
- GUI チェックポイント・ダイアログ・ボックスについて
- プロパティ検査と標準の検査
- プロパティ検査への引数の指定
- プロパティの期待値の編集
- GUI チェックポイントの期待結果の変更

GUI オブジェクトの検査について

テスト・スクリプトで GUI チェックポイントを使って，アプリケーションの GUI オブジェクトを検査し，不具合を検出できます。例えば，特定のダイアログ・ボックスを開いたときに［OK］，［キャンセル］，［ヘルプ］ボタンが有効になることを検証できます。

対象となる GUI オブジェクトを指し，WinRunner に検査させたいプロパティを選択します。WinRunner が推奨する標準のプロパティを検査することも，検査したいプロパティを選択することもできます。GUI オブジェクトと選択したプロパティに関する情報は「チェックリスト」に保存されます。その後，WinRunner は GUI オブジェクトのプロパティの現在値をキャプチャして，その情報を「期待結果」として保存します。そして，「GUI チェックポイント」がテスト・スクリプトに自動的に挿入されます。このチェックポイントは，テスト・スクリプトに obj_check_gui または win_check_gui ステートメントとして記録されます



テストを実行すると，WinRunner はテスト対象アプリケーションにおける GUI オブジェクトの現在の状態と期待結果を比較します。期待結果と現在の結果が一致しない場合，GUI チェックポイントは失敗となります。GUI チェックポイントはループに含めることができます。GUI チェックポイントがループ内で実行されると，チェックポイントの各反復の結果は別のエントリとしてテスト結果に表示されます。チェックポイントの各反復の結果は，WinRunner の［テスト結果］ウィンドウに表示できます。詳細については，第 20 章「テスト結果の分析」を参照してください。

検査する GUI オブジェクトで，GUI マップにまだ含まれていないものは，自動的に仮 GUI マップ・ファイルに追加されます。詳細については，第 3 章「WinRunner の GUI オブジェクトの識別方法」を参照してください。

正規表現を使用して，名前が可変の編集オブジェクトや静的テキスト・オブジェクトに GUI チェックポイントを作成できます。詳細については，第 27 章「正規表現の使い方」のを参照してください。

WinRunner は，Active X コントロール，Visual Basic，Power Builder などの様々なアプリケーション開発環境に対する専用のサポートを組み込みで提供します。適切なアドイン・サポートをロードすれば，WinRunner はこれらのコントロールを認識して標準の GUI オブジェクトを処理するのと同様に処理します。こうしたオブジェクトを対象に，標準の GUI オブジェクトの場合と同じように，GUI チェックポイントを作成できます。WinRunner は，ActiveX と Visual Basic のサブオブジェクトを検査するための付加的な専用のサポートを組み込みで提供します。

詳細については，第 11 章「ActiveX と Visual Basic のコントロールの使用」を参照してください。WinRunner の PowerBuilder 用サポートについては，第 12 章「PowerBuilder のアプリケーションの検査」を参照してください。

テーブルの内容とプロパティを検査する GUI チェックポイントを作成することもできます。詳細に付いては，第 13 章「テーブル内容の検査」を参照してください。

失敗した GUI チェックポイントのオプションの設定

GUI チェックポイントが失敗するたびに，選択した受信者に電子メールを送信するよう，またチェックポイントが失敗したウィンドウまたは画面のビットマップをキャプチャするよう，WinRunner に指示できます。これらのオプションは［一般オプション］ダイアログ・ボックスで設定します。

GUI チェックポイントが失敗したら電子メールを送信するよう WinRunner に指示するには，次の手順を実行します。

1 ［**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・ボックスが開きます。

2 オプション表示枠で［**通知**］カテゴリを選択します。通知オプションが表示されます。

3 ［**GUI チェックポイントの失敗**］を選択します。

4 オプション表示枠で［**通知**］＞［**電子メール**］カテゴリを選択します。電子メール・オプションが表示されます。

5 ［**電子メールのサービスを有効にする**］オプションを選択して，関連するサーバと送信者情報を設定します。

6 オプション表示枠で［**通知**］＞［**受信者**］カテゴリを選択します。電子メールの受信者オプションが表示されます。

7 必要に応じて受信者の追加，削除，変更を行い，GUI チェックポイントの失敗時に電子メールを送信する受信者を設定します。

電子メールには，テストとチェックポイントの詳細なサマリと，プロパティ検査の期待値と実際の値が含まれます。

詳細については，560 ページ「通知オプションの設定」を参照してください。

チェックポイントが失敗したら，ビットマップをキャプチャするよう WinRunner に指示するには，次の手順を実行します。

1 ［**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・ボックスが開きます。

2 オプション表示枠で［**実行**］＞［**設定**］カテゴリをクリックします。実行設定オプションが表示されます。

3 ［**検証失敗の時，ビットマップをキャプチャする**］を選択します。

4 ［**Window**］，［**Desktop**］または［**Desktop Area**］を選択して，チェックポイントの失敗時にキャプチャするものを指定します。

5 ［**Desktop Area**］を選択した場合は，キャプチャするデスクトップの座標を指定します。

テストを実行すると，キャプチャされたビットマップが結果フォルダに保存されます。

詳細については，545 ページ「テストの実行オプションの設定」を参照してください。

単数のプロパティ値の検査

GUI オブジェクトの単数のプロパティを検査できます。例えば，ボタンが使用可能かどうか，またはリストの項目が選択されているかどうかを検査できます。プロパティ値で GUI チェックポイントを作成するには，［プロパティのチェック］ダイアログ・ボックスを使用して次の関数から 1 つをテスト・スクリプトに追加します。

button_check_info	**scroll_check_info**
edit_check_info	**static_check_info**

list_check_info **win_check_info**

obj_check_info

これらの関数の使い方については，「**TSL リファレンス**」を参照してください。

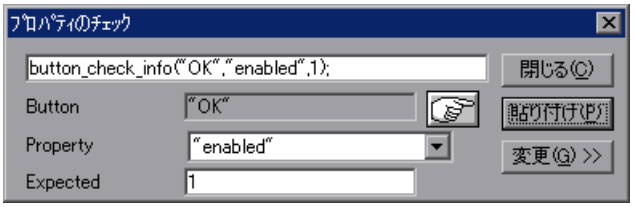
プロパティ値で GUI チェックポイントを作成するには以下のようにします。

1 ［**挿入**］＞［**GUI チェックポイント**］＞［**単数プロパティ**］を選択します。アナログ・モードで記録する場合は，無関係なマウス動作を避けるため，［GUI チェックポイント－単数のプロパティ］ソフトキーを押します。

WinRunner のウィンドウが最小化されると，マウス・ポインタが指差し型になり，画面にヘルプ・ウィンドウが開きます。

2 オブジェクトをクリックします。

［プロパティのチェック］ダイアログ・ボックスが開き，選択されたオブジェクトの標準の関数を表示します。WinRunner は引数の値を自動的に関数に割り当てます。



3 プロパティ検査の属性は変更できます。

- 割り当てられた属性を変更するには，［**Property**］リストから値を選択します。期待値を［expected］テキスト・ボックスで更新します。
- 異なるオブジェクトを選択するには，指差しボタンをクリックしてから，アプリケーションのオブジェクトをクリックします。WinRunner は新しい引数の値を自動的に関数に割り当てます。

  選択した関数と互換性のないオブジェクトをクリックすると，「現在の関数を選択したオブジェクトに適用できません」という旨のメッセージが表示されます。［OK］をクリックしてメッセージを消し，［閉じる］をクリックして［プロパティのチェック］ダイアログ・ボックスを閉じます。ステップ *1* と *2* を繰り返します。

4 ［**貼り付け**］をクリックして，テスト・スクリプトにステートメントを貼り付けます。

関数がスクリプトの挿入ポイントの位置に貼り付けられます。［プロパティのチェック］ダイアログ・ボックスが閉じます。

注 : オブジェクトの別の関数を変更するには，［**変更**］をクリックします。［関数ジェネレータ］ダイアログ・ボックスが開き，関数のリストを表示します。関数ジェネレータの使い方については，第 34 章「関数の生成」を参照してください。

単数のオブジェクトの検査

GUI チェックポイントを作成して，テスト対象アプリケーションの単数のオブジェクトを検査できます。オブジェクトをその標準のプロパティで検査することも，検査するプロパティを指定することもできます。

各標準オブジェクト・クラスには，1 組の標準の検査があります。標準オブジェクト，検査できるプロパティ，標準の検査の全リストは，161 ページ「プロパティ検査と標準の検査」を参照してください。

注 : gui_ver_set_default_checks 関数を使用して，オブジェクトに標準の検査を設定できます。詳細については，「**TSL リファレンス**」と『**WinRunner カスタマイズ・ガイド**』を参照してください。

標準の検査による GUI チェックポイントの作成

WinRunner が推奨する標準の検査をプロパティに対して実行する GUI チェックポイントを作成することができます。例えば，プッシュ・ボタンを検査する GUI チェックポイントを作成すると，標準の検査はプッシュ・ボタンが有効になっていることを検証します。

標準の検査を使って GUI チェックポイントを作成するには，次の手順を実行します。

1 ［**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で［**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。アナログ・モードで記録を行っている場合は，余計なマウスの動きが記録されないように［GUI チェックポイントーオブジェクト / ウィンドウ］ソフトキーを押します。［GUI チェックポイントーオブジェクト / ウィンドウ］ソフトキーはコンテキスト・センシティブ・モードでも使用できます。

［WinRunner］ウィンドウが最小化され，マウス・ポインタが指差し型に変わり，画面にヘルプ・ウィンドウが表示されます。

2 オブジェクトをクリックします。

3 WinRunner は，検査する GUI オブジェクトのプロパティの現在値をキャプチャし，それをテストの期待結果フォルダに格納します。その後，［WinRunner］ウィンドウが再び表示され，GUI チェックポイントが obj_check_gui ステートメントとしてテスト・スクリプトに挿入されます。詳細については 139 ページ「GUI チェックポイント・ステートメントについて」を参照してください。

検査するプロパティを指定した場合の GUI チェックポイントの作成

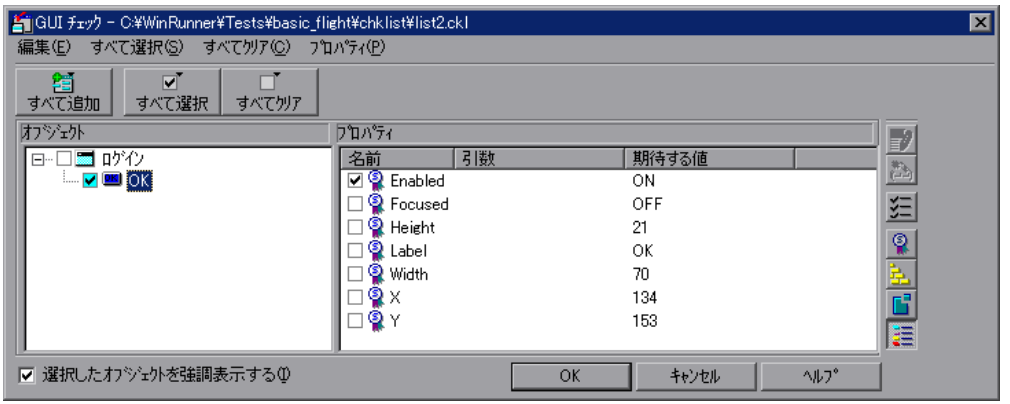
オブジェクトのどのプロパティを検査するか指定することができます。

検査するプロパティを指定して GUI チェックポイントを作成するには，次の手順を実行します。

1 ［**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。アナログ・モードで記録を行っている場合は，余計なマウスの動きが記録されないように［GUI チェックポイントーオブジェクト / ウィンドウ］ソフトキーを押します。［GUI チェックポイントーオブジェクト / ウィンドウ］ソフトキーはコンテキスト・センシティブ・モードでも使用できます。

［WinRunner］ウィンドウが最小化され，マウス・ポイントが指差し型に変わり，画面にヘルプ・ウィンドウが表示されます。

2 **オブジェクトまたはウィンドウをダブルクリックします。[GUI チェック] ダイアログ・ボックスが表示されます。**



3 [**オブジェクト**] 表示枠でオブジェクト名をクリックします。[**プロパティ**] 表示枠には，選択したオブジェクトのすべてのプロパティがリストされます。

4 検査したいプロパティを選択します。

➤ プロパティの期待値を編集するには，まずプロパティを選択します。次に [**期待結果値を編集**] ボタンをクリックするか，[**期待する値**] カラムをダブルクリックして，これを編集します。詳細については，173 ページ「プロパティの期待値の編集」を参照してください。

➤ 検査を追加して引数を指定するには，まず，引数を指定したいプロパティを選択します。次に [**引数を指定**] ボタンをクリックするか，[**引数**] カラムをダブルクリックします。[引数] カラムに省略記号（3 つの点）が表示されている場合は，このプロパティの検査に引数を指定しなければなりません（標準の引数が指定されている場合は，引数を指定する必要はありません）。標準のオブジェクトを検査する場合は，編集オブジェクトと静的テキスト・オブジェクトの特定のプロパティにのみ引数を指定します。また，非標準オブジェクトの特定のプロパティの検査にも引数を指定します。詳細については，167 ページ「プロパティ検査への引数の指定」を参照してください。

➤ オブジェクトのプロパティの表示オプションを変更するには，プロパティ表示の切り替え用ボタンを使用します。詳細については，151 ページ「[GUI チェック] ダイアログ・ボックス」を参照してください。

5 [**OK**] をクリックして，[GUI チェックポイント] ダイアログ・ボックスを閉じます。

WinRunner は，GUI 情報をキャプチャし，それをテストの期待結果フォルダに格納します。その後，WinRunner のウィンドウが再表示され，GUI チェックポイントが **obj_check_gui** または **win_check_gui** ステートメントとしてテスト・スクリプトに挿入されます。詳細については，139 ページ「GUI チェックポイント・ステートメントについて」を参照してください。

［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，148 ページ「GUI チェックポイント・ダイアログ・ボックスについて」を参照してください。

ウィンドウ内の複数のオブジェクトの検査

GUI チェックポイントを使って，1 つのウィンドウ内の複数のオブジェクトを検査できます。標準オブジェクトと検査できるプロパティの全リストは，161 ページ「プロパティ検査と標準の検査」を参照してください。

複数のオブジェクトに GUI チェックポイントを作成するには，次の手順を実行します。

1 ［**挿入**］＞［**GUI チェックポイント**］＞［**複数のオブジェクト**］を選択するか，ユーザ定義ツールバーの［**複数のオブジェクトの GUI チェックポイント**］ボタンをクリックします。アナログ・モードで記録している場合は，余計なマウスの動きが記録されないように［GUI チェックポイント－複数のオブジェクト］ソフトキーを押します。［GUI チェックポイントの作成］ダイアログ・ボックスが開きます。

2 ［**追加**］ボタンをクリックします。マウス・ポインタが指差し型に変わり，画面ヘルプ・ウィンドウが表示されます。

3 オブジェクトを 1 つ追加するには，これを 1 度クリックします。ウィンドウのタイトル・バーや，メニュー・バーを クリックすると , ウィンドウ内のすべてのオブジェクトを検査するよう促すメッセージがヘルプ・ウィンドウに表示されます。ウィンドウ内のすべてのオブジェクトの検査については，137 ページ「ウィンドウ内のすべてのオブジェクトの検査」を参照してください。

4 指差しポインタはまだアクティブなままです。検査したいオブジェクトに上記の手順 3 を繰り返して，オブジェクトを必要なだけ選択できます。

注 : 異なるウィンドウからのオブジェクトを 1 つのチェックポイントに挿入することはできません。

---

5 マウスの右ボタンをクリックすると，選択処理が終わり，マウス・ポインタも元の形状に戻ります。［GUI チェックポイント作成］ダイアログ・ボックスが再び開きます。

6 ［オブジェクト］表示枠には，GUI チェックポイントに含まれるウィンドウとオブジェクトの名前が表示されます。検査するオブジェクトを指定するには，［**オブジェクト**］表示枠でオブジェクト名をクリックします。

［プロパティ］表示枠には，オブジェクトのすべてのプロパティが表示されます。標準のプロパティが選択されます。

➤ プロパティの期待値を編集するには，まずプロパティを選択します。次に，［**期待結果値を編集**］ボタンをクリックするか，［**期待する値**］カラムの値をダブルクリックして，これを編集します。詳細については，173 ページ「プロパティの期待値の編集」を参照してください。

➤ 検査を追加して引数を指定するには，まず，引数を指定したいプロパティを選択します。次に［**引数を指定**］ボタンをクリックするか，［**引数**］カラムをダブルクリックします。［引数］カラムに省略記号（3 つの点）が表示されている場合は，このプロパティの検査に引数を指定しなければなりません（標準の引数が指定されている場合は，引数を指定する必要はありません）。標準のオブジェクトを検査する場合は，編集オブジェクトと静的テキスト・オブジェクトの特定のプロパティにのみ引数を指定します。

また，非標準オブジェクトの特定のプロパティの検査にも引数を指定します。詳細については，167 ページ「プロパティ検査への引数の指定」を参照してください。

➤ オブジェクトのプロパティの表示オプションを変更するには，プロパティ表示の切り替え用ボタンを使用します。詳細については，154 ページ「［GUI チェックポイント作成］ダイアログ・ボックス」を参照してください。

7 チェックリストを保存し，［GUI チェックポイント作成］ダイアログ・ボックスを閉じるには，［**OK**］をクリックします。

WinRunner は，選択された GUI オブジェクトのプロパティの現在値をキャプチャし，それを期待結果フォルダに格納します。テスト・スクリプトには，**win_check_gui** ステートメントが挿入されます。詳細については，139 ページ「GUI チェックポイント・ステートメントについて」を参照してください。

［GUI チェックポイント作成］ダイアログ・ボックスの詳細については，148 ページ「GUI チェックポイント・ダイアログ・ボックスについて」を参照してください。

ウィンドウ内のすべてのオブジェクトの検査

GUI チェックポイントを作成して，1 つのウィンドウ内のすべての GUI オブジェクトを対象に標準の検査を行うことができます。1 つのウィンドウ内の全 GUI オブジェクトを対象に実行する検査を指定することもできます。

各標準オブジェクト・クラスには，1 組の標準の検査があります。標準オブジェクト，検査できるプロパティ，標準の検査の全リストは，161 ページ「プロパティ検査と標準の検査」を参照してください。

注：gui_ver_set_default_checks 関数を使って，オブジェクトに標準の検査を設定できます。詳細については，「**TSL リファレンス**」および『**WinRunner カスタマイズ・ガイド**』を参照してください。

標準の検査によるウィンドウ内のすべてのオブジェクトの検査

1 つのウィンドウ内のすべての GUI オブジェクトの標準のプロパティを検査する GUI チェックポイントを作成できます。

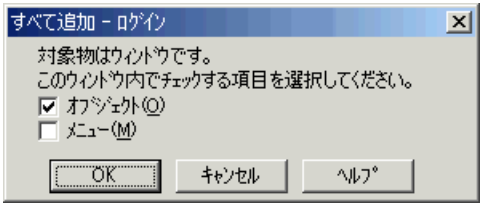
1 つのウィンドウ内のすべての GUI オブジェクトを対象に標準の検査を行う GUI チェックポイントを作成するには，次の手順を実行します。

1 ［**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にある［**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。アナログ・モードで記録している場合，余計なマウスの動きが記録されないように［GUI チェックポイント－オブジェクト / ウィンドウ］ソフトキーを押します。［GUI チェックポイントの作成］ダイアログ・ボックスが開きます。［GUI チェックポイント－オブジェクト / ウィンドウ］ソフトキーはコンテキスト・センシティブ・モードでも使用できます。

［WinRunner］ウィンドウが最小化され，マウス・ポインタが指差し型に変わり，画面にヘルプ・ウィンドウが表示されます。

2 検査したいウィンドウのタイトル・バーまたはメニュー・バーをクリックします。

［**すべて追加**］ダイアログ・ボックスが開きます。



3 ［**オブジェクト**］，［**メニュー**］のどちらか，または両方を選択して，チェックリストに含めるオブジェクトの種類を示します。［オブジェクト］だけを選択した場合（標準の設定），メニューを除くウィンドウ内のすべてのオブジェクトがチェックリストに含まれます。チェックリストにメニューを含めるには，［メニュー］を選択します。

4 ［**OK**］をクリックして，ダイアログ・ボックスを閉じます。

WinRunner は GUI オブジェクトとメニュー項目の期待値をキャプチャし，この情報をテストの期待結果フォルダに格納します。［WinRunner］ウィンドウが再び表示され，テスト・スクリプトに **win_check_gui** ステートメントが挿入されます。

ウィンドウ内のすべてのオブジェクトに実行する検査の指定

GUI チェックポイントを使用して，1 つのウィンドウ内のすべての GUI オブジェクトを対象に実行する検査を指定できます。

1 つのウィンドウ内のすべての GUI オブジェクトに対して実行する検査を指定する GUI チェックポイントを作成するには，次の手順を実行します。

1 ［**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。アナログ・モードで記録している場合，余計なマウスの動きが記録されないように［GUI チェックポイント－オブジェクト / ウィンドウ］ソフトキーを押します。［GUI チェックポイントの作成］ダイアログ・ボックスが開きます。［GUI チェックポイント－オブジェクト / ウィンドウ］ソフトキーはコンテキスト・センシティブ・モードでも使用できます。

［WinRunner］ウィンドウが最小化され，マウス・ポインタが指差し型に変わり，画面にヘルプ・ウィンドウが表示されます。

2 検査したいウィンドウのタイトル・バーまたはメニュー・バーをダブルクリックします。

WinRunner は，ウィンドウ内のすべてのオブジェクトを含む新しいチェックリストを生成します。これには数秒かかります。

［GUI チェック］ダイアログ・ボックスが開きます。

3 実行する検査を指定し，［**OK**］をクリックしてダイアログ・ボックスを閉じます。詳細については，151 ページ「［GUI チェック］ダイアログ・ボックス」を参照してください。

WinRunner は，GUI 情報をキャプチャし，それをテストの期待結果フォルダに格納します。［WinRunner］ウィンドウが再び表示され，テスト・スクリプトに **win_check_gui** ステートメントが挿入されます。

GUI チェックポイント・ステートメントについて

単数オブジェクトの GUI チェックポイントは，スクリプト内に **obj_check_gui** ステートメントとして現れます。1 つのウィンドウの複数の GUI チェックポイントは，スクリプト内に **win_check_gui** ステートメントとして現れます。**obj_check_gui** と **win_check_gui** ステートメントのどちらも，「チェックリスト」と関連付けられ，「期待結果ファイル」に期待結果を格納します。

- 「**チェックリスト**」には，検査する必要のあるオブジェクトとプロパティが表示されます。**obj_check_gui** ステートメントに対しては，チェックリストには 1 つのオブジェクトしか表示されません。**win_check_gui** ステートメントに対しては，チェックリストにはウィンドウ内で検査するすべてのオブジェクトのリストが含まれます。GUI チェックポイントを作成する場合，新しいチェックリストを作成するか，既存のチェックリストを使用します。既存のチェックリストの使用については，141 ページ「GUI チェックポイントでの既存の GUI チェックリストの使用」を参照してください。
- 「**期待結果ファイル**」には，チェックリスト内の各オブジェクトに対するプロパティの期待値が含まれます。これらのプロパティの値は，チェックポイントを作成したときにキャプチャされ，後で手作業で，あるいはテストを更新モードで実行して更新できます。詳細については 433 ページ「期待結果を更新するためのテスト実行」を参照してください。テストを実行するたびに，プロパティの期待値はオブジェクトのプロパティの現在値と比較されます。

obj_check_gui 関数の構文は次のとおりです。

obj_check_gui (object, checklist, expected results file, time);

object は，GUI オブジェクトの論理名です。*checklist* は，検査するオブジェクトとプロパティを定義するチェックリストの名前です。*expected results file* はプロパティの期待値を格納するファイルの名前です。*time* は，前回の入力イベント発生時からプロパティの現在値のキャプチャまでの間の最大遅延間隔を秒単位で示したものです。この間隔は，テスト実行中に **timeout_msec** テスト・オプションの値に加算されます。**timeout_msec** テスト・オプションの詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

例えば，フライト予約アプリケーションの［ログイン］ウィンドウで［OK］ボタンをクリックすると，結果のステートメントは次のようになります。

```
obj_check_gui ("OK", "list1.ckl", "gui1", 1);
```

win_check_gui 関数の構文は次のとおりです。

win_check_gui (window, checklist, expected results file, time);

window は，GUI ウィンドウの論理名です。*checklist* は，検査するオブジェクトとプロパティを定義するチェックリストの名前です。*expected results file* は，プロパティの期待値を格納するファイルの名前です。*time* は，前回の入力イベント発生時からプロパティの現在値のキャプチャまでの間の最大遅延間隔を秒単位で示したものです。この間隔は，テスト実行中に **timeout_msec** テスト・オプションの値に加算されます。**timeout_msec** テスト・オプションの詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

例えば , サンプルのフライト予約アプリケーションの［ログイン］ウィンドウのタイトル・バーをクリックすると，結果のステートメントは次のようになります。

```
win_check_gui (" ログイン ", "list1.ckl", "gui1", 1);
```

WinRunner がテスト内の最初のチェックリストを *list1.ckl*，最初の期待結果ファイルを *gui1* と命名します。**obj_check_gui** と **win_check_gui** 関数の詳細については，「**TSL リファレンス**」を参照してください。

GUI チェックポイントでの既存の GUI チェックリストの使用

既存の GUI チェックリストを使用して，GUI チェックポイントを作成できます。この方法は，GUI チェックリストを使用して，現在のテストまたは別のテストに新しい GUI チェックポイントを作成したい場合に便利です。例えば，テスト中，数回にわたって特定のオブジェクトの同じプロパティを検査したいとします。これらのオブジェクトのプロパティは，検査する時間によって，期待値が異なる可能性があります。

新しい GUI チェックポイントを作成するたびに新しい GUI チェックリストを作成することができますが，GUI チェックリストはできるだけ多くのチェックポイントで「再利用」することを奨めます。複数の GUI チェックポイントで単独の GUI チェックリストを使用すれば，テストで使われる GUI チェックポイントの保守に要する時間と労力を減らせるため，テスト工程を簡略化できます。

WinRunner がアプリケーション内で検査するオブジェクトを特定できるように，テストを実行する前に適切な GUI マップ・ファイルをロードしておく必要があります。
GUI マップ・ファイルのロードに関する情報については，59 ページ「GUI マップ・ファイルのロード」を参照してください。

注：複数のテストで 1 つのチェックリストを使用できるようにしたい場合は，このチェックリストを共有フォルダに保存しなければなりません。GUI チェックリストの共有フォルダへの保存については，143 ページ「GUI チェックリストの共有フォルダへの保存」を参照してください。

GUI チェックポイントで既存の GUI チェックリストを使用するには，次の手順を実行します。

1 ［**挿入**］＞［**GUI チェックポイント**］＞［**複数のオブジェクト**］を選択するか，ユーザ定義ツールバーの［複数オブジェクトの GUI チェックポイント］ボタンをクリックします。

［GUI チェックポイント作成］ダイアログ・ボックスが開きます。

2 ［**開く**］をクリックします。［チェックリストを開く］ダイアログ・ボックスが開きます。

3 ［**共有**］をクリックして，チェックリストが共有フォルダにあるかどうか確かめます。

4 チェックリストを選択し，［**OK**］をクリックします。

［チェックリストを開く］ダイアログ・ボックスが閉じ，選択されたリストが［GUI チェックポイント作成］ダイアログ・ボックスに表示されます。

5 チェックリストに表示されるオブジェクトを含む，テスト対象アプリケーションでウィンドウが開いていなければウィンドウを開きます。

6 ［**OK**］をクリックします。

WinRunner はプロパティの現在値をキャプチャし，テスト・スクリプトに **win_check_gui** ステートメントが挿入されます。

GUI チェックリストの変更

GUI チェックポイント用に作成したチェックリストには変更を加えることができます。チェックリストには，検査が必要なオブジェクトとプロパティだけが含まれています。これらのプロパティの値に対する期待結果は含まれません。

以下のことが可能です。

- チェックリストを共有フォルダに保存して他のユーザにも利用できるようにする
- チェックリストを編集する

注： GUI チェックリストだけでなく，GUI チェックポイントの期待結果も変更できます。詳細については，175 ページ「GUI チェックポイントの期待結果の変更」を参照してください。

GUI チェックリストの共有フォルダへの保存

標準設定では，GUI チェックポイントのチェックリストは，現在のテストのフォルダに格納されます。チェックリストにより幅広くアクセスできるよう，チェックリストが共有フォルダに格納されるよう指定すれば，これを複数のテストで使用できます。

WinRunner で共有チェックリストが格納される標準のフォルダは「**WinRunner のインストール・フォルダ の chklist**」です。［一般オプション］ダイアログ・ボックスの［フォルダ］カテゴリにある［**共有チェックリスト**］ボックスを使って，別のフォルダを選択できます。詳細については，第 22 章「グローバル・テスト・オプションの設定」を参照してください。

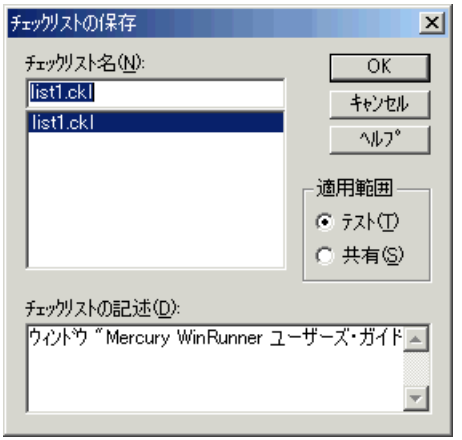
共有フォルダに GUI チェックリストを保存するには，次の手順を実行します。

1 ［**挿入**］＞［**GUI チェックリスト編集**］を選択します。［チェックリストを開く］ダイアログ・ボックスが開きます。GUI チェックリストの拡張子は .ckl，データベース・チェックリストの拡張子は .cdl です。

データベース・チェックリストの詳細については，304 ページ「標準のデータベース・チェックポイントの変更」を参照してください。

2 GUI チェックリストを選択し，［**OK**］をクリックします。［チェックリストを開く］ダイアログ・ボックスが閉じます。［GUI チェックリスト編集］ダイアログ・ボックスに，選択したチェックリストが表示されます。

3 ［**名前を付けて保存**］をクリックして，チェックリストを保存します。［チェックリストの保存］ダイアログ・ボックスが開きます。



4 ［**適用範囲**］で，［**共有**］をクリックします。共有チェックリストの名前を入力します。［OK］をクリックしてチェックリストを保存し，ダイアログ・ボックスを閉じます。

5 ［**OK**］をクリックして，［GUI チェックリスト編集］ダイアログ・ボックスを閉じます。

GUI チェックリストの編集

既存の GUI チェックリストは編集できます。GUI チェックリストには，検査するオブジェクトとプロパティだけが含まれます。これらのプロパティの値に対する期待結果は含まれません。

すでにチェックリストが定義されているウィンドウにチェックポイントを追加する場合に，GUI チェックリストを編集したいことがあります。

GUI チェックリストを編集する場合，以下のことが可能です。

- ウィンドウ内の検査するオブジェクトの変更
- オブジェクト内の検査するプロパティの変更
- 既存のプロパティ検査の引数の変更
- 新しいプロパティ検査への引数の指定

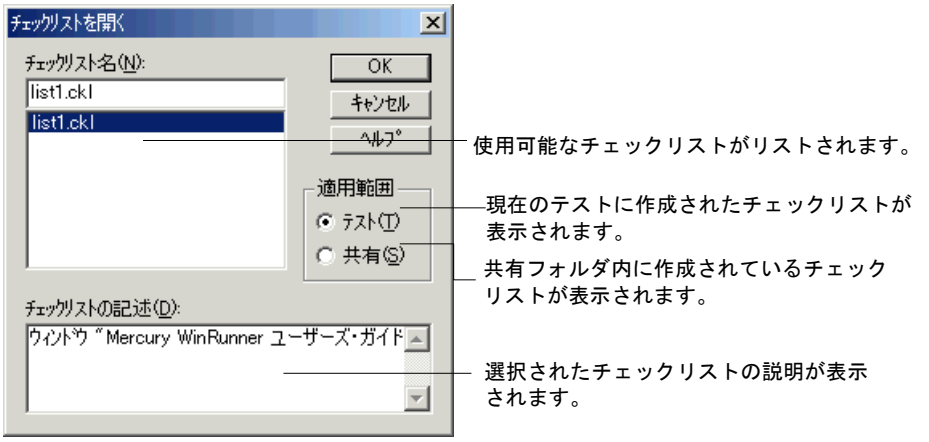
作業を開始する前に，チェックリスト内のオブジェクトは GUI マップにロードされていなければなりません。GUI マップのロードについては，59 ページ「GUI マップ・ファイルのロード」を参照してください。

既存の GUI チェックリストを編集するには，次の手順を実行します。

1 ［**挿入**］＞［**GUI チェックリスト編集**］を選択します。［チェックリストを開く］ダイアログ・ボックスが開きます。

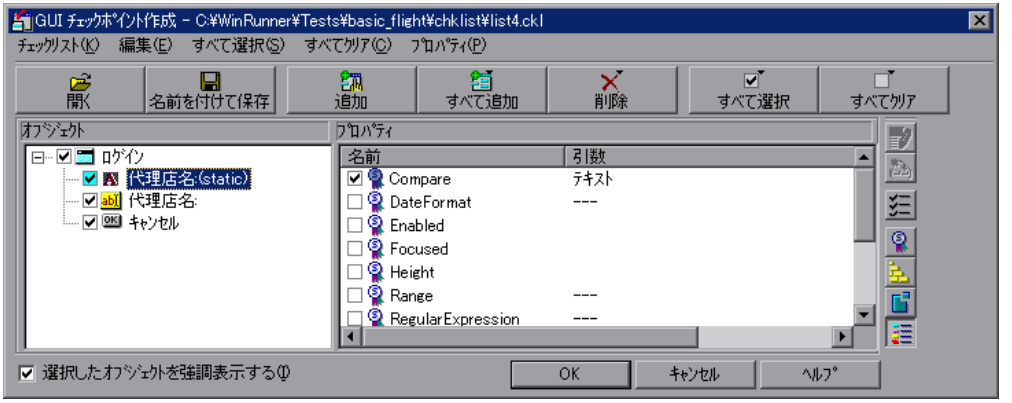
2 現在のテストのチェックリストの一覧が表示されます。共有フォルダ内のチェックリストを見たい場合は，［**共有**］をクリックします。

共有 GUI チェックリストの詳細については，143 ページ「GUI チェックリストの共有フォルダへの保存」を参照してください。



3 GUI チェックリストを選択します。

4 [**OK**] をクリックします。[チェックリストを開く] ダイアログ・ボックスが閉じます。[GUI チェックリスト編集] ダイアログ・ボックスが開き，選択されたチェックリストが表示されます。



5 特定のオブジェクトに対して検査するプロパティのリストを見るには，[オブジェクト] 表示枠でそのオブジェクトの名前をクリックします。[**プロパティ**] 表示枠には，選択されたオブジェクトのすべてのプロパティが表示されます。オブジェクトのプロパティの表示オプションを変更するには，プロパティ表示の切り替え用ボタンを使用します。詳細については，158 ページ「[GUI チェックリスト編集] ダイアログ・ボックス」を参照してください。

➤ オブジェクトの付加的なプロパティを検査するには，[**オブジェクト**] 表示枠でオブジェクトを選択します。[**プロパティ**] 表示枠で，検査するプロパティを選択します。

➤ チェックリストからオブジェクトを削除するには，[**オブジェクト**] 表示枠でオブジェクトを選択します。[**削除**] ボタンをクリックしてから，[**オブジェクト**] オプションを選択します。

➤ チェックリストにオブジェクトを追加するには，テスト対象アプリケーションで対象となるウィンドウを開いておく必要があります。[**追加**] ボタンをクリックします。マウス・ポインタが指差し型に変わり，ヘルプ・ウィンドウが表示されます。

チェックリストに含めたい各オブジェクトをクリックします。オブジェクトの選択を止めるときにはマウスを右クリックします。[GUI チェックリスト編集] ダイアログ・ボックスが再び開きます。

［**プロパティ**］表示枠で，検査したいプロパティを選択するか，標準の検査を承認します。

注：異なるウィンドウからのオブジェクトを 1 つのチェックポイントに挿入することはできません。

➤ 1 つのウィンドウ内のすべてのオブジェクトまたはメニューをチェックリストに追加するには，まずテスト対象アプリケーションのウィンドウがアクティブになっていることを確認します。［**すべて追加**］ボタンをクリックして，［**オブジェクト**］か［**メニュー**］を選択します。

注：編集されているチェックリストが **obj_check_gui** ステートメントに含まれている場合，このステートメントは単数オブジェクト専用であるため，このチェックリストに付加的なオブジェクトを追加することはできません。

➤ 検査を追加して引数を指定するには，まず引数を指定したいプロパティを選択します。次に［**引数を指定**］ボタンをクリックするか，［**引数**］カラムをダブルクリックします。［引数］カラムに省略記号（3 つの点）が表示されている場合は，このプロパティの検査に引数を指定しなければなりません（標準の引数が指定されている場合は，引数を指定する必要はありません）。標準のオブジェクトを検査する場合は，編集オブジェクトと静的テキスト・オブジェクトの特定のプロパティにのみ引数を指定します。また，非標準オブジェクトの特定のプロパティの検査にも引数を指定します。詳細については，167 ページ「プロパティ検査への引数の指定」を参照してください。詳細については，167 ページ「プロパティ検査への引数の指定」を参照してください。

6 次のいずれかの方法でチェックリストを保存します。

➤ チェックリストを既存の名前で保存するには，［**OK**］をクリックして，［GUI チェックリスト編集］ダイアログ・ボックスを閉じます。WinRunner は，既存のチェックリストを上書きするかどうかたずねるメッセージを表示します。［**OK**］をクリックします。

➤ チェックリストを別の名前で保存するには，［**名前を付けて保存**］ボタンをクリックします。［チェックリストの保存］ダイアログ・ボックスが開きます。新しい名前か，標準の名前を入力します。［**OK**］をクリックします。［OK］をクリックして［GUI チェックリスト編集］ダイアログ・ボックスを閉じると，チェックリストは自動的に標準の名前で保存されるので，［名前を付けて保存］ボタンをクリックする必要はありません。

新しい GUI チェックポイントのステートメントは，テスト・スクリプトに挿入されません。

［GUI チェックリスト編集］ダイアログ・ボックスの詳細については，148 ページ「GUI チェックポイント・ダイアログ・ボックスについて」を参照してください。

注 :「検証」実行モードでテストを実行する前に，チェックリストで行った変更に合わせて期待結果を更新しなければなりません。期待結果を更新するには，テストを「更新」実行モードで実行します。「更新」実行モードでのテストの実行については，422 ページ「WinRunner のテスト実行モード」を参照してください。

GUI チェックポイント・ダイアログ・ボックスについて

GUI チェックポイントを作成して GUI オブジェクトを検査する際，検査するオブジェクトとプロパティの指定，新しいチェックリストの作成，既存のチェックリストの変更などが行えます。［GUI チェック］ダイアログ・ボックス，［GUI チェックポイント作成］，［GUI チェックリスト編集］ダイアログ・ボックスという 3 つのダイアログ・ボックスを使用して，GUI チェックポイントを保持できます。

標準設定では，各 GUI チェックポイントのダイアログ・ボックスの一番上に表示されるツールバーに，テキスト付きの大きなボタンが表示されますが，テキストなしの小さいボタンを表示することもできます。下に大きいボタンと小さいボタンを示します。

大きい［追加］ボタン

小さい［追加］ボタン

GUI チェックポイントのダイアログ・ボックスを小さいボタンで表示するには，次の手順を実行します。

1 ダイアログ・ボックスの左上隅をクリックします。

2 ［大きいボタン］オプションのチェックを外します。

GUI チェックポイントのダイアログ・ボックスに表示されるメッセージ

GUI チェックポイントのダイアログ・ボックスには，以下のメッセージが表示されます。

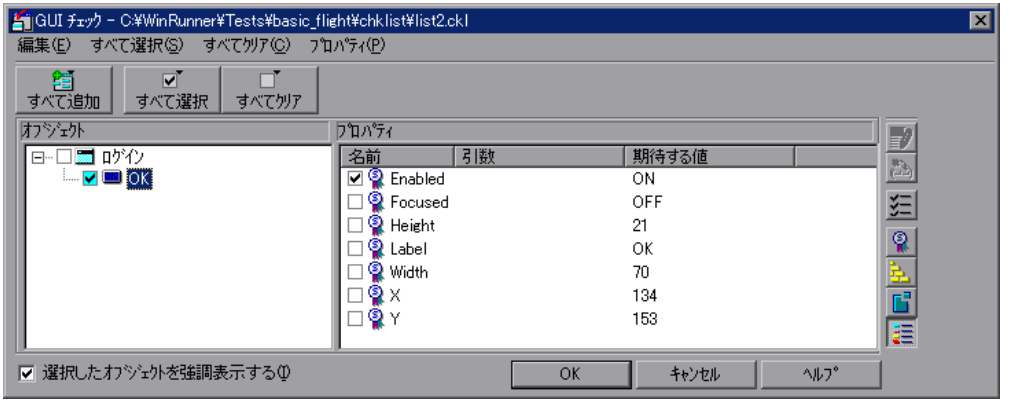
メッセージ	意味	ダイアログ・ボックス	場所
複雑な値	選択されたプロパティ検査の期待値または実際の値が複雑すぎて，カラムに表示できません。このメッセージは，テーブル内容の検査時によく現れます。	［GUI チェック］，［GUI チェックポイント作成］，［GUI 検証結果］*（下の注参照）	［プロパティ］表示枠，［期待する値］カラム，［実際の値］カラム
該当なし	選択されたプロパティ検査の期待値がキャプチャされませんでした。この検査が期待値を持つためには引数を指定する必要があるか，この検査の期待値は検査をチェックポイントに追加したとき初めてキャプチャされるからです。	［GUI チェック］，［GUI チェックポイント作成］，［GUI 検証結果］*（下の注参照）	［プロパティ］表示枠，［期待する値］カラム

メッセージ	意味	ダイアログ・ボックス	場所
次を取得できません	選択されたプロパティの期待値または実際の値をキャプチャできませんでした。	［GUI チェック］，［GUI チェックポイント作成］，［GUI 検証結果］*（下の注参照）	［プロパティ］表示枠，［期待する値］カラム，［実際の値］カラム
この オブジェクトのプロパティを検出できません	指定されたオブジェクトにプロパティがありませんでした。	［GUI チェック］，［GUI チェックポイント作成］，［GUI リスト編集］	［プロパティ］表示枠
このオブジェクトのプロパティをキャプチャできませんでした	このチェックポイントが作成されたときに，このオブジェクトにプロパティ検査が指定されませんでした。	［GUI 検証結果］*（下の注参照）	［プロパティ］表示枠

注 : ［GUI 検証結果］ダイアログ・ボックスについては，175 ページ「GUI チェックポイントの期待結果の変更」か，第 20 章「テスト結果の分析」を参照してください。

[GUI チェック] ダイアログ・ボックス

[GUI チェック] ダイアログ・ボックスを使って，1 つのオブジェクトまたは 1 つのウィンドウに指定した検査を含む GUI チェックポイントを作成できます。このダイアログ・ボックスは，[**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，オブジェクトまたはウィンドウをダブルクリックすると開きます。



[**オブジェクト**] 表示枠には，GUI チェックポイントに含めるウィンドウとオブジェクトの名前が表示されます。[**プロパティ**] 表示枠には，選択されたオブジェクトのすべてのプロパティが表示されます。チェックマークは，その項目が選択されており，チェックポイントに含まれていることを示します。

[**オブジェクト**] 表示枠でオブジェクトを選択すると，[**選択したオブジェクトを強調表示**] オプションを選択してあり，画面上でそのオブジェクトが可視であれば，実際の GUI オブジェクトが強調表示されます。

注：引数を必要とするプロパティ検査に引数が指定されていないと，その検査の [**期待する値**] カラムに **< 該当なし >** が表示されます。検査に指定される引数により期待値が決定されるので，引数を指定しないと期待値は使用できません。

［GUI チェック］ダイアログ・ボックスには以下のオプションがあります。

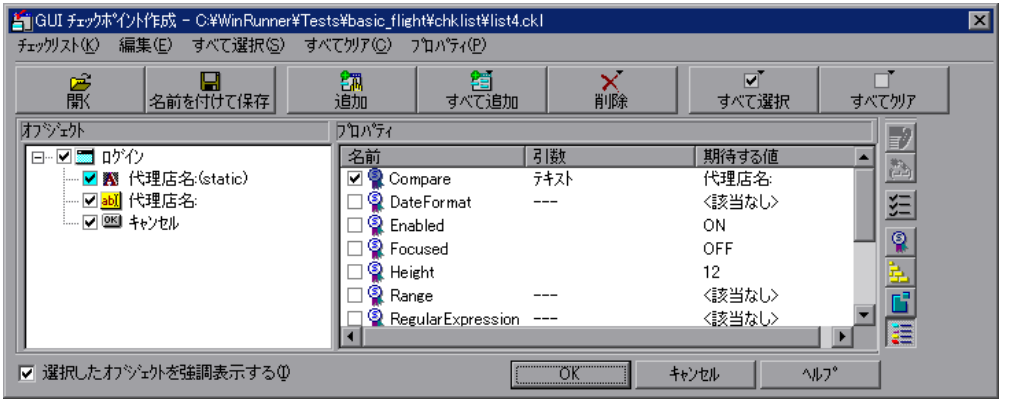
ボタン	説明
	**［すべて追加］**ボタンで，**ウィンドウ内のすべてのオブジェクトまたはメニューをチェックリストに追加します。**
	**［すべて選択］**ボタンで，［GUI チェック］ダイアログ・ボックスのオブジェクト，プロパティ，または指定されたクラスのオブジェクトをすべて選択します。指定されたクラスのすべてのオブジェクトを選択したい場合は，［オブジェクトのクラス］ダイアログ・ボックスが開きます。選択するオブジェクトのクラスを指定します。
	**［すべてクリア］**ボタンで，**［GUI チェック］ダイアログ・ボックスの**オブジェクト，プロパティ，または指定されたクラスのオブジェクトをすべてクリアします。指定されたクラスのすべてのオブジェクトをクリアしたい場合は，［オブジェクトのクラス］ダイアログ・ボックスが開きます。
	［**プロパティ リスト**］ボタンで，**gui_ver_add_class** 関数を使ってカスタマイズされたクラスに対してのみ定義された **ui_function** パラメータを呼び出します。このボタンは，［オブジェクト］表示枠で少なくとも 1 つのオブジェクトが，**gui_ver_add_class** 関数によって **ui_function** パラメータが定義されているクラスに属している場合にのみ表示されます。詳細については，『**WinRunner カスタマイズ・ガイド**』を参照してください。
	**［期待結果値の編集］**ボタンで，選択されたプロパティの期待値を編集できます。詳細については，173 ページ「プロパティの期待値の編集」を参照してください。
	**［引数を指定］**ボタンで，選択されているプロパティの検査に引数を指定できます。詳細については，167 ページ「プロパティ検査への引数の指定」を参照してください。
	**［選択されたプロパティのみ表示］**ボタンで，チェック・ボックスが選択されているプロパティだけを表示します（このボタンで，すべてのプロパティの表示と選択されているプロパティだけの表示を切り替えることができます）。標準設定では，すべてのプロパティが表示されます。
	**［標準プロパティのみ表示］**ボタンで，標準プロパティだけを表示します。

ボタン	説明
	**［標準以外のプロパティのみ表示］**ボタンで，Visual Basic，PowerBuilder，ActiveX コントロールのプロパティなど，非標準のプロパティだけを表示します。
	**［ユーザプロパティのみ表示］**ボタンで，ユーザ定義のプロパティだけを表示します。ユーザ定義のプロパティ検査を作成するには，『**WinRunner カスタマイズ・ガイド**』を参照してください。
	**［すべてのプロパティを表示］**ボタンで，標準プロパティ，非標準プロパティ，ユーザ定義プロパティを含むすべてのプロパティを表示します。

［OK］をクリックしてダイアログ・ボックスを閉じると，WinRunner は変更を保存し，プロパティの現在値をキャプチャして，それをテストの期待結果フォルダに格納します。［WinRunner］ウィンドウが再び表示され，GUI チェックポイントが **obj_check_gui** または **win_check_gui** ステートメントとしてテスト・スクリプトに挿入されます。

[GUI チェックポイント作成] ダイアログ・ボックス

[GUI チェックポイント作成] ダイアログ・ボックスを使って，標準の検査を使ってあるいは検査するプロパティを指定することによって，複数のオブジェクトを対象とする GUI チェックリストを作成することができます。[GUI チェックポイント作成] ダイアログ・ボックスを開くには，[**挿入**] > [**GUI チェックポイント**] > [**複数のオブジェクト**] を選択するか，ユーザ定義ツールバーで [**複数のオブジェクトの GUI チェックポイント**] ボタンをクリックします。



[**オブジェクト**] 表示枠には，ウィンドウの名前と GUI チェックポイントに含まれるオブジェクトが表示されます。[**プロパティ**] 表示枠には，選択されたオブジェクトのすべてのプロパティが表示されます。チェックマークは，項目が選択されており，チェックポイントに含まれることを示します。

[**オブジェクト**] 表示枠でオブジェクトを選択すると，[**選択したオブジェクトを強調表示**] オプションを選択してあり，画面上でそのオブジェクトが可視であれば，実際の GUI オブジェクトが強調表示されます。

注：引数を必要とするプロパティ検査に引数が指定されていないと，その検査の [**期待する値**] カラムに **< 該当なし >** が表示されます。検査に指定される引数により期待値が決定されるので，引数を指定しないと期待値は使用できません。

［GUI チェックポイント作成］ダイアログ・ボックスには，以下のオプションがあります。

ボタン	説明
	**［開く］**ボタンで，既存の GUI チェックリストを開きます。
	**［名前を付けて保存］**ボタンで，開いている GUI チェックリストを異なる名前で保存します。［名前を付けて保存］ボタンをクリックせずに，［OK］をクリックして［GUI チェックポイント作成］ダイアログ・ボックスを閉じると，WinRunner はチェックリストを標準の名前で保存します。［名前を付けて保存］オプションは，チェックリストを「共有チェックリスト」フォルダに保存する場合に特に便利です。
	**［追加］**ボタンで，自分の GUI チェックリストにオブジェクトを追加します。
	**［すべて追加］**ボタンで，ウィンドウ内のすべてのオブジェクトまたはメニューを GUI チェックリストに追加します。
	**［削除］**ボタンで，オブジェクトを 1 つだけ，または GUI チェックリストに現れるオブジェクトすべてを削除します。
	**［すべて選択］**ボタンで，［GUI チェックポイント作成］ダイアログ・ボックスのオブジェクト，プロパティ，または指定されたクラスのオブジェクトをすべて選択します。指定されたクラスのすべてのオブジェクトを選択したい場合は，［ オブジェクトのクラス］ダイアログ・ボックスを開き，選択するオブジェクトのクラスを指定します。
	**［すべてクリア］**ボタンで，［GUI チェックポイント作成］ダイアログ・ボックスのオブジェクト，プロパティ，または指定されたクラスのオブジェクトをすべてクリアします。指定されたクラスのすべてのオブジェクトをクリアしたい場合は，［オブジェクトのクラス］ダイアログ・ボックスを開き，クリアするオブジェクトのクラスを指定します。

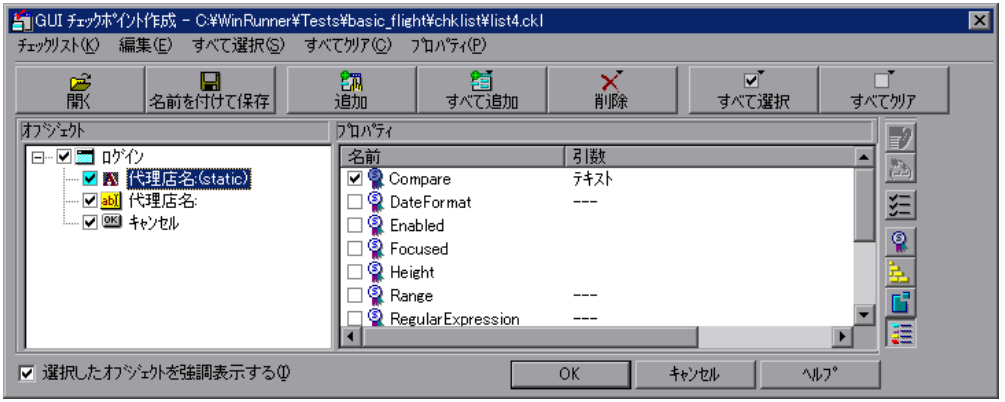
ボタン	説明
	［**プロパティ リスト**］ボタンで，**gui_ver_add_class** 関数を使ってカスタマイズされたクラスに対してのみ定義された **ui_function** parameter を呼び出します。このボタンは，［オブジェクト］表示枠で少なくとも 1 つのオブジェクトが，**gui_ver_add_class** 関数によって **ui_function** parameter が定義されているクラスに属している場合にのみ表示されます。詳細については，『**WinRunner カスタマイズ・ガイド**』を参照してください。
	［**期待結果値を編集**］ボタンで，選択されたプロパティの期待値を編集できます。詳細については，173 ページ「プロパティの期待値の編集」を参照してください。
	［**引数を指定**］ボタンで，選択されているプロパティの検査に引数を指定できます。詳細については，167 ページ「プロパティ検査への引数の指定」を参照してください。
	［**選択されたプロパティのみ表示**］ボタンで，チェック・ボックスが選択されているプロパティだけを表示します（このボタンで，すべてのプロパティの表示と選択されているプロパティだけの表示を切り替えることができます）。標準設定では，すべてのプロパティが表示されます。
	［**標準プロパティのみ表示**］ボタンで，標準プロパティだけを表示します。
	［**標準以外のプロパティのみ表示**］ボタンで，Visual Basic，PowerBuilder，ActiveX コントロールのプロパティなど，非標準のプロパティだけを表示します。
	［**ユーザプロパティのみ表示**］ボタンで，ユーザ定義のプロパティだけを表示します。ユーザ定義のプロパティ検査を作成するには，『**WinRunner カスタマイズ・ガイド**』を参照してください。
	［**すべてのプロパティを表示**］ボタンで，標準プロパティ，非標準プロパティ，ユーザ定義プロパティを含むすべてのプロパティを表示します。

［OK］をクリックしてダイアログ・ボックスを閉じると，WinRunner は変更を保存し，プロパティの現在値をキャプチャして，それをテストの期待結果フォルダに格納します。［WinRunner］ウィンドウが再び表示され，GUI チェックポイントが **win_check_gui** ステートメントとしてテスト・スクリプトに挿入されます。

［GUI チェックリスト編集］ダイアログ・ボックス

［GUI チェックリスト編集］ダイアログ・ボックスを使用して，チェックリストを更新できます。チェックリストにはオブジェクトとプロパティのリストが含まれます。チェックリストは，これらのプロパティの現在値をキャプチャしません。したがって，このダイアログ・ボックスで，オブジェクトのプロパティの期待値は編集できません。

［GUI チェックリスト編集］ダイアログ・ボックスを開くには，［**挿入**］＞［**GUI チェックリスト編集**］を選択します。



［**オブジェクト**］表示枠には，ウィンドウの名前と GUI チェックリストに含まれるオブジェクトが表示されます。［**プロパティ**］表示枠には，選択されたオブジェクトのすべてのプロパティが表示されます。チェックマークは，項目が選択されており，このチェックリストを使用するチェックポイントで検査されることを示します。

［オブジェクト］表示枠でオブジェクトを選択すると，［**選択したオブジェクトを強調表示**］オプションを選択してあり，画面上でそのオブジェクトが可視であれば，実際の GUI オブジェクトが強調表示されます。

［GUI チェックリスト編集］ダイアログ・ボックスには，以下のオプションがあります。

ボタン	説明
	［**開く**］ボタンで，既存の GUI チェックリストを開きます。
	［**名前を付けて保存**］ボタンで，GUI チェックリストを別の場所に保存します。［名前を付けて保存］ボタンをクリックせずに，［OK］をクリックして［GUI チェックリスト編集］ダイアログ・ボックスを閉じると，WinRunner はチェックリストを標準の名前で保存します。［名前を付けて保存］オプションは，チェックリストを「共有チェックリスト」フォルダに保存する場合に特に便利です。
	［**追加**］ボタンで，自分の GUI チェックリストにオブジェクトを追加します。
	［**すべて追加**］ボタンで，ウィンドウ内のすべてのオブジェクトまたはメニューを GUI チェックリストに追加します。
	［**削除**］ボタンで，オブジェクトを 1 つだけ，または GUI チェックリストに現れるオブジェクトすべてを削除します。
	［**すべて選択**］ボタンで，［GUI チェックリスト編集］ダイアログ・ボックスのオブジェクト，プロパティ，または指定されたクラスのオブジェクトをすべて選択します。指定されたクラスのすべてのオブジェクトを選択したい場合は，［オブジェクトのクラス］ダイアログ・ボックスを開き，選択するオブジェクトのクラスを指定します
	［**すべてクリア**］ボタンで，［GUI チェックリスト編集］ダイアログ・ボックスのオブジェクト，プロパティ，または指定されたクラスのオブジェクトをすべてクリアします。指定されたクラスのすべてのオブジェクトをクリアしたい場合は，［オブジェクトのクラス］ダイアログ・ボックスを開き，クリアするオブジェクトのクラスを指定します。

ボタン	説明
	［**プロパティ リスト**］ボタンで，**gui_ver_add_class** 関数を使ってカスタマイズされたクラスに対してのみ定義された **ui_function** パラメータ を呼び出します。このボタンは，［オブジェクト］表示枠で少なくとも 1 つのオブジェクトが，**gui_ver_add_class** 関数によって **ui_function** パラメータが定義されているクラスに属している場合にのみ表示されます。詳細については，『**WinRunner カスタマイズ・ガイド**』を参照してください。
	［**引数を指定**］ボタンで，選択されているプロパティの検査に引数を指定できます。詳細については，167 ページ「プロパティ検査への引数の指定」を参照してください。
	［**選択されたプロパティのみ表示**］ボタンで，チェック・ボックスが選択されているプロパティだけを表示します（このボタンで，すべてのプロパティの表示と選択されているプロパティだけの表示を切り替えることができます)。標準設定では，すべてのプロパティが表示されます。
	［**標準プロパティのみ表示**］ボタンで，標準プロパティだけを表示します。
	［**標準以外のプロパティのみ表示**］ボタンで，Visual Basic, PowerBuilder，ActiveX コントロールのプロパティなど，非標準のプロパティだけを表示します。
	［**ユーザプロパティのみ表示**］ボタンで，ユーザ定義のプロパティだけを表示します。ユーザ定義のプロパティ検査を作成するには，『**WinRunner カスタマイズ・ガイド**』を参照してください。
	［**すべてのプロパティを表示**］ボタンで，標準プロパティ，非標準プロパティ，ユーザ定義プロパティを含むすべてのプロパティを表示します。

［OK］をクリックしてダイアログ・ボックスを閉じると，WinRunner はチェックリストを上書きするかどうかたずねるメッセージを表示します。チェックリストを上書きしても，編集したチェックリストを使ったチェックポイントでキャプチャされた期待結果は変更されないまま残ります。

新しい GUI チェックポイント・ステートメントはテスト・スクリプトには挿入されません。

注 :「検証」実行モードでテストを実行する前に，チェックリストで行った変更に合わせて期待結果を更新しなければなりません。期待結果を更新するには，テストを「更新」実行モードで実行します。「更新」実行モードでのテストの実行については，422 ページ「WinRunner のテスト実行モード」を参照してください。

プロパティ検査と標準の検査

GUI チェックポイントを作成すると，アプリケーションの GUI オブジェクトに対して行う検査の種類を決定できます。各オブジェクト・クラスに対して，WinRunner が推奨する標準の検査が用意されています。例えば，プッシュ・ボタンを選択した場合，標準の検査はプッシュ・ボタンが有効になっているかどうかを調べます。この方法の代わりに，ダイアログ・ボックスで，検査するオブジェクトのプロパティを指定することもできます。例えば，プッシュ・ボタンの幅，高さ，ラベル，ウィンドウ内での位置（x，y 座標）を検査することもできます。

「**標準の検査**」を使用するには，［**挿入**］＞［**GUI チェックポイント**］コマンドを選択します。アプリケーションのウィンドウまたはオブジェクトをクリックします。WinRunner は，自動的にウィンドウまたはオブジェクトの情報をキャプチャし，テスト・スクリプトに GUI チェックポイントを挿入します。

あるオブジェクトに対して検査するプロパティを指定するには，［**挿入**］＞［**GUI チェックポイント**］コマンドを選択します。そして，対象となるウィンドウまたはオブジェクトをダブルクリックします。［GUI チェック］ダイアログ・ボックスで WinRunner に検査させたいプロパティを選択します。［OK］をクリックして検査を保存し，ダイアログ・ボックスを閉じます。WinRunner は，GUI オブジェクトに関する情報をキャプチャし，テスト・スクリプトに GUI チェックポイントを挿入します。

以下に，様々なオブジェクト・クラスに利用できる検査の種類を示します。

calendar クラス

calendar クラスのオブジェクトについては以下のプロパティを検査できます。

Enabled：カレンダが選択できるかどうかを検査します。

Focused：キーボード入力がカレンダに送られるかどうかを検査します。

Height：カレンダの高さをピクセル単位で検査します。

Selection：カレンダで選択されている日付（標準の検査）。

Width：カレンダの幅をピクセル単位で検査します。

X：ウィンドウ原点からのカレンダの左上隅の x 座標を検査します。

Y：ウィンドウ原点からのカレンダの左上隅の y 座標を検査します。

check_button クラスと radio_button クラス

チェック・ボックス（check_button クラスのオブジェクト）またはラジオ・ボタンについては以下のプロパティを検査できます。

Enabled：ボタンが選択できるかどうかを検査します。

Focused：キーボード入力がボタンに送られるかどうかを検査します。

Height：ボタンの高さをピクセル単位で検査します。

Label：ボタンのラベルを検査します。

State：ボタンの状態（オンかオフ）を検査します（標準の検査）。

Width：ボタンの幅をピクセル単位で検査します。

X：ウィンドウ原点からのボタンの左上隅の x 座標を検査します。

Y：ウィンドウ原点からのボタンの左上隅の y 座標を検査します。

edit クラスと static_text クラス

edit クラスと static_text クラスのオブジェクトについては以下のプロパティを検査できます。

以下の 5 つのプロパティ（Compare，DateFormat，Range，RegularExpression，TimeFormat）の検査を行う場合は，引数を指定しなければなりません。プロパティ検査への引数の指定については，167 ページ「プロパティ検査への引数の指定」を参照してください。

Compare：オブジェクトの内容を検査します（標準の検査）。この検査には引数を指定します。以下の引数を指定できます。

- 内容をテキストとして，大文字と小文字を区別しながら検査します（標準の設定）。
- 内容をテキストとして，大文字と小文字を区別せずに検査します。
- 内容を数値として検査します。

DateFormat：オブジェクトの内容が指定された日付書式であるかどうか検査します。この検査には引数（日付書式）を指定しなければなりません。
WinRunner では，様々な日付書式がサポートされています。利用できる日付書式のリストは 169 ページ「日付書式」を参照してください。

Enabled：オブジェクトを選択できるかどうかを検査します。

Focused：キーボード入力がこのオブジェクトに送られるかどうかを検査します。

Height：オブジェクトの高さをピクセル単位で検査します。

Range：オブジェクトの内容が指定された範囲にあるかどうか検査します。この検査には引数を指定（範囲の上限と下限）しなければなりません。

RegularExpression：オブジェクト内の文字列が正規表現の条件に適合するかどうか検査します。この検査には引数（文字列）を指定しなければなりません。正規表現の先頭に感嘆符（!）を指定する必要はありません。詳細については，第 27 章「正規表現の使い方」を参照してください。

TimeFormat：オブジェクトの内容が指定された時間書式であるかどうか検査します。この検査には引数（時間書式）を指定しなければなりません。
WinRunner は以下の時間書式をサポートしています。以下に例も示します。

hh.mm.ss	10.20.56
hh:mm:ss	10:20:56
hh:mm:ss ZZ	10:20:56 AM

Width：オブジェクトの幅をピクセル単位で検査します。

X：ウィンドウ原点からのオブジェクトの左上隅の x 座標を検査します。

Y：ウィンドウ原点からのオブジェクトの左上隅の y 座標を検査します。

list クラス

list オブジェクトについては以下のプロパティを検査できます。

Content：全リストの内容を検査します。

Enabled：リスト内の項目を選択できるかどうか検査します。

Focused：キーボード入力がこのリストに送られるかどうか検査します。

Height：リストの高さをピクセル単位で検査します。

ItemsCount：リスト内の項目の数を検査します。

Selection：現在のリスト選択（標準の検査）を検査します。

Width：リストの幅をピクセル単位で検査します。

X：ウィンドウ原点からのリストの左上隅の x 座標を検査します。

Y：ウィンドウ原点からのリストの左上隅の y 座標を検査します。

Menu_item クラス

メニューは，メニューをクリックするだけでは直接アクセスできません GUI チェックポイントの対象にメニューを含めるには，ウィンドウのタイトル・バーかメニュー・バーをクリックします。［**すべて追加**］ダイアログ・ボックスが開きます。［**メニュー**］オプションを選択します。ウィンドウのすべてのメニューがチェックリストに追加されます。それぞれのメニューは，個別にリストされます。

メニュー項目については以下のプロパティを検査できます。

HasSubMenu：メニュー項目にサブメニューがあるかどうか検査します。

ItemEnabled：メニューが有効かどうか検査します（標準の検査）。

ItemPosition：メニュー内の各項目の場所を検査します。

SubMenusCount：サブメニューの項目数を数えます。

object クラス

標準のオブジェクト・クラスにマップされていないオブジェクトについては，以下のプロパティを検査できます。

Enabled：オブジェクトが選択されるかどうかを検査します。

Focused：キーボード入力がこのオブジェクトに送られるかどうかを検査します。

Height：オブジェクトの高さをピクセル単位で検査します（標準の検査）。

Width：オブジェクトの幅をピクセル単位で検査します（標準の検査）。

X：ウィンドウ原点からの GUI オブジェクトの左上隅の x 座標を検査します（標準の検査）。

Y：ウィンドウ原点からの GUI オブジェクトの左上隅の y 座標を検査します（標準の検査）。

push_button クラス

プッシュボタンについては以下のプロパティを検査できます。

Enabled：ボタンが選択できるかどうか検査します（標準の検査）。

Focused：キーボード入力がこのボタンに送られるかどうかを検査します。

Height：ボタンの高さをピクセル単位で検査します。

Label：ボタンのラベルを検査します。

Width：ボタンの幅をピクセル単位で検査します。

X：ウィンドウ原点からのボタンの左上隅の x 座標を検査します。

Y：ウィンドウ原点からのボタンの左上隅の y 座標を検査します。

Scroll クラス

スクロールバーについては以下のプロパティを検査できます。

Enabled：スクロールバーが選択できるかどうか検査します。

Focused：キーボード入力がこのスクロールバーに送られるかどうか検査します。

Height：スクロールバーの高さをピクセル単位で検査します。

Position：スクロールバーのスクロールつまみの現在の位置を検査します（標準の検査）。

Width：スクロールバーの幅をピクセル単位で検査します。

X：ウィンドウ原点からのスクロールバーの左上隅の x 座標を検査します。

Y：ウィンドウ原点からのスクロールバーの左上隅の y 座標を検査します。

window クラス

ウィンドウについては，以下のプロパティを検査できます。

CountObjects：ウィンドウ内の GUI オブジェクトの数を数えます（標準の検査）。

Enabled：ウィンドウが選択できるかどうか検査します。

Focused：キーボード入力がこのウィンドウに送られるかどうか検査します。

Height：ウィンドウの高さをピクセル単位で検査します。

Label：ウィンドウのラベルを検査します。

Maximizable：ウィンドウを最大化できるかどうか検査します。

Maximized：ウィンドウが最大化されているかどうか検査します。

Minimizable：ウィンドウを最小化できるかどうか検査します。

Minimized：ウィンドウが最小化されているかどうか検査します。

Resizable：ウィンドウの大きさを変更できるかどうか検査します。

SystemMenu：ウィンドウにシステム・メニューがあるかどうか検査します。

Width：ウィンドウの幅をピクセル単位で検査します。

X：ウィンドウの左上隅の x 座標を検査します。

Y：ウィンドウの左上隅の y 座標を検査します。

プロパティ検査への引数の指定

オブジェクトを対象に様々なプロパティ検査が行えます。edit クラスや static_text クラスのオブジェクトを対象に以下のプロパティ検査を実行したい場合は，これらの検査に引数を指定しなければなりません。

- Compare
- DateFormat
- Range
- RegularExpression
- TimeFormat

edit クラスまたは static_text クラスのオブジェクトのプロパティ検査に引数を指定するには，次の手順を実行します。

1 引数を指定したいプロパティを持つオブジェクトが含まれる GUI チェックポイント・ダイアログ・ボックスの 1 つが開いていることを確かめます。開いていなければ，［**挿入**］＞［**GUI チェックポイント**］＞［**複数のオブジェクト**］か，［**挿入**］＞［**GUI チェックリスト編集**］のいずれかを選択して，該当するダイアログ・ボックスを開きます。

2 ダイアログ・ボックスの［**オブジェクト**］表示枠で，検査するオブジェクトを選択します。

3 ダイアログ・ボックスの［**プロパティ**］表示枠で，希望のプロパティ検査を選択します。

4 次のいずれかを行います。

- ［**引数を指定**］ボタンをクリックします。
- 標準の引数（Compare 検査の場合）か，該当する［**引数**］カラムの省略記号（その他の検査の場合）をダブルクリックします。
- マウスを右クリックして，ポップアップメニューから［**引数を指定**］を選択します。

選択されたプロパティ検査のダイアログ・ボックスを開きます。

注：チェック・ボックス（引数を指定する必要のないプロパティ検査のチェック・ボックスを除く）を選択すると，選択されたプロパティ検査のダイアログ・ボックスが自動的に開きます。

---

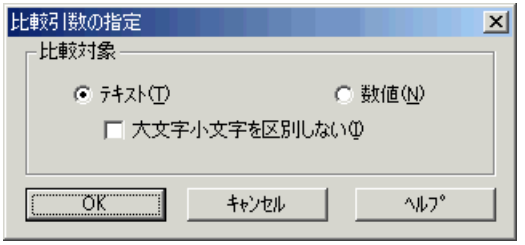
5 開いたダイアログ・ボックスで引数を指定します。例えば，Date Format 検査には，日付書式を指定します。特定のプロパティ検査への引数の指定については，以下を参照してください。

6 ［OK］をクリックして，引数指定用のダイアログ・ボックスを閉じます。

7 終わったら，［OK］をクリックして，GUI チェックポイント・ダイアログ・ボックスを閉じます。

Compare プロパティ検査

edit クラスまたは static_text クラスのオブジェクトの内容を検査します（標準の検査）。［**比較引数の指定**］ダイアログ・ボックスを開きます。



➤ 内容をテキストとして検査する場合（標準設定）は，［**テキスト**］をクリックします。

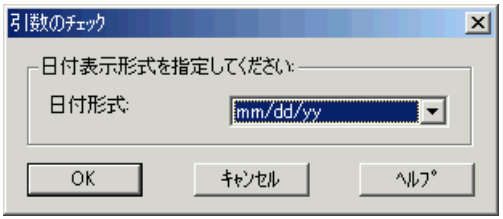
➤ テキストの検査時に大文字と小文字を無視する場合は，［**大文字小文字を区別しない**］チェック・ボックスを選択します。

➤ 内容を数値として検査する場合は，［**数値**］をクリックします。

Compare プロパティ検査には，オブジェクトをテキストとして比較する場合，大文字と小文字を区別する引数が標準設定されています。

DateFormat プロパティ検査

edit クラスまたは static_text クラスのオブジェクトの内容が，指定されている日付書式になっているかどうかを検査します。日付書式を指定するには，［引数のチェック］ダイアログ・ボックスのドロップダウン・リストから該当する形式を選択します。



日付書式

WinRunner では，以下の日付書式がサポートされています。以下に各形式の例も示します。

mm/dd/yy	09/24/04
dd/mm/yy	24/09/04
dd/mm/yyyy	24/09/2004
yy/dd/mm	04/24/09
dd.mm.yy	24.09.04
dd.mm.yyyy	24.09.2004
dd-mm-yy	24-09-04
dd-mm-yyyy	24-09-2004
yyyy-mm-dd	2004-09-24
Day, Month dd, yyyy	Friday（または Fri）, September（または Sept）24, 2004
dd Month yyyy	24 September 2004
Day dd Month yyyy	Friday（または Fri）24 September （または Sept）2004

注：日または月が 0 で始まっている（例えば 9 月を 09 と表すなど）場合，0 がなくても形式の検査は成功します。

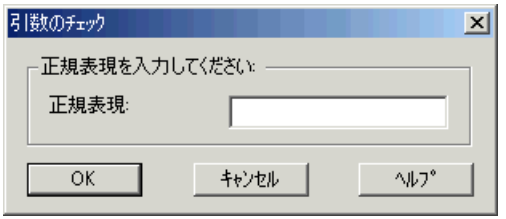
Range プロパティ検査

edit クラスまたは static_text クラスのオブジェクトの内容が指定された範囲内にあるかどうか検査します。［引数のチェック］ダイアログ・ボックスの，上の編集フィールドに下限を，下の編集フィールドに上限を指定します。

注：数に先行するどの通貨記号 も，このチェックで比較する前に取り除かれます。

RegularExpression プロパティ検査

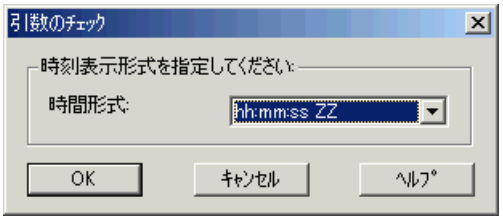
edit クラスまたは static_text クラスのオブジェクト内の文字列が，正規表現の条件に適合しているかどうか検査します。［引数のチェック］ダイアログ・ボックスで，［正規表現］ボックスに文字列を入力します。正規表現の先頭に感嘆符（!）を指定する必要はありません。詳細については，第 27 章「正規表現の使い方」を参照してください。



注：2 つの “¥” 文字（“¥¥”）は，単一の “¥” 文字として判断されます。

TimeFormat プロパティ検査

edit クラスまたは static_text クラスのオブジェクトの内容が指定された時間書式になっているかどうか検査します。時間書式を指定するには，［引数のチェック］ダイアログ・ボックスで，ドロップダウン・リストから該当する時間書式を選択します。



WinRunner では，以下の時間書式がサポートされています。以下に例も示します。

時間書式

hh.mm.ss	10.20.56
hh:mm:ss	10:20:56
hh:mm:ss ZZ	10:20:56 AM

GUI チェックポイント・ダイアログ・ボックスの終了

引数が必要なプロパティ検査を選択しながらも，実際に引数を指定せずに［OK］をクリックしてダイアログ・ボックスを閉じると，引数を指定するよう促すメッセージが表示されます。

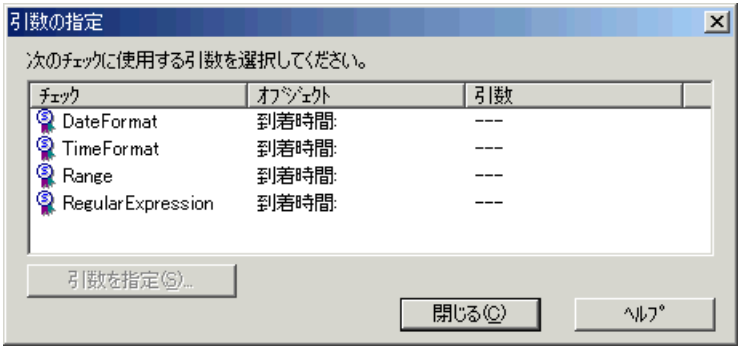
1 つのプロパティ検査に対するの引数の指定

引数が必要なプロパティ検査を選択しながらも，引数を指定せずに［OK］をクリックして GUI チェックポイント・ダイアログ・ボックスを閉じると，そのプロパティ検査の［引数のチェック］ダイアログ・ボックスが開きます。

複数のプロパティ検査に対する引数の指定

引数が必要な複数のプロパティ検査のチェック・ボックスを選択しながらも，引数を指定せずにダイアログ・ボックスを閉じようとすると，［引数を指定］ダイアログ・ボックスが開きます。このダイアログ・ボックスを使って，該当するプロパティ検査に引数を指定できます。

下の例は，ユーザがサンプルのフライト予約アプリケーションの「到着時間：」編集オブジェクトの Date Format，TimeFormat，Range，RegularExpression のプロパティ検査に引数を指定せずに，［OK］をクリックして，［GUI チェックポイント作成］ダイアログ・ボックスを閉じたときの様子を示します。



［**チェック**］カラムに「**プロパティ検査**」が表示されます。［**オブジェクト**］カラムには，オブジェクトの「**論理名**」が表示されます。省略記号が［**引数**］カラムに表示され，プロパティ検査に引数が指定されていないことを示します。

［引数を指定］ダイアログ・ボックスで引数を指定するには，次の手順を実行します。

1 ［**チェック**］カラムでプロパティ検査を選択します。

2 ［**引数を指定**］ボタンをクリックします。あるいは，プロパティ検査をダブルクリックします。

3 そのプロパティ検査に引数を指定するためのダイアログ・ボックスが開きます。

4 上で説明した手順でプロパティ検査の引数を指定します。

5 ［**OK**］をクリックして，引数指定用ダイアログ・ボックスを閉じます。

6 すべてのプロパティ検査の［**引数**］カラムに引数が表示されるまで上記のステップを繰り返します。

7 ダイアログ・ボックス内のすべてのプロパティ検査の引数を指定したら，［**閉じる**］をクリックして，開いている GUI チェックポイント・ダイアログ・ボックスに戻ります。

8 ［**OK**］をクリックして，開いている GUI チェックポイント・ダイアログ・ボックス を閉じます。

プロパティの期待値の編集

GUI チェックポイントを作成する場合，WinRunner は検査するオブジェクトのプロパティの現在値をキャプチャします。こうした現在値は，「**期待結果フォルダ**」に「**期待値**」として保存されます。

テストを実行する場合，WinRunner はこれらのプロパティの値を再びキャプチャします。WinRunner はテスト中にキャプチャされた新しい値をテストの期待結果フォルダに格納されている期待値と比較します。

プロパティの値を GUI チェックポイントでキャプチャした後，テスト・スクリプトを実行するときになってこの値を変更したくなったとします。このような場合は，［GUI チェック］ダイアログ・ボックスか［GUI チェックポイント作成］ダイアログ・ボックスでプロパティの期待値を編集します。

［GUI チェックリスト編集］ダイアログ・ボックスでは，プロパティの期待値を編集できません。［GUI チェックリスト編集］ダイアログ・ボックスを開くと，WinRunner は現在値をキャプチャしません。したがって，このダイアログ・ボックスには編集できる期待値が表示されません。

注：すでに GUI チェックポイントに含まれているプロパティ検査の期待値を編集したい場合は，GUI チェックポイントの期待結果を変更しなければなりません。詳細については，175 ページ「GUI チェックポイントの期待結果の変更」を参照してください。

オブジェクトのプロパティの期待値を編集するには , 次の手順を実行します。

1 期待値を編集するオブジェクトがアプリケーションで開いていることを確認します。

注：オブジェクトが表示されると，WinRunner は［GUI チェック］ダイアログ・ボックスまたは［GUI チェックポイント］ダイアログ・ボックスでオブジェクトのプロパティの期待値を表示できません。

---

2 ［GUI チェック］ダイアログ・ボックスまたは［GUI チェックポイント作成］ダイアログ・ボックスがまだ開いていない場合は，［**挿入**］＞［**GUI チェックポイント**］＞［**複数のオブジェクト**］をクリックし，［**GUI チェックポイント作成**］ダイアログ・ボックスを開きます。次に，［**開く**］をクリックして，期待値を編集するチェックリストを開きます。［**GUI チェック**］ダイアログ・ボックスは，新しい GUI チェックポイントを作成するときにしか開きません。

3 ［**オブジェクト**］表示枠でオブジェクトを選択します。

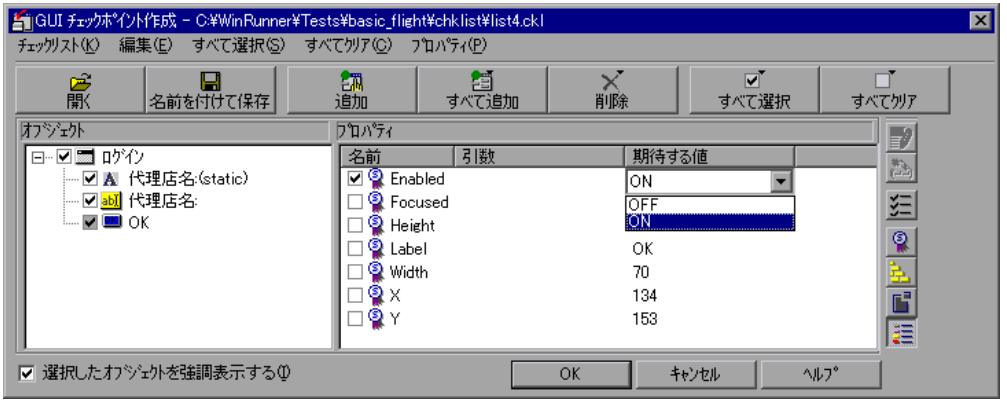
4 ［**プロパティ**］表示枠で，期待値を編集したいプロパティを選択します。

5 以下のいずれかを行います。

- ［期待結果値の編集］ボタンをクリックします。
- 既存の期待値（現在の値）をダブルクリックします。
- マウスを右クリックして，ポップアップ・メニューから［**期待結果値の編集**］を選択します。

プロパティに応じて，編集フィールド，編集ボックス，リスト・ボックス，スピン・ボックス，あるいは新しいダイアログ・ボックスなどが開きます。

例えば，push_button クラス・オブジェクトの **Enabled** プロパティの期待値を編集すると，リスト・ボックスが開きます。



6 プロパティの期待値を必要に応じて編集します。

7 [**OK**] をクリックして，ダイアログ・ボックスを閉じます。

GUI チェックポイントの期待結果の変更

既存の GUI チェックポイントの期待結果は，チェックポイント内のプロパティ検査の期待値を変更することにより変更できます。この変更は，テスト・スクリプトの実行の前後に行えます。

既存の GUI チェックポイントの期待結果を変更するには，次の手順を実行します。

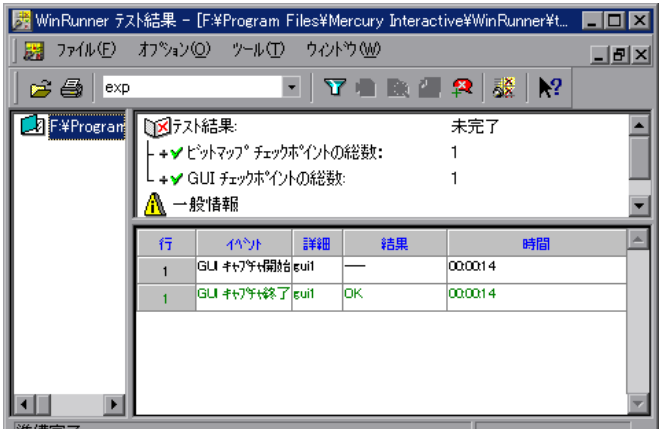
1 [**ツール**] > [**テスト結果**] を選択するか，[**テスト結果**] をクリックします。

WinRunner テスト結果ウィンドウが開きます。

2 期待結果フォルダを表示します。

➤ 統一レポート・ビューでは，［**開く**］ボタンをクリックするか，［**ファイル**］＞［**開く**］を選択します。［テスト結果を開く］ダイアログ・ボックスが開きます。「**exp**」を選択して，［**開く**］をクリックします。

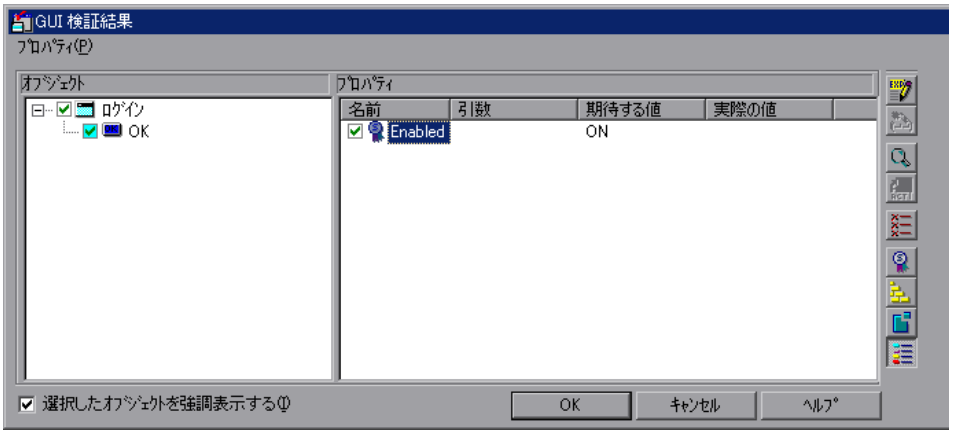
➤ WinRunner レポート・ビューでは，［**結果**］ボックスで「exp」を選択します。



3 ［**イベント**］カラムから「**GUI 検査終了**」イベントを探して GUI チェックポイントを見つけます。

注：WinRunner レポート・ビューで作業している場合は，［**TSL を表示**］ボタンを使用して，強調表示された行番号でテスト・スクリプトを開くことができます。

4 希望の「**GUI 検査終了**」エントリを選択して表示します。［GUI 検証結果］ダイアログ・ボックスが開きます。



5 期待結果を変更したいプロパティ検査を選択します。［**期待結果値の編集**］ボタンをクリックします。［**期待する値**］カラムで，値を変更します。［OK］をクリックして，ダイアログ・ボックスを閉じます。

注：プロパティ検査の期待値は，GUI チェックポイントを作成している間に変更することもできます。詳細については 173 ページ「プロパティの期待値の編集」を参照してください。

GUI チェックポイントの期待値は，テスト実行後に変更することもできます。詳細については，487 ページ「WinRunner レポート・ビューでのチェックポイントの期待結果の更新」を参照してください。

［テスト結果］ウィンドウでの作業の詳細については，第 2 章「統一レポート・ビューでのテスト結果の分析」を参照してください。

第 10 章

Web オブジェクトでの作業

WebTest アドインとともに WinRunner を使用すると，Netscape および Internet Explorer で Web サイトのオブジェクトに対する状況依存的な操作の記録，再生を行えます。

WebTest アドインでは，Web オブジェクトのプロパティの表示や，Web サイトの Web オブジェクトの情報の取得，Web サイトの機能性を検査するチェックポイントの作成ができます。

注：AOL のブラウザを使用して，サイトの Web オブジェクトを対象にテストを記録し実行できますが，［戻る］，［次へ］，［移動］等のブラウザ要素を対象にオブジェクトを記録または実行することはできません。

本章では，以下の項目について説明します。

- Web オブジェクトの作業について
- 記録済み Web オブジェクトのプロパティ表示
- テストでの Web オブジェクト・プロパティの使用
- Web オブジェクトの検査について

Web オブジェクトの作業について

WebTest アドインでテストを作成すると，WinRunner はフレーム，テキスト・リンク，画像，テーブル，Web フォーム・オブジェクトのような Web オブジェクトを認識します。各オブジェクトには異なる複数のプロパティがあります。これらのプロパティを使用して，オブジェクトの認識，プロパティの値の取得および検査，Web 関数の実行ができます。

Web サイトが期待した通りに作動するかどうかを検査することもできます。例えば，フレーム，テーブル，セルの内容や構成，URL のリンク，画像のソースおよび種類，テキスト・リンクの色やフォントを検査します。

注：Web サイトのテストを始めるためにブラウザを開く前に，WebTest アドインがロードされた状態で WinRunner を起動してください。詳細については，21 ページ「WinRunner アドインのロード」を参照してください。

記録済み Web オブジェクトのプロパティ表示

GUI スパイの［**記録済み**］タブを使用して，ユーザが Windows オブジェクトを記録するのと同じように WinRunner が選択したオブジェクトに対して記録するオブジェクトのプロパティおよびその値を見ることができます。

記録済み Web オブジェクトのプロパティは次のように表示します。

1 WinRunner を起動します。

2 Web ブラウザを開きます。

注：Web ブラウザを開く前に，WebTest アドインがロードされた状態で WinRunner を起動してください。

---

3 ［**ツール**］＞［**GUI スパイ**］を選択して［GUI スパイ］ダイアログ・ボックスを開きます。

4 Web サイトのオブジェクトをスパイする間に WinRunner ウィンドウを表示したくない場合は，［**WinRunner を非表示にする**］を選択します（GUI スパイ・ウィンドウではありません）。

5 ［**スパイ**］をクリックして Web ページ内のオブジェクトを指します。オブジェクトは強調表示されウィンドウ名，オブジェクト名，記録済みプロパティ，およびその値が表示されます。

6 ［GUI スパイ］ダイアログ・ボックス内のオブジェクトの記述をキャプチャするには，任意のオブジェクトを指し，STOP ショートカット・キーを押します。（標準のショートカット・キーの組み合わせは左 Ctrl ＋ F3 です）。

GUI スパイの詳細は，34 ページ「GUI オブジェクトのプロパティの表示」を参照してください。

注：

GUI スパイの［**すべて標準**］タブでは，記録されていない追加 Web オブジェクトのプロパティは表示しません。各 Web オブジェクトに関連するプロパティの一覧は，テストでの Web オブジェクト・プロパティの使用を参照してください。

［GUI マップの構成設定］ツールですべての Web オブジェクトの設定を行うことはできません。［GUI マップの構成設定］ツールでは，*html_frame*，*html_edit*，*html_check_button*，*html_combobox*，*html_listbox*，*html_radio_button*，*html_push_button* といったウィンドウ・ハンドル（HWND）で WinRunner の Web オブジェクトの認識方法を変更します。［GUI マップの構成設定］ツールで *html_text_link* および *html_rect* といった Web 指向オブジェクトの認識方法を設定することはできません。認識方法の変更は，GUI マップ構成を定義する関数（**set_record_attr**，**set_record_method**）を使用します。

［GUI マップの構成設定］ツールの詳細については，第 24 章「GUI マップの構成設定」を参照してください。GUI マップ構成を定義する TSL 関数の詳細については，「**TSL リファレンス**」を参照してください。

テストでの Web オブジェクト・プロパティの使用

チェックポイントを作成するには，記述的プログラミングを使用してステートメントを書きます。**web_obj_get_info** や **_web_set_tag_attr** などの TSL 関数を利用するには各 Web オブジェクトで使用できるプロパティを知る必要があります。

この節では次の各 Web オブジェクトで利用できるプロパティを一覧にして定義します。

➤ Web オブジェクトのプロパティの使用

- フレーム・オブジェクト・プロパティの使用
- Web 画像プロパティの使用
- テキスト・リンク・プロパティの使用
- Web テーブル・プロパティの使用およびテーブル・セルの使用
- フォーム・オブジェクト・プロパティの使用（ラジオ・ボタン，チェック・ボックス，編集ボックス，リスト，コンボ・ボックス，Web ボタンなどを含みます）。

Web オブジェクトの検査の詳細は，191 ページ「Web オブジェクトの検査について」を参照してください。

プログラミングの詳細は，第 28 章「プログラミングによるテスト・スクリプトの 機能強化」を参照してください。

web_obj_get_info など，Web サイトのテストに利用できる TSL 関数の詳細については，「**TSL リファレンス**」を参照してください。

Web オブジェクトのプロパティの使用

次の表は，Web フレーム（html_frame class）を除く全 Web オブジェクトに適用されるオブジェクトのプロパティです。

プロパティ名	説明
attribute/<prop_name>	オブジェクトの特定内部プロパティにアクセスできます。詳細は，183 ページ「Attribute/<prop_name> コメントの使用」を参照してください。
bgcolor	オブジェクトの背景色。
class	オブジェクトの WinRunner クラス。
class_name	HTML で表示されるオブジェクトのクラス。
color	オブジェクトの色。
current_bgcolor	現在のスタイルで定義されている項目の背景色プロパティ。 Internet Explorer でのみサポートされています。
current_color	現在のスタイルで定義されている項目の色プロパティ。

プロパティ名	説明
focused	オブジェクトにフォーカスがあるかどうかを示します。 **可能値 :** **1:** 真 **0:** 偽
height	オブジェクトの高さ（単位：ピクセル）。
html_id	オブジェクトの HTML 識別子。
inner_html	オブジェクトの開始および終了タグの間に含まれる HTML コード。
inner_text	オブジェクトの開始および終了タグの間に含まれるテキスト。
outer_html	オブジェクトの HTML コードおよびその内容。 Internet Explorer でのみサポートされています。
source_index	オブジェクトの HTML タグがソース・コードに現れる順序を他の HTML タグと比べて示すために WinRunner がオブジェクトに割り当てるセレクタの値。 **開始値 = 0.** Internet Explorer でのみサポートされています。
tag_name	オブジェクトの HTML タグ。
visible	オブジェクトが可視状態かどうかを示します **可能値 :** **1:** 真 **0:** 偽
width	オブジェクトの幅（単位：ピクセル）。

Attribute/<prop_name> コメントの使用

attribute/<prop_name> コメントを使用して，内部プロパティに応じて，Internet Explorer 内の Web オブジェクトを認識できます。

例えば，同じ Web ページ内で会社ロゴ・マークが 2 つあるとします。

```
<IMG src="logo.gif" LogoID="122">
<IMG src="logo.gif" LogoID="123">
```

次のように，オブジェクトの記述でユーザ定義の **LogoID** プロパティを使用してプログラミングするとクリックしたいロゴ画像を認識できます。

```
web_image_click("{class: object, MSW_class: html_rect, logoID: 123}" , 164 , 253 )
```

プログラミングの詳細は，第 28 章「プログラミングによるテスト・スクリプトの 機能強化」を参照してください。

オブジェクト・クラスの論理名に使用するプロパティの設定

Web オブジェクト・クラスには，定義済みの標準設定プロパティがあります。この値はオブジェクトの論理名として使用されます。Web オブジェクト・クラスの標準設定の論理名プロパティは，**_web_set_tag_attr** 関数を使用して変更できます。

オブジェクトの論理名にユーザ定義のプロパティを使用する場合は，**_web_set_tag_attr** ステートメントで **attribute/<prop_name>** 表記法を使用できます。

例えば，Web ページに次のソース・コードがあるとします。

```
<input type="text" name="InputName1" maxlength="20" size="20" value="name" MyAttr="Your Name">
<input type="text" name="InputName2" maxlength="20" size="20" value="name" MyAttr="My Name">
```

標準設定では，論理名としてテキスト・ボックス（上記の例では，InputName1 または InputName2）の name 属性が使用されます。論理名として **MyAttr** プロパティの値を 使用する場合は，次の行を使用します。

```
_web_set_tag_attr("html_edit", "attribute/MyAttr");
```

詳細については，「**TSL リファレンス**」を参照してください。

フレーム・オブジェクト・プロパティの使用

html_frame MSW クラス・オブジェクトで作業する場合に次のオブジェクト・プロパティが使用できます。

プロパティ名	説明
frame_title	フレーム名
html_id	フレームの HTML 識別子。Netscape 4.x ではサポートされていません。
name	オブジェクト の WinRunner 名。これが WinRunner が使用するオブジェクトの論理名です。このプロパティの値はフレームの **name** プロパティが存在する場合は，それが使用されます。無い場合，フレームの **title** プロパティが存在する場合は，それが使用されます。それ以外の場合は，フレームの **url** プロパティが使用されます。
page_title	フレームに含まれるページ名
url	フレームの URL

Web 画像プロパティの使用

html_rect MSW クラス・オブジェクトで作業する場合には，全オブジェクトでサポートされているプロパティに加え，次のプロパティが使用できます。

プロパティ名	説明
alt	オブジェクトのボタン名
element_name	<IMG> タグ内で指定されている名前のプロパティ
file_name	オブジェクトのファイル名（パスは除く）
name	オブジェクト の WinRunner 名。これが WinRunner が使用するオブジェクトの論理名です。このプロパティの値は画像の **alt** プロパティが存在する場合は，それが使用されます。無い場合，画像の **name** プロパティが存在する場合は，それが使用されます。それ以外の場合は，画像の **src** プロパティが使用されます。

プロパティ名	説明
src	オブジェクトのソース位置（完全パス）
type	画像の種類 **可能値 :** **Server side** **Client side** **Plain**

テキスト・リンク・プロパティの使用

html_text_link MSW クラス・オブジェクトで作業する場合には，全オブジェクトでサポートされているプロパティに加え，次のプロパティが使用できます。

プロパティ名	説明
currrent_font	スタイルで定義されているリンクのフォント・プロパティ
element_name	<A HREF> タグ内で指定された name プロパティ
font	リンクのフォント
text	リンクに関連するテキスト
url	リンクの URL

Web テーブル・プロパティの使用

テーブルで作業する場合，テーブル・オブジェクトまたはセル・オブジェクトで関数を実行できます。

テーブル

html_table MSW クラス・オブジェクトで作業する場合には，全オブジェクトでサポートされているプロパティに加え，次のプロパティが使用できます。

プロパティ名	説明
columns	テーブル内の列数
name	オブジェクト の WinRunner 名。これが WinRunner が使用するオブジェクトの論理名です。このプロパティの値には，テーブル内で **name** プロパティが存在する最初のオブジェクトの値が使用されます。
rows	テーブル内の行数
table_index	テーブルがソース・コードに現れる順序を他の テーブルと比べて示すためのセレクタの値。 **開始値 = 0.**
text	テーブル内に含まれるテキスト

テーブル・セル

html_cell MSW クラス・オブジェクトで作業する場合には，全オブジェクトでサポートされているプロパティに加え，次のプロパティが使用できます。

プロパティ名	説明
columns	セルがあるテーブル列。テーブル内の最初の列は 1。
rows	セルがあるテーブル行。テーブル内の最初の行は 1。
table_index	セルのテーブルがソース・コードに現れる順序を他のテーブルと比べて示すためのセレクタの値。 **開始値 = 0.**
text	セル内に含まれるテキスト

フォーム・オブジェクト・プロパティの使用

Web フォームで作業する場合は，ラジオ・ボタン，チェック・ボックス，編集ボックス，リスト・ボックス，コンボ・ボックス，ボタンに関数を使用できます。

ラジオ・ボタン

html_radio_button MSW クラス・オブジェクトで作業する場合には，全オブジェクトでサポートされているプロパティに加え，次のプロパティが使用できます。

プロパティ名	説明
checked	ラジオ・ボタンが選択されているかどうかを示します。 **可能値 :** **1:** 真 **0:** 偽
element_name	<input> タグ内で指定された name プロパティ。
enabled	ラジオ・ボタンが選択できるように有効になっているかを示します。 **可能値 :** **1:** 真 **0:** 偽
name	オブジェクトの WinRunner 名。これが WinRunner が使用するオブジェクトの論理名です。このプロパティの値には，ラジオ・ボタンの **name** プロパティ値が使用されます。
part_value	ボタンの値（ラベル） **Internet Explorer でのみサポートされています。**
value	ボタンの値（ラベル）

チェック・ボックス

html_check_button MSW クラス・オブジェクトで作業する場合には，全オブジェクトでサポートされているプロパティに加え，次のプロパティが使用できます。

プロパティ名	説明
checked	チェック・ボックスが選択されているかどうかを示します。 **可能値 :** **1:** 真 **0:** 偽
element_name	<input> タグ内で指定された name プロパティ。

プロパティ名	説明
enabled	チェック・ボックスが選択できるように有効になっているかを示します。 **可能値 :** **1:** 真 **0:** 偽
name	オブジェクトの WinRunner 名。これが WinRunner が使用するオブジェクトの論理名です。このプロパティの値には，チェック・ボックスの **name** プロパティ値が使用されます。
part_value	ボタンの値（ラベル） **Internet Explorer でのみサポートされています。**
value	ボタンの値（ラベル）

編集ボックス

html_edit MSW クラス・オブジェクトで作業する場合には，全オブジェクトでサポートされているプロパティに加え，次のプロパティが使用できます。

プロパティ名	説明
cols	編集ボックス（列）の幅
element_name	<input> タグ内で指定された name プロパティ。
enabled	編集ボックスが選択できるように有効になっているかを示します。 **可能値 :** **1:** 真 **0:** 偽
kind	編集ボックスの種類を示します。 **可能値 :** **single-line** **multi-line**
name	オブジェクトの WinRunner 名。これが WinRunner が使用するオブジェクトの論理名です。このプロパティの値には，ラジオ・ボタンの **name** プロパティ値が使用されます。
rows	編集ボックスの高さ（行）
type	HTML タグで定義されたオブジェクトの種類 例 : <input type=text>

リストおよびコンボ・ボックス

html_listbox および **hmtl_combobox** MSW クラス・オブジェクトで作業する場合には，全オブジェクトでサポートされているプロパティに加え，次のプロパティが使用できます。

プロパティ名	説明
element_name	<input> タグ内で指定された name プロパティ。
is_multiple	リストが複数の項目から選択できるようになっているかを示します。 **可能値 :** **1:** 真 **0:** 偽
name	オブジェクトの WinRunner 名。これが WinRunner が使用するオブジェクトの論理名です。このプロパティの値には，ラジオ・ボタンの **name** プロパティ値が使用されます。
selection	リストで選択されている項目（セミコロンで区切ります）

Web ボタン

html_push_button MSW クラス・オブジェクトで作業する場合には，全オブジェクトでサポートされているプロパティに加え，次のプロパティが使用できます。

プロパティ名	説明
element_name	<input> タグ内で指定された name プロパティ。
enabled	ボタンが選択できるように有効になっているかを示します。 **可能値 :** **1:** 真 **0:** 偽
name	オブジェクトの WinRunner 名。これが WinRunner が使用するオブジェクトの論理名です。このプロパティの値はボタンの **value** プロパティが存在する場合は，それが使用されます。無い場合，ボタンの **innertext** プロパティが存在する場合は，それが使用されます。それ以外の場合は，ボタンの **name** プロパティが使用されます。

プロパティ名	説明
part_value	ボタンの HTML タグが <INPUT> の場合のボタンの「value」プロパティ値。ボタンの HTML タグが <BUTTON> の場合にはボタンの「innertext」プロパティ値。 **Internet Explorer でのみサポートされています。**
value	ボタンの値（ラベル）

Web オブジェクトの検査について

テスト・スクリプト内で GUI チェクポイントを使って，Web サイトの Web オブジェクトの振る舞いを検査できます。実行したテスト間で Web ページのフレーム，テーブル，セル，リンク，画像に違いがあるかどうかを検査できます。WinRunner が推奨する標準のプロパティに基づいて，GUI チェックポイントを定義するか，その他のプロパティを選択して，ユーザ定義の検査を定義します。GUI チェックポイントに関する一般的な情報については，第 9 章「GUI オブジェクトの検査」を参照してください。

また，Web オブジェクトおよび Web ページ内のテキストの読み取りと検査を行うテキスト・チェックポイントを，テスト・スクリプトに追加することもできます。

次のチェックポイントを作成できます。

- 標準的なフレーム・プロパティの検査
- フレーム内のオブジェクト数の検査
- フレーム，テーブル，セルの構造の検査
- フレーム，セル，リンク，画像の内容の検査
- テーブル内の列数と行数の検査
- リンクの URL の検査
- 画像と画像リンクのソースまたはタイプの検査
- テキスト・リンクの色またはフォントの検査
- 壊れたリンクの検査
- フレーム内のリンクと画像の検査

- テーブルの内容の検査
- テーブルの内容の検査
- テーブル内のセルの検査
- テキストの検査

標準的なフレーム・プロパティの検査

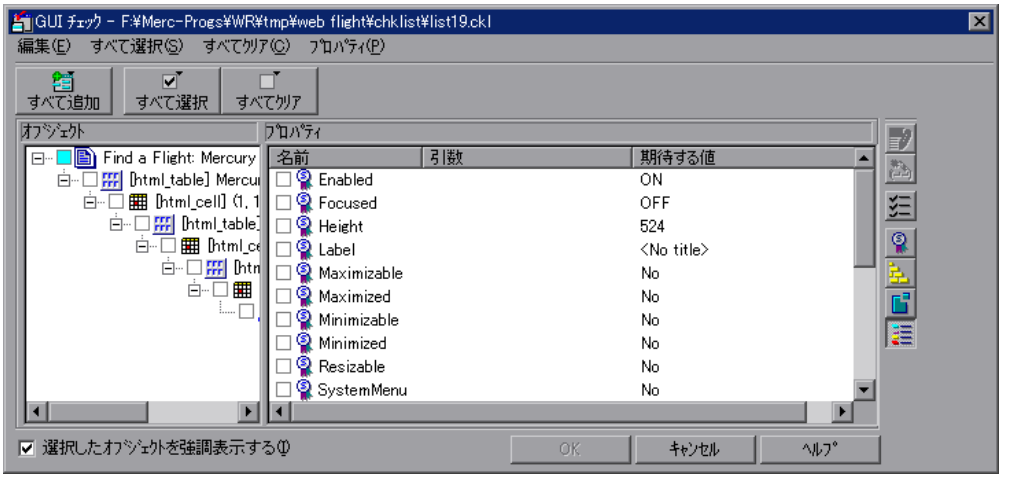
標準的なフレーム・プロパティを検査する GUI チェックポイントを作成できます。

標準的なフレーム・プロパティを検査するには，次の手順を実行します。

1 ［**挿入**］＞［**GUI チェックポイント**］＞［**オブジェクト / ウィンドウ**］を選択します。

WinRunner のウィンドウが最小化されアイコンになり，マウス・ポインタが指差しポインタに変わり，ヘルプ・ウィンドウが開きます。

2 Web ページ上のオブジェクト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，オブジェクトが強調表示されます。



3 ［**オブジェクト**］列で，フレームが選択されていることを確認します。

［プロパティ］列には，そのフレームに対して用意されている標準プロパティと標準検査が表示されます。

4 ［**プロパティ**］列で，検査するプロパティを選択します。

次の標準プロパティを検査できます。

- ［**Enabled**］は，フレームを選択できるかどうかを検査します。
- ［**Focused**］は，キーボードの入力がこのフレームに送信されるかどうかを検査します。
- ［**Label**］はフレームのラベルを検査します。
- ［**Minimizable**］，［**Maximizable**］，［**Minimized**］，［**Maximized**］は，フレームを最小化または最大化できるかどうかを検査します。
- ［**Resizable**］は，フレームのサイズを変更できるかどうかを検査します。
- ［**SystemMenu**］は，フレームにシステム・メニューがあるかどうかを検査します。
- ［**Width**］と［**Height**］は，フレームの幅と高さをピクセル単位で検査します。
- ［**X**］と［**Y**］は，フレームの左上角の x 座標と y 座標を検査します。

5 ［**OK**］をクリックし，ダイアログ・ボックスを閉じます。

WinRunner はオブジェクトの情報をキャプチャし，それをテストの期待結果フォルダに格納します。画面が WinRunner のウィンドウに戻り，チェックポイントがテスト・スクリプトに **win_check_gui** ステートメントとして表示されます。**win_check_gui** 関数の詳細については，「**TSL リファレンス**」を参照してください。

フレーム内のオブジェクト数の検査

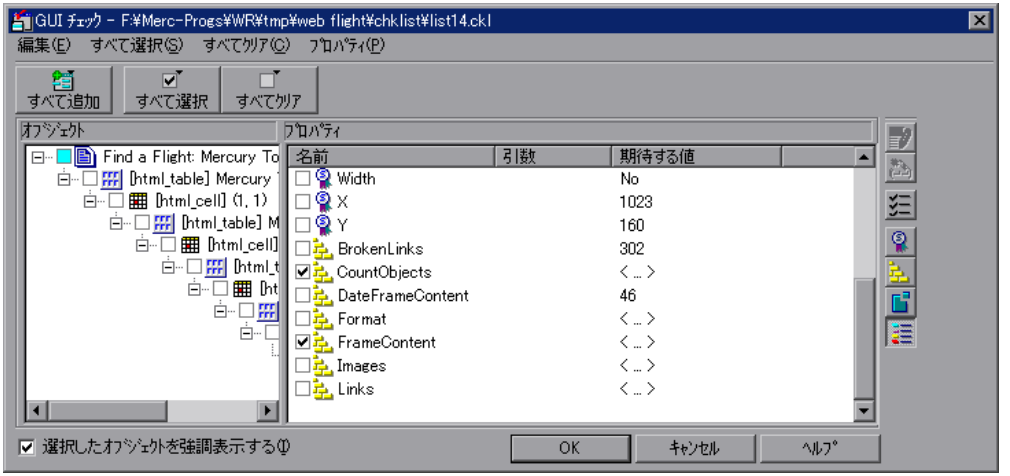
フレーム内のオブジェクトの数を検査する GUI チェックポイントを作成できます。

フレーム内のオブジェクト数を検査するには，次の手順を実行します。

1 ［**挿入**］＞［**GUI チェックポイント**］＞［**オブジェクト / ウィンドウ**］を選択します。

WinRunner のウィンドウが最小化されアイコンになり，マウス・ポインタが指差しポインタに変わり，ヘルプ・ウィンドウが開きます。

2 Web ページ上のオブジェクト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，オブジェクトが強調表示されます。



3 ［**オブジェクト**］列で，フレームが選択されていることを確認します。

［プロパティ］列には，検査できるプロパティが表示されます。

4 ［**プロパティ**］列で，［**CountObjects**］チェック・ボックスを選択します。

5 プロパティの期待値を編集するには，［**CountObjects**］を強調表示させます。

［**期待結果値の編集**］ボタンをクリックするか，［**期待する値**］列の値をダブルクリックし，値を編集します。スピン・ボックスが開きます。

オブジェクト数の期待値を入力します。

6 ［**OK**］をクリックし，［GUI チェック］ダイアログ・ボックスを閉じます。

WinRunner はオブジェクトの情報をキャプチャし，それをテストの期待結果フォルダに格納します。画面が WinRunner のウィンドウに戻り，チェックポイントがテスト・スクリプトに **win_check_gui** ステートメントとして表示されます。**win_check_gui** 関数の詳細については，「**TSL リファレンス**」を参照してください。

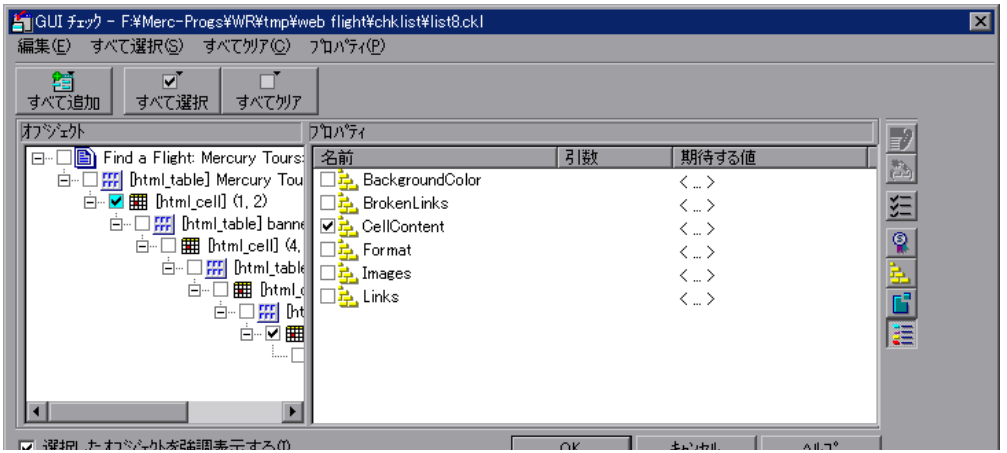
フレーム，テーブル，セルの構造の検査

Web ページ上のフレーム，テーブル，セルの構造を検査する GUI チェックポイントを作成できます。

フレーム，テーブル，セルの構造を検査するには，次の手順を実行します。

1 ［**挿入**］＞［**GUI チェックポイント**］＞［**オブジェクト / ウィンドウ**］を選択します。WinRunner のウィンドウが最小化されアイコンになり，マウス・ポインタが指差しポインタに変わり，ヘルプ・ウィンドウが開きます。

2 Web ページ上のオブジェクト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，オブジェクトが強調表示されます。



3 ［**オブジェクト**］列で，オブジェクトを選択します。

［プロパティ］列には，検査できるプロパティが表示されます。

4 ［**プロパティ**］列で，［**Format**］チェック・ボックスを選択します。

5 プロパティの期待値を編集するには，［**Format**］を強調表示させます。

［**期待結果値の編集**］ボタンをクリックするか，［**期待する値**］列の値をダブルクリックし，値を編集します。フレーム，テーブル，セルの構造を示すテキスト・ファイルが，メモ帳で開きます。

期待される構造を変更します。

テキスト・ファイルを保存し，メモ帳を閉じます。

6 ［**OK**］をクリックし，［GUI チェック］ダイアログ・ボックスを閉じます。

WinRunner はオブジェクトの情報をキャプチャし，それをテストの期待結果フォルダに格納します。画面が WinRunner のウィンドウに戻り，チェックポイントがテスト・スクリプトに **obj_check_gui** または **win_check_gui** ステートメ

ントとして表示されます。**obj_check_gui** および **win_check_gui** 関数の詳細については，「**TSL リファレンス**」を参照してください。

フレーム，セル，リンク，画像の内容の検査

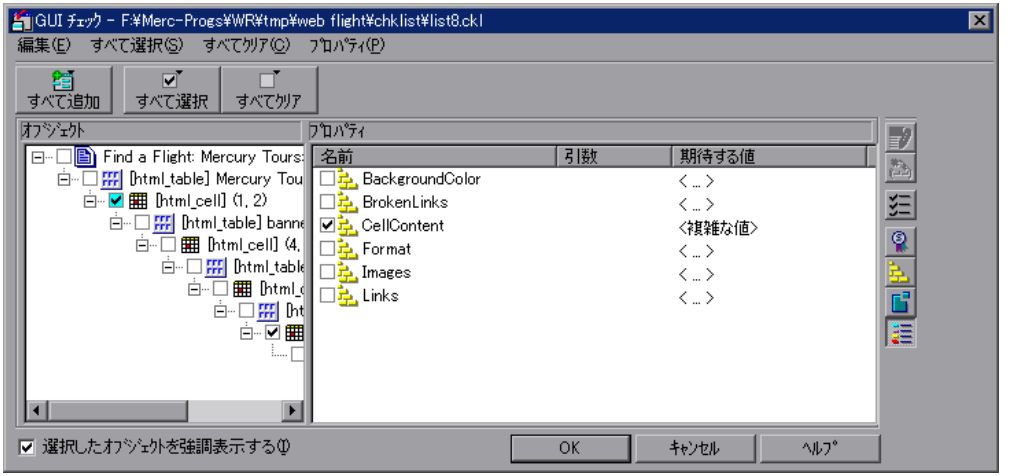
フレーム，セル，テキスト・リンク，画像リンク，画像の内容を検査する GUI チェックポイントを作成できます。テーブルの内容を検査するには，207 ページ「テーブルの内容の検査」を参照してください。

内容を検査するには，次の手順を実行します。

1 ［**挿入**］＞［**GUI チェックポイント**］＞［**オブジェクト / ウィンドウ**］を選択します。

WinRunner のウィンドウが最小化されアイコンになり，マウス・ポインタが指差しポインタに変わり，ヘルプ・ウィンドウが開きます。

2 Web ページ上のオブジェクト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，オブジェクトが強調表示されます。



3 ［**オブジェクト**］列で，オブジェクト（フレーム，セル，テキスト・リンク，画像リンク，画像のいずれか）を選択します。［プロパティ］列には，検査できるプロパティが表示されます。

4 ［**プロパティ**］列で，次のいずれかの検査を選択します。

➤ オブジェクトがフレームである場合，［**FrameContent**］チェック・ボックスを選択します。

- オブジェクトがセルである場合，［**CellContent**］チェック・ボックスを選択します。
- オブジェクトがテキスト・リンクである場合，［**Text**］チェック・ボックスを選択します。
- オブジェクトが画像リンクである場合，［**ImageContent**］チェック・ボックスを選択します。
- オブジェクトが画像である場合，［**ImageContent**］チェック・ボックスを選択します。

5 プロパティの期待値を編集するには，プロパティを強調表示させます。

［**ImageContent**］プロパティの期待値は編集できません。

6 ［**期待値を編集**］ボタンをクリックするか，［**期待する値**］列の値をダブルクリックし，値を編集します。

- ［**FrameContent**］プロパティの場合，エディタが開きます。
- ［**CellContent**］プロパティの場合，エディタが開きます。
- ［**Text**］プロパティの場合，編集ボックスが開きます。

7 期待値を変更します。

8 ［**OK**］をクリックし，［GUI チェック］ダイアログ・ボックスを閉じます。

WinRunner はオブジェクトの情報をキャプチャし，それをテストの期待結果フォルダに格納します。画面が WinRunner のウィンドウに戻り，チェックポイントがテスト・スクリプトに **obj_check_gui** または **win_check_gui** ステートメントとして表示されます。**obj_check_gui** および **win_check_gui** 関数の詳細については，「**TSL リファレンス**」を参照してください。

テーブル内の列数と行数の検査

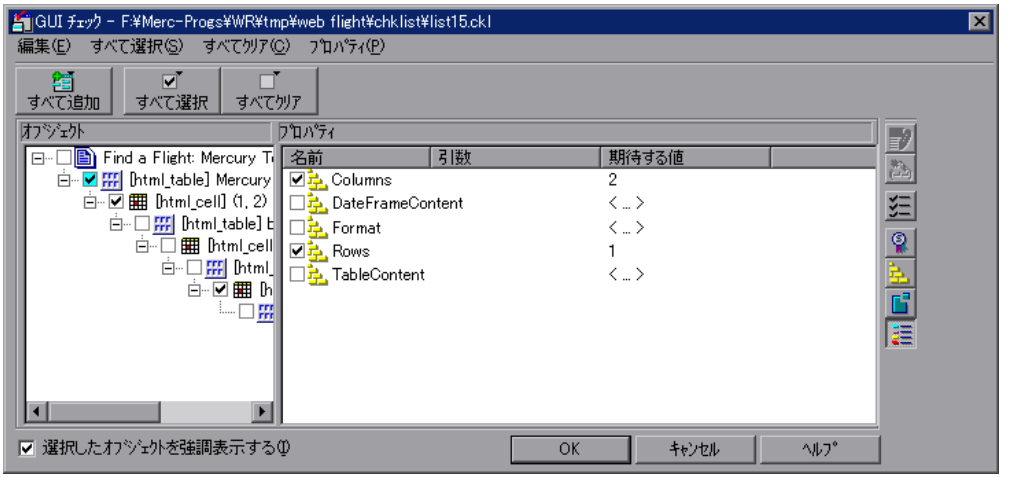
テーブル内の列数と行数を検査する GUI チェックポイントを作成できます。

テーブル内の列数と行数を検査するには，次の手順を実行します。

1 ［**挿入**］＞［**GUI チェックポイント**］＞［**オブジェクト / ウィンドウ**］を選択します。

WinRunner のウィンドウが最小化されアイコンになり，マウス・ポインタが指差しポインタに変わり，ヘルプ・ウィンドウが開きます。

2 Web ページ上のテーブル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，オブジェクトが強調表示されます。



3 ［**オブジェクト**］列で，テーブルが選択されていることを確認します。［プロパティ］列には，検査できるプロパティが表示されます。

4 ［**プロパティ**］列で，［**Columns**］または［**Rows**］チェック・ボックスを選択します。

5 プロパティの期待値を編集するには，［**Colimns**］または［**Rows**］を強調表示させます。

➤ ［**期待結果値の編集**］ボタンをクリックするか，［**期待する値**］列の値をダブルクリックし，値を編集します。スピン・ボックスが開きます。

➤ プロパティの期待値を任意の値に変更します。

6 ［**OK**］をクリックし，［GUI チェック］ダイアログ・ボックスを閉じます。

WinRunner はオブジェクトの情報をキャプチャし，それをテストの期待結果フォルダに格納します。画面が WinRunner のウィンドウに戻り，チェックポイントがテスト・スクリプトに **obj_check_gui** または **win_check_gui** ステートメントとして表示されます。**obj_check_gui** および **win_check_gui** 関数の詳細については，「**TSL リファレンス**」を参照してください。

リンクの URL の検査

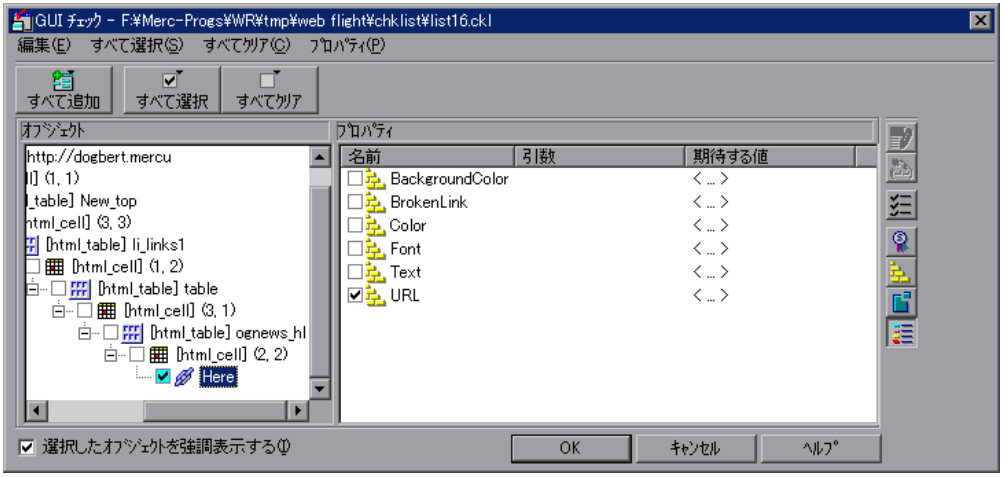
Web ページ内のテキスト・リンクまたは画像リンクの URL を検査する GUI チェックポイントを作成できます。

リンクの URL を検査するには，次の手順を実行します。

1 ［**挿入**］＞［**GUI チェックポイント**］＞［**オブジェクト / ウィンドウ**］を選択します。

WinRunner のウィンドウが最小化されアイコンになり，マウス・ポインタが指差しポインタに変わり，ヘルプ・ウィンドウが開きます。

2 Web ページ上のテキスト・リンク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，オブジェクトが強調表示されます。



3 ［**オブジェクト**］列で，リンクが選択されていることを確認します。［**プロパティ**］列には，検査できるプロパティが表示されます。

4 リンクのアドレスを検査するために，［**プロパティ**］列で［**URL**］チェック・ボックスを選択します。

5 URL プロパティの期待値を編集するには，［**URL**］を強調表示させます。

➤ ［**期待結果値の編集**］ボタンをクリックするか，［**期待する値**］列の値をダブルクリックし，値を編集します。編集ボックスが開きます。

➤ 期待値を編集します。

6 ［**OK**］をクリックし，［GUI チェック］ダイアログ・ボックスを閉じます。

WinRunner はオブジェクトの情報をキャプチャし，それをテストの期待結果フォルダに格納します。画面が WinRunner のウィンドウに戻り，チェックポイントがテスト・スクリプトに **obj_check_gui** ステートメントとして表示されます。**obj_check_gui** 関数の詳細については，「**TSL リファレンス**」を参照してください。

画像と画像リンクのソースまたはタイプの検査

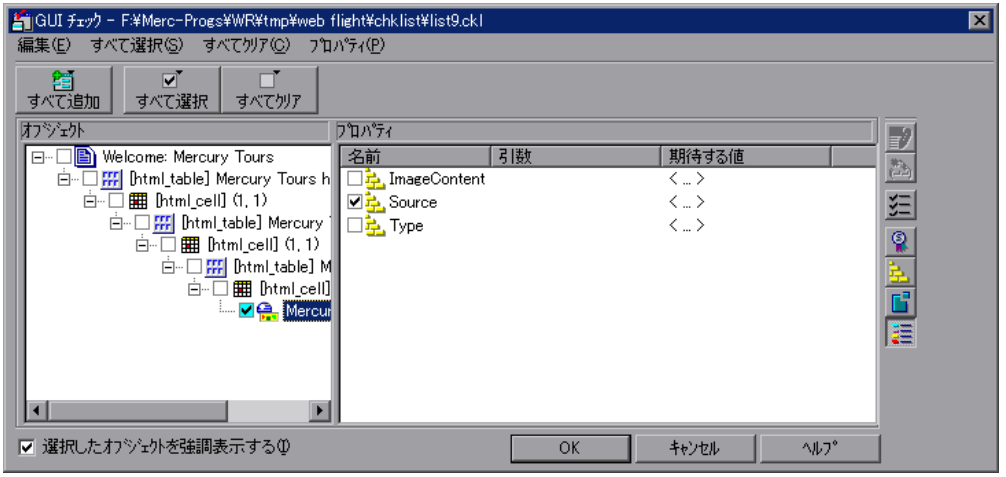
Web ページ内の画像または画像リンクのソースとタイプを検査する GUI チェックポイントを作成できます。

画像または画像リンクのソースまたはタイプを検査するには，次の手順を実行します。

1 ［**挿入**］＞［**GUI チェックポイント**］＞［**オブジェクト / ウィンドウ**］を選択します。

WinRunner のウィンドウが最小化されアイコンになり，マウス・ポインタが指差しポインタに変わり，ヘルプ・ウィンドウが開きます。

2 Web ページ上の画像または画像リンク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，オブジェクトが強調表示されます。



3 ［**オブジェクト**］列で，画像または画像リンクが選択されていることを確認します。［プロパティ］列には，検査できるプロパティが表示されます。

4 ［プロパティ］列で，プロパティの検査を選択します。

- ［**Source**］は，画像の場所を示します。
- ［**Type**］は，オブジェクトが単なる画像，画像リンク，画像マップのいずれであるかを示します。

5 プロパティの期待値を編集するには，プロパティを強調表示させます。

- ［**期待結果値の編集**］ボタンをクリックするか，［**期待する値**］列の値をダブルクリックし，値を編集します。編集ボックスが開きます。
- 期待値を編集します。

6 ［**OK**］をクリックし，［GUI チェック］ダイアログ・ボックスを閉じます。

WinRunner はオブジェクトの情報をキャプチャし，それをテストの期待結果フォルダに格納します。画面が WinRunner のウィンドウに戻り，チェックポイントがテスト・スクリプトに **obj_check_gui** ステートメントとして表示されます。**obj_check_gui** 関数の詳細については，「**TSL リファレンス**」を参照してください。

テキスト・リンクの色またはフォントの検査

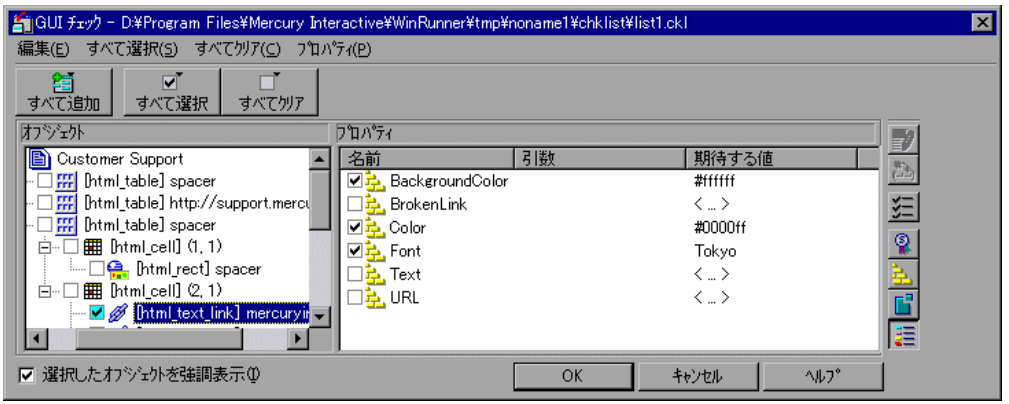
Web ページ内のテキスト・リンクの色とフォントを検査する GUI チェックポイントを作成できます。

テキスト・リンクの色またはフォントを検査するには，次の手順を実行します。

1 ［**挿入**］＞［**GUI チェックポイント**］＞［**オブジェクト / ウィンドウ**］を選択します。

WinRunner のウィンドウが最小化されアイコンになり，マウス・ポインタが指差しポインタに変わり，ヘルプ・ウィンドウが開きます。

2 Web ページ上のテキスト・リンク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，オブジェクトが強調表示されます。



3 [**オブジェクト**] 列で，テキスト・リンクが選択されていることを確認します。[プロパティ] 列には，検査できるプロパティが表示されます。

4 [プロパティ] 列で，プロパティの検査を選択します。

- [**BackgroundColor**] はテキスト・リンクの背景色を示します。
- [**Color**] はテキスト・リンクの色を示します。
- [**Font**] はテキスト・リンクのフォントを示します。

5 プロパティの期待値を編集するには，プロパティを強調表示させます。

[**期待結果値の編集**] ボタンをクリックするか，[**期待する値**] 列の値をダブルクリックし，値を編集します。ボックスが開きます。

期待値を編集します。

6 [**OK**] をクリックし，[GUI チェック] ダイアログ・ボックスを閉じます。

WinRunner はオブジェクトの情報をキャプチャし，それをテストの期待結果フォルダに格納します。画面が WinRunner のウィンドウに戻り，チェックポイントがテスト・スクリプトに **obj_check_gui** ステートメントとして表示されます。

obj_check_gui 関数の詳細については，「**TSL リファレンス**」を参照してください。

壊れたリンクの検査

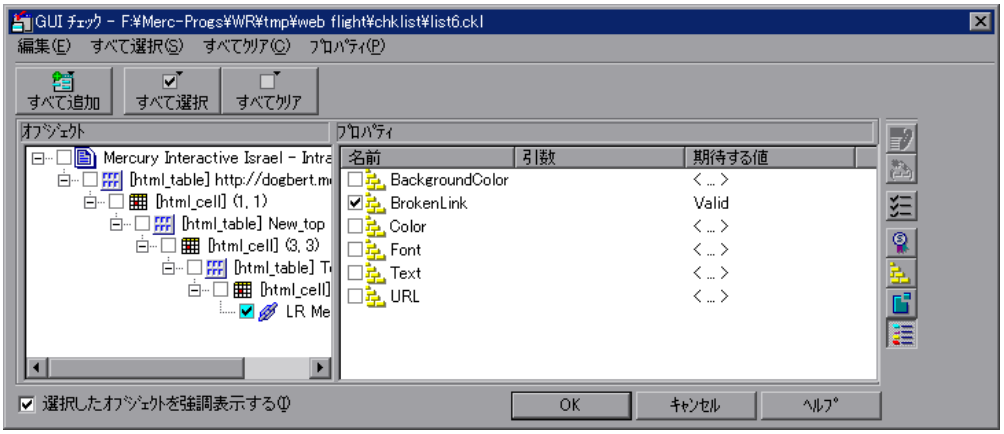
テキスト・リンクまたは画像リンクが有効かどうかを検査するチェックポイントを作成できます。フレーム内の 1 つの壊れたリンクを検査するチェックポイント，またはすべての壊れたリンクを検査するチェックポイントを作成できます。

1 つの壊れたリンクを検査するには，次の手順を実行します。

1 ［**挿入**］＞［**GUI チェックポイント**］＞［**オブジェクト / ウィンドウ**］を選択します。

WinRunner のウィンドウが最小化されアイコンになり，マウス・ポインタが指差しポインタに変わり，ヘルプ・ウィンドウが開きます。

2 Web ページ上のリンク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，オブジェクトが強調表示されます。



3 ［**オブジェクト**］列で，リンクが選択されていることを確認します。［プロパティ］列には，検査できるプロパティが表示されます。

4 ［**プロパティ**］列で，［**BrokenLink**］チェック・ボックスを選択します。

5 プロパティの期待値を編集するには，［**BrokenLink**］を強調表示させます。

［**期待結果値の編集**］ボタンをクリックするか，［**期待する値**］列の値をダブルクリックし，値を編集します。コンボ・ボックスが開きます。

［**Valid**］または［**NotValid**］を選択します。［Valid］は，リンクが有効であることを示し，［NotValid］は，リンクが壊れていることを示します。

6 ［**OK**］をクリックし，［GUI チェック］ダイアログ・ボックスを閉じます。

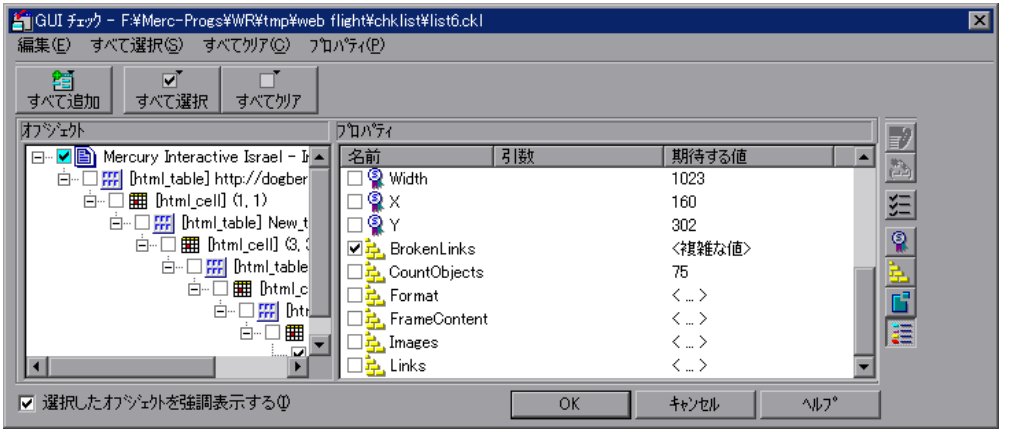
WinRunner はオブジェクトの情報をキャプチャし，それをテストの期待結果フォルダに格納します。画面が WinRunner のウィンドウに戻り，チェックポイントがテスト・スクリプトに **obj_check_gui** または **win_check_gui** ステートメントとして表示されます。**obj_check_gui** および **win_check_gui** 関数の詳細については，「**TSL リファレンス**」を参照してください。

フレーム内のすべての壊れたリンクを検査するには，次の手順を実行します。

1 ［**挿入**］＞［**GUI チェックポイント**］＞［**オブジェクト / ウィンドウ**］を選択します。

WinRunner のウィンドウが最小化されアイコンになり，マウス・ポインタが指差しポインタに変わり，ヘルプ・ウィンドウが開きます。

2 Web ページ上のオブジェクト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，オブジェクトが強調表示されます。



3 ［**オブジェクト**］列で，フレームが選択されていることを確認します。

［プロパティ］列には，検査できるプロパティが表示されます。

4 ［**プロパティ**］列で，［**BrokenLinks**］チェック・ボックスを選択します。

5 プロパティの期待値を編集するには，［**BrokenLink**］を強調表示させます。

［**期待結果値の編集**］ボタンをクリックするか，［**期待する値**］列の値をダブルクリックし，値を編集します。［チェックの編集］ダイアログ・ボックスが開きます。

検査するリンクと，使用する検証方式および検証タイプを指定できます。期待データを編集することもできます。このダイアログ・ボックスの詳細については，209 ページ「テーブル内のセルの検査」を参照してください。

終了したら，［**OK**］をクリックし，［チェックの編集］ダイアログ・ボックスを保存して閉じます。［GUI チェック］ダイアログ・ボックスに戻ります。

6 ［**OK**］をクリックし，［GUI チェック］ダイアログ・ボックスを閉じます。

WinRunner はオブジェクトの情報をキャプチャし，それをテストの期待結果フォルダに格納します。画面が WinRunner のウィンドウに戻り，チェックポイントがテスト・スクリプトに **win_check_gui** ステートメントとして表示されます。**win_check_gui** 関数の詳細については，「**TSL リファレンス**」を参照してください。

フレーム内のリンクと画像の検査

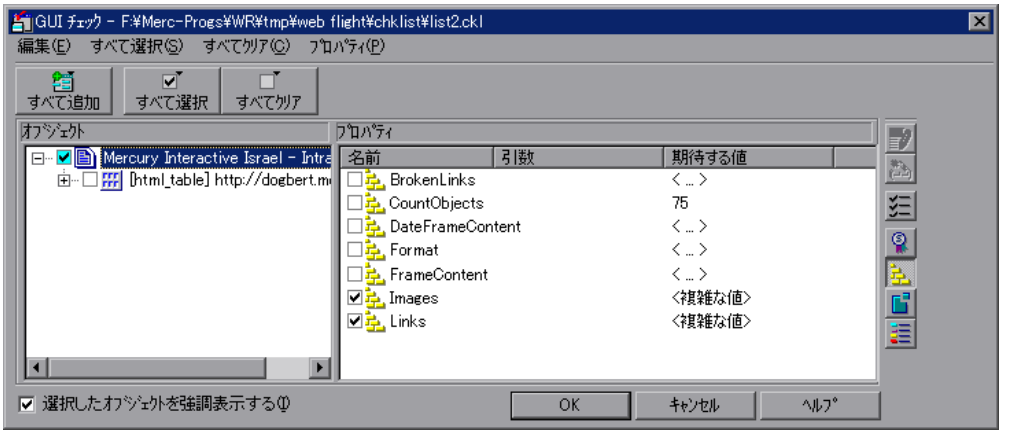
フレーム内の画像リンク，テキスト・リンク，画像を検査するチェックポイントを作成できます。

フレーム内のリンクと画像を検査するには，次の手順を実行します。

1 ［**挿入**］＞［**GUI チェックポイント**］＞［**オブジェクト / ウィンドウ**］を選択します。

WinRunner のウィンドウが最小化されアイコンになり，マウス・ポインタが指差しポインタに変わり，ヘルプ・ウィンドウが開きます。

2 Web ページ上のオブジェクト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，オブジェクトが強調表示されます。



3 [**オブジェクト**] 列で，フレーム・オブジェクトが選択されていることを確認します。

[プロパティ] 列には，検査できるプロパティが表示されます。

4 [**プロパティ**] 列で，次のいずれかの検査を選択します。

➤ 画像または画像リンクを検査するには，[**Images**] チェック・ボックスを選択します。

➤ テキスト・リンクを検査するには，[**Links**] チェック・ボックスを選択します。

5 プロパティの期待値を編集するには，[**Images**] を強調表示させます。

[**期待結果値の編集**] ボタンをクリックするか，[**期待する値**] 列の値をダブルクリックし，値を編集します。[**チェックの編集**] ダイアログ・ボックスが開きます。

テーブル内で検査する画像またはリンク，使用する検証方式と検証タイプを指定できます。期待データを編集することもできます。このダイアログ・ボックスの使い方の詳細については，209 ページ「テーブル内のセルの検査」を参照してください。

終了したら，[**OK**] をクリックし，[チェックの編集] ダイアログ・ボックスを保存して閉じます。[GUI チェック] ダイアログ・ボックスに戻ります。

6 [**OK**] をクリックし，[GUI チェック] ダイアログ・ボックスを閉じます。

WinRunner はオブジェクトの情報をキャプチャし，それをテストの期待結果フォルダに格納します。画面が WinRunner のウィンドウに戻り，チェックポイントがテスト・スクリプトに **win_check_gui** ステートメントとして表示されます。**win_check_gui** 関数の詳細については，「**TSL リファレンス**」を参照してください。

テーブルの内容の検査

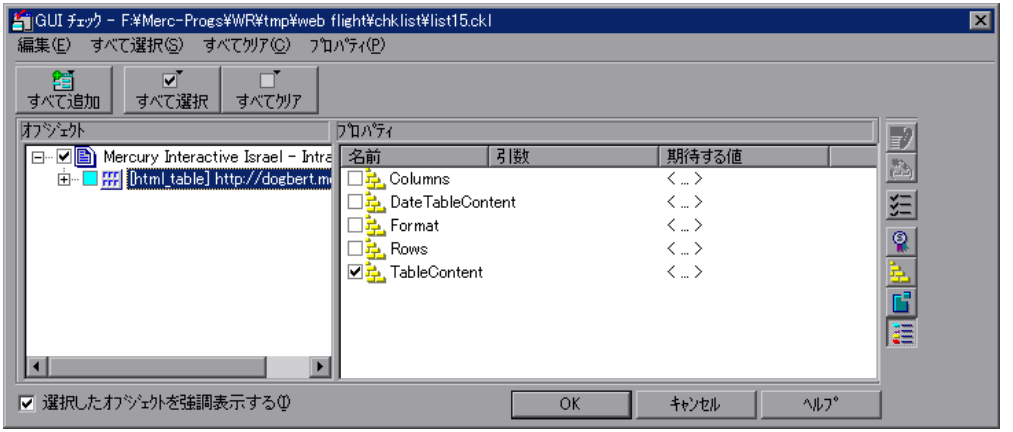
テーブルのテキストの内容を検査するチェックポイントを作成できます。

テーブルの内容を検査するには，次の手順を実行します。

1 [**挿入**] > [**GUI チェックポイント**] > [**オブジェクト / ウィンドウ**] を選択します。

WinRunner のウィンドウが最小化されアイコンになり，マウス・ポインタが指差しポインタに変わり，ヘルプ・ウィンドウが開きます。

2 Web ページ上のテーブル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，オブジェクトが強調表示されます。



3 [**オブジェクト**] 列で，テーブルが選択されていることを確認します。[プロパティ] 列には，検査できるプロパティが表示されます。

4 [**プロパティ**] 列で，[**TableContent**] チェック・ボックスを選択します。

5 プロパティの期待値を編集するには，[**TableContent**] を強調表示させます。

［**期待結果値の編集**］ボタンをクリックするか，［**期待する値**］列の値をダブルクリックし，値を編集します。［**チェックの編集**］ダイアログ・ボックスが開きます。

テーブル内で検査する列または行，使用する検証方式と検証タイプを指定できます。期待データを編集することもできます。このダイアログ・ボックスの使い方の詳細については，209 ページ「テーブル内のセルの検査」を参照してください。

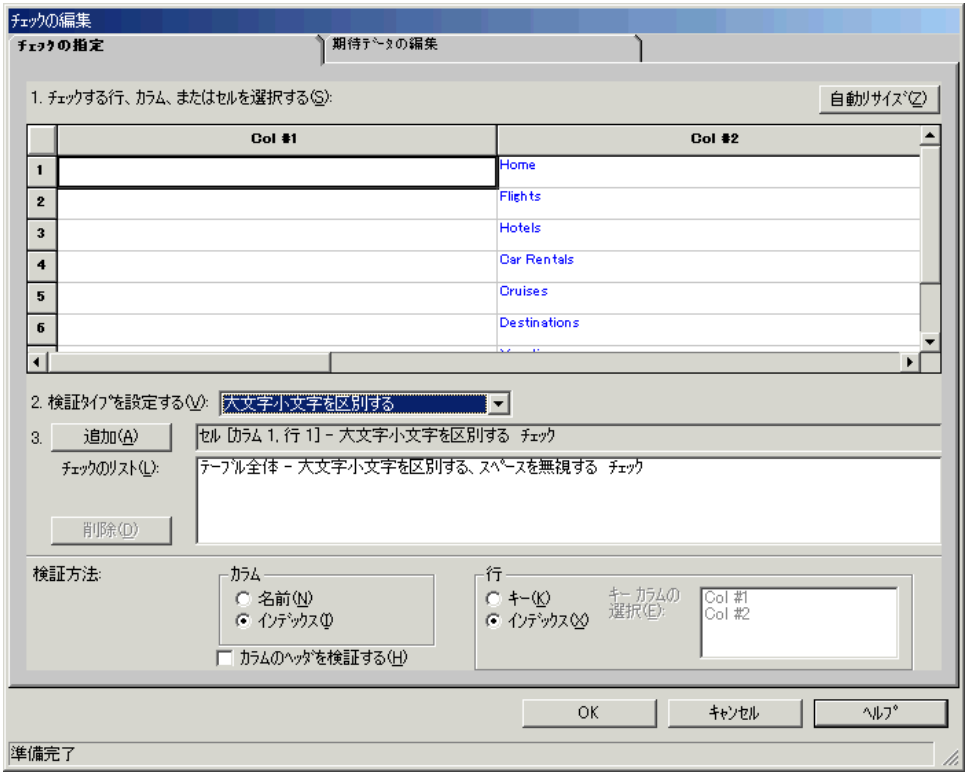
終了したら，［**OK**］をクリックし，［チェックの編集］ダイアログ・ボックスを保存して閉じます。［GUI チェック］ダイアログ・ボックスに戻ります。

6 ［**OK**］をクリックし，［GUI チェック］ダイアログ・ボックスを閉じます。

WinRunner はオブジェクトの情報をキャプチャし，それをテストの期待結果フォルダに格納します。画面が WinRunner のウィンドウに戻り，チェックポイントがテスト・スクリプトに **win_check_gui** ステートメントとして表示されます。**win_check_gui** 関数の詳細については，「**TSL リファレンス**」を参照してください。

テーブル内のセルの検査

［チェックの編集］ダイアログ・ボックスでは，検査するテーブルのセル，使用する検証方式と検証タイプを指定できます。この検査に含まれるテーブル・セルの期待データを編集することもできます。



［**チェックの指定**］タブでは，GUI チェックリストに保存される，以下の情報を指定できます。

- 検査するテーブルのセル
- 検証方式
- 検証タイプ

1 つの列で構成されるテーブルでの検査を作成している場合，［チェックの編集］ダイアログ・ボックスの［チェックの指定］タブの内容は，上図と異なります。詳細については，213 ページ「1 つの列で構成されるテーブルの検証方式の指定」を参照してください。

検査するセルの指定

［**チェックの一覧**］表示枠には，実行されるすべての検査が，検証タイプとともに表示されます。チェックポイントの［チェックの編集］ダイアログ・ボックスを初めて開いたときには，以下に示す標準の検査が表示されます。

- 複数の列で構成されるテーブルの標準の検査は，テーブル全体を対象とし，列の名前と行のインデックスによって検査を行うものです。検査では，大文字と小文字を区別します。
- 1 つの列で構成されるテーブルの標準の検査は，テーブル全体を対象とし，行の位置によって検査を行うものです。検査では，大文字と小文字を区別します。

注：テーブルに同名の列が複数ある場合，重複している列は無視されるため，それらの検査は行われません。このため，列のインデックス・オプションを選択する必要があります。

標準の設定を使用しない場合，実行する検査を指定する前に，標準の検査を削除する必要があります。［**チェックの一覧**］ボックスで［Entire Table - Case Sensitive check］エントリを選択し，［**削除**］ボタンをクリックします。または，［**チェックの一覧**］ボックスでこのエントリをダブルクリックします。強調表示された検査を削除するかどうかを尋ねる WinRunner のメッセージが表示されます。［**はい**］をクリックします。

次に，実行する検査を指定します。選択した各セルに対して，異なる検証タイプを選択できます。このため，セルを選択する前に検証タイプを指定します。詳細については，214 ページ「検証タイプの指定」を参照してください。

内容を検査するセルを強調表示させます。次に，［**追加**］ツールバー・ボタンをクリックし，これらのセルの検査を追加します。また，検査するセルは，次の方法でも指定できます。

- 1 つのセルを検査するには，そのセルをダブルクリックします。
- 行内のすべてのセルを検査するには，行のヘッダをダブルクリックします。
- 列内のすべてのセルを検査するには，列のヘッダをダブルクリックします。
- テーブル全体を検査するには，左上角をダブルクリックします。

検査するセルの説明が，［**チェックの一覧**］ボックスに表示されます。

検証方式の指定

WinRunner によるテーブル内の列または行の識別方法を制御する検証方式を選択できます。検証方式は，テーブル全体に適用されます。検証方式の指定方法は，複数の列で構成されるテーブルと 1 つの列で構成されるテーブルでは異なります。

複数の列で構成されるテーブルの検証方式の指定

列

- **［名前］**：WinRunner は，列の名前に基づいて選択項目を検索します。テーブル内で 列の位置が異なっても，不一致とは見なされません。
- **［インデックス］**：WinRunner は，列のインデックスまたは列の位置に基づいて選択項目を検索します。テーブル内で列の位置が異なる場合は，不一致と見なされます。テーブルに同名の列が複数ある場合，このオプションを選択します。詳細については，210 ページを参照してください。このオプションを選択すると，［**カラムのヘッダを検証**］チェック・ボックスが有効になります。このチェック・ボックスを選択すると，セルと同様に列のヘッダも検査できます。

行

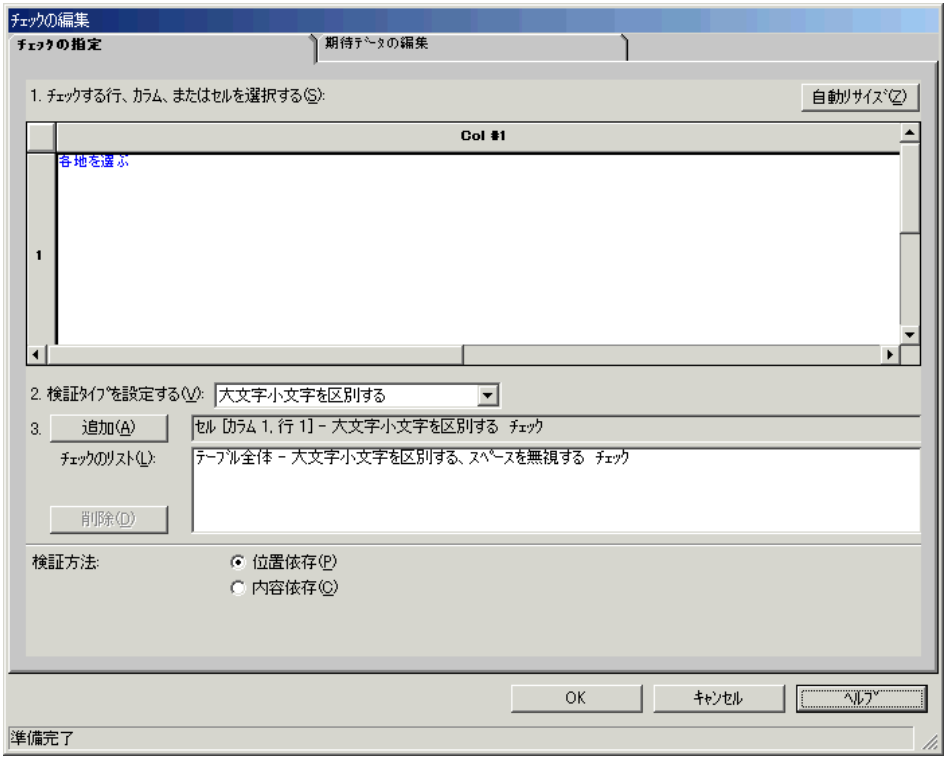
➤ **［キー］**：WinRunner は，［**キーカラムの選択**］リスト・ボックスで指定された列のデータに基づいて，選択された行を検索します。例えば，行の到着時間に基づいて，x ページのテーブルの 2 行目を識別するように指定できます。行の位置が異なっても，不一致とは見なされません。選択されたキーよって，行が一意に識別されない場合，WinRunner は最初に一致する行を検査します。行を一意に識別するために，複数の列をキーとして使用できます。

注：キーとして指定されている 1 つまたは複数の列のセルの値が変更されると，WinRunner は対応する行を識別できず，その行の検査は，「Not Found」エラーが発生して失敗します。この場合，キーとして別の列を選択するか，インデックス検証方式を使用します。

➤ **［インデックス ］**（標準設定）：WinRunner は，行のインデックスまたは行の位置に基づいて，選択項目を検索します。行の位置が異なる場合，不一致と見なされます。

1 つの列で構成されるテーブルの検証方式の指定

1 つの列で構成されるテーブルの［チェックの指定］タブの［検証方式］ボックスは，複数の列で構成されるテーブルのものとは異なります。1 つの列で構成されるテーブルの標準の検査は，テーブル全体を対象とし，行の位置によって検査を行うものです。検査では，大文字と小文字を区別します。



- **［位置依存］：**WinRunner は，列内の項目の位置に基づいて選択項目を検査します。
- **［内容依存］：**WinRunner は，列内の項目の位置を無視し，項目の内容に基づいて選択項目を検査します。

検証タイプの指定

WinRunner は，いくつかの異なる方法でテーブルの内容を検証できます。選択した各セルに対して，異なる検証タイプを選択できます。

- ［**大文字小文字を区別する**］（標準）：WinRunner は，選択項目のテキストの内容を比較します。期待データと実際のデータの間で大文字と小文字，またはテキストが異なるときは，不一致と見なされます。
- ［**大文字小文字を区別しない**］：WinRunner は，選択項目のテキストの内容を比較します。期待データと実際のデータの間でテキストの内容が異なるときだけ，不一致と見なされます。
- ［**数値**］：WinRunner は，数値に基づいて，選択されたデータを評価します。例えば，WinRunner は「2」と「2.00」を同じ数値として認識します。
- ［**数値の範囲**］：WinRunner は，選択されたデータと数値の範囲を比較します。最小値と最大値は，ユーザ指定の実数です。この比較は，実際のテーブル・データが，期待結果とではなく，ユーザ定義の範囲と比較される点が，テキストおよび数値の検証とは異なります。

注：このオプションを選択すると，先頭が数値でない文字列はすべて不一致と見なされます。先頭が「e」の文字列は，数値に変換されます。

- ［**文字小文字を区別する，スペースを無視する**］：WinRunner は，大文字と小文字を区別し，スペースの違いは無視し，内容に基づいてセルのデータを検査します。WinRunner は，大文字と小文字の違いと内容の違いを不一致としてレポートします。
- ［**文字小文字を区別しない，スペースを無視する**］：WinRunner は，大文字と小文字の違いとスペースの違いを無視し，内容に基づいてセルのデータを検査します。WinRunner は，内容の違いだけを不一致としてレポートします。

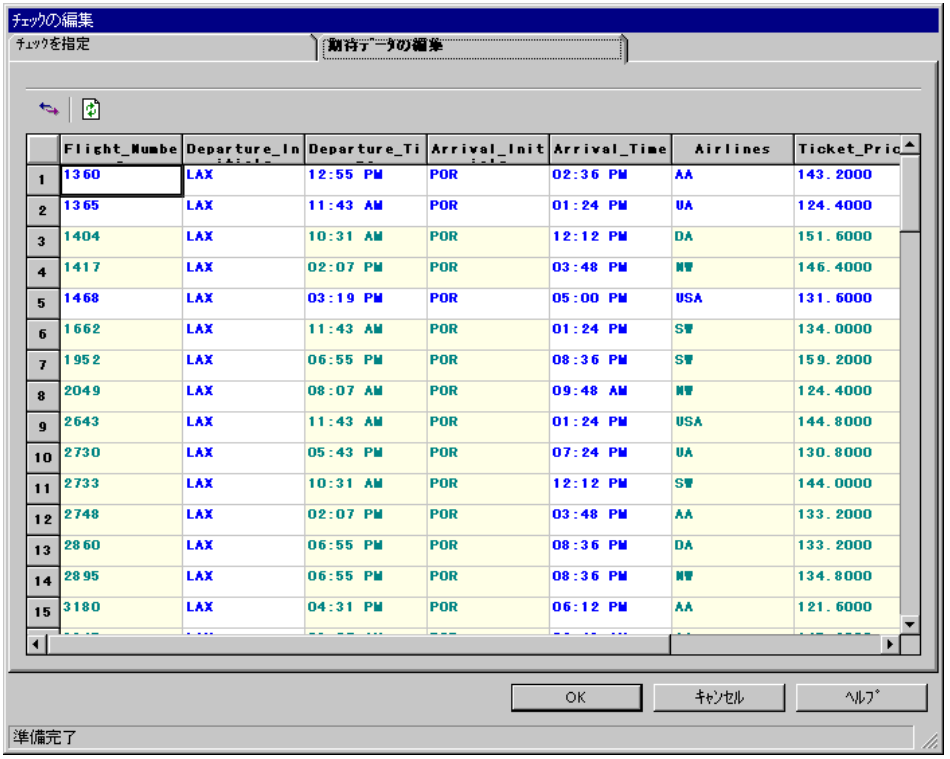
［**OK**］をクリックし，［チェックの編集］ダイアログ・ボックスの両方のタブに対する変更を保存します。ダイアログ・ボックスが閉じ，［GUI チェック］ダイアログ・ボックスに戻ります。

期待データの編集

テーブル内の期待データを編集するには，［**期待データの編集**］タブをクリックします。前に［チェックの指定］タブの変更を保存した場合，［**テーブルを再ロード**］をクリックすると，チェックリストから選択したテーブルの項目を再ロードできます。保存されたデータを再ロードするかどうかを尋ねる WinRunner のメッセージが表示されます。［**はい**］をクリックします。

前に［チェックの指定］タブの変更を保存し，［チェックの編集］ダイアログ・ボックスを再度開いた場合，［期待データの編集］タブのテーブルは，色分けされて表示されます。

検査に含まれるセルは，青い文字と白い背景色で表示されます。検査から除外されるセルは，緑の文字と黄色の背景色で表示されます。



セル内のデータの期待値を編集するには，セルをダブルクリックします。カーソルがセルに表示されます。セルの内容を任意の値に変更します。［**OK**］をクリックし，［チェックの編集］ダイアログ・ボックスの両方のタブに行った変更を保存します。ダイアログ・ボックスが閉じ，［GUI チェック］ダイアログ・ボックスに戻ります。

テキストの検査

テスト・スクリプトでテキスト・チェックポイントを使用して，Web オブジェクトまたは Web ページの領域のテキストを読み取ったり検査したりできます。テストの作成中に，テキストを含むオブジェクトまたはフレームを指します。WebTest はテキストを読み取り，テスト・スクリプトに TSL ステートメントを書き込みます。次に，テキストの内容を確認するために，テスト・スクリプトに単純なプログラミング要素を追加できます。

テキスト・チェックポイントを使って，次のことができます。

- **web_obj_get_text** または **web_frame_get_text** を使って，Web オブジェクトまたはフレームのテキスト文字列またはすべてのテキストを読み取ることができます。
- **web_obj_text_exists** または **web_frame_text_exists** を使って，Web オブジェクトまたはフレームのテキスト文字列の有無を検査できます。

フレームまたはオブジェクトのすべてのテキストの読み取り

web_obj_get_text または **web_frame_get_text** を使って，フレームまたはオブジェクトに表示されるすべてのテキストを読み取ることができます。

フレームまたはオブジェクトのすべてのテキストを読み取るには，次の手順を実行します。

1 ［**挿入**］＞［**テキストの取得**］＞［**オブジェクト / ウィンドウから**］を選択します。

WinRunner が最小化され，マウス・ポインタが指差しポインタになり，ヘルプ・ウィンドウが開きます。

2 Web オブジェクトまたはフレームをクリックします。

WinRunner がオブジェクトのテキストをキャプチャし，**web_obj_get_text** または **web_frame_get_text** ステートメントがテスト・スクリプトに挿入されます。

注：WebTest アドインがロードされていない場合，または Web 以外のオブジェクトが選択された場合，WinRunner はテスト・スクリプトに **win_get_text** または **obj_get_text** ステートメントを生成します。**_get_text** 関数の詳細については，「**TSL リファレンス**」を参照してください。Web 以外のオブジェクトに含まれるテキストの検査の詳細については，第 16 章「テキストの検査」を参照してください。

フレームまたはオブジェクトのテキスト文字列の読み取り

web_obj_get_text または **web_frame_get_text** 関数を使って，フレームまたはオブジェクトのテキスト文字列を読み取ることができます。

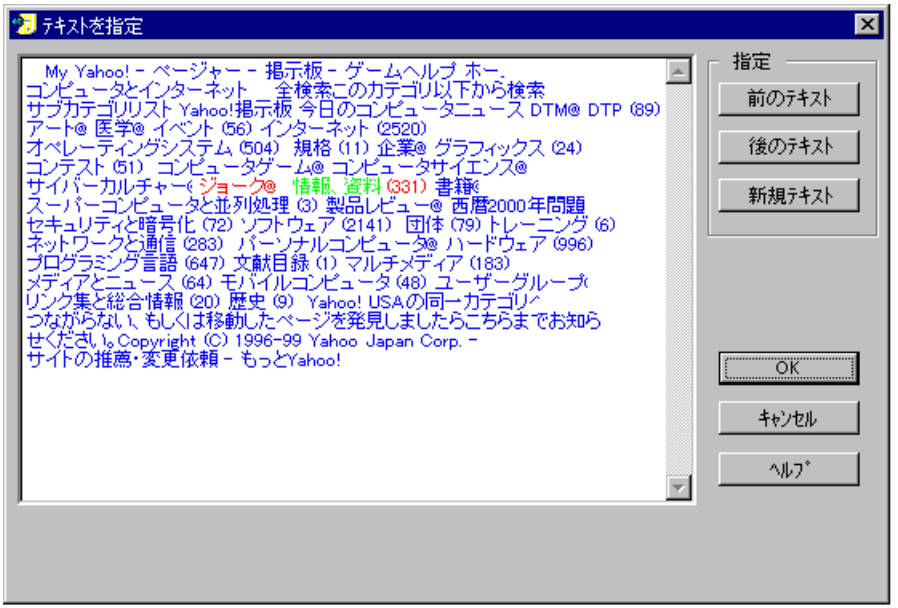
フレームまたはオブジェクトのテキスト文字列を読み取るには，次の手順を実行します。

1 ［**挿入**］＞［**テキストの取得**］＞［**指定範囲から（Web のみ）**］を選択します。

WinRunner が最小化され，マウス・ポインタが指差しポインタになり，ヘルプ・ウィンドウが開きます。

2 読み取るテキスト文字列を強調表示させます。

3 強調表示されたテキスト文字列でマウス・ボタンを右クリックし，その文字列をキャプチャします。［テキストを指定］ダイアログ・ボックスが開きます。



読み取られるテキスト文字列が緑色で下線付きで表示されます。選択したテキストの前後の赤い太字のテキストは，文字列の境界を定義します。

4 選択したテキストを変更できます。

- 強調表示されているテキストを変更するには，新しいテキスト文字列を強調表示し，［**新規テキスト**］をクリックします。新たに選択したテキストが緑色で表示されます。このテキスト文字列の前後にあるテキストが赤で表示されます。
- 選択したテキストの左にある赤いテキスト文字列を変更するには，新しいテキスト文字列を強調表示させ，［**前のテキスト**］をクリックします。
- 選択したテキストの右にある赤いテキスト文字列を変更するには，新しいテキスト文字列を強調表示させ，［**後のテキスト**］をクリックします。

5 ［**OK**］をクリックし，［テキストを指定］ダイアログ・ボックスを閉じます。

WinRunner のウィンドウに戻り，**web_obj_get_text** または **web_frame_get_text** ステートメントがテスト・スクリプトに挿入されます。

フレームまたはオブジェクトのテキスト文字列の有無の検査

web_obj_text_exists または **web_frame_text_exists** を使って，オブジェクトまたはフレームのテキスト文字列の有無を検査できます。

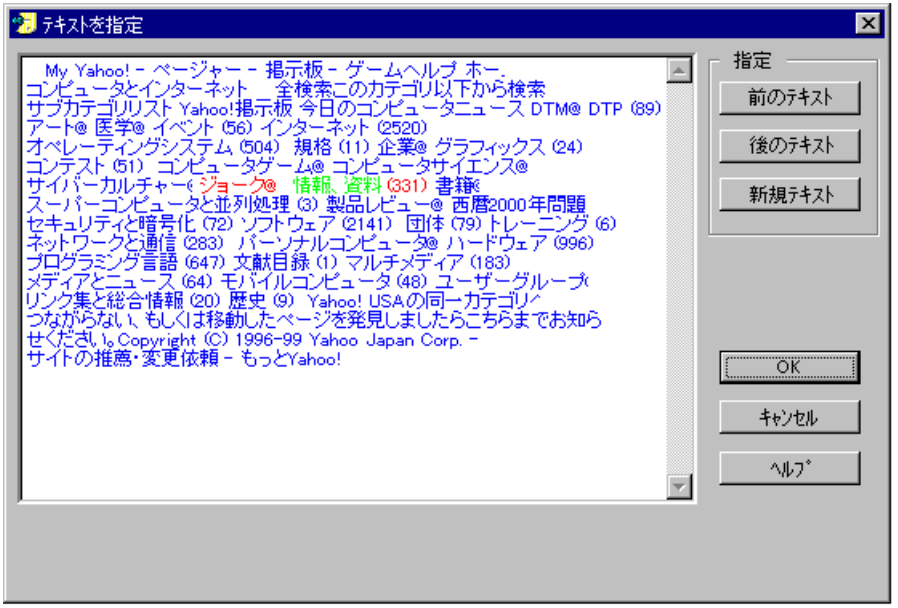
フレームまたはオブジェクトのテキスト文字列の有無を検査するには，次の手順を実行します。

1 ［**挿入**］＞［**テキストの取得**］＞［**Web テキストチェックポイント**］を選択します。

WinRunner が最小化され，マウス・ポインタが指差しポインタになり，ヘルプ・ウィンドウが開きます。

2 検査するテキスト文字列を強調表示させます。

3 強調表示されたテキスト文字列でマウス・ボタンを右クリックし，その文字列をキャプチャします。［テキストを指定］ダイアログ・ボックスが開きます。



検査するテキスト文字列が緑色で下線付きで表示されます。選択したテキストの前後にある赤い太字のテキストは，文字列の境界を定義します。

4 選択したテキストを変更できます。

- ➤ 強調表示されているテキストを変更するには，新しいテキスト文字列を強調表示し，［**新規テキスト**］をクリックします。

  新たに選択したテキストが緑色で表示されます。このテキスト文字列の前後にあるテキストが赤で表示されます。

- ➤ 選択したテキストの左にある赤いテキスト文字列を変更するには，新しいテキスト文字列を強調表示させ，［**前のテキスト**］をクリックします。

- ➤ 選択したテキストの右にある赤いテキスト文字列を変更するには，新しいテキスト文字列を強調表示させ，［**後のテキスト**］をクリックします。

5 ［**OK**］をクリックし，［テキストを指定］ダイアログ・ボックスを閉じます。

WinRunner のウィンドウに戻り，**web_obj_text_exists** または **web_frame_text_exists** ステートメントがテスト・スクリプトに挿入されます。

注：テストの実行後，**check_text** ステートメントが［テスト結果］ウィンドウに表示されます。

第 11 章

ActiveX と Visual Basic のコントロールの使用

WinRunner は，Visual Basic およびその他のアプリケーションの ActiveX コントロール（OLE または OCX コントロールとも呼ばれます）と Visual Basic コントロールをテストするコンテキスト・センシティブなテストをサポートします。

本章では，以下の項目について説明します。

- ActiveX と Visual Basic のコントロールの使い方について
- Visual Basic アプリケーションに対する適切なサポートの選択
- ActiveX と Visual Basic コントロールのプロパティの表示
- ActiveX と Visual Basic のコントロールのプロパティ値の取得と設定
- ActiveX コントロール・メソッドのアクティブ化
- Visual Basic のラベル・コントロールの使用
- ActiveX と Visual Basic のコントロールのサブオブジェクトの検査
- ActiveX コントロールでの TSL テーブル関数の使用

ActiveX と Visual Basic のコントロールの使い方について

多くのアプリケーションには，サードパーティによって開発された ActiveX コントロールと Visual Basic コントロールが含まれています。WinRunner はコントロールに対してコンテキスト・センシティブな操作を記録，実行し，コントロールのプロパティの検査を行います。

WinRunner は，すべての標準（組み込み）Visual Basic および ActiveX コントロールをサポートします。さらに，ActiveX コントロールの中には，ユーザー定義のコンテキスト・センシティブ（状況依存）サポートがあるものもあります。サポートされているコントロールの一覧は，223 ページ「サポートされている ActiveX コントロール」を参照してください。

WinRunner は Visual Basic アプリケーションで，ActiveX コントロールと Visual Basic コントロールの 2 つのタイプのサポートを提供します。次のいずれかを行えます。

- ActiveX コントロールと Visual Basic コントロール用のアドイン・サポートをインストールしてロードします。（non-agent サポートとも言います）
- WinRunner エージェントをアプリケーションにコンパイルして，Visual Basic コントロール用のアドイン・サポートをインストールしてロードします。

適切なアドイン・サポートをロードすれば，WinRunner は ActiveX コントロールと Visual Basic コントロールを認識して，これらを標準の GUI オブジェクトとして扱います。標準の GUI オブジェクトのプロパティを検査するのと同じように ActiveX コントロールと Visual Basic コントロールのプロパティを検査できます。詳細については，第 9 章「GUI オブジェクトの検査」を参照してください。

いつでも，GUI スパイを使って ActiveX コントロールまたは Visual Basic コントロールのプロパティの現在の値を見ることができます。さらに，TSL 関数を使って ActiveX と Visual Basic のコントロールのプロパティの値を取得したり設定したりできます。また，ActiveX コントロール・メソッドをアクティブにすることも可能です。

注：non-agent サポートを使用している場合，ActiveX コントロールを含んでいるアプリケーションを開始する前に WinRunner を起動しなくてはなりません。

WinRunner には，Visual Basic のラベル・コントロールや，テーブルの ActiveX コントロールの内容またはプロパティの検査を行う，専用のサポート機能が組み込みで用意されています。特定の ActiveX コントロールをサポートする TSL テーブル関数の詳細については，239 ページ「ActiveX コントロールでの TSL テーブル関数の使用」を参照してください。ActiveX テーブル・コントロールの内容を検査する方法の詳細については，第 13 章「テーブル内容の検査」を参照してください。

サポートされている ActiveX コントロール

WinRunner はすべての ActiveX コントロールをサポートしています。さらに，ユーザー定義のコンテキスト・センシティブ（状況依存）サポートがあるものもあります。次のリストは，そういった特別なサポートのあるコントロールのサマリです。最新のサポートされているコントロール，および ProgID の詳細やバージョン情報は **WinRunner の Read Me（最初にお読みください）**を参照してください。

ボタン・オブジェクト

ボタン・オブジェクトについては，次の ActiveX コントロールがサポートされています。

- Infragistics (Sheridan) ActiveThreeD Control
  Infragistics (Sheridan) Data CommandButton Control
  Infragistics (Sheridan) OLE Data CommandButton Control

カレンダ・オブジェクト

カレンダ・オブジェクトについては，次の ActiveX コントロールがサポートされています。

- Crescent CSCalendar Control
- Infragistics (Sheridan) MonthView Control

チェック・ボックス・オブジェクト

heck box オブジェクトについては，次の ActiveX コントロールがサポートされています。

- Infragistics (Sheridan) ActiveThreeD Control

コンボ・ボックス・オブジェクト

コンボ・ボックス・オブジェクトについては，次の ActiveX コントロールがサポートされています。

- Infragistics (Sheridan) Data Combo Control
  Infragistics (Sheridan) OLE Data Combo Control

編集オブジェクト

編集オブジェクトについては，次の ActiveX コントロールがサポートされています。

- FarPoint InputPro Control

リスト・オブジェクト

リスト・オブジェクトについては，次の ActiveX コントロールがサポートされています。

- FarPoint ListPro Control
- Microsoft ListView Control

メニューおよびツールバー・オブジェクト

メニューおよびツールバー・オブジェクトについては，次の ActiveX コントロールがサポートされています。

- DataDynamics ActiveBar Control
- Infragistics UltraToolBar Control
- Infragistics (Sheridan) ActiveToolBars Control
  Infragistics (Sheridan) ActiveToolBars Plus Control

ラジオ・ボタン・オブジェクト

ラジオ・ボタン・オブジェクトについては，次の ActiveX コントロールがサポートされています。

- Infragistics (Sheridan) ActiveThreeD Control

ラジオ・グループオブジェクト

ラジオ • グループ・オブジェクトについては，次の ActiveX コントロールがサポートされています。

- Infragistics (Sheridan) Data Option Set Control
  Infragistics (Sheridan) OLE Data Option Set Control

タブ・オブジェクト

タブ・オブジェクトについては，次の ActiveX コントロールがサポートされています。

- Microsoft TabStrip Control
- Infragistics (Sheridan) ActiveTabs Control

テーブル・オブジェクト

ActiveX テーブルについては，次の ActiveX コントロールがサポートされています。

- Apex True DBGrid Control,
  Apex True OLE DBGrid Control
- FarPoint Spread Control
  FarPoint Spread (OLEDB) Control
- Infragistics UltraGrid（テスト実行のみサポート）
- Microsoft DataBound Grid Control
  Microsoft DataGrid Control
  Microsoft FlexGrid Control
  Microsoft Grid Control
  Microsoft Hierarchical FlexGrid Control
- Infragistics (Sheridan) Data Grid Control
  Infragistics (Sheridan) OLE DBGrid
  Infragistics (Sheridan) DBData Option Set
  Infragistics (Sheridan) OLEDBData Option Set
  Infragistics (Sheridan) DBCombo
  Infragistics (Sheridan) OLE DBCombo
  Infragistics (Sheridan) DBData Command
  Infragistics (Sheridan) OLEDBData Command

ツールバー・オブジェクト

ツールバー・オブジェクトについては，次の ActiveX コントロールがサポートされています。

- DataDynamics ActiveBar Control
- Microsoft Toolbar Control
- Infragistics (Sheridan) ActiveToolBars Control
  Infragistics (Sheridan) ActiveToolBars Plus Control

ツリー・オブジェクト

ツリー・オブジェクトについては，次の ActiveX コントロールがサポートされています。

- Microsoft TreeView Control
- Infragistics (Sheridan) ActiveTreeView Control

Visual Basic アプリケーションに対する適切なサポートの選択

WinRunner は，Visual Basic アプリケーションで ActiveX と Visual Basic の 2 つのタイプのコントロールをサポートします。次のいずれかを行えます。

- ActiveX コントロールと Visual Basic コントロール用のアドイン・サポートをインストールしてロードします。
- WinRunner エージェントをアプリケーションにコンパイルして，Visual Basic コントロール用のアドイン・サポートをインストールしてロードします。

ActiveX のアドイン・サポートと Visual Basic コントロールを使用すると，次のようなことができます。

- サポートされている ActiveX コントロールと Visual Basic コントロールに対する操作を含むテストを記録して実行できます。
- 内部のActiveXコントロールとVisual Basicコントロールの名前を一意に識別できます。
- 標準の Visual Basic コントロールのプロパティを検査する GUI チェックポイントを作成できます。
- TSL 関数 **ActiveX_get_info** と **ActiveX_set_info** を, ActiveX コントロールと Visual Basic コントロールに対して使用できます。
- **ActiveX_activate_method** TSL 関数を使用して，ActiveX コントロールでメソッドを有効にできます。

WinRunner エージェントを使用しない ActiveX と Visual Basic アドイン・サポートの使用

WinRunner インストール時に ActiveX と Visual Basic アプリケーションのアドイン・サポートをインストールできます。詳細については，『**WinRunner インストール・ガイド**』を参照してください。WinRunner の各セッションのロード時に，インストール済みのアドインからロードするものを選択できます。詳細については，21 ページ「WinRunner アドインのロード」を参照してください。

WinRunner エージェントと Visual Basic アドイン・サポートの使用

WinRunnerAddIn.Connect という名の WinRunner エージェントをアプリケーションに追加して一緒にコンパイルできます。エージェントは WinRunner の CD-ROM の *vbdev* フォルダに入っています。エージェントのインストール方法およびコンパイル方法については，同じフォルダに入っている *readme.wri* ファイルを参照してください。Visual Basic アプリケーションのアドイン・サポートは，WinRunner のインストール時にインストールできます。詳細については，『**WinRunner インストール・ガイド**』を参照してください。WinRunner の各セッションのロード時に，インストール済みのアドインからロードするものを選択できます。詳細については，21 ページ「WinRunner アドインのロード」を参照してください。

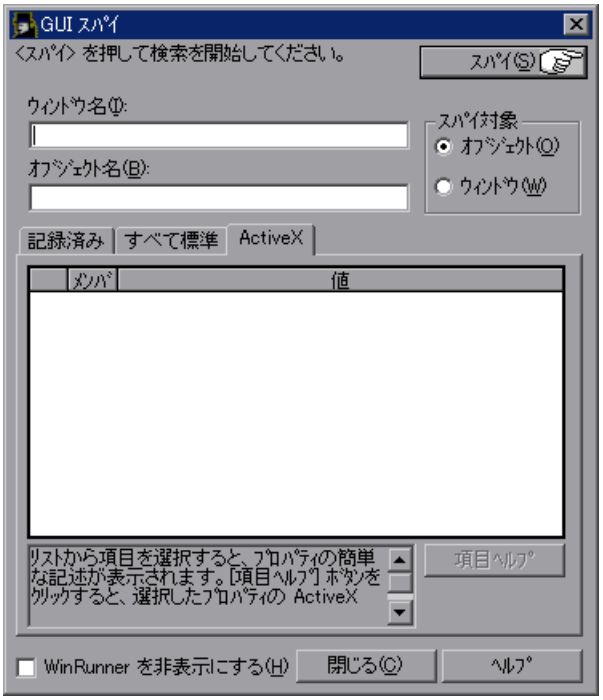
ActiveX と Visual Basic コントロールのプロパティの表示

GUI スパイの［ActiveX］タブを使って，ActiveX コントロールのプロパティやプロパティ値およびメソッドなどを見ることができます。GUI スパイは［ツール］メニューから選択して開きます。ActiveX コントロールに対して GUI スパイを使用するには，WinRunner の起動時に ActiveX アドインをロードしなくてはなりません。また［GUI チェックポイント］ダイアログ・ボックスを使って ActiveX と Visual Basic コントロール・プロパティを表示することもできます。［GUI チェックポイント］ダイアログ・ボックスの使い方については，第 9 章「GUI オブジェクトの検査」を参照してください。

ActiveX または Visual Basic コントロールのプロパティを表示するには，次の手順を実行します。

1 ［**ツール**］＞［**GUI スパイ**］を選択して［**GUI スパイ**］ダイアログ・ボックスを開きます。

2 ［ActiveX］タブをクリックします。



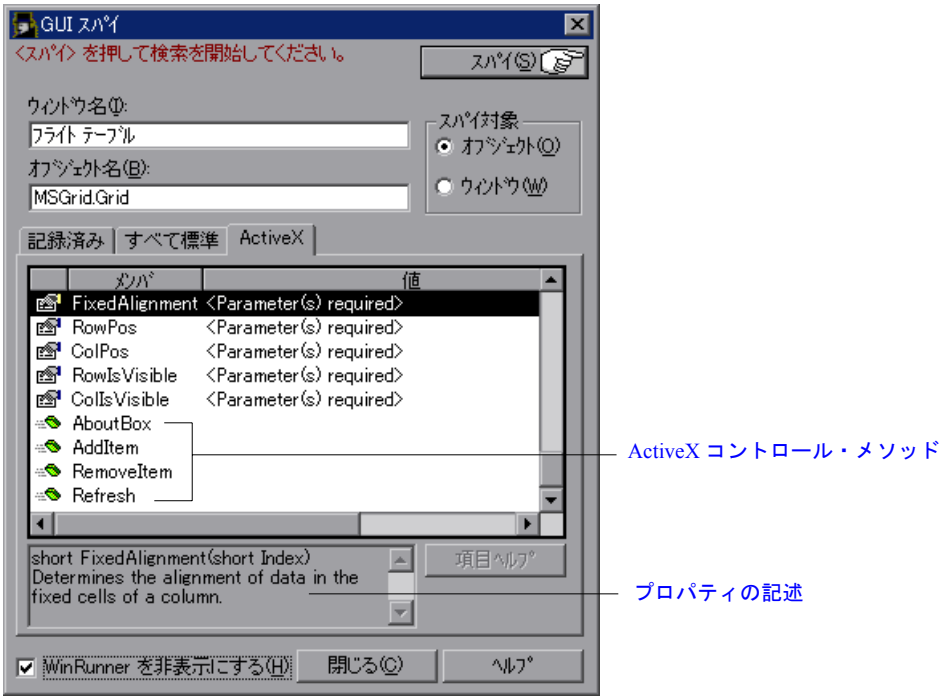
3 ［**スパイ**］をクリックして，ポインタを ActiveX コントロールか Visual Basic コントロールに合わせます。

コントロールが強調表示され，アクティブなウィンドウ名，オブジェクト名，オブジェクトの記述（プロパティとその値）が各フィールドに表示されます。ポインタを他のオブジェクトに移動すると，オブジェクトが強調表示され，［**オブジェクト名**］ボックスにその名前が表示されます。

4 ［GUI スパイ］ダイアログ・ボックスでオブジェクトの記述をキャプチャするには，キャプチャするオブジェクトをポインタで指して，STOP ソフトキーを押します（標準のソフトキーの組み合わせは Ctrl Left + F3）。

次の例では，Visual Basic のサンプルのフライト予約アプリケーションで［フライトテーブル］をポインタで指して STOP ソフトキーを押し，FixedAlignment プ

ロパティを強調表示します。次に示すように，GUI スパイに［**ActiveX**］タブが表示されます。



ヘルプ・ファイルがこの ActiveX コントロールにインストールされていれば，［**項目ヘルプ**］をクリックして，この画面を表示できます。

プロパティを強調表示すると，記述がプロパティに含まれていれば，一番下のグレーの枠に表示されます。

5 ［**閉じる**］をクリックして，GUI スパイを閉じます。

注 : ［値］カラムに「**Object Reference**」が表示されていれば，それはオブジェクトのサブオブジェクトとそれらのプロパティを参照しています。［値］カラムに **<Parameter(s) Required>** が表示されていれば，型の配列か 2 次元配列のどちらかであることを示しています。**ActiveX_get_info** 関数を使って，これらの値を取り出せます。**ActiveX_get_info** 関数の詳細については，230 ページ「ActiveX と Visual Basic のコントロールのプロパティ値の取得と設定」か，「**TSL リファレンス**」を参照してください。

ActiveX と Visual Basic のコントロールのプロパティ値の取得と設定

TSL 関数，**ActiveX_get_info** と **ActiveX_set_info** を使って，アプリケーション内の ActiveX と Visual Basic コントロールのプロパティ値の取得と設定が行えます。これらの関数は，［関数ジェネレータ］を使ってテスト・スクリプトに挿入できます。［関数ジェネレータ］の使い方については，第 34 章「関数の生成」を参照してください。

ヒント : GUI スパイの［ActiveX］タブから ActiveX コントロール・プロパティのプロパティを表示できます。詳細については，227 ページ「ActiveX と Visual Basic コントロールのプロパティの表示」を参照してください。

ActiveX または Visual Basic のコントロールのプロパティ値の取得

ActiveX_get_info 関数を使って，任意の ActiveX または Visual Basic のコントロールのプロパティ値を取得します。プロパティには，パラメータや 1 次元または 2 次元の配列はありません。入れ子にすることもできます。

パラメータのない ActiveX プロパティの構文は次のとおりです。

ActiveX_get_info (ObjectName, PropertyName, OutValue [, IsWindow]);

1 次元配列の ActiveX プロパティの構文は次のとおりです。

ActiveX_get_info (ObjectName, PropertyName (X) , OutValue [, IsWindow]);

2 次元の配列の ActiveX プロパティの構文は次のとおりです。

ActiveX_get_info (ObjectName, PropertyName (X , Y) , OutValue [, IsWindow]);

ObjectName	ActiveX/Visual Basic コントロールの名前。
PropertyName	任意の ActiveX/Visual Basic コントロールのプロパティ。

ヒント：GUI スパイの［ActiveX］タブを使って，ActiveX コントロールのプロパティを表示できます。

OutValue	プロパティの値を格納する出力値。
IsWindow	操作がウィンドウ上で行われるかどうかの表示。操作がウィンドウ上で行われる場合は，TRUE に設定します。

注：

IsWindow パラメータは，この関数を Visual Basic フォームに適用して，そのプロパティまたはサブ・オブジェクトのプロパティを取得する場合のみ使用します。ラベル・コントロールのプロパティを取得するためには，このパラメータを TRUE に設定しておかなくてはなりません。ラベル・コントロールのプロパティについては，234 ページ「Visual Basic のラベル・コントロールの使用」.を参照してください。

入れ子になったプロパティ値を取得するには，ドットで区切られたインデックス付きまたはインデックスなしのプロパティの組み合わせて使用できます。下に例を示します。

```
ActiveX_get_info("Grid", "Cell(10,14).Text", Text);
```

ActiveX または Visual Basic コントロールのプロパティ値の設定

ActiveX_set_info 関数を使って，任意の ActiveX または Visual Basic のコントロールのプロパティ値を設定します。プロパティには，パラメータや 1 次元もしくは 2 次元の配列はありません。プロパティは入れ子にすることもできます。

パラメータを持たない ActiveX プロパティの構文は次のとおりです。

ActiveX_set_info (ObjectName, **PropertyName**, *Value* **[,** *Type*
[, IsWindow]]);

1 次元配列の ActiveX プロパティの構文は次のとおりです。

ActiveX_set_info (ObjectName, PropertyName (X) , Value [, Type
[, IsWindow]]);

2 次元配列の ActiveX プロパティの構文は次のとおりです。

ActiveX_set_info (ObjectName, PropertyName (X , Y) , Value [, Type
[, IsWindow]]);

ObjectName	ActiveX/Visual Basic コントロールの名前。
PropertyName	任意の ActiveX/Visual Basic コントロールのプロパティ。

ヒント：GUI スパイの［ActiveX］タブを使って，ActiveX コントロールのプロパティを表示できます。

Value	プロパティに適用される値。
Type	プロパティに適用される値のタイプ。次のタイプを使用できます。

VT_I2(short)	VT_I4 (long)	VT_R4 (float)
VT_R8 (float double)	VT_DATE (date)	VT_BSTR (string)
VT_ERROR (S code)	VT_BOOL (boolean)	VT_UI1 (unsigned char)

IsWindow 操作がウィンドウ上で行われるかどうかを示すパラメータ。操作がウィンドウ上で行われる場合は，TRUE に設定します。

注：

IsWindow パラメータは，この関数を Visual Basic フォームに適用して，そのプロパティまたはサブ・オブジェクトのプロパティを取得する場合のみ使用します。ラベル・コントロールのプロパティを設定するためには，このパラメータを TRUE に設定しておかなくてはなりません。ラベル・コントロールのプロパティの設定については，234 ページ「Visual Basic のラベル・コントロールの使用」を参照してください。

注：入れ子になったプロパティ値の設定には，ドットで区切られたインデックス付きまたはインデックスなしのプロパティの組み合わせて使用できます。下に例を示します。

```
ActiveX_set_info("Book", "Chapter(7).Page(2).Caption", "SomeText");
```

これらの関数と使用法については，「**TSL リファレンス**」を参照してください。

ActiveX コントロール・メソッドのアクティブ化

ActiveX_activate_method 関数を使って，ActiveX コントロールの ActiveX メソッドを呼び出します。関数ジェネレータを使って，関数をテスト・スクリプトに挿入できます。構文は次のとおりです。

ActiveX_activate_method (*object*, *ActiveX_method*, *return_value*
[, *parameter1*,...,*parameter8* **]);**

この関数の詳細については，「**TSL リファレンス**」を参照してください。

Visual Basic のラベル・コントロールの使用

WinRunner は，Visual Basic アプリケーションの以下のラベル（静的テキスト・コントロール）をサポートします。

- GUI チェックポイントの作成
- ラベル・コントロールの名前の取得
- ラベル・プロパティの取得
- ラベル・プロパティの設定

GUI チェックポイントの作成

Visual Basic のラベル・コントロールに GUI チェックポイントを作成できます。

Visual Basic のラベル・コントロールを検査するには，次の手順を実行します。

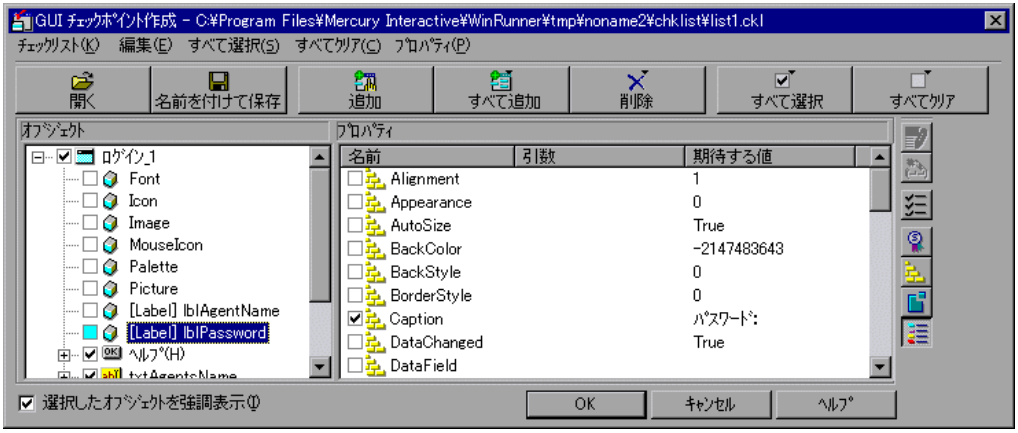
1 ［**挿入**］＞［**GUI チェックポイント**］＞［**複数のオブジェクト**］を選択します。［GUI チェックポイント作成］ダイアログ・ボックスが開きます。

2 ［**追加**］ボタンをクリックして，ラベル・コントロールの含まれる Visual Basic フォームをクリックします。

3 ［すべて追加］ダイアログ・ボックスが開きます。このチェックポイントで他に何も検査していなければ，［オブジェクト］チェックボックスをクリアできます。［**OK**］をクリックします。右クリックしてオブジェクトの追加を終了します。［GUI チェックポイント作成］ダイアログ・ボックスの［オブジェクト］枠にすべてのラベルが VB フォーム・ウィンドウのサブ・オブジェクトとして一覧表示されます。これらのサブ・オブジェクトの名前は **vb_names** で "[Label]" という文字列が先頭に付いています。

4 ［オブジェクト］枠でラベル・コントロールを選択すると，そのプロパティと値が［プロパティ］枠に表示されます。ラベル・コントロールの標準の検査

は，**Caption** プロパティ検査です。他のプロパティ検査を選択して実行することもできます。



ラベル・コントロールの名前の取得

vb_get_label_names 関数を使って，Visual Basic フォームのラベル・コントロールの一覧を取得します。この関数の構文は次のとおりです。

vb_get_label_names (*window*, *name_array*, *count*);

window	Visual Basic フォームの論理名。
name_array	格納配列の名前の出力パラメータ。
count	配列内の要素数の出力パラメータ。

この関数は，指定されたフォーム・ウィンドウのすべてのラベル・コントロールの名前を取得します。これらの名前は配列の添え字として格納されます。

注：配列インデックスの最初の要素は 1 番になります。

この関数の詳細と使用例については，「**TSL リファレンス**」を参照してください。

ラベル・プロパティの取得

ActiveX_get_info 関数を使って、Visual Basic フォームのラベル・コントロールのプロパティ値を取得します。この関数の詳細については，230 ページ「ActiveX と Visual Basic のコントロールのプロパティ値の取得と設定」を参照してください。

ラベル・プロパティの設定

ActiveX_set_info 関数を使って，ラベル・コントロールのプロパティの値を設定します。この関数の詳細については，230 ページ「ActiveX と Visual Basic のコントロールのプロパティ値の取得と設定」を参照してください。

ActiveX と Visual Basic のコントロールのサブオブジェクトの検査

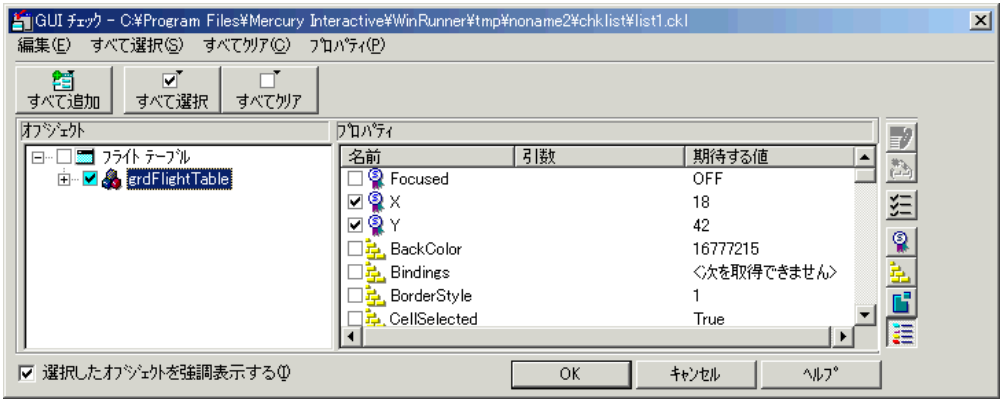
ActiveX と Visual Basic のコントロールには，それぞれのプロパティを含むサブオブジェクトを含めることができます。サブオブジェクトには，例えばフォントがあります。フォントはサブオブジェクトであるため，テスト対象アプリケーションで強調表示することができません。適切なアドイン・サポートをロードすれば，［GUI チェック］ダイアログ・ボックスを使ってサブオブジェクトのプロパティを検査する GUI チェックポイントを作成することができます。GUI チェックポイントの詳細については，第 9 章「GUI オブジェクトの検査」を参照してください。

下の例では，WinRunner が ActiveX テーブル・コントロールの Font サブオブジェクトのプロパティを検査します。下の手順の例では，Visual Basic 用のアドイン・サポートをロードした WinRunner と，サンプルの Visual Basic Flights アプリケーションを使用します。

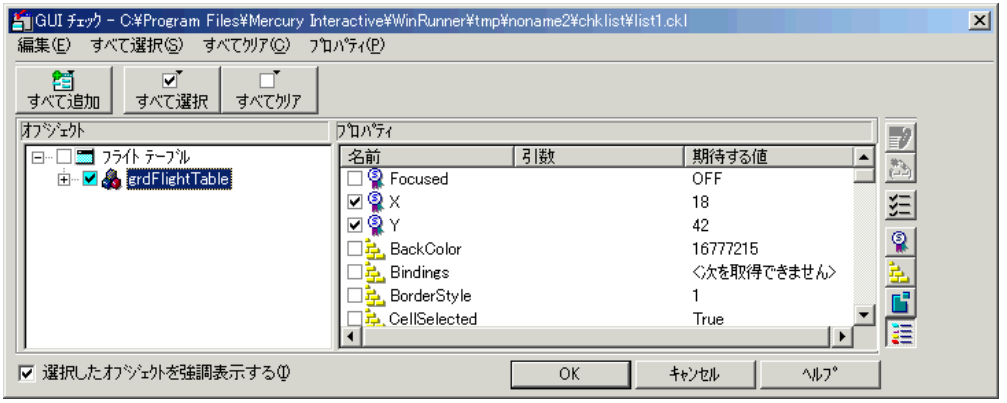
ActiveX または Visual Basic のコントロールのサブオブジェクトを検査するには，次の手順を実行します。

1 ［**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。

2 テストしているアプリケーションのコントロールをダブルクリックします。WinRunner ではコントロールの情報のキャプチャに数秒かかります。［**GUI チェック**］ダイアログ・ボックスが開きます。



3 **［オブジェクト］**表示枠で，オブジェクトのとなりにある拡張記号（+）をクリックして，サブオブジェクトを表示します。そのサブオブジェクトを選択して，ActiveX コントロールのプロパティを表示します。

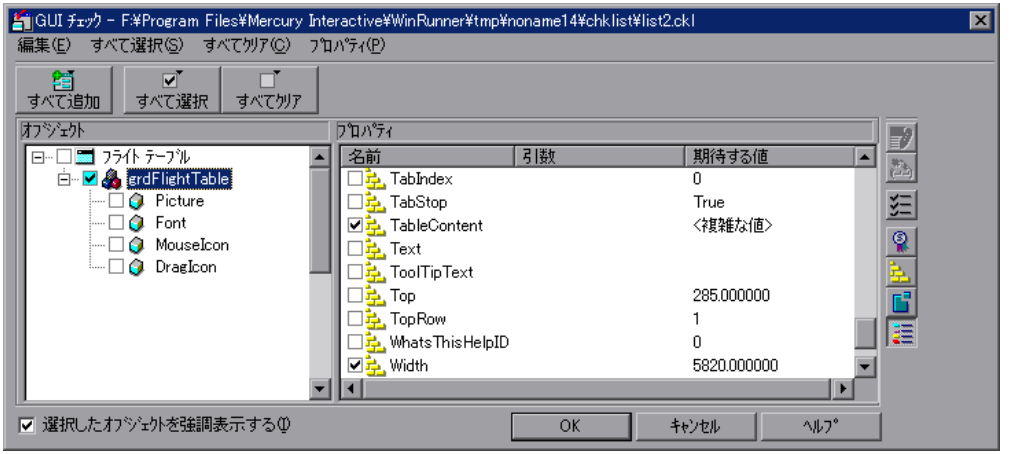


［**オブジェクト**］表示枠にはオブジェクトと，そのサブオブジェクトが表示されます。この例では，サブオブジェクトが「grdFlightTable」オブジェクトの下に表示されています。［**プロパティ**］表示枠には，［**オブジェクト**］表示枠で強調表示されているサブオブジェクトのプロパティが表示されます。各サブオブジェクトには複数の標準のプロパティ検査があることに注目してください。こ

の例では，Font サブオブジェクトのプロパティが表示され，Font サブオブジェクトの Name プロパティが標準の検査として選択されています。

どのテーブルのサブオブジェクトを検査するのか指定します。まず始めに［**オブジェクト**］表示枠でサブオブジェクトを選択し，次に［プロパティ］表示枠で検査対象プロパティを選択します。

この ActiveX コントロールはテーブルなので，標準設定では Height，Width，Table Content のプロパティの検査が選択されています。これらの検査を実行したくない場合は，該当するチェック・ボックスをクリアします。テーブルの内容の検査については，第 13 章「テーブル内容の検査」を参照してください。



4 ［**OK**］をクリックして，ダイアログ・ボックスを閉じます。

obj_check_gui ステートメントがテスト・スクリプトに挿入されます。**obj_check_gui** 関数の詳細については，第 9 章「GUI オブジェクトの検査」または「**TSL リファレンス**」を参照してください。

ActiveX コントロールでの TSL テーブル関数の使用

TSL の **tbl_** 関数は，いくつかの ActiveX コントロールの多くに対して使うことができます。WinRunner には，ActiveX コントロールに対するサポート機能と下の表の関数が組み込まれています。それぞれの関数の詳細，使用例，および ActiveX コントロールでサポートされているバージョンについては，「**TSL リファレンス**」を参照してください。

	Data Bound Grid コントロール	FarPoint Spreadsheet コントロール	MicroHelp MH3d List コントロール	Microsoft Grid コントロール	Sheridan Data Grid コントロール	True DBGrid コントロール
tbl_activate_cell	✔	✔	✔	✔	✔	✔
tbl_activate_header	✔	✔	✔	✔	✔	✔
tbl_get_cell_data	✔	✔	✔	✔	✔	✔
tbl_get_cols_count	✔	✔	✔	✔	✔	✔
tbl_get_column_name	✔	✔	✔	✔	✔	✔
tbl_get_rows_count		✔	✔	✔	✔	✔
tbl_get_selected_cell	✔	✔	✔	✔	✔	✔
tbl_get_selected_row	✔	✔	✔		✔	✔
tbl_select_col_header	✔	✔	✔	✔	✔	✔
tbl_set_cell_data	✔	✔	✔	✔	✔	✔
tbl_set_selected_cell	✔	✔	✔	✔	✔	✔
tbl_set_selected_row	✔	✔	✔	✔		✔

第 12 章

PowerBuilder のアプリケーションの検査

WinRunner に PowerBuilder アプリケーションへのサポートを追加して作業する場合には，GUI チェックポイントを作成して，アプリケーション内の PowerBuilder オブジェクトを検査できます。

本章では，次の項目について説明します。

- PowerBuilder のアプリケーションの検査について
- ドロップダウン・オブジェクトのプロパティ検査
- DataWindow のプロパティの検査
- DataWindow 内のオブジェクトのプロパティの検査
- DataWindow 内の計算カラムの処理

PowerBuilder のアプリケーションの検査について

GUI チェックポイントを使って，アプリケーションの PowerBuilder オブジェクトの「**プロパティ**」を検査できます。これらのプロパティを検査する際，その標準の GUI プロパティはもちろん PowerBuilder オブジェクトの「**内容**」も検査できます。この章では，次の PowerBuilder オブジェクトのプロパティの検査手順について説明します。

- DropDown オブジェクト
- DataWindow
- DataWindow のカラム
- DataWindow のテキスト
- DataWindow のレポート
- DataWindow のグラフ

➤ DataWindow の計算カラム

ドロップダウン・オブジェクトのプロパティ検査

DropDown リストあるいは DropDown DataWindow のプロパティ（内容を含む）を検査する GUI チェックポイントを作成できます。通常の DataWindow の検査対象のプロパティと同じプロパティ（内容を含む）を DropDown DataWindow 内で検査できます。ただし，DropDown オブジェクトに GUI チェックポイントを作成する前に，まずテスト・スクリプト内に **tbl_set_selected_cell** ステートメントを作成しなければなりません。［GUI チェックポイント－オブジェクト / ウィンドウ］ソフトキーを使って，記録実行中に GUI チェックポイントを作成します。テーブルに対して作る場合と同じように，DropDown オブジェクトの内容を検査する GUI チェックポイントを作成します。テーブルの検査の詳細については，第 13 章「テーブル内容の検査」を参照してください。

標準の検査による DropDown オブジェクトのプロパティ検査

DropDown オブジェクトの標準の検査を実行する GUI チェックポイントを作成できます。DropDown オブジェクトの標準の検査には，オブジェクト全体の内容に対する大文字と小文字を区別する検査が含まれます。WinRunner はカラム名と行のインデックス番号を使ってオブジェクトのセルを特定して検査します。

さらに，実行する検査を指定して DropDown オブジェクトを検査できます。詳細については，「実行する検査の指定時の DropDown オブジェクトのプロパティ検査」を参照してください。

DropDown オブジェクトのプロパティを標準の検査で検査するには，次の手順を実行します。

1 ［**テスト**］＞［**記録 - コンテキストセンシティブ**］を選択するか，［**記録 - コンテキストセンシティブ**］ボタンをクリックします。

2 DropDown オブジェクト内でクリックして，**tbl_set_selected_cell** ステートメントをテスト・スクリプト内に記録します。

3 記録中に［GUI チェックポイント－オブジェクト / ウィンドウ］ソフトキーを押します。

4 DropDown オブジェクト内で 1 回クリックします。

WinRunner は GUI 情報をキャプチャして，テストの期待結果フォルダに格納します。［WinRunner］ウィンドウが再び表示され，**obj_check_gui** ステートメントがテスト・スクリプトに挿入されます。**obj_check_gui** 関数の詳細については，「**TSL リファレンス**」を参照してください。

実行する検査の指定時の DropDown オブジェクトのプロパティ検査

DropDown オブジェクトで実行する検査を指定する GUI チェックポイントを作成できます。GUI チェックポイントの作成中に DropDown オブジェクト内でダブルクリックすると，［GUI チェック］ダイアログ・ボックスが開きます。例えば，DropDownListBox の検査中に，［GUI チェック］ダイアログ・ボックスで **DropDownListBoxContent** プロパティの検査をダブルクリックして，［チェックの編集］ダイアログ・ボックスを開きます。［チェックの編集］ダイアログ・ボックスで，オブジェクト内容の検査範囲を設定し，検証の種類と方法を選択して，DataWindow の内容の期待値を編集できます。

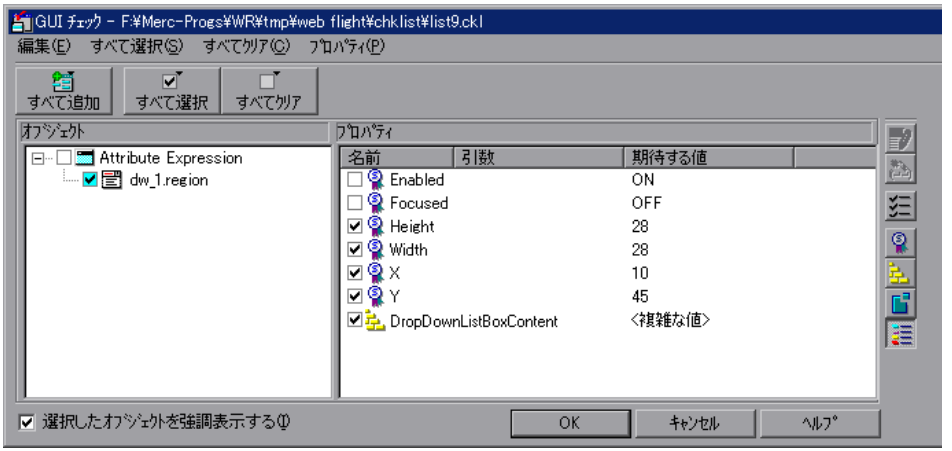
実行する検査の選択中に DropDown オブジェクトのプロパティを検査するには，次の手順を実行します。

1 ［**テスト**］＞［**記録 - コンテキストセンシティブ**］を選択するか，［**記録 - コンテキストセンシティブ**］ボタンをクリックします。

2 DropDown オブジェクト内でクリックして，**tbl_set_selected_cell** ステートメントをテスト・スクリプト内に記録します。

3 記録実行中に［GUI チェックポイント－オブジェクト / ウィンドウ］ソフトキーを押します。

4 DropDown オブジェクト内をダブルクリックします。［GUI チェック］ダイアログ・ボックスが開きます。



上の例は DropDown リストの［GUI チェック］ダイアログ・ボックスを表示します。DropDown DataWindow のための［GUI チェック］ダイアログ・ボックスは DataWindow のダイアログ・ボックスと似ています。

5 ［**プロパティ**］表示枠内で，DropDownListBoxContent 検査を選択して，［期待結果値を編集］ボタンをクリックします。あるいは［期待する値］カラム内の＜複雑な値＞エントリをダブルクリックします。

［チェックの編集］ダイアログ・ボックスが開きます。

6 実行する検査を選択したり，期待データを編集したりできます。このダイアログ・ボックスの使用法の詳細については，257 ページ「［チェックの編集］ダイアログ・ボックスについて」を参照してください。

7 検査が終了したら，［**OK**］をクリックして変更を保存します。［チェックの編集］ダイアログ・ボックスを閉じて，［GUI チェック］ダイアログ・ボックスに戻ります。

8 ［**OK**］をクリックして，［GUI チェック］ダイアログ・ボックスを閉じます。

WinRunner は GUI 情報をキャプチャして，テストの期待結果フォルダに格納します。［WinRunner］ウィンドウが再び表示され，**obj_check_gui** ステートメントがテスト・スクリプトに挿入されます。**obj_check_gui** 関数の詳細については，「**TSL リファレンス**」を参照してください。

注：内容の検査実行中に検査対象オブジェクトを追加したい時には，［**挿入**］＞［**GUI チェックポイント**］＞［**複数のオブジェクト**］コマンド（［**挿入**］＞［**GUI チェックポイント**］＞［**オブジェクト / ウィンドウ**］コマンドの代わり）を使用して，**win_check_gui** ステートメントをテスト・スクリプトに挿入してください。DropDown オブジェクトの標準 GUI プロパティの検査情報の詳細については，第 9 章「GUI オブジェクトの検査」を参照してください。

DataWindow のプロパティの検査

GUI チェックポイントを作成して，DataWindow のプロパティを検査できます。検査できるプロパティの中の 1 つに DataWindow の内容を検査する **DWTableContent** があります。テーブルに対して作る場合と同じように，DataWindow に対して内容検査を作成します。テーブル内容の検査の詳細については，第 13 章「テーブル内容の検査」を参照してください。

標準の検査による DataWindow のプロパティの検査

GUI チェックポイントを作ることで，標準の検査で DataWindow のプロパティを検査できます。DataWindow の様々なタイプに応じて，異なった標準の検査があります。

DataWindow のプロパティを標準の検査で検査するには，次の手順を実行します。

1 ［**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。

2 DataWindow 内で 1 回クリックします。

WinRunner は GUI 情報をキャプチャして，テストの期待結果フォルダに格納します。［WinRunner］ウィンドウが再び表示され，**obj_check_gui** ステートメントがテスト・スクリプトに挿入されます。**obj_check_gui** 関数の詳細については，「**TSL リファレンス**」を参照してください。

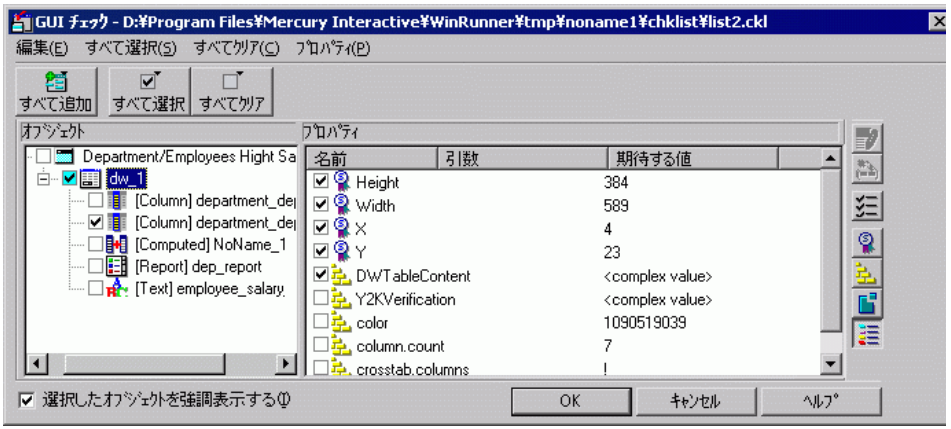
実行する検査の指定中の DataWindow のプロパティの検査

GUI チェックポイントを作成することで，実行する検査の選択中に，DataWindow のプロパティを検査できます。

実行する検査の指定中に DataWindow のプロパティを検査するには，次の手順を実行します。

1 ［**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。

2 DataWindow 内でダブルクリックします。［GUI チェック］ダイアログ・ボックスが開きます。



DataWindow 内のオブジェクトのプロパティがダイアログ・ボックス内に表示されていることに注目してください。WinRunner はそれらのオブジェクトの検査を実行できます。詳細については，「DataWindow 内のオブジェクトのプロパティの検査」を参照してください。

3 **DWTableContent** 検査を選択して，［**期待結果値を編集**］ボタンをクリックするか，［**期待する値**］**カラム内の**「< 複雑な値 >」エントリをダブルクリックします。［チェックの編集］ダイアログ・ボックスが開きます。

4 実行する検査を選択したり，期待値を編集したりできます。このダイアログ・ボックスの詳細については，257 ページ「［チェックの編集］ダイアログ・ボックスについて」を参照してください。

5 検査が終了したら，［**OK**］をクリックして変更を保存します。［チェックの編集］ダイアログ・ボックスを閉じて，［GUI チェック］ダイアログ・ボックスに戻ります。

6 ［**OK**］をクリックして，［GUI チェック］ダイアログ・ボックスを閉じます。

WinRunner は GUI 情報をキャプチャして，テストの期待結果フォルダに格納します。［WinRunner］ウィンドウが再び表示され，**obj_check_gui** ステートメントがテスト・スクリプトに挿入されます。**obj_check_gui** 関数の詳細については，「**TSL リファレンス**」を参照してください。

DataWindow 内のオブジェクトのプロパティの検査

以下の DataWindow オブジェクトのプロパティを検査する GUI チェックポイントを作成できます。

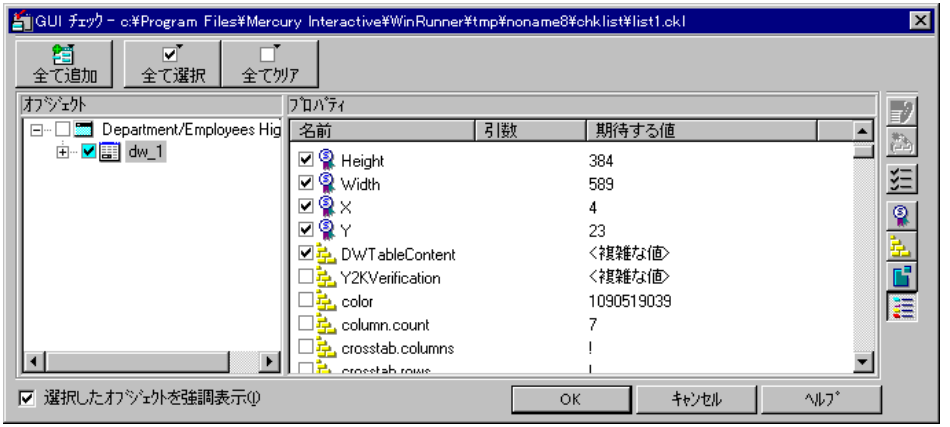
- DataWindow
- DataWindow のカラム
- DataWindow のテキスト
- DataWindow のレポート
- DataWindow のグラフ
- DataWindow の計算カラム

DataWindow のオブジェクトはテスト中のアプリケーション内では強調表示できません。GUI チェックポイントを作ることで，［GUI チェック］ダイアログ・ボックスを使用して，DataWindow 内のオブジェクトのプロパティを検査できます。GUI チェックポイントの詳細については，第 9 章「GUI オブジェクトの検査」を参照してください。

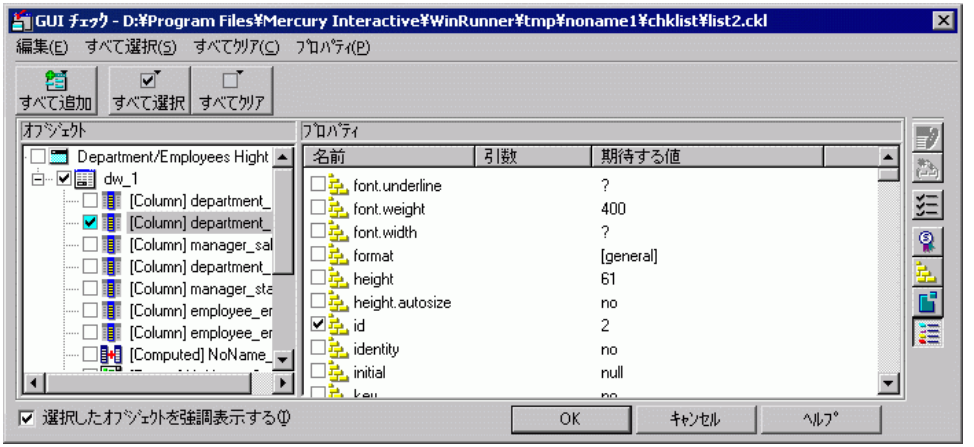
DataWindow 内オブジェクトのプロパティを検査するには，次の手順を実行します。

1 ［**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。

2 テスト中のアプリケーション内の DataWindow をダブルクリックします。WinRunner が DataWindow の情報をキャプチャするのには数秒かかる場合があります。［GUI チェック］ダイアログ・ボックスが開きます。



3 **［オブジェクト］**表示枠で DataWindow 横の拡張記号「+」をクリックしてオブジェクトを表示し，オブジェクトを 1 つ選択してそのプロパティを表示します。



［**オブジェクト**］表示枠は DataWindow とそのオブジェクトを表示します。［**プロパティ**］表示枠は［**オブジェクト**］表示枠内で強調表示されている DataWindow のオブジェクトのプロパティを表示します。オブジェクトはカラム，計算カラム，テキスト，グラフ，およびレポートの場合があります。各オ

ブジェクトには 1 つ以上の標準プロパティ検査が含まれることに注意してください。

DataWindow の中のどのオブジェクトを検査するか指定します。最初に［オブジェクト］表示枠内でオブジェクトを選びます。次に［プロパティ］表示枠内で検査するプロパティを選択します。

4 ［**OK**］をクリックして，ダイアログ・ボックスを閉じます。

obj_check_gui ステートメントがテスト・スクリプトに挿入されます。**obj_check_gui** 関数の詳細については，第 9 章「GUI オブジェクトの検査」か「**TSL リファレンス**」を参照してください。

注：DataWindow 内のオブジェクトが［GUI チェックポイント］ダイアログ・ボックスの［オブジェクト］表示枠内で「Noname」と表示されている場合には，オブジェクトには内部名はありません。

DataWindow 内の計算カラムの処理

DataWindow の詳細バンド内に計算カラムが置かれている場合には，WinRunner はそれらのテストと記録を実行できます。WinRunner は **tbl_get_selected_cell**, **tbl_activate_cell**，および **tbl_get_cell_data** という TSL 関数を使って，計算カラムのテストと記録を実行します。これらの TSL 関数の使用の詳細については，「**TSL リファレンス**」を参照してください。

また，WinRunner は **tbl_get_cell_data** という TSL 関数を使用することで，DataWindow の詳細バンドにはない計算カラムのデータも取得できます。この TSL 関数の詳細については，「**TSL リファレンス**」を参照してください。

DataWindow の詳細バンド内の計算カラムの内容を検査するには，**DWComputedContent** のプロパティ検査を使用します。

計算カラムはデータベースに含まれないので，インデックスでは参照できません。したがって，計算カラムは名前で参照しなくてはなりません。

➤ 計算カラムを選択する操作を記録します。カラムの名前がテスト・スクリプトに挿入された **tbl_selected_cell** ステートメントの中に表示されます。

➤ 計算カラムが表示されるDataWindowを対象にGUIチェックポイントを実行します。計算カラムの名前は親 DataWindow の名前の下の［オブジェクト］表示枠内に表示されます。

第 13 章

テーブル内容の検査

Visual Basic，PowerBuilder，Delphi，Oracle などのアプリケーション開発環境のサポートを追加した WinRunner では，GUI チェックポイントを作成してアプリケーションのテーブルの内容を検査できます。

本章では，以下の項目について説明します。

- テーブル内容の検査について
- 標準の検査によるテーブル内容の検査
- 検査を指定してのテーブル内容の検査
- ［チェックの編集］ダイアログ・ボックスについて

テーブル内容の検査について

テーブルは，一般に Visual Basic，PowerBuilder，Delphi，Oracle など，特別な開発環境の一部です。これらのツールキットを使用すると，データベース情報をグリッドに表示できます。本章で説明するテーブルの検査を実行するには，該当する開発環境のアドイン・サポート機能をインストールしてロードしなければなりません。WinRunner のインストール時には，Visual Basic アプリケーションまたは PowerBuilder アプリケーションのサポート機能をインストールするかどうか選択できます。また，Delphi や Oracle など，その他の開発環境のサポート機能も個々にインストールできます。［アドインマネージャ］ダイアログ・ボックスで，WinRunner の各セッションにロードするアドイン・サポート機能を選択できます。［アドインマネージャ］ダイアログ・ボックスについては，第 2 章「WinRunner の概要」を参照してください。［アドインマネージャ］ダイアログ・ボックスを表示する方法については，第 22 章「グローバル・テスト・オプションの設定」を参照してください。

これらのツール用の WinRunner サポート機能をインストールすると，テスト・スクリプトにテーブルの内容を検査する GUI チェックポイントを追加できます。

テーブルをクリックして，WinRunner に検査させたいプロパティを選ぶことによって，テーブル内容に GUI チェックポイントを作成できます。WinRunner で推奨されている標準のプロパティを検査することも，検査するプロパティを指定することもできます。検査されるテーブルとプロパティに関する情報は「**チェックリスト**」に保存されます。次に WinRunner はテーブル・プロパティの現在の値をキャプチャし，この情報を「**期待結果**」として保存します。GUI チェックポイントはテスト・スクリプトに自動的に挿入されます。このチェックポイントは **obj_check_gui** または **win_check_gui** ステートメントとしてテスト・スクリプト中に記録されます。GUI チェックポイントとチェックリストの詳細については，第 9 章「GUI オブジェクトの検査」を参照してください。

テストを実行すると，WinRunner はテーブル内のプロパティの現在の状態と期待結果を比較します。期待結果と現在の結果が一致しないと，GUI チェックポイントは失敗します。チェックポイントの結果は，WinRunner の［テスト結果］ウィンドウで見ることができます。詳細については，第 20 章「テスト結果の分析」を参照してください。

検査した GUI オブジェクトのうち，まだ GUI マップの中にないものは，自動的に仮 GUI マップ・ファイルに追加されます。詳細については，第 3 章「WinRunner の GUI オブジェクトの識別方法」を参照してください。

本章では，テーブルの内容を検査するための手順を順を追って説明します。

PowerBuilder の DropDown リストや DataWindow の内容を検査する GUI チェックポイントを作成することもできます。DropDown リストは，単一カラムのテーブルと同じように検査できます。DataWindow は，複数カラムのテーブルと同じように検査できます。詳細については，第 12 章「PowerBuilder のアプリケーションの検査」を参照してください。

テーブル内容を検査するだけでなく，テーブルの他のプロパティも検査できます。テーブルに ActiveX プロパティが含まれている場合，これらを GUI チェックポイントで検査できます。WinRunner には，テーブルである ActiveX コントロール用のビルトイン・サポートも用意されています。詳細については，第 11 章「ActiveX と Visual Basic のコントロールの使用」を参照してください。また，テーブルの標準 GUI プロパティも GUI チェックポイントで検査できます。詳細については，第 9 章「GUI オブジェクトの検査」を参照してください。

標準の検査によるテーブル内容の検査

テーブルの内容に対して標準の検査を実行する GUI チェックポイントを作成できます。

標準の検査は，テーブル全体の内容に対して大文字と小文字を区別した検査を行います。WinRunner は，カラム名と行のインデックス番号により，テーブルのセルの場所を特定します。

実行する検査を指定したテーブルの内容に対しても検査を実行できます。詳細については，254 ページ「検査を指定してのテーブル内容の検査」を参照してください。

標準の検査でテーブルの内容を検査するには，次の手順を実行します。

1 ［**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で［**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。

2 テスト中のアプリケーションでテーブルをクリックします。

WinRunner は，テーブルに関する情報をキャプチャするのに数秒かかる場合があります。

obj_check_gui ステートメントが，テスト・スクリプトに挿入されます。**obj_check_gui** 関数の詳細については，「**TSL リファレンス**」を参照してください。

注：テーブル内容に対して検査を行っている間に他のテーブル・オブジェクトのプロパティを検査したい場合は，［**挿入**］＞［**GUI チェックポイント**］＞［**複数のオブジェクト**］コマンド（［**挿入**］＞［**GUI チェックポイント**］＞［**オブジェクト / ウィンドウ**］コマンドではなく）を使用します。このコマンドは，**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。テーブルの標準 GUI プロパティの検査については，第 9 章「GUI オブジェクトの検査」を参照してください。テーブルの ActiveX コントロール・プロパティの検査については，第 11 章「ActiveX と Visual Basic のコントロールの使用」を参照してください。

検査を指定してのテーブル内容の検査

GUI チェックポイントを使用して，テーブルの内容に対して実行する検査を指定できます。チェックを指定したテーブル内容に GUI チェックポイントを作成するには，GUI チェックポイント・コマンドを選択して，テーブル内でダブルクリックします。

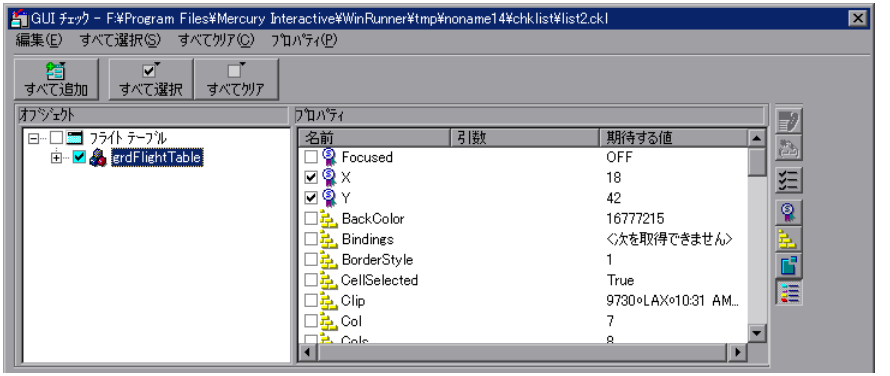
下に示す例では，Visual Basic のアドイン・サポートがインストールされている WinRunner と，サンプルの Visual Basic Flights アプリケーションを使用します。

実行する検査を指定したテーブルの内容を検査するには，次の手順を実行します。

1 ［**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で［**オブジェクト / ウィンドウの GUI チェックポイント**］をクリックします。

2 テスト中のアプリケーションでテーブルをダブルクリックします。

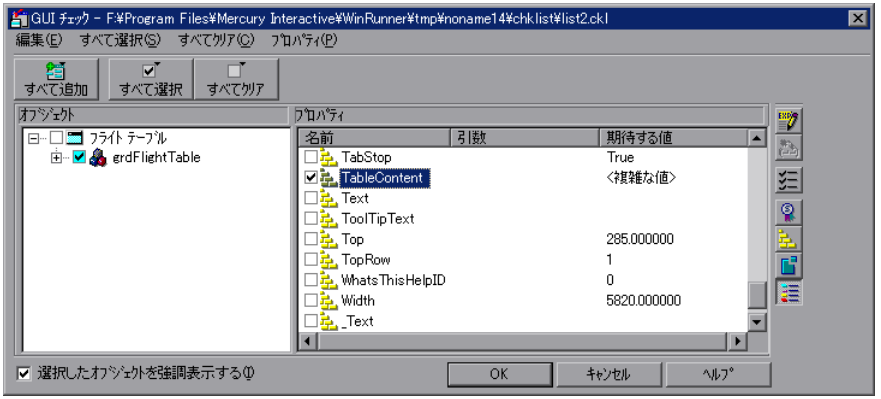
WinRunner は，テーブルに関する情報をキャプチャするのに数秒かかる場合があります。その後，［**GUI チェック**］ダイアログ・ボックスが開きます。



このダイアログ・ボックスには，テーブルの一意のテーブル・プロパティが非標準オブジェクトとして表示されます。

3 **TableContent** プロパティ検査が［**プロパティ**］表示枠に表示されるまで，ダイアログ・ボックスで下にスクロールするか，ダイアログ・ボックスの大きさを変更します。

テーブル内容のプロパティ検査は，使用するツールキットによって，**TableContent** 以外の名前が付けられている場合があります。



4 **TableContent**（またはこれに該当する）プロパティ検査を選択し，［**期待結果値を編集**］ボタンをクリックします。このプロパティ検査の［期待する値］カラムには，**< 複雑な値 >** と表示されます。これはこの検査の期待値が複雑すぎてこのカラムに表示できないからです。

［**チェックの編集**］ダイアログ・ボックスが開きます。

5 検査するセルを選択したり，期待データを編集したりできます。このダイアログ・ボックスの使用法については，257 ページ「［チェックの編集］ダイアログ・ボックスについて」を参照してください。

6 終了したら，［OK］をクリックして変更を保存します。［チェックの編集］ダイアログ・ボックスが閉じて，［GUI チェック］ダイアログ・ボックスに戻ります。

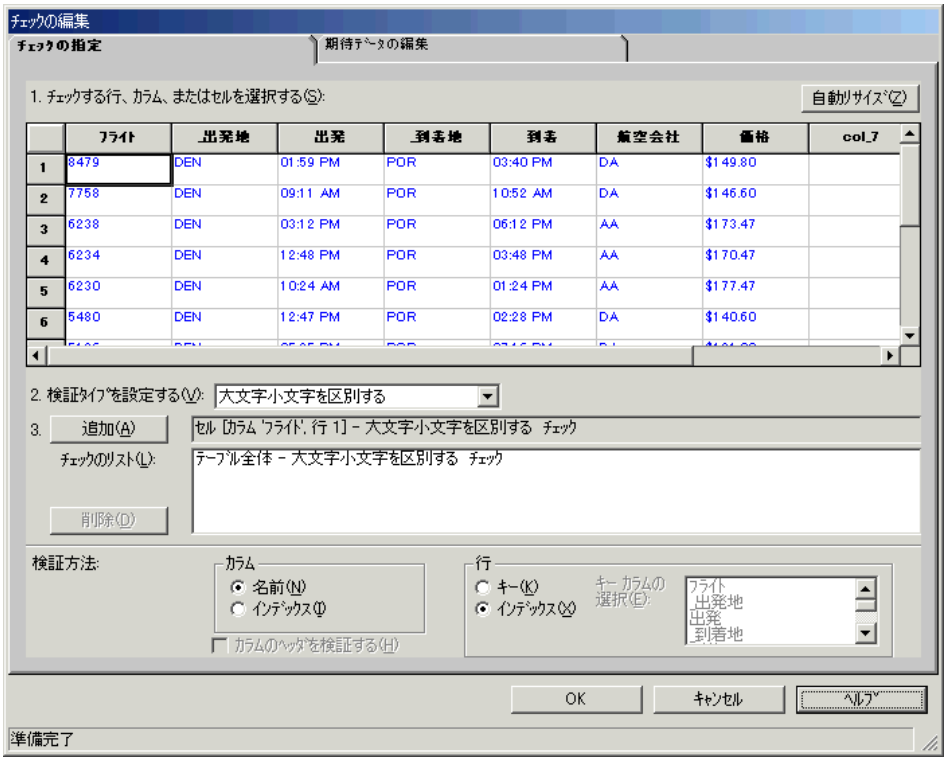
7 ［OK］をクリックして，［GUI チェック］ダイアログ・ボックスを閉じます。

obj_check_gui ステートメントがテスト・スクリプトに挿入されます。**obj_check_gui** 関数の詳細については，「**TSL リファレンス**」を参照してください。

注：テーブル内容に対して検査を行っている間に他のテーブル・オブジェクトのプロパティを検査したい場合は，［**挿入**］＞［**GUI チェックポイント**］＞［**複数のオブジェクト**］コマンド（［**挿入**］＞［**GUI チェックポイント**］＞［**オブジェクト / ウィンドウ**］コマンドではなく）を使用します。このコマンドは，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。テーブルの標準 GUI プロパティの検査については，第 9 章「GUI オブジェクトの検査」を参照してください。テーブルの ActiveX コントロール・プロパティの検査については，第 11 章「ActiveX と Visual Basic のコントロールの使用」を参照してください。

［チェックの編集］ダイアログ・ボックスについて

［**チェックの編集**］ダイアログ・ボックスを使用すると，テーブル内の検査対象セルと，使用する検証方式と検証のタイプを指定できます。検査に含まれるテーブル・セルの期待データを編集することもできます。



［**チェックの指定**］タブで，検査するテーブルのセル，検証方法，検証タイプを指定できます。

単一カラムのテーブルで検査を作成している場合，［チェックの編集］ダイアログ・ボックスの［チェックの指定］タブの内容は上の図と異なる場合があります。

詳細については，261 ページ「単一カラムのテーブルの検証方式の指定」. を参照してください。

検査対象セルの指定

［**チェックのリスト**］表示枠には，検証タイプをはじめ，実行されるすべての検査が表示されます。あるチェックポイントに対して［チェックの編集］ダイアログ・ボックスを初めて開いた場合は，標準の検査が表示されます。

- 複数カラムのテーブルに対する標準の検査は，カラム名と行のインデックス番号に基づく，テーブル全体に対する検査です。この検査では大文字と小文字が区別されます。
- 単一カラムのテーブルに対する標準の検査は，行の位置に基づく，テーブル全体に対する検査です。この検査では大文字と小文字が区別されます。

注：テーブルに同じ名前のカラムが複数含まれている場合，WinRunner は重複カラムを無視してこれらに対する検査を行いません。したがって，カラムのインデックス・オプションを選択しなければなりません。

標準の設定を使用したくない場合は，実行する検査を指定する前に標準の検査を削除しなければなりません。［**チェックのリスト**］ボックスで「全テーブル - ‘ 大小文字の区別 ’ チェック」エントリを選択し，［**削除**］**ボタン**をクリックします。または，［**チェックのリスト**］ボックスでこのエントリをダブルクリックします。WinRunner から強調表示されている検査を削除するかどうか確認を求めるメッセージが表示されます。［**はい**］をクリックします。

次に実行する検査を指定します。選択したセルに応じて，異なる検証のタイプを選択できるので，セルを選択する前に検証のタイプを指定してください。詳細については，262 ページ「検証タイプの指定」を参照してください。

検査対象セルを強調表示します。次に，**ツールバーの［追加］ボタンをクリックして，これらのセルの検査を追加します。**または，次のようにすることもできます。

- セルをダブルクリックして検査します。
- 行ヘッダをダブルクリックして，その行のすべてのセルを検査します。
- カラム・ヘッダをダブルクリックして，そのカラムのすべてのセルを検査します。
- 左上隅をダブルクリックして，テーブル全体を検査します。

［**チェックのリスト**］ボックスに検査するセルの説明が表示されます。

検証方式の指定

検証方式を選択して，WinRunner のテーブル内のカラムまたは行の識別方法を制御できます。検証方式は，テーブル全体に適用されます。検証方式の指定は，複数カラムのテーブルの場合と単一カラムのテーブルの場合で異なります。

複数カラムのテーブルに対する検証方式の指定

カラム

- **［名前］**：WinRunner は，カラム名に基づいて対象を探します。テーブル内でカラムの場所を移動しても，不一致にはなりません。
- **［インデックス］**：WinRunner は，カラムのインデックスまたは位置に基づいて対象を探します。テーブル内でカラムの位置を移動すると，不一致となります。このオプションは，テーブルに同じ名前のカラムが複数含まれる場合に選択します。詳細については，258 ページを参照してください。このオプションを選択すると，［**カラムのヘッダを検証する**］チェック・ボックスが使用できます。このチェック・ボックスを選択すると，セルの他にカラム・ヘッダも検査できます。

行

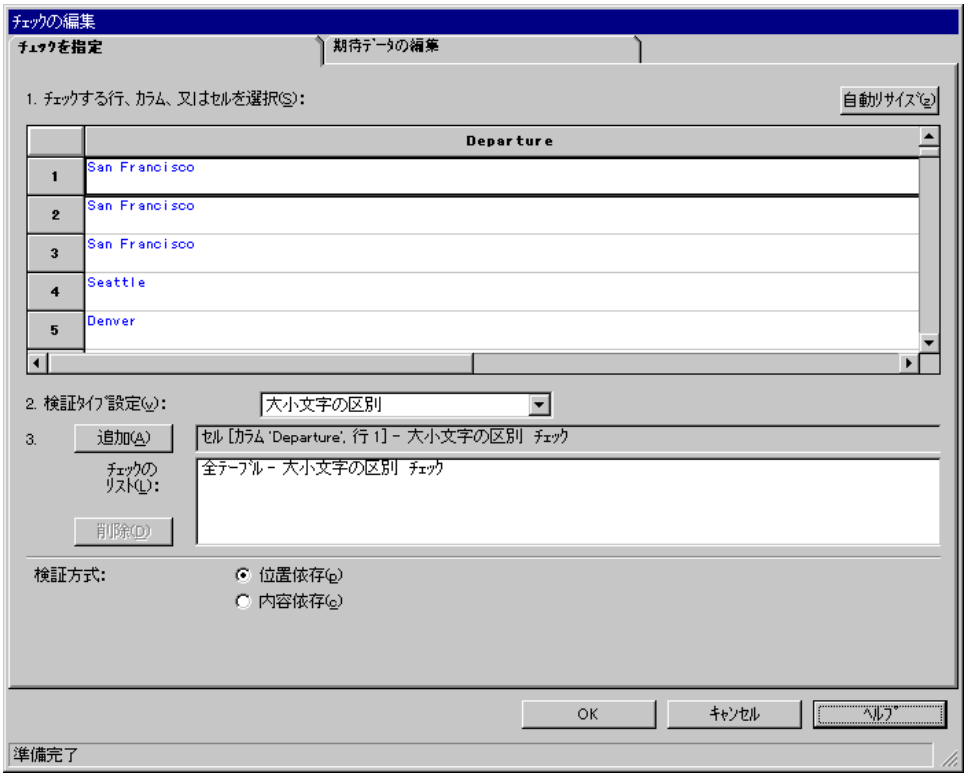
- **［キー］**：WinRunner は，テーブル内のすべてのカラムの名前をリストする［**キー選択**］リスト・ボックスで指定された「キー」カラム内のデータ（複数も可）に基づいて対象の中の行を探します。例えば，WinRunner に秒の到着時間に基づいて，263 ページのテーブル内の 2 行目を特定させることができます。行が移動しても不一致にはなりません。キー選択で行を一意に特定できない場合，WinRunner は最初に一致する行のみ検査します。1 つ以上のキー・カラムを指定して，行を一意に特定することができます。

注：キー・カラムの 1 つまたは複数のセルの値が変わると，WinRunner は対応する行を特定できず，「Not Found」エラーが出てその行の検査に失敗します。このエラーが生じたら，他のキー・カラムを選択するか，インデックス検証方法を使用します。

- **［インデックス］**（標準の設定）：WinRunner は，行のインデックスまたは位置に基づいて対象を探します。行の位置を移動すると，不一致となります。

単一カラムのテーブルの検証方式の指定

単一カラムのテーブルの［チェックの指定］タブにある［検証方式］ボックスは，複数カラム・テーブルのそれとは異なります。単一カラムのテーブルの標準の検査では，テーブル全体がその対象となります。検査では，行の位置が使われ，大文字と小文字が区別されます。



- **［位置依存］**：WinRunner は，カラム内の項目の場所に基づいて対象を検査します。
- **［内容依存］**：WinRunner はカラム内の項目の内容に基づいて対象を検査します。このとき，カラム内の場所は無視されます。

検証タイプの指定

WinRunner では，様々な方法でテーブルの内容を検証できます。セルの選択に合せて，異なる検証のタイプを選択できます。

- ［**大文字小文字を区別する**］（標準）：WinRunner は選択したテキストの内容を比較します。期待データと実際のデータ間の大文字と小文字の違いやテキスト内容の違いはすべて不一致となります。
- ［**大文字小文字を区別しない**］：WinRunner は選択したテキストの内容を比較します。期待データと実際のデータ間のテキスト内容の違いだけ不一致になります。
- ［**数値**］：WinRunner は，選択されたデータを数値として評価します。WinRunner は，例えば「2」と「2.00」を同じ数字と認識します。
- ［**数値の範囲**］：WinRunner は，選択されたデータを数値の範囲と比較します。最小値と最大値には，どちらも任意の実数を指定します。この比較は，実際のテーブル・データを，期待結果ではなく，指定された範囲と比較をする点がテキストおよび数値内容の検証とは異なります。

注：このオプションは，数値で始まらない文字列をすべて不一致とします。「e」で始まる文字列は数値に変換されます。

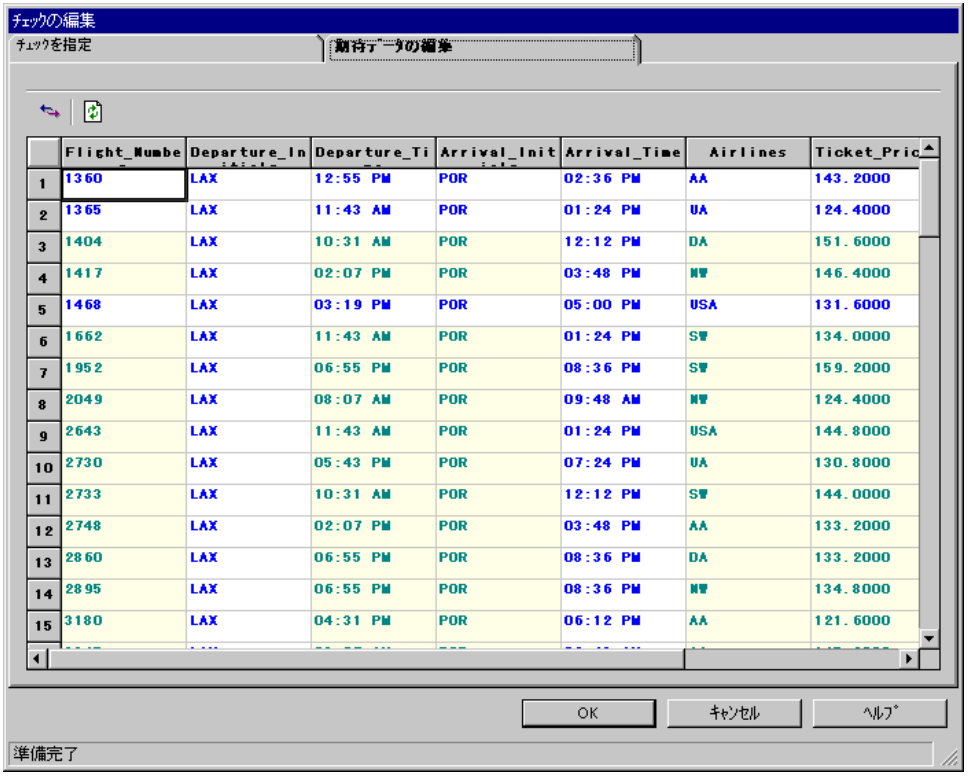
- ［**大文字小文字を区別する，スペースを無視する**］：WinRunner は，空白文字の違いを無視して，大文字と小文字また内容に基づいてセル内のデータを検査します。WinRunner は，大文字と小文字の違いやテキスト内容の違いをすべて不一致として報告します。
- ［**大文字小文字を区別しない，スペースを無視する**］：WinRunner は，大文字と小文字の違いと空白文字の違いを無視して，内容に基づいてセルの内容を検査します。WinRunner は，内容の違いだけを不一致として報告します。

［**OK**］をクリックして，［チェックの編集］ダイアログ・ボックスの両方のタブで行った変更を保存します。［チェックの編集］ダイアログ・ボックスを閉じ，［GUI チェック］ダイアログ・ボックスに戻ります。

期待データの編集

テーブル内の期待データを編集するには，［**期待データの編集**］タブをクリックします。［チェックの指定］タブで変更を加えた場合は，［**テーブルを再ロード**］をクリックしてチェックリストからテーブル選択を再ロードできます。保存したデータを再ロードするかどうかを確認するメッセージが表示されます。［はい］をクリックします。

［チェックの指定］タブの変更を保存した後で，［チェックを編集］ダイアログ・ボックスを開き直すと，［期待データの編集］タブにテーブルが色分けされて表示されます。検査に含まれるセルは，白い背景に青い文字で表示されます。検査から除外されるセルは，黄色の背景に緑の文字で表示されます。



セル内のデータの期待値を編集するには，セル内でダブルクリックします。セル内にカーソルが現れます。セルの内容を必要に応じて変更します。［**OK**］をクリックして，［チェックの編集］ダイアログ・ボックス内の両方のタブの変

更を保存します。ダイアログ・ボックスを閉じ，［GUI チェック］ダイアログ・ボックスに戻ります。

第 14 章

データベースの検査

データベース実行時レコード・チェックポイントを追加すれば，テスト実行中のアプリケーションの情報と，データベースに記録されている情報を比較できます。

標準のデータベース・チェックポイントをテスト・スクリプトに追加すれば，異なったバージョンのアプリケーション内でデータベースの内容を検査できます。

本章では，以下の項目について説明します。

- データベースの検査について
- データベース実行時レコード・チェックポイントの作成
- 実行時データベース・レコード・チェックリストの編集
- データベースを対象とする標準の検査の作成
- データベースを対象とするユーザ定義の検査の作成
- ［データベース チェックポイント］ダイアログ・ボックスのメッセージ
- ［データベース チェックポイント］ウィザードでの作業
- ［チェックの編集］ダイアログ・ボックスについて
- 標準のデータベース・チェックポイントの変更
- 標準のデータベース・チェックポイントの期待結果の変更
- 標準のデータベース・チェックポイントのパラメータ化
- データベースの指定
- TSL 関数を使用してのデータベース作業

データベースの検査について

データベース・チェックポイントを作成するときは，データベースに対してクエリを定義すると，データベース・チェックポイントによって**結果セット**に含まれている値が検査されます。結果セットとは，クエリの結果から取得される値のセットです。

データベース・チェックポイントで使用されるクエリを定義するには，以下の方法があります。

- Microsoft Query を使って，データベースに対し「**クエリ**」を定義できます。データベースに対するクエリの結果を「**結果セット**」と言います。Microsoft Query は，Microsoft Office の「**カスタム・インストール**」からインストールできます。
- ODBC クエリは，SQL ステートメントを作成して，手作業で定義できます。
- Data Junction を使って，データベースを「**ターゲット**」テキスト・ファイルに変換する「**変換**」ファイルを作成できます（標準のデータベース・チェックポイントのみ）。Data Junction は通常 WinRunner パッケージに含まれていません。Data Junction の購入する場合は Mercury にお問い合わせください。Data Junction を使用した作業の詳細については，Data Junction パッケージのマニュアルを参照してください。

本章では，わかりやすいように，ODBC クエリの結果または Data Junction による変換後のターゲットを結果セットと呼びます。

データベース実行時レコード・チェックポイントについて

データベース実行時レコード・チェックポイントを作成して，テスト実行中にアプリケーションに表示される値とデータベース内の対応する値を比較できます。比較の結果がチェックポイントに対して指定した成功の基準に達していなければ，そのチェックポイントは失敗となります。データベース実行時レコード・チェックポイントは，一致するレコードが 1 つ以上の場合，一致するレコードが 1 つだけの場合，一致するレコードがない場合というように，「成功」の基準を自分で定義できます。データベース・チェックポイントはループに含めることができます。データベース・チェックポイントをループで実行すると，チェックポイントの各反復の結果はエントリごとにテスト結果に表示されます。チェックポイントの結果は［テスト結果］ウィンドウに表示されます。詳細については，第 20 章「テスト結果の分析」を参照してください。

実行時レコード・チェックポイントは，実行するたびにデータベース内の情報が変わる場合に役立ちます。実行時レコード・チェックポイントを使用すると，アプリケーションに表示された情報がデータベースに正しく挿入されたか，あるいは逆に，データベースから情報が正しく取得され画面に表示されたかを検証できます。

データベース実行時レコード・チェックポイントを作成する場合，アプリケーションおよびデータベースに含まれるデータは通常同じ形式です。データの形式が異なる場合は，276 ページ「形式の違うデータの比較」に示す指示に従って，データベース実行時レコード・チェックポイントを作成してください。これは WinRunner の上級ユーザ向けの機能です。

標準のデータベース・チェックポイントについて

標準のデータベース・チェックポイントを作成して，テスト実行中の結果セットのプロパティの現在値を，記録中にキャプチャした期待値または実行前に設定された期待値と比較できます。期待結果と現在の結果が一致しないと，データベース・チェックポイントは失敗します。

標準のデータベース・チェックポイントは，テスト実行前に期待値を設定できる場合に役立ちます。標準のデータベース・チェックポイントには，標準とユーザ定義の 2 種類があります。

WinRunner では，データベースに対して定義したクエリの結果に基づいて，データベース・チェックポイントを作成します。**データベース・チェックポイント**は，標準の検査かユーザ定義の検査のどちらかになります。ユーザ定義の検査では，検査するプロパティを指定します。

標準の検査を使って結果セットの全内容を検査できます。また，ユーザ定義の検査を使えば，結果セットの行数やカラム数など一部の内容についても検査できます。検査対象となる結果セットのプロパティについての情報は，「**チェックリスト**」に保存されます。その後に WinRunner は，現在のデータベースについての情報をキャプチャし，この情報を「**期待結果**」として保存します。「**データベース・チェックポイント**」は自動的にテスト・スクリプトに挿入されます。このチェックポイントは，テスト・スクリプトに **db_check** ステートメントとして記録されます。

例えば，アプリケーションのデータベースをテスト・スクリプト内で初めて検査すると，次のステートメントが生成されます。

```
db_check("list1.cdl", "dbvf1");
```

list1.cdl には，検査するデータベースとプロパティについての情報を含む「**チェックリスト**」の名前、また dbvf1 には「期待結果ファイル」の名前が入ります。チェックリストは，テストの「***chklist***」フォルダに格納されます。

Microsoft Query または ODBC で作業している場合，「***chklist***」フォルダは，データベースと SQL ステートメントについての情報を含む「***.sql**」クエリ・ファイルを参照します。Data Junction で作業している場合，「***chklist***」フォルダは，データベースと変換についての情報を含む「***.djs**」変換ファイルを参照します。クエリを定義すると，WinRunner はチェックリストを作成してテストの「***chklist***」フォルダに格納します。期待結果ファイルは，テストの「***exp***」フォルダに格納されます。**db_check** 関数の詳細については，「**TSL リファレンス**」を参照してください。

テストを実行すると，データベース・チェックポイントは，テスト対象のアプリケーション内のデータベースの現在のステータスと期待結果を比較します。期待結果と現在の結果が一致しないと，データベース・チェックポイントは失敗します。データベース・チェックポイントはループに含めることができます。データベース・チェックポイントをループで実行すると，チェックポイントの各反復の結果はエントリごとにテスト結果に表示されます。チェックポイントの結果は［テスト結果］ウィンドウで表示できます。詳細については，第 20 章「テスト結果の分析」を参照してください。

テスト実行前または実行後に，既存の標準のデータベース・チェックポイントの期待結果を変更できます。既存のデータベース・チェックポイントのクエリを変更することもできます。これは，ネットワーク上でデータベースを新しい位置に移動する場合に便利です。

ODBC/Microsoft Query を使ってデータベース・チェックポイントを作成する場合は，SQL ステートメントにパラメータを追加してチェックポイントをパラメータ化できます。これは，クエリを定義する SQL ステートメントが変更されるクエリに対してデータベース・チェックポイントを作成する場合に便利です。詳細については，317 ページ「標準のデータベース・チェックポイントのパラメータ化」を参照してください。

失敗したデータベース・チェックポイントのオプションの設定

データベース・チェックポイントが失敗するたびに，選択した受信者に電子メールを送信するよう，またチェックポイントが失敗したウィンドウまたは画面のビットマップをキャプチャするよう，WinRunner に指示できます。これらのオプションは［一般オプション］ダイアログ・ボックスで設定します。

データベース・チェックポイントが失敗したら電子メールを送信するよう WinRunner に指示するには，次の手順を実行します。

1 ［**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・ボックスが開きます。

2 オプション表示枠で［**通知**］カテゴリを選択します。通知オプションが表示されます。

3 ［**データベース チェックポイントの失敗**］を選択します。

4 オプション表示枠で［**通知**］＞［**電子メール**］カテゴリを選択します。電子メール・オプションが表示されます。

5 ［**電子メールのサービスを有効にする**］オプションを選択して，関連するサーバと送信者情報を設定します。

6 オプション表示枠で［**通知**］＞［**受信者**］カテゴリを選択します。電子メールの受信者オプションが表示されます。

7 必要に応じて受信者の追加，削除，変更を行い，データベース・チェックポイントの失敗時に電子メールを送信する受信者を設定します。

電子メールには，テストとチェックポイントの詳細なサマリと，チェックポイントに使用された接続文字列と SQL クエリの詳細が含まれます。詳細については，560 ページ「通知オプションの設定」を参照してください。

チェックポイントが失敗したら，ビットマップをキャプチャするよう WinRunner に指示するには，次の手順を実行します。

1 ［**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・ボックスが開きます。

2 オプション表示枠で［**実行**］＞［**設定**］カテゴリをクリックします。実行設定オプションが表示されます。

3 ［**検証失敗の時，ビットマップをキャプチャする**］を選択します。

4 ［**Window**］，［**Desktop**］または［**Desktop Area**］を選択して，チェックポイントの失敗時にキャプチャするものを指定します。

5 ［**Desktop Area**］を選択した場合は，キャプチャするデスクトップの座標を指定します。

テストを実行すると，キャプチャされたビットマップが結果フォルダに保存されます。詳細については，518 ページ「[一般オプション］ダイアログ・ボックスでのグローバル・テスト・オプションの設定」を参照してください。

データベース実行時レコード・チェックポイントの作成

データベース実行時レコード・チェックポイントをテストに追加して，テスト実行中にアプリケーションに表示される情報と，データベース内の対応するレコードの現在値を比較できます。

実行時レコード・チェックポイント・ウィザードを実行して，データベース実行時レコード・チェックポイントを追加します。ウィザードを終了すると，スクリプトに適切な **db_record_check** ステートメントが挿入されます。

データベース実行時レコード・チェックポイントを作成する場合，アプリケーションおよびデータベースに含まれるデータは通常同じ形式です。データの形式が異なる場合は，276 ページ「形式の違うデータの比較」に示す指示に従って，データベース実行時レコード・チェックポイントを作成してください。これは WinRunner の上級ユーザ向けの機能です。

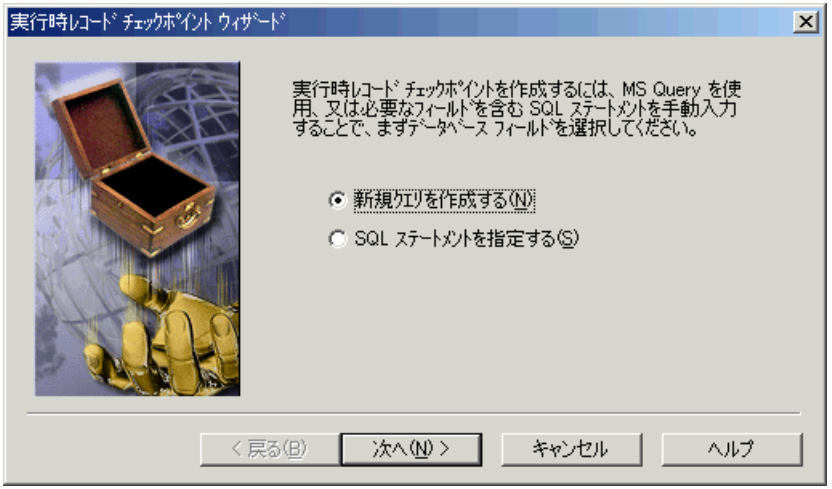
実行時レコード・チェックポイント・ウィザードの使用

実行時レコード・チェックポイント・ウィザードは，クエリの定義，クエリのレコードに対応する情報を含むアプリケーション・コントロールの特定，チェックポイントに対する成功基準の定義などを順を追って示します。

ウィザードを開くするには，［**挿入**］＞［**データベース チェックポイント**］＞［**実行時レコードのチェック**］を選択します。

クエリの定義画面

クエリの定義画面では，データベースを選択してチェックポイントに対してクエリを定義できます。Microsoft Query を使って，データベースから新しいクエリを作成することができます。SQL ステートメントを手作業で定義することもできます。



次のオプションを選択できます。

- **[新規クエリを作成する]**：Microsoft Query を開いて，新しいクエリを作成できます。クエリの定義が終了すると WinRunner に戻ります。詳細については，321 ページ「ODBC/Microsoft Query でのクエリの作成」を参照してください。このオプションは，お使いのマシンに Microsoft Query がインストールされている場合のみ有効です。

  [SQL ステートメントを指定する]：ウィザードの［**SQL ステートメントを指定**］画面を開いて，接続文字列と SQL ステートメントを指定します。詳細については，295 ページ「SQL ステートメントを指定」を参照してください。

[SQL ステートメントを指定] 画面

[SQL ステートメントを指定] 画面では，データベースの接続文字列と SQL ステートメントを手作業で指定できます。

必要な情報を入力します。

- **[接続文字列]**：接続文字列を入力して，[作成] ボタンをクリックします。
  - **[作成]**：[データソースの選択] ダイアログ・ボックスを開きます。[データソースの選択] ダイアログ・ボックスで *.dsn ファイルを選択し，ボックスに 接続文字列を挿入できます。
- **[SQL]**：SQL ステートメントを入力します。

注：db_record_check 関数で，”SELECT * from ...” **タイプの** SQL ステートメントは使用できません。代わりに，テーブル名とフィールド名を指定しなければなりません。期待される SQL の形式は以下のとおりです。これは，The reason for this is that WinRunner がどのデータベース・フィールドが WinRunner スクリプト内のどの変数と一致するかを知っておく必要があるからです。

SELECT table_name1.field_name1, table_name2.field_name2, ... FROM table_name1, table_name2, ... [WHERE ...]

データベース・フィールドとの対応付け画面

データベース・フィールドとの対応付け画面では，表示されているデータベース・フィールドと 一致するアプリケーションのコントロールまたはテキストを特定できます。クエリに含まれている各フィールドでこのステップを繰り返します。

この画面には，次のオプションがあります。

- **データベース フィールド：**クエリからデータベースのフィールドを表示します。指差し型ポインタを使って，表示されているフィールド名と一致するコントロールまたはテキストを特定します。

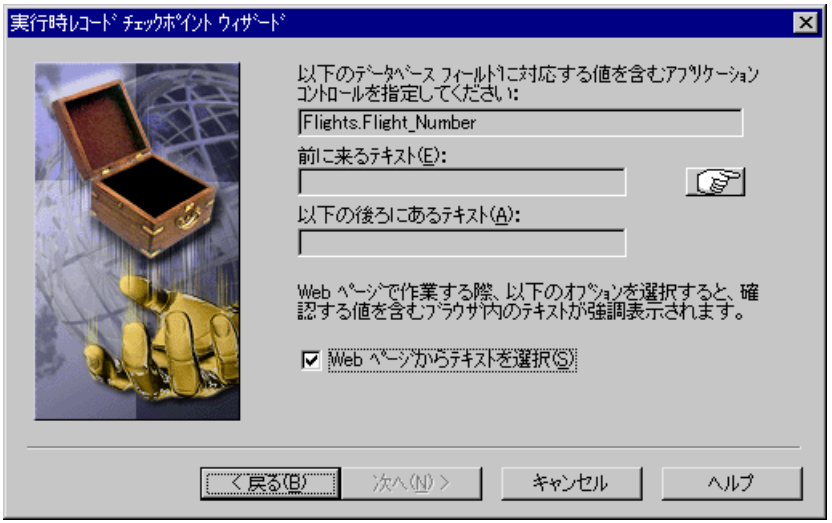
➤ **[論理名]**：アプリケーションで選択したコントロールの論理名を表示します。

➤ **[前に来るテキスト]**：検査するテキストのすぐ前に現れるテキストが表示されます。

（[**Web ページからテキストを選択**] チェック・ボックスがチェックされているときだけ表示されます。）

➤ **[以下の後ろにあるテキスト]**：検索するテキストのすぐ後に現れるテキストが表示されます。

（[**Web ページからテキストを選択**] チェック・ボックスが選択されているときだけ表示されます。）



➤ **[Web ページからテキストを選択]**: 検証する値が含まれている Web ページのテキストを指定できます。

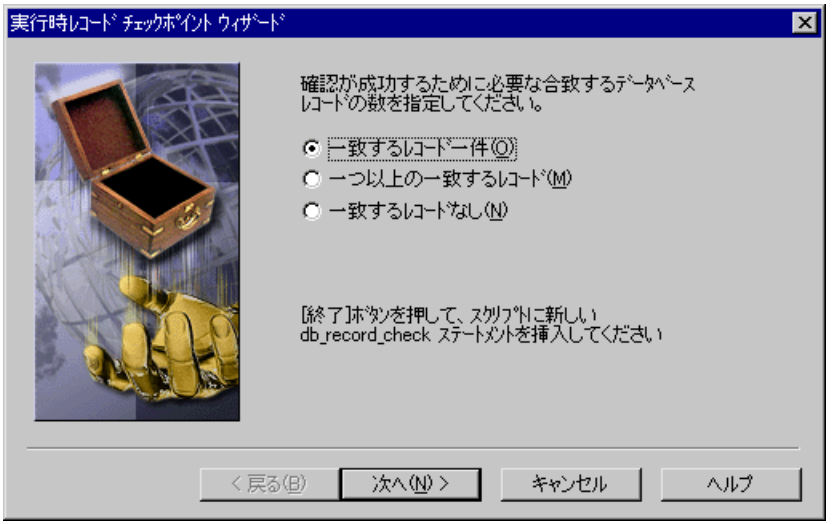
注：

Web ページからテキストを選択するときは，ポインタを使用して選択してください。

Web ページのテキスト文字列にマップされているデータベース・フィールドにデータベース・チェックポイントを作成するには，WebTest アドインをロードする必要があります。必要に応じて，チェックポイントを作成する前に WebTest アドインをロードして WinRunner を再起動しなければなりません。アドインのロードについては，21 ページ「WinRunner アドインのロード」を参照してください。

一致するレコード件数の設定画面

一致するレコード件数の設定画面では，チェックポイントを成功させるのに必要な，データベース・レコードとの一致数を指定できます。



- **[一致するレコード一件]**：一致するデータベース・レコードが 1 つだけ検索されたら，チェックポイントの成功と設定します。
- **[一つ以上の一致するレコード]**：一致するデータベース・レコードが 1 つ以上検索されたら，チェックポイントの成功と設定します。
- **[一致するレコードなし]**：一致するデータベース・レコードが 1 つも検索されなかったら，チェックポイントの成功と設定します。

実行時レコード・チェックポイント・ウィザードで［**完了**］をクリックすると，**db_record_check** ステートメントがスクリプトに挿入されます。**db_record_check** 関数の詳細については，「**TSL リファレンス**」を参照してください。

形式の違うデータの比較

アプリケーションのデータをデータベース内のデータと比較するときに，データの形式が異なる場合は，実行時レコード・チェックポイント・ウィザードを使わずに，次の指示でデータベース実行時レコード・チェックポイントを作成できます。これは WinRunner の上級ユーザ向けの機能です。

例えば，サンプルのフライト予約アプリケーションの［クラス］ボックスには 3 つのラジオ・ボタンがあります。このボックスが有効なときには，常にラジオ・ボタンが 1 つ選択されています。サンプルのフライト予約アプリケーションのデータベースには，一致するクラスとして 1，2 または 3 の値を持つフィールドが 1 つあります。

アプリケーションのデータとデータベースのデータが同じ値がどうかを検査するには，次のステップを実行します。

1 アプリケーションの記録を行い，画面でデータを検証する場所に来たら，テストを停止します。テストから，アプリケーションの値を手作業で取り出します。

2 アプリケーションから取り出した値を元に，データベースの期待値を計算します。このステップを実行するには，両方の値セットの間のマッピング関係が分からなくてはなりません。次の例を参照してください。

3 計算した値を編集フィールドかエディタ（メモ帳など）に追加します。編集フィールドは，計算した値それぞれに 1 つずつ必要です。例えば，複数の［メモ帳］ウィンドウまたは複数の編集フィールドを持つアプリケーションを使用します。

4 GUI マップ・エディタを使って，WinRunner を学習させます。

- 検査する値を含むアプリケーションのコントロール
- 計算した値に使用する編集フィールド

5 TSL ステートメントを次の操作を実行するテスト・スクリプトに追加します。

- アプリケーションから値を取り出す
- アプリケーションから取り出した値を元にデータベースの期待値を計算する
- 期待値を編集フィールドに書き込む

6 270 ページ「実行時レコード・チェックポイント・ウィザードの使用」で説明したように，実行時レコード・チェックポイント・ウィザードを使って **db_record_check** ステートメントを作成します。

表示される指示に従って，期待値を持つアプリケーションのコントロールではなく，計算された期待値を入力する編集フィールドの場所を示します。

ヒント：テストを実行する場合は，計算された値を含む編集フィールドを含むアプリケーションを開いていることを確認してください。

データベース実行時レコード・チェックポイントの形式の違うデータの比較例

次のスクリプトの一部は，サンプルのフライト予約アプリケーションのラジオ・ボタンに対するデータベース内の［クラス］フィールドの検査に使用されます。ステップについては，277 ページの説明を参照してください。

ステップ 1

```
# アプリケーションの GUI オブジェクトから値を取り出します。
button_get_state("First",vFirst);
button_get_state("Business",vBusiness);
button_get_state("Economy",vEconomy);
```

ステップ 2

```
# データベースに対する期待値を計算します。
if (vFirst)
    expDBval = "1" ;
else if (vBusiness)
    expDBval = "2" ;
else if (vEconomy)
    expDBval = "3" ;
```

ステップ 3

```
# 計算した値をチェックポイントで使用される編集フィールドに追加します。
set_window("Untitled - Notepad", 1);
edit_set("Edit", expDBval);
```

ステップ 4

```
# ウィザードを使って，データベース実行時レコード・チェックポイントを作成します。
db_record_check("list1.cvr", DVR_ONE_MATCH);
```

実行時データベース・レコード・チェックリストの編集

実行時データベース・レコード・チェックポイント用に作成したチェックリストを変更できます。チェックリストには，データベースへの接続文字列，SQL ステートメントまたはクエリ，データ・ソース内のデータベース・フィールド，ご使用のアプリケーションのコントロール，アプリケーション間のマッピング情報が含まれています。実行時データベース・レコード・チェックポイントの成功条件は含まれていません。

実行時データベース・レコード・チェックリストの編集では，次のことができます。

- ODBC の使用，または手作業でデータ・ソース接続文字列を変更します。
- SQL ステートメントの変更，または Microsoft Query で他のクエリを選択します。
- データ・ソース内で異なる使用データベース・フィールドを選択します（追加または削除）。
- 既にチェックリストにあるデータベース・フィールドを他のアプリケーション・コントロールに一致させます。
- アプリケーション・コントロールに新しいデータベース・フィールドを一致させます。

既存の実行時データベース・レコード・チェックリストを編集するには，次の手順を実行します。

1 ［**挿入**］＞［**実行時レコード チェックリストの編集**］を選択します。

［実行時レコード チェックポイント ウィザード］が開きます。

2 編集する実行時データベース・レコード・チェックリストを選択します。［次へ］をクリックします。

注：標準設定では，実行時データベース・レコード・チェックリストの名前は，各テストで **list1.cvr** から順につけられています。

ヒント：編集するチェックリスト名は，**db_record_check** ステートメントで確認することができます。

3［SQL ステートメントを指定します］画面が開きます。

この画面上では，次のことが行えます。

- 手作業で接続文字列を変更するか，［編集］をクリックして［データ・ソースの選択］ダイアログ・ボックスを開き，新しい接続文字列を作成するための .dsn ファイルを選択します。
- SQL ステートメントを手作業で変更するか，［Microsoft Query］ボタンをクリックして Microsoft Query を開き，クエリを再定義します。

注：Microsoft Query がお使いのマシンにインストールされていない場合は，［Microsoft Query］ボタンは表示されません。

［次へ］をクリックします。

4 データベース・フィールドを対応付ける画面が開きます。

「New」アイコンは，このデータベース・フィールドが以前はチェックリストに含まれていなかったことを示します。

➤ チェックリストに含まれていたデータベース・フィールドでは，割り当てられているアプリケーション・コントロールと共にデータベース・フィールドが表示されます。指差しを使って，表示されたフィールド名を他のアプリケーション・コントロールまたは，Web ページのテキスト文字列に割り当てます。

注：Web ページのテキスト文字列に割り当てられているデータベース・フィールドを編集するには，WebTest アドインがロードされていることを確認してください。必要に応じて，チェックリスト内のオブジェクトを編集する前に，WinRunner を再起動して WebTest アドインをロードします。アドインのロードの詳細は，21 ページ「WinRunner アドインのロード」を参照してください。

➤ SQL ステートメントまたは Microsoft Query を変更して，それらがデータ・ソース内の新たなデータベース・フィールドを参照している場合は，チェックリストに新しいデータベース・フィールドが追加されます。このデータベース・フィールドをアプリケーション・コントロールに対応付けてください。指差しを使って，表示されているフィールド名に対応付けるコントロールまたは，テキストをポイントします。

ヒント：新しいデータベース・フィールドには［New］アイコンがつきます。

注：Web ページ内でデータベース・フィールドを割り当てるには，［Web ページからテキストを選択］チェック・ボックスをクリックします。このチェック・ボックスは，WebTest アドインがロードされているときに有効になっています。ウィザード画面で追加オプションが表示されます。これらのオプションの詳細は，273 ページ「データベース・フィールドとの対応付け画面」を参照してください。

［次へ］をクリックします。

注：Microsoft Query のクエリ，または SQL ステートメント内のそれぞれのデータベース・フィールドにつき 1 回，「データベース・フィールドとの対応付け」画面が表示されます。この画面が表示されたら画面の説明に従ってください。

5 ［完了］画面が表示されます。

［完了］をクリックすると，実行時レコード・チェックポイントで使用されたチェックリストを変更できます。

注：テスト・スクリプト内で **db_record_check** ステートメントの第 2 パラメータを変更すると，チェックポイントの成功条件を変更できます。第 2 パラメータには，次の値が含まれるようにします。

- DVR_ONE_OR_MORE_MATCH - チェックポイントは 1 つまたはそれ以上の一致するデータベース・レコードが見つかると成功します。
- DVR_ONE_MATCH - チェックポイントは，完全に一致するデータベース・レコードが 1 つ見つかると成功します。

- DVR_NO_MATCH - チェックポイントは，一致するデータベース・レコードが 1 つも見つからないと成功します。

詳しくは，「**TSL リファレンス**」を参照してください。

ヒント：既存の実行時レコード・チェックポイントを複数使用できます。ご使用のテスト・スクリプトで実行時レコード・チェックポイントを既に作成していて，同じテストで，同じデータ・ソースおよび SQL ステートメントまたはクエリを他の実行時レコード・チェックポイントを使用したいとします。例えば複数の異なる **db_record_check** ステートメントにそれぞれ異なる成功条件を指定する場合などです。この場合，新しく作成する各チェックポイントで繰り返し［実行時レコード・チェックポイント］ウィザードを実行する必要はありません。そのかわり，手作業で既存のチェックリストを参照する **db_record_check** ステートメントを入力します。同じように，**db_record_check** ステートメントが既存のチェックリストを参照するように変更することもできます。

データベースを対象とする標準の検査の作成

データベースを対象とする標準の検査を作成する場合，次の基準に従って結果セット全体を検査する標準のデータベース・チェックポイントを作成します。

- データベースの複数のカラムのクエリに対する標準の検査は，カラム名と行インデックスに基づくテスト結果全体に対する検査で，大文字と小文字が区別されます。
- データベースの単一のカラムのクエリに対する標準の検査は，行の位置に基づくテスト結果全体に対する検査で，大文字と小文字が区別されます。

結果セットの一部の内容だけを検査する，内容の期待値を編集する，あるいはカラム数や行数を数えるには，標準の検査ではなくユーザ定義の検査を作成します。データベースを対象とするユーザ定義の検査の作成方法については，287 ページ「データベースを対象とするユーザ定義の検査の作成」を参照してください。

ODBC または Microsoft Query でのデータベースを対象とする標準の検査の作成

ODBC または Microsoft Query を使用して，データベースを対象とする標準の検査を作成できます。

ODBC または Microsoft Query を使用してデータベースを対象とする標準の検査を作成するには，次の手順を実行します。

1 ［挿入］＞［データベース チェックポイント］＞［標準チェック］を選ぶか，ユーザ定義ツールバーの［**標準設定データベース チェックポイント**］ボタンをクリックします。［**アナログ**］モードで記録している場合は，マウスの余計な動きが記録されないように［データベース チェック（標準)］ソフトキーを押します。コンテキスト・センシティブ・モードでも［データベース チェック(標準)］ソフトキーを押すことができます。

注：標準のデータベース・チェックポイントを初めて作成する場合，Microsoft Query または［データベース チェックポイントウィザード］が開きます。次回標準のデータベース・チェックポイントを作成するときには，前回使用したツールが開きます。［データベース チェックポイントウィザード］が開いた場合は，291 ページ「[データベース チェックポイント］ウィザードでの作業」の指示に従ってください。

---

2 Microsoft Query がインストールされているマシンで新しいクエリを作成する場合は，クエリを作成するための手順を示す画面が開きます。

次回標準のデータベース・チェックポイントを作成するときにこのメッセージを表示しないようにするには，［**次回にこのメッセージを表示する**］チェック・ボックスをクリアにします。

［**OK**］をクリックして手順の画面を閉じます。

Microsoft Query がインストールされていないと，ODBC クエリを手作業で定義できる［データベース チェックポイントウィザード］が開きます。詳細については，292 ページ「ODBC（Microsoft Query）オプションの指定」を参照してください。

3 クエリを定義またはコピーするか，SQL ステートメントを指定します。詳細については 321 ページ「ODBC/Microsoft Query でのクエリの作成」か 295 ページ「SQL ステートメントを指定」を参照してください。

注：生成する **db_check** ステートメント内の SQL ステートメントをパラメータ化する場合，Microsoft Query のウィザードの最終画面で［**Microsoft Query でデータの表示またはクエリの編集を行う**］をクリックします。320 ページ「SQL ステートメントをパラメータ化するためのガイドライン」の指示に従ってください。

---

4 WinRunner がデータベース・クエリをキャプチャするには，数秒かかります。その後 WinRunner のウィンドウに戻ります。

WinRunner は，**クエリが指定したデータをキャプチャし，テストの「exp」フォルダに格納します。WinRunner は，msqr*.sql クエリ・ファイルを作成し，これをテストの「*chklist*」**フォルダのデータベース・チェックリストに格納します。データベース・チェックポイントは，**db_check** ステートメントとしてテスト・スクリプトに挿入されます。**db_check** 関数の詳細については，「**TSL リファレンス**」を参照してください。

Data Junction でのデータベースを対象とする標準の検査の作成

Data Junction を使用して，データベースを対象とする標準の検査を作成できます。

データベースを対象とする標準の検査を作成するには，次の手順を実行します。

1 ［**挿入**］＞［**データベース チェックポイント**］＞［**標準チェック**］を選ぶか，ユーザ定義ツールバーの［**標準設定データベース チェックポイント**］ボタンをクリックします。［**アナログ**］モードで記録している場合は，マウスの余計な動きが記録されないように［データベース チェック（標準)］ソフトキーを押します。コンテキスト・センシティブ・モードでも［データベース チェック(標準)］ソフトキーを押すことができます。

注：標準のデータベース・チェックポイントを初めて作成する際は，［Microsoft Query］または［データベース チェックポイント］ウィザードが開きます。次回から標準のデータベース・チェックポイントを作成するときには，前回のクライアントが開きます。Microsoft Query を使用すると Microsoft Query が開き，Data Junction を使用すると［データベース チェックポイント］ウィザードが開きます。Data Junction を使用してデータベース・チェックポイントを作成する場合，必ず［データベース チェックポイント ］ウィザードが開きます。

［データベース チェックポイント］ウィザードの詳細については，291 ページ「［データベース チェックポイント］ウィザードでの作業」を参照してください。

2 クエリを作成するための手順を示す画面が開きます。次回から標準のデータベース・チェックポイントを作成するときにこのメッセージを表示しない場合，［**次回にこのメッセージを表示する**］チェック・ボックスをクリアにします。

［**OK**］をクリックして手順の画面を閉じます。

3 新しい変換ファイルを作成するか既存のファイルを使用します。詳細については，287 ページ「データベースを対象とするユーザ定義の検査の作成」を参照してください。

4 WinRunner がデータベース・クエリをキャプチャするには，数秒かかります。その後 WinRunner のウィンドウに戻ります。

WinRunner は，**クエリによって指定されたデータをキャプチャし，テストの「exp」フォルダに格納します。**WinRunner は，***.djs 変換ファイルを作成し，それをテストの「*chklist*」**フォルダのチェックリストに格納します。データベース・チェックポイントは，**db_check** ステートメントとしてテスト・スクリプトに挿入されます。**db_check** 関数の詳細については，「**TSL リファレンス**」を参照してください。

データベースを対象とするユーザ定義の検査の作成

データベースを対象とするユーザ定義検査を作成するときに，検査する結果セットのプロパティを指定できる標準のデータベース・チェックポイントを作成します。

データベースを対象とするユーザ定義検査を作成すると，以下のことができます。

- 結果セット全体の内容または一部の内容を検査する。
- 結果セットの内容の期待結果を編集する。
- 結果セット内の行数を数える。
- 結果セット内のカラム数を数える。

ODBC，Microsoft Query，または Data Junction を使用して，データベースを対象とするユーザ定義検査を作成できます。

データベースを対象とするユーザ定義の検査を作成するには，次の手順を実行します。

1 ［**挿入**］＞［**データベース チェックポイント**］＞［**カスタムチェック**］を選びます。アナログ・モードで記録している場合は，マウスの余計な動きが記録されないように［データベース チェック（標準)］ソフトキーを押します。コンテキスト・センシティブ・モードでも［データベース チェック（標準)］ソフトキーを押すことができます。

［データベース チェックポイント］ウィザードが開きます。

2 291 ページ「［データベース チェックポイント］ウィザードでの作業」に示す［データベース チェックポイント］ウィザードの使用時の指示に従います。

3 新しいクエリを作成する場合，クエリを作成するための手順を示す画面が開きます。

次回から標準のデータベース・チェックポイントを作成するときにこのメッセージを表示しない場合は，［**次回にこのメッセージを表示する**］チェック・ボックスをクリアにします。

4 ODBC または Microsoft Query を使用する場合は，クエリを定義またはコピーするか，SQL ステートメントを指定します。詳細については，321 ページ「ODBC/Microsoft Query でのクエリの作成」か 295 ページ「SQL ステートメントを指定」を参照してください。

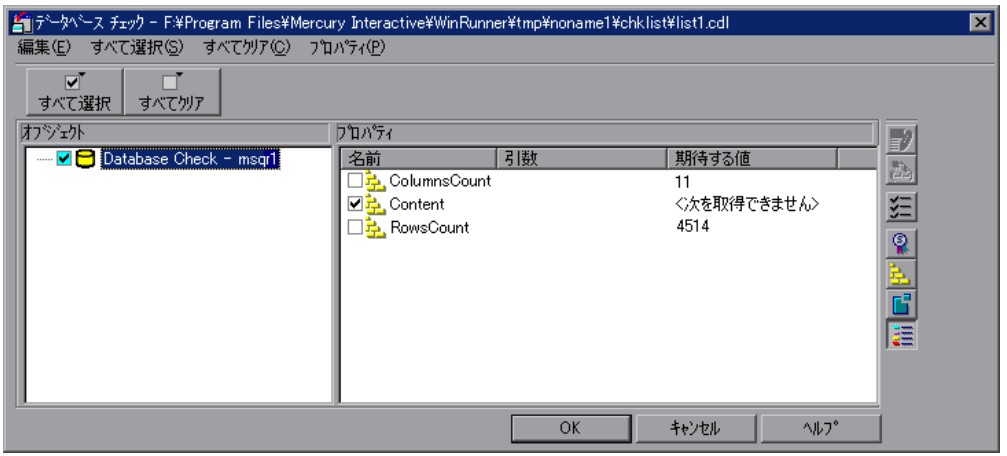
Data Junction を使用する場合は，新しい変換ファイルを作成するか，既存のファイルを使用します。詳細については，322 ページ「Data Junction での変換ファイルの作成」を参照してください。

5 Microsoft Query を使って，生成する **db_check** ステートメント内の SQL ステートメントをパラメータ化する場合，Microsoft Query のウィザードの最終画面で

［**Microsoft Query でデータの表示またはクエリの編集を行う**］をクリックします。320 ページ「SQL ステートメントをパラメータ化するためのガイドライン」の指示に従ってください。

6 WinRunner は，データベース・クエリをキャプチャして元のウィンドウに戻るまで数秒かかります。

［**データベース チェック**］ダイアログ・ボックスが開きます。



［**オブジェクト**］表示枠には「Database Check」と，データベース・チェックポイントに含まれる **.sql* クエリ・ファイルか ***.djs 変換ファイル**の名前が含まれます。［**プロパティ**］表示枠には，結果セットを対象に実行できる様々な種類の検査の一覧が表示されます。チェックマークは，その項目が選択され，チェックポイントに含まれていることを示します。

7 データベース上で実行する検査の種類を選択します。次の検査を実行できます。

ColumnsCount：結果セット内のカラム数を数えます。

Content：284 ページ「データベースを対象とする標準の検査の作成」の説明にあるような結果セットの内容を検査します。

RowsCount：結果セット内の行数を数えます。

8 プロパティの期待結果を編集するには，まず期待結果を選択し，次に［**期待結果値の編集**］ボタンをクリックするか，［期待する値］カラムの値をダブルクリックします。

➤ 結果セットを対象とする **ColumnsCount** 検査や **RowsCount** 検査の期待結果は，プロパティ検査に対応する［**期待する値**］フィールドに表示されます。

これらのプロパティ検査の期待値を編集すると，スピン・ボックスが開きます。スピン・ボックスに現れる数字を変更します。

➤ 結果セットを対象とする **Content** 検査の場合，検査に対応する［**期待する値**］フィールドに＜**複雑な値**＞と表示されます。結果セットの内容は複雑なのでこのカラム内に表示できないのです。期待値を編集すると，［**チェックの編集**］ダイアログ・ボックスが開きます。［**チェックの指定**］タブ内で，クエリでキャプチャしたデータに基づいて，結果セットに対して実行する検査を選択できます。［**期待データの編集**］タブ内で，結果セット内にあるデータの期待結果を変更できます。

詳細については，298 ページ「［チェックの編集］ダイアログ・ボックスについて」を参照してください。

9 ［**OK**］**をクリックして**［データベース チェック］ダイアログ・ボックスを閉じます。

WinRunner は，現在のプロパティの値をキャプチャし，テストの「***exp***」フォルダに格納します。WinRunner は，データベース・クエリをテストの「***chklist***」フォルダに格納します。データベース・チェックポイントは **db_check** ステートメントとしてテスト・スクリプトに挿入されます。**db_check** 関数の詳細については，「**TSL リファレンス**」を参照してください。

[データベース チェックポイント] ダイアログ・ボックスのメッセージ

次のメッセージが［データベース チェック］または［データベース チェックポイント結果］ダイアログ・ボックス内の［期待する値］または［実際の値］フィールドの［プロパティ］表示枠に現れることがあります。

メッセージ	意味
複雑な値	選択されたプロパティ検査の期待値や実際の値が複雑すぎてカラムに表示できません。このメッセージは，内容の検査時に現れます。
次を取得できません	選択されたプロパティの期待値または実際の値をキャプチャできませんでした。

注：［データベース チェックポイント結果］ダイアログ・ボックスの詳細については，第 20 章「テスト結果の分析」を参照してください。

[データベース チェックポイント] ウィザードでの作業

［データベース チェックポイント］ウィザードは，ユーザ定義のデータベース・チェックポイントを作成するとき，または Data Junction で作業するときに，必ず開きます。データベース・チェックポイントは，SQL ステートメントを使って作成することもできます。SQL ステートメントを使用する場合，ユーザ定義のデータベース検査を作成し，ODBC（Microsoft Query）オプションを選びます。

ODBC/Microsoft Query モードまたは Data Junction モードで作業できます。最後に使ったツールによって，ODBC（Microsoft Query）または Data Junction の画面が開きます。ウィザードの最初の画面でモードを切り替えられます。

［データベース チェックポイント］ウィザードでは，以下のことができます。

- ODBC（Microsoft Query）モードと Data Junction モードの切り替え
- Microsoft Query を使用せずに SQL ステートメントを指定

- データベース・チェックポイント内の既存のクエリと変換の使用

ODBC（Microsoft Query）画面

［データベース チェックポイント］ウィザードには，ODBC（Microsoft Query）で作業するための画面が 3 つあります。これらの画面では以下のことができます。

- 一般オプションの設定
  - Data Junction モードへの切り替え
  - 新しいクエリの作成，既存のクエリの使用，SQL ステートメントの指定のどれかを選択
  - クエリ内の行数の制限
  - 手順を説明する画面の表示
- 既存のソース・クエリ・ファイルの選択
- SQL ステートメントの指定

ODBC（Microsoft Query）オプションの指定

ユーザ定義のデータベース・チェックポイントを作成する，または ODBC モードで作業する場合，次の画面が開きます。

次のオプションを選べます。

- ［**新規クエリの作成**］: Microsoft Query が開き，新しい ODBC ***.sql*** クエリ・ファイルを下に指定した名前で作成できます。クエリを定義すると次のようになります。
  - 標準のデータベース・チェックポイントを作成している場合，**db_check** ステートメントがテスト・スクリプトに挿入されます。
  - ユーザ定義のデータベース・チェックポイントを作成している場合，［データベース チェック］ダイアログ・ボックスが開きます。［データベース チェック］ダイアログ・ボックスの詳細については，287 ページ「データベースを対象とするユーザ定義の検査の作成」を参照してください。
- ［**既存のクエリをコピー**］: 既存の ODBC クエリをほかのクエリ・ファイルからコピーできる［**ファイルを選択**］画面がウィザードで開きます。詳細については，294 ページ「ソース・クエリ・ファイルの選択」を参照してください。
- ［**SQL ステートメントを指定**］:接続文字列と SQL ステートメントを指定できる［**SQL クエリーを指定**］画面がウィザードで開きます。詳細については，295 ページ「SQL ステートメントを指定」を参照してください。
- ［**新規クエリーファイル**］: このデータベース・チェックポイントに新しい ***.sql*** クエリ・ファイルの標準名を表示します。参照ボタンを使って別の ***.sql*** クエリ・ファイルを参照することもできます。
- ［**最大行数**］: このチェック・ボックスを選択して，データベース内で検査する最大の行数を入力します。このチェック・ボックスが選択されていない場合，上限はありません。このオプションは，**db_check** ステートメントにパラメータを追加します。詳細については「**TSL リファレンス**」を参照してください。
- ［**Microsoft Query の使用法を表示**］: 手順を説明する画面を表示します。

ソース・クエリ・ファイルの選択

このデータベース・チェックポイント内で既存のクエリ・ファイルを使う場合，以下の画面が開きます。

クエリ・ファイルのパス名を入力するか［**参照**］ボタンを使ってクエリ・ファイルを見つけます。クエリ・ファイルを選択したら，［**表示**］ボタンを使ってファイルを開いて表示できます。

- 標準のデータベース・チェックポイントを作成している場合，**db_check** ステートメントがテスト・スクリプトに挿入されます。
- ユーザ定義のデータベース・チェックポイントを作成している場合，［データベース チェック］ダイアログ・ボックスが開きます。［データベース チェック］ダイアログ・ボックスの詳細については，287 ページ「データベースを対象とするユーザ定義の検査の作成」を参照してください。

SQL ステートメントを指定

このデータベース・チェックポイント内で使用する SQL ステートメントを指定
作業用テストする場合，以下の画面が開きます。

この画面では，接続文字列と SQL ステートメントを指定しなければなりません。

- ［**接続文字列**］: 接続文字列を入力するか，［**作成**］をクリックして［データソースの選択］ダイアログ・ボックスを開きます。このダイアログ・ボックスでは，接続文字列をフィールドに挿入する ***.dsn*** ファイルを選択できます。
- ［**SQL**］: SQL ステートメントを入力します。

注：パラメータを含む SQL ステートメントを作成する場合，手順を説明する画面が開きます。SQL ステートメントのパラメータ化の詳細については，317 ページ「標準のデータベース・チェックポイントのパラメータ化」を参照してください。

次のことを行います。

- 標準のデータベース・チェックポイントを作成している場合は，**db_check** ステートメントがテスト・スクリプトに挿入されます。

- ユーザ定義のデータベース・チェックポイントを作成している場合は，［データベース チェック］ダイアログ・ボックスが開きます。［データベース チェック］ダイアログ・ボックスの詳細については，287 ページ「データベースを対象とするユーザ定義の検査の作成」を参照してください。

［データベース チェックポイント］ウィザード内の Data Junction 画面

［データベース チェックポイント］ウィザードには，Data Junction を使って作業するための画面が 2 つあります。これらの画面では以下のことができます。

- 一般オプションの設定
  - ODBC（Microsoft Query）モードへの切り替え
  - 新しい変換ファイルを作成するか，既存の変換ファイルを使用するか選択
  - 手順を解説する画面の表示
- 変換ファイルの指定

Data Junction オプションの設定

前回 Data Junction で作業を行った場合，あるいは Data Junction のみインストールされている場合，標準のデータベース・チェックポイントを初めて作成するときに，次の画面が開きます。

以下のオプションを選べます。

- ［**新規変換の作成**］: Data Junction を開き，新しい変換ファイルを作成できます。詳細については，322 ページ「Data Junction での変換ファイルの作成」を参照してください。変換ファイルを作成すると［データベース チェックポイント］ウィザード画面が再び開き，このファイルを指定できます。詳細については，297 ページ「Data Junction 変換ファイルの選択」を参照してください。
- ［**既存の変換を使用**］: 既存の変換ファイルを指定できる［**変換ファイルを選択してください**］画面がウィザードで開きます。詳細については，297 ページ「Data Junction 変換ファイルの選択」を参照してください。
- ［**Data Junction の使用法を表示**］（［**新規変換の作成**］が選択されている場合にのみ有効）: Data Junction で作業するための手順が表示されます。

Data Junction 変換ファイルの選択

Data Junction で作業する場合，以下のウィザード画面が開きます。

変換ファイルのパス名を入力するか［**参照**］ボタンを使って変換ファイルを見つけます。変換ファイルを選択したら，［**表示**］ボタンを使ってファイルを開いて表示できます。

次のオプションも選べます。

- **[変換をテストのフォルダにコピー]**：指定した変換ファイルをテスト・フォルダにコピーします。
- **[最大行数]**：このチェック・ボックスを選択して，データベース内で検査する最大の行数を入力します。このチェック・ボックスが選択されていない場合，上限はありません。

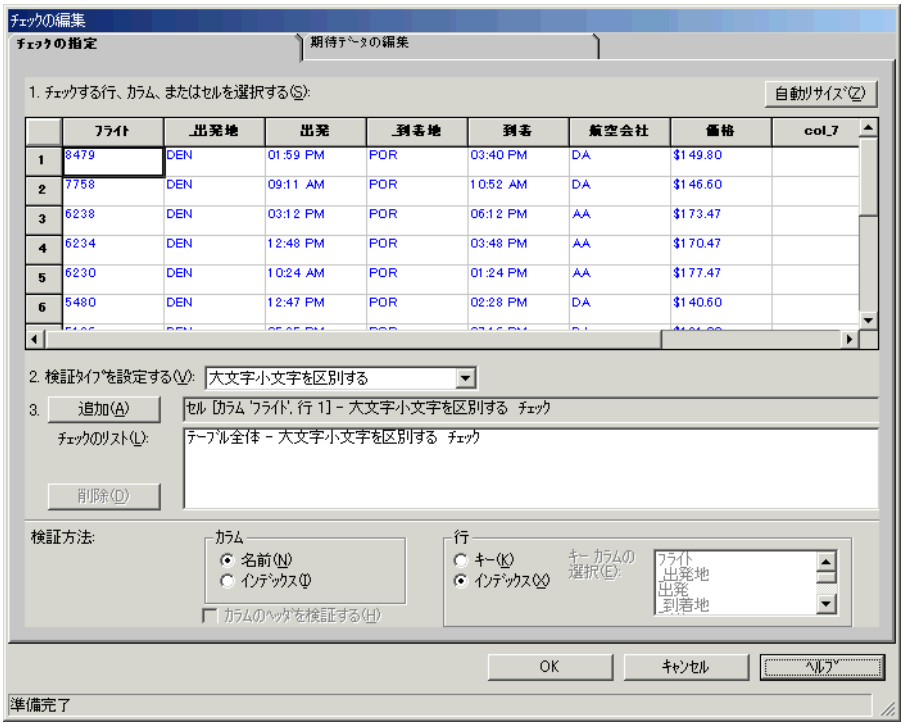
オプションの選択を終えたら，次の手順を実行します。

- 標準のデータベース・チェックポイントを作成している場合は，**db_check** ステートメントがテスト・スクリプトに挿入されます。
- ユーザ定義のデータベース・チェックポイントを作成している場合は，［データベース チェック］ダイアログ・ボックスが開きます。［データベース チェック］ダイアログ・ボックスの詳細については，287 ページ「データベースを対象とするユーザ定義の検査の作成」を参照してください。

［チェックの編集］ダイアログ・ボックスについて

［チェックの編集］ダイアログ・ボックスを使って，検査対象セル，使用する検証方法，検証のタイプを指定できます。また，検査に含まれているデータベースのセルの期待データも編集できます。（［チェックの編集］ダイアログ・

ボックスの開き方については，287 ページ「データベースを対象とするユーザ定義の検査の作成」を参照してください。）



［チェックの指定］タブ内で，データベース・チェックリスト内に保存されている次の情報を指定できます。

- 検査するデータベースのセル
- 検証方法
- 検証のタイプ

1 つのカラムの結果セットを対象とする検査を作成している場合，［チェックの編集］ダイアログ・ボックスの［チェックの指定］タブの内容は，上記で示している内容とは異なります。詳細については，301 ページ「単一カラムの結果セットに対する検証方法の指定」を参照してください。

検査するセルの指定

［**チェックの一覧**］ボックスには，検証のタイプも含めて，実行されるすべての検査が表示されます。チェックポイントの［**チェックの編集**］ダイアログ・ボックスを初めて開くと，標準の検査が表示されます。

- 複数のカラムの結果セットに対する標準の検査は，カラム名と行インデックスに基づく結果セット全体が対象になります。
- 1 つのカラムの結果セットに対する標準の検査は，行の位置に基づく結果セット全体が対象になります。

注：結果セットに同じ名前の複数のカラムが含まれていると，WinRunner は，同じ名前のカラムを認識せず，それらのカラムに対して検査を実行しません。この場合，データベースに対してユーザ定義検査を作成し，カラム・インデックス・オプションを選択します。

標準設定を受け入れたくなければ，実行する検査を指定する前に，標準の検査を削除しなければなりません。［**チェックの一覧**］ボックスで［全テーブル - ‘大小文字の区別 ’ チェック］というエントリを選択し，［**削除**］ボタンをクリックします。あるいは，［**チェックの一覧**］ボックスでこのエントリをダブルクリックします。強調表示されている検査を削除するかどうか尋ねる WinRunner のメッセージが出たら［**はい**］をクリックします。

次に，実行する検査を指定します。選んだセルのグループごとに異なった検証のタイプを選べます。そのため，セルを選択する前に検証のタイプを指定します。詳細については，302 ページ「検証のタイプの指定」を参照してください。

内容の検査を実行するセルを強調表示します。次に，ツールバーの［**追加**］ボタンをクリックして，これらのセルに検査を追加します。あるいは，次のようにもできます。

- セルを検査するにはセルをダブルクリックします。
- 行の中にあるすべてのセルを検査するには，行ヘッダをダブルクリックします。
- カラム内のすべてのセルを検査するには，カラム・ヘッダをダブルクリックします。
- 結果セット全体を検査するには，左上角をダブルクリックします。

検査するセルの説明が［**チェックの一覧**］ボックスに現れます。

検証方法の指定

検証方法を選択し，WinRunner の結果セットでのカラムと行の認識方法を制御できます。検証方法は結果セット全体に適用されます。検証方法の指定方法は，複数のカラムの結果セットと 1 つのカラムの結果セットで異なります。

複数のカラムの結果セットに対する検証方法の指定

カラム

- ［**名前**］（標準設定）：WinRunner はカラム名に基づいて対象を探します。テーブル内でカラムの位置を変更しても，結果は不一致にはなりません。
- ［**インデックス**］：WinRunner は，インデックス，位置，カラムに基づいて，選択したカラムを探します。テーブル内でカラムの位置を変更すると結果は不一致になります。テーブル内に同じ名前の複数のカラムが含まれている場合には，このオプションを選択します。詳細については，300 ページの注を参照してください。このオプションを選択すると［**カラムのヘッダを検証する**］チェック・ボックスが使用できます。これを有効にすると，セルのほかにカラム・ヘッダも検査できます。

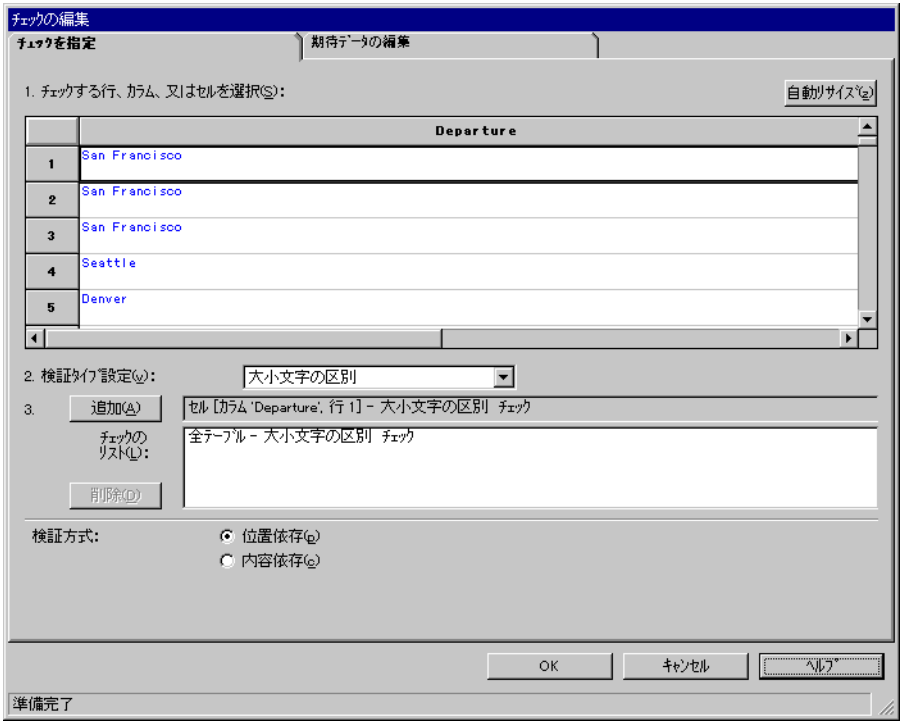
行

- ［**キー**］：WinRunner は，結果セット内のすべてのカラムの名前をリストする［**キーカラムの選択**］リスト・ボックスで指定されたキー（複数も可）に基づいて対象の中の行を探します。行の位置を移動しても不一致とはなりません。キー選択は一意の行を特定せず，最初に一致する行のみ検査されます。
- ［**インデックス**］（標準設定）：WinRunner は，インデックス，位置，行に基づいて対象を探します。行の位置を移動すると，不一致となります。

単一カラムの結果セットに対する検証方法の指定

単一カラムの結果セットの［チェックの指定］タブにある［検証方式］ボックスは，複数カラムの結果セットのそれとは異なります。単一カラムの結果セッ

トの標準の検査では，結果セット全体がその対象となります。検査では，行の位置が使われ，大文字と小文字が区別されます。



- ［**位置依存**］: WinRunner は，カラム内の項目の場所に基づいて対象を検査します。
- ［**内容依存**］: WinRunner はカラム内の項目の内容に基づいて対象を検査します。このとき，リスト内の場所は無視されます。

検証のタイプの指定

WinRunner では，様々な方法で結果セットの内容を検証できます。セルの選択に合せて，異なる検証のタイプを選択できます。

- ［**大文字小文字を区別する**］（標準）: WinRunner は選択したテキストの内容を比較します。期待データと実際のデータ間の大文字と小文字の違いやテキスト内容の違いはすべて不一致となります。

- ［**大文字小文字を区別しない**］：**WinRunner** は選択したテキストの内容を比較します。期待データと実際のデータ間のテキスト内容の違いだけ不一致になります。
- ［**数値**］：WinRunner は，選択されたデータを数値として評価します。WinRunner は，例えば「2」と「2.00」を同じ数字と認識します。
- ［**数値の範囲**］：WinRunner は，選択されたデータを数値の範囲と比較します。最小値と最大値には，どちらも任意の実数を指定します。この比較は，実際のテーブル・データを，期待結果ではなく，指定された範囲と比較をする点がテキストおよび数値内容の検証とは異なります。

注：このオプションは，数値で始まらない文字列をすべて不一致とします。文字列を e で開始すると数値として認識します。

- ［**大文字小文字を区別する，スペースを無視する**］：WinRunner は，空白文字の違いを無視して，大文字と小文字または内容に基づいてフィールドのデータを検査します。WinRunner は，大文字と小文字の違いやテキスト内容の違いをすべて不一致として報告します。
- ［**大文字小文字を区別しない，スペースを無視する**］：WinRunner は，大文字と小文字の違いと空白文字の違いを無視して，内容に基づいてフィールドの内容を検査します。WinRunner は，内容の違いだけを不一致として報告します。

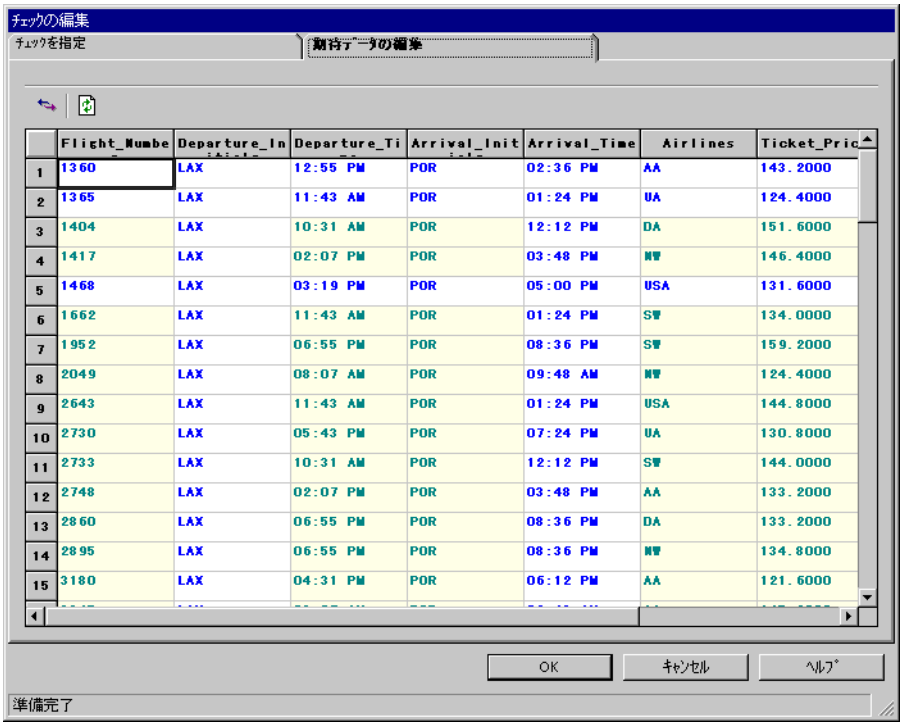
［**OK**］をクリックして，［**チェックの編集**］ダイアログ・ボックスの両方のタブで行った変更を保存します。ダイアログ・ボックスが閉じ，［**データベースチェック**］ダイアログ・ボックスに戻ります。

期待データの編集

結果セット内の期待データを編集するには，［**期待データの編集**］タブをクリックします。［チェックの指定］タブで変更を加えた場合は，［**テーブルを再ロード**］をクリックしてチェックリストからテーブル選択を再ロードできます。保存したデータを再ロードするかどうかを確認する WinRunner メッセージが表示されます。［はい］をクリックします。

［**チェックの指定**］タブの変更を保存した後で［**チェックの編集**］ダイアログ・ボックスを再び開くと，［**期待データの編集**］タブにテーブルが色分けされて

表示されます。検査に含まれるセルは，白い背景に青で表示されます。検査から除外されるセルは，黄色い背景に緑で表示されます。



セル内のデータの期待値を編集するには，セル内をダブルクリックします。セル内にカーソルが現れます。セルの内容を必要に応じて変更します。［**OK**］をクリックして，［チェックの編集］ダイアログ・ボックスの両方のタブで行った変更を保存します。ダイアログ・ボックスが閉じ，［データベース チェック］ダイアログ・ボックスに戻ります。

標準のデータベース・チェックポイントの変更

既存の標準のデータベース・チェックポイントを次のように変更できます。

- チェックリストを共有フォルダに保存して，ほかのユーザもチェックリストを使えるようにできます。
- 既存のチェックリストで検査するデータベース・プロパティを変更できます。

➤ 既存のチェックリスト内のクエリを変更できます。

注：データベース・チェックリストだけでなく，データベース・チェックポイントの期待結果も変更できます。詳細については，315 ページ「標準のデータベース・チェックポイントの期待結果の変更」を参照してください。

データベース・チェックリストを共有フォルダに保存

標準設定により，データベース・チェックポイントのチェックリストは，現在のテストのフォルダに格納されます。複数のテストで同じチェックリストを使用できるように，チェックリストを共有フォルダに置いて幅広いアクセスを可能にします。

注：*.sql* ファイルは共有データベース・チェックリスト・フォルダには保存されません。

WinRunner が共有チェックリストを格納する標準のフォルダは，「**WinRunner のインストール・フォルダ /chklist**」になります。違うフォルダを選ぶには，［一般オプション］ダイアログ・ボックスの［フォルダ］カテゴリにある［**共有チェックリスト**］ボックスを使います。詳細については，第 22 章「グローバル・テスト・オプションの設定」を参照してください。

データベース・チェックリストを共有フォルダに保存するには，次の手順を実行します。

1 ［**挿入**］＞［**データベース チェックリストの編集**］を選びます。

［チェックリストを開く］ダイアログ・ボックスが開きます。

2 データベース・チェックリストを選択し［**OK**］をクリックします。GUI チェックリストには「.ckl」，**データベース**・チェックリストには「.cdl」という拡張子が付いています。GUI チェックリストの詳細については，143 ページ「GUI チェックリストの変更」を参照してください。

［チェックリストを開く］ダイアログ・ボックスが閉じます。［データベースチェックリストを編集］ダイアログ・ボックスには，選択したデータベース・チェックリストが表示されます。

3 **［名前を付けて保存］**をクリックしてチェックリストを保存します。

［チェックリストの保存］ダイアログ・ボックスが開きます。

4 ［**適用範囲**］の下の［共有］をクリックします。**共有**チェックリストの名前を入力します。［**OK**］をクリックしてチェックリストを保存し，ダイアログ・ボックスを閉じます。

5 **［OK］をクリックして**［データベース チェックリストを編集］ダイアログ・ボックスを閉じます。

データベース・チェックリストの編集

既存のデータベース・チェックリストを編集できます。データベース・チェックリストには，データベースの **msqr*.sql** クエリ・ファイルまたは ***.djs** 変換ファイルへの参照と，検査するプロパティへの参照しか含まれていません。これらのプロパティの値の期待結果は含まれていません。

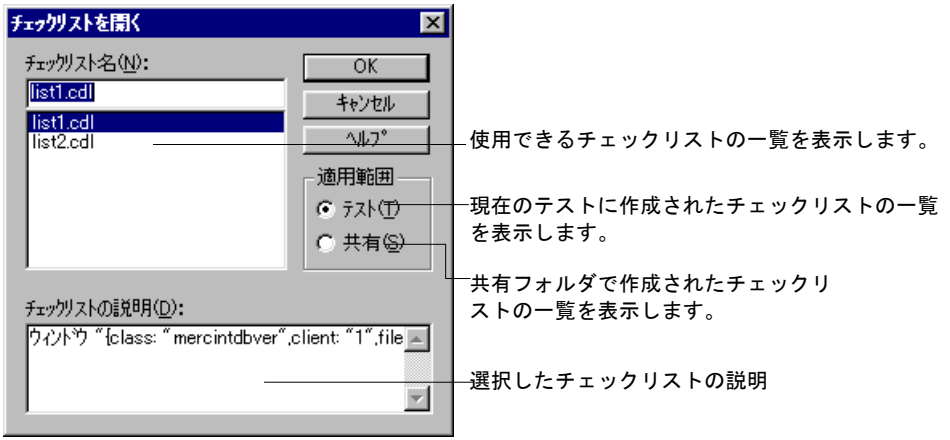
データベース・チェックリストを編集して，検査するデータベース内のプロパティを変更したほうがよいでしょう。

既存のデータベース・チェックリストを編集するには，次の手順を実行します。

1 ［**挿入**］＞［**データベース チェックリストの編集**］を選びます。［チェックリストを開く］ダイアログ・ボックスが開きます。

2 現在のテストのチェックリストの一覧が表示されます。共有フォルダ内のチェックリストを見るには［**共有**］をクリックします。

データベース・チェックリストの共有の詳細については，305 ページ「データベース・チェックリストを共有フォルダに保存」を参照してください。

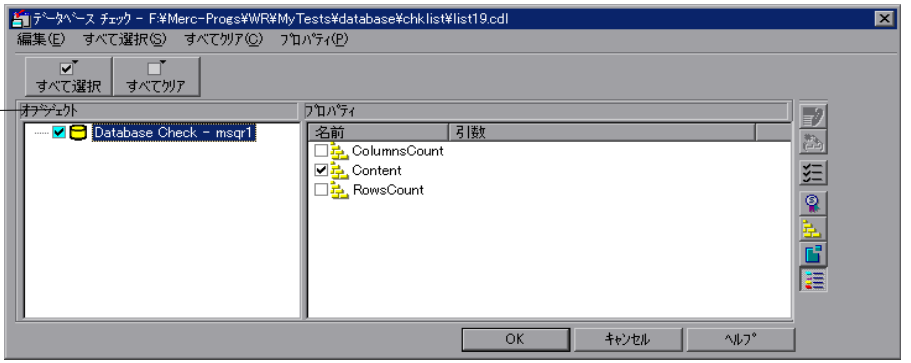


3 データベース・チェックリストを選択します。

4 ［**OK**］をクリックします。

［チェックリストを開く］ダイアログ・ボックスが閉じて，［データベース チェックリストを編集］ダイアログ・ボックスが開き，選択したチェックリストが表示されます。

［**オブジェクト**］表示枠には，「Database Check」と，データベース・チェックポイントに含まれる ***.sql クエリ・ファイルまたは *.djs** 変換ファイル**の名前**が含まれます。［**プロパティ**］表示枠には，データベースを対象に実行できる様々な種類の検査の一覧が表示されます。チェックマークは，その項目が選択され，チェックポイントに含まれていることを示します。



［**修正**］ボタンを使って，データベース・チェックポイントを変更できます。詳細については，309 ページ「既存のデータベース・チェックポイント内のクエリの変更」を参照してください。

［**プロパティ**］表示枠で，データベース・チェックリストを編集して，次の検査の種類を含めたり削除したりできます。

ColumnsCount：結果セット内のカラム数を数えます。

Content：結果セットの内容を検査します。詳細については，284 ページ「データベースを対象とする標準の検査の作成」を参照してください。

RowsCount：結果セット内の行数を数えます。

5 次のうちいずれかの方法でチェックリストを保存します。

➤ 既存の名前でチェックリストを保存するには，［**OK**］をクリックして［データベース チェックリストを編集］ダイアログ・ボックスを閉じます。WinRunner がメッセージを表示して既存のチェックリストを上書きするかどうか尋ねてきたら［**OK**］をクリックします。

➤ 別の名前でチェックリストを保存するには，［**名前を付けて保存**］ボタンをクリックします。［チェックリストの保存］ダイアログ・ボックスが開いたら，新しい名前を入力するか標準の名前を使います。［**OK**］をクリックします。［**名前を付けて保存**］ボタンをクリックせずに［**OK**］をクリックして［データベース チェックリストを編集］ダイアログ・ボックスを閉じると，WinRunner は，自動的に現在のチェックリスト名で保存します。

新しいデータベース・チェックポイント・ステートメントは，テスト・スクリプトに挿入されません。

注：［検証］実行モードでテストを実行する前に，加えた変更にあわせてチェックリスト内の期待結果を更新しなければなりません。期待結果を更新するには，［更新］実行モードでテストを実行します。［更新］実行モードでのテスト実行の詳細については，422 ページ「WinRunner のテスト実行モード」を参照してください。

既存のデータベース・チェックポイント内のクエリの変更

［データベース チェックリストを編集］ダイアログ・ボックスで既存のデータベース・チェックリスト内のクエリを変更できます。クエリの変更は，データベースをネットワーク上の新しい位置に移動した場合などに必要になります。クエリの変更には，クエリを作成したときと同じツールを使わなければなりません。

ODBC/Microsoft Query で作成したクエリの変更

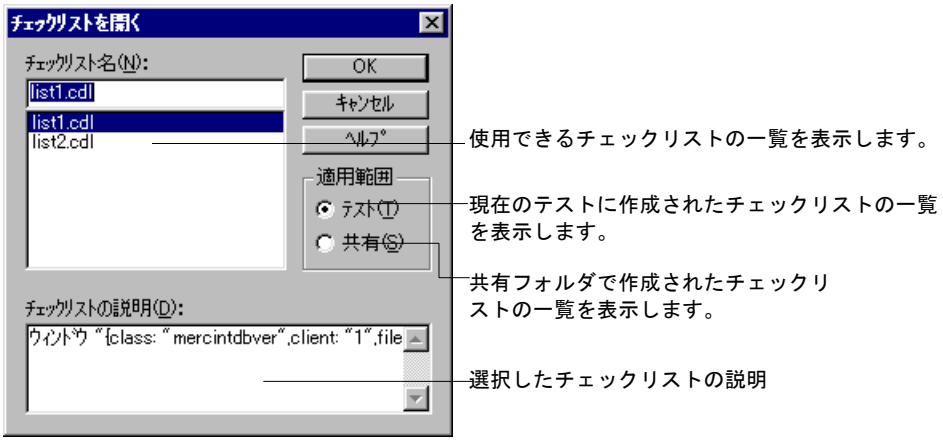
［データベース チェックリストを編集］ダイアログ・ボックスでは，ODBC/Microsoft Query で作成したクエリを変更できます。

ODBC/Microsoft Query で作成したデータベース・チェックポイントを変更するには，次の手順を実行します。

1 ［**挿入**］＞［**データベース チェックリストの編集**］を選び，［チェックリストを開く］ダイアログ・ボックスを開きます。

2 現在のテストのチェックリストの一覧が表示されます。共有フォルダ内のチェックリストを見るには［**共有**］をクリックします。

データベース・チェックリストの共有の詳細については，305 ページ「データベース・チェックリストを共有フォルダに保存」を参照してください。



3 データベース・チェックリストを選択します。

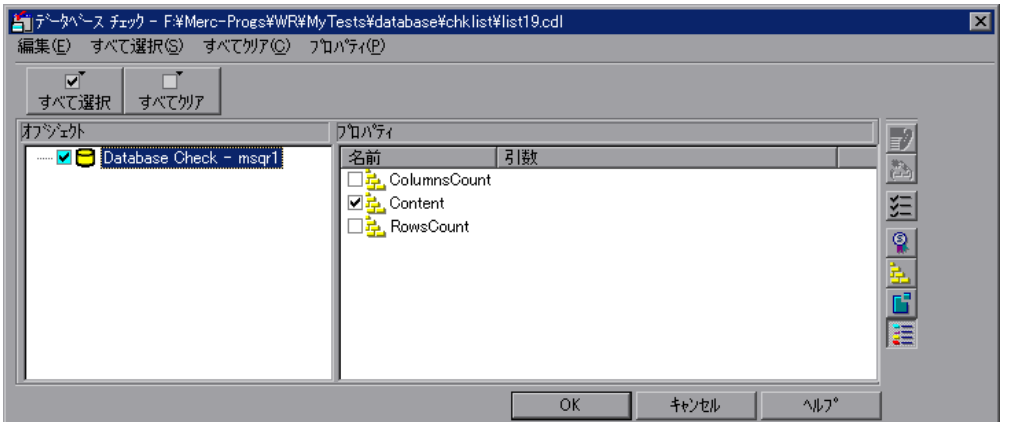
4 **［OK］**をクリックします。

［チェックリストを開く］ダイアログ・ボックスが閉じて，［データベース チェックリストを編集］ダイアログ・ボックスが開き，選択したチェックリストが表示されます。

［**オブジェクト］表示枠**には，「Database Check」と，データベース・チェックポイントに含まれる ***.sql* クエリ・ファイルの名前**が含まれます。

［**プロパティ**］表示枠には，データベースを対象に実行できる様々な種類の検査の一覧が表示されます。チェックマークは，その項目が選択されていて，チェックポイントに含まれていることを示します。検査対象のプロパティの変

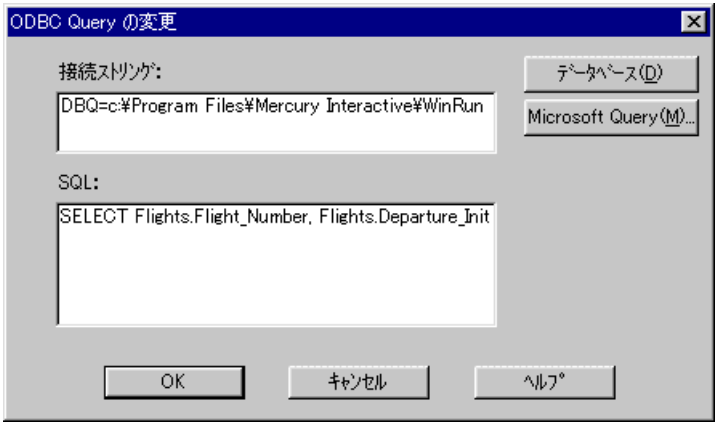
更の詳細については，307 ページ「データベース・チェックリストの編集」を参照してください。



修正

5 **[オブジェクト]** カラムでクエリ・ファイルまたは変換ファイルの名前を強調表示して［**修正**］をクリックします。

［ODBC Query の変更］ダイアログ・ボックスが開きます。



6 接続文字列や SQL ステートメントを変更して，ODBC クエリを変更します。［**データベース**］をクリックして［ODBC Select Data Source］ダイアログ・ボックスを開きます。このダイアログ・ボックスでは，接続文字列をフィールドに挿入する ***.dsn** ファイルを選択できます。［**Microsoft Query**］をクリックして Microsoft Query を開けます。

7 ［**OK**］をクリックして［チェックリストを編集］ダイアログ・ボックスに戻ります。

8 ［**OK**］をクリックして［チェックリストを編集］ダイアログ・ボックスを閉じます。

注：このチェックリストを使うすべてのテストは，［検証］実行モードで実行する前に，［更新］実行モードで実行しなければなりません。詳細については，433 ページ「期待結果を更新するためのテスト実行」を参照してください。

Data Junction で作成したクエリの変更

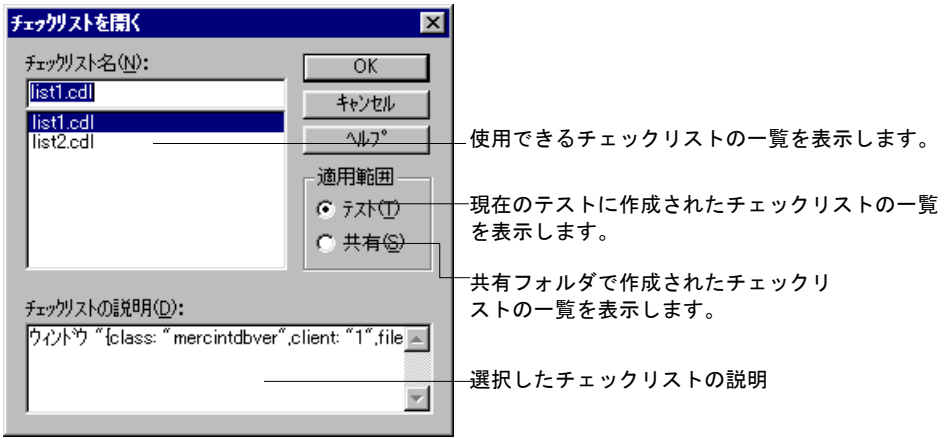
データベース・チェックポイントで使う Data Junction 変換ファイルは，Data Junction で直接変更できます。変換ファイルのパス名を表示するには，以下の手順に従ってください。

データベース・チェックポイント内の Data Junction 変換ファイルのパス名を見るには，次の手順を実行します。

1 ［**挿入**］＞［**データベース チェックリストの編集**］を選びます。［チェックリストを開く］ダイアログ・ボックスが開きます。

2 現在のテストのチェックリストの一覧が表示されます。共有フォルダ内のチェックリストを見るには［**共有**］をクリックします。

データベース・チェックリストの共有の詳細については，305 ページ「データベース・チェックリストを共有フォルダに保存」を参照してください。



3 データベース・チェックリストを選択します。

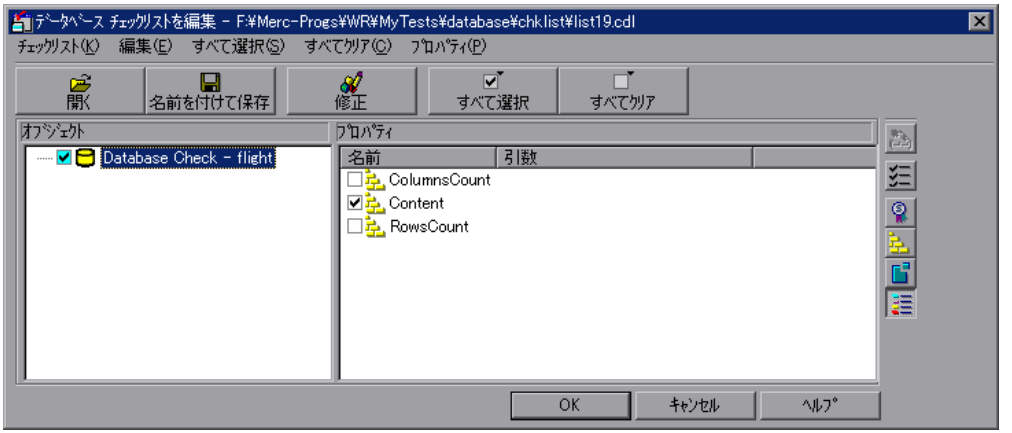
4 **[OK]** をクリックします。

［チェックリストを開く］ダイアログ・ボックスが閉じ，［データベース チェックリストを編集］ダイアログ・ボックスが開いて，選択したチェックリストが表示されます。

［**オブジェクト**］表示枠には，「Database Check」と，データベース・チェックポイントに含まれる ***.djs 変換ファイルの名前**が含まれます。

［**プロパティ**］表示枠には，データベースを対象に実行できる様々な種類の検査の一覧が表示されます。チェックマークは，その項目が選択されていて，チェックポイントに含まれていることを示します。検査対象のプロパティの変

更の詳細については，307 ページ「データベース・チェックリストの編集」を参照してください。



5 ［**オブジェクト**］カラムで変換ファイルの名前を強調表示して［**修正**］をクリックします。

Data Junction を使って変換ファイルおよび変換ファイルのパス名を変更するように促すメッセージが表示されます。

6 ［**OK**］をクリックしてメッセージを閉じ，［チェックリストを編集］ダイアログ・ボックスに戻ります。

7 ［**OK**］をクリックして［チェックリストを編集］ダイアログ・ボックスを閉じます。

8 Data Junction で直接変換ファイルを変更します。

注：このチェックリストを使うすべてのテストは，［検証］実行モードで実行する前に，［更新］実行モードで実行しなければなりません。詳細については，433 ページ「期待結果を更新するためのテスト実行」を参照してください。

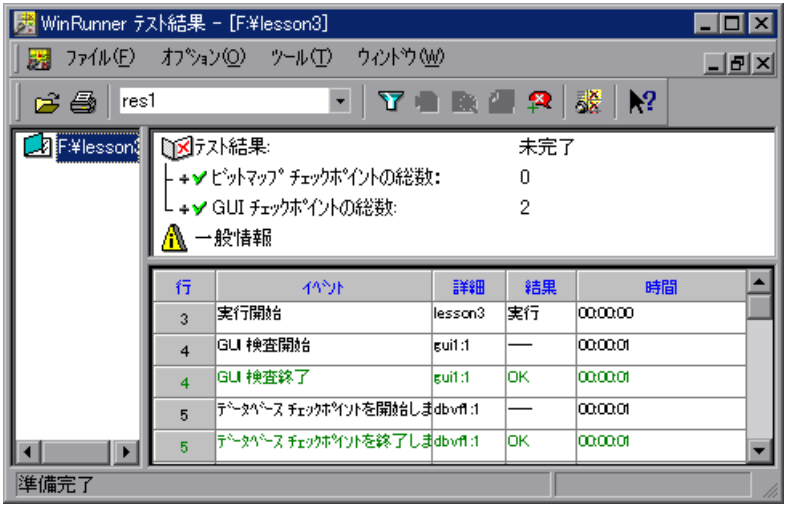
標準のデータベース・チェックポイントの期待結果の変更

チェックポイント内でプロパティ検査の期待結果を変更して，既存の標準のデータベース・チェックポイントの期待結果を変更できます。この変更はテスト実行前または実行後に行えます。

既存のデータベース・チェックポイントの期待結果を変更するには，次の手順を実行します。

1 ［**ツール**］＞［**テスト結果**］を選ぶか，または［**テスト結果**］をクリックします。

［**WinRunner テスト結果**］ウィンドウが開きます。結果の保存先に **exp** を選択します。

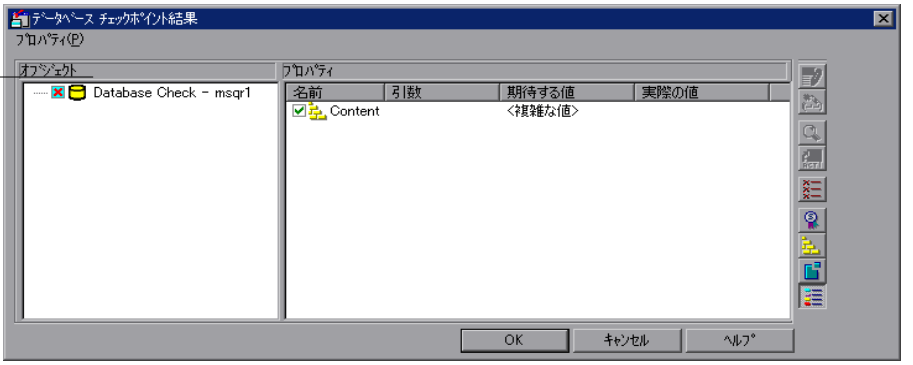


2 「**データベース キャプチャ終了**」エントリを探してデータベース・チェックポイントを見つけます。

注：WinRunner レポート・ビューを使用している場合は，［**TSL を表示**］**ボタンを使って**，強調表示されている行番号の位置が表示されるようにテスト・スクリプトを開けます。

---

3 「**データベース キャプチャ終了**」エントリを選択して表示します。［**データベース チェックポイント結果**］ダイアログ・ボックスを開きます。



4 変更する期待結果のプロパティ検査を選択します。［**期待結果値の編集**］ボタンをクリックします。［**期待する値**］カラム内で，必要に応じて値を変更します。［OK］をクリックしてダイアログ・ボックスを閉じます。

注：

データベース・チェックポイントを作成している間でも，プロパティ検査の期待値を変更できます。詳細については，287 ページ「データベースを対象とするユーザ定義の検査の作成」を参照してください。

また，テスト実行後に，データベース・チェックポイントの期待値を実際の値に更新できます。詳細については 487 ページ「WinRunner レポート・ビューでのチェックポイントの期待結果の更新」を参照してください。

［テスト結果］ウィンドウを使った作業の詳細については，第 2 章「統一レポート・ビューでのテスト結果の分析」を参照してください。

標準のデータベース・チェックポイントのパラメータ化

ODBC（Microsoft Query）を使用して標準のデータベース・チェックポイントを作成する場合，パラメータを SQL ステートメントに追加してチェックポイントをパラメータ化できます。これは，SQL ステートメントでクエリの変更を定義するクエリに対してデータベース・チェックポイントを作成するのに便利です。例えば，サンプルのフライト予約アプリケーションで作業している場合，クエリを作成するときに，月曜日にデンバーから出発するすべてのフライトの記録を選択するとします。さらに，同じクエリを使って火曜日にサンフランシスコから出発するすべてのフライトも検査するとします。この場合，新しいクエリを作成したり，この変更を反映するために既存のクエリ内の SQL ステートメントを書き直したりする必要はなく，SQL ステートメントをパラメータ化して，出発地の値にパラメータを使えます。パラメータは「Denver」または「San Francisco」という値で置き換えることができます。同様に、曜日をパラメータ化して、「Monday」や「Tuesday」という値で置き換えることもできます。

パラメータ化されたクエリについて

パラメータ化されたクエリとは，WHERE 句の少なくとも 1 つのフィールドがパラメータ化されているクエリです。パラメータ化されているフィールドの値は疑問符（**?**）で示されます。例えば，以下に示す SQL ステートメントは，サンプルのフライト予約アプリケーションのデータベースに対するクエリに基づいています

```
SELECT Flights.Departure, Flights.Flight_Number, Flights.Day_Of_Week
FROM Flights
WHERE (Flights.Departure=?) AND (Flights.Day_Of_Week=?)
```

- **SELECT**：クエリに含むカラムを定義します。
- **FROM**：データベースのパスを指定します。
- **WHERE**（オプション）：条件またはクエリで使用するフィルタを指定します。
- **Departure**：フライトの出発地を表すパラメータ。
- **Day_Of_Week**：フライトの曜日を表すパラメータ。

パラメータ化されたクエリを実行するには，パラメータに値を指定する必要があります。

Microsoft Query をお使いの方へ：クエリの作成に Microsoft Query をお使いの場合は，パラメータが括弧符号で指定されます。Microsoft Query で［SQL］ステートメントを生成すると，括弧が疑問符（**?**）で置き換えられます。Microsoft Query の［SQL］ボタンをクリックすると，クエリに追加した基準に基づいて生成される SQL ステートメントを見ることができます。

パラメータ化されたデータベース・チェックポイントの作成

パラメータ化されたクエリを使って，パラメータ化されたチェックポイントを作成します。データベース・チェックポイントを作成する場合，テスト・スクリプトに **db_check** ステートメントを挿入します。チェックポイント内の SQL ステートメントをパラメータ化すると，**db_check** 関数には 4 つ目の任意の引数，**parameter_array 引数**ができます。次に示すようなステートメントがテスト・スクリプトに挿入されます。db_check("list1.cdl", "dbvf1", NO_LIMIT, dbvf1_params);

parameter_array 引数には，パラメータ化されたチェックポイントのパラメータに代わる値が含まれます。

WinRunner はテストの記録時に期待結果セットをキャプチャできません。通常のデータベース・チェックポイントとは異なり，パラメータ化されたチェックポイントを記録するには，期待結果セットをキャプチャするというステップが必要になります。したがって，配列ステートメントを使ってパラメータを置き換える値を追加しなければなりません。配列ステートメントは，次のようなものになります。

```
dbvf1_params[1] = “Denver”;
dbvf1_params[2] = “Monday”;
```

配列ステートメントは，テスト・スクリプトの **db_check** ステートメントの前に挿入します。テストを「検証」モードで実行する前に，まずテストを「更新」モードで実行して，期待結果セットをキャプチャしておく必要があります。

パラメータ化した SQL ステートメントを db_check ステートメントに挿入するには，次の手順を実行します。

1 以下のどちらかの方法でパラメータ化されたSQLステートメントを作成します。

- Microsoft Query でクエリを定義したら，条件を追加します。条件の値は一連の大括弧（**[ ]**）として指定します。条件を追加したら，［**ファイル**］＞［**プログラムを終了し，WinRunner に戻ります**］をクリックします。WinRunner に戻るまでに数秒かかることがあります。
- ODBC で作業している場合，各パラメータの代わりに疑問符（**?**）を使って，パラメータ化した SQL ステートメントを［データベース チェックポイント］ウィザードに入力します。詳細については，295 ページ「SQL ステートメントを指定」を参照してください。

2 データベース・チェックポイントの作成を終了します。

- 「**標準**」のデータベース・チェックポイントを作成している場合，WinRunner は，データベース・クエリをキャプチャします。
- 「**ユーザ定義**」のデータベース・チェックポイントを作成している場合，［データベース チェック］ダイアログ・ボックスが開きます。データベースを対象に実行する検査を選択できます。詳細については，287 ページ「データベースを対象とするユーザ定義の検査の作成」を参照してください。［データベース チェック］ダイアログ・ボックスを閉じると，WinRunner はデータベース・クエリをキャプチャします。

注 :「**ユーザ定義**」のデータベース・チェックポイントを作成している場合，［データベース チェック］ダイアログ・ボックスを閉じるときに，「The expected value of one or more selected checks is not valid. Continuing might cause these checks to fail. Do you wish to modify your selection?（**選択された 1 つまたは複数の検査の値が有効ではありません。このまま続行すると，失敗する可能性があります。選択を変更しますか？**）」というメッセージが表示されます。

［**No**］**をクリックします**（このメッセージは，ダイアログ・ボックスの［期待する値］カラムの下に＜次を取得できません＞が表示されるために現れます）。

実際のところ，WinRunner がデータベース・クエリのキャプチャを終了するのは値を指定して［更新］実行モードでテストを実行した後です）。［データベース チェック］ダイアログ・ボックスのメッセージの詳細については，291 ページ「［データベース チェックポイント］ダイアログ・ボックスのメッセージ」を参照してください。

---

3 メッセージ・ボックスから手順を実行するよう指示されます。この手順は，以下でも解説されています。［**OK**］をクリックしてメッセージ・ボックスを閉じます。

WinRunner のウィンドウが再び表示され，以下のような **db_check** ステートメントがテスト・スクリプトに挿入されます。

```
db_check("list1.cdl", "dbvf1", NO_LIMIT, dbvf1_params);
```

4 SQL ステートメント内の変数に値を提供する配列を作成して，これらのステートメントを **db_check** ステートメントの前に挿入します。例えば，以下のような行をテスト・スクリプトに挿入します。

```
dbvf1_params[1] = “Denver”;
dbvf1_params[2] = “Monday”;
```

この配列は，317 ページの SQL ステートメント内の疑問符（**?**）を新しい値と置き換えます。以下に示す TSL で配列を追加する際のガイドラインに従って，SQL ステートメントをパラメータ化します。

5 ［更新］実行モードでテストを実行し，これらの値を使って SQL ステートメントを更新します。

この作業を終えたら，［検証］実行モードで SQL ステートメントを使用してテストを実行できます。SQL ステートメント内のパラメータを変更するには，TSL で配列を変更します。

SQL ステートメントをパラメータ化するためのガイドライン

db_check ステートメント内の SQL ステートメントをパラメータ化する場合，以下のガイドラインに従います。

- カラムが数値であれば，パラメータの値としてテキスト文字列または数字を指定できます。
- カラムとパラメータの値がテキストの場合，単純なテキスト文字列を指定できます。
- カラムがテキストでパラメータの値が数字の場合，「'100'」のよう単一な引用符（**' '**）で括ります。引用符で括らない場合，構文エラーのメッセージが表示されます。

- 日付，時間，タイム・スタンプの場合は，以下のように特別な構文を使って指定します。

日付	{d '1999-07-11'}
時間	{t '19:59:27'}
タイム・スタンプ	{ts '1999-07-11 19:59:27'}

注：日付と時間の形式は，ODBC ドライバによって異なることがあります。

データベースの指定

データベース・チェックポイント作成中，どのデータベースを検査するのか指定しなければなりません。次のツールを使って，検査するデータベースを指定できます。

- ODBC/Microsoft Query
- Data Junction（標準のデータベース・チェックポイントのみ）

ODBC/Microsoft Query でのクエリの作成

Microsoft Query を使ってデータ・ソースを選択して，データ・ソース内のクエリを定義したり，接続文字列や SQL ステートメントを手作業で定義したりできます。

Microsoft Query を使わずに ODBC でクエリを作成するには，データベース・チェックポイント・ウィザードで，接続文字列と SQL ステートメントを指定します。詳細については，295 ページ「SQL ステートメントを指定」を参照してください。

データソースを選び Microsoft Query でクエリを定義するには，次の手順を実行します。

1. 新しいデータ・ソースまたは，既存のデータ・ソースを選びます。
2. クエリを定義します。

注：生成する **db_check** ステートメント内の SQL ステートメントをパラメータ化する場合，Microsoft Query 8.00 のウィザードの最終画面で［**Microsoft Query でデータの表示またはクエリの編集を行う**］をクリックします。320 ページ「SQL ステートメントをパラメータ化するためのガイドライン」の指示に従ってください。

---

3 クエリの定義を終えたら次の手順を実行します。

- バージョン 2.00 の場合，［**ファイル**］＞［**プログラムを終了し，WinRunner に戻ります**］を選んで Microsoft Query を閉じ WinRunner に戻ります。
- バージョン 8.00 の場合，クエリ・ウィザードの最終画面で［**プログラムを終了し，WinRunner に戻ります**］＞［**完了**］をクリックして Microsoft Query を終了します。または，［**Microsoft Query でデータの表示またはクエリの編集を行う**］＞［**完了**］をクリックします。データの表示後または編集後に［**ファイル**］＞［**プログラムを終了し，WinRunner に戻ります**］を選んで Microsoft Query を閉じ，WinRunner に戻ります。

4 WinRunner でデータベース・チェックポイントの作成を続けます。

- データベースを対象とする標準の検査を作成する場合，287 ページのステップ 4 から始まる手順に従ってください。
- データベースを対象とするユーザ定義の検査を作成する場合，289 ページのステップ 6 から始まる手順に従ってください。

Microsoft Query の作業の詳細については，Microsoft Query のマニュアルを参照してください。

Data Junction での変換ファイルの作成

Data Junction を使って，データベースをターゲット・テキスト・ファイルに変換する変換ファイルを作成できます。WinRunner は，Data Junction のバージョン 6.5 と 7.0 をサポートします。

Data Junction で変換ファイルを作成するには，次の手順を実行します。

1 ソース・データベースを指定して，そのデータベースに接続します。

2 ASCII（区切り文字付き）のターゲットとなる言語タイプを選択し，ターゲット・ファイルを指定して，そのファイルに接続します。［Replace File/Table］出力モードを選びます。

注：Data Junction のバージョン 7.0 で作業しており，ソース・データベースには区切り文字（CR，LF，タブ）を含む値がある場合，［Target Properties］ダイアログ・ボックスで **TransliterationIn** プロパティの値に「¥r¥n¥t」を指定しなければなりません。**TransliterationOut** プロパティの値は空でなければなりません。

---

3 ソース・ファイルをターゲット・ファイルにマップします。

4 ソース・ファイルをマップしたら，［**ファイル**］＞［**変換をエクスポート**］をクリックして変換を ***.djs** 変換ファイルにエクスポートします。

5 データベース・チェックポイント・ウィザードの［**変換ファイルを選択してください**］画面が表示されます。297 ページ「Data Junction 変換ファイルの選択」. の手順に従ってください。

6 WinRunner でデータベース・チェックポイントの作成を続けます。

- データベースを対象とする標準の検査を作成する場合，287 ページのステップ 4 から始まる手順に従ってください。
- データベースを対象とする標準の検査を作成する場合，289 ページのステップ 6 から始まる手順に従ってください。

Data Junction の詳しい使い方については，Data Junction のマニュアルを参照してください。

TSL 関数を使用してのデータベース作業

WinRunner は，データベースで作業するための複数の TSL 関数（**db_**）を提供します。

関数ジェネレータを使用して，データベース関数をテスト・スクリプトに挿入できます。または，これらの関数を使うステートメントを手作業でプログラムすることもできます。関数ジェネレータの作業の詳細については第 34 章「関数の生成」を，これらの関数の詳細については「**TSL リファレンス**」を参照してください。

データベースでのデータの検査

db_check 関数を使用して，ODBC（Microsoft Query）または Data Junction で標準のデータベース・チェックポイントを作成します。この関数の詳細については，284 ページ「データベースを対象とする標準の検査の作成」および 287 ページ「データベースを対象とするユーザ定義の検査の作成」を参照してください。**db_check** ステートメントのパラメータ化の詳細については，317 ページ「標準のデータベース・チェックポイントのパラメータ化」を参照してください。

データベースでのデータに対するアプリケーションの実行時のデータの検査

db_record_check 関数を使用して，ODBC（Microsoft Query）または Data Junction で標準のデータベース・レコード・チェックポイントを作成します。この関数の詳細については，270 ページ「データベース実行時レコード・チェックポイントの作成」を参照してください。

ODBC（Microsoft Query）で作業する際の TSL 関数

ODBC（Microsoft Query）で作業する場合，以下のステップを順番通りに実行しなければなりません。

1. データベースに接続します。
2. クエリを実行し，SQL ステートメントに基づいて結果セットを作成します（このステップはオプションです。このステップは，Microsoft Query を使用せずにクエリを作成 / 実行する場合にのみ実行します）。
3. データベースから情報を取得します。
4. データベースの接続を切断します。

これらのステップを実行するための TSL 関数を以下に説明します。

1 データベースへの接続

db_connect 関数は，新しいデータベース・セッションを作成し，ODBC データベースへの接続を確立します。この関数の構文は次のとおりです。

db_connect (*session_name*, *connection_string* **);**

session_name はデータベース・セッションの論理名です。*connection_string* は，ODBC データベースへの接続パラメータです。

2 SQL ステートメントに基づいたクエリの実行および結果セットの作成

db_execute_query 関数は，SQL ステートメントに基づいてクエリを実行し，結果セットを作成します。この関数の構文は次のとおりです。

db_execute_query (*session_name*, *SQL*, *record_number* **);**

session_name はデータベース・セッションの論理名です。*SQL* は SQL ステートメントです。*record_number* は，結果セットのレコード数を返す出力パラメータです。

3 データベース情報の取得

データベースの単一フィールドの値を返すには

db_get_field_value 関数は，データベース内の単一フィールドの値を返します。この関数の構文は次のとおりです。

db_get_field_value (*session_name*, *row_index*, *column* **);**

session_name はデータベース・セッションの論理名です。*row_index* は，行の数値インデックスです（先頭の行は必ず「0」と番号付けされます）。*column* は，データベース内のカラムにあるフィールド名またはカラムの数値インデックスです（先頭のカラムは必ず「0」と番号付けされます）。

カラム・ヘッダの内容と数を返すには

db_get_headers 関数は，クエリ内のカラム・ヘッダの数と内容を返します。戻り値は連結されてタブで区切られます。この関数の構文は次のとおりです。

db_get_headers (*session_name*, *header_count*, *header_content* **);**

session_name はデータベース・セッションの論理名です。*header_count* は，クエリ内のカラム・ヘッダの数です。*header_content* は，カラム・ヘッダであり，連結されてタブで区切られます。

行内容を返すには

db_get_row 関数は，行の内容を返します。戻り値は，連結されてタブで区切られます。この関数の構文は次のとおりです。

db_get_row (*session_name*, *row_index*, *row_content* **);**

session_name はデータベース・セッションの論理名です。*row_index* は，行の数値インデックスです（先頭の行は必ず「0」と番号付けされます）。*row_content* は，フィールドの値が連結されてタブで区切られた行内容です。

テキスト・ファイルへのレコード・セットの出力

db_write_records 関数は，レコード・セットをタブで区切り，テキスト・ファイルに出力します。この関数の構文は次のようになります。

db_write_records (*session_name*, *output_file* **[,** *headers* **[,** *record_limit* **]]);**

session_name はデータベース・セッションの論理名です。*output_file* は，レコード・セットを出力するテキスト・ファイルの名前です。*headers* は，オプションの Boolean パラメータです。テキスト・ファイルに出力するレコード・セットでカラム・ヘッダを含めたり除外します。*record_limit* は，テキスト・ファイルに出力するレコード・セットの最大のレコード数です。NO_LIMIT（標準値）という値は，レコード・セットのレコード数に上限がないことを示します。

最後の操作の最後のエラー・メッセージを返すには

db_get_last_error 関数は，ODBC または Data Junction における最後の操作の最後のエラー・メッセージを返します。この関数の構文は次のとおりです。

db_get_last_error (*session_name*, *error* **);**

session_name はデータベース・セッションの論理名です。*error* はエラー・メッセージです。

4 データベースへの接続の切断

db_disconnect 関数は，WinRunner とデータベースの接続を切断し，データベース・セッションを終了します。この関数の構文は次のとおりです。

db_disconnect (*session_name* **);**

session_name はデータベース・セッションの論理名です。

Data Junction で作業するための TSL 関数

Data Junction で作業する場合には，以下の 2 つの関数が使えます。

Data Junction のエクスポート・ファイルの実行

db_dj_convert 関数は，Data Junction のエクスポート・ファイル（.djs ファイル）を実行します。この関数の構文は次のとおりです。

db_dj_convert (*djs_file* **[,** *output_file* **[,** *headers* **[,** *record_limit* **]]]);**

djs_file は，Data Junction のエクスポート・ファイルです。*output_file* は，ターゲット・ファイルの名前をオーバーライドするためのオプション・パラメータです。*headers* は，Data Junction のエクスポート・ファイルにカラム・ヘッダを含めたり除外したりするオプションの Boolean パラメータです。*record_limit* は変換するレコードの最大数です。

最後の操作の最後のエラー・メッセージを返すには

db_get_last_error 関数は，ODBC または Data Junction における最後の操作の最後のエラー・メッセージを返します。この関数の構文は次のとおりです。

db_get_last_error (*session_name*, *error* **);**

session_name は，Data Junction で作業する場合は無視されます。*error* はエラー・メッセージです。

第 15 章

ビットマップの検査

WinRunner では，キャプチャしたビットマップを比較して，テスト対象アプリケーションの 2 つのバージョンを比較できます。これは，図形やグラフなどのアプリケーションの GUI 以外の部分を検査する場合に便利です。

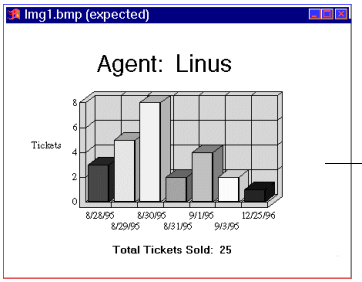
本章では，以下の項目について説明します。

- ビットマップの検査について
- ビットマップ・チェックポイントの作成
- ウィンドウとオブジェクトのビットマップの検査
- 領域ビットマップのチェック

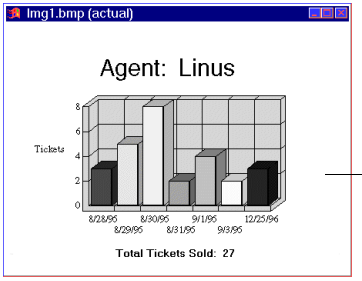
ビットマップの検査について

アプリケーション内のオブジェクト，ウィンドウまたは画面の領域を，ビットマップとして検査できます。テスト作成中に，検査対象を指定します。WinRunner は，指定されたビットマップをキャプチャし，それをテストの期待結果フォルダ（exp）に格納して，テスト・スクリプトにチェックポイントを挿入します。このテストを実行すると，WinRunner はテスト対象アプリケーションで現在表示されているビットマップと先に格納した「**期待結果**」ビットマップを比較します。不一致があった場合は，WinRunner は現在の「**実際の**」ビットマップをキャプチャし，「**差異**」ビットマップを生成します。これら 3 つのビットマップ（期待結果，実際，差異）を比較することで，何が問題なのかを知ることができます。

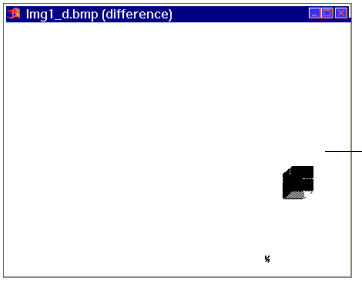
例えば，アプリケーションがデータベースの統計情報を示すグラフを表示したとします。アプリケーションの別のバージョンのグラフのビットマップと比較するために，グラフのビットマップをキャプチャできます。期待結果としてキャプチャしたグラフとテスト実行中にキャプチャしたグラフとの間に差異がある場合は，WinRunner は，ピクセルごとの差異を示すビットマップを生成します。



テスト作成時にキャプチャされた期待グラフでは、25 枚のチケットが売られています。



テスト実行時にキャプチャされた実際のグラフでは，27 枚のチケットが売られています。最後のカラムが高くなったのはチケット枚数が増えたためです。



差異ビットマップは 2 つのグラフの異なる部分，最後のカラムの高さと売れたチケット数の個所を表示します。

ビットマップ・チェックポイントの作成

コンテキスト・センシティブ・モードで作業している場合，ウィンドウ，オブジェクト，画面の特定領域のビットマップをキャプチャできます。WinRunner は，チェックポイントとして，**win_check_bitmap** または **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。

［**挿入**］＞［**ビットマップ チェックポイント**］を選択してビットマップの検査を開始します。ウィンドウやその他の GUI オブジェクトをキャプチャするには，対象オブジェクトをマウスでクリックします。領域ビットマップをキャプチャするには，検査する対象領域を十字カーソルを使って指定します。

アナログ・モードでテストを記録する場合，［ウィンドウ ビットマップのチェック］ソフト キーまたは［スクリーン領域のチェック］ソフトキーを押してビットマップ・チェックポイントを作成します。ソフトキーを使うことで，WinRunner が余分なマウスの動きを記録してしまうのを避けることができます。テストをプログラミング言語でプログラムする場合は，アナログ関数 **check_window** を使用してビットマップを検査できます。詳細は，「**TSL リファレンス**」を参照してください。

テストの実行ごとにウィンドウ名やオブジェクト名が変わる場合は，GUI マップ・エディタで正規表現を定義できます。これによりオブジェクト名の一部またはそのすべてを無視するように WinRunner を指定することができます。正規表現の使用については，第 7 章「GUI マップの編集」を参照してください。

ビットマップ・チェックポイントをループに含めることができます。ビットマップ・チェックポイントをループで実行すると，チェックポイントの各反復結果は異なるエントリに表示されます。チェックポイントの結果は［テスト結果］ウィンドウで見ることができます。詳細については，第 20 章「テスト結果の分析」を参照してください。

データ駆動テスト実行の際の注意：データ駆動テストでビットマップ・チェックポイントを使用するには，テスト・スクリプト内でそれを含むステートメントをパラメータ化してください。詳細は，361 ページ「テストのデータ駆動型テストへの変換」を参照してください。

画面表示ドライバの違いに対する対応

チェックポイントの作成時とテストの実行時に使用されている画面表示ドライバが異なると，同一ビットマップのビットマップ・チェックポイントが失敗することがあります。表示ドライバが異なると，ビットマップの表示に異なる色定義を使用することがあるからです。例えば，白はドライバによって RGB(255,255,255) としても，RGB (231,231,231) としても表示されます。

色の最大差異を指定して，WinRunner が差異を無視するように指定します。

無視できる色差異の程度を設定するには次の手順を実行します。

1 <**WinRunner インストール・フォルダ** >¥*dat* フォルダから **wrun.ini** を開きます。

2 XR_COLOR_DIFF_PRCNT= パラメータを [WrCfg] セクションに追加します。

3 無視する最大パーセンテージの値を入力します。

上記の例では，各 RGB コンポーネント (255:231) の差異は 9.4% ですから，XR_COLOR_DIFF_PRCNT パラメータの設定を 10 に設定すると WinRunner でビットマップを同じ物として扱うことができます。

```
[WrCfg]
XR_COLOR_DIFF_PRCNT=10
```

ビットマップ・チェックポイントとキャプチャ・オプションの設定

ビットマップ・チェックポイントが失敗するたびに，選択された受信者に電子メールを送信するよう，また，何らかのチェックポイントが失敗するたびに，ウィンドウまたは画面のビットマップをキャプチャするよう，WinRunner に指示できます。これらのオプションは，［一般オプション］ダイアログ・ボックスで設定します。

また，テスト実行の特定の時点でウィンドウまたは画面のビットマップをキャプチャするよう，WinRunner に指示するステートメントをスクリプトに挿入できます。

ビットマップ・チェックポイントの失敗時に WinRunner から電子メールが送信されるようにするには，次の手順を実行します。

1 ［**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・ボックスが開きます。

2 オプション表示枠で［**通知**］カテゴリをクリックします。通知オプションが表示されます。

3 ［**ビットマップ チェックポイントの失敗**］を選択します。

4 オプション表示枠で［**通知**］＞［**電子メール**］カテゴリをクリックします。電子メール・オプションが表示されます。

5 ［**電子メールのサービスを有効にする**］オプションを選択して，関連するサーバと送信者情報を設定します。

6 オプション表示枠で［**通知**］＞［**受信者**］カテゴリをクリックします。電子メールの受信者オプションが表示されます。

7 必要に応じて受信者の詳細を追加，削除，変更し，ビットマップ・チェックポイントの失敗時に電子メールを送信する受信者を設定します。

電子メールには，テストおよびビットマップ・チェックポイントの詳細なサマリが含まれ，期待画像，実際の画像，差異画像のファイル名が表示されます。

詳細については，560 ページ「通知オプションの設定」を参照してください。

チェックポイントの失敗時にビットマップをキャプチャするには，次の手順を実行します。

1 ［**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・ボックスが開きます。

2 オプション表示枠で［**実行**］＞［**設定**］カテゴリをクリックします。実行設定オプションが表示されます。

3 ［**検証失敗の時，ビットマップをキャプチャする**］を選択します。

4 ［**Window**］，［**Desktop**］または［**Desktop area**］を選択して，チェックポイントの失敗時にキャプチャするものを指定します。

5 ［**Desktop area**］を選択した場合は，キャプチャするデスクトップ領域の座標を指定します。

テストを実行すると，キャプチャされたビットマップはテスト結果フォルダに保存されます。

詳細については，545 ページ「テストの実行オプションの設定」を参照してください。

テストの実行中にビットマップをキャプチャするには，次の手順を実行します。

win_capture_bitmap ステートメントまたは **desktop_capture_bitmap** ステートメントを入力します。次の構文を使用します。

```
win_capture_bitmap(image_name [, window, x, y, width, height]);
```

または

```
desktop_capture_bitmap (image_name [, x, y, width, height]);
```

ステートメントには，希望の画像の名前だけを入力し，パスや拡張子などは含めないでください。ビットマップは **.bmp** 拡張子付きで，テスト結果フォルダのサブフォルダに自動的に保存されます。

詳細については，「**TSL リファレンス**」を参照してください。

ウィンドウとオブジェクトのビットマップの検査

アプリケーションの任意のウィンドウまたはオブジェクトを指して，そのビットマップをキャプチャできます。オブジェクトとウィンドウをキャプチャする方法は全く同じです。まず，［**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**］を選択します。アプリケーションのウィンドウ上でマウス・ポインタを移動すると，オブジェクトやウィンドウが点滅します。ウィンドウ・ビットマップをキャプチャするには，ウィンドウのタイトル・バーをクリックします。ウィンドウ内のオブジェクトをビットマップとしてキャプチャするには，オブジェクトそのものをクリックします。

記録中にアクティブでないウィンドウのオブジェクトをキャプチャすると，WinRunner は自動的に **set_window** ステートメントを生成します。

ウィンドウまたはオブジェクトをビットマップとしてキャプチャするには，次の手順を実行します。

1 ［**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。アナログ・モードで記録している場合は，［ウィンドウ ビットマップのチェック］ソフトキーを押します。

WinRunner ウィンドウが最小化され，マウス・ポインタが指差し型に変わり，ヘルプ・ウィンドウが開きます。

2 オブジェクトまたはウィンドウを指してクリックします。WinRunner はビットマップをキャプチャし，スクリプトに **win_check_bitmap** ステートメントか **obj_check_bitmap** ステートメントを生成します。

ウィンドウのビットマップについて生成される TSL ステートメントの構文は，次のとおりです。

win_check_bitmap（*object*, *bitmap*, *time*）;

オブジェクトのビットマップの場合の構文は次のとおりです。

obj_check_bitmap（*object*, *bitmap*, *time*）;

例えば，フライト予約アプリケーションのメイン・ウィンドウのタイトル・バーをクリックした場合，次のようなステートメントが生成されます。

win_check_bitmap (" フライト予約 ", "Img2", 1);

しかし，同じウィンドウの［フライト予定日］をクリックした場合，次のようなステートメントが生成されます。

obj_check_bitmap (" フライト予定日：", "Img1", 1);

win_check_bitmap および **obj_check_bitmap** 関数の詳細については，「**TSL リファレンス**」を参照してください。

注：win_check_bitmap および **obj_check_bitmap** 関数の動作は，*delay_msec*, *timeout_msec* ，および *min_diff* テスト実行オプションに設定されている値によって変わります。テスト実行オプションの詳細とその変更方法については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。また、［一般オプション］ダイアログ・ボックスをグローバルに使って，付随するテスト実行オプションの［**ウィンドウの同期化のための遅延**］，［**チェックポイントと CS ステートメントのタイムアウト**］，［**ビットマップ間の違いを差異として認識するしきい値**］を設定することもできます。詳細は，第 22 章「グローバル・テスト・オプションの設定」を参照してください。

領域ビットマップのチェック

画面に任意の矩形領域を指定して，比較するためにビットマップとしてキャプチャできます。領域のサイズは任意です。また，単独のウィンドウの特定の領域，あるいは複数のウィンドウと交わる領域も指定できます。領域を示す矩形は，矩形の左上角および右下角の座標によって定義されます。この座標は，その領域が含まれているウィンドウの左上角からの相対座標です。領域が，複数のウィンドウにまたがる場合や，タイトルを持たないウィンドウ内にある場合（例えば，ポップアップ・ウィンドウなど），座標は画面全体（ルート・ウィンドウ）の左上角からの相対座標です。

画面の領域をビットマップとしてキャプチャするには，次の手順を実行します。

1 ［**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，［**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。アナログ・モードで記録を行っている場合は，［スクリーン領域のチェック］ソフトキーを押します。

WinRunner ウィンドウが最小化され，マウス・ポインタが十字型に変わり，ヘルプ・ウィンドウが開きます。

2 キャプチャする領域を定義します。マウスの左ボタンを押して，矩形が対象領域全体を囲むまでマウスをドラッグしてからマウス・ボタンを放します。

3 マウスの右ボタンをクリックして操作を完了します。WinRunner は指定された領域をキャプチャし，**win_check_bitmap** ステートメントをスクリプトに生成します。

注：win_check_bitmap 関数の動作は，*delay_msec*，*timeout_msec*，および *min_diff* テスト・オプションの現在の設定によって変わります。テスト実行オプションの詳細とその変更方法については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。［一般オプション］ダイアログ・ボックスを使って，付随するテスト実行オプションの［**ウィンドウの同期化のための遅延**］，［**チェックポイントと CS ステートメントのタイムアウト**］，［**ビットマップ間の違いを差異として認識するしきい値**］をグローバルに設定することもできます。詳細は，第 22 章「グローバル・テスト・オプションの設定」を参照してください。

画面の領域について生成される **win_check_bitmap** ステートメントの構文は，次のとおりです。

win_check_bitmap (*window, bitmap, time, x, y, width, height*);

例えば，Windows のフライト予約アプリケーションで特定の領域を指定した場合，次のようなステートメントが生成されます。

```
win_check_bitmap (" フライト予約 ", "Img3", 1, 9, 159, 104, 88);
```

win_check_bitmap 関数の詳細 については，「**TSL リファレンス**」を参照してください。

第 16 章

テキストの検査

WinRunner では，アプリケーションの GUI オブジェクトに限らず，任意の領域のテキストを読み取って検査できます。

本章では，以下の項目について説明します。

- テキストの検査について
- テキストの読み取り
- テキストの検索
- テキストの比較
- WinRunner によるフォントの学習

テキストの検査について

テスト・スクリプトでテキスト・チェックポイントを使って，GUI オブジェクトや画面の領域からテキストを読み取って検査できます。これには，テスト作成の際に，テキストを含んでいるオブジェクトまたはウィンドウを指定します。WinRunner は，テキストを読み取り，テスト・スクリプトに TSL ステートメントを書き出します。そうしたテスト・スクリプトに，簡単なプログラミング要素を追加して，テキストの内容を検証できます。

テキスト・チェックポイントを使って以下のことができます。

- **obj_get_text**，**win_get_text** を使って，アプリケーションの GUI オブジェクトやウィンドウからテキストを読み取ることができます。
- **obj_check_text**，**win_check_text** を使って，アプリケーションの GUI オブジェクトやウィンドウからテキストを読み取り，期待テキストと比較することができます。

- **win_find_text**，**obj_find_text** を使って，オブジェクトやウィンドウでテキストを検索できます。
- **obj_move_locator_text** または **win_ move_ locator_ text** を使って，オブジェクトまたはウィンドウのテキストにマウス・ポイントを移動できます。
- **obj_click_on_text** または **win_ click_ on_ text** を使って，オブジェクトまたはウィンドウのテキストをクリックできます。
- **compare_text** を使って，2 つの文字列を比較できます。

GUI オブジェクトでテキスト・チェックポイントを使用するのは，**text プロパティ**を検査するのに GUI チェックポイントを使用できない場合にのみ使用すべきです。例えば，ユーザ定義のグラフ・オブジェクトのテキストを検査したいとします。このユーザ定義オブジェクトは，標準のオブジェクト・クラス（プッシュ・ボタン，リスト，メニューなど）にマップできないため，WinRunner はそれを汎用の object クラスにマップします。このようなオブジェクトに対する GUI チェックポイントでは，オブジェクトの幅，高さ，x 座標と y 座標，および，そのオブジェクトが使用可能になっているか，あるいはフォーカスがあるかどうかしか検査できません。オブジェクト内のテキストを検査できません。それには，テキスト・チェックポイントを作成しなければなりません。

次のスクリプトは，**win_check_text** 関数を使って，Kim Smith というテキストを含むフライト予約ウィンドウの［**名前**］編集ボックスを検査します。

```
set_window (" フライト予約 ", 3);
text_check=obj_check_text (" 名前 :","Kim Smith");
if (text_check==0)
    report_msg (" 名前は正しいです。");
```

WinRunner は，画面に表示されているテキストをほとんどのアプリケーションで読み取ることができます。テキスト認識メカニズムがドライバ・ベースの認識に設定されている場合，この処理は自動的に行われます。しかし，テキスト認識メカニズムが画像ベースの認識に設定されている場合は，WinRunner はアプリケーションが使っているフォントを学習しなければならないことがあります。WinRunner にフォントを学習させるには，フォント学習ユーティリティを使います。フォントの学習を行うべき状況およびその方法については，350 ページ「WinRunner によるフォントの学習」で説明します。テキスト認識メカニズムの設定方法の詳細については，541 ページ「テキスト認識オプションの設定」を参照してください。

注：Windows ベースのアプリケーションで WinRunner テキスト認識メカニズムを使用する場合は，不要なテキスト情報（隠しテキストや同じ文字列のコピーとして複数回表示される影付きテキストなど）が取り込まれてしまう場合もあります。

また，テキスト認識は，使用しているオペレーティング・システムのバージョンやインストールしているサービス・パック，インストールされているその他のツールキット，アプリケーションで使用されている API などによって実行セッションごとに異なる可能性があります。

したがって，可能な限り，アプリケーション・ウィンドウからテキストを取得または検査する場合には標準 GUI チェックポイントを挿入して，オブジェクトの **value**（または同様の）プロパティを検査することを強くお勧めします。

詳細については，543 ページ「Windows アプリケーションに対してテキスト認識を使用する際の注意事項」を参照してください。

テキストの読み取り

アプリケーションの GUI オブジェクトまたはウィンドウに含まれているすべてのテキスト，あるいは画面の特定領域のテキストを読み取ることができます。テキストを取得して変数にすることも，取得したテキストを指定した値と比較することもできます。

テキストを取得して変数にするには，**win_get_text**，**obj_get_text**，および **get_text** 関数を使用します。これらの関数は，［**挿入**］＞［**テキストの取得**］コマンドを使用して自動的に生成することも，手作業で生成することも，プログラミングして生成することも可能です。どの場合も，読み取られたテキストは出力変数に割り当てられます。

GUI オブジェクトに含まれているすべてのテキストを読み取るには，［**挿入**］＞［**テキストの取得**］＞［**オブジェクト / ウィンドウから**］を選択してから，マウス・ポインタでオブジェクトをクリックします。ウィンドウまたはオブジェクトの特定領域にあるテキストを読み取るには，［**挿入**］＞［**テキストの取得**］＞［**画面領域から**］コマンドを選択してから，十字カーソルを使って対象テキストを四角で囲みます。

ほとんどの場合，WinRunner は GUI オブジェクトのテキストを自動的に識別できます。しかし，テキストを読み取ろうとして，テスト・スクリプトに「# テキストが見つかりませんでした」というコメントが挿入された場合，それは WinRunner がテキストを自動的に認識できなかったことを示します。WinRunner にテキストを識別できるようにするには，画像ベースのテキスト認識を使用して，WinRunner にアプリケーションのフォントを学習させる必要があります。詳細については，350 ページ「WinRunner によるフォントの学習」を参照してください。

ウィンドウ内のテキストまたはオブジェクトを期待テキストの値と比較するには，**win_check_text** または **obj_check_text** 関数を使用します。

ウィンドウまたはオブジェクトのすべてのテキストの読み取り

ウィンドウに表示されているすべてのテキスト読み取るには，**win_get_text** を使い，オブジェクトのすべてのテキストを読み取るには，**obj_get_text** を使います。

ウィンドウまたはオブジェクトに表示されているすべてのテキスト読み取るには，次の手順を実行します。

1 ［**挿入**］＞［**テキストの取得**］＞［**オブジェクト / ウィンドウから**］を選択するか，**ユーザ定義ツールバーの**［**オブジェクト / ウィンドウからのテキスト取得**］ボタンをクリックします。アナログ・モードで記録を行っている場合は，［テキスト取得ーオブジェクト / ウィンドウから］ソフトキーを押します。

WinRunner ウィンドウが最小化され，マウス・ポインタが指差し型に変わり，画面に［テキスト取得］ダイアログ・ボックスが開きます。

2 ウィンドウまたはオブジェクトをクリックします。WinRunner は，オブジェクトのテキストをキャプチャし，**win_get_text** または **obj_get_text** ステートメントを生成します。

対象がウィンドウの場合，ステートメントの構文は次のとおりです。

win_get_text (*window*, *text* **);**

window は，ウィンドウの名前です。*text* は，ウィンドウに表示されているすべてのテキストを格納する出力変数です。このテキストは，テスト・スクリプトが読みやすくなるよう，関数の記録時にコメントとしてスクリプトに挿入されます。

例えば，［**挿入**］＞［**テキストの取得**］＞［**オブジェクト / ウィンドウから**］を選択して，Windows の時計アプリケーションをクリックすると，以下のようなステートメントがテスト・スクリプトに記録されます。

```
# 時計の設定 10:40:46 AM 8/8/95
win_get_text(" 時計 ", text);
```

対象が GUI オブジェクトの場合の構文は，次のとおりです。

obj_get_text (*object*, *text* **);**

obj_get_text 関数のパラメータは，**win_get_text** と全く同じです。

注：WebTest アドインをロードして，Web オブジェクトを選択すると WinRunner はテスト・スクリプトに **web_frame_get_text** ステートメントか **web_obj_get_text** ステートメントを生成します。詳細については第 10 章「Web オブジェクトでの作業」または「**TSL リファレンス**」を参照してください。

画面の領域に含まれているテキストの読み取り

win_get_text 関数と **obj_get_text** 関数を使って，ウィンドウまたは他の GUI オブジェクトの特定の領域のテキストを読み取ることができます。

ウィンドウまたはオブジェクトの特定領域のテキストを読み取るには，次の手順を実行します。

1 ［**挿入**］＞［**テキストの取得**］＞［**画面領域から**］を選択するか**ユーザ定義ツールバーの**［**画面領域からテキスト取得**］ボタンをクリックします。アナログ・モードで記録を行っている場合は，［テキスト取得－スクリーン領域から］ソフトキーを押します。

WinRunner ウィンドウが最小化されて，マウスとキーボード入力の記録が止まり，マウス・ポインタが十字型に変わります。

2 十字カーソルを使って，読み取り対象となるテキストを四角で囲みます。それには，まずマウス・カーソルを対象テキストの 1 つの角に移動します。そこで，マウスの左ボタンを押します。押し下げたままマウスをドラッグして，対象テキストをすっかり囲んだら，マウス・ボタンを放します。マウスの右ボタンをクリックすると文字列がキャプチャされます。

キャプチャを行う前に，文字列をプレビューできます。これを行うには，マウスの左ボタンを放さずにマウスの右ボタンを押下します（マウスにボタンが 3 つある場合は，テキストを囲む四角を指定してからマウスの左ボタンを放して，マウスの中央ボタンをクリックします）。文字列が，四角の真下または画面の左上角に表示されます。

WinRunner は，テスト・スクリプトに，次の構文の **win_get_text** ステートメントを生成します。

win_get_text (*window, text, x1,y1,x2,y2* **);**

例えば，［テキストの取得］＞［画面領域から］を選択して，十字カーソルを使って Windows の時計アプリケーションの日付の部分だけを囲む四角を指定した場合，テスト・スクリプトには，以下のようなステートメントが記録されます。

```
win_get_text (" 時計 ", text, 38, 137, 166, 185); # 8/13/95
```

window は，ウィンドウの名前です。*text* は，ウィンドウに表示されているテキストを格納する出力変数です。***x1，y1，x2，y2*** は，指定したウィンドウを基準にしてテキストを読み取る位置を定義する座標です。この関数が記録されると，キャプチャされたテキストも，スクリプトにコメントとして挿入されます。

テスト・スクリプトに挿入されるコメントは，画面で読み取られるテキストの行数と同じ行数を占めます。例えば，キャプチャされたテキストが 3 行であった場合，コメントも 3 行となります。

画面のテキストは，アナログの TSL 関数 **get_text** をテスト・スクリプトにプログラミングして，読み取ることもできます。詳細については，「**TSL リファレンス**」を参照してください。

注：学習フォントを使ったテキストを読み取ると，WinRunner はテキストを 1 行だけ読み取ります。テキストが複数行にわたる場合は，最も左にある行だけが読み取られます。左のマージンが同じ行が複数ある場合は，一番最後の行が読み取られます。詳細については，350 ページ「WinRunner によるフォントの学習」を参照してください。

ウィンドウまたはオブジェクト内のテキストの検査

WinRunner がオブジェクトまたはウィンドウから取得するテキストの値を期待テキストの値と比較するには，**win_check_text,** or **obj_check_text** 関数を使用できます。

get_text 関数と同様，**check_text** 関数は，ウィンドウまたはオブジェクト内のすべてのテキスト，または指定した座標のテキストのみを検査できます。

期待テキストと実際のテキストが一致する場合は，**check_text** 関数は E_OK (0) 戻りコードを返します。

ウィンドウ内のテキストを検査する場合は，次の構文を使用します。

win_check_text (*window*, *expected_text* **[**, *x1*, *y1*, *x2*, *y2* **]);**

オブジェクト内のテキストを検査する場合は，次の構文を使用します。

obj_check_text (*object*, *expected_text* **[**, *x1*, *y1*, *x2*, *y2* **]);**

詳細については，「**TSL リファレンス**」を参照してください。

テキストの検索

以下の TSL 関数を使って，画面に表示されているテキストを検索できます。

- **win_find_text**，**obj_find_text**，**find_text** 関数は，指定されたテキスト文字列の位置を決定します。
- **obj_move_locator_text**，**win_move_locator_text**，**move_locator_text** 関数は，マウス・ポインタを指定されたテキスト文字列まで移動します。
- **win_click_on_text**，**obj_click_on_text**，**click_on_text** 関数は，ポインタをテキスト文字列まで移動し，マウス・クリックを実行します。

これらの関数は，テスト・スクリプトにプログラミングする必要があります。プログラミングするには，関数ジェネレータを使用するか，テスト・スクリプトにステートメントを入力します。テスト・スクリプトに関数をプログラミングする方法の詳細については，第 34 章「関数の生成」を参照してください。指定する関数については「**TSL リファレンス**」を参照してください。

テキスト文字列の位置の取得

win_find_text，**obj_find_text** 関数は，**win_get_text** および **obj_get_text** 関数と逆の処理を行います。**get_text** 関数が指定されたオブジェクトまたはウィンドウで見つかったテキストを取得するのに対し，**find_text** 関数は指定された文字列を探し，ウィンドウまたはオブジェクトを基準とする相対的な位置を返します。

win_find_text と **obj_find_text** はコンテキスト・センシティブ関数であり，次に示すように構文が似ています。

win_find_text (*window, string, result_array* **[** ,x_1,y_1,x_2,y_2 **] [** ,*string_def* **]);**

obj_find_text (*object, string, result_array* **[** *,x1,y1,x2,y2* **] [** *,string_def* **]);**

window または *object* には，WinRunner が指定されたテキストを検索する対象となるウィンドウまたはオブジェクトの名前を指定します。*string* には，検索するテキストを指定します。*result_array* には，文字列の位置を格納する 4 つの要素を持つ配列の名前を指定します。オプションの x_1，y_1，x_2，y_2 には，検索対象となる画面領域の左上角と右下角の x 座標と y 座標を指定します。これらのパラメータが定義されていない場合は，WinRunner はウィンドウ全体，またはオブジェクト全体を検索対象領域とします。オプションの *string_def* には，WinRunner の文字列検索の方法を指定します。

win_find_text と **obj_find_text** 関数は，検索が失敗すると 1 を返し，成功すると 0 を返します。

以下の例では，**win_find_text** を使って，フライト予約アプリケーションのグラフ・オブジェクトのどこに合計が表示されるかを調べます。

```
set_window (" グラフ ", 10);
win_find_text (" グラフ ", " 販売済みチケット合計枚数：", result_array,
640,480,366,284, FALSE);
```

画面のテキストは，アナログ TSL 関数，**find_text** を使って検索することも可能です。

find_text の詳細については，「**TSL リファレンス**」を参照してください。

注：学習したフォントを使うテキストを対象に **win_find_text**，**obj_find_text**，または **find_text** 関数を使った場合，WinRunner は 1 つの完全な単語しか検索しません。つまり，*string* パラメータで使用する正規表現には空白を含めてはならず，*string_def* の標準の値である FALSE だけが有効です。

テキスト文字列へのポインタの移動

win_move_locator_text と **obj_move_locator_text** 関数は，指定されたウィンドウまたはオブジェクトの指定されたテキスト文字列を検索します。テキストが見つかると，マウス・ポインタはテキストの中央に移動されます。

win_move_locator_text と **obj_move_locator_text** 関数は，コンテキスト・センシティブ関数であり，次に示すように構文が似ています。

win_move_locator_text (*window, string,* **[** *,x1,y1,x2,y2* **] [** *,string_def* **]);**

obj_move_locator_text (*object, string,* **[** *,x1,y1,x2,y2* **] [** *,string_def* **]);**

window または *object* には，WinRunner の検索対象のウィンドウまたはオブジェクトの名前を指定します。*string* には，マウス・ポインタの移動先となるテキストを指定します。オプションの ***x1***，***y1***，***x2***，***y2*** パラメータには，検索対象となるウィンドウまたはオブジェクトの領域の左上角と右下角の x 座標と y 座標を指定します。オプションの *string_def* には，WinRunner の文字列検索の方法を指定します。

以下の例では，**obj_move_locator_text** を使って，Windows オンライン・ヘルプの索引のトピック文字列にマウス・ポインタを移動します。

```
function verify_cursor(win,str)
{
    auto text,text1,rc;

    # トピック文字列を検索し，ロケータをテキストに移動する。文書の
    # 最後にスクロールし，見つからなければ，最初から繰り返す。
    set_window (win, 1);
    obj_mouse_click ("MS_WINTOPIC", 1, 1, LEFT);
     type ("<kCtrl_L-kHome_E>");
     while(rc=obj_move_locator_text("MS_WINTOPIC",str,TRUE)){
        type ("<kPgDn_E>");
        obj_get_text("MS_WINTOPIC", text);
        if(text==text1)
            return E_NOT_FOUND;
    text1=text;
    }
 }
```

TSL アナログ関数，**move_locator_test** を使ってもマウス・ポインタをテキスト文字列に移動できます。**move_locator_test** の詳細については，「**TSL リファレンス**」を参照してください。

指定されたテキスト文字列をクリックする方法

win_click_on_text と **obj_click_on_text** 関数は，指定されたウィンドウまたは GUI オブジェクト内の指定されたテキスト文字列を検索し，画面ポインタをその文字列の中央に移動し，文字列をクリックします。

win_click_on_text と **obj_click_on_text** 関数はコンテキスト・センシティブ関数であり，次に示すように構文が似ています。

win_click_on_text (*window, string,* **[** ,x_1,y_1,x_2,y_2 **] [** ,*string_def* **]**
[,*mouse_button* **]);**

window または *object* には，WinRunner の検索対象のウィンドウまたはオブジェクトの名前を指定します。*string* には，マウス・ポインタをクリックするテキストを指定します。オプションの *x1*，*y1*，*x2*，*y2* パラメータには，検索対象のウィンドウまたはオブジェクトの領域を指定します。オプションの *string_def* には，WinRunner の文字列検索の方法を指定します。オプションの *mouse_button* は使用するマウス・ボタンを指定します。

次の例では，**obj_click_on_text** を使って，トピックの検索にジャンプするためにオンライン・ヘルプの索引をクリックします。

```
function show_topic(win,str)

{
    auto text,text1,rc,arr[];

    # オブジェクト内でトピック文字列を検索する。見つからない場合は,
    # 文書の最後までスクロールする。
    set_window (win, 1);
    obj_mouse_click ("MS_WINTOPIC", 1, 1, LEFT);
     type ("<kCtrl_L-kHome_E>");
     while(rc=obj_click_on_text("MS_WINTOPIC",str,TRUE,LEFT)){
            type ("<kPgDn_E>");
            obj_get_text("MS_WINTOPIC", text);
            if(text==text1)
                return E_GENERAL_ERROR;
            text1=text;
            }
    }
```

click_on_text 関数については，「**TSL リファレンス**」を参照してください。

テキストの比較

compare_text 関数を使って，2 つの文字列を比較できます。その際に，無視すべき差異も指定できます。この関数は，単独で使用したり，**win_get_text** や **obj_get_text** 関数と組み合わせて使ったりできます。

compare_text 関数の構文は，次のとおりです。

variable = **compare_text (** *str1*, *str2* **[** ,*chars1*, *chars2* **]);**

str1 および *str2* パラメータには，比較するリテラル文字列または文字列変数を指定します。

オプションの *chars1* と *chars2* パラメータには，比較の際に無視すべきリテラル文字，またはそれを格納した文字列変数を指定します。*chars1* および *chars2* には複数の文字を指定することも可能です。

compare_text 関数は，比較対象文字列が同じであると判断した場合には 1 を返し，異なると判断した場合には 0 を返します。例えば，テスト・スクリプトの中で，**get_text** で返された「File」というテキスト文字列を比較するとします。小文字の「l」（エル）と大文字の「I」（アイ）は非常によく似ているので，以下のように，この 2 つの文字を無視するように指定できます。

```
t = get_text (10, 10, 90, 30);
if (compare_text (t, "File", "l", "I"))
        move_locator_abs (10, 10);
```

WinRunner によるフォントの学習

フォント・エキスパート・ユーティリティは，WinRunner がアプリケーションのテキストを自動的に読み取ることができなかった場合にのみ使います。このような場合，アプリケーションで使っているフォントを WinRunner に学習させる必要があります。

多くの場合，WinRunner は GUI オブジェクトを自動的に識別できます。しかし，テキストを読み取ろうとしたときに “#no text was found” というコメントがテスト・スクリプトに挿入されている場合は，WinRunner がアプリケーションのフォントを識別できなかったことを意味します。

WinRunner がテキストを識別できるようにするには，フォント・エキスパート・ユーティリティを使用して WinRunner にアプリケーション・フォントを教え，テストの実行時には画像テキスト認識メカニズムを使用します。

WinRunner にフォントを学習させるためには，主に次の手順を実行します。

1 アプリケーションで使っている文字セット（フォント）をフォント・エキスパートを使用して，WinRunner に学習させます。

2 1 つ，または複数のフォントを含むフォント・グループを作成します。

「**フォント・グループ**」とは，特定のテストのためにグループ化された一連のフォントの集まりです。WinRunner では，一度に 1 つのフォント・グループしかアクティブにすることができません。学習したフォントが認識されるには，そのフォントがアクティブなフォント・グループになければなりません。ただし，学習したフォントは，複数のフォント・グループに割り当てることができます。

3 ［一般オプション］の［**記録開始**］＞［**テキスト認識**］カテゴリで，［**画像ベースのテキスト認識を使用する**］オプションを選択し，［**フォント グループ**］ボックスに作成したフォント・グループを入力します。

4 テキスト関数を使う前に，TSL の setvar 関数を使って，適切なフォント・グループをアクティブにしておきましょう。

学習されたフォントとフォント・グループはすべて「**フォント・ライブラリ**」に格納されます。このライブラリは，*wrun.ini* ファイルの XR_GLOB_FONT_LIB パラメータによって指定されます。標準設定では，ライブラリは WinRunner がインストールされているディレクトリの下の *fonts* サブフォルダにあります。

フォントの学習

WinRunner がアプリケーションのテキストを読み取ることができなかった場合は，フォント・エキスパートを使ってフォントを学習します。

フォントを学習するには，次の手順を実行します。

1 ［**ツール**］＞［**フォント エキスパート**］を選択するか，［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**Fonts Expert**］を選択します。［**フォント エキスパート**］ウィンドウが表示されます。

2 ［**フォント**］＞［**学習**］を選択します。［フォントの学習］ウィンドウが表示されます。

3 ［**フォント名**］ボックスに新しいフォントの名前を入力します（最大 8 文字で，拡張子は指定しません）。

4 ［**フォントを選択**］ボタンをクリックします。［フォント］ダイアログ・ボックスが表示されます。

5 フォント名，スタイル，およびサイズを該当するリストから選択します。

ヒント：担当プログラマにフォント名，スタイル，サイズを問い合わせることもできます。

---

6 ［**OK**］をクリックします。

7 ［**フォントを学習**］ボタンをクリックします。

この処理が完了すると，［既存の文字］ボックスには，学習したフォントが表示され，［プロパティ］ボックスには，学習したフォントのプロパティが表示されます。WinRunner は，学習したフォントのデータを収めた *font_name.mfn* という形式の名前のファイルを作成し，それをフォント・ライブラリに格納します。

8 ［**閉じる**］をクリックします。

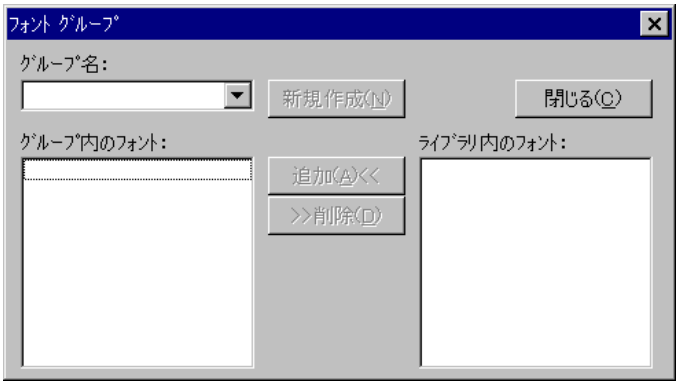
フォント・グループの作成

フォントを学習したら，そのフォントをフォント・グループに割り当てなければなりません。1 つフォントを複数のフォント・グループに割り当てることも可能です。

注：フォント・グループに属するフォントの数が増えれば，それだけテキストの認識率が低下する傾向があるため，各グループには 2 つ以上のフォントは含めないようにしましょう。

新しいフォント・グループを作成するには，次の手順を実行します。

1 ［フォント エキスパート］ウィンドウで，［**フォント**］＞［**グループ**］を選択します。［フォント グループ］ダイアログ・ボックスが開きます。



2 ［**グループ名**］ボックスに一意の名前を入力します（長さ 8 文字まで，拡張子は指定しません）。

3 ［**ライブラリ内のフォント**］リストから，フォント・グループに入れるフォントの名前を選択します。

4 ［**新規**］をクリックします。すると，WinRunner は新しいフォント・グループを作成します。処理が完了すると，登録したフォントが［グループ内のフォント］リスト・ボックスに表示されます。

WinRunner は，作成したフォント・グループのデータを含む *group_name.grp* という形式の名前のファイルを作成し，それをフォント・ライブラリに格納します。

既存のフォント・グループにフォントを追加するには，次の手順を実行します。

1 ［フォント エキスパート］ウィンドウで，［**フォント**］＞［**グループ**］を選択します。［フォント グループ］ダイアログ・ボックスが表示されます。

2 ［**グループ名**］リストから，フォントの追加対象となるフォント・グループを選択します。

3 ［**ライブラリ内のフォント**］リストで，追加するフォントの名前をクリックします。

4 ［**追加**］をクリックします。

フォント・グループからフォントを削除するには，次の手順を実行します。

1 ［フォント エキスパート］ウィンドウで，［**フォント**］＞［**グループ**］を選択します。［フォント グループ］ダイアログ・ボックスが表示されます。

2 ［**グループ名**］リストから削除したいフォント・グループを選択します。

3 ［**グループ内のフォント**］リストで，削除するフォントの名前をクリックします。

4 ［**削除**］をクリックします。

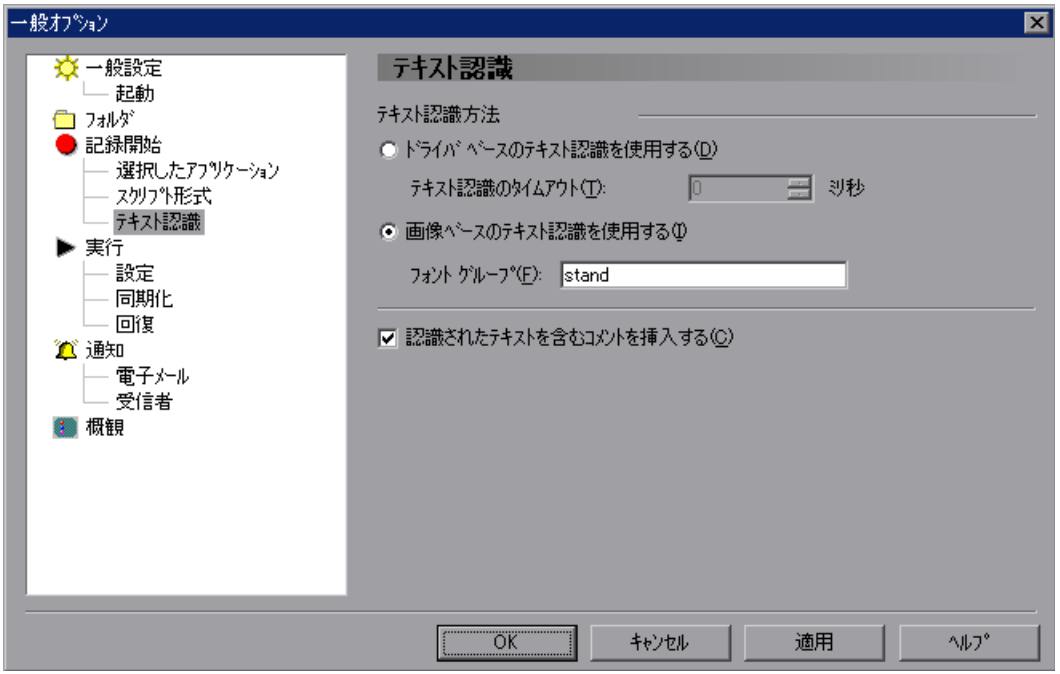
学習済みのフォントを対象としたテストの実行

WinRunner に，フォント・グループ内のフォントを使用させるには，WinRunner の標準テキスト認識メカニズムではなく画像テキスト認識メカニズムを使用し，アプリケーションで使用されているフォントを含むフォント・グループをアクティブにする必要があります。

WinRunner が学習フォントを認識できるようにするには，次の手順を実行します。

1 ［**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・ボックスが開きます。

2 ［**記録開始**］＞［**テキスト認識**］カテゴリをクリックします。



3 ［**画像ベースのテキスト認識を使用する**］を選択します。

4 ［**フォント グループ**］ボックスにフォント・グループを入力します。

5 ［**OK**］をクリックして選択を保存し，ダイアログ・ボックスを閉じます。

一度にアクティブにできるフォント・グループは 1 つだけです。標準設定では，これは *wrun.ini* ファイルの XR_FONT_GROUP システム・パラメータによって指定されます。しかし，テスト・スクリプト内で異なるフォント・グループをアクティブにすることも，*fontgrp* テスト・オプションを指定して **setvar** 関数をアクティブにすることもできます。

例えば，テスト・スクリプト内で editor という名前のフォント・グループをアクティブにするには，以下のコマンドをスクリプトに追加します。

```
setvar ("fontgrp", "editor");
```

［一般オプション］ダイアログ・ボックスでテキスト認識の設定を行う方法の詳細については，第 22 章「グローバル・テスト・オプションの設定」を参照してください。テスト・スクリプト内で **setvar** を使ったフォント・グループの選択の詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

第 17 章

データ駆動型テストの作成

WinRunner では，外部テーブルに格納されたデータによって駆動するテストを作成し実行できます。

本章では，以下の項目について説明します。

- データ駆動型テストの作成について
- データ駆動型テストの工程
- 変換用基本テストの作成
- テストのデータ駆動型テストへの変換
- データ・テーブルの準備
- データベースからのデータのインポート
- データ駆動型テストの実行と分析
- テストへのメイン・データ・テーブルの割り当て
- データ駆動型チェックポイントとビットマップ同期化ポイントの使用
- データ駆動型テストでの TSL 関数の使用
- データ駆動型テスト作成のガイドライン

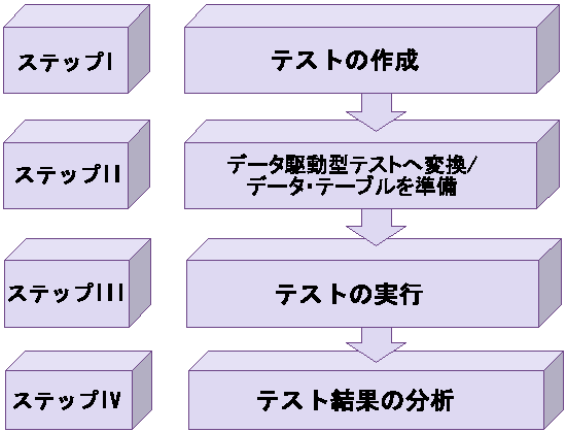
データ駆動型テストの作成について

アプリケーションをテストするときには，複数のセットのデータに対して同じ操作を実行したらどうなるか検査したいことがあります。例えば，10 種類の独立したデータのセットに対してアプリケーションがどのように応答するかを検査したいとしましょう。10 の独立したテストを作成し，それぞれに専用のデータ・セットを持たせることができます。しかし，その代わりに 10 回実行するループを持つ「**データ駆動型**」テストを作成することもできます。10 回の各「**反復**」で，テストは異なるデータ・セットによって駆動されます。WinRunner でテストを駆動するデータを使用するには，テスト内の固定値を変数で置換しなければなりません。テスト内の変数は，「**データ・テーブル**」に格納されているデータにリンクしています。データ駆動型テストは，データ駆動テスト・ウィザードを使用したり，データ駆動型ステートメントをテスト・スクリプトに手動で追加することにより作成できます。

データ駆動型テストの工程

非データ駆動型テストのテスト工程は，テストの作成，テストの実行，結果の分析という 3 つの段階で構成されます。データ駆動テストを作成する場合は，テストの作成と実行の間にもう 1 つ，2 つの作業から成る段階が追加されます。この段階では，テストをデータ駆動型テストに変換し，対応するデータ・テーブルを作成します。

次の図に，WinRunner でのデータ駆動型テストのテスト工程の概略を示します。

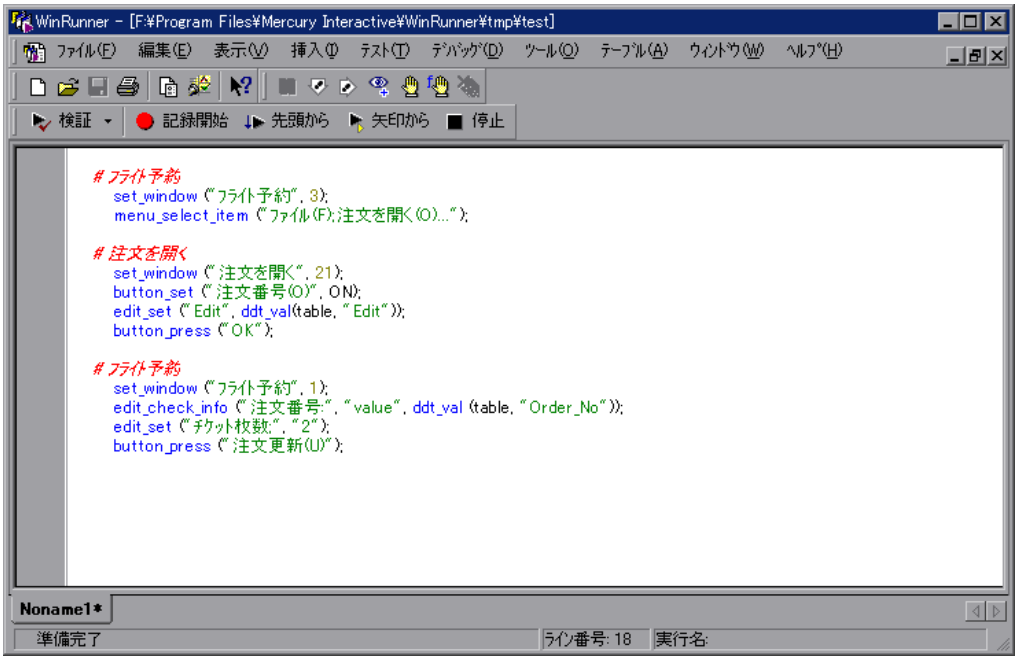


変換用基本テストの作成

データ駆動型テストを作成するには，まず基本となるテストを作成してから，これを変換します。

データ駆動型のテストは，普通と同様に，1 つのデータセットを持つテストを記録することによって作成します。次の例では，さまざまな注文に対して，正しく注文を開き更新するかを検査します。このテストでは，ある乗客のフライト・データを使って記録されます

このテストを記録するには，注文を開いて，［**挿入**］＞［**GUI チェックポイント**］＞［**単数プロパティ**］コマンドを使って正しい注文が開かれることを検査します。注文のチケット枚数を変更して，注文を更新します。次のようなテスト・スクリプトが作成されます。



このテストの目的は，正しい注文が開かれたかどうかを検査することです。通常は［**挿入**］＞［**GUI チェックポイント**］＞［**オブジェクト / ウィンドウ**］コマンドを使って，**obj_check_gui** ステートメントをテスト・スクリプトに挿入します。しかし，すべての **_check_gui** ステートメントにはチェックリストへの参照が含まれていて，チェックリストには固定値が含まれていないため，データ駆動型テストを作成中にチェックリストをパラメータ化することはできません。

次の 2 つの選択肢があります。

➤ 上の例のような場合には，［**挿入**］＞［**GUI チェックポイント**］＞［**単数プロパティ**］コマンドを使ってチェックリストのないプロパティ・チェックを作成します。この場合 ,**edit_check_info** ステートメントは注文番号を表示する編集フィールドの内容を検査します。オブジェクトの単数のプロパティ検査の詳細については，第 9 章「GUI オブジェクトの検査」を参照してください。

WinRunner は，テスト実行中にステートメントが失敗した時はいつでも［テスト結果］ウィンドウにイベントを書くことができます。このオプションを設定するには，［一般オプション］ダイアログ・ボックスの［**実行**］＞［**設定**］カテゴリで［**単数のプロパティが失敗したらテストを失敗とする**］チェック・ボックスを選択するか，**setvar** 関数を使用して ***single_prop_check_fail*** テスト・オプションを設定します。詳細については，第 22 章「グローバル・テスト・オプションの設定」または第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

［**挿入**］＞［**GUI チェックポイント**］＞［**単数プロパティ**］コマンドを使って，次の ***_check_*** 関数を使うプロパティ検査を作成できます。

button_check_info	**scroll_check_info**
edit_check_info	**static_check_info**
list_check_info	**win_check_info**
obj_check_info	

次の **_check** 関数を使って，チェックリストを作成せずにオブジェクトの単数のプロパティを検査することもできます。ステートメントは，これらの関数を使って手作業で作成することも，また関数ジェネレータを使って作成することもできます。詳細については，第 34 章「関数の生成」を参照してください。

button_check_state
edit_check_selection
edit_check_text
list_check_item
list_check_selected
scroll_check_pos
static_check_text

特定の関数の詳細については「**TSL リファレンス**」を参照してください。

➤ または，データ駆動型 GUI チェックポイントとビットマップ・チェックポイント，およびビットマップ同期化ポイントを作成することも可能です。データ駆動型 GUI チェックポイント，ビットマップ・チェックポイント，ビットマップ同期化ポイントの作成については，391 ページ「データ駆動型チェックポイントとビットマップ同期化ポイントの使用」を参照してください。

テストのデータ駆動型テストへの変換

テストをデータ駆動型テストに変換するための主な手順は次のとおりです。

1. チェックポイント・ステートメントと記録されたステートメントの固定値をパラメータで置換し，パラメータの値を含むデータ・テーブルを作成します。これをテストの「**パラメータ化**」と言います。
2. テストにステートメントと関数を追加して，テストがデータ・テーブルから値を読み込み，各データの反復を読み取る間これをループで実行するようにします。
3. データ・テーブルを開いて閉じるステートメントをテスト・スクリプトに追加します。
4. データ・テーブルに変数名を割り当てます（これはデータ駆動テスト・ウィザードを使うときは必須です。それ以外の場合は省略可能です）。

これらの手順は，データ駆動テスト・ウィザードを使っても，手作業でテスト・スクリプトを修正しても行えます。

データ駆動テスト・ウィザードを使ったデータ駆動型テストの作成

データ駆動テスト・ウィザードを使って，スクリプトの全体または一部をデータ駆動型テストに変換できます。例えば，テスト・スクリプトには，複数セットのデータに対して繰り返す必要のない，記録済みの操作，チェックポイント，およびその他のステートメントを含めることができます。これで，複数セットのデータを含む 1 つのループ内で実行したいテスト・スクリプトの一部だけをパラメータ化するだけで済みます。

データ駆動型テストを作るには，次の手順を実行します。

1 テスト・スクリプトの一部だけをデータ駆動型テストに変えるには，まずテスト・スクリプト内で対象となる行を選択します。

2 ［**テーブル**］＞［**データ駆動テストウィザード**］を選択します。

➤ ウィザードを開く前にテスト・スクリプトの一部を選択してあれば，363 ページの手順 3 に進んでください。

➤ スクリプトの任意の行を選択していなければ，次の画面が開きます。

テストの一部だけをデータ駆動型のテストに変えるには，［**キャンセル**］をクリックします。テスト・スクリプトで変更する行を選択して［データ駆動テストウィザード］を再び開きます。

テスト・スクリプト全体をデータ駆動テストに変えるには，［**次へ**］をクリックします。

3 次のウィザード画面が開きます。

［**新規 / 既存の Excel テーブルを使用する**］ボックスには，WinRunner が作成するエクセル・ファイルの名前が表示されます。このファイルには，データ駆動型テスト用のデータが格納されます。このテスト用に標準のデータ・テーブルを使用し，データ・テーブルに別の名前を入力するか，参照ボタンを使って既存のデータ・テーブルのパスを見つけます。標準設定では，データ・テーブルはテスト・フォルダ内に格納されています。

［**変数に名前を割り当てる**］ボックスには，データ・テーブルを参照するための変数名を入力するか，標準の名前「table」を使います。

データ駆動型テストでは，まず選択した Excel データ・テーブルをテーブル変数の値として割り当てます。スクリプトを通して，テーブル変数名だけが使用されます。こうしておくと，後からスクリプトに別のデータ・テーブルを割り当てる場合に，スクリプト全体を変更する必要がないので，作業が簡単になります。

以下のオプションを選択してください。

➤ ［**ステートメントを追加し，データ駆動テストを作成する**］：**自動的にステートメントを追加してテストをループで実行します。**データ・テーブルを参照する変数名を設定します。括弧（{ と }），**for** ステートメントおよび **ddt_get_row_count** ステートメントを選択したテスト・スクリプトに追加して，データ・テーブルから読み取りながらループ内でこれを実行します。**ddt_open** ステートメントと **ddt_close** ステートメントをテスト・スクリプト

に追加して，データ・テーブルを開いて閉じます。これは，テーブル内で行を繰り返すのに必要です。

これらのステートメントは，テスト・スクリプトに手作業で追加することもできます。詳細とサンプルのステートメントは，371 ページ「テスト・スクリプトへのデータ・テーブルを開閉してテストをループで実行するステートメントの追加」を参照してください。

このオプションを選択しないと，データ駆動型テストにループとデータ・テーブルを開閉するステートメントを含まなければならないことを警告するメッセージが表示されます。

注：以前テスト・スクリプトの同じ部分を対象にデータ駆動テスト・ウィザードを実行した際にこのオプションを選択した場合は，このオプションを選択してはいけません。

➤ ［**データベースからデータをインポートする**］：データベースからデータをインポートします。このオプションは，**ddt_open** の後に **ddt_update_from_db** ステートメントと **ddt_save** ステートメントを追加します。詳細については，372 ページ「データベースからのデータのインポート」を参照してください。

データベースからデータをインポートするには，Microsoft Query か Data Junction のいずれかをお使いのマシンにインストールしなければなりません。Microsoft Query は，Microsoft Office の「**カスタム・インストール**」でインストールできます。Data Junction は，WinRunner のパッケージに含まれているものではありません。Data Junction を購入する場合は , お近くの Mercury Interactive 社の製品取扱代理店まで連絡してください。Data Junction の使用法については，Data Junction パッケージのマニュアルを参照してください。

注：［**ステートメントを追加し，データ駆動テストを作成する**］オプションが［**データベースからデータをインポートする**］**オプション**と一緒に選択されていなければ，データ・テーブルを参照する変数名もデータ駆動テスト・ウィザードで設定されます。加えて，**ddt_open** と **ddt_close** ステートメントもテスト・スクリプトに追加されます。テストには繰り返しがないので，**ddt_close** ステートメントは，選択されたテキスト・ブロックの最後ではなく，**ddt_** ステートメントのブロックの最後になります。

- ［**新規 / 既存テストをパラメータ化する**］：選択されたチェックポイントと記録済みのステートメントの固定値を，ddt_val 関数を使ってパラメータに置き換え，データ・テーブルにパラメータの変数値を含むカラムを追加します。

  ［**行ごと**］：選択したテスト・スクリプトの各行に対してウィザード画面を開きます。ここで特定の行をパラメータ化するかどうかを決め，パラメータ化する場合は，データをパラメータ化するときに新しいカラムをデータ・テーブルに追加するか，既存のカラムを使用するかを選択できます。

  ［**自動**］：すべてのデータを **ddt_val** ステートメントに置き換えて，新しいカラムをデータ・テーブルに追加します。関数の最初の引数は，データ・テーブルのカラムの名前です。置き換えられたデータはテーブルに挿入されます。

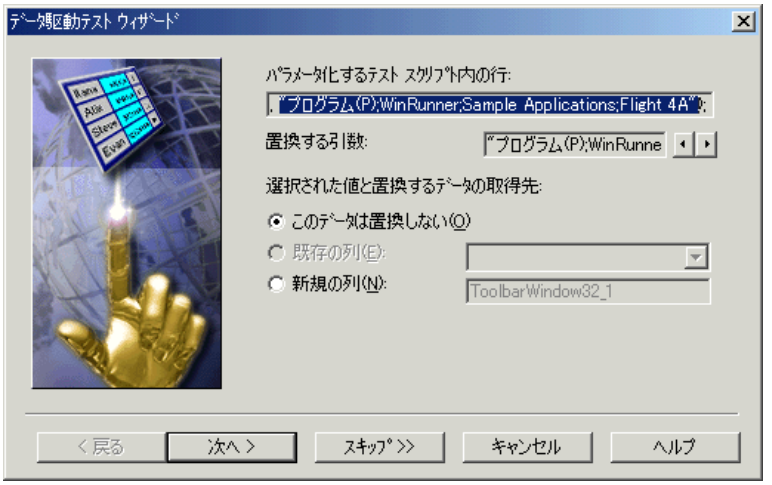
注：テストは手作業でパラメータ化することもできます。詳細については，372 ページ「テスト・スクリプト内の値のパラメータ化」を参照してください。

注：*dat* フォルダにある ***ddt_func.ini*** ファイルには，データ駆動テスト・ウィザードがデータ駆動型テストの作成中にパラメータ化できる TSL 関数がリストされます。このファイルには，標準でパラメータ化できる各関数の引数のインデックスも含まれています。このリストを変更して，パラメータ化できる標準の関数の引数を変更できます。またテストの作成中にユーザ定義関数または他の TSL 関数を含むステートメントのパラメータ化できるように，このリストを変更することもできます。ユーザ定義関数の作成方法については，第 30 章「ユーザ定義関数の作成」を参照してください。

［**次へ**］をクリックします。

どのチェック・ボックスも選択しなければ，［キャンセル］ボタンが有効になります。

4 前の画面で［**データベースからデータをインポートする**］チェック・ボックスを選択した場合は，372 ページ「データベースからのデータのインポート」に進んでください。そうでなければ，次のウィザード画面が開きます。



［**パラメータ化するテストスクリプト内の行**］ボックスに，パラメータ化するテスト・スクリプト行が表示されます。強調表示されている値は，パラメータに置き換えることができます。

［**選択された値と置換するデータの取得先**］ボックスには，パラメータに置き換えることのできる引数（値）が表示されます。矢印を使って置き換える他の引数を選択できます。

選択されたデータを置き換えるかどうか選択し，置き換える場合はその方法も選択します。

- ［**このデータは置換しない**］：このデータはパラメータ化しません。
- ［**既存の列**］：パラメータがこのテストのデータ・テーブル内にすでに存在する場合は，既存のパラメータをリストから選択します。
- ［**新規の列**］：このテストのデータ・テーブル内のこのパラメータに新しいカラムを作成し，選択されたデータをデータ・テーブルのこのカラムに追加します。新しいパラメータの標準の名前は上で選択された TSL ステートメン

ト内のオブジェクトの名前です。この名前を受け入れるか，新しい名前を割り当てます。

359 ページで示したサンプルのフライト予約アプリケーションのテスト・スクリプトには，ユーザが入力した固定値を含むステートメントがいくつか含まれています。

この例では，新しいデータ・テーブルを使用しているので，パラメータはまだありません。この例では，テスト・スクリプトで最初にパラメータ化された行に対し，ユーザは［**新規パラメータのデータ**］ラジオ・ボタンをクリックしています。標準設定では，新しいパラメータはオブジェクトの論理名です。この名前は変更できます。例えば，ここでは新規パラメータの名前が「フライト予定日」に変更されています。

テスト・スクリプト行,

```
edit_set ("Edit", "6");
```

は，次のデータで置換されます。

```
edit_set("Edit",ddt_val(table,"Edit"));
```

テスト・スクリプト行,

```
edit_check_info(" 注文番号：","value",6);
```

は次のデータで置換されます。

```
edit_check_info(" 注文番号：","value",ddt_val(table,"Order_No"));
```

- テスト・スクリプトで行を他にもパラメータ化する場合は，［**次へ**］をクリックします。ウィザードはそのテスト・スクリプトの中で次にパラメータ化できる行を表示します。テスト・スクリプトでパラメータ化できる各行にこの手順を繰り返します。テスト・スクリプトでパラメータ化できる行がなくなると，ウィザードの最終画面が開きます。
- テスト・スクリプトで他の行をパラメータ化せずに最終画面に進むには，［**スキップ**］をクリックします。

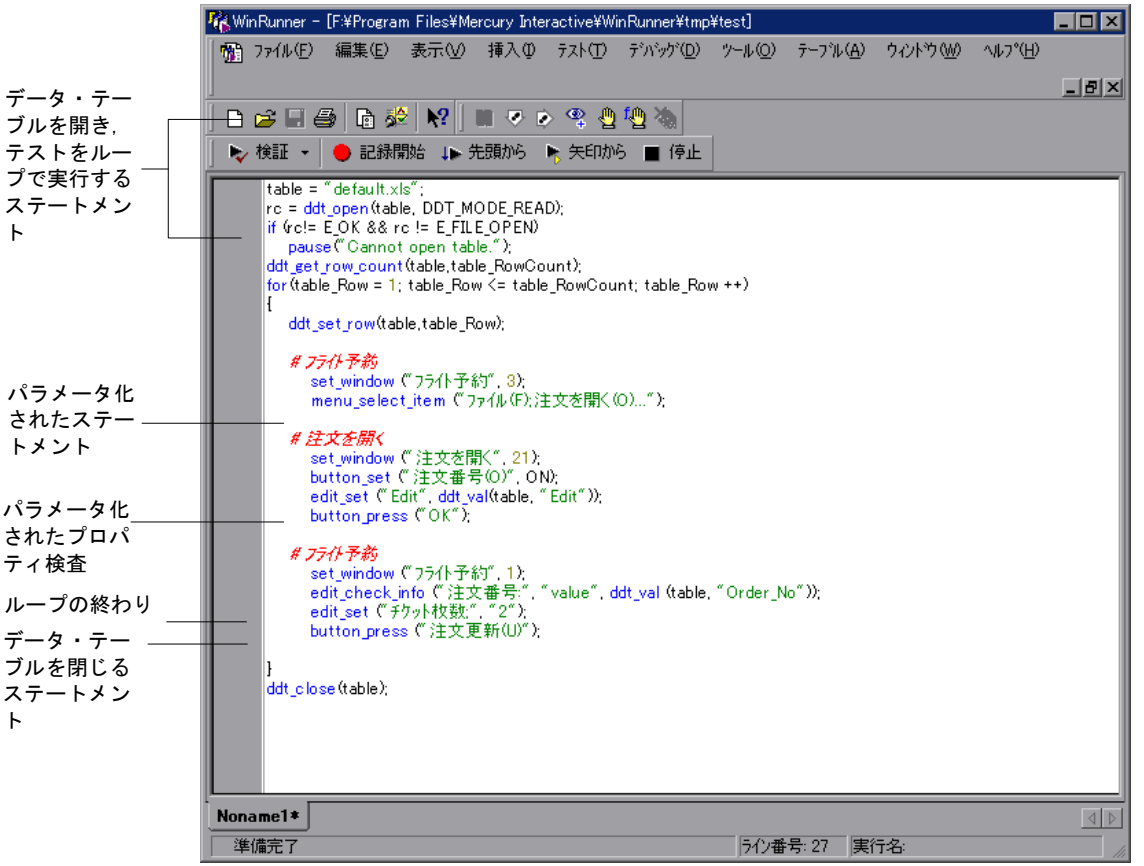
5 ウィザードの最終画面が開きます。

- ウィザードを閉じた後にデータ・テーブルを開く場合は，［**データ テーブルを表示する**］を選びます。

- 前の画面で指定したタスクを実行してウィザードを終了するには［**終了**］をクリックします。
- テスト・スクリプトに変更を一切行わずにウィザードを終了するには，［**キャンセル**］をクリックします。

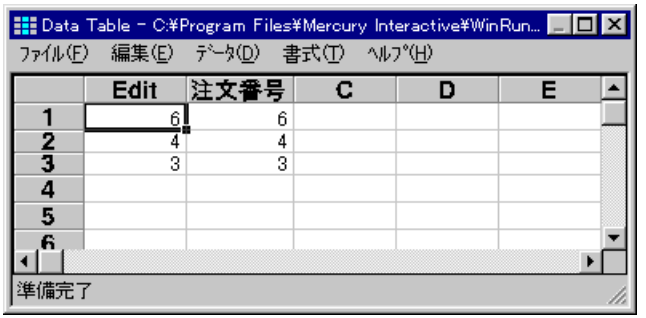
注 : テスト・スクリプトをパラメータ化した後，最後のウィザード画面が表示される前に［**キャンセル**］をクリックすると，データ・テーブルには追加したデータが含まれます。データ・テーブルにデータを保存する場合は，データ・テーブルを開いて，保存します。

データ駆動テスト・ウィザードの実行が終了すると，359 ページで例に示したサンプルのテスト・スクリプトが下に示すように変更されます。



データ・テーブルを開く（[**テーブル**] > [**データテーブル**]）と，[**データテーブルを開く，または作成します**] ダイアログ・ボックスが開きます。データ駆動テスト・ウィザードで指定したデータ・テーブルを選択します。データ・テーブルを開くとテーブルで作成されたエントリを見ることができ，データを編集できます。

前の例に対して，次のエントリがデータ・テーブルに作成されています。



手作業によるデータ駆動テストの作成

データ駆動テストは，データ駆動テスト・ウィザードを使わずに，手作業で作成することもできます。データ駆動テストを手作業で作成するには，以下の手順をすべて行う必要があります。

- データ・テーブルの定義
- テスト・スクリプトへの，データ・テーブルを開閉してテストをループで実行するステートメントの追加
- データベースからのデータのインポート（任意）
- データ・テーブルの作成と，テスト・スクリプトの値のパラメータ化

データ・テーブルの定義

次のステートメントをスクリプトのパラメータ化される部分の直前に追加します。これにより，データ・テーブルの名前とパスを識別します。単一のテストで複数のデータ・テーブルを使用することも，複数のテストで単一のデータ・テーブルを使用することもできます。詳細については，402 ページ「データ駆動型テスト作成のガイドライン」を参照してください。

```
table="Default.xls";
```

データ・テーブルに異なる名前を使っている場合は，該当する名前で置き換えてください。標準設定では，データ・テーブルは，テストのフォルダに格納されています。データ・テーブルを他の場所に格納する場合は，このステートメントにパスも含めなければなりません。

例えば，

table1 = “default.xls”;

は，テスト・フォルダに標準の名前で格納されているデータ・テーブルです。

table2 = “table.xls”;

は，テスト・フォルダに新しい名前で格納されているデータ・テーブルです。

table3 = “C:\\Data-Driven Tests\\Another Test\\default.xls”;

は，標準の名前と新しいパスを持つデータ・テーブルです。このデータ・テーブルは，別のテストのフォルダに格納されています。

注：WinRunner のバージョン 5.0 と 5.01 で作成されたスクリプトには，代わりに次のようなステートメントが含まれている場合があります。

table=getvar("testname") & "\\Default.xls";

このステートメントは WinRunner のバージョン 6.0 以降でも有効です。しかし WinRunner 6.0 で作成されたスクリプトでは相対パスを使用するため，ステートメントに完全パスを指定する必要はありません。

テスト・スクリプトへのデータ・テーブルを開閉してテストをループで実行するステートメントの追加

次のステートメントをテーブルを定義した直後にテスト・スクリプトに追加します。

```
rc=ddt_open (table);
if (rc!= E_OK && rc != E_FILE_OPEN)
    pause("Cannot open table.");
ddt_get_row_count(table,table_RowCount);
for(table_Row = 1; table_Row <= table_RowCount ;table_Row ++ )
{
    ddt_set_row(table,table_Row);
```

これらのステートメントは，テストのデータ・テーブルを開き，データ・テーブルのデータの各行の最後にある括弧で括られたステートメントを実行します。

次のステートメントを，テスト・スクリプトのパラメータ化された部分の直後に追加します。

```
}
ddt_close (table);
```

これらのステートメントは，各行に対して上の括弧内に含まれるステートメントを実行します。これらは，データ・テーブルの次の行のデータを使って，テストを連続して繰り返し駆動します。これらは，データ・テーブルの次の行が空だと，これらのステートメントは括弧内のステートメントの実行を停止し，データ・テーブルを閉じます。

データベースからのデータのインポート

ddt_update_from_db と **ddt_save** ステートメントをテスト・スクリプトの **ddt_open** ステートメントの後に追加しなければなりません。インポートするデータを指定するには，Microsoft Query を使用します。詳細については，372 ページ「データベースからのデータのインポート」を参照してください。**ddt_** 関数の詳細については，396 ページ「データ駆動型テストでの TSL 関数の使用」または「**TSL リファレンス**」を参照してください。

テスト・スクリプト内の値のパラメータ化

359 ページ「変換用基本テストの作成」のサンプルのテスト・スクリプトには，ユーザが入力した固定値を含む複数のステートメントが含まれます。

```
edit_set("Edit", "6");

edit_check_info(" 注文番号 : ","value",6);
```

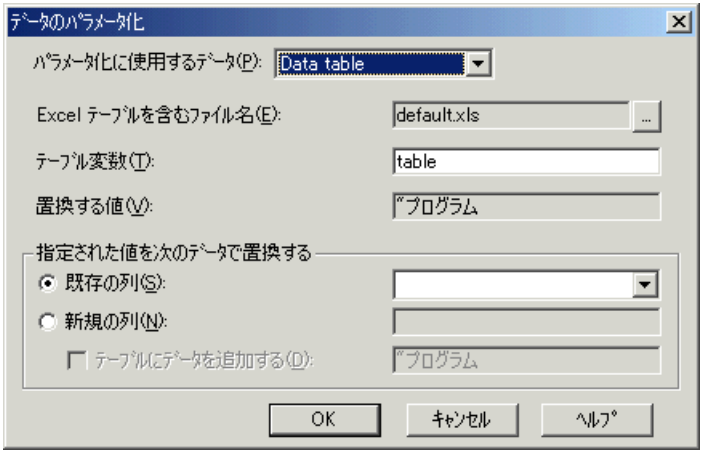
［**データのパラメータ化**］ダイアログ・ボックスを使用して，ステートメントをパラメータ化して，パラメータでデータを置き換えます。

データ・テーブルを使ってステートメントをパラメータ化するには，次の手順を実行します。

1 テスト・スクリプトで，最初に現れるパラメータ化したいデータを選択します。例えば，上のテスト・スクリプトの最初の **edit_set** ステートメントでは，「6」を選択します。

2 ［**テーブル**］＞［**データのパラメータ化**］を選択します。［データのパラメータ化］ダイアログ・ボックスが開きます。

3［**パラメータ化に使用するデータ**］ボックスで，「**Data table**」を選択します。



4［**Excel テーブルを含むファイル名**］**ボックスで**，データ・テーブルの標準の名前と場所を受け入れる，データ・テーブルで別の名前を入力する，あるいは参照ボタンを使ってデータ・テーブルのパスを配置できます。データ・テーブルの標準の名前は「**default.xls**」で，テスト・フォルダに格納されていますので注意してください。
前回テストで別のデータを使用している場合は，代わりにその名前が表示されます。

［**新規の列**］をクリックします。WinRunner がボックス内にパラメータの名前を示します。この名前を受け入れるか，別の名前を選択できます。WinRunner は，データ・テーブルのパラメータと同じ名前でカラムを作成します。

テスト・スクリプトで選択された引用符付きのデータが［**テーブルにデータを追加する**］ボックスに現れます。

- テスト・スクリプト内で現在選択されているデータをデータ・テーブルに含める 場合は，［**テーブルにデータを追加**］チェック・ボックスを選択します。このボックスでデータを変更できます。
- 現在選択されているデータをデータ・テーブル内のテスト・スクリプトに含めたくない場合は，［**テーブルにデータを追加**］チェック・ボックスをクリアします。
- データ・テーブルにすでにあるカラムにデータを割り当てる既存のパラメータに，データを割り当てることもできます。既存のパラメータを使用する場合は，［**既存の列**］をクリックして，リストから既存のカラムを選択します。

5 ［**OK**］をクリックします。

テスト・スクリプトで，テスト・スクリプト内で選択されたデータが **ddt_val** ステートメントに置き換えられます。このステートメントにはテーブルの名前とデータ・テーブル内に対応するカラムで作成したパラメータの名前が含まれます。

この例では，値「6」が，テーブル名とパラメータ「Edit」を含む **ddt_val** ステートメントで置き換えられ，元のステートメントは次のようになります。

```
edit_set ("Edit",ddt_val(table,"Edit"));
```

データ・テーブルに，割り当てたパラメータの名前を持つ新しいカラムが作成されます。この例では，「Edit」というヘッダの新しいカラムが作成されます。

6 パラメータ化したい各引数に，手順 1 から 5 までを繰り返します

ddt_val 関数の詳細については，396 ページ「データ駆動型テストでの TSL 関数の使用」または「**TSL リファレンス**」を参照してください。

データ・テーブルの準備

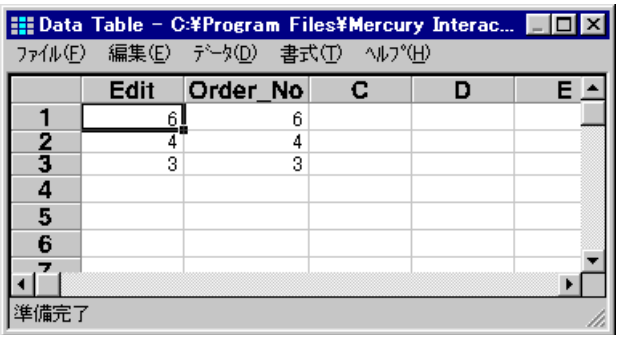
各データ駆動型テストに，少なくとも 1 つはデータ・テーブルを用意しなければなりません。データ・テーブルには，WinRunner がデータ駆動型テストの変数を置換するのに使う値が含まれます。

データ駆動テスト・ウィザードを使う場合でも［データのパラメータ化］ダイアログ・ボックスを使う場合でも，通常はテストの変換工程の一環としてデータ・テーブルを作成しますが，Excel で個別にテーブルを作成して後からテストにリンクさせることも可能です。

テストを作成したら，データをテーブルに手作業で追加することも，既存のデータベースからインポートすることもできます。

次のデータ・テーブルに 3 セットのデータを示します。これらはこの章で使うテスト用に入力されたものです。データの最初のセットは，WinRunner の［**テーブル**］＞［**データのパラメータ化**］コマンドを使って入力されたもので

す。後の 2 つのデータ・セットはデータ・テーブルに手作業で入力されたものです。



- データ・テーブルの各行は一般に，テストの 1 回の繰り返しの間に，パラメータ化されたすべてのフィールドに対して WinRunner が発行する値を表します。例えば，10 行のテーブルに関連付けられたテスト内のループは 10 回実行されます。
- テーブル内の各カラムは，1 つのパラメータに対する値のリストを表します。テストの各繰り返しで，この値のうちの 1 つが使用されます。

注：カラム・ヘッダの最初の文字は，アンダスコア（_）か英字でなければなりません。以降の文字はアンダスコアでも英字でも，数字でもかまいません。

手作業によるデータのデータ・テーブルへの追加

データ・テーブルに手作業でデータを追加するには，データ・テーブルを開いて，適切なカラムに値を入力します。

データ・テーブルにデータを手作業で追加するには，次の手順を実行します。

1 ［**テーブル**］＞［**データテーブル**］を選択します。［**データテーブルを開く，または作成します**］ダイアログ・ボックスが開きます。テスト・スクリプトで指定したデータ・テーブルを選択して開くか，新しい名前を入力して新しいデータ・テーブルを作成します。データ・テーブルがデータ・テーブル・ビューアで開きます。

2 テーブルに手作業でデータを入力します。

3 空のセルにカーソルを移動し，データ・テーブルで［**ファイル**］＞［**上書き保存**］を選択します。

注 : データ・テーブルに行った変更は，データ・テーブルを閉じることで自動的に保存されるわけではありません。データ・テーブルで［ファイル］＞［上書き保存］を選択するか，**ddt_save** ステートメントを使って，データ・テーブルを保存する必要があります。データ・テーブルのメニュー・コマンドについては，376 ページ「データ・テーブルの編集」を参照してください。**ddt_save** 関数については，396 ページ「データ駆動型テストでの TSL 関数の使用」を参照してください。データ駆動型テストを実行する際に，データ・テーブル・ビューアを開いておく必要はありません。

データベースからのデータのインポート

データ・テーブルにデータを手作業で追加するほかに，既存のデータベースからのデータをテーブルにインポートできます。データのインポートには，Microsoft Query か Data Junction を使用できます。データベースからデータをインポートする方法の詳細については，383 ページ「データベースからのデータのインポート」を参照してください。

データ・テーブルの編集

データ・テーブルには，WinRunner がパラメータ化された入力フィールドのために使い，テストを実行するときに検査する値が含まれます。データ・テーブル内の情報は，テーブルに直接タイプ入力して編集できます。データ・テーブルの使用法は，Excel のスプレッドシートと同じです。セルに Excel の式と関数を挿入することも可能です。

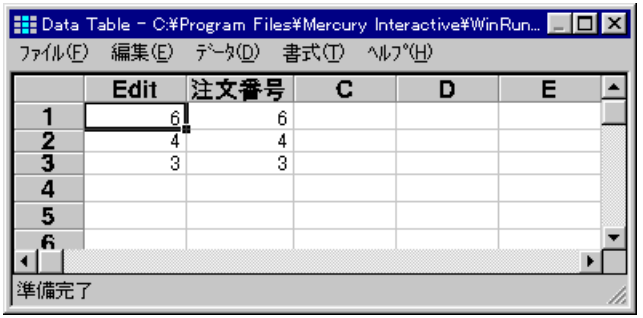
注 : データの形式を変更（例えば日付の変更など）したくない場合は，データ・テーブルに入力する文字列は引用符（'）で始めなくてはなりません。これにより，セル内の文字列の形式を変更しないようエディタに指示します。

データ・テーブルを編集するには，次の手順を実行します。

1 テストを開きます。

2 ［**テーブル**］＞［**データテーブル**］を選択します。［**データテーブルを開く，または作成します**］ダイアログ・ボックスが開きます。

3 テストのデータ・テーブルを選択します。テストのデータ・テーブルが開きます。



4 この後で説明するメニュー・コマンドを使って，データ・テーブルを編集します。

5 カーソルを空のセルに移動し，［**ファイル**］＞［**上書き保存**］を選択して変更を保存します。

6 ［**ファイル**］＞［**閉じる**］を選択し，データ・テーブルを閉じます。

［ファイル］メニュー

［ファイル］メニューから，データ・テーブルのインポート，エクスポート，終了，保存，印刷が行えます。WinRunner は，テストのデータ・テーブルを自動的にテスト・フォルダに保存し，これに **default.xls** という名前を付けます。**default.xls** データ・テーブル以外にもデータ・テーブルを開いて保存できます。これにより，1 つのテスト・スクリプトで複数のデータ・テーブルを使用することもできます。

［ファイル］メニューには以下のコマンドがあります。

［ファイル］コマンド	説明
［新規作成］	新しいデータ・テーブルを作成します。
［開く］	既存のデータ・テーブルを開きます。**ddt_open** 関数によってすでに開かれているデータ・テーブルを開こうとすると，データ・テーブル・エディタでテーブルを開く前に，開いているデータ・テーブルを保存して閉じるよう指示されます。
［上書き保存］	アクティブなデータ・テーブルを同じ名前で同じ場所に保存します。データ・テーブルは，Microsoft Excel 形式でもタブ付きのテキスト・ファイル形式でも保存できます。
［名前を付けて保存］	［ファイル名を付けて保存］ダイアログ・ボックスを開きます。このダイアログ・ボックスで，データ・テーブルを保存する名前と場所を指定できます。データ・テーブルは，Microsoft Excel 形式でもタブ付きのテキスト・ファイル形式でも保存できます。
［インポート］	既存のテーブル・ファイルをデータ・テーブルにインポートします。インポートできるのは，Microsoft Excel ファイルかタブで区切られたテキスト・ファイルです。**ddt_open** 関数によってすでに開かれているデータ・テーブルを開こうとすると，データ・テーブル・エディタでテーブルを開く前に，開いているデータ・テーブルを保存して閉じるよう指示されます。 Excel ファイルの場合は，最初の行のセルがデータ・テーブル・ビューアのカラム・ヘッダになります。新しいテーブル・ファイルをインポートすると，現在データ・テーブルにあるすべてのデータが置き換えられます。
［エクスポート］	データ・テーブルを Microsoft Excel ファイルまたはタブで区切られたテキスト・ファイルとして保存します。Excel ファイルとして保存する場合は，データ・テーブル・ビューアのカラム・ヘッダが最初の行のセルになります。

［ファイル］コマンド	説明
［閉じる］	データ・テーブルを閉じます。データ・テーブルに行った変更は，データ・テーブルを閉じるときに自動的には保存されません。変更を保存するには，［上書き保存］コマンドを使用してください。
［印刷］	データ・テーブルを印刷します。
［印刷設定］	プリンタ，ページの向き，用紙サイズを選択できます。
［終了］	データ・テーブルを閉じます。データ・テーブルを閉じても，変更は自動的には変更されません。変更を保存するには，［保存］コマンドを使用します。

［編集］メニュー

［編集］メニューから，データ・テーブルで選択したセルの移動，コピー，検索を行えます。［編集］メニューには以下のコマンドがあります。

［編集］コマンド	説明
［切り取り］	データ・テーブルの選択範囲を切り取り，それをクリップボードにコピーします。
［コピー］	データ・テーブルの選択範囲を，クリップボードにコピーします。
［貼り付け］	クリップボードの内容をデータ・テーブルの現在選択されている部分に貼り付けます。
［値を貼り付け］	クリップボードからデータ・テーブルの現在選択されている部分に値を貼り付けます。値に適用されている書式は無視されます。また，数式の結果のみが貼り付けられ，数式そのものは無視されます。
［クリア］ - ［すべて］	選択されたセルの書式（［書式］メニューのコマンドで書式が指定されている場合）と値（数式も含む）の両方を消去します。
［クリア］- ［書式］s	［書式］メニューのコマンドで書式が指定されている場合，選択されたセルの書式を消去します。選択されたセルの値（数式も含む）は消去しません。

［編集］コマンド	説明
［クリア］ - ［内容］	選択されたセルの値（数式も含む）だけを消去します。選択されたセルの書式は消去しません。
［挿入］	現在選択している場所に空のセルを挿入します。挿入されたセルに隣接するセルがずれて，新しいセル用の場所が作られます。
［削除］	現在選択されている部分を削除します。削除されたセルに隣接するセルがずれて，なくなったセルの隙間を埋めます。
［右方向へコピー］	選択された範囲の一番左のセルのデータを右のすべてのセルにコピーして，適切な範囲を埋めます。
［左方向へコピー］	選択された範囲の一番上のセルのデータを下のすべてのセルにコピーして，適切な範囲を埋めます。
［検索］	指定された値を含むセルを検索します。テーブルの行ごとまたはカラムごとに検索したり，大文字と小文字を区別する指定をしたり，完全に一致するセルだけを検索したりできます。
［置換］	指定された値を含むセルを検索し，それを別の値で置換します。テーブルの行ごとまたはカラムごとに検索したり，大文字と小文字を区別したり，セル全体が一致するものだけを検索したりすることができます。また，すべてを置き換えることもできます。
［ジャンプ］	指定されたセルに移動します。移動先のセルがアクティブなセルになります。

［データ］メニュー

［データ］メニューから，数式の再計算，セルのソート，およびオート・フィル・リストの編集を行えます。［データ］メニューには以下のコマンドがあります。

［データ］コマンド	説明
［再計算］	データ・テーブルで任意の数式セルの再計算を行います。

［データ］コマンド	説明
［並べ替え］	選択されたセルを行，カラム，キーによってソートします。
［自動挿入リスト］	自動挿入リストの作成，編集，削除を行います。 自動挿入リストには，月や曜日などといったよく使われるテキストの並びが含まれます。新しいリストを追加するときには，各項目間をセミコロンで区切ります。 自動挿入リストを使うには，最初の項目をデータ・テーブルのセルに入力します。カーソルを横または下にドラッグすると，WinRunner は自動挿入リストに従って自動的にその範囲のセルを埋めます。

［書式］メニュー

［書式］メニューを使って選択されたセル（単数または複数）のデータの書式を設定します。［書式］メニューには以下のコマンドがあります。

［書式］コマンド	説明
［標準］	書式を［標準］に設定します。標準形式では必要な桁数の小数を使い，カンマは使わずに表示します。
［通貨 (0)］	書式を，カンマを使い，小数は使わない，通貨形式に設定します。
［通貨 (2)］	書式を，カンマを使い，2 桁の小数を使う，通貨形式に設定します。
［固定］	書式を，カンマを使い，小数を使わない固定精度の形式に設定します。
［パーセント］	書式を，小数なしの百分率形式に設定します。数値は末尾にパーセント記号（%）を付けて百分率として表示されます。
［分数］	書式を分数形式に設定します。
［指数］	書式を，小数部分 2 桁の科学的表記法に設定します。
［yyyy/MM/dd］	日付の書式を yyyy/MM/dd 形式にします。
［h:mm AM/PM］	時間の書式を h:mm AM/PM 形式にします。

［書式］コマンド	説明
ユーザ設定数値	数値をユーザが指定した数値形式に設定します。
検証ルール	セルまたはセルの範囲に入力されたデータをテストするための検証規則を設定します。検証規則はテストを行うための式，および検証が失敗したときに表示するテキストからなります。

データ・テーブルの技術仕様

次の表にデータ・テーブルの技術仕様を示します。

最大カラム数	256
最大行数	16,384
最大のカラム幅	1020 文字
最大の行の高さ	409 ポイント
最大の式の長さ	1024 文字
数値の精度	15 桁
最大の正数	9.99999999999999E307
最大の負数	-9.99999999999999E307
最小の正数	1E-307
最小の負数	-1E-307
テーブル・フォーマット	タブで区切られたテキスト・ファイルまたは Microsoft Excel ファイル。
有効なカラム名	カラム名に空白文字を含むことはできません。文字，番号，アンダスコア（_）だけを含むことができます。

データベースからのデータのインポート

既存のデータベースからデータ・テーブルにデータをインポートするためには，データ駆動テスト・ウィザードを使ってデータを指定しなければなりません。［**データベースからデータをインポートする**］チェック・ボックスを選択すると，データ駆動テスト・ウィザードからデータベースに接続するのに使用するプログラムを指定するよう指示されます。ODBC/Microsoft Query または Data Junction を選択できます。

データベースからデータをインポートするためには，Microsoft Query か Data Junction をお使いのマシンにインストールしなければなりません。Microsoft Query は，Microsoft Office の「カスタム・インストール」でインストールできます。Data Junction は，自動的に WinRunner のパッケージに含まれているものではありません。Data Junction を購入する場合は , お近くの Mercury の製品取扱代理店まで連絡してください。Data Junction の使用法については Data Junction パッケージのマニュアルを参照してください。

注：データ・テーブルのデータをデータベースの既存のカラムからのデータで置き換えるよう選択し，すでにデータ・テーブル内に同じヘッダを持つカラムがすでに存在する場合は，そのカラムのデータはデータベースから自動的に更新されます。データベースからのデータは，データベースからインポートされたすべての行に対して，データ・テーブル内の関連カラムのデータを上書きします。

Microsoft Query を使用したデータベースからのデータのインポート

Microsoft Query を使って，データ・ソースを選択し，データ・ソース内にクエリを定義します。

Microsoft Query のオプションの設定

［**次を使用してデータベースに接続**］オプションに［Microsoft Query］を選択すると，以下のウィザード画面が現れます。

次のオプションを選択できます。

- ［**新規クエリの作成**］: Microsoft Query が開いて，下のフィールドに指定した名前で，新しい ODBC ***.sql*** クエリ・ファイルを作成できます。詳細については，385 ページ「新しいソース・クエリ・ファイルの作成」を参照してください。
- ［**既存のクエリをコピー**］: ウィザードに［**ソース クエリ ファイルを選択してください**］画面が開いて，別のクエリ・ファイルから既存の ODBC クエリをコピーできます。詳細については，386 ページ「ソース・クエリ・ファイルの選択」を参照してください。
- ［**SQL ステートメントを指定**］: ウィザードに［**SQL ステートメントを指定**］画面が開いて，接続文字列と SQL ステートメントを指定できます。詳細については，387 ページ「SQL ステートメントの指定」を参照してください。
- ［**新規クエリ ファイル**］: データベースからインポートするデータ用の新しい ***.sql*** クエリ・ファイルの標準の名前を表示します。参照ボタンを使って，他の ***.sql*** クエリ・ファイルを参照できます。
- ［**最大行数**］: このチェック・ボックスを選択して，インポートするデータベースの最大行数を入力します。このチェック・ボックスをクリアにすると，行数

の制限がなくなります。このオプションは **db_check** ステートメントにパラメータを追加します。詳細ついては「**TSL リファレンス**」を参照してください。

➤ [**Microsoft Query の使用法を表示**]: Microsoft Query の使用法を説明する画面を表示します。

新しいソース・クエリ・ファイルの作成

最後のステップで［**新規クエリを作成**］を選択すると Microsoft Query が開きます。新規または既存のデータ・ソースを選択してクエリを定義し，定義し終わったら、次のことをします。

➤ バージョン 8.00 の場合，［Microsoft Query ウィザード］の［完了］画面で［**プログラムを終了し，WinRunner に戻ります**］をクリックしてから［**完了**］をクリックして，Microsoft Query を終了します。あるいは，［**Microsoft Query でデータの表示またはクエリの編集を行う**］をクリックして，［**完了**］をクリックします。データを表示または編集した後は，［**ファイル**］＞［**プログラムを終了し，WinRunner に戻ります**］を選択して Microsoft Query を閉じ，WinRunner に戻ります。

クエリの定義が終了したら，データ駆動テスト・ウィザードに戻り，テストのデータ駆動型テストへの変換を終了します。詳細については，366 ページのステップ 4 を参照してください。

ソース・クエリ・ファイルの選択

最後の手順で［**既存のクエリをコピー**］を選択すると次の画面が開きます。

クエリ・ファイルのパス名を入力するか，［**参照**］ボタンを使って，ファイル名を指定します。クエリ・ファイルを選択すると［**表示**］ボタンを使って表示するファイルを開くことができます。

終了したら，［**次へ**］をクリックして，データ駆動型テストの作成を終了します。詳細については 366 ページのステップ 4 を参照してください。

SQL ステートメントの指定

最後の手順で［**SQL ステートメントを指定**］を選択すると，次の画面が表示されます。

この画面では，接続文字列と SQL ステートメントを指定しなくてはなりません。

- ［**接続文字列**］：接続文字列を入力するか，［**作成**］をクリックして，［データソースの選択］ダイアログ・ボックスを開きます。このダイアログ・ボックスでは，ボックスに接続文字列を挿入する ***.dsn*** ファイルを選択できます。
- ［**SQL**］：SQL ステートメントを入力します。

終了したら，［**次へ**］をクリックして，データ駆動型テストの作成を終了します。詳細については，366 ページのステップ 4 を参照してください。

Microsoft Query を使ってデータベースからデータをインポートすると，クエリ情報が ***msqrN.sql*** （「N」は一意の数値です）と言う名前のクエリ・ファイルに保存されます。標準設定では，このファイルはテスト・フォルダ（標準のデータ・テーブルが格納されている）に格納されます。データ駆動テスト・ウィザードは，完全パスではなく，相対パスを使って **ddt_update_from_db** ステートメントを挿入します。

テスト実行中に相対パスを指定すると，WinRunner はテスト・フォルダでクエリ・ファイルを検索します。**ddt_update_from_db** ステートメントのクエリ・ファイルに完全パスを指定すると，WinRunner は完全パスを使って，クエリ・ファイルの場所を見つけます。

Microsoft Query の使用法については，Microsoft Query のマニュアルを参照してください。

データ駆動型テストの実行と分析

データ駆動型テストの実行および分析の方法は，WinRunner テストと同じです。次の 2 つの節で，実行と分析の方法について説明します。

テストの実行

データ駆動型テストを作成したら，通常の WinRunner テストと同じようにこれを実行します。WinRunner は，テスト・スクリプト内のパラメータをデータ・テーブルの値で置き換えます。WinRunner でテストを実行すると，データ・テーブルが開きます。テストの各繰り返しで，アプリケーションで記録した操作を実行し，指定した検査を行います。テストの実行の詳細については，第 19 章「テスト実行について」を参照してください。

データをデータベースからインポートする場合は，テストを実行すると，**ddt_update_from_db** 関数がデータベースのデータでデータ・テーブルを更新します。データベースからのデータのインポートについては，372 ページ「データベースからのデータのインポート」を参照してください。
ddt_update_from_db 関数については，396 ページ「データ駆動型テストでの TSL 関数の使用」または「**TSL リファレンス**」を参照してください。

テスト結果の分析

テストの実行が完了したら，通常の WinRunner テストと同様，テスト結果を表示できます。［テスト結果］ウィンドウには，GUI チェックポイントやビットマップ・チェックポイント，ファイル比較，エラー・メッセージなど，テスト実行中に生じる主なイベントの記述が含まれます。各反復の間に，何らかのイベントが生じると，テスト結果は，各反復のイベントに対して個々の結果を記録します。

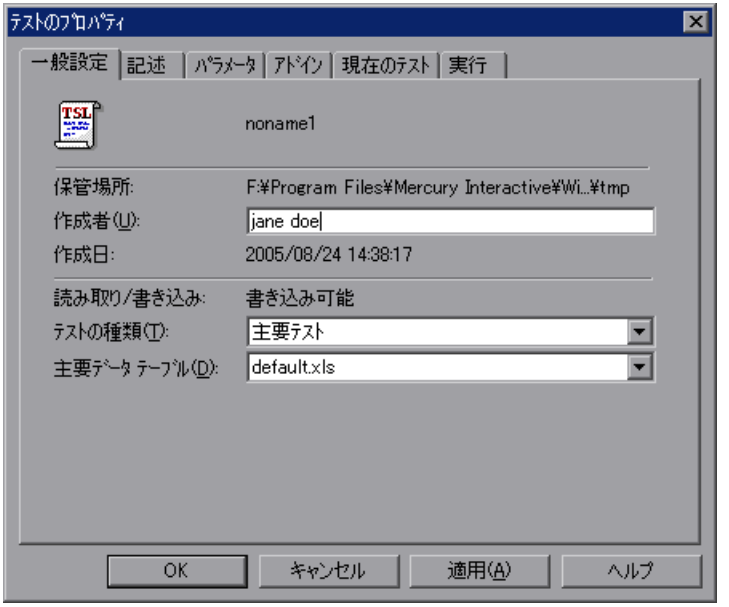
例えば，テスト・スクリプトに **ddt_report_row** ステートメントを挿入すると，WinRunner はテスト結果にデータ・テーブルの行を出力します。テスト・スクリプト内の **ddt_report_row** ステートメントの各反復につき，［テスト結果］ウィンドウ のテスト・ログ・テーブルに行が 1 行挿入されます。この行には，［イベント］カラムに「テーブルの行」と表示されています。この行をダブルクリックすると，テストの各反復において WinRunner が使用するすべてのパラメータ化されたデータが表示されます。**ddt_report_row** 関数については，401 ページ「データ・テーブルのアクティブ行のテスト結果への報告」または「**TSL リファレンス**」を参照してください。テスト結果の表示については，第 2 章「統一レポート・ビューでのテスト結果の分析」を参照してください。

テストへのメイン・データ・テーブルの割り当て

テストへのメイン・データ・テーブルの設定は，［テストのプロパティ］ダイアログ・ボックスの［**一般設定**］タブで簡単に行えます。メイン・データ・テーブルは，［**ツール**］＞［**データ テーブル**］を選択するか，データ駆動テスト・ウィザードを開いたときに，標準で選択されるテーブルです。

テストにデータ・テーブルを割り当てるには，次の手順を実行します。

1



2 [**主要データ テーブル**] リストから，割り当てたいデータ・テーブルを選択します。テスト・フォルダに格納されているデータ・テーブルは，すべてリストに表示されます。

3 [**OK**] をクリックします。選択したデータ・テーブルが新しいメイン・データ・テーブルとして割り当てられます。

注 : [テストのプロパティ] ダイアログ・ボックスからメイン・データ・テーブルを選択した後で，別のデータ・テーブルを開こうとすると，最後に開いたデータ・テーブルがメイン・データ・テーブルがメイン・データ・テーブルになります。

データ駆動型チェックポイントとビットマップ同期化ポイントの使用

データ駆動テストを作成すると，TSL ステートメントで固定値をパラメータ化できます。しかし，GUI チェックポイントとビットマップ・チェックポイントおよびビットマップ同期化ポイントには特定の固定値を含めることができません。その代わり，これらには以下の要素を含めることができます。

- GUI チェックポイント・ステートメント（**obj_check_gui** または **win_check_gui**）には，テストの ***chklist*** フォルダに格納されたチェックリストへの参照や，テストの *exp* フォルダに格納された期待結果を含めることができます。
- ビットマップ・チェックポイント・ステートメント（**obj_check_bitmap** または **win_check_bitmap**）またはビットマップ同期化ポイント・ステートメント（**obj_wait_bitmap** または **win_wait_bitmap**）には，テストの ***exp*** フォルダに格納されたビットマップへの参照を含めることができます。

注：データ駆動型テストで GUI オブジェクトのプロパティを検査する場合は，GUI チェックポイントを作成するより，単数のプロパティ検査を作成するほうがよいでしょう。単数のプロパティ検査はチェックリストを含まないため，簡単にパラメータ化できます。［**挿入**］＞［**GUI チェックポイント**］＞［**単数プロパティ**］**コマンドを使って**，チェックリストなしでプロパティ検査を作成します。データ駆動型テストでの単数のプロパティ検査の使い方については，359 ページ「変換用基本テストの作成」を参照してください。オブジェクトの単数のプロパティ検査については，第 9 章「GUI オブジェクトの検査」を参照してください。

GUI チェックポイント，ビットマップ・チェックポイント，ビットマップ同期化ポイントのステートメントをパラメータ化するには，各期待結果に対する参照ごとにデータ・テーブルに仮の値を挿入します。まず，各チェックポイントあるいはビットマップ同期化ポイントごとに個別のカラムを作成します。次に，キャプチャされた期待結果を表す仮の値をカラムに入力します。仮の値はそれぞれ，一意の名前を持ちます（例えば，gui_exp1，gui_exp2 など）。［更新］モードでテストを実行すると，WinRunner はテストの各繰り返しで（つまり，データ・テーブルの各行に対して）期待結果をキャプチャし，テストの *exp* フォルダに結果をすべて保存します。

- GUI チェックポイント・ステートメントに対しては，WinRunner はオブジェクト・プロパティの期待値をキャプチャします。

➤ ビットマップ・チェックポイント・ステートメントまたはビットマップ同期化ポイント・ステートメントに対しては，WinRunner はビットマップをキャプチャします。

データ駆動チェックポイントまたはビットマップ同期化ポイントを作成するには，次の手順を実行します。

1 記録またはプログラミングによって最初のテストを作成します。

下に示す記録済みのテスト例では，メモ帳アプリケーションの［検索］ダイアログ・ボックスが開き，テキストを検索して適切なメッセージが表示されるかどうかを検査します。この例では GUI チェックポイント，ビットマップ・チェックポイント，同期化ポイントがすべて使用されていることに注意してください。

```
set_window (" 無題 - メモ帳 ", 12);
menu_select_item (" 検索 (S); 検索 (F)...");
set_window (" 検索 ", 5);
edit_set (" 検索する文字列 (N):", "John");
button_press (" 次を検索 (N)");
set_window(" メモ帳 ", 10);
obj_check_gui("Message", "list1.ckl", "gui1", 1);
win_check_bitmap(" メモ帳 ", "img1", 5, 30, 23, 126, 45);
obj_wait_bitmap("Message", "img2", 13);
set_window (" メモ帳 ", 5);
button_press ("OK");
set_window (" 検索 ", 4);
button_press (" キャンセル ");
```

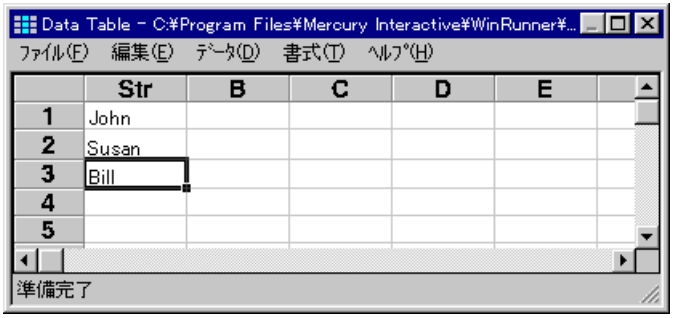
2 データ駆動テスト・ウィザード（［**テーブル**］＞［**データ駆動テスト ウィザード**］）を使ってこのスクリプトをデータ駆動テストにし，テスト・スクリプト内のステートメントのデータの値をパラメータ化します。詳細については，362 ページ「データ駆動テスト・ウィザードを使ったデータ駆動型テストの作成」を参照してください。また，テスト・スクリプトへのこうした変更は手作業でも行えます。詳細については，370 ページ「手作業によるデータ駆動テストの作成」を参照してください。

次の例では，データ駆動テストはいくつかの異なる文字列を検索します。WinRunner は，これらすべての文字列をデータ・テーブルから読み取ります。

```
set_window (" 無題 - メモ帳 ", 12);
menu_select_item (" 検索 (S); 検索 (F)...");
```

```
table = "default.xls";
rc = ddt_open(table, DDT_MODE_READ);
if (rc!= E_OK && rc != E_FILE_OPEN)
    pause("Cannot open table.");
ddt_get_row_count(table,RowCount);
for (i = 1; i <= RowCount; i++) {
    ddt_set_row(table,i);
    set_window (" 検索 ", 5);
    edit_set (" 検索する文字列 (N):", ddt_val(table, "Str"));
    button_press (" 次を検索 (N)");
    set_window(" メモ帳 ", 10);

    # まだパラメータ化されていない GUI チェックポイント・ステートメント。
    obj_check_gui("message", "list1.ckl", "gui1", 1);

    # まだパラメータ化されていないビットマップ・チェックポイント。
    win_check_bitmap(" メモ帳 ", "img1", 5, 30, 23, 126, 45);

    # まだパラメータ化されていない同期化ポイントのステートメント。
    obj_wait_bitmap("message", "img2", 13);
    set_window (" メモ帳 ", 5);
    button_press ("OK");
}
ddt_close(table);


set_window (" 検索 ", 4);
button_press (" キャンセル ");
```

例えば，データ・テーブルは次のようになります。

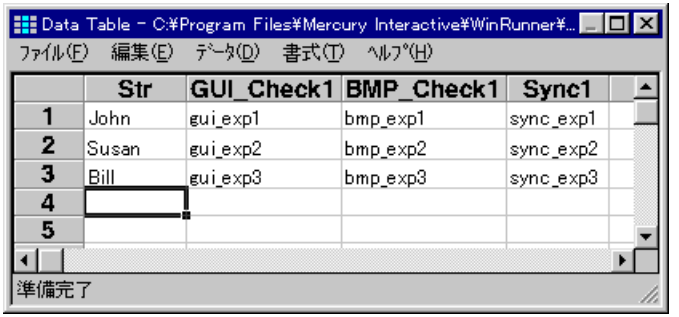


データ駆動型テストの GUI チェックポイントとビットマップ・チェックポイントおよび同期化ポイントは，テスト実行の 2 番目と 3 番目の繰り返しで失敗し

ます。チェックポイントと同期化ポイントは，これらのポイントの値が元の記録済みテストの「John」という文字列を使ってキャプチャされたため失敗します。したがって，これらはデータベースから取り出されたその他の文字列と一致しません。

3 データ・テーブルに，パラメータ化される各チェックポイントまたは同期化ポイント用のカラムを作成します。カラム内の各行に，仮の値を入力します。各仮の値は一意でなければなりません。

例えば，前の手順のデータ・テーブルが次のようになります。



4 ［**テーブル**］＞［**データのパラメータ化**］を選択して，［**パラメータの指定**］ダイアログ・ボックスを開きます。［**既存のパラメータ**］ボックスで，各チェックポイントと同期化ポイントの期待値を変更し，データ・テーブルからその値を使えるようにします。詳細については，372 ページ「テスト・スクリプト内の値のパラメータ化」を参照してください。テスト・スクリプトは手作業で変更することもできます。

例えば、サンプルのテキストは次のようになっているはずです。

```
set_window (" 無題 - メモ帳 ", 12);
menu_select_item (" 検索 (S); 検索 (F)...");
table = "default.xls";
rc = ddt_open(table, DDT_MODE_READ);
if (rc!= E_OK && rc != E_FILE_OPEN)
    pause("Cannot open table.");
ddt_get_row_count(table,RowCount);
for (i = 1; i <= RowCount; i++) {
    ddt_set_row(table,i);
    set_window (" 検索 ", 5);
    edit_set (" 検索する文字列 (N):", ddt_val(table, "Str"));
```

```
    button_press (" 次を検索 (N)");
    set_window(" メモ帳 ", 10);

    # GUI チェックポイント・ステートメントがパラメータ化されている。
    obj_check_gui("message", "list1.ckl",
                  ddt_val(table, "GUI_Check1"), 1);

    # ビットマップ・チェックポイント・ステートメントがパラメータ化されている。
    win_check_bitmap(" メモ帳 ",
                  ddt_val(table, "BMP_Check1"), 5, 30, 23, 126, 45);

    # 同期化ポイント・ステートメントがパラメータ化されている。
    obj_wait_bitmap("message",
              ddt_val(table, "Sync1"), 13);
    set_window (" メモ帳 ", 5);
    button_press ("OK");
}
ddt_close(table);
set_window (" 検索 ", 4);
button_press (" キャンセル ");
```

5 実行モード・ボックスで［**更新**］を選択し，更新モードでテストを実行します。［**実行**］コマンドを選択して，テストを実行します。

更新モードでテストを実行すると，WinRunner はデータ・テーブルから期待値の名前を読み取ります。WinRunner はデータ・テーブルの GUI チェックポイント，ビットマップ・チェックポイント，ビットマップ同期化ポイントを見つけられないため，これらの値をアプリケーションから再度キャプチャし，その値をテストの *exp* フォルダに期待結果として保存します。GUI チェックポイントの期待値は期待結果として保存されます。ビットマップ・チェックポイントとビットマップ同期化ポイントの期待値はビットマップとして保存されます。

更新モードでテストを実行したら，データ・テーブル内のこれらの一連のデータの期待値は，再度キャプチャされ，保存されます。

その後，検証モードでテストを実行し，アプリケーションの振る舞いを検査できます。

注 : ［更新］モードでテストを実行すると，WinRunner は GUI チェックポイントとビットマップ・チェックポイントの期待値を自動的にキャプチャし直します。WinRunner はビットマップ同期化ポイントの期待値のキャプチャし直す前にそれを通知します。

データ駆動型テストでの TSL 関数の使用

WinRunner にはデータ駆動型テストで利用できる TSL 関数がいくつか用意されています。

関数ジェネレータを使って，以下の関数をテスト・スクリプトに挿入することができます。あるいは，これらの関数を使うステートメントを手作業でプログラミングすることもできます。関数ジェネレータの使い方については，第 34 章「関数の生成」を参照してください。TSL 関数の詳細については，「**TSL リファレンス**」を参照してください。

注 : 他の **ddt_** 関数を使用する前に，**ddt_open** 関数を使ってデータ・テーブルが開いていなければなりません。データ・テーブルを保存するには **ddt_save** 関数を使い，データ・テーブルを終了するには，**ddt_close** 関数を使わなくてはなりません。

データ・テーブルのオープン

ddt_open 関数は，指定されたデータ・テーブルを作成するか，開きます。データ・テーブルは，Microsoft Excel ファイル，あるいはタブで区切られたテキスト・ファイルです。Excel ファイルまたはタブで区切られたテキスト・ファイルの最初の行には，パラメータの名前が含まれます。この関数の構文は次のとおりです。

ddt_open (*data_table_name* [, *mode*]);
data_table_name は，データ・テーブルの名前です。**mode** はデータ・テーブルを開いているモードで，DDT_MODE_READ（読み取り専用）または DDT_MODE_READWRITE（読み取りまたは書き込み）のどちらかです。

データ・テーブルの保存

ddt_save 関数は，指定されたデータ・テーブルの情報を保存します。この関数の構文は次のとおりです。

ddt_save (*data_table_name*);

data_table_name は，データ・テーブルの名前です。

ddt_save 関数はデータ・テーブルを閉じません。データ・テーブルを閉じるには，次に説明するように **ddt_close** 関数を使用します。

データ・テーブルのクローズ

ddt_close 関数は，指定されたデータ・テーブルを閉じます。この関数の構文は次のとおりです。

ddt_close (*data_table_name*);

data_table_name は，データ・テーブル・ファイルの名前です。

ddt_close 関数はデータ・テーブルへの変更を保存しません。データ・テーブルを閉じる前に変更を保存するには，**ddt_save** 関数を使います。

データ・テーブルのエクスポート

ddt_export 関数は，1 つのテーブル・ファイルの情報を別のテーブル・ファイルにエクスポートします。この関数の構文は次のとおりです。

ddt_export (*data_table_filename1*, *data_table_filename2*);

data_table_filename1 は，エクスポート元のデータ・テーブル・ファイルの名前です。***data_table_filename2*** は，エクスポート先のデータ・テーブル・ファイルの名前です。

データ・テーブル・エディタの表示

ddt_show 関数は，指定されたデータ・テーブルのエディタを表示または非表示にします。この関数の構文は次のとおりです。

ddt_show (*data_table_name* [, *show_flag*]);

data_table_name は，テーブルの名前です。***show_flag*** は，エディタが表示する（標準 =1）または，非表示 （0）を示す値です。

データ・テーブルの行数の取得

ddt_get_row_count 関数は，指定されたデータ・テーブルの行数を取得します。この関数の構文は次のとおりです。

ddt_get_row_count (*data_table_name*, *out_rows_count*);

data_table_name は，データ・テーブルの名前です。***out_rows_count*** は，データ・テーブル内の行の総数を格納する出力変数です。

データ・テーブル内のアクティブな行の次の行への変更

ddt_next_row 関数は，指定されたデータ・テーブル内の次の行をアクティブな行にします。この関数の構文は次のとおりです。

ddt_next_row (*data_table_name*);

data_table_name は，データ・テーブルの名前です。

データ・テーブル内のアクティブ行の設定

ddt_set_row 関数は，指定されたデータ・テーブル内のアクティブな行を設定します。この関数の構文は次のとおりです。

ddt_set_row (*data_table_name*, *row*);

data_table_name は，データ・テーブルの名前です。***row*** は，データ・テーブル内の新しいアクティブ行です。

テーブルの現在行の値の設定

ddt_set_val 関数は，テーブルの現在の行の値を書き込みます。この関数の構文は次のとおりです。

ddt_set_val (*data_table_name*, *parameter*, *value*);

data_table_name は，データ・テーブルの名前です。***parameter*** は，値が挿入されるカラムの名前です。***value*** は，テーブル内に書きこまれる値です。

注：データ・テーブルを DDT_MODE_READWRITE（読み取りまたは書き込みモード）で開いている場合は，この関数しか使用できません。

テーブルの新しい内容を保存するには，**ddt_set_val** ステートメントの後に **ddt_save** ステートメントを追加します。テストの最後で，**ddt_close** ステートメントを使ってテーブルを閉じます。

テーブル内の行の値の設定

ddt_set_val_by_row 関数は，テーブル内の指定された行の値を設定します。この関数の構文は次のとおりです。

ddt_set_val_by_row (*data_table_name*, *row*, *parameter*, *value*);

data_table_name は，データ・テーブルの名前です。***row*** は，テーブルの行番号です。これは，既存の行番号または現在の行番号に 1 を加えた数です。***parameter*** は，値が挿入されるカラムの名前です。***value*** は，テーブルに書き込まれる値です。

注：データ・テーブルを DDT_MODE_READWRITE（読み取りまたは書き込みモード）で開いている場合は，この関数しか使用できません。

テーブルの新しい内容を保存するには，**ddt_set_val** ステートメントの後に **ddt_save** ステートメントを追加します。テストの最後で，**ddt_close** ステートメントを使ってテーブルを閉じます。

データ・テーブルのアクティブな行の取得

ddt_get_current_row 関数は，指定されたデータ・テーブルのアクティブな行を取得します。この関数の構文は次のとおりです。

ddt_get_current_row (*data_table_name*, *out_row*);

data_table_name は，データ・テーブルの名前です。***out_row*** は，データ・テーブルで指定された行を格納する出力変数です。

データ・テーブル内のパラメータが有効かどうかの判定

ddt_is_parameter 関数は，指定されたデータ・テーブルが有効かどうかを判定します。この関数の構文は次のとおりです。

ddt_is_parameter (*data_table_name*, *parameter*);

data_table_name はデータ・テーブルの名前です。***parameter*** は，データ・テーブル内のパラメータの名前です。

データ・テーブル内のパラメータのリストを返す

ddt_get_parameters 関数は，指定されたデータ・テーブル内のすべてのパラメータのリストを返します。この関数の構文は次のとおりです。

ddt_get_parameters (*data_table_name*, *params_list, params_num* **);**

data_table_name は，データ・テーブルの名前です。**params_list** は，タブで区切られた，データ・テーブル内のすべてのパラメータのリストを返す出力パラメータです。**params_name** は，**params_list** のパラメータ数を返す出力パラメータです。

データ・テーブル内のアクティブな行のパラメータの値を返す

ddt_val 関数は，指定されたデータ・テーブルのアクティブな行のパラメータの値を返します。この関数の構文は次のとおりです。

ddt_val (*data_table_name*, *parameter*);

data_table_name は，データ・テーブルの名前です。***parameter*** は，データ・テーブル内のパラメータの名前です。

データ・テーブル内の行のパラメータの値を返す

ddt_val_by_row 関数は，指定されたデータ・テーブルの指定された行のパラメータの値を返します。この関数の構文は次のとおりです。

ddt_val_by_row (*data_table_name*, *row_number*, *parameter*);

data_table_name は，データ・テーブルの名前です。***parameter*** は，データ・テーブル内のパラメータの名前です。***row_number*** はデータ・テーブル内の行数です。

データ・テーブルのアクティブ行のテスト結果への報告

ddt_report_row 関数は，指定されたデータ・テーブルのアクティブ行をテスト結果へ報告します。この関数の構文は次のとおりです。

ddt_report_row (*data_table_name*);

data_table_name は，データ・テーブルの名前です。

データベースからのデータのデータ・テーブルへのインポート

ddt_update_from_db 関数は，データ・テーブルにデータベースからのデータをインポートします。この関数は，［データ駆動テストウィザード］の［データベースからデータをインポートする］オプションを選択すると，テスト・スクリプトに挿入されます。テストを実行すると，この関数はデータ・テーブルをデータベースからのデータで更新します。この関数の構文は次のとおりです。

ddt_update_from_db (*data_table_name*, *file,out_row_count*
[, *max_rows*]);

data_table_name は，データ・テーブルの名前です。

file は，Microsoft Query でユーザが定義したクエリが含まれた ****.sql*** ファイル，あるいは Data Junction で定義された変換が含まれた ****.djs*** ファイルです。***out_row_count*** は，データ・テーブルから取り出した行数が含まれた出力パラメータです。***max_rows*** は，データベースから取り出す最大行数が指定されている入力パラメータです。最大行数を指定しない場合，標準設定では，行数の制限はありません。

注：DDT_MODE_READWRITE（読み取りまたは書き込み）モードでデータ・テーブルを開くには，**ddt_open** 関数を使わなければなりません。**ddt_update_from_db** 関数を使うと，テーブル内の新しい内容は自動的には保存されません。テーブル内の新しい内容を保存するには，**ddt_close** 関数の前に **ddt_save** 関数を使います。

データ駆動型テスト作成のガイドライン

データ駆動型テストを作成するときには，以下のガイドラインを考慮に入れて行ってください。

- データ駆動型テストには複数のパラメータ化されたループを含めることができます。
- **default.xls データ・テーブル**以外のデータ・テーブルを開いて保存できます。これにより，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。データ・テーブルの［**新規作成**］，［**開く**］，［**上書き保存**］，［**名前を付けて保存**］コマンドを使って，データ・テーブルを開いて保存できます。詳細については，376 ページ「データ・テーブルの編集」を参照してください。

注：あるテストでデータ・テーブルが開かれている間に別のテストでデータ・テーブルを開くと，片方のテストでデータ・テーブルに加えた変更は，もう一方のテストでは反映されません。データ・テーブルで変更を保存するには，別のテストでデータ・テーブルを開く前に，テストのデータ・テーブルを保存して，閉じなければなりません。

- データ駆動型テストを実行する前に，データ駆動型テスト内に矛盾する可能性のある要素がないか調べる必要があります。データ駆動テスト・ウィザードとデータのパラメータ化ダイアログ・ボックスを使って，選択したチェックポイントと記録されたステートメント内のすべての固定値を見つけることができますが，外からの入力に基づいて変化するオブジェクトのラベルなどは検査しません。こうした矛盾のほとんどは，以下のいずれかの方法で解決することができます。
  - 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。
  - ［GUI マップの構成設定］ダイアログ・ボックスを使って，WinRunner が問題のあるオブジェクトを認識するのに使用する物理的記述を変更します。
- TSL ステートメントを使って，テスト実行中にアクティブな行を変えることができます。詳細については，396 ページ「データ駆動型テストでの TSL 関数の使用」を参照してください。

- TSL ステートメントを使って，テスト実行中にアクティブでない行の内容を読み取ることができます。詳細については，396 ページ「データ駆動型テストでの TSL 関数の使用」を参照してください。
- テストでパラメータ化したループ内に **tl_step** または他のレポート・ステートメントを追加して，各繰り返しで使用されたデータの結果を見ることができます。
- データ駆動型テストの実行中にデータ・テーブル内のすべてのデータを使う必要はありません。
- テスト・スクリプトのごく一部だけ，あるいはその中の 1 つのループだけをパラメータ化することもできます。
- テストの実行中にWinRunnerがパラメータ化されたGUIオブジェクトを見つけられない場合には，パラメータ化された引数がテスト・スクリプト内で，引用符で囲まれていないことを確認します。
- GUI チェックポイント，ビットマップ・チェックポイント，ビットマップ同期化ポイントなどを含むステートメントをパラメータ化できます。詳細については，391 ページ「データ駆動型チェックポイントとビットマップ同期化ポイントの使用」を参照してください。
- 定数は，他の文字列や値と同じようにパラメータ化できます。
- データ・テーブルの使い方は，Excel のスプレッドシートと同じです。セルに数式を挿入することもできます。
- 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はありません。
- **ddt_set_val** と **ddt_set_val_by_row** 関数を使って，テストの実行中にデータ・テーブルにデータを挿入できます。そして，**ddt_save** 関数を使って，データ・テーブルに変更を保存します。

注：標準設定では，データ・テーブルはテスト・フォルダに格納されます。

第 18 章

テスト実行の同期化

テスト実行時に，アプリケーションのパフォーマンスが一定でない場合，同期化を行うことで問題を解決できます。テスト・スクリプトに同期化ポイントを挿入して，WinRunner にテスト実行を一時的に停止させ，合図を待ってからテストを再開するようにできます。

本章では，以下の項目について説明します。

- テスト実行の同期化について
- オブジェクトとウィンドウの待機
- オブジェクトとウィンドウのプロパティ値の待機
- オブジェクトやウィンドウのビットマップの待機
- スクリーン領域のビットマップの待機
- テストの同期化のためのヒント

テスト実行の同期化について

アプリケーションは，テスト実行ごとに，ユーザの入力に対して必ずしも同じ速度で応答するわけではありません。これは，特にネットワーク経由でアプリケーションをテストする場合には，一般的です。同期化ポイントをテスト・スクリプトに挿入すると，WinRunner はテスト実行を一時的に停止し，テスト対象アプリケーションの準備が整うまで待ってからテストを再開します。

同期化ポイントには，オブジェクト / ウィンドウ同期化ポイント，プロパティ値同期化ポイント，およびビットマップ同期化ポイントの 3 種類があります。

- オブジェクトまたはウィンドウが現れるまで WinRunner に待機させたい場合は，オブジェクト / ウィンドウ同期化ポイントを作成します。

- オブジェクトまたはウィンドウが特定のプロパティを持つまで WinRunner に待機させたい場合は，プロパティ値同期化ポイントを作成します。
- 視角的なきっかけが表示されるまで WinRunner を待機させたい場合は，ビットマップ同期化ポイントを作成します。ビットマップ同期化ポイントでは，WinRunner は，オブジェクト，ウィンドウ，あるいはスクリーン領域が現れるまで待機します。

例えば，描画アプリケーションで，別のアプリケーションからビットマップをインポートして，それを回転する処理をテストするとします。人間のユーザなら，ビットマップが完全に再描画されてからそれを回転しようとするでしょう。しかし，WinRunner の場合はテスト・スクリプト内でインポート・コマンドの後と回転コマンドの前に同期化ポイントが必要になります。同期化ポイントにより，WinRunner はテスト実行のたびにインポート・コマンドが完了するまで待ってからビットマップの回転を行うようになります。

別の例として，アプリケーションをテストしている間に，ボタンが使用可能であるかどうかを検査したいとします。おそらくボタンは，アプリケーションがネットワーク上での操作を完了した後に使用可能になるでしょう。アプリケーションがネットワーク経由の操作を完了するのにかかる時間は，ネットワーク上の負荷によって異なります。人間であれば，そのボタンをクリックする前に処理が完了してボタンが使用可能になるまで待つでしょう。しかし，WinRunner の場合，ネットワークの操作を開始した後と，ボタンをクリックする前に同期化ポイントが必要です。その同期化ポイントにより，WinRunner はテストが実行されるたびに，ボタンをクリックする前にボタンが使用可能になるのを待つようになります。

テストを同期化して，アプリケーションのウィンドウのビットマップや GUI オブジェクト，あるいは画面の任意の矩形領域を待機できます。またテストを同期化して，GUI オブジェクトのプロパティ値が「enabled」になるまで待機できます。同期化ポイントを作成するには，［**挿入**］＞［**同期化ポイント**］コマンドを選択して，テスト対象アプリケーションの領域かオブジェクトを示します。選択する［同期化ポイント］コマンドによって WinRunner は，スクリーン領域の GUI オブジェクトのプロパティ値か GUI オブジェクトのビットマップ，どちらかをキャプチャして，それを［期待結果］フォルダ（*exp*）に格納します。［期待結果］フォルダに保存する前であれば，キャプチャした GUI オブジェクトのプロパティ値を変更できます。

ビットマップ同期化ポイントはビットマップをキャプチャする同期化ポイントです。テスト・スクリプトの中では，**win_wait_bitmap** または **obj_wait_bitmap** ステートメントとして示されます。プロパティ値の同期化ポイントはプロパティ値をキャプチャする同期化ポイントです。テスト・スクリプトの中では，**button_wait_info** や **list_wait_info** のように，**_wait_info** ステートメントとして示されます。テストを実行すると，WinRunner はテスト実行を一時停止し，期待結果ビットマップ，またはプロパティ値が現れるまで待機します。その後，現在の「**実際の**」ビットマップまたはプロパティ値と，先に保存してある「**期待結果**」のビットマップまたはプロパティ値を比較します。ビットマップまたはプロパティ値が現れると，テスト実行が再開されます。

注：wait 関数と **wait_info** 関数はミリ秒で実装されるため，テストの実行方法に影響はありません。

オブジェクトとウィンドウの待機

同期化ポイントを作成して，指定したオブジェクトまたはウィンドウが現れるのを WinRunner に待機させることができます。例えば，WinRunner に，ウィンドウ内で操作を実行する前にそのウィンドウが開くのを待機するよう指示したり，操作したいオブジェクトが表示されるまで待機させたりできます。

WinRunner は，テスト・スクリプトの次のステートメントを実行するまで，標準のタイムアウトしか待機しません。この標準のタイムアウトは，テスト・スクリプトで **setvar** 関数に **timeout_msec テスト・オプションを使って**設定できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。また，標準のタイムアウトは，［一般オプション］ダイアログ・ボックスの［**実行**］＞［**設定**］カテゴリで，［**チェックポイントと CS ステートメントのタイムアウト**］ボックスを使ってグローバルに設定することもできます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

オブジェクトの同期化ポイントを作成するには **obj_exists** 関数を使い，ウィンドウの同期化ポイントを作成するには **win_exists** 関数を使います。これらの関数の構文は次のとおりです。

obj_exists (object [, time]);

win_exists (window [, time]);

object には，オブジェクトの論理名が入ります。オブジェクトはどのクラスに属していても構いません。**window** には，ウィンドウの論理名が入ります。**time** には，標準のタイムアウトに追加する時間（単位：ミリ秒）を指定し，次のステートメントが実行されるまでに WinRunner が待機する最大待ち時間を新しく決定します。

この関数をテスト・スクリプトに挿入する方法は，2 通りあります。関数ジェネレータを使って挿入する方法と，手作業で挿入する方法です。関数ジェネレータの使用法については，第 34 章「関数の生成」を参照してください。また，関数の詳細と使用例については，「**TSL リファレンス**」を参照してください。

オブジェクトとウィンドウのプロパティ値の待機

プロパティ値の同期化ポイントを作成できます。これにより，WinRunner は指定したプロパティ値が GUI オブジェクトに現れるのを待機します。例えば，ボタンが使用可能になるまで，またはリストから項目が選択されるまで WinRunner に待機するように設定できます。

テストを同期化する方法は，オブジェクトとウィンドウのどちらのプロパティ値も同じです。まず，［**挿入**］＞［**同期化ポイント**］＞［**オブジェクト / ウィンドウ プロパティ**］を選択します。アプリケーションの上でマウス・ポインタを移動すると，オブジェクトやウィンドウが点滅します。ウィンドウを指定するには，対象となるウィンドウのタイトル・バーまたはメニュー・バーをクリックします。オブジェクトを選択するには，オブジェクトをクリックします。

選択したウィンドウまたはオブジェクトの名前を含んでいるダイアログ・ボックスが開きます。ここで，ウィンドウのプロパティ，検査するオブジェクト，プロパティの期待値，WinRunner が同期化ポイントを待機する時間などを指定できます。

選択した GUI オブジェクトによって，次の関数のいずれかのステートメントがテスト・スクリプトに追加されます。

GUI オブジェクト	TSL 関数
ボタン	**button_wait_info**
編集フィールド	**edit_wait_info**
リスト	**list_wait_info**
メニュー	**menu_wait_info**
汎用の「object」クラスにマップされているオブジェクト	**obj_wait_info**
スクロール・バー	**scroll_wait_info**
スピン・ボックス	**spin_wait_info**
静的テキスト	**static_wait_info**
ステータス・バー	**statusbar_wait_info**
タブ	**tab_wait_info**
ウィンドウ	**win_wait_info**

テスト実行中，WinRunner は GUI オブジェクトで指定されたプロパティ値が検出されるまでテスト実行を一時的に停止します。その後，指定されたプロパティの現在の値を，その期待値と比較します。プロパティ値が一致すれば，WinRunner はテストを再開します。

指定された GUI オブジェクトのプロパティ値が表示されず，***mismatch_break*** テスト・オプションが ON になっていると，WinRunner はエラー・メッセージを表示します。**mismatch_break** テスト・オプションの詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。［一般オプション］ダイアログ・ボックスの［**実行**］＞［**設定**］カテゴリの対応する［**検証が失敗したら停止する**］オプションを使ってグローバルにテスト・オプションを設定することもできます。このテスト・オプションをグローバルに設定する方法の詳細については，第 22 章「グローバル・テスト・オプションの設定」を参照してください。

キャプチャするウィンドウまたはオブジェクトの名前が，テスト実行ごとに変わるような場合は，GUI マップにおける物理的記述に正規表現を定義できます。そうした場合，WinRunner は名前の全部または一部を無視するようになります。詳細については，第 7 章「GUI マップの編集」および第 27 章「正規表現の使い方」を参照してください。

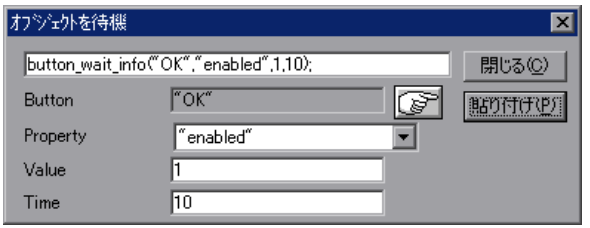
記録の際，アクティブ・ウィンドウ以外のウィンドウのオブジェクトをキャプチャすると，WinRunner は **set_window** ステートメントを自動的に生成します。

プロパティ値の同期化ポイントを挿入するには，次の手順を実行します。

1 ［**挿入**］＞［**同期化ポイント**］＞［**オブジェクト / ウィンドウ プロパティ**］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 プロパティ同期化ポイント**］ボタンをクリックします。マウス・ポインタが指差し型に変わります。

2 対象となるオブジェクトまたはウィンドウを強調表示します。オブジェクトを強調表示するには，マウス・ポインタをそのオブジェクトの上に移動します。ウィンドウを強調表示するには，マウス・ポインタをタイトル・バーまたはメニュー・バーの上に移動します。

3 マウスの左ボタンをクリックします。オブジェクトとウィンドウのどちらをクリックしたかによって，［**オブジェクトを待機**］または［**ウィンドウを待機**］のいずれかのダイアログ・ボックスが開きます。



4 ウィンドウやオブジェクトで実行するプロパティ検査のパラメータを指定します。次の中から指定します。

- **ウィンドウまたは<オブジェクトの種類>**：クリックしたウィンドウまたはオブジェクトの名前が読み取り専用ボックスに現れます。別のウィンドウやオブジェクトを選択するには，指差しポインタでクリックします。
- ［**Property**］：リストから検査対象のオブジェクトまたはウィンドウのプロパティを選択します。このボックスには標準で，上で指定したウィンドウまたはオブジェクトの種類の標準プロパティが表示されます。
- ［**Value**］：検査対象のオブジェクトまたはウィンドウのプロパティの期待値を入力します。このボックスには標準で，このプロパティの現在の値が表示されます。
- ［**Time**］：WinRunner が同期化ポイントで待機する時間（単位：秒）を入力します。**timeout_msec** テスト・オプションで指定した待機時間に加算されます。**timeout_msec** テスト・オプションで指定した WinRunner が待機する標準の時間を変更できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。［一般オプション］ダイアログ・ボックスの［**実行**］＞［**設定**］カテゴリの［**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，標準のタイムアウトの値を変更することもできます。詳細については，第 22 章「グローバル・テスト・オプションの設定」を参照してください。

注：上のパラメータで行ったいずれの変更も，直ちにダイアログ・ボックス上のテキスト・ボックスに表示されます。

---

5 ［**貼り付け**］をクリックして，ステートメントをテスト・スクリプトに貼り付けます。

ダイアログ・ボックスが閉じて，オブジェクトのプロパティ値を検査する **_wait_info** ステートメントがテスト・スクリプトに挿入されます。例えば，**button_wait_info** の構文は次のとおりです。

button_wait_info (*button*, *property*, *value*, *time* **);**

button は，ボタンの名前です。*property* は，button オブジェクト・クラスで使われている任意のプロパティです。*value* は，テスト実行を再開する前に表示される値です。

time は，WinRunner が待機しなければならない秒数で，**timeout_msec** テスト・オプションに加算されています。**_wait_info** ステートメントの詳細については，「**TSL リファレンス**」を参照してください。

例えば，フライト予約アプリケーションをテストする際に，乗客とフライト情報を入力して，［注文挿入］をクリックして，航空券を注文したとします。そして，アプリケーションによる注文の処理が数秒を要したとします。操作が完了したら［注文削除］をクリックして注文を削除します。

テストを滞りなく実行するには，テスト・スクリプトに **button_wait_info** ステートメントを挿入する必要があります。この関数を使って，WinRunner に，［注文削除］ボタンが使用可能なるまで 10 秒間（にタイムアウト時間を加算した時間）だけ待機するようにします。これにより，アプリケーションがまだ注文を処理している最中に削除が行われないことが保証されます。以下のような処理をテスト・スクリプトに指定します。

```
button_press (" 注文挿入 ");
button_wait_info (" 注文削除 ","enabled",1,"10");
button_press (" 注文削除 ");
```

注 : ［関数ジェネレータ］を使って，ウィンドウやオブジェクトのプロパティ値を待機する同期化ポイントを作成することもできます。［関数ジェネレータ］の使い方については，第 34 章「関数の生成」を参照してください。これらの関数の使い方の詳細については，「**TSL リファレンス**」を参照してください。

オブジェクトやウィンドウのビットマップの待機

オブジェクトやウィンドウのビットマップが，テスト対象アプリケーションに表示されるまで待機する，ビットマップ同期化ポイントを作成できます。

テストを同期化する方法は，オブジェクトとウィンドウのどちらのビットマップの場合も同じです。まず，［**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**］を選択します。マウス・ポインタをアプリケーションの上で移動すると，オブジェクトやウィンドウが点滅します。すべてのウィンドウのビットマップを指定するには，そのウィンドウのタイトル・バーまたはメニュー・バーをクリックします。オブジェクトのビットマップを選択するには，そのオブジェクトをクリックします。

テスト実行の際，WinRunner は指定されたビットマップが再描画されるまでテスト実行を一時的に停止します。その後，現在のビットマップを，先にキャプチャしてある期待結果ビットマップと比較します。ビットマップが一致すれば，WinRunner はテストを再開します。

不一致の場合 WinRunner は，*mismatch_break* テスト・オプションが ON になっていればエラー・メッセージを表示します。*mismatch_break* テスト・オプションの詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

［一般オプション］ダイアログ・ボックスの［**実行**］＞［**設定**］カテゴリの対応する［**検証が失敗したら停止する**］オプションを使ってグローバルにテスト・オプションを設定することもできます。このテスト・オプションのグローバルな設定の詳細に関しては，第 22 章「グローバル・テスト・オプションの設定」を参照してください。

記録の際，アクティブ・ウィンドウ以外のウィンドウのオブジェクトをキャプチャすると，WinRunner は，**set_window** ステートメントを自動的に生成します。

オブジェクトやウィンドウに対するビットマップ同期化ポイントを挿入するには，次の手順を実行します。

1 ［**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の［**オブジェクト / ウィンドウのビットマップ同期化ポイント**］をクリックします。アナログ・モードで記録を行う場合は，［ウィンドウビットマップの同期化］ソフトキーを押します。マウス・ポインタが指差し型に変わります。

2 対象となるウィンドウまたはオブジェクトを強調表示します。オブジェクトを強調表示するには，マウス・ポインタをオブジェクトの上に移動します。ウィンドウを強調表示するには，マウス・ポインタをタイトル・バーかメニュー・バーの上に移動します。

3 マウスの左ボタンをクリックして，操作を完了します。WinRunner はビットマップをキャプチャし，次の構文でテスト・スクリプトに **obj_wait_bitmap** または **win_wait_bitmap** ステートメントを生成します。

obj_wait_bitmap (*object*, *image*, *time* **);**

win_wait_bitmap (*window*, *image*, *time* **);**

例えば，フライト予約アプリケーションを対象とするテストで，テスト・スクリプトに同期化ポイントを挿入するとします。［フライト予定日］を指定した場合，以下のようなステートメントが生成されます。

```
obj_wait_bitmap (" フライト予定日 : ", "Img5", 22);
```

obj_wait_bitmap と **win_wait_bitmap** 関数の詳細については，「**TSL リファレンス**」を参照してください。

注：obj_wait_bitmap および **win_wait_bitmap** の実行は，**delay_msec**，**timeout_msec**，および *min_diff* テスト・オプションに設定されている値によって変わります。テスト・オプションの詳細とその変更方法については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。これらのテスト・オプションを［一般オプション］ダイアログ・ボックスの［**実行**］＞［**同期化**］カテゴリおよび［**実行**］＞［**設定**］カテゴリの対応する［**ウィンドウの同期化のための遅延**］ボックス，［**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，［**ビットマップ間の違いを差異として認識するしきい値**］ボックスを使ってグローバルに設定できます。グローバル・テスト・オプションの設定の詳細については，第 22 章「グローバル・テスト・オプションの設定」を参照してください。

スクリーン領域のビットマップの待機

アプリケーションの特定の領域のビットマップを待機する同期化ポイントを作成できます。画面で任意の矩形領域を定義でき，またその領域を同期化ポイントのビットマップとしてキャプチャできます。

まず，［**挿入**］＞［**同期化ポイント**］＞［**画面領域のビットマップ**］を選択します。マウス・ポインタをアプリケーションの上で移動すると，ポインタが十字型に変わり，画面の左上角にヘルプ・ウィンドウが表示されます。

十字のカーソルを使って，対象領域を囲む矩形を定義します。領域の大きさは任意です。また，単独のウィンドウの領域，あるいは複数のウィンドウと交わる領域を指定できます。WinRunner は，左上角および右下角の座標によって矩形を定義します。この座標は，その領域があるウィンドウまたはオブジェクトの左上角からの相対座標です。領域が，複数のウィンドウにまたがる場合，またはタイトルを持たないウィンドウ内である場合（例えば，ポップアップ・ウィンドウなど），座標は画面全体（ルート・ウィンドウ）からの相対座標です。

テスト実行の際，WinRunner は指定されたビットマップが表示されるまでテスト実行を一時的に停止します。その後，現在のビットマップを，期待結果ビットマップと比較します。ビットマップが一致すれば，WinRunner はテストを再開します。

不一致の場合 WinRunner は，*mismatch_break* テスト・オプションが ON になっているとエラー・メッセージを表示します。*mismatch_break* テスト・オプションの詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。これらのテスト・オプションを［一般オプション］ダイアログ・ボックスの［**実行**］＞［**設定**］カテゴリの対応する［**検証が失敗したら停止する**］ボックスを使って設定できます。テスト・オプションのグローバルな設定の詳細については，第 22 章「グローバル・テスト・オプションの設定」を参照してください。

スクリーン領域に対するビットマップ同期化ポイントを定義するには，次の手順を実行します。

1 ［**挿入**］＞［**同期化ポイント**］＞［**画面領域のビットマップ**］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同期化ポイント**］ボタンをクリックします。アナログ・モードで記録を行う場合は，［スクリーン領域の同期化］ソフトキーを押します。

WinRunner のウィンドウをアイコン化するとポインタが十字型に変わり，画面の左上角にヘルプ・ウィンドウが表示されます。

2 キャプチャ対象の領域を指定します。マウスの左ボタンをクリックし，矩形が対象領域全体を囲むまでマウスをドラッグしてから，マウス・ボタンを放します。

3 マウスの右ボタンをクリックして，操作を完了します。WinRunner はビットマップをキャプチャし，次の構文でテスト・スクリプトに **win_wait_bitmap** または **obj_wait_bitmap** ステートメントを生成します。

win_wait_bitmap (*window*, *image*, *time*, *x*, *y*, *width*, *height* **);**

obj_wait_bitmap (*object*, *image*, *time*, *x*, *y*, *width*, *height* **);**

例えば，フライト予約アプリケーションで注文を更新しているとします。注文の更新を確認するメッセージの表示とテストの継続を同期化しなければなりません。ステータス・バーに「注文を更新しました」というメッセージが表示されるまで待機する同期化ポイントを挿入します。

WinRunner は，以下のようなステートメントを生成します。

```
obj_wait_bitmap (" 注文を更新しました ...", "Img7", 10);
```

win_wait_bitmap と **obj_wait_bitmap** 関数の詳細については，「**TSL リファレンス**」を参照してください。

注：win_wait_bitmap ステートメントおよび **obj_wait_bitmap** ステートメントの動作は，**delay_msec**，**timeout_msec**，および *min_diff* テスト・オプションに設定されている値によって変わります。テスト・オプションの詳細とその変更方法については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。これらのテスト・オプションを［一般オプション］ダイアログ・ボックスの［**実行**］＞［**設定**］カテゴリおよび［**実行**］＞［**同期化**］カテゴリの対応する［**ウィンドウの同期化のための遅延**］ボックス，［**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，［**ビットマップ間の違いを差異として認識するしきい値**］を使ってグローバルに設定できます。テスト・オプションのグローバルな設定の詳細については，第 22 章「グローバル・テスト・オプションの設定」を参照してください。

テストの同期化のためのヒント

- **テストの中止と一時停止：**一時停止ソフトキーやストップ・ソフトキーを使って，同期化のステートメントを待機しているテストを中止したり、一時停止したりします。ソフトキーの使い方については，110 ページ「ソフトキーを使用したテスト作成コマンドのアクティブ化」を参照してください。
- **アナログ・モードでの記録：**アナログ・モードでテストを記録するときは，「オブジェクト / ウィンドウ ビットマップの同期化」ソフトキーまたは「スクリーン領域の同期化」ソフトキーを押して，ビットマップの同期化ポイントを作成します。これにより不要なマウス操作を記録しないようにできます。テストをプログラミングする場合，アナログの TSL 関数，**wait_window** を使って，ビットマップを待機できます。詳細については，「**TSL リファレンス**」を参照してください。
- **データ駆動型テスト：**データ駆動型テストでビットマップの同期化ポイントを使用するには，同期化ポイントを含むテスト・スクリプトのステートメントをパラメータ化しなければなりません。データ駆動型テストでのビットマップ同期化ポイントの使い方については，391 ページ「データ駆動型チェックポイントとビットマップ同期化ポイントの使用」を参照してください。

第 4 部

テストの実行 ー 基本

第 19 章

テスト実行について

テスト・スクリプトを開発したら，テストを実行して，アプリケーションの動作を検査します。

本章では，以下の項目について説明します。

- テスト実行について
- WinRunner のテスト実行モード
- WinRunner の実行コマンド
- ソフトキーを使った実行コマンドの選択
- アプリケーションを検査するためのテスト実行
- テスト・スクリプトをデバッグするためのテスト実行
- 期待結果を更新するためのテスト実行
- テスト実行時の入力パラメータの値の指定
- テスト・オプションによるテスト実行の制御
- テスト実行に関する一般的な問題の解決法

テスト実行について

テストを実行すると、WinRunner はテスト・スクリプトを 1 行ずつ解釈します。テスト・スクリプトの左側のマージンに表示される実行矢印は，現在どの TSL ステートメントが解釈されているのかを示します。テストを実行すると，WinRunner はあたかも人が操作しているようにアプリケーションを操作します。

テストは以下の 3 つのモードで実行できます。

- 検証実行モード。アプリケーションをチェックします。
- デバッグ実行モード。テスト・スクリプトのデバッグを行います。
- 更新実行モード。期待結果を更新します。

実行モードは，テスト・ツールバーにあるリストからモードを選択します。標準のテスト実行モードは，検証モードです。

注：検証実行モードは，テストにのみ使用でき，コンポーネントを使った作業では使用できません。コンポーネントを使用する際の標準のモードはデバッグ・モードです。

WinRunner の［**テスト**］および［**デバッグ**］メニュー・コマンドを使ってテストを実行します。テスト全体，あるいはその一部だけを実行することができます。コンテキスト・センシティブ・テストを実行する場合は，必要な GUI マップ・ファイルがあらかじめロードしておく必要があります。詳細については，第 5 章「グローバル GUI マップ・ファイル・モードでの作業」を参照してください。

テストを個別に実行したり，バッチ・テストを使って一連のテストをまとめて実行したりできます。バッチ・テストは，長時間にわたるテストを，夜間あるいは負荷の少ない時間に実行する場合に特に便利です。詳細については，第 36 章「バッチ・テストの実行」を参照してください。

WinRunner のテスト実行モード

WinRunner では，テストを実行するモードが 3 つあります。検証モード，デバッグ・モード，および更新モードです。テスト工程の各フェーズに応じて，それぞれのモードを使います。［一般オプション］ダイアログ・ボックスで標準の実行モードを設定できます。

検証

検証モードは，アプリケーションを検査する際に使います。WinRunner は，アプリケーションの「**現在の**」応答を，「**期待している**」応答と比較します。現在の応答と期待している応答の間に違いがあれば，それらはすべてキャプチャされ，「**検証結果**」として保存されます。標準設定では，テストの実行が終わると［テスト結果］ウィンドウが開き，検証結果が表示されます。詳細については，第 20 章「テスト結果の分析」を参照してください。

必要に応じて，任意の数の検証結果を保存できます。それには，テストを実行するたびに，結果を新しいディレクトリに保存するようにします。結果を保存するディレクトリの名前は，［テスト実行］ダイアログ・ボックスで指定します。［テスト実行］ダイアログ・ボックスは，検証モードでテストを実行するたびに開きます。検証モードでテスト・スクリプトを実行する方法の詳細については，430 ページ「アプリケーションを検査するためのテスト実行」を参照してください。

注：テストを［検証］モードで実行する前に，作成したチェックポイントの期待結果がなければなりません。テストの期待結果を更新する必要がある場合は，424 ページに示すように，テストを［更新］モードで実行してください。

デバッグ

デバッグ・モードは，テスト・スクリプトのバグを検出する際に使います。デバッグ・モードでのテストの実行は，検証モードでの実行と同じです。ただし，デバッグ結果は必ず「**デバッグ**」フォルダに保存されます。デバッグ結果は 1 組しか保存されないため，テストをデバッグ・モードで実行した場合，［テスト実行］ダイアログ・ボックスは自動的に開きません

テストをデバッグ・モードでテストの実行が終了しても，［テスト結果］ウィンドウは自動的に開きません。［テスト結果］ウィンドウを開いてデバッグの結果を表示するには，メイン・ツールバーにある［テスト結果］ボタンをクリックするか，［ツール］＞［テスト結果］コマンドを選択します。

テスト・スクリプトのデバッグの際には，WinRunner のデバッグ機能を使います。

- ［ステップ］コマンドを使ってテストの実行を制御できます。詳細については，第 38 章「テスト実行の制御」を参照してください。
- ブレーク・ポイントを設定して，テスト・スクリプト内の指定した場所でテストの実行中に一時停止します。詳細については，第 39 章「ブレークポイントの使用」を参照してください。
- ウォッチ・リストを使って，テストの実行中にテスト・スクリプト内の変数を監視します。詳細については，第 40 章「変数の監視」を参照してください。
- 呼び出しチェーンを使って，テスト・フローをナビゲートします。詳細については，を参照してください。第 29 章「テストの呼び出し」を参照してください。
- ［実行］ダイアログ・ボックスの［入力パラメータ］オプションを使って，テストを呼び出しチェーンに含める前にテストが様々なパラメータを処理する方法について検査します。詳細については，第 29 章「テストの呼び出し」を参照してください。

デバッグ・モードでテスト・スクリプトを実行する方法の詳細については，432 ページ「テスト・スクリプトをデバッグするためのテスト実行」を参照してください。

ヒント：テストをより速く実行させるには，テスト・スクリプトのデバッグ中にタイムアウト変数をゼロに変更します。タイムアウト変数を変更する方法の詳細については，第 22 章「グローバル・テスト・オプションの設定」および第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

更新

更新モードは，テストの「**期待**結果」を更新するとき，または新しい期待結果フォルダを作成するときに使います。例えば，プッシュ・ボタンの標準の状態が使用可能から使用不能になった場合に，プッシュ・ボタンを検査する GUI チェックポイントの期待結果を「**更新**」することができます。また，Windows XP でアプリケーションを実行するときの期待結果と，Windows NT でアプリケーションを実行するときの別の期待結果がある場合に，期待結果の追加セットを「**作成**」できます。追加の期待結果の生成の詳細については，433 ページ「複数の期待結果の生成」を参照してください。

標準設定では，WinRunner は期待結果を既存の期待結果に上書きして，*exp* フォルダに保存します。

テストの期待結果を更新する方法は 2 つあります。

➤ 実行コマンドを使ってテスト全体を実行し，既存の期待結果をすべてグローバルに上書きする方法。

➤ ［矢印から実行］コマンドまたは［ステップ］コマンドを使って，個々のチェックポイントおよび同期化ポイントの期待結果を更新する方法。

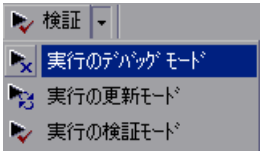
テスト・スクリプトを更新モードで実行する方法の詳細については，433 ページ「期待結果を更新するためのテスト実行」を参照してください。

テストの実行モードの設定 t

実行モード・ツールバー・ボタンを使って，テストの実行モードを設定します。

開いているテストの実行モードを設定するには，次の手順を実行します。

1 テスト・ツールバーの［**検証**］ツールバー・ボタンの横の矢印をクリックします。



2 テストに使用する実行モードを選択します。選択した実行モードに合わせてツールバー・ボタンのアイコンとテキストが変わります。

標準設定の実行モードの設定

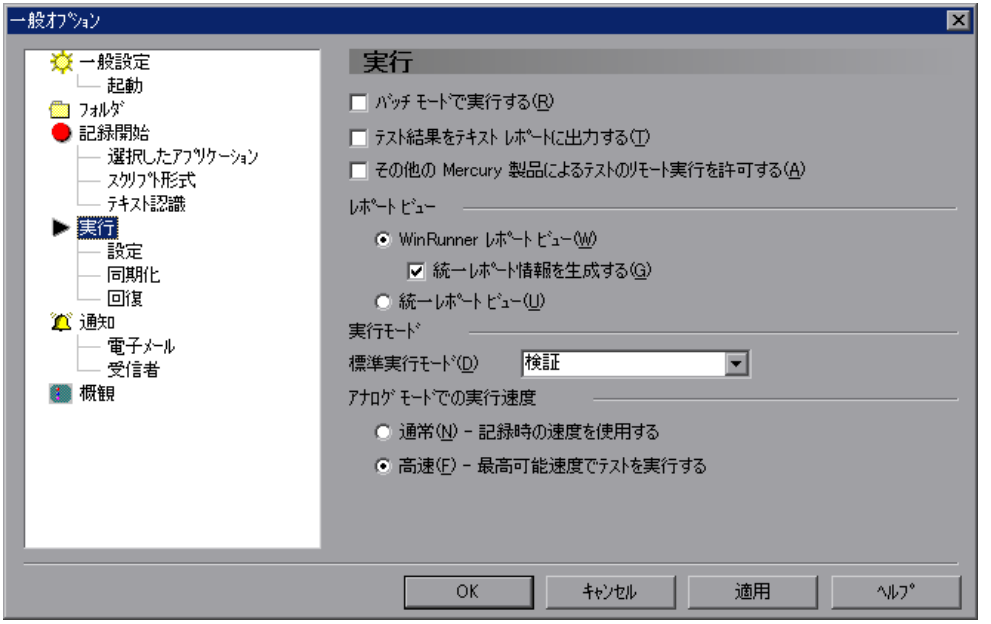
［一般オプション］ダイアログ・ボックスの［実行］カテゴリで標準設定の実行モードを設定できます。ここで設定されるモードによって，テストを開くときのモードが決定されます。

例えば，標準設定の実行モードとして［**デバッグ**］を設定すると，開くテストはデバッグ実行モードで開きます。特定のテストの実行モードを変更した場合は，そのテストが開いている間だけ有効となります。テストを保存して閉じ，再度開くと，テストは再び標準の実行モード（この例では［**デバッグ**］）で開きます。

標準の実行モードを設定するには，次の手順を実行します。

1 ［**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・ボックスが表示されます。

2 ［**実行**］カテゴリを選択します。



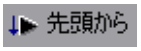
3 ［**標準実行モード**］ボックスで実行モードを選択します。

4 ［**OK**］をクリックして，更を保存し，［一般オプション］ダイアログ・ボックスを閉じます。

注：このオプションは，設定を変更した後に開いたテストにのみ適用されます。すでに WinRunner で開いていたテストには影響しません。

WinRunner の実行コマンド

テストを実行するには，実行コマンドを使います。テストの実行中，テスト・スクリプトの左のマージンにある実行矢印が，そのとき解釈されている TSL ステートメントを示します。



［先頭から実行］

［**先頭から実行**］コマンドを選ぶか，対応する［**先頭から実行**］ボタンをクリックして，テスト・スクリプトの先頭からアクティブなテストを実行します。テストが別のテストを呼び出している場合，WinRunner は呼び出し先のテストのスクリプトを表示します。実行は，テスト・スクリプトの最後で停止します。

［矢印から実行］

［**矢印から実行**］コマンドを選ぶか，対応する［**矢印から実行**］ボタンをクリックして，実行矢印が示すスクリプト行からアクティブなテストを実行します。この点を除けば，［矢印から実行］コマンドは［先頭から実行］コマンドと同じです。

［最小化して実行］のコマンド

［最小化して実行］のコマンドは，テスト実行の際にテスト対象アプリケーションが画面全体を利用できるようにしたい場合に使います。［最小化して実行］のコマンドは，テスト実行の際に，WinRunner ウィンドウを最小化してアイコンにします。テストが終了した場合，またはテスト実行を停止あるいは一時停止した場合，WinRunner ウィンドウが自動的に元の大きさで表示されます。［最小化して実行］のコマンドでは，テストをテスト・スクリプトの先頭から，または，実行矢印から実行できます。

［**最小化して実行**］のコマンドは次のとおりです。

- ［**最小化して実行**］＞［**先頭から**］コマンド
- ［**最小化して実行**］＞［**矢印から**］コマンド

［ステップ］コマンド

［ステップ］コマンドを使用するか，［ステップ］ボタンをクリックして，テスト・スクリプトを 1 行だけ実行します。［ステップ］コマンドの詳細については，第 38 章「テスト実行の制御」を参照してください。

ステップ・ボタンは次のとおりです。

［**ステップ実行**］ボタン

［**ステップ イントゥ**］ボタン

［ステップ］コマンドは次のとおりです。

- ［**ステップ**］コマンド
- ［**ステップ イントゥ**］コマンド
- ［**ステップ アウト**］コマンド
- ［**カーソル位置まで実行**］コマンド

［停止］

［**停止**］コマンドを選択するか，［**停止**］ボタンをクリックすることで，テスト実行を直ちに停止することができます。テストを停止すると，テストの変数と配列は未定義となります。ただし，テスト・オプションは，現在の値を保持したままです。詳細については，437 ページ「テスト・オプションによるテスト実行の制御」を参照してください。

テストを停止した後は，**load** コマンドを使ってロードした関数にしか起動できません。［実行］のコマンドを使ってコンパイルした関数は起動できません。これらの関数を再コンパイルしてから起動する必要があります。詳細については，第 31 章「テストでのユーザ定義関数の利用」を参照してください。

一時停止

［**一時停止**］コマンドを選択するか，［**一時停止**］ボタンをクリックすることで，テスト実行を一時停止できます。実行を直ちに終了する［停止］とは異なり，テストはそれまでに解釈された TSL ステートメントがすべて実行されるまでテストの実行を継続します。テストを一時停止すると，テスト・オプションだけでなく，テストの変数と配列も値を保持し続けます。詳細については，437 ページ「テスト・オプションによるテスト実行の制御」を参照してください。

一時停止したテストを再開するには，適切な実行コマンドを選択します。テストは，テストを一時停止した場所から再開されます。

ソフトキーを使った実行コマンドの選択

WinRunner のコマンドには，ソフトキーを使用して実行できるものがいくつかあります。WinRunner ウィンドウが画面上でアクティブになっていなくても，あるいは WinRunner ウィンドウが最小化されているときでも，WinRunner はソフトキーからの入力を読み込みます。標準のソフトキーは自分で設定できます。ソフトキーの設定の詳細については，第 42 章「WinRunner のユーザ・インタフェースの カスタマイズ」を参照してください。

次の表は，テストを実行するための標準のソフトキーの設定です。

コマンド	標準ソフトキー組み合わせ	機能
RUN FROM TOP	左 Ctrl+F5	テストを先頭から実行します。
RUN FROM ARROW	左 Ctrl+F7	スクリプトの矢印によって示された行からテストを実行します。
STEP	F6	テスト・スクリプト内の現在の行だけを実行します。
STEP INTO	左 Ctrl+F8	**ステップ**と似ています。ただし，現在の行がテストまたは関数を呼び出すと，呼び出し先のテストまたは関数が WinRunner ウィンドウに現れますが，実行はされません。
STEP OUT	CTRL LEFT + 7	**ステップ・イントゥ**と一緒に使用します。呼び出し先テストまたはユーザ定義関数の実行を完了します。
STEP TO CURSOR	左 Ctrl+F9	挿入ポイントによって示された行までテストを実行します。
PAUSE	PAUSE	あらかじめ解釈されている TSL ステートメントがすべて実行されるとテストを停止します。実行は，この場所から再開できます。
STOP	左 Ctrl+F3	テスト実行を停止します。

アプリケーションを検査するためのテスト実行

アプリケーションの動作を検査するためにテストを実行すると，WinRunner は現在の結果を期待結果と比較します。実行の際には，テストの検証結果を保存するフォルダを指定します。

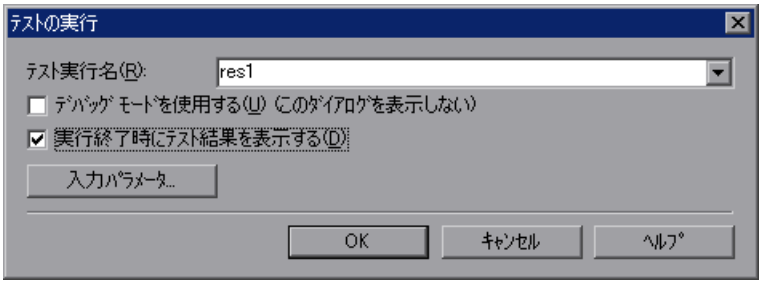
アプリケーションを検査するためにテストを実行するには，次の手順を実行します。

1 テストがまだ開いていなければ，［**ファイル**］＞［**テストを開く**］を選択するか，［**開く**］ボタンをクリックして，テストを開きます。

2 テスト・ツールバーにある実行モードのリストから［**検証**］が選択されていることを確認します。

3 適切な［**テスト**］メニュー・コマンドを選択するか，［**実行**］ボタンのいずれかをクリックします。

［**テスト実行**］ダイアログ・ボックスが開き，検証結果のために標準のテスト実行名を表示します。



4 テスト結果を標準のテスト実行名で保存できます。別の名前を使うには，新しい名前を入力するか，リストから既存の名前を選択します。

テスト実行の後に，WinRunner にテスト結果を自動的に表示させるには（標準の動作です），［**実行終了時にテスト結果を表示する**］チェック・ボックスを選択します。

5 入力パラメータに値を指定するには，［**入力パラメータ**］ボタンをクリックしてテスト実行に使用する値を［入力パラメータ］ダイアログ・ボックスに入力します。詳細については，436 ページ「テスト実行時の入力パラメータの値の指定」を参照してください。

6［**OK**］をクリックします。［テスト実行］ダイアログ・ボックスが閉じ，WinRunner はテストを実行します。テスト結果は，指定したテスト実行名で保存されます。

テスト・スクリプトをデバッグするためのテスト実行

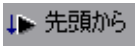
テスト・スクリプトをデバッグするためにテストを実行すると，WinRunner は現在の結果と期待結果を比較します。あらゆる違いがデバッグ結果フォルダに保存されます。WinRunner はデバッグ・モードでテストを実行するたびに，以前のデバッグ結果を上書きします。

テスト・スクリプトをデバッグするためにテストを実行するには，次の手順を実行します。

1 まだ開いていなければ［**ファイル**］＞［**テストを開く**］を選択して，テストを開きます。

2 標準ツールバーにある実行モードのリストから［**デバッグ**］を選択します。

3 適切な［**実行**］または［**デバッグ**］メニュー・コマンドを選択します。



テスト全体を実行するには，［**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。テストがテスト・スクリプトの先頭から実行され，1 組のデバッグ結果が生成されます。

テストの一部だけを実行するには，以下のコマンドの中からどれか 1 つを選ぶか，対応するボタンから 1 つを選んでクリックします。テストは，選択した実行コマンドに従って実行され，デバッグ結果が生成されます。

［**テスト**］＞［**矢印から実行**］

［**テスト**］＞［**最小化して実行**］＞［**矢印から**］

［**デバッグ**］＞［**ステップ**］

［**デバッグ**］＞［**ステップ イントゥ**］

［**デバッグ**］＞［**ステップ アウト**］

［**デバッグ**］＞［**カーソル行まで実行**］

テストは，選択した実行コマンドに従って実行され，1 組のデバッグ結果が生成されます。

期待結果を更新するためのテスト実行

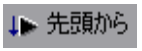
期待結果を更新するためにテストを実行すると，以前に作成した期待結果が，新しい結果に置き換えられ，以降のテスト実行における比較の基準になります。

期待結果を更新するためにテストを実行するには，次の手順を実行します。

1 まだ開いていなければ［**ファイル**］＞［**テストを開く**］を選択して，テストを開きます。

2 テスト・ツールバーにある実行モードのリストから［**更新**］を選択します。

3 適切な［**テスト**］メニュー・コマンドを選択します。



期待結果の組全体を更新するには，［**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。

期待結果の一部だけを更新するには，次のコマンドから 1 つを選ぶか，対応するボタンの中の 1 つをクリックします。

［**テスト**］＞［**矢印から実行**］

［**テスト**］＞［**最小化して実行**］＞［**矢印から**］

［**デバッグ**］＞［**ステップ**］

［**デバッグ**］＞［**ステップ イントゥ**］

［**デバッグ**］＞［**ステップ アウト**］

［**デバッグ**］＞［**カーソル行まで実行**］

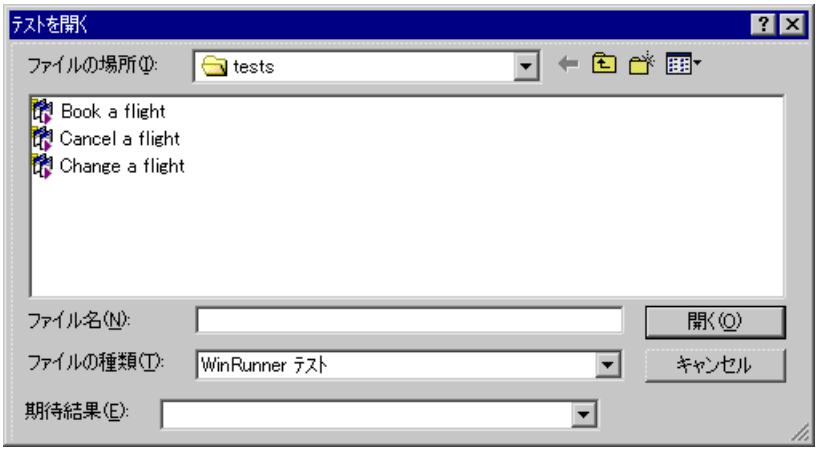
WinRunner は選択された［**テスト**］メニュー・コマンドに従ってテストを実行し，期待結果を更新します。期待結果の標準のフォルダは「*exp*」です。

複数の期待結果の生成

任意のテストの複数の期待結果のセットを生成できます。アプリケーションの応答が時間帯によって変わる場合などには，複数の期待結果のセットを用意したいと思うでしょう。このような場合，それぞれの時間帯に応じて，複数の期待結果セットを生成します。

現在の期待結果と異なる期待結果のセットを生成するには，次の手順を実行します。

1 ［**ファイル**］＞［**テストを開く**］を選択するか，［**開く**］ボタンをクリックします。［テストを開く］ダイアログ・ボックスが開きます。



2 ［**テストを開く**］ダイアログ・ボックスで，複数の期待結果の組の生成先のテストを選択します。［**期待結果**］ボックスに，新しい期待結果を保存するためのフォルダの名前として，一意の名前を入力します。

注：すでに開いているテストに新しい期待結果セットを作成するには，［**ファイル**］＞［**テストを開く**］を選択するか，［**開く**］ボタンをクリックして［テストを開く］ダイアログ・ボックスを開き，開いているテストを選んでから［**期待結果**］ボックスに新しい期待結果フォルダの名前を入力します。

---

3 ［**開く**］をクリックします。［テストを開く］ダイアログ・ボックスが閉じます。

4 テスト・ツールバーにある実行モードのリストから［**更新**］を選択します。

先頭から

5 ［**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て，新しい期待結果セットを生成します。

WinRunner はテストを実行して，指定されたフォルダに新しい期待結果セットを生成します。

複数の期待結果セットを持つテストの実行

テストに複数の期待結果がある場合は，テストを実行する前に，どの期待結果を使うかを指定しなければなりません。

複数の期待結果セットがあるテストを実行するには，次の手順を実行します。

1 ［**ファイル**］＞［**テストを開く**］を選択するか，［**開く**］ボタンをクリックします。［テストを開く］ダイアログ・ボックスが開きます。

注：すでにテストが開いており，それが目的の期待結果とは異なる期待結果を対象としている場合は，［**ファイル**］＞［**テストを開く**］を選択するか，［**開く**］ボタンをクリックして，もう一度［テストを開く］ダイアログ・ボックスを開いて，開くテストを選んでから，［**期待結果**］ボックスに正しい期待結果フォルダを選択します。

---

2 ［**テストを開く**］ダイアログ・ボックスで，実行したいテストをクリックします。［**期待結果**］ボックスには，選択したテストのすべての期待結果セットが含まれています。

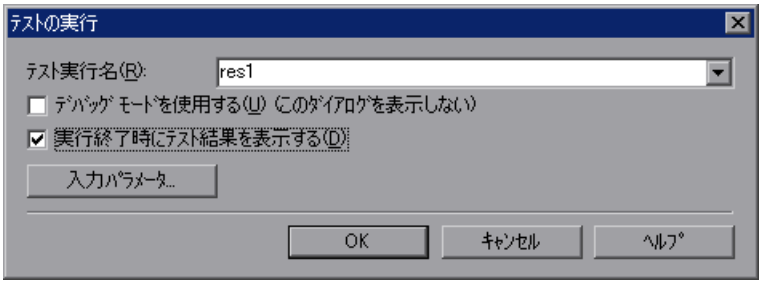
3 ［**期待結果**］ボックスのリストから，必要な期待結果セットを選択して，［**開く**］をクリックします。［テストを開く］ダイアログ・ボックスが閉じます。

4 テスト・ツールバーにある実行モードのドロップダウン・リストから［**検証**］を選択します。

5 適切な［**実行**］メニュー・コマンドを選択します。［**テスト実行**］ダイアログ・ボックスが開き，検証結果の標準のテスト実行名が表示されます。例えば，「*res1*」になります。

6 入力パラメータに値を指定するには，［**入力パラメータ**］をクリックして，［入力パラメータ］ダイアログ・ボックスでテスト実行に使用する値を入力しま

す。詳細については，436 ページ「テスト実行時の入力パラメータの値の指定」を参照してください。



7 ［**OK**］をクリックすれば，テストの実行が開始され，テスト結果が標準のフォルダに保存されます。別の検証結果フォルダを使いたい場合は，新しい名前を入力するか，リストから既存の名前を選択します。

［テスト実行］ダイアログ・ボックスが閉じます。WinRunner は選択された［実行］メニュー・コマンドに従ってテストを実行し，期待結果を指定されたフォルダに保存します。

テスト実行時の入力パラメータの値の指定

テストに 1 つまたは複数の入力パラメータがある場合，テストの実行を開始するときにこれらのパラメータに値を指定します。これらの値は，現在のテスト実行にのみ使用され，テストと一緒に保存はされません。

テストを実行するときに入力パラメータに値を指定しないと，入力パラメータに標準設定の値を指定していれば，その値が使用されます。標準設定の値を指定していないとパラメータは空の値を返します。テストは実行されますが，ステップに空でない値が必要な場合はステップが失敗する可能性があります。

標準設定のパラメータ値の詳細については，503 ページ「テスト・パラメータの管理」を参照してください。

入力パラメータに値を指定するには，次の手順を実行します。

1 ［テストの実行］ダイアログ・ボックスで，［**入力パラメータ**］ボタンをクリックします。［入力パラメータ］ダイアログ・ボックスが開きます。

2 ［入力パラメータ］ダイアログ・ボックスで値を指定する入力パラメータの行の［**値**］セルをクリックします。表示される編集領域に値を入力します。

3 値を指定する入力パラメータごとに手順 2 を繰り返します。

4 ［**OK**］をクリックします。

入力パラメータの詳細については 700 ページ「テスト・パラメータを使った作業」を参照してください。

テスト・オプションによるテスト実行の制御

WinRunner のテスト・オプションを使って，テストの実行を制御できます。例えば，ビットマップ・チェックポイントでビットマップが現れるまで，WinRunner を待機させる時間や，テストの実行速度などを設定できます。

テスト・オプションは，［一般オプション］ダイアログ・ボックスで設定します。このダイアログ・ボックスは，［**ツール**］＞［**一般オプション**］を選択すると開きます。また，**setvar** 関数を使って，テスト・スクリプトの中からテスト・オプションを設定することも可能です。

テスト・オプションにはそれぞれ標準値があります。例えば，ビットマップ・オプション間の許容差異（ビットマップが不一致となるピクセルの最小数）の標準値は 0 です。これは［一般オプション］ダイアログ・ボックスの［**実行**］＞［**設定**］カテゴリでグローバルに設定できます。テスト・オプションのグローバルな設定の詳細については，第 22 章「グローバル・テスト・オプションの設定」を参照してください。

また，**setvar** 関数を使用して，テスト・スクリプト内から対応する *min_diff* オプションを設定できます。テスト・スクリプト内からテスト・オプションを設定する方法については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

新しい値をテスト・オプションに割り当てると，WinRunner の終了時に WinRunner の設定への変更を保存するよう WinRunner から指示されます。

テスト実行に関する一般的な問題の解決法

コンテキスト・センシティブ・テストを実行すると，WinRunner から実行ウィザードが開く場合があります。一般に，実行ウィザードが開くのは，WinRunner がアプリケーション内でオブジェクトやウィンドウの場所を特定できないときです。その場合，次のようなメッセージが表示されます。

これには，以下の原因が考えられます。それぞれについて解決策を示します。

考えられる原因	解決策
WinRunner を終了したときに保存しなかった一時 GUI マップを使って作業していた。一時 GUI マップに保存されたオブジェクトは，必ずしもセッション間で保存されないため，WinRunner の終了後は GUI マップ・ファイル内の存在に依存することはできません。	WinRunner を使ってアプリケーションを再学習し，GUI オブジェクトの論理名と物理的記述を GUI マップに保存します。終了したら、GUI マップ・ファイルを忘れずに保存します。テストを実行するときには，必ず GUI マップ・ファイルを**ロード**します。詳細については，第 5 章「グローバル GUI マップ・ファイル・モードでの作業」を参照してください。
GUI マップ・ファイルを保存したが，そのファイルがロードされていない。	テスト用に GUI ファイルをロードします。GUI ファイルは［GUI マップ エディタ］からその都度手作業でロードできます。また，テスト・スクリプトの先頭に **GUI_load** ステートメントを追加することもできます。詳細については，第 5 章「グローバル GUI マップ・ファイル・モードでの作業」を参照してください。

考えられる原因	解決策
オブジェクトに動的ラベルが付いていないため，テストの実行中にオブジェクトが特定されなかった。例えばウィンドウのタイトルバーにはアプリケーション名が表示されたり，アクティブなドキュメント名が表示されたりする場合があります（サンプルのフライト予約アプリケーションでは，［FAX 注文番号］ウィンドウのラベルが変わります)。このようにラベルが変化するオブジェクトを含むアプリケーションをテストする場合にこの問題が生じます。	正規表現を使って，WinRunner がオブジェクトをその物理的記述の一部に基づいて認識できるようにします。正規表現の詳細については，第 27 章「正規表現の使い方」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 が問題のあるオブジェクトを認識するのに使う物理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，第 24 章「GUI マップの構成設定」を参照してください。
オブジェクト / ウィンドウの物理的記述が GUI マップの物理的記述と一致していない。	GUI マップの物理的記述を修正します。詳細については，77 ページ「論理名と物理的記述の修正」を参照してください。
テスト・スクリプトでのオブジェクト / ウィンドウの論理名が GUI マップの論理名と一致していない。	GUI マップのオブジェクト / ウィンドウの論理名を修正します。詳細については，77 ページ「論理名と物理的記述の修正」を参照してください。
	テスト・スクリプト内のオブジェクト / ウィンドウの論理名を手作業で修正します。
オブジェクト / ウィンドウの必須またはオプションのプロパティ（［GUI マップの構成設定］に表示）が GUI マップに示されているものと異なる。	［クラスの構成設定］ダイアログ・ボックスで，WinRunner がそのオブジェクト・クラスに対して学習した必須プロパティとオプションのプロパティを，GUI マップの物理的記述と一致するよう設定してください。詳細については，596 ページ「標準またはユーザ定義クラスの構成設定」を参照してください。
	WinRunner は，GUI マップのオブジェクト / ウィンドウを再学習して，オブジェクト・クラスに設定されている必須プロパティとオプションのプロパティを学習する必要があります。詳細については，第 5 章「グローバル GUI マップ・ファイル・モードでの作業」を参照してください。

ヒント：WinRunner は，アプリケーションのオブジェクトの記録を開始する前に，［GUI マップ エディタ］から系統的にアプリケーションを学習します。詳細については，第 5 章「グローバル GUI マップ・ファイル・モードでの作業」を参照してください。

注：テストの実行中に生じる GUI マップの問題を解決する方法については，63 ページ「グローバル GUI マップ・ファイル・モードで作業する場合のガイドライン」を参照してください。

第 20 章

テスト結果の分析

WinRunner からテストまたはコンポーネントを実行した後に，テスト実行中に起きたすべての主要なイベントのレポートを［テスト結果］ウィンドウに表示できます。結果は，標準 WinRunner レポート・ビューまたは統一レポート・ビューで表示できます。

本章では，次の項目について説明します。

- テスト結果の分析について
- 統一レポート・ビューの結果ウィンドウについて
- テスト結果の表示のカスタマイズ
- WinRunner レポート・ビューの結果ウィンドウについて
- テスト実行の結果の表示
- チェックポイントの結果の表示
- シングル・プロパティ検査の結果の分析
- GUI チェックポイントの結果の分析
- テーブルの内容を対象にする GUI チェックポイントの結果の分析
- テーブルの内容を対象とする GUI チェックポイントの期待結果の分析
- ビットマップ・チェックポイントの結果の分析
- データベース・チェックポイントの結果の分析
- データベース・チェックポイントの内容の検査の期待結果の分析
- WinRunner レポート・ビューでのチェックポイントの期待結果の更新
- ファイルの比較の結果の表示
- テスト実行中に検出された不具合の報告

テスト結果の分析について

テストを実行した後に，その結果を［テスト結果］ウィンドウに表示できます。このウィンドウの外観は［一般オプション］ダイアログ・ボックスの［**実行**］カテゴリで選択した［レポート ビュー］オプションによって決まります。表示される結果のタイプは，現在選択している実行モードによって異なります。

テスト結果ビューについて

テスト結果ビューには，次の 2 つのタイプがあります。

- **WinRunner レポート・ビュー**：テスト結果を Windows 形式のビューアに表示します。

  QuickTest テストへの呼び出しを含むテストを実行した場合，WinRunner レポート・ビューには QuickTest テストの結果に関する基本情報しか表示されません。

  QuickTest テストを呼び出すテストを実行するときは，統一レポート・ビューを使用することをお勧めします。

- **統一レポート・ビュー**：テスト結果を HTML スタイルのビューアに表示します。

  統一レポート・ビューアは QuickTest Professional のテスト結果に使用される形式と同一です。

  QuickTest テスト（バージョン 6.5 以降）への呼び出しを含むテストを実行した場合，統一レポート・ビューでは，呼び出された QuickTest テストにおける各ステップの詳細な結果を表示できます。

どちらのレポート・ビューを選択した場合でも，テスト結果ウィンドウには，GUI チェックポイント，ビットマップ・チェックポイント，データベース・チェックポイント，ファイルの比較，エラー・メッセージなど，テスト実行中に起きた主要なイベントの説明が常に表示されます。また，結果の分析に役立つテーブルやイメージも表示されます。

テスト結果のタイプについて

WinRunner のテストに対しては，検証結果，デバッグ結果，または期待結果を表示できます。WinRunner のコンポーネントに対しては，デバッグ結果または期待結果を表示できます。

検証結果では，テスト名のみが［テスト結果］タイトルバーに表示されます。デバッグ結果では，テストまたはコンポーネント名の横に **[debug]** と表示され

ます。期待結果では，テストまたはコンポーネント名の横に **[exp]** と表示されます。

［テスト結果］ウィンドウを WinRunner レポート・ビューで開く場合，ウィンドウには常に最新の実行の結果が表示されます。一方，［テスト結果］ウィンドウを統一レポート・ビューで開く場合，WinRunner のメイン・ウィンドウで選択した実行モードに対応した結果のタイプがあればその結果が表示されます。

例えば，現在選択しているテストが［**検証**］実行モードに設定されている場合，統一レポート・ビューを開くと，最新の検証結果が表示されます。現在選択しているコンポーネントが［**デバッグ**］実行モードに設定されている場合は，デバッグ結果が表示されます。

統一レポート・ビューを開くときに，現在選択している実行モードに対応したテストやコンポーネントについてその結果が存在しない場合には，［テスト結果を開く］ダイアログ・ボックスが［テスト結果］ウィンドウ上に開き，他の結果を選択して表示できます。

注：コンポーネントで作業しているときは，WinRunner でコンポーネントを開いている間に［テスト結果］ウィンドウを開くと，個々のコンポーネントの結果のみ表示できます。［テスト結果を開く］ダイアログ・ボックスを使用して個々のコンポーネントの結果を参照することはできません。［テスト結果を開く］ダイアログ・ボックスを使用して参照できるのは，WinRunner テストの結果，または完全なビジネス・プロセス・テストの結果です。

実行モードの詳細については，422 ページ「WinRunner のテスト実行モード」を参照してください。

テスト結果を開く方法の詳細については，465 ページ「テスト結果を開いて選択したテスト実行を表示する方法」を参照してください。

統一レポート・ビューの結果ウィンドウについて

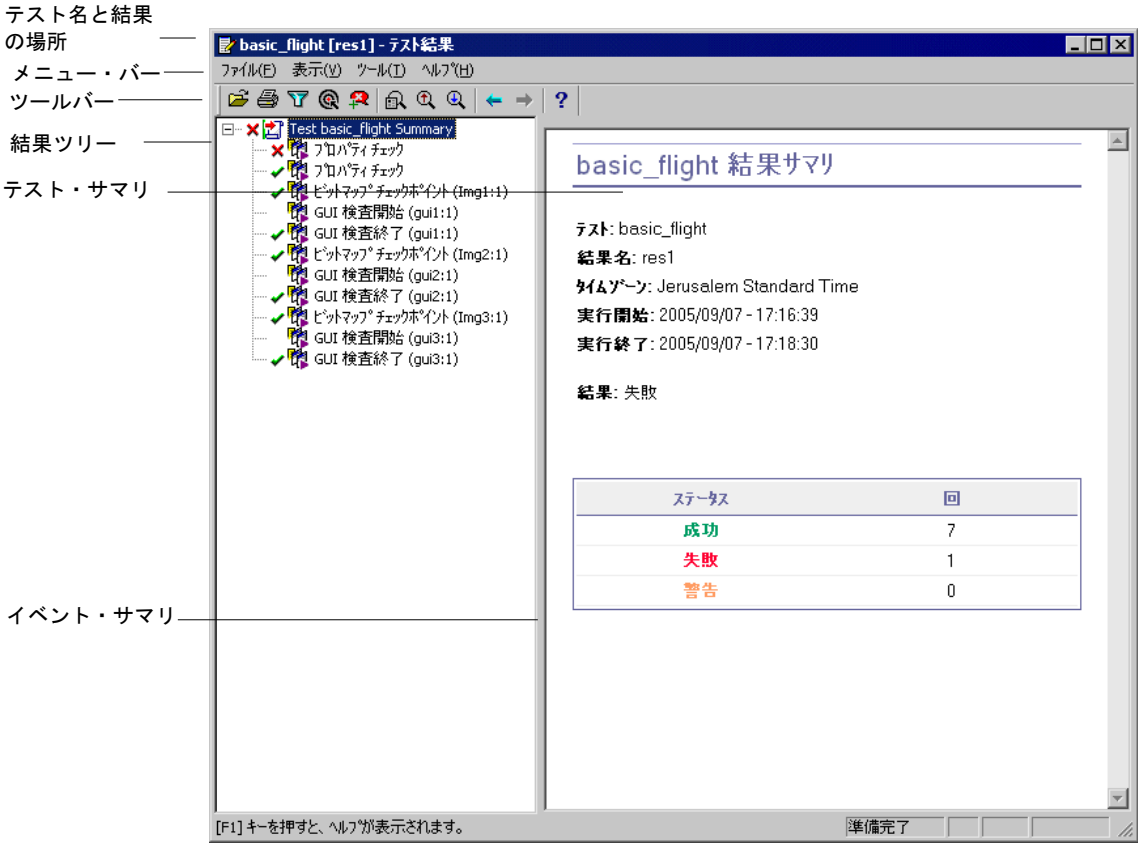
WinRunner を初めて使用する場合や，WinRunner テストと QuickTest テストを統合しようとしている場合は，統一レポート・ビューを使用することをお勧めし

ます。呼び出される QuickTest テストの結果を分析する方法の詳細については，第 47 章「QuickTest Professional との統合」を参照してください。

統一レポートを表示するには，［**ツール**］＞［**一般オプション**］を選択します。［**実行**］カテゴリで，［**統一レポート ビュー**］が選択されていることを確認します。

注：テストの統一レポート・ビューは，テストの実行時に［**統一レポート ビュー**］または［**統一レポート情報を生成する**］オプションを選択した場合にのみ表示できます。［**WinRunner レポート ビュー**］を選択して［**統一レポート情報を生成する**］をクリアした状態でテストを実行した場合は，そのテストの実行に対する統一レポートを表示することはできません。

［テスト結果］ウィンドウを開くには，［**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 の［テスト結果］ウィンドウが統一レポート・ビューで開きます。



テスト結果ウィンドウを開く方法の詳細については，462 ページ「テスト実行の結果の表示」を参照してください。

テスト名と結果の場所

統一レポート・ビューのタイトルバーには，テストの名前とテスト結果フォルダの名前が表示されます。

メニュー・バーとツールバー

メニュー・バーには，テスト結果の分析に使用できるオプションがあります。これらのオプションのいくつかは，対応する［テスト結果］ツールバー・ボタ

ンを使用して実行することもできます。詳細については，下記の説明を参照してください。

- ［**ファイル**］メニュー：テスト結果を開くおよび印刷するためのオプションと，［テスト結果］ウィンドウを閉じるためのオプションがあります。

  - **［開く］**：［テスト結果を開く］ダイアログ・ボックスを開きます。このダイアログ・ボックスで，テストを選択し，そのテストの最新の結果を開くことができます。

  - **［印刷］**：［印刷］ダイアログ・ボックスを開きます。このダイアログ・ボックスで，印刷のためのオプションと，印刷する結果の書式を設定するためのオプションを選択できます。印刷レポート用にデザインをカスタマイズしたユーザ定義 XSL ファイルを選択することもできます。詳細については，451 ページ「テスト結果の印刷」を参照してください。
  - **［印刷プレビュー］**：［印刷プレビュー］ダイアログ・ボックスを開きます。このダイアログ・ボックスで，結果情報とその表示方法を決めるためのオプションを選択できます。オンライン・プレビュー用にデザインをカスタマイズしたユーザ定義 XSL ファイルを選択することもできます。詳細については，452 ページ「テスト結果のプレビュー」を参照してください。
  - **最近使用したファイル**：［テスト結果］ウィンドウで開いた，最近使用した 4 個のファイルを表示します。
  - **［終了］**：［テスト結果］ウィンドウを閉じます。
- **［表示］**：テスト結果ウィンドウの構成要素を表示するためのオプションと，テスト結果の特定の要素を分析するためのオプションがあります。
  - **［テスト結果ツールバー］**：テスト結果ツールバーの表示または非表示を切り替えます。
  - **［ステータス バー］**：テスト結果のステータス・バーの表示または非表示を切り替えます。

  - **［フィルタ］**：［フィルタ］ダイアログ・ボックスを開きます。このダイアログ・ボックスで，表示するテスト・ステップのタイプを選択できます。詳細については，450 ページ「テスト結果の絞り込み」を参照してください。
  - **［すべて展開］**：テスト・ツリーのすべてのステップ・ノードを展開します。
  - **［すべて折りたたみ］**：テスト・ツリーのすべてのステップ・ノードを折りたたみます。

➤ **［ツール］**：Quality Center に接続して不具合を追加する際のオプションと，指定された結果ステータスを持つステップを検索するためにテストをナビゲートする際のオプションがあります。

➤ **［不具合の追加］**：［テスト結果］ウィンドウがすでに Quality Center に接続している場合にこのオプションを選択すると，Quality Center の［不具合の追加］ダイアログ・ボックスが開きます。このダイアログ・ボックスで，不具合を Quality Center プロジェクトに追加できます。

まだ接続していない場合にこのオプションを選択すると，［Quality Center への接続］ダイアログ・ボックスが開きます。Quality Center に接続した後，Quality Center の［不具合の追加］ダイアログ・ボックスが開きます。

➤ **［Quality Center に接続］**：［Quality Center への接続］ダイアログ・ボックスが開きます。このダイアログ・ボックスで，［テスト結果］ウィンドウから Quality Center プロジェクトに接続できます。

注：統一レポート・ビューアはスタンドアロンのアプリケーションです。したがって，WinRunner が Quality Center に接続している場合でも，［テスト結果］ウィンドウから不具合を通知するためには［テスト結果］ウィンドウから Quality Center プロジェクトに接続する必要があります。

➤ **［検索］**：［検索］ダイアログ・ボックスが開きます。このダイアログ・ボックスで，テストを上下にナビゲートし，選択したステータスの結果ステップを検索できます。詳細については，449 ページ「結果ステップの検索」を参照してください。

［検索］ダイアログ・ボックスで検索条件を設定した後，［**前を検索**］ボタンと［**次を検索**］ツールバー・ボタンを使用して，検索条件を満たす前または次のステップに移動できます。

また，［**前のノードに移動**］ボタンと［**次のノードに移動**］ツールバー・ボタンを使用して，テスト結果レポートをナビゲートすることもできます。

➤ **［ヘルプ］**：［テスト結果］ウィンドウに関する追加情報にアクセスするためのオプションがあります。

➤ **［目次と索引］**：［テスト結果ヘルプ］ファイルを開きます。

➤ **［テスト結果のバージョン情報］**：テスト結果アプリケーションに関するサマリ情報を表示するウィンドウが開きます。

結果ツリー

［結果］ツリーには，テストの実行中に実行されたすべてのイベントが階層表示されます。結果ツリーでイベントを選択すると，イベントの詳細が［イベント サマリ］表示枠に表示されます。ツリー内では，ツリーまたは個々のノードを展開または折りたたむことができます。また，［**フィルタ**］オプションや［**検索**］オプションを使用してナビゲーションを簡単にすることもできます。

テスト・サマリ

実行開始時間，実行終了時間，テストの総実行時間，ユーザ名，チェックポイント結果のサマリなど，テストの実行に関する概要情報があります。

注：WinRunner レポート・ビューとは異なり，統一レポート・ビューでは GUI チェックポイント・サマリでの単数のプロパティ・チェックがカウントされます。そのため，統一レポート・ビューでの GUI チェックポイントの総数は WinRunner レポート・ビューに表示される数と異なる場合があります。

イベント・サマリ

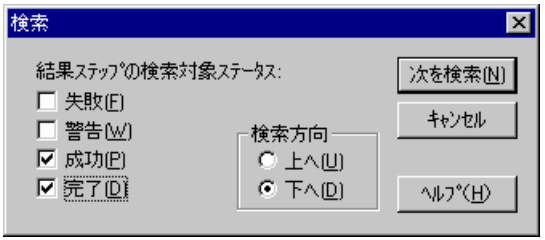
イベント・タイプ，ステータス，行番号，イベント時間，検査対象の基本説明など，結果ツリーで現在選択されているイベントに関するサマリ情報があります。

チェックポイント（単数のプロパティ・チェックを含む）の場合，イベント・サマリにはイベント詳細へのリンクも含まれます。例えば，ビットマップ・チェックポイント用の［**イベント詳細を表示する**］をクリックした場合は，期待される画像，実際の画像，および差分画像が開きます。GUI チェックポイント用のリンクをクリックした場合は，［GUI 検証結果］ウィンドウが開きます。

注：チェックポイントの詳細を表示するためには，テスト結果を実行するコンピュータに WinRunner がインストールされている必要があります。

結果ステップの検索

［検索］ダイアログ・ボックスでは，エラーや警告など特定のステップを，テスト結果から検索することができます。例えば，**成功**と**完了**など，ステータスの組み合わせを選択して検索できます。



次のオプションがあります。

オプション	詳細
失敗	失敗したステップを検索します。
警告	警告が発行されたステップを検索します。
成功	成功したステップを検索します。
完了	実行を完了したステップを検索します。
検索方向	テスト結果のステップを**上へ**（先頭）に向かって検索するか，**下へ**（末尾）に向かって検索するか指定します。

テスト結果の絞り込み

［フィルタ］ダイアログ・ボックスでは，どの結果をそのステータスに従ってテスト結果ツリーに表示するかを絞り込むことができます。

注：［**反復**］オプションと［**内容**］オプションは QuickTest の［テスト結果］ウィンドウからのみ使用できます。テスト結果を WinRunner で表示しているときは，これらのオプションは使用できません。

次のオプションがあります。

オプション	説明
［ステータス］	• ［**失敗**］：失敗したステップのテスト結果を表示します。 • ［**警告**］：ステータスが「警告」のステップ（成功はしなかったがテストが失敗する原因にはならなかったステップ）に関するテスト結果を表示します。 • ［**成功**］：成功したステップのテスト結果を表示します。 • ［**完了**］：ステータスが「完了」のステップ（ステップの実行に成功したが，成功，失敗，警告のステータスを受け取らなかったステップ）に関するテスト結果を表示します。

テスト結果の印刷

［テスト結果］ウィンドウから，テスト結果を印刷できます。印刷するレポートのタイプを選択できます。また，ユーザ定義のレポートの作成や印刷も行えます。

テスト結果を印刷するには，次の手順を実行します。

1 ［**印刷**］ボタンをクリックするか，［**ファイル**］＞［**印刷**］を選択します。［印刷］ダイアログ・ボックスが開きます。

2 ［**印刷範囲**］オプションを選択します。

- ［**すべて**］：すべてのテストまたはコンポーネントの結果を印刷します。
- ［**選択した部分**］：テスト結果ツリーで選択した分岐のテスト結果情報を印刷します。

3 ［**印刷部数**］で，印刷する部数を指定します。

4 ［**印刷形式**］オプションを選択します。

- ［**簡略**］：テスト結果ツリーの各項目のサマリ行（使用可能な場合）を印刷します。このオプションは，手順 2 で［**すべて**］を選択した場合のみ使用できます。
- ［**詳細**］：テスト結果ツリーの各項目の使用可能な情報をすべて印刷します。
- ［**ユーザ定義 XSL**］：ユーザ定義の **.xsl** ファイルを参照したり選択したりできます。印刷するレポートに含める情報やその表示形式を指定するユー

ザ定義の **.xsl** ファイルを作成できます。詳細については，454 ページ「テスト結果の表示のカスタマイズ」を参照してください。

注：［**印刷形式**］オプションは，WinRunner バージョン 8.0 以降で作成したテスト結果にのみ使用できます。

---

5 ［**印刷**］をクリックして，選択したテスト結果情報を標準の Windows プリンタに出力します。

テスト結果のプレビュー

テスト結果は，印刷する前に画面に表示できます。表示する情報の形式や範囲を選択できます。また，情報をユーザ定義形式で表示できます。

注：［**印刷プレビュー**］オプションは，WinRunner バージョン 8.0 以降で作成したテスト結果にのみ使用できます。

テスト結果をプレビューするには，次の手順を実行します。

1 ［**ファイル**］＞［**印刷プレビュー**］を選択します。［印刷プレビュー］ダイアログ・ボックスが開きます。

2 [**印刷範囲**] オプションを選択します。

- [**すべて**]：すべてのテストまたはコンポーネントの結果をプレビューします。
- [**選択した部分**]：テスト結果ツリーで選択した分岐のテスト結果情報をプレビューします。

3 [**印刷形式**] オプションを選択します。

- [**簡略**]：テスト結果ツリーの各項目のサマリ行（使用可能な場合）をプレビューします。このオプションは，手順 2 で [**すべて**] を選択した場合にのみ使用できます。
- [**詳細**]：テスト結果ツリーの各項目の使用可能な情報をすべてプレビューします。
- [**ユーザ定義 XSL**]：ユーザ定義の **.xsl** ファイルを参照したり選択したりできます。プレビューに含める情報やその表示形式を指定するユーザ定義の **.xsl** ファイルを作成できます。詳細については，454 ページ「テスト結果の表示のカスタマイズ」を参照してください。

4 [**プレビュー**] をクリックし，画面にテスト結果のプレビューを表示します。

ヒント：プレビューに表示されない情報がある場合は（例えば，チェックポイント名が長すぎてディスプレイに表示されないなど），[印刷プレビュー] ウィンドウの [**ページの設定**] ボタンをクリックして，ページの向きを [**縦**] から [**横**] に変更します。

テスト結果の表示のカスタマイズ

WinRunner 実行セッションの結果はそれぞれ 1 つの **.xml** ファイル（**results.xml** と呼びます）に保存されます。この **.xml** ファイルには，ディスプレイの左表示枠にあるそれぞれのテスト結果ノードに関する情報が保存されます。

テスト結果ツリーの各ノードは **results.xml** ファイルの要素です。また，テスト結果に表示される異なるタイプの情報を表す要素もあります。サンプルの **results.xml** は **results.xml** ファイルの基本的な構造を示しています。この図では，ステップ要素ノードが折りたたまれています。さまざまなタイプのステップを含んだテストの **results.xml** ファイルを表示することで，ステップ要素の子要素および属性を表示できます。

```
- <Report ver="2.0" tmZone="Standard Time">
    <General productName="WinRunner" productVer="8.00" os="Windows 2000" docLocation="D:\WR\Tests\MyTest"
      host="MyComputer" />
  - <Doc rID="T0" type="Test">
    - <DName>
        <![CDATA[ MyTest  ]]>
      </DName>
    - <Res>
        <![CDATA[ res1  ]]>
      </Res>
    + <Step rID="T1">
    + <Step rID="T2">
    </Doc>
  </Report>
```

テストから QuickTest テストを呼び出している場合，QuickTest 呼び出しの下にあるノードの構造は多少異なります。QuickTest の results.xml の構造に関する詳細については，QuickTest Professional のマニュアルを参照してください。

.xml ファイルからテスト結果情報を取得し，XSL を使用して必要な情報を印刷用や印刷プレビュー表示用としてカスタマイズした形式で表示できます。

XSL には，どのテスト結果情報を表示するか，それをどこでどのように表示，印刷するかを示すツールがあります。また，**.xsl** ファイルが参照する **.css** ファイルを修正して，レポートの外観（フォント，色など）を変更できます。

カスタマイズしたファイルを最初から作成するよりも，WinRunner で提供されている既存の **.xsl** ファイルと **.css** ファイルを修正するほうが簡単です。これらのファイルは**＜ WinRunner のインストール・フォルダ＞\UnifiedReport\dat** にあり，名前は次のとおりです。

- **PShort.xsl**：［印刷］ダイアログ・ボックスで［**簡略**］オプションを選択する場合にテスト結果レポートで印刷する内容を指定します。
- **PDetails.xsl**：［印刷］ダイアログ・ボックスで［**詳細**］オプションを選択する場合にテスト結果レポートで印刷する内容を指定します。
- **PSelection.xsl**：［印刷］ダイアログ・ボックスで［**選択した部分**］オプションを選択する場合にテスト結果レポートで印刷する内容を指定します。
- **PResults.css**：テスト結果の印刷プレビューの外観を指定します。このファイルは 3 つの **.xsl** ファイルすべてによって参照されています。

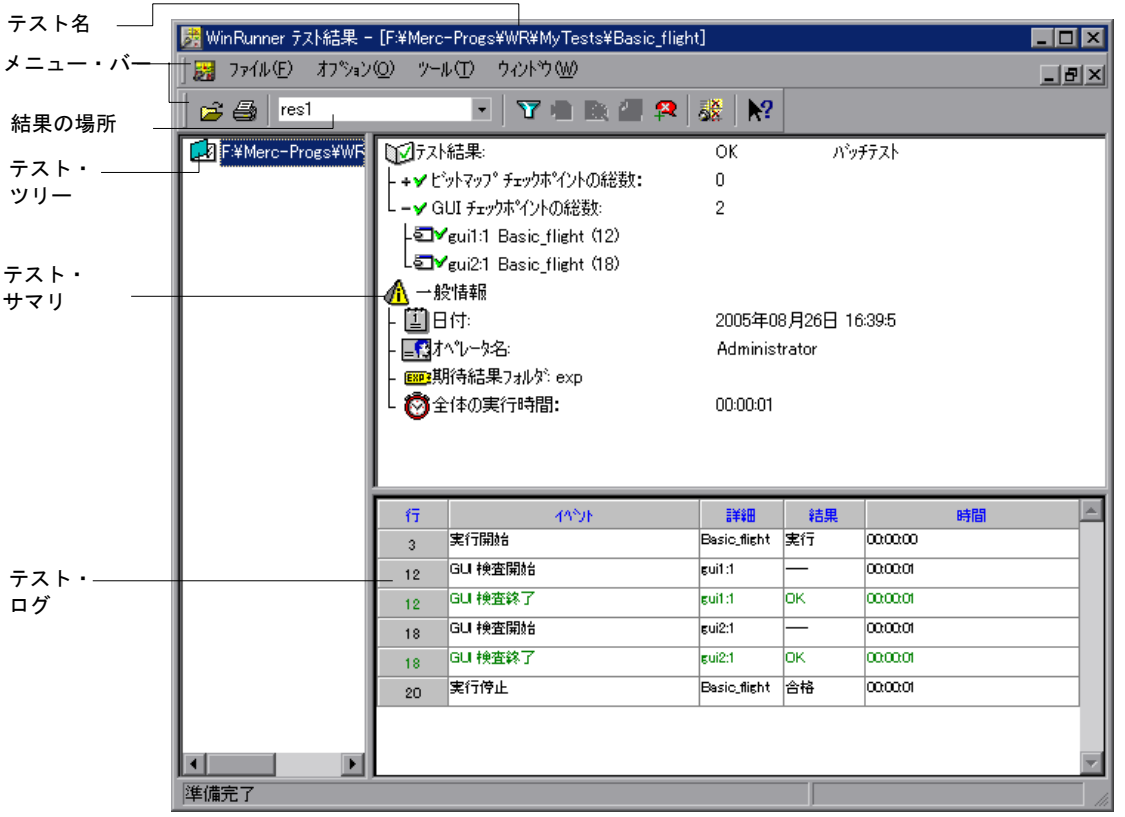
WinRunner レポート・ビューの結果ウィンドウについて

以前のバージョンの WinRunner で作業をした経験があり，呼び出し先の QuickTest テストの結果を分析しない場合は，**WinRunner レポート・ビュー**を使用したほうが便利に感じるかもしれません。

WinRunner レポートを表示するには，［**ツール**］＞［**一般オプション**］を選択します。［**実行**］カテゴリで，［**WinRunner レポート ビュー**］が選択されていることを確認します。

注：標準設定では，WinRunner レポートが表示され，後でテストを実行する場合に統一レポートを表示することを選択できるように，統一レポート・ファイルが作成されます。WinRunner で統一レポート・ファイルを生成しないようにする場合は，［**統一レポート情報を生成する**］オプションをクリアします。

［テスト結果］ウィンドウを開くには，［**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 の［テスト結果］ウィンドウが WinRunner レポート・ビューで開きます。



注：Mercury の［テスト結果］ウィンドウの背景はカスタマイズできます。詳細については，566 ページ「概観オプションの設定」を参照してください。

［テスト結果］ウィンドウを開く方法の詳細については，462 ページ「テスト実行の結果の表示」を参照してください。

テスト名

テスト結果タイトル・バーにテストのフル・パスが表示されます。

メニュー・バーとツールバー

メニュー・バーには，テスト結果の分析に使用できるオプションがあります。これらのオプションのいくつかは，対応する［**テスト結果**］ツールバー・ボタンを使用して実行することもできます。詳細については，下記の説明を参照してください。

- ［**ファイル**］メニュー：テスト結果を開く，閉じる，および印刷するためのオプションと，［テスト結果］ウィンドウを閉じるためのオプションがあります。

  - ［**開く**］：テストを選択し，そのテストの最新の結果を開くことができます。
  - ［**閉じる**］：アクティブなテスト結果ウィンドウを閉じます。

  - ［**印刷**］：［印刷］ダイアログ・ボックスを開きます。このダイアログ・ボックスで，テスト・サマリ表示枠とテスト・ログ表示枠に表示されている情報をテキスト形式で印刷できます。
  - ［**終了**］：WinRunner の［テスト結果］ビューアを終了します。

- ［**オプション**］メニュー：テスト結果の特定の要素を表示および分析するためのオプションがあります。

  - ［**フィルタ**］：［フィルタ］ダイアログ・ボックスを開きます。このダイアログ・ボックスで，テスト・ログに含めるイベントを選択できます。
  - ［**ビットマップ コントロール**］：［ビットマップ コントロール制御］ダイアログ・ボックスを開きます。このダイアログ・ボックスで，ビットマップ・チェックポイント用のビットマップ表示に含める画像を選択できます。詳細については，481 ページ「ビットマップ・チェックポイントの結果の分析」を参照してください。

  - ［**TSL の表示**］：WinRunner テストを WinRunner ウィンドウで開き（まだ開いていない場合），テスト・ログで現在選択している結果行に対応した WinRunner テスト内の行を強調表示します。

  - ［**表示**］：テスト・ログで現在選択している行の結果詳細を開きます。テスト・ログ内の行をダブルクリックしても，このオプションが選択されます。

  - ［**更新**］：選択したチェックポイントの実際の結果に一致するように，選択されたビットマップ，GUI，またはデータベース・チェックポイントに対する期待データを更新します。失敗したビットマップ，GUI，またはデータベース・チェックポイントを選択しているときのみ使用できます。

  - ［**不一致のみ**］：ステータスが［**成功**］または［**OK**］のビットマップ，GUI，またはデータベース・チェックポイントを非表示にします。このオ

プションは，プロパティ検査やその他の非チェックポイント・イベントには効果がありません。

- ［**ツール**］メニュー：テキスト形式の結果ファイルを生成するためのオプションと，不具合を Quality Center に通知するためのオプションがあります。
  - ［**テキスト レポート**］：アクティブなテスト結果ウィンドウのテスト結果をテキスト形式で生成し表示します。
  - 
    ［**不具合レポート**］：テスト・ログ内で選択したイベントの不具合を，現在接続している Quality Center プロジェクトに通知します（このオプションは，Quality Center プロジェクトに接続している場合のみ使用できます）。
- ［**ウィンドウ**］メニュー：追加のテスト結果ウィンドウを開くためのオプションと，それらのウィンドウをメインの［テスト結果］ウィンドウの中で配置するためのオプションがあります。
  - ［**新規ウィンドウ**］：現在アクティブな結果ウィンドウの結果のコピーを表示する，新しい［テスト結果］ウィンドウを開きます。表示されている結果とは別の実行結果を表示するには，結果名を［**結果の場所**］ボックスから選択します。
  - ［**重ねて表示**］：開いているすべての［テスト結果］ウィンドウを重ねて表示します。
  - ［**並べて表示**］：開いているすべての［テスト結果］ウィンドウを左右に並べて表示します。
  - ［**アイコンを整列**］：［テスト結果］ウィンドウ内で最小化されているテスト結果アイコンを整列します。
- 
  ［**ヘルプ**］：［**ヘルプ**］ツールバー・ボタンをクリックし，［テスト結果］ウィンドウ内をクリックして，［WinRunner テスト結果ヘルプ］を表示します。

結果の場所

［**結果の場所**］ボックスでは，どのテスト結果を表示するかを選択できます。指定したテスト実行の，期待結果（**exp**）または実際の結果を表示できます。

テスト・ツリー

テスト・ツリーには，テスト実行中に実行されたすべてのテストが表示されます。ツリーの先頭のテストは，「**呼び出し元のテスト**」といいます。呼び出し元のテストの下にあるテストは，「**呼び出し先のテスト**」といいます。テストの結果を表示するには，ツリーの中で表示したいテストの名前をクリックします。

テスト・サマリ

テスト・サマリには，次の情報が表示されます。

➤ テスト結果

テストが成功したか失敗したかを示します。バッチ・テストの場合は，バッチ・テストが呼び出したテストの結果ではなく，バッチ・テスト自体の結果になります。ツリー内の［テスト結果］分岐をダブルクリックすると，次の内容が表示されます。

ビットマップ・チェックポイントの総数：テスト実行中に現れたビットマップ・チェックポイントの総数が表示されます。ダブルクリックして，チェックポイントの詳細な一覧を表示します。一覧にはそれぞれ，チェックポイントに関する重要な情報が表示されます。例を次に示します。

これには次の意味があります。

要素	説明
	チェックポイントが成功したことを示します。
Img2	キャプチャされたビットマップ・ファイルの名前。
:1	このチェックポイントがスクリプトで初めて実行されたことを示します。
checkpt_loop	テストの名前。
(7)	テスト・スクリプトの 7 行目に **obj_check_bitmap** または **win_check_bitmap** ステートメントがあります。

ビットマップ・チェックポイントの内容を表示するには，一覧でビットマップ・チェックポイントをダブルクリックします。詳細については，481 ページ「ビットマップ・チェックポイントの結果の分析」を参照してください。

GUI チェックポイントの総数：テスト実行中に現れた GUI およびデータベース・チェックポイントの総数が表示されます。

注：統一レポート・ビューとは異なり，WinRunner レポート・ビューでは GUI チェックポイント・サマリでの単数のプロパティ・チェックはカウントされません。そのため，WinRunner レポート・ビューでの GUI チェックポイントの総数は統一レポート・ビューに表示される数と異なる場合があります。

ダブルクリックして，チェックポイントの詳細な一覧を表示します。例えば，一覧にある次の要素を考えます。

gui1:4 checkpt_loop (12)

これには次の意味があります。

要素	説明
gui1	期待結果ファイルの名前。
:4	このチェックポイントがスクリプトで実行されたのが 4 回目であることを示します。
checkpt_loop	テストの名前。
(12)	テスト・スクリプトの 12 行目に **obj_check_gui** または **win_check_gui** ステートメントがあります。

GUI チェックポイントの［GUI チェックポイントの結果］ダイアログ・ボックスを表示するには，その GUI チェックポイントの詳細な説明をダブルクリックします。詳細については，471 ページ「GUI チェックポイントの結果の分析」を参照してください。

➤ 一般情報

［一般情報］アイコンをダブルクリックして，以下のテスト詳細を表示します。

日付：テスト実行の日時。

オペレータ名：テストを実行したユーザの名前。

期待結果フォルダ：GUI チェックポイントとビットマップ・チェックポイントを比較する際に使用する期待結果フォルダの名前。

全体の実行時間：テスト実行の開始から終了までの経過時間の合計（時間：分：秒の形式）。

テスト・ログ

テスト・ログは，テスト実行中に起きたすべての主要なイベントに関する詳細な情報を提供します。この情報には，テストの開始と終了，GUI チェックポイントとビットマップ・チェックポイント，ファイルの比較，テスト・フローの進行の変更，システム変数の変更，表示されたレポート・メッセージ，他のテストの呼び出し，実行時のエラーなどが含まれます。

- 不一致や失敗を説明する行は赤で表示され，成功したイベントを説明する行は緑で表示されます。
- ［**行**］カラムには，イベントが発生したテスト・スクリプトの行番号が表示されます。
- ［**イベント**］カラムには，テスト全体の開始や終了，チェックポイントの開始や終了など，イベントの説明が表示されます。
- ［**詳細**］カラムには，テストの開始時や終了時にはテストの名前，チェックポイントの場合は期待結果ファイルの名前，**tl_step** ステートメントの場合はメッセージなど，イベントに関する特定の情報が表示されます。
- ［**結果**］カラムには，成功か失敗かを判断する必要のあるイベントの結果が表示されます。
- ［**時間**］カラムには，テストの実行開始からイベントを開始するまでに要した時間が（時間，分，秒単位で）表示されます。

ログのイベントをダブルクリックして，以下の情報を表示します。

- ビットマップ・チェックポイントの場合は，テスト実行中にキャプチャした期待ビットマップと実際のビットマップを表示できます。不一致が検出された場合，期待ビットマップと実際のビットマップの差異を示すイメージを表示できます。
- GUI チェックポイントの場合は，テーブルで結果を表示できます。テーブルには，チェックポイント内のすべての GUI オブジェクトの一覧と，各オブジェクトの検証結果の一覧が表示されます。
- ファイルの比較の場合は，比較した 2 つのファイルを表示できます。不一致が検出された場合は，ファイルの一致しない行が強調表示されます。

➤ バッチ・モードで別のテストを呼び出す場合は，**call** ステートメントが渡されたかどうかを表示できます。**call** ステートメントが成功しても，テストの失敗とみなす通常の基準によっては，呼び出されたテストそのものが失敗となる場合もあります。テストの失敗とみなすための基準は，［一般オプション］ダイアログ・ボックスの［**実行**］＞［**設定**］カテゴリで設定できます。詳細については，第 22 章「グローバル・テスト・オプションの設定」を参照してください。

テスト実行の結果の表示

テストまたはコンポーネントの実行後に，その結果を［テスト結果］ウィンドウに表示できます。［テスト結果］ウィンドウには，現在のテストまたはコンポーネントの最新の結果が表示されます。［テスト結果］ウィンドウでは，検証結果（テストの場合），期待結果，およびデバッグ結果を表示できます。

テスト実行の結果を表示するには，次の手順を実行します。

1 ［一般オプション］ダイアログ・ボックスの［**実行**］カテゴリで，必要なレポート・ビューが選択されていることを確認します。詳細については，442 ページ「テスト結果の分析について」および 545 ページ「テストの実行オプションの設定」を参照してください。

2 ［テスト結果］ウィンドウを表示するには，［**ツール**］＞［**テスト結果**］を選択するか，WinRunner のメイン・ウィンドウで［**テスト結果**］ボタンをクリックします。

アクティブでないテストの結果を表示するには，［**開く**］ボタンをクリックするか，［**ファイル**］＞［**開く**］を選択します。［テスト結果を開く］ダイアログ・ボックスで，結果を表示するテストを選択または参照します。

注：統一レポート・ビューからテストを参照する場合は，［**ファイルの種類**］編集ボックスでテスト・タイプとして［**WinRunner テスト**］が選択されていることを確認します。

［テスト実行］ダイアログ・ボックスの［**実行終了時にテスト結果を表示する**］チェックボックスを選択している場合（標準設定），［検証］モードでテストを実行すると，テスト終了時に［テスト結果］ウィンドウが自動的に開きます。詳細については，第 19 章「テスト実行について」を参照してください。

1 標準設定では，直前に実行したテスト実行の結果が［テスト結果］ウィンドウに表示されます。

他のテスト実行の結果を表示するには，次の手順を実行します。

- 統一レポート・ビューでは，［**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，［テスト結果を開く］ダイアログ・ボックスからテスト実行を選択します。詳細については，465 ページ「テスト結果を開いて選択したテスト実行を表示する方法」を参照してください。
- WinRunner レポート・ビューでは，［**ファイルの場所**］ボックスをクリックしてテスト実行を選択します。

2 レポートをテキスト形式で表示するには，WinRunner レポート・ビューを表示し，［テスト結果］ウィンドウで［**ツール**］＞［**テキスト レポート**］を選択します。レポートがメモ帳ウィンドウに表示されます。

3 テスト・ログの［イベント］カラムで特定の種類の結果だけを表示するには，［**オプション**］＞［**フィルタ**］を選択するか［**フィルタ**］ボタンをクリックします。

4 ［テスト結果］ウィンドウからテスト結果を直接印刷するには，［**ファイル**］＞［**印刷**］を選択するか［**印刷**］ボタンをクリックします。

［**印刷**］ダイアログ・ボックスで，印刷する部数を選び，［**OK**］をクリックします。テスト結果はテキスト形式で印刷されます。

1 ［テスト結果］ウィンドウを閉じるには，［**ファイル**］＞［**終了**］を選びます。

Quality Center データベースのテスト実行の結果を表示するには，次の手順を実行します。

1 ［**ツール**］＞［**テスト結果**］を選択するか，WinRunner のメイン・ウィンドウで［**テスト結果**］ボタンをクリックします。

［テスト結果］ウィンドウが開き，アクティブなテストの中で最も新しいテスト実行のテスト結果が表示されます。

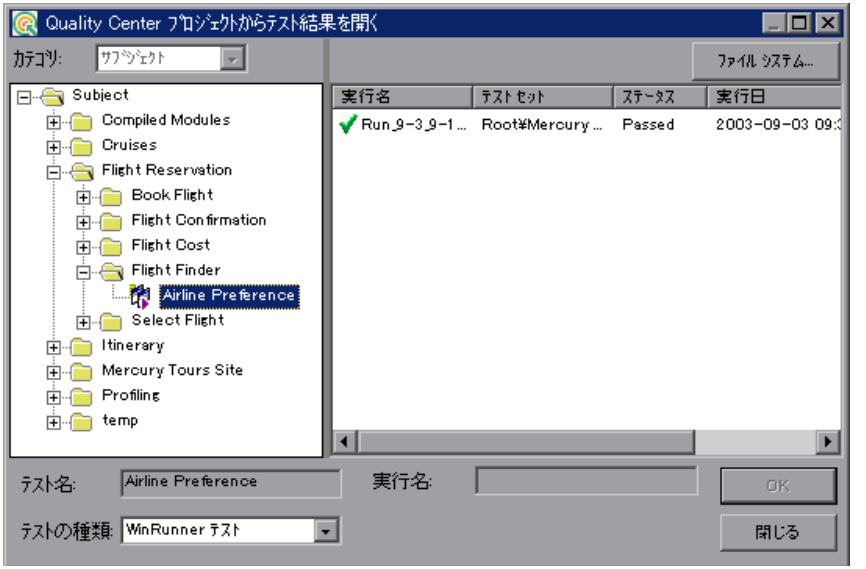
2 Quality Center に接続します。

- 統一レポート・ビューでは，［**Quality Center に接続**］ボタンをクリックします。または，［**ツール**］＞［**Quality Center に接続**］を選択します。
- WinRunner レポート・ビューでは，WinRunner のメイン・ウィンドウに切り替え，［**ツール**］＞［**Quality Center に接続**］を選択します。

3 Quality Center のテスト結果を選択します。

➤ 統一レポート・ビューでは，［**ファイル**］＞［**開く**］を選択します。［テスト結果を開く］ダイアログ・ボックスに，［テスト結果］ウィンドウで現在開いているテストの結果が表示されます。別のテストの結果を表示するには，［**参照**］をクリックします。［Quality Center プロジェクトからテスト結果を開く］ダイアログ・ボックスが開き，テスト計画ツリーが表示されます。

➤ WinRunner レポート・ビューでは，［**ファイル**］＞［**開く**］を選択します。［Quality Center プロジェクトからテスト結果を開く］ダイアログ・ボックスが開き，テスト計画ツリーが表示されます。



4 ［**テストのタイプ**］ボックスで，［**WinRunner テスト**］，［**WinRunner バッチ テスト**］，または［**すべてのテスト**］を選択します。

5 テスト計画ツリーの関連するサブジェクトを選択します。ツリーを展開して下位レベルを表示するには，閉じているフォルダをダブルクリックします。サブレベルの表示を折りたたむには，開いたフォルダをダブルクリックします。

6 表示するテスト実行を選択します。

［**実行名**］カラムには，テスト実行の名前が含まれ，テスト実行が成功したか失敗したかが示されます（このダイアログ・ボックスを WinRunner レポート・

ビューから開いた場合は，選択した実行の［実行名］が読み取り専用の［**実行名**］編集ボックスにも表示されます)。

［**テスト セット**］カラムには，テスト・セットの名前が含まれます。

［**ステータス**］カラムのエントリは，テストが成功したか失敗したかを示します。

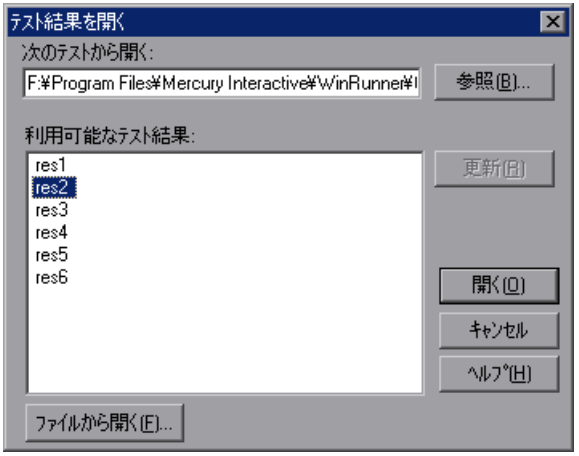
［**実行日**］カラムには，テスト・セットが実行された日付と時刻が表示されます。

7 ［**OK**］をクリックして，選択したテストの結果を表示します。

Quality Center データベースのテスト実行の結果を表示する方法の詳細については，第 48 章「テスト工程の管理」を参照してください。

テスト結果を開いて選択したテスト実行を表示する方法

現在のテストまたはコンポーネントに関して保存されている結果を表示できます。また，他の WinRunner テストまたはビジネス・プロセス・テストに関して保存されている結果も表示できます。［テスト結果を開く］ダイアログ・ボックスからテスト結果を選択して開き，表示します。



現在開いているテストのテスト実行の結果が一覧表示されます。結果セットの 1 つを表示するには，セットを選択して［**開く**］をクリックします。

ヒント：指定したテストのパスを変更した後に結果リストを更新するには，［**更新**］をクリックします。

他のテストに関するテスト実行の結果を表示するには，WinRunner 内で対象のテストを検索するか，またはファイル・システムで統合結果（.qtp）ファイルを検索します。

対象のテストの結果を検索するには，次の手順を実行します。

1 ［テスト結果を開く］ダイアログ・ボックスで，テスト・フォルダのパスを入力するか，［**参照**］をクリックして［テストを開く］ダイアログ・ボックスを開きます。

2 ［**テストのタイプ**］ボックスで，［**WinRunner テスト**］または［**ビジネス プロセス テスト**］を選択します。

3 表示するテスト結果が含まれているテストを見つけて強調表示し，［**開く**］をクリックします。

4 ［テスト結果を開く］ダイアログ・ボックスで，表示するテスト結果セットを強調表示し，［**開く**］をクリックします。

テスト結果ファイルごとに結果を検索するには，次の手順を実行します。

1 ［テスト結果を開く］ダイアログ・ボックスから，［**ファイルから開く**］ボタンをクリックして［結果ファイルの選択］ダイアログ・ボックスを開きます。

2 テスト結果が格納されているフォルダを参照します。標準設定では，結果フォルダの名前は**＜テスト名＞ \res*X*\Report**（**X** はテスト結果の番号 ID）です。

3 表示する統合テスト結果レポート（.qtp）ファイルを選択して強調表示し，［**開く**］をクリックします。

［テスト結果］ウィンドウから Quality Center への接続

［テスト結果］ウィンドウから Quality Center に手作業で不具合を送信する場合や，Quality Center に格納されているテスト結果を表示する場合は，Quality Center に接続する必要があります。

接続プロセスには 2 つの段階があります。まず，WinRunner 統一レポートからローカルまたはリモートの Quality Center Web サーバに接続します。このサーバによって，WinRunner と Quality Center プロジェクトの間の接続が処理されます。

次に，不具合を報告するプロジェクトを選択します。

Quality Center プロジェクトはパスワードで保護されているため，ユーザ名とパスワードを指定する必要があります。

WinRunner 統一レポートから Quality Center に接続するには，次の手順を実行します。

1 ［**ツール**］＞［**Quality Center に接続**］を選択します。［Quality Center への接続］ダイアログ・ボックスが表示されます。

2 ［**サーバ**］ボックスに，Quality Center がインストールされている Web サーバの URL アドレスを入力します。

注：ローカル・エリア・ネットワーク（LAN）または広域エリア・ネットワーク（WAN）を介してアクセス可能な Web サーバを選択できます。

3 ［**接続**］をクリックします。

サーバへの接続が確立されると，［サーバ］ボックスにサーバ名が読み取り専用形式で表示されます。

4 TestDirector 7.5 以降または Quality Center のプロジェクトに接続する場合は，［**ドメイン**］ボックスで，TestDirector または Quality Center のプロジェクトが格納されているドメインを選択します。

TestDirector 7.2 のプロジェクトに接続する場合は，この手順はスキップしてください。

5 ［**プロジェクト**］ボックスで，作業対象のプロジェクトを選択します。

6 ［**ユーザ名**］ボックスに，選択したプロジェクトを開くためのユーザ名を入力します。

7 ［**パスワード**］ボックスにパスワードを入力します。

8 ［**接続**］ボタンをクリックし，WinRunner 統一レポートから，選択したプロジェクトに接続します。

選択したプロジェクトへの接続が確立されると，［プロジェクト］ボックスにプロジェクト名が読み取り専用形式で表示されます。

9 次回 WinRunner の起動時または WinRunner 統一レポートを開くときに，Quality Center サーバと選択したプロジェクトに自動的に再接続するには，［**起動時に再接続する**］チェック・ボックスを選択します。

10 ［**起動時に再接続する**］チェック・ボックスを選択すると，［**起動時に再接続できるようにパスワードを保存する**］チェック・ボックスが有効になります。起動時に再接続するためのパスワードを保存するには，この［**起動時に再接続できるようにパスワードを保存する**］チェック・ボックスを選択します。

パスワードを保存しない場合，起動時に WinRunner から Quality Center に接続するときに，パスワードを入力するよう求められます。

11 ［**閉じる**］をクリックし，［Quality Center への接続］ダイアログ・ボックスを閉じます。Quality Center のアイコンと Quality Center サーバのアドレスがステータス・バーに表示され，WinRunner 統一レポートから現在 Quality Center プロジェクトに接続していることが示されます。

ヒント：ステータス・バーの Quality Center アイコンをダブルクリックすると，［Quality Center への接続］ダイアログ・ボックスが開きます。

Quality Center のプロジェクトから，またはサーバから，あるいはその両方から，切断することができます。WinRunner 統一レポートをプロジェクトから切断する前に Quality Center サーバから切断した場合，WinRunner 統一レポートからそのプロジェクト・データベースへの接続は自動的に切断されます。

チェックポイントの結果の表示

テストの特定のチェックポイントの結果を表示できます。チェックポイントは，アプリケーション内のオブジェクトの振る舞いにおける変更を特定するのに便利です。

チェックポイントの結果詳細を表示する手順は，使用しているレポート・ビューによって異なります。

統一レポート・ビューからチェックポイントの結果を表示するには，次の手順を実行します。

1 ［**ツール**］＞［**テスト結果**］を選択するか，WinRunner のメイン・ウィンドウで［**テスト結果**］ボタンをクリックして，［テスト結果］ウィンドウを開きます。

2 結果ツリーで，検査するチェックポイントを探します。

- 失敗した検査の前には赤の **X** が付き，成功した検査の前には緑のチェック・マークが付きます。
- チェックポイント・ノードはそれぞれチェックポイントのタイプを示します。シングル・プロパティ検査を除くすべてのチェックポイント・ノード

には，チェックポイントの名前と反復も一覧表示されます。これらは，表示するノードを特定するのに役立ちます。

次に例を示します。

end GUI checkpoint (gui3:2)

gui3 は，チェックポイントに対する期待結果ファイルの名前です。コロンの後の 2 は，このチェックポイントがスクリプトの中で実行されたのが 2 回目であること（例えば，ループでの 2 回目の反復など）を示します。

3 分析するチェックポイントのノードをクリックします。チェックポイントに関する基本的な詳細が［**イベント サマリ**］表示枠に表示されます。

4 ［イベント サマリ］表示枠で，［**イベント詳細を表示する**］リンクをクリックします。関連するダイアログ・ボックスが開きます。

5 ［**OK**］をクリックし，ダイアログ・ボックスを閉じます。

以降の各項では，各種のイベント・タイプに対して提供される結果情報について説明します。

WinRunner レポート・ビューからチェックポイントの結果を表示するには，次の手順を実行します。

1 ［**ツール**］＞［**テスト結果**］を選択するか，WinRunner のメイン・ウィンドウで［**テスト結果**］ボタンをクリックして，［テスト結果］ウィンドウを開きます。

2 テスト・ログで，検査するチェックポイントの一覧があるエントリを探します。

- ➤ 失敗した検査は赤で表示され，成功した検査は緑で表示されます。
- ➤ ［**詳細**］カラムにはチェックポイントに関する情報が表示されます。これらは個々のチェックポイントを特定するのに役立ちます。例えば，gui3:2 という例を考えます。

  gui3 は，チェックポイントに対する期待結果ファイルの名前です。コロンの後の 2 は，このチェックポイントがスクリプトの中で実行されたのが 2 回目であること（例えば，ループでの 2 回目の反復など）を示します。

3 テスト・ログで，該当するエントリをダブルクリックします。または，エントリを強調表示して，［**オプション**］＞［**表示**］を選択するか［**表示**］ボタンをクリックします。関連するダイアログ・ボックスが開きます。

4 ［**OK**］をクリックし，ダイアログ・ボックスを閉じます。

以降の各項では，各種のイベント・タイプに対して提供される結果情報について説明します。

シングル・プロパティ検査の結果の分析

プロパティ検査は，アプリケーション内のオブジェクトのプロパティにおける変更を特定するのに便利です。例えば，ボタンが有効か無効か，リストの項目が選択されているかどうかなどを検査できます。

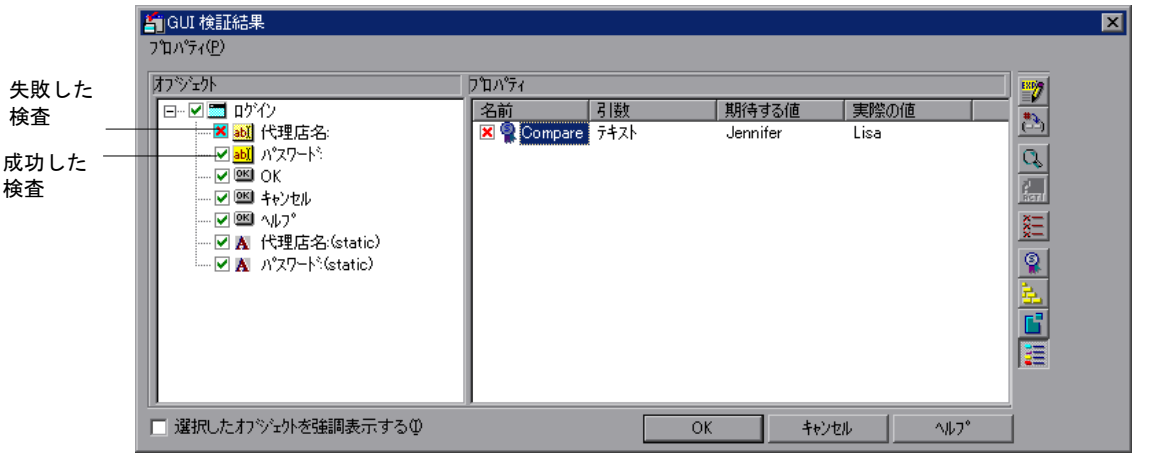
プロパティ検査の期待結果と実際の結果は，［プロパティ］ダイアログ・ボックスに表示されます。このダイアログ・ボックスは［テスト結果］ウィンドウから開けます。

詳細については，第 9 章「GUI オブジェクトの検査」を参照してください。

GUI チェックポイントの結果の分析

GUI チェックポイントは，アプリケーション内の GUI オブジェクトの外観と振る舞いの変更を特定するのに便利です。GUI チェックポイントの結果は［GUI

検証結果］ダイアログ・ボックスに表示されます。このダイアログ・ボックスは［テスト結果］ウィンドウから開けます。



ダイアログ・ボックスには，検査したオブジェクトと実行した検査の種類のすべての一覧が表示されます。検査は，「成功」または「失敗」として記録されます。期待結果と実際の結果も表示されます。オブジェクトが 1 つでも失敗すると，GUI チェックポイント全体が「失敗」としてテスト・ログに記録されます。

WinRunner レポート・ビューで作業しているときは，チェックポイントの期待値を更新できます。詳細については，487 ページ「WinRunner レポート・ビューでのチェックポイントの期待結果の更新」を参照してください。このダイアログ・ボックスの他のオプションの詳細については，473 ページ「［GUI 検証結果］ダイアログ・ボックスのオプション」を参照してください。

詳細については，第 9 章「GUI オブジェクトの検査」を参照してください。

［GUI 検証結果］ダイアログ・ボックスのオプション

［GUI 検証結果］ダイアログ・ボックスには次のオプションがあります。

ボタン	詳細
	［**期待結果値の編集**］：選択したプロパティの期待値を編集できます。詳細については，173 ページ「プロパティの期待値の編集」を参照してください。
	［**引数を指定**］：選択したプロパティ検査の引数を指定できます。詳細については，167 ページ「プロパティ検査への引数の指定」を参照してください。
	［**期待値と実際の値を比較**］：選択したプロパティ検査の期待値と実際の値を表示する［値の比較］ボックスを開きます。テーブルの内容の検査では，検査の期待値と実際の値を表示するデータ比較ビューアを開きます。
	［**期待値の更新**］：期待値を実際の値に更新します。この値は保存されている期待結果の値に上書きされるので注意してください。このオプションは WinRunner レポート・ビューで作業しているときのみ使用できます。
	［**失敗のみ表示**］：失敗した検査だけを表示します。
	［**標準プロパティのみ表示**］：標準のプロパティだけを表示します。
	［**標準以外のプロパティのみ表示**］：Visual Basic，PowerBuilder，ActiveX コントロールのプロパティのような，非標準のプロパティだけを表示します。
	［**ユーザ・プロパティのみ表示**］：ユーザ定義のプロパティ検査のみを表示します。ユーザ定義のプロパティ検査の作成については，『**WinRunner カスタマイズ・ガイド**』を参照してください。
	［**すべてのプロパティを表示**］：標準，非標準，ユーザ定義のプロパティを含むすべてのプロパティを表示します。

テーブルの内容を対象にする GUI チェックポイントの結果の分析

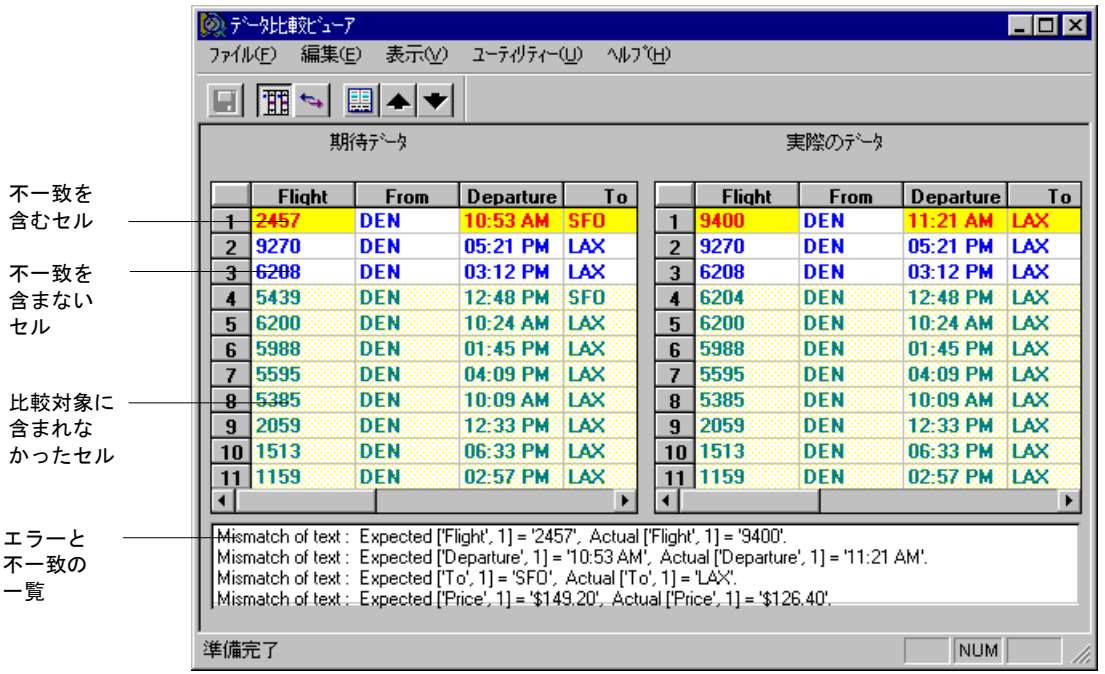
テーブルの内容を対象にする GUI チェックポイントの結果を表示できます。GUI チェックポイントの結果は，［GUI 検証結果］ダイアログ・ボックスに表示されます。このダイアログ・ボックスは［テスト結果］ウィンドウから開けます。このダイアログ・ボックスでは，GUI チェックポイントに含まれているすべてのオブジェクトの一覧と，実行したすべての検査の種類の一覧が表示されます。検査は，「成功」または「失敗」として記録されます。期待結果と実際の結果も表示されます。オブジェクトが 1 つでも失敗すると，GUI チェックポイント全体が「失敗」としてテスト・ログに記録されます。テーブルの内容を対象にする検査の詳細については，第 13 章「テーブル内容の検査」を参照してください。

テーブルの内容を対象にする GUI チェックポイントの結果を表示するには，次の手順を実行します。

1 469 ページ「チェックポイントの結果の表示」で説明した方法で［GUI 検証結果］ダイアログ・ボックスを開きます。

2 テーブル内容チェックポイントを強調表示して［**表示**］ボタンをクリックするか，テーブル内容チェックポイントをダブルクリックします。上の例では，テーブル内容チェックのラベルが「**フライト**」となっています。

データ比較ビューアが開き，期待結果と実際の結果の両方が表示されます。すべてのセルが色分けされ，すべてのエラーと不一致の一覧がウィンドウ下部に表示されます。



次の配色を使用して，ウィンドウで強調表示されている相違を判断します。

- **白い背景に青い文字：**セルは比較対象になっており，不一致は検出されなかった。
- **アイボリーの背景にシアンの文字：**セルが比較対象になっていなかった。
- **黄色い背景に赤い文字：**セルにテキストの不一致が含まれていた。
- **緑の背景にマゼンタの文字：**セルは検証されたが，対応するテーブルにはなかった。
- **背景の色のみ：**セルは空（テキストがない）になっている。

1 標準設定では，データ比較ビューアの［期待データ］のテーブルと［実際のデータ］のテーブルのスクロールは同期します。一方のテーブルで任意のセルをクリックすると，もう一方のテーブルの対応するセルが赤く点滅します。

テーブルごとに別々にスクロールする場合は，［**ユーティリティ**］＞［**スクロールを同期**］を無効にするか，［**スクロールを無効**］ボタンをクリックします。テーブルの隠れている部分を表示するには，必要に応じてスクロール・バーを使用します。

2 データ比較ビューアの下部に表示されているエラーや不一致のリストを絞り込む場合は，次のオプションを使用します。

- **特定のカラムの不一致だけを表示する場合：**テーブルでカラムのヘッダ（カラム名）をダブルクリックします。
- **1 行の不一致を表示する場合：**テーブルの行番号をダブルクリックします。
- **1 つのセルの不一致を表示する場合：**不一致のあるセルをダブルクリックします。

- **前の不一致を表示する場合：**上矢印ボタンをクリックします。

- **次の不一致を表示する場合：**下矢印ボタンをクリックします。

- **すべての不一致を表示する場合：**［**ユーティリティ**］＞［**すべての不一致を表示**］を選択するか，［**すべての不一致を表示**］ボタンをクリックします。
- **リストをクリアする場合：**不一致のないセルをダブルクリックします。
- **リストの不一致に対応するセルを表示する場合：**ダイアログ・ボックス下部のリストで不一致をクリックすると，テーブル内の対応するセルが赤く点滅します。不一致のあるセルが隠れている場合，片方または両方のテーブルが自動的にスクロールして該当するセルが表示されます。

注：WinRunner レポート・ビューで作業しているときは，［GUI 検証結果］ダイアログ・ボックスから［チェックの編集］ダイアログ・ボックスを開いてデータを編集できます。［チェックの編集］ダイアログ・ボックスを開くには，

テーブル内容プロパティ検査を強調表示して，［**期待結果値を編集**］ボタンをクリックします。［チェックの編集］ダイアログ・ボックスでの作業の詳細については，257 ページ「［チェックの編集］ダイアログ・ボックスについて」を参照してください。

---

3 ［**ファイル**］＞［**閉じる**］を選択して，データ比較ビューアを閉じます。

テーブルの内容を対象とする GUI チェックポイントの期待結果の分析

テストの実行前または実行後に，テーブルの内容を対象にする GUI チェックポイントの期待結果を表示できます。GUI チェックポイントの期待結果は，［GUI 検証結果］ダイアログ・ボックスに表示されます。このダイアログ・ボックスは［テスト結果］ウィンドウから開けます。テーブルの内容を対象にする GUI チェックポイントの期待結果を［テスト結果］ウィンドウから表示するには，期待（「exp」）モードを選ぶ必要があります。

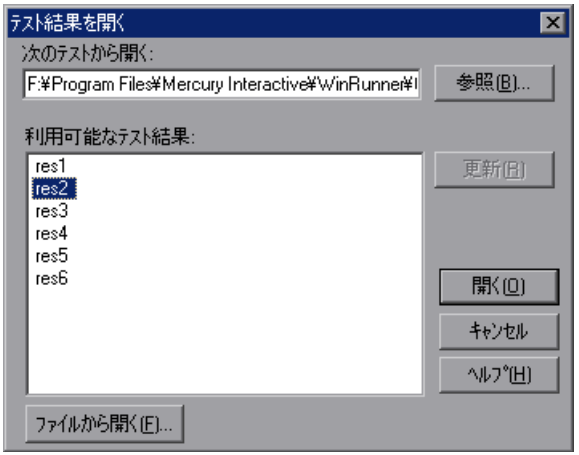
テーブルを対象にする GUI チェックポイントの期待結果は［チェックの編集］ダイアログ・ボックスでも表示できます。詳細については，第 13 章「テーブル内容の検査」を参照してください。

テーブルの内容を対象にする GUI チェックポイントの期待結果を表示するには，次の手順を実行します。

1 ［テスト結果］ウィンドウを開き，期待結果を表示するテストを表示します。詳細については，469 ページ「チェックポイントの結果の表示」を参照してください。

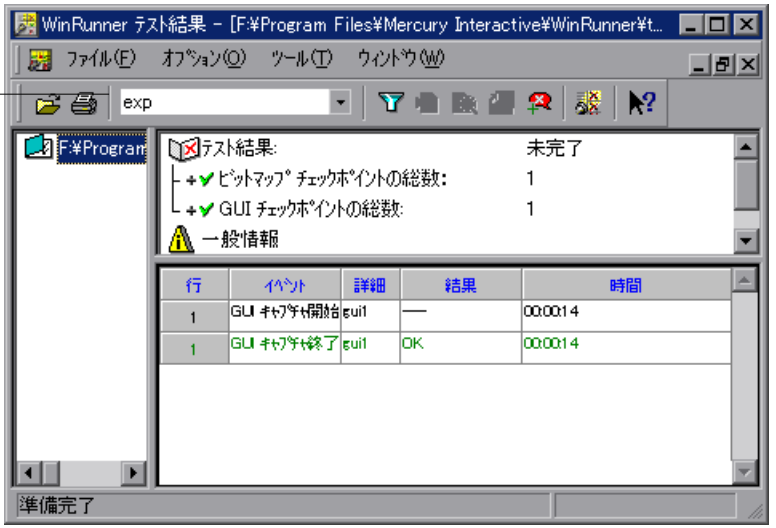
2 期待結果を表示します。

➤ 統一レポート・ビューでは，［**開く**］ボタンをクリックします。または，［**ファイル**］＞［**開く**］を選択します。［テスト結果を開く］ダイアログ・ボックスが開きます。［**exp** ］を選択し，［**開く**］をクリックします。



➤ WinRunner レポート・ビューでは，［結果の場所］ボックスで［**exp**］を選択します。

［結果の場所］ボックス

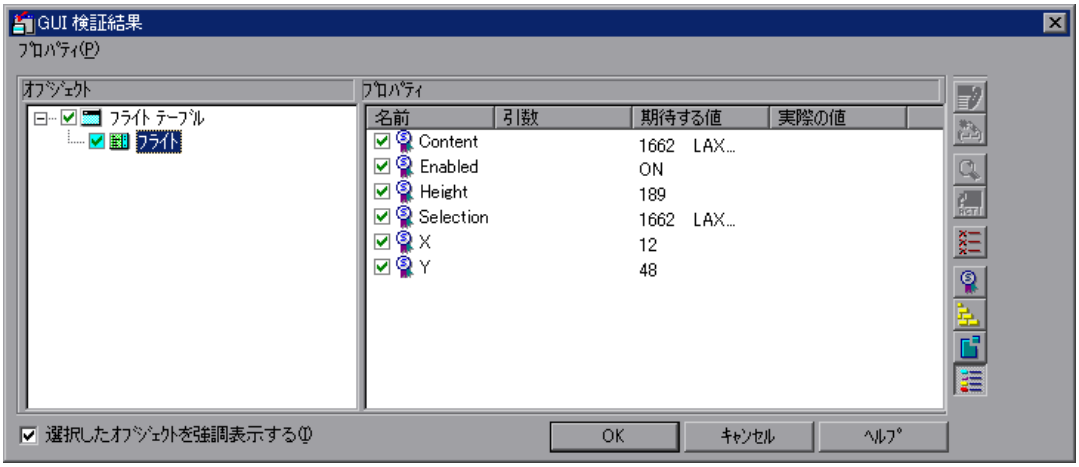


3 期待結果を表示します。

➤ 統一レポート・ビューでは，分析する検査の結果ツリー・ノードをクリックします。チェックポイントに関する基本的な詳細が［**イベント サマリ**］表示枠に表示されます。［イベント サマリ］表示枠で，［**イベント詳細を表示する**］リンクをクリックします。

➤ WinRunner レポート・ビューでは，テスト・ログにあるテーブル検査の［**GUI キャプチャ終了**］エントリをダブルクリックします。または，エントリを強調表示して，［**オプション**］＞［**表示**］を選択するか［**表示**］ボタンをクリックします。

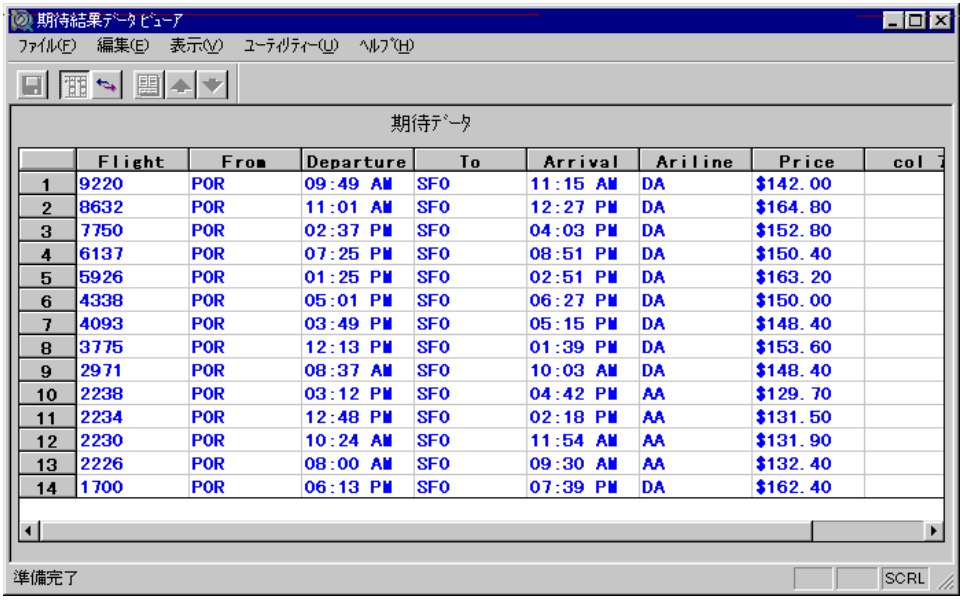
［**GUI 検証結果**］ダイアログ・ボックスが開き，選択した GUI チェックポイントの期待結果が表示されます。



注：GUI チェックポイントの「期待」結果を表示しているので，「実際」の値は表示されません。

4 テーブル内容検査を強調表示して［**表示**］ボタンをクリックするか，テーブル内容検査をダブルクリックします。

期待結果データ・ビューアが開き，期待結果が表示されます。



	Flight	From	Departure	To	Arrival	Airline	Price	col 7
1	9220	POR	09:49 AM	SFO	11:15 AM	DA	$142.00	
2	8632	POR	11:01 AM	SFO	12:27 PM	DA	$164.80	
3	7750	POR	02:37 PM	SFO	04:03 PM	DA	$152.80	
4	6137	POR	07:25 PM	SFO	08:51 PM	DA	$150.40	
5	5926	POR	01:25 PM	SFO	02:51 PM	DA	$163.20	
6	4338	POR	05:01 PM	SFO	06:27 PM	DA	$150.00	
7	4093	POR	03:49 PM	SFO	05:15 PM	DA	$148.40	
8	3775	POR	12:13 PM	SFO	01:39 PM	DA	$153.60	
9	2971	POR	08:37 AM	SFO	10:03 AM	DA	$148.40	
10	2238	POR	03:12 PM	SFO	04:42 PM	AA	$129.70	
11	2234	POR	12:48 PM	SFO	02:18 PM	AA	$131.50	
12	2230	POR	10:24 AM	SFO	11:54 AM	AA	$131.90	
13	2226	POR	08:00 AM	SFO	09:30 AM	AA	$132.40	
14	1700	POR	06:13 PM	SFO	07:39 PM	DA	$162.40	

注：WinRunner レポート・ビューで作業しているときは，［GUI 検証結果］ダイアログ・ボックスから［チェックの編集］ダイアログ・ボックスを開いてデータを編集できます。［チェックの編集］ダイアログ・ボックスを開くには，**TableContent**（または対応する）プロパティ検査を強調表示して，［**期待結果値を編集**］ボタンをクリックします。［チェックの編集］ダイアログ・ボックスでの作業の詳細については，257 ページ「［チェックの編集］ダイアログ・ボックスについて」を参照してください。

---

5 ［**ファイル**］＞［**閉じる**］を選択して期待結果データ・ビューアを閉じます。

ビットマップ・チェックポイントの結果の分析

ビットマップ・チェックポイントは，アプリケーションの期待ビットマップと実際のビットマップとを比較します。［テスト結果］ウィンドウでは，期待結果と実際の結果のイメージを表示できます。［検証］モードまたは［デバッグ］モードでテストを実行中に，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，期待ビットマップ，実際のビットマップ，差異ビットマップが表示されます。［更新］モードでのテストの実行中に不一致が生じた場合は，期待ビットマップだけが表示されます。

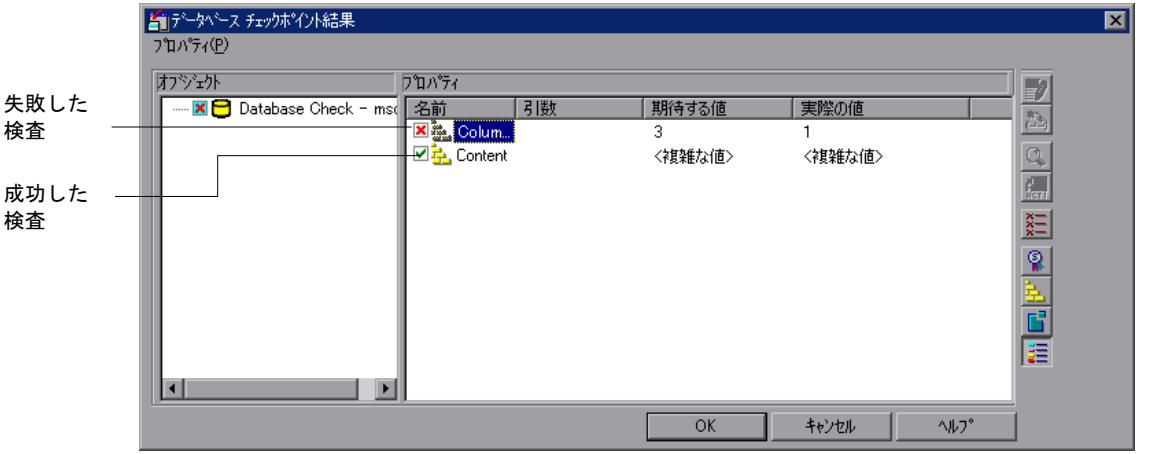
期待　　実際　　差異

WinRunner レポート・ビューで結果を表示している場合は，ビットマップ・チェックポイントの結果を表示するときに，表示するビットマップの種類（期待，実際，差異）を設定できます。制御を設定するには，［テスト結果］ウィンドウで［**オプション**］＞［**ビットマップ コントロール**］を選択します。

注：まったく同一のビットマップを対象としたビットマップ・チェックポイントでも，チェックポイントの作成時およびテストの実行時に異なるディスプレイ・ドライバを使用している場合は，失敗することがあります。これは，異なるディスプレイ・ドライバによって，同じビットマップがわずかに異なる色定義を使用して描画されることがあるためです。詳細については，332 ページ「画面表示ドライバの違いに対する対応」を参照してください。

データベース・チェックポイントの結果の分析

データベース・チェックポイントは，アプリケーション内のデータベースで変更された内容や構造を特定するのに便利です。データベース・チェックポイントの結果は，［データベース チェックポイント結果］ダイアログ・ボックスに表示されます。このダイアログ・ボックスは［テスト結果］ウィンドウから開けます。



ダイアログ・ボックスでは，検査したデータベースと実行した検査の種類が表示されます。検査は，「成功」または「失敗」として記録されます。期待結果と実際の結果も表示されます。データベースでプロパティ検査が 1 つでも失敗すると，データベース・チェックポイント全体が「失敗」としてテスト・ログに記録されます。

WinRunner レポート・ビューで作業しているときは，チェックポイントの期待値を更新できます。詳細については，487 ページ「WinRunner レポート・ビューでのチェックポイントの期待結果の更新」を参照してください。このダイアログ・ボックスの他のオプションの詳細については，483 ページ「［データベース チェックポイント結果］ダイアログ・ボックスのオプション」を参照してください。

注：WinRunner レポート・ビューで作業しているときは，［データベース チェックポイント結果］ダイアログ・ボックスから［チェックの編集］ダイアログ・ボックスを開いてデータを編集できます。［チェックの編集］ダイアログ・ボックスを開くには，**Content** 検査を強調表示して，［**期待結果値の編集**］ボタンをクリックします。［チェックの編集］ダイアログ・ボックスでの作業の詳細については，298 ページ「［チェックの編集］ダイアログ・ボックスについて」を参照してください。

詳細については，第 14 章「データベースの検査」を参照してください。

［データベース チェックポイント結果］ダイアログ・ボックスのオプション

［データベース チェックポイント結果］ダイアログ・ボックスには次のオプションがあります。

ボタン	詳細
	［**期待結果値の編集**］：選択したプロパティの期待値を編集できます。詳細については，287 ページ「データベースを対象とするユーザ定義の検査の作成」を参照してください。
	［**期待値と実際の値を比較**］：選択したプロパティ検査の期待値と実際の値を表示する［値の比較］ボックスを開きます。**Content** 検査では，検査の期待値と実際の値を表示するデータ比較ビューアを開きます。
	［**期待値の更新**］：期待値を実際の値に更新します。この値は保存されている期待結果の値に上書きされますので注意してください。このオプションは WinRunner レポート・ビューで作業しているときのみ使用できます。
	［**失敗のみ表示**］：失敗した検査だけを表示します。
	［**標準プロパティのみ表示**］：標準のプロパティだけを表示します。

ボタン	詳細
	［**標準以外のプロパティのみ表示**］：Visual Basic，PowerBuilder，ActiveX コントロールのプロパティのような，非標準のプロパティだけを表示します。
	［**すべてのプロパティを表示**］：標準，非標準，ユーザ定義のプロパティを含むすべてのプロパティを表示します。

データベース・チェックポイントの内容の検査の期待結果の分析

テストの実行前または実行後に，データベース・チェックポイントの内容の検査の期待結果を表示できます。データベース・チェックポイントの期待結果は，［データベース チェックポイント結果］ダイアログ・ボックスに表示されます。このダイアログ・ボックスは［テスト結果］ウィンドウから開けます。データベース・チェックポイントの内容の検査の期待結果を［テスト結果］ウィンドウから表示するには，［結果の場所］ボックスで期待（「**exp**」）モードを選ぶ必要があります。

データベース内容チェックポイントの期待結果は［チェックの編集］ダイアログ・ボックスでも表示できます。詳細については，第 14 章「データベースの検査」を参照してください。

データベース・チェックポイントの内容の検査の期待結果を表示するには，次の手順を実行します。

1 ［テスト結果］ウィンドウを開き，期待結果を表示するテストを表示します。詳細については，469 ページ「チェックポイントの結果の表示」を参照してください。

2 期待結果を表示します。

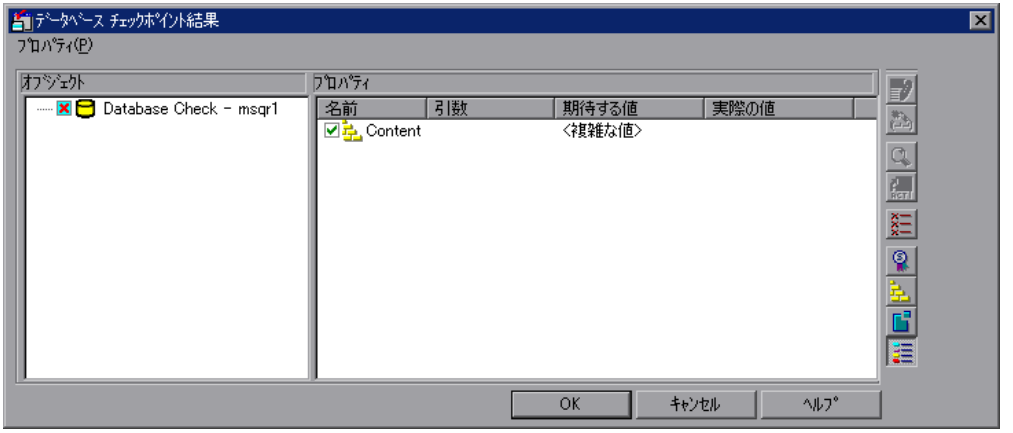
- 統一レポート・ビューでは，［**開く**］ボタンをクリックします。または，［**ファイル**］＞［**開く**］を選択します。［テスト結果を開く］ダイアログ・ボックスが開きます。［**exp** ］を選択し，［**開く**］をクリックします。
- WinRunner レポート・ビューでは，［結果の場所］ボックスで［**exp**］を選択します。

テストの「期待」結果を表示しているので，実行されたデータベース・チェックポイントの総数はゼロになります。

3 期待結果を表示します。

- 統一レポート・ビューでは，分析するデータベース検査の結果ツリー・ノードをクリックします。チェックポイントに関する基本的な詳細が［**イベント サマリ**］表示枠に表示されます。［イベント サマリ］表示枠で，［**イベント詳細を表示する**］リンクをクリックします。
- WinRunner レポート・ビューでは，テスト・ログにあるテーブル検査の［**GUI キャプチャ終了**］エントリをダブルクリックします。または，エントリを強調表示して，［**オプション**］＞［**表示**］を選択するか［**表示**］ボタンをクリックします。

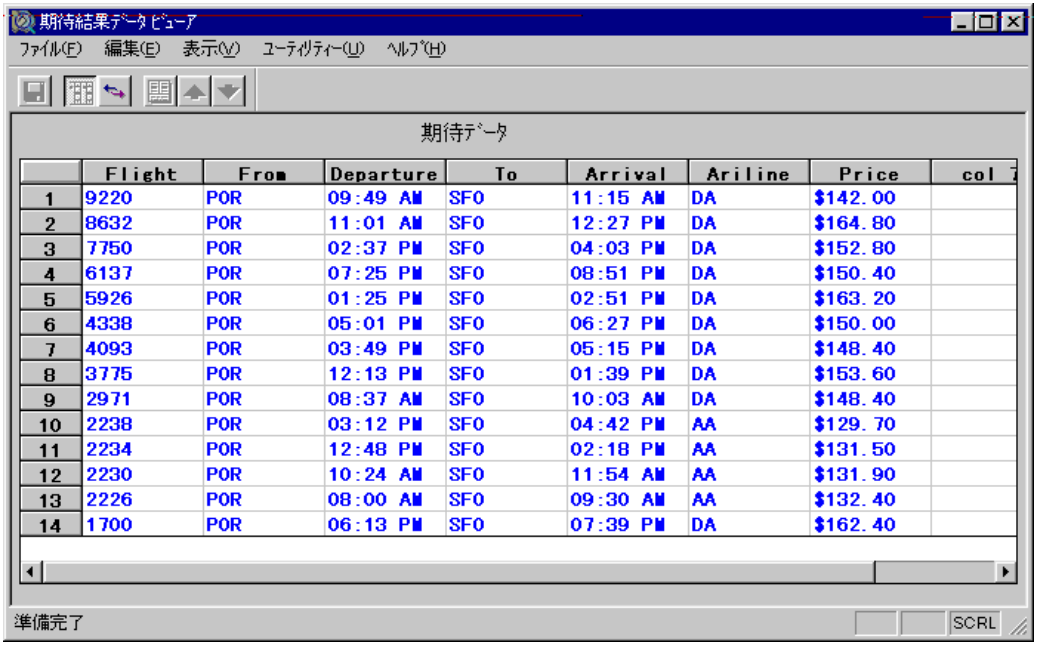
［データベース チェックポイント結果］ダイアログ・ボックスが開き，選択したデータベース・チェックポイントの期待結果が表示されます。



データベース・チェックポイントの「期待」結果を表示しているので，「実際」の値は表示されません。

4 データベース内容検査を強調表示して［**表示**］ボタンをクリックするか，データベース内容検査をダブルクリックします。

期待結果データ・ビューアが開き，期待結果が表示されます。



	Flight	From	Departure	To	Arrival	Ariline	Price	col 7
1	9220	POR	09:49 AM	SFO	11:15 AM	DA	$142.00	
2	8632	POR	11:01 AM	SFO	12:27 PM	DA	$164.80	
3	7750	POR	02:37 PM	SFO	04:03 PM	DA	$152.80	
4	6137	POR	07:25 PM	SFO	08:51 PM	DA	$150.40	
5	5926	POR	01:25 PM	SFO	02:51 PM	DA	$163.20	
6	4338	POR	05:01 PM	SFO	06:27 PM	DA	$150.00	
7	4093	POR	03:49 PM	SFO	05:15 PM	DA	$148.40	
8	3775	POR	12:13 PM	SFO	01:39 PM	DA	$153.60	
9	2971	POR	08:37 AM	SFO	10:03 AM	DA	$148.40	
10	2238	POR	03:12 PM	SFO	04:42 PM	AA	$129.70	
11	2234	POR	12:48 PM	SFO	02:18 PM	AA	$131.50	
12	2230	POR	10:24 AM	SFO	11:54 AM	AA	$131.90	
13	2226	POR	08:00 AM	SFO	09:30 AM	AA	$132.40	
14	1700	POR	06:13 PM	SFO	07:39 PM	DA	$162.40	

注：WinRunner レポート・ビューで作業しているときは，［データベース検証結果］ダイアログ・ボックスから［チェックの編集］ダイアログ・ボックスを開いてデータを編集できます。［チェックの編集］ダイアログ・ボックスを開くには，**Content** 検査を強調表示して，［**期待結果値の編集**］ボタンをクリックします。［チェックの編集］ダイアログ・ボックスでの作業の詳細については，298 ページ「［チェックの編集］ダイアログ・ボックスについて」を参照してください。

5 ［**ファイル**］＞［**閉じる**］を選択して期待結果データ・ビューアを閉じます。

WinRunner レポート・ビューでのチェックポイントの期待結果の更新

実際のデータが正確であるものの期待データが間違っているために，ビットマップ，GUI，またはデータベース・チェックポイントが失敗する場合は，WinRunner レポート・ビューを使用して期待結果フォルダ（**exp**）内のデータを更新できます。

GUI チェックポイントとデータベース・チェックポイントに対しては，チェックポイント全体の結果を更新することも，チェックポイント内の特定の検査の結果を更新することもできます。

チェックポイント全体に対する期待結果を更新するには，次の手順を実行します。

1 ［テスト結果］ウィンドウの WinRunner レポート・ビューのテスト・ログで，不一致となったチェックポイント・エントリを強調表示します。

2 ［**オプション**］＞［**更新**］を選択するか，［**更新**］ボタンをクリックします。

3 ダイアログ・ボックスに「結果を更新すると元に戻せません。それでも続けますか？」という内容の警告が表示されます。［**はい**］をクリックして結果を更新します。

チェックポイント内の特定の検査に対する期待値を更新するには，次の手順を実行します。

1 ［テスト結果］ウィンドウの WinRunner レポート・ビューのテスト・ログで，チェックポイントのエントリをダブルクリックし，［**オプション**］＞［**表示**］を選択するか［**表示**］ボタンをリックします。

関連するダイアログ・ボックスが開きます。

2 ［**プロパティ**］表示枠で，失敗した検査を強調表示します。

3 ［**期待値の更新**］ボタンをクリックします。

4 ダイアログ・ボックスに「期待値を実際値で置換すると，保存されている期待値が上書きされます。続けますか？」という警告が表示されます。［**はい**］をクリックして結果を更新します。

5 ［**OK**］をクリックし，ダイアログ・ボックスを閉じます。

ファイルの比較の結果の表示

テスト・スクリプトで，**file_compare** ステートメントを使用して 2 つのファイルの内容を比較した場合は，WDiff ユーティリティを使用して結果を表示できます。このユーティリティは［テスト結果］ウィンドウからアクセスできます。

ファイルの比較の結果を表示するには，次の手順を実行します。

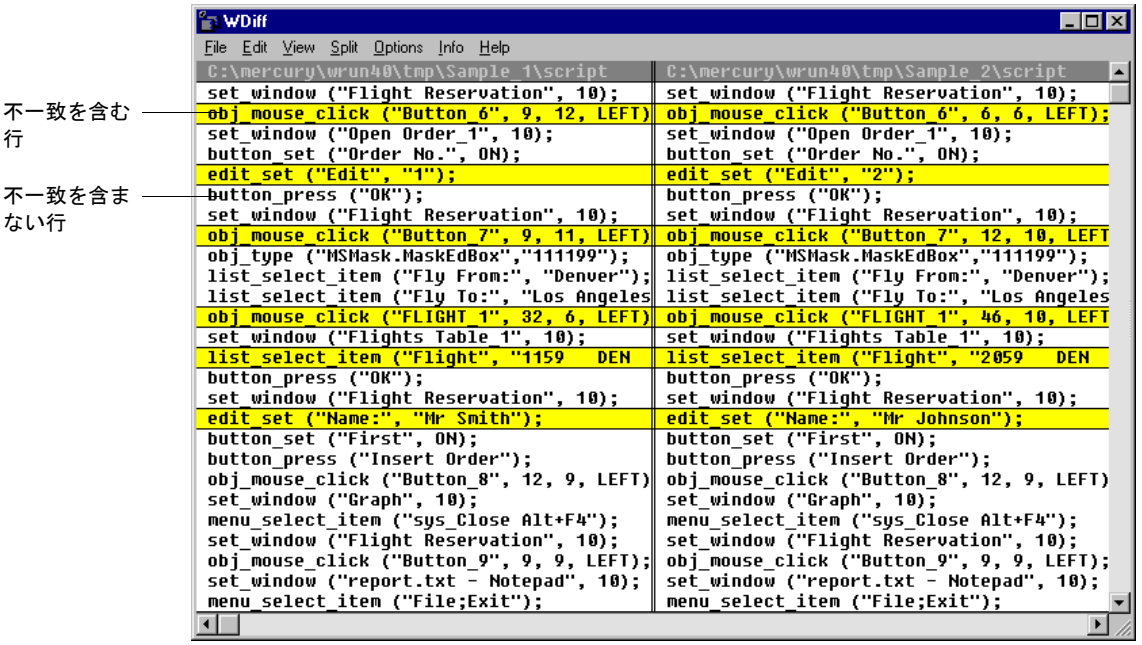
1 ［テスト結果］ウィンドウを開き，ファイル比較結果を表示するテストを表示します。詳細については，469 ページ「チェックポイントの結果の表示」を参照してください。

2 ファイルの比較を表示します。

➤ 統一レポート・ビューでは，分析する file_compare イベントの結果ツリー・ノードをクリックします。チェックポイントに関する基本的な詳細が［**イベント サマリ**］表示枠に表示されます。［イベント サマリ］表示枠で，［**イベント詳細を表示する**］リンクをクリックします。

➤ WinRunner レポート・ビューでは，テスト・ログの **file compare** イベントをダブルクリックします。または，イベントを強調表示して，［**オプション**］＞［**表示**］を選択するか［**表示**］ボタンをクリックします。

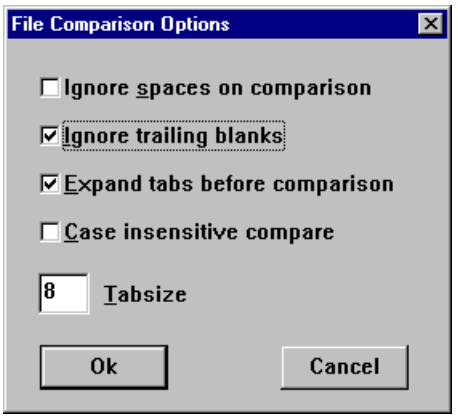
WDiff ユーティリティ・ウィンドウが開きます。



WDiff ユーティリティでは，比較する両方のファイルが表示されます。不一致を含むファイルの行は強調表示されます。**file_compare** ステートメントの先頭のパラメータで定義されたファイルはウィンドウの左側に表示されます。

- ファイル内の次の不一致を表示するには，［**View**］＞［**Next Diff**］を選択するか，Tab キーを押します。強調表示されている次の行までウィンドウがスクロールします。前の差異を表示するには，［**View**］＞［**Prev Diff**］を選択するか，Backspace キーを押します。
- 不一致を含むファイルの行だけを表示するように選択できます。ファイルの比較結果を絞り込むには，［**Option**］＞［**View**］＞［**Hide Matching Areas**］を選択します。両方のファイルについて，強調表示された部分だけがウィンドウに表示されます。

- 実際の結果と期待結果の比較方法を変更するには，［**Option**］＞［**File Comparison**］を選択します。［File Comparison］ダイアログ・ボックスが開きます。



どのオプションを変更したときも，2 つのファイルは再度読み込まれ，比較されます。

- ［**Ignore spaces on comparison**］：比較の際，タブの桁や空白を無視します。
- ［**Ignore trailing blanks**］**（標準）**：比較の際，行の最後にある空白を無視します。
- ［**Expand tabs before comparison**］**（標準）**：テキスト内のタブ文字（16 進法の 09）による空白が，次のタブ・ストップに届くまで増やされます。タブ・ストップ間の空白数は，**Tabsize** パラメータで指定します。［**Ignore spaces on comparison**］オプションが選択されている場合は，この［**Expand tabs before comparison**］オプションは無視されます。
- ［**Case insensitive compare**］：ファイル比較の際，大文字と小文字を区別しません。
- ［**Tabsize**］：タブサイズ（タブ・ストップ間の空白数）は，1 ～ 19 の間で選択します。標準のサイズは 8 です。［**Expand tabs before comparison**］オプションも設定している場合は，ファイル比較時にこのオプションが優先されます。タブは常に，指定する空白数だけ増やされます。

3 ［**File**］＞［**Exit**］を選択し，WDiff ユーティリティを閉じます。

テスト実行中に検出された不具合の報告

ソフトウェアの不具合を効率よく探し出し，修正することは，開発プロセスにとって非常に重要です。ソフトウェア開発者，テスト担当者，およびエンド・ユーザは，テスト・プロセスのどの段階でも不具合を検出し，不具合プロジェクトに追加することが可能です。Mercury の Quality Center の［不具合の追加］ダイアログ・ボックスを使用すると，アプリケーションの設計上の問題点を報告し，不具合レポートから得られるデータを追跡できます。

例えば，フライト予約アプリケーションをテストするとします。航空券を発注しようとするときにエラーが発生することがわかりました。その時点で，不具合を開いて報告することができます。これには，不具合のサマリ説明と詳細説明，不具合が発見された場所，および不具合の再現可能性が含まれます。レポートには，スクリーン・キャプチャ，Web ページ，テキスト・ドキュメントなど，問題の把握と解決に必要な関連ファイルを含めることもできます。

テスト実行によってテスト中のアプリケーションで不具合が検出された場合，［テスト結果］ウィンドウから直接，不具合を報告できます（Quality Center プロジェクト接続時）。

［テスト結果］ウィンドウから不具合を報告する際，テストに関する基本的な情報と，選択されているチェックポイント（該当する場合）に関する基本的な情報が，不具合の説明に自動的に追加されます（［不具合の追加］オプションは TestDirector 7.2 または Quality Center で作業しているときのみサポートされます）。

［不具合の追加］ダイアログ・ボックスの使用

［不具合の追加］ダイアログ・ボックスは Quality Center の不具合追跡コンポーネントです。これは，Mercury の Web ベースのテスト管理ツールです。アプリケーションの不具合を，Quality Center プロジェクトに直接報告できます。また，追加した不具合について，アプリケーションの開発者やソフトウェアのテスト担当者が解決したと判断するまで，状況を追跡することもできます。

［不具合の追加］ダイアログ・ボックスの設定

［不具合の追加］ダイアログ・ボックスを起動するには，あらかじめ Test Director 7.2 または Quality Center をインストールし，WinRunner を Quality Center サーバおよびプロジェクトに接続しておく必要があります。接続プロセスには 2 つの段階があります。最初に，WinRunner をサーバに接続します。このサーバによって，WinRunner と Quality Center プロジェクトの間の接続が処理されます。次に，WinRunner からアクセスするプロジェクトを選択します。このプロ

ジェクトには，テスト，テスト実行情報，および，テスト対象のアプリケーションの不具合情報が格納されます。WinRunner から Quality Center への接続の詳細については，467 ページ「[テスト結果］ウィンドウから Quality Center への接続」を参照してください。

Quality Center のインストールの詳細については，**Mercury** の『**Quality Center インストール・ガイド**』を参照してください。

［不具合の追加］ダイアログ・ボックスを使用した不具合の報告

Quality Center に接続しているときは，アプリケーションで検出された不具合を WinRunner の［テスト結果］ウィンドウから直接報告できます。

［不具合の追加］ダイアログ・ボックスを使用して不具合を報告するには，次の手順を実行します。

1 WinRunner レポート・ビューで作業している場合は，WinRunner のメイン・ウィンドウから Quality Center に接続します。詳細については，467 ページ「[テスト結果］ウィンドウから Quality Center への接続」を参照してください。

統一レポート・ビューで作業している場合は，手順 4 に従い，［テスト結果］ウィンドウから Quality Center に直接接続できます。

2 ［テスト結果］ウィンドウを開き，追加する不具合のあるテストを表示します。詳細については，469 ページ「チェックポイントの結果の表示」を参照してください。

3 可能であれば，［テスト結果］の中で，報告する不具合に対応した行を選択します。

4 ［不具合の追加］ダイアログ・ボックスを開きます。

➤ 統一レポート・ビューでは，［**不具合の追加**］ボタンをクリックします。または，［**ツール**］＞［**不具合の追加**］を選択します。まだ［テスト結果］ウィンドウから Quality Center に接続していない場合は，［Quality Center に接続］ダイアログ・ボックスが開きます。467 ページ「[テスト結果］ウィンドウから Quality Center への接続」で説明した手順に従い，Quality Center に接続します。接続が完了したら，［**閉じる**］をクリックします。すると，［Quality Center への接続］ダイアログ・ボックスが閉じ，［不具合の追加］ダイアログ・ボックスが開きます。

➤ WinRunner レポート・ビューでは，［**不具合報告**］ボタンをクリックします。または，［**ツール**］＞［**不具合レポート**］を選択します。

［不具合の追加］ダイアログ・ボックスが開きます。［テスト結果］の中で選択されている行に関する情報が，説明に含まれます。

5 不具合の簡単な説明を［**サマリ**］に入力します。

6 残りの不具合テキスト・ボックスに，必要に応じて情報を入力します。赤いラベルの付いたテキスト・ボックスの情報はすべて入力する必要があります。

7 不具合の詳細な説明を［**詳細**］ボックスに入力します。

［不具合の追加］ダイアログ・ボックスのデータをクリアするには，［**クリア**］ボタンをクリックします。

8 不具合レポートに添付ファイルを追加することができます。

- テキスト・ファイルを添付するには，［**ファイルの添付**］ボタンをクリックします。
- Web ページを添付するには，［**URL の添付**］ボタンをクリックします。
- 画像をキャプチャして添付するには，［**画面キャプチャの添付**］ボタンをクリックします。

9 対象の不具合と，Quality Center プロジェクト内の既存の不具合とを比較するには，［**類似した不具合を検索**］をクリックします。これを使用すると，類似した不具合レコードがすでに存在するかどうかを確かめることができ，重複を避けるのに役立ちます。類似した不具合が見つかった場合は，［類似した不具合］ダイアログ・ボックスに表示されます。

10 不具合をデータベースに追加するには，［**送信**］ボタンをクリックします。Quality Center で，新しい不具合に不具合 ID が割り当てられます。

11 ［**閉じる**］をクリックします。

［不具合の追加］ダイアログ・ボックスの使用の詳細については，『**Mercury Quality Center ユーザーズ・ガイド**』を参照してください。

テスト実行中の不具合の報告

qcdb_add_defect ステートメントをテストに挿入すると，テスト・スクリプトで定義した条件に基づいて不具合を Quality Center プロジェクトに追加するよう，WinRunner を設定できます。ステートメントにはサマリ・フィールドや詳細フィールドのデータを含めることができるほか，他のフィールド名や値を指定することもできます。

例えば，テストの最初にフライト予約アプリケーションにログインするとします。ログインに失敗した場合に，不具合のサマリおよび詳細のほか，［**Detected by**］フィールドや［**Assigned to**］フィールドの値を示す，不具合を報告できます。

qcdb_add_defect ステートメントを挿入するときは次の構文を使用します。

qcdb_add_defect (＜サマリ＞ , ＜詳細＞ , ＜各不具合フィールド＞);

不具合フィールドを入力するときは，次の形式を使用します。**" ＜フィールド名 1＞ = ＜値 1＞ ; ＜フィールド名 2＞ = ＜値 2＞ ; ＜フィールド名 N＞ = ＜値 N＞ "**

フィールド・ラベルではなく必ず**フィールド名**を入力してください。例えば，フィールド・ラベル **Detected By** の場合は，そのフィールド名 **BG_DETECTED_BY** を使用します。詳細については，Quality Center のマニュアルを参照してください。

テストに **qcdb_add_defect** ステートメントが含まれている場合は，テストを実行する前に，適切な Quality Center プロジェクトに接続していることを確認してください。

第 5 部

基本設定

第 21 章

単独のテストのプロパティの設定

［テストのプロパティ］ダイアログ・ボックスを使用して，単独のテストのプロパティを設定できます。テストのプロパティの設定は，WinRunner のテストに関する情報を格納し，WinRunner によるテストの実行方法を制御するために行います。

本章では，次の項目について説明します。

- 単独のテストのプロパティの設定について
- ［テストのプロパティ］ダイアログ・ボックスからのテストのプロパティの設定
- テストの一般情報の文書化
- テストの説明情報の文書化
- テスト・パラメータの管理
- テストへのアドインの関連付け
- 現在のテスト設定の確認
- 起動アプリケーションおよび起動関数の定義

単独のテストのプロパティの設定について

特定のテストに関する情報を文書化するためにテスト プロパティを設定したり，特定のテストのオプションを指定するテスト プロパティを設定したりできます。例えば，テストの詳細説明を入力したり，テストに必要なアドインを指定したりできます。

すべてのテストに影響を与えるテスト・オプションも設定できます。詳細については，第 22 章「グローバル・テスト・オプションの設定」を参照してください。

[テストのプロパティ] ダイアログ・ボックスからのテストのプロパティの設定

[テストのプロパティ] ダイアログ・ボックスでは，開いている任意のテストについてテスト固有のプロパティを設定できます。

テストのプロパティを設定するには，次の手順を実行します。

1 [**ファイル**] > [**テストのプロパティ**] を選択します。

[テストのプロパティ] ダイアログ・ボックスが開きます。このダイアログ・ボックスは，内容ごとに 6 つのタブに分かれています。

2 テストのプロパティを設定するには，以降の各節の説明に従い，該当するタブを選択してオプションを設定します。

3 変更を適用し，［テストのプロパティ］ダイアログ・ボックスを開いたままにする場合は，［**適用**］をクリックします。

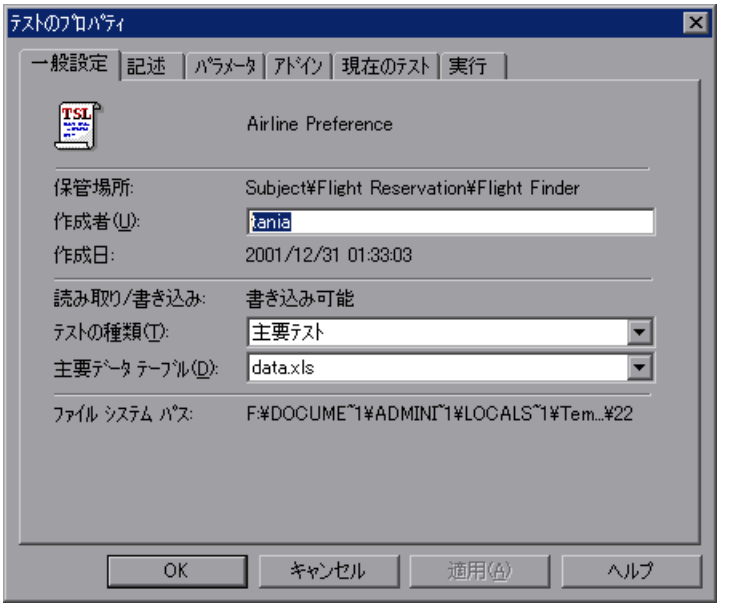
4 終わったら［**OK**］をクリックし，変更を保存してダイアログ・ボックスを閉じます。

［テストのプロパティ］ダイアログ・ボックスには，次のタブがあります。

タブ	説明
［一般設定］	テストに関する一般情報を設定できます。
［説明］	テストに関する説明情報を入力できます。
［パラメータ］	入力および出力のテスト・パラメータを定義できます。
［アドイン］	テストに必要なアドインを指定できます。
［現在のテスト］	テストに対する現在のフォルダと実行モードの設定を表示できます。
［実行］	起動アプリケーションおよび関数を定義できます。

テストの一般情報の文書化

［テストのプロパティ］ダイアログ・ボックスの［**一般設定**］タブでは，テストに関する一般情報を文書化および表示できます。例えば，テストの作成者名を入力したり，テストがメイン・テストかコンパイル済みモジュールかを選択したりできます。また，テストの入力データとして使用する Microsoft Excel ファイルを指定したり，他のサマリ情報を表示したりすることもできます。



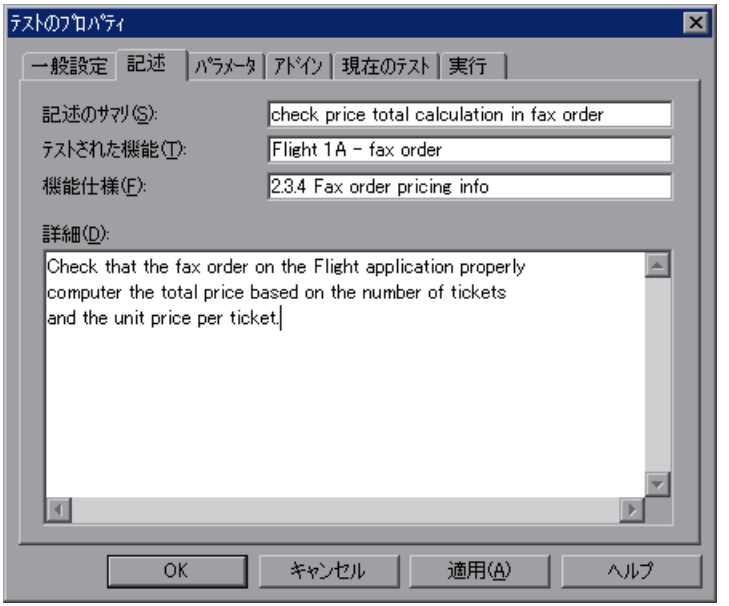
このタブには以下の情報が表示されます。

オプション	説明
	テストの名前を表示します。
［保管場所］	ファイル・システム内部または Quality Center ツリー内のテストの場所を表示します。
［作成者］	テストの作成者名を指定できます。
［作成日］	テストが作成された日時を表示します。

オプション	説明
[読み取り / 書き込み]	テストが読み取り専用か（テスト・フォルダかスクリプトのどちらかがファイル・システムで読み取り専用としてマークされているか），または書き込み可能かを示します。テストが読み取り専用の場合，［テストのプロパティ］ダイアログ・ボックスにある編集可能なプロパティ・フィールドはすべて無効になります。
[テストの種類]	テストが［**メイン テスト**］（標準のテスト）か［**コンパイル済みモジュール**］かを示します。コンパイル済みモジュールの詳細については，724 ページ「コンパイル済みモジュールの作成」を参照してください。
[主要データ テーブル]	テスト用のメインのデータ・テーブルを示します。詳細については，389 ページ「テストへのメイン・データ・テーブルの割り当て」を参照してください。
[ファイル システム パス]	テストのシステム・ファイル・パスを表示します。この情報は，Quality Center に接続していて現在のテストを Quality Center プロジェクトから開いているときのみ表示されます。
[バージョン コントロール]	テストのバージョン・コントロール情報を表示します。この情報は，バージョン・コントロールをサポートしている Quality Center プロジェクトに接続しているときのみ表示されます。

テストの説明情報の文書化

［テストのプロパティ］ダイアログ・ボックスの［**記述**］タブでは，テストに関する説明情報を文書化できます。テストのサマリ説明，テスト対象のアプリケーション機能（1 つ以上），関連する機能仕様ドキュメントへの参照（1 つ以上）を入力できるほか，テストの目的，内容，または要件に関する詳細も入力できます。

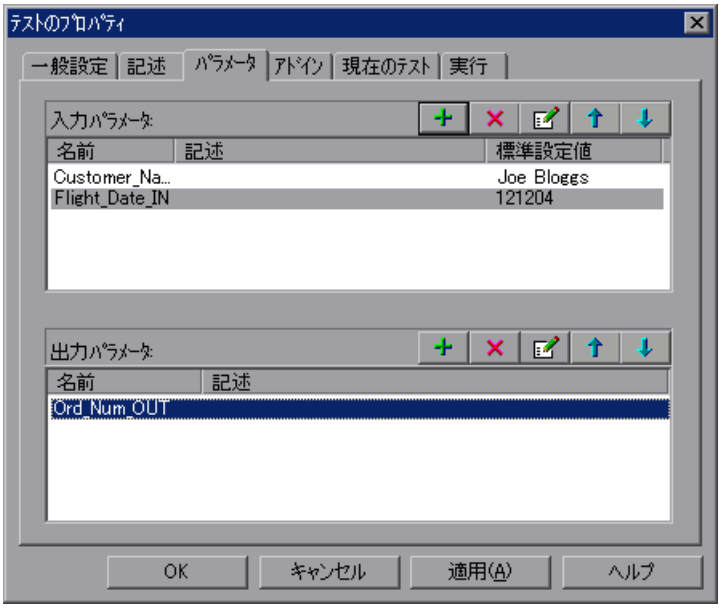


このタブには以下の情報が表示されます。

オプション	説明
［記述のサマリ］	テストの簡単なサマリを指定できます。
［テストされた機能］	テスト対象のアプリケーション機能の説明を指定できます。
［機能仕様］	テスト対象とする，アプリケーションの機能の機能仕様（1 つ以上）への参照を指定できます。
［詳細］	テストの詳細説明を入力できます。

テスト・パラメータの管理

［テストのプロパティ］ダイアログ・ボックスの［**パラメータ**］タブでは，パラメータの追加（宣言），変更，または削除によって，テスト・パラメータを管理できます。



［テスト パラメータ］リストには，既存のテスト・パラメータが表示されます。テストが別のテストから呼び出されると，［**パラメータ**］タブに列挙されている入力パラメータに，呼び出し元のテストで指定された値が割り当てられます。そして，出力パラメータから，現在のテストの中で生成された値が呼び出し元のテストに返されます。

入力パラメータに対して標準設定値を割り当てることができます。入力パラメータの標準設定値は，呼び出し元のテストからそのテスト呼び出しの中で入力パラメータの値が渡されない場合に使用されます。

呼び出し元のテストから入力パラメータ値を受け取り，呼び出し元のテストに出力パラメータを返すためには，このダイアログ・ボックスでテスト・パラメータを宣言する必要があります。このタブで列挙するパラメータの順序によって，呼び出し元のテストでパラメータを指定する順序が決まります。テスト呼び出しの中では，入力パラメータが出力パラメータよりも前に出現します。

ヒント：他の WinRunner テストまたは他の Mercury 製品によってすでに呼び出されているテストについて，そのパラメータの追加，削除，または順序の変更を行う場合は，必ず呼び出し元のテストまたは製品の中でパラメータを適切に調整してください。

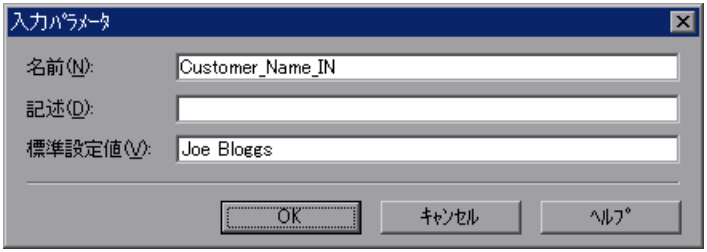
注：テスト・パラメータは［メイン テスト］タイプのテストでのみ使用されます。コンパイル済みモジュールでは使用されません。

パラメータの詳細については，700 ページ「テスト・パラメータを使った作業」を参照してください。

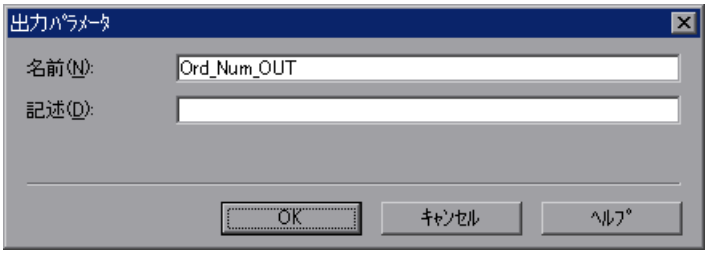
新しい入力パラメータまたは出力パラメータを定義するには，次の手順を実行します。

1 ［テストのプロパティ］ダイアログ・ボックスの［**パラメータ**］タブで，パラメータを追加する対象となるパラメータ・リスト（［**入力**］または［**出力**］）に対応する［**追加**］ボタン をクリックします。

［入力パラメータ］ダイアログ・ボックスまたは［出力パラメータ］ダイアログ・ボックスが開きます。入力パラメータの場合，ダイアログ・ボックスには［**標準設定値**］を入力するためのテキスト・ボックスがあります。



出力パラメータの場合，ダイアログ・ボックスには［**標準設定値**］テキスト・ボックスがありません。



2 パラメータの［**名前**］と［**記述**］を入力します。入力パラメータの場合，パラメータの［**標準設定値**］を指定できます。

ヒント：パラメータ名では，パラメータのタイプを示す IN または OUT という接頭辞または接尾辞を使用することをお勧めします。こうすることでテストが読みやすくなり，他のテスト設計者がテストへの呼び出しステートメントの中で何を入力するべきか判断しやすくなります。

---

3 ［**OK**］をクリックします。該当する［**テスト パラメータ**］リストにパラメータが追加されます。

4 パラメータの順序を変更するには，［**上**］および［**下**］矢印ボタンを使用します。

注：パラメータ値は順に割り当てられるため，［パラメータ］タブでのパラメータの列挙順序によって，呼び出し元のテストでパラメータに割り当てられる値が決まります。テスト呼び出しの中では，入力パラメータが常に出力パラメータよりも前に出現します。

---

5 ［**OK**］をクリックし，ダイアログ・ボックスを閉じます。

パラメータをパラメータ・リストから削除するには，次の手順を実行します。

1 ［テストのプロパティ］ダイアログ・ボックスの［**パラメータ**］タブで，削除するパラメータの名前を選択します。

2 削除するパラメータ・タイプに対応した［**削除**］ボタン をクリックします。

3 ［**OK**］をクリックし，ダイアログ・ボックスを閉じます。

パラメータ・リスト内のパラメータを変更するには，次の手順を実行します。

1 ［テストのプロパティ］ダイアログ・ボックスの［**パラメータ**］タブで，変更するパラメータの名前を選択します。

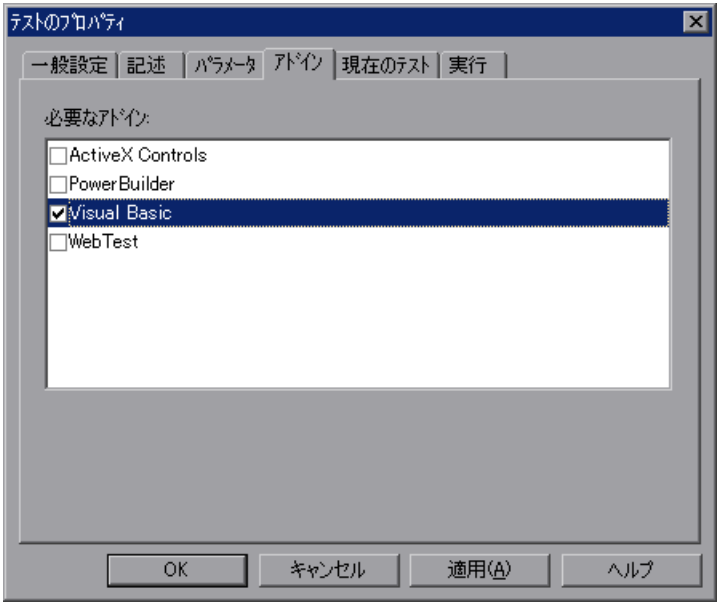
2 ［**パラメータの変更**］ボタン をクリックします。または，パラメータ名をダブルクリックします。［パラメータのプロパティ］ダイアログ・ボックスが開き，パラメータの現在の名前と説明が表示されます。

3 必要に応じてパラメータを編集します。

4 ［**OK**］をクリックし，ダイアログ・ボックスを閉じます。変更されたパラメータが［**テストのパラメータ**］リストに表示されます。

テストへのアドインの関連付け

［テストのプロパティ］ダイアログ・ボックスの［**アドイン**］タブでアドインを選択することにより，テストに必要な WinRunner アドインを指定できます。



［**アドイン**］タブには，現在インストールされているアドインごとに，対応するチェック・ボックスが 1 つずつあります。新しいテストの作成を始めると，その時点で読み込み済みになっているアドインが必要なアドインとして自動的に選択されます。選択されているチェック・ボックスを変更することで，どのアドインをテストで実際に必要とするのかを指定できます。この情報をもとに，自分や他のユーザはこのテストを正常に実行するためにどのアドインを読み込む必要があるのかを判断できます。また，Quality Center はこの情報をもとに，必要なアドインが読み込まれていることを確認します。詳細については，507 ページ「Quality Center からのアドインを使用したテストの実行」を参照してください。

注：どのアドインが読み込まれているかは，［**WinRunner のバージョン情報**］ダイアログ・ボックス（［**ヘルプ**］＞［**バージョン情報**］）でいつでも確認できます。読み込まれているアドインには「**+**」という印が付きます。

アドインをテストに関連付けるには，次の手順を実行します。

1 ［**ファイル**］＞［**テストのプロパティ**］を選択し，［**テストのプロパティ**］ダイアログ・ボックスを表示します。

2 ［**アドイン**］タブをクリックします。

3 テストに必要なアドイン（1 つ以上）を選択します。

Quality Center からのアドインを使用したテストの実行

［**アドイン**］タブは，WinRunner からテストを実行しているユーザに情報を提供するだけでなく，選択されたアドインを WinRunner テストの実行時に読み込むよう Quality Center に指示する役目も果たします。

Quality Center からテストを実行すると，そのテスト用として［**アドイン**］タブで選択されているアドインが Quality Center によって読み込まれます。WinRunner がすでに開いているものの，必要なアドインが読み込まれていない場合は，Quality Center によって WinRunner がいったん閉じられて再び開き，適切なアドインが読み込まれます。必要なアドインが 1 つ以上インストールされていない場合は，Quality Center で **「テストを開くことができません」**というエラー・メッセージが表示されます。

Quality Center からの WinRunner テストの実行に関する詳細については，『**Mercury Quality Center ユーザーズ・ガイド**』を参照してください。

現在のテスト設定の確認

現在のテストに対するフォルダおよび実行モードの情報は，［テストのプロパティ］ダイアログ・ボックスの［**現在のテスト**］タブに表示される読み取り専用ビューで確認できます。

［現在の行番号］

このボックスには，テスト・スクリプト内での実行矢印の現在位置に対応した行番号が表示されます。

getvar 関数を使用して，対応する **line_no** テスト・オプションの値をテスト・スクリプトの中から取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

[現在のフォルダ]

このボックスには，テスト用の現在の作業フォルダが表示されます。

getvar 関数を使用して，対応する **curr_dir** テスト・オプションの値をテスト・スクリプトの中から取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

[期待結果フォルダ]

このボックスには，現在のテスト実行に関連付けられている期待結果フォルダのフル・パスが表示されます。

getvar 関数を使用して，対応する **exp** テスト・オプションの値をテスト・スクリプトの中から取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

このオプションは対応する **-exp** コマンド・ライン・オプションを使用して設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

[検証結果フォルダ]

このボックスには，現在のテスト実行に関連付けられている検証結果フォルダのフル・パスが表示されます。

getvar 関数を使用して，対応する **result** テスト・オプションの値をテスト・スクリプトの中から取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

[実行モード]

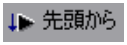
このボックスには，現在の実行モード（検証，デバッグ，または更新）が表示されます。

getvar 関数を使用して，対応する **runmode** テスト・オプションの値をテスト・スクリプトの中から取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

起動アプリケーションおよび起動関数の定義

「起動」アプリケーションおよび関数とは，テストの実行前に WinRunner によって実行されるアプリケーションおよび関数のことです。例えば，航空券予約アプリケーションを起動アプリケーションとして設定し，テストの実行前にその航空券予約アプリケーションにログインする起動関数を定義できます。

起動アプリケーションと起動関数は，［テストのプロパティ］ダイアログ・ボックスの［**実行**］タブで定義します。起動アプリケーションと起動関数のオプションは，テストの作成中に定義できます。また，起動アプリケーションや起動関数の定義を変更せず，テストを実行する前に起動アプリケーションや起動関数を実行するかどうかを選択することもできます。



WinRunner では，［**先頭から実行**］を選択したときや［**最小化の状態で実行する**］＞［**先頭から**］を選択したときなど，テストを始めから実行したときのみ，［**実行**］タブの設定が適用されます。これらのオプションの詳細については，427 ページ「WinRunner の実行コマンド」を参照してください。

WinRunner では，呼び出し先のテストの［**実行**］タブの設定は，［**ステップ イントゥ**］を使用して呼び出し先のテストを開かない限り，呼び出し先のテストが実行された時点で適用されます。テストの呼び出しの詳細については，第 29 章「テストの呼び出し」を参照してください。［**ステップ イントゥ**］オプションの詳細については，第 38 章「テスト実行の制御」を参照してください。

注 : テストの開始前にアプリケーションと関数を実行することを選択した場合は，起動アプリケーションが起動関数よりも先に実行されます。

起動アプリケーションの定義

起動アプリケーションを定義するときは，アプリケーションへのパス，必要なすべてのパラメータ，および，アプリケーションを起動してからテストを実行するまでに WinRunner が待機する時間の長さを指定します。

このほか，アプリケーションを実行するための以下の方法があります。

➤ **invoke_application** 関数を使用して，テスト・スクリプトの中からいつでもアプリケーションを実行できます。この方法は，テストの実行中にアプリケーショ

ンを実行する場合に使用します。詳細については，686 ページ「テスト・スクリプトからのアプリケーションの起動」を参照してください。

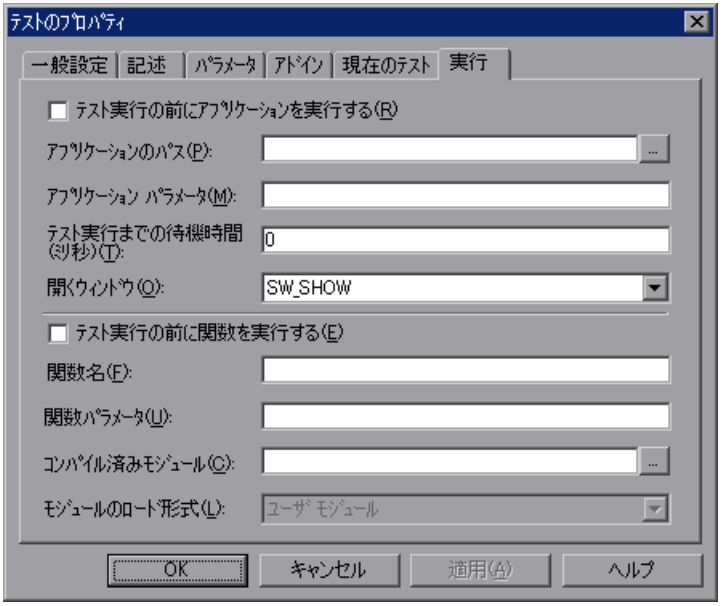
➤ WinRunner を実行するときに，コマンド・ラインからアプリケーションを実行できます。この方法は，WinRunner の起動前にアプリケーションを実行する場合に使用します。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

注：起動アプリケーションとして指定したアプリケーションがテストの実行時にすでに実行されている場合，テストの最初でアプリケーションの新しいインスタンスが開くことはありません。

起動アプリケーションを定義するには，次の手順を実行します。

1 ［**ファイル**］＞［**テストのプロパティ**］を選択し，［**テストのプロパティ**］ダイアログ・ボックスを表示します。

2 ［**実行**］タブをクリックします。



3 次回のテスト実行で起動アプリケーションを実行する場合は，［**テスト実行の前にアプリケーションを実行する**］チェック・ボックスを選択します。

4 ［**アプリケーションのパス**］ボックスに，アプリケーション・パスを入力するか，または参照ボタン  を使用して実行するアプリケーションのある場所に移動します。アプリケーションのフル・パスを入力します。引用符は使用しないでください。

指定できるのは **.exe** ファイルと **.com** ファイルのみです。別の拡張子を持つファイルを実行する必要がある場合は，そのファイルを扱う **.exe** または **.com** アプリケーションを［**アプリケーションのパス**］ボックスで指定します。そして，ファイル名を［**アプリケーション パラメータ**］ボックスで指定します。

例えば，**.htm** ファイルを実行する必要があるとします。この場合は，ブラウザのパスを［**アプリケーションのパス**］ボックスに入力します。例えば，C:¥Program Files¥Internet Explorer¥IEXPLORE.EXE などと入力します。そして，**.htm** ファイルのフル・パスを［**アプリケーション パラメータ**］ボックスに入力します。

5 必要なすべてのアプリケーション・パラメータを［**アプリケーション パラメータ**］ボックスに，カンマ（,）で区切って入力します。［**アプリケーション パラメータ**］ボックス内のテキストには引用符を含めることもできます。アプリケーション・パラメータの詳細については，アプリケーションのマニュアルを参照してください。

6 ［**テスト実行までの待機時間（ミリ秒）**］ボックスに，アプリケーションを起動してからテストを実行するまでにシステムが待機する時間の長さを入力します。または，標準設定値（0 ミリ秒）を受け入れます。

7 ［**開くウィンドウ**］ボックスで，アプリケーション・ウィンドウを開く方法を選択します。次のオプションを選択できます。

オプション	説明
SW_HIDE	ウィンドウを非表示にし，別のウィンドウをアクティブにします。
SW_SHOWNORMAL	ウィンドウをアクティブにして表示します。ウィンドウが最大化または最小化されている場合は，Windows によって元のサイズと位置に復元されます。このフラグは，ウィンドウを初めて表示するときに指定します。

オプション	説明
SW_SHOWMINIMIZED	ウィンドウをアクティブにし，最小化されたウィンドウとして表示します。
SW_SHOWMAXIMIZED	ウィンドウをアクティブにし，最大化されたウィンドウとして表示します。
SW_SHOWNOACTIVATE	ウィンドウを直前のサイズと位置で表示します。アクティブなウィンドウはアクティブのままです。
SW_SHOW	ウィンドウをアクティブにし，現在のサイズと位置で表示します。
SW_MINIMIZE	指定されたウィンドウを最小化し，z 順序で次の最上位ウィンドウをアクティブにします。
SW_SHOWMINNOACTIVE	ウィンドウを最小化されたウィンドウとして表示します。アクティブなウィンドウはアクティブのままです。
SW_SHOWNA	ウィンドウを現在の状態で表示します。アクティブなウィンドウはアクティブのままです。
SW_RESTORE	ウィンドウをアクティブにして表示します。ウィンドウが最大化または最小化されている場合は，Windows によって元のサイズと位置に復元されます。このフラグは，最小化されているウィンドウを復元するときに指定します。

注：このオプションは **-app_open_win** コマンド・ライン・オプションを使用して設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

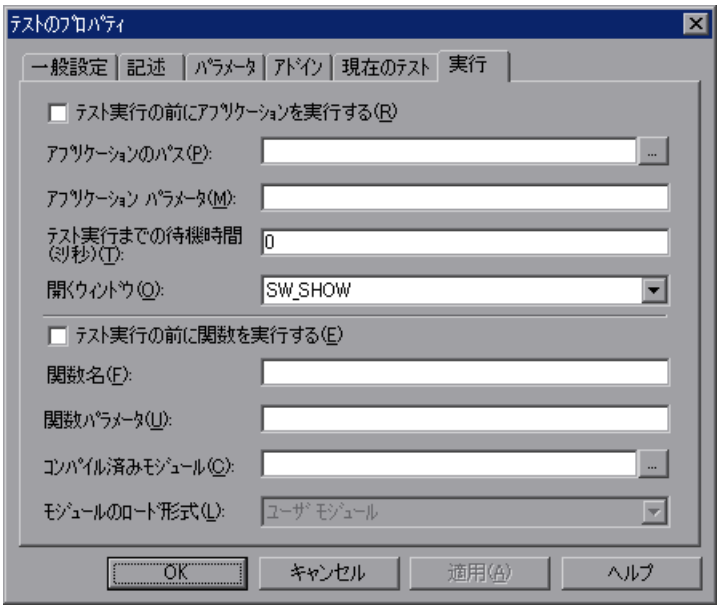
起動関数の定義

起動関数として，TSL 関数またはコンパイル済みモジュールに含まれるユーザ定義関数を使用できます。起動関数を定義するときは，関数の名前，関数パラメータ（必要な場合），およびコンパイル済みモジュールの名前とタイプ（ユーザ定義関数の場合）を指定します。TSL 関数の詳細については，第 34 章「関数の生成」，および「**TSL リファレンス**」を参照してください。ユーザ定義関数とコンパイル済みモジュールの詳細については，第 30 章「ユーザ定義関数の作成」および第 31 章「テストでのユーザ定義関数の利用」を参照してください。

起動関数を定義するには，次の手順を実行します。

1 ［**ファイル**］＞［**テストのプロパティ**］を選択し，［**テストのプロパティ**］ダイアログ・ボックスを表示します。

2 ［**実行**］タブをクリックします。



3 次回のテスト実行で起動関数を実行する場合は，［**テスト実行前に関数を実行する**］チェック・ボックスを選択します。

4 ［**関数名**］ボックスに，関数の名前を入力します。

注：関数名には英数字とアンダースコアのみ使用できます。数字で始めたり，括弧を含めたりすることはできません。

---

5 必要なすべてのパラメータを［**関数パラメータ**］ボックスに入力します。

6 関数がコンパイル済みモジュールの一部である場合は，関数を含んでいるコンパイル済みモジュールの名前を［**コンパイル済みモジュール**］ボックスに入力するか，または参照ボタン を使用してコンパイル済みモジュールのある場所に移動します。

注：呼び出し元のテストとコンパイル済みモジュールの両方が Quality Center に保存されている場合は，関数を呼び出すときにフル・パスを使用する必要があります。

---

7 関数がコンパイル済みモジュールの一部である場合は，コンパイル済みモジュールのタイプを［**モジュールのロード形式**］ボックスで選択します。システム・モジュールとユーザ・モジュールの詳細については，735 ページ「コンパイル済みモジュールのロードとアンロード」を参照してください。

第 22 章

グローバル・テスト・オプションの設定

［一般オプション］ダイアログ・ボックスでグローバル・テスト・オプションを設定することによって，WinRunner がテストをどのように記録して実行するかを制御できます。

本章では，以下の項目について説明します。

- グローバル・テスト・オプションの設定について
- ［一般オプション］ダイアログ・ボックスでのグローバル・テスト・オプションの設定
- 一般オプションの設定
- フォルダ・オプションの設定
- 記録オプションの設定
- テストの実行オプションの設定
- 通知オプションの設定
- 概観オプションの設定
- 適切なタイムアウトと遅延設定の選択

グローバル・テスト・オプションの設定について

WinRunner のテスト・オプションは，テスト・スクリプトの記録とテストの実行方法に影響を与えます。これらのオプションは，WinRunner を開く方法やメイン・ウィンドウの表示方法にも影響を与えます。例えば，WinRunner でテストを実行する速度を設定したり，WinRunner によるキーボード入力の記録方法を決めたり，WinRunner のメイン・ウィンドウの背景スタイルを選択したりできます。

これらやその他のテスト・オプションは，［一般オプション］ダイアログ・ボックスを使用してすべてのテストに対して設定できます。

また，**setvar** 関数および **getvar** 関数を使用して，テストの実行中にオプションの一部を設定および取得することもできます。これらの関数を使用すると，すべてのテスト，単独のテスト，あるいは単独のテストの一部分に対して，テスト・オプションを設定および表示できます。

テスト・スクリプトからのテスト・オプションの設定と取得の詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

［一般オプション］ダイアログ・ボックスでのグローバル・テスト・オプションの設定

テストを記録または実行する前に，［一般オプション］ダイアログ・ボックスを使ってテスト・オプションを変更できます。設定した値は現在のテスト・セッションの中のすべてのテストに適用されます。

テスト・セッションが終わると，WinRunner はテスト・オプションへの変更を WinRunner のコンフィギュレーションに保存するようにプロンプトを出します。これによって，新しい値を次回からのテスト・セッションで使い続けることができます。

［一般オプション］ダイアログ・ボックスは「オプション・ツリー」と「オプション表示枠」で構成されます。オプション・ツリーでカテゴリまたはサブカテゴリをクリックすると，対応するオプションがオプション表示枠に表示されます。

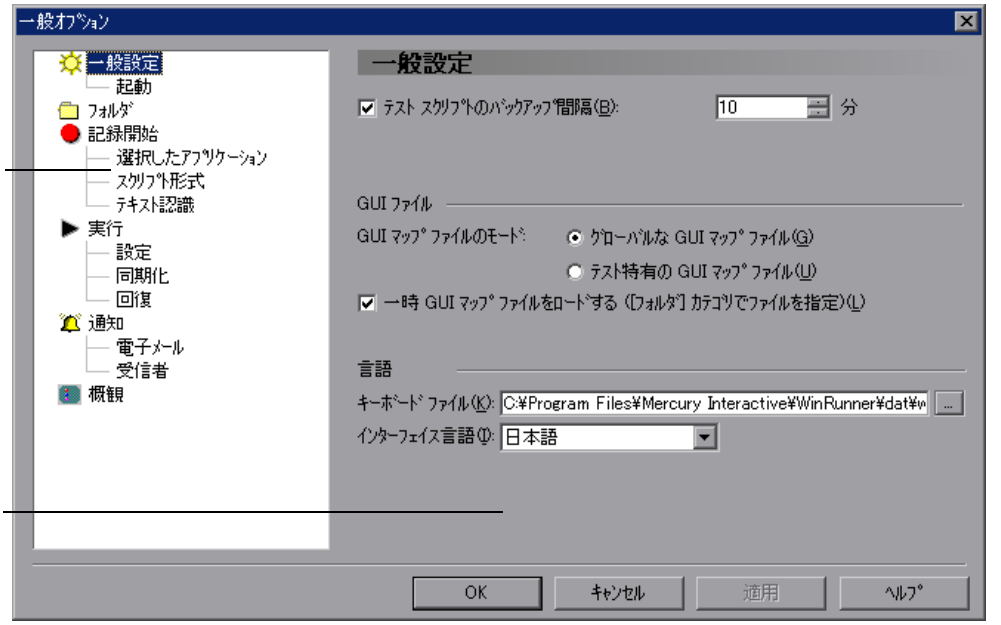
［一般オプション］ダイアログ・ボックスには，次のカテゴリおよびサブカテゴリがあります。

カテゴリ	内容
［一般設定］	GUI マップ設定や言語設定その他の一般テスト・オプション用のオプションがあります。
＞ **［起動］**	WinRunner を開いたときに何が実行されるのかを制御するオプションがあります。

カテゴリ	内容
［フォルダ］	WinRunner ファイルのフォルダ場所と，相対パス解決のための検索パスを指定します。
［記録開始］	テストを記録するためのオプションがあります。
> **［選択したアプリケーション］**	記録するアプリケーションを選択するためのオプションがあります。
> **［スクリプト形式］**	スクリプトの表示形式と可読性を制御するためのオプションがあります。
> **［テキスト認識］**	アプリケーションでテキストを認識するためのオプションがあります。
［実行］	テストを実行するためのオプションがあります。
> 設定	テスト実行中における特定の状況を処理するためのオプションがあります。
> ［同期化］	テスト実行の同期化設定を定義します。
> 回復	回復ファイルと Web 例外ファイルを指定するためのオプションがあります。
［通知］	電子メール通知を送信するための条件を指定できます。
> ［電子メール］	使用するメール・サーバその他の電子メール設定を指定するためのオプションがあります。
> ［受信者］	電子メール通知を受信する受信者を指定できます。
［概観］	WinRunner の表示形式を制御するためのオプションがあります。

共通テスト・オプションを設定するには，次の手順を実行します。

1 ［**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・ボックスが開きます。



2 オプション・ツリーでカテゴリまたはサブカテゴリをクリックし，対応するオプションをオプション表示枠に表示します。

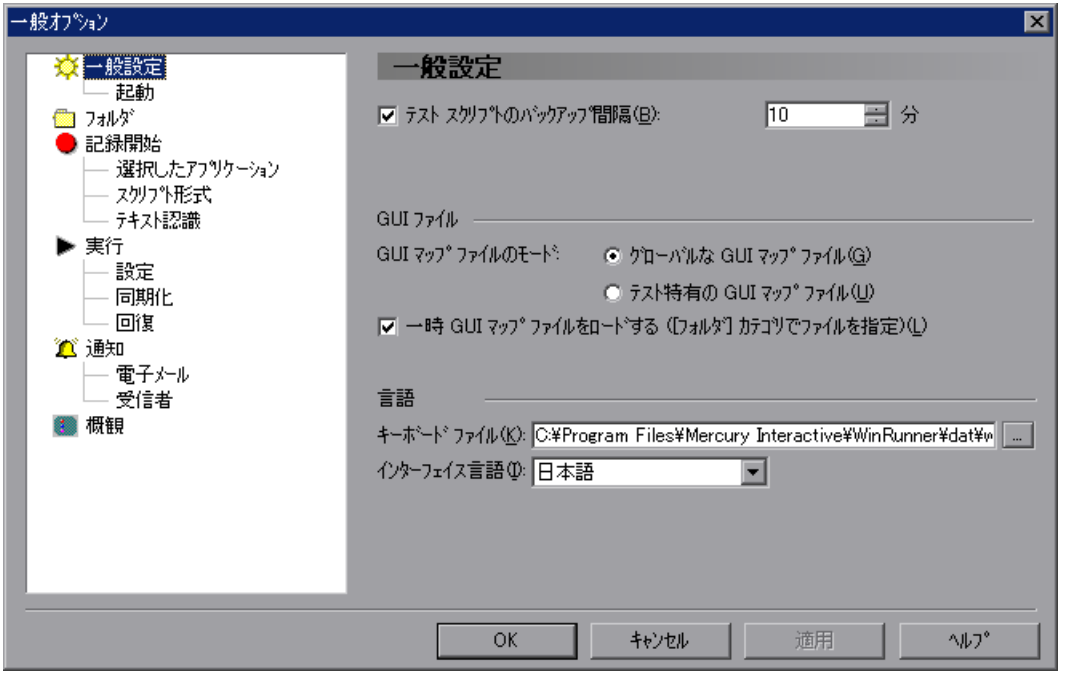
3 以降の項で説明する手順に従って必要なオプションを設定します。

4 変更を適用し，［一般オプション］ダイアログ・ボックスを表示したままにするには，［**適用**］をクリックします。

5 終わったら［**OK**］をクリックし，変更を保存してダイアログ・ボックスを閉じます。

一般オプションの設定

［**一般設定**］カテゴリには，GUI マップ設定や言語設定その他の一般テスト・オプション用のオプションがあります。



このカテゴリにあるオプションに加えて，［**起動**］サブカテゴリにある追加の記録オプションも設定できます。

［**一般設定**］カテゴリには次のオプションがあります。

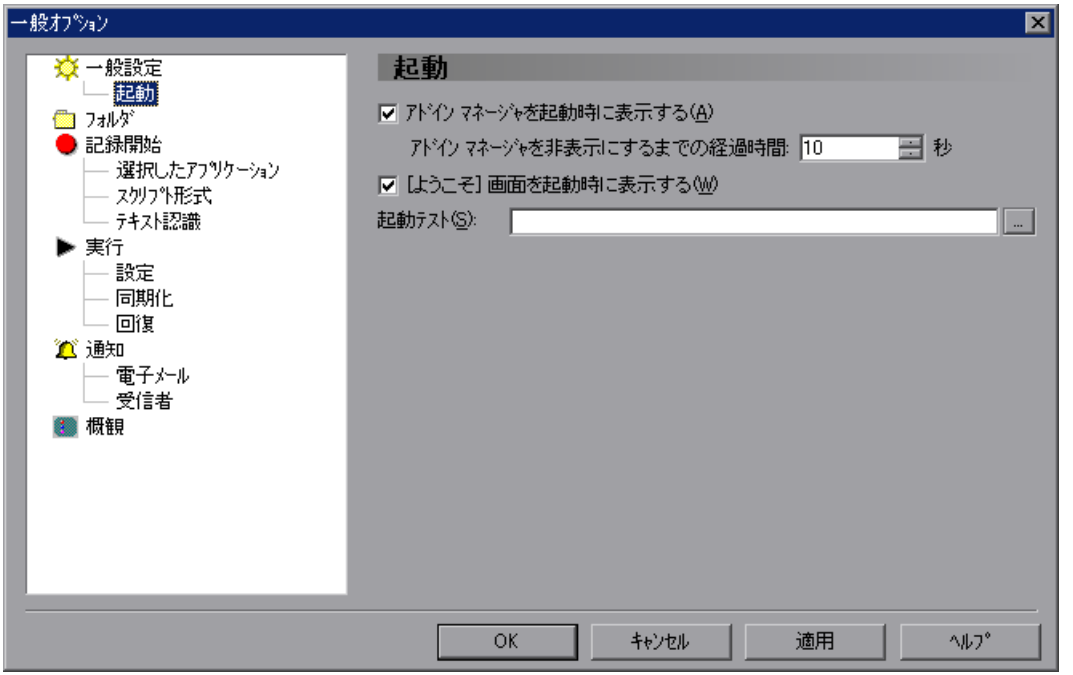
オプション	詳細
［テスト スクリプトのバックアップ間隔］	指定したインターバルで定期的にスクリプトのバックアップ・ファイルを作成するよう指定します。このオプションを選択すると，**script.sav** という名前のテスト・フォルダにバックアップ・ファイルが作成されます。これはスクリプトの単純なテキスト・ファイルです。バックアップを保存するたびに，前の **script.sav** ファイルに上書きされます。 **標準設定** = 選択されている，10（分）
［タイプ保存ダイアログを表示する］	このオプションは，WinRunner が Quality Center サーバに接続されているときにのみ表示されます。 ［タイプの選択］ダイアログ・ボックスを表示します。このダイアログ・ボックスで，新しいスクリプトを WinRunner テストまたはスクリプト化されたコンポーネントとして保存できます。 **注：** ［タイプの選択］ダイアログ・ボックスの下部にある［**このダイアログを次回から表示しない**］チェック・ボックスをチェックすると，［**一般設定**］表示枠での選択もクリアされます。 **標準設定** = 選択されている

オプション	詳細
[GUI マップ ファイルのモード]	WinRunner の GUI マップ・ファイル・モードを設定します。 • ［**グローバルな GUI マップ ファイル**］: アプリケーション全体，またはアプリケーション内の各ウィンドウに対して，GUI マップ・ファイルを作成できます。複数のテストで共通の GUI マップ・ファイルを参照できます。詳細については，第 5 章「グローバル GUI マップ・ファイル・モードでの作業」を参照してください。 • ［**テスト特有の GUI マップ ファイル**］: 作成する各テストに対して GUI マップ・ファイルを自動的に作成できます。GUI マップ・ファイルの作成，保存，ロードなどの心配をしなくて済みます。詳細については，第 6 章「テスト特有の GUI マップ・ファイル・モードでの作業」を参照してください。 **注：** この設定を有効にするには，WinRunner を再起動する必要があります。WinRunner 6.02 以前で作成されたテストで作業する場合は，［グローバル GUI マップ ファイル］モードで作業する必要があります。 **標準設定** = グローバルな GUI マップ・ファイル
[一時 GUI マップ ファイルをロード]	WinRunner の起動時に一時 GUI マップ ファイルを自動的にロードします。 **注：** ［**テスト特有の GUI マップ ファイル**］オプションが選択されているときは，このオプションは無効になります。これは，テストごとに別々の GUI マップ・ファイルで作業している場合，一時 GUI マップ・ファイルは存在しないためです。 一時 GUI マップ・ファイルの場所は，［一般オプション］ダイアログ・ボックスの［**フォルダ**］カテゴリで設定できます。 **標準設定** = 選択されている

オプション	詳細
[キーボード ファイル]	キーボード定義ファイルのパスを指定します。このファイルは，記録中にキーボード入力したときに，テスト・スクリプトに表示される言語を指定します。 **標準設定 = ＜WinRunner のインストール・フォルダ＞¥dat¥win_scan.kbd**
[インタフェース言語]	WinRunner が英語以外のオペレーティング・システムにインストールされている場合は，[**インタフェース言語**] オプションが表示されることがあります。このオプションでは，WinRunner のインタフェース言語を選択できます。

起動オプションの設定

[**起動**] カテゴリには，WinRunner を開いたときに何が実行されるのかを制御するオプションがあります。

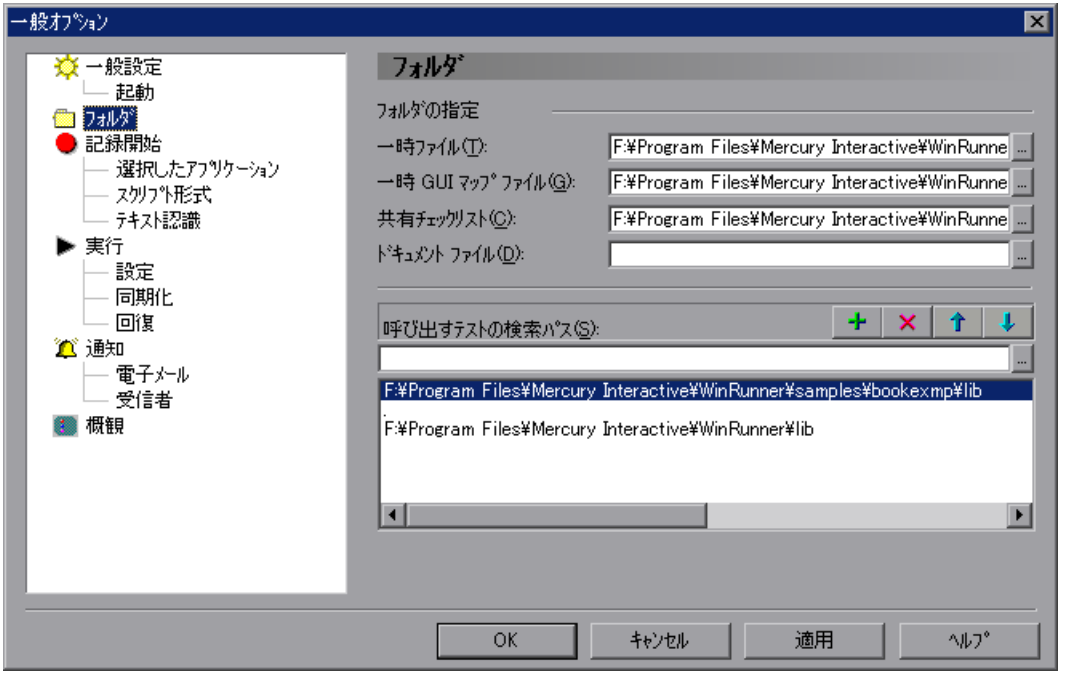


［**起動**］カテゴリには次のオプションがあります。

オプション	詳細
［起動時にアドイン マネージャを表示する］	WinRunner の起動時に［アドイン マネージャ］ダイアログ・ボックスを表示します。 ［アドイン マネージャ］ダイアログ・ボックスと，WinRunner の起動時におけるインストール済みアドインのロードの詳細については，21 ページ「WinRunner アドインのロード」を参照してください。 **標準設定** = 選択されている
［アドイン マネージャを非表示にするまでの経過時間］	アドイン・マネージャを表示し続ける秒数を指定します。この後，アドイン・マネージャが閉じ，以前の WinRunner セッションでロードされたものと同じアドインが自動的にロードされます。 **標準設定** = 10 秒
［[ようこそ] 画面を起動時に表示する］	WinRunner の起動時にようこそ画面を表示します。 **注：** ようこそ画面の下部にある［**起動時に表示**］チェック・ボックスをクリアすると，［**起動**］表示枠での選択もクリアされます。 **標準設定** = 選択されている
［起動テスト］	起動テストの場所を指定します。 起動テストを使用して，記録の構成設定，コンパイル済みモジュールのロード，および GUI マップ・ファイルのロードを，WinRunner の起動時に実行できます。 詳細については，第 45 章「特殊な構成の初期化」を参照してください。 **注：** 起動テストの場所は，テスト・スクリプト・ウィザードからも設定できます。 起動テストは初期化（**tslinit**）テストとは別に（代わりとしてではなく）使用できます。 Quality Center スクリプトを起動テストとして指定できます。指定する場合は，［Quality Center への接続］ダイアログ・ボックスで［**起動時に再接続する**］が選択されていることを確認してください。詳細については，964 ページ「プロジェクトの接続と接続の解除」を参照してください。 **標準設定** = **＜ WinRunner のインストール・フォルダ＞**

フォルダ・オプションの設定

［**フォルダ**］カテゴリでは，WinRunner ファイルの場所と，相対パス解決のための検索パスを指定できます。



［**フォルダ**］カテゴリには次のオプションがあります。

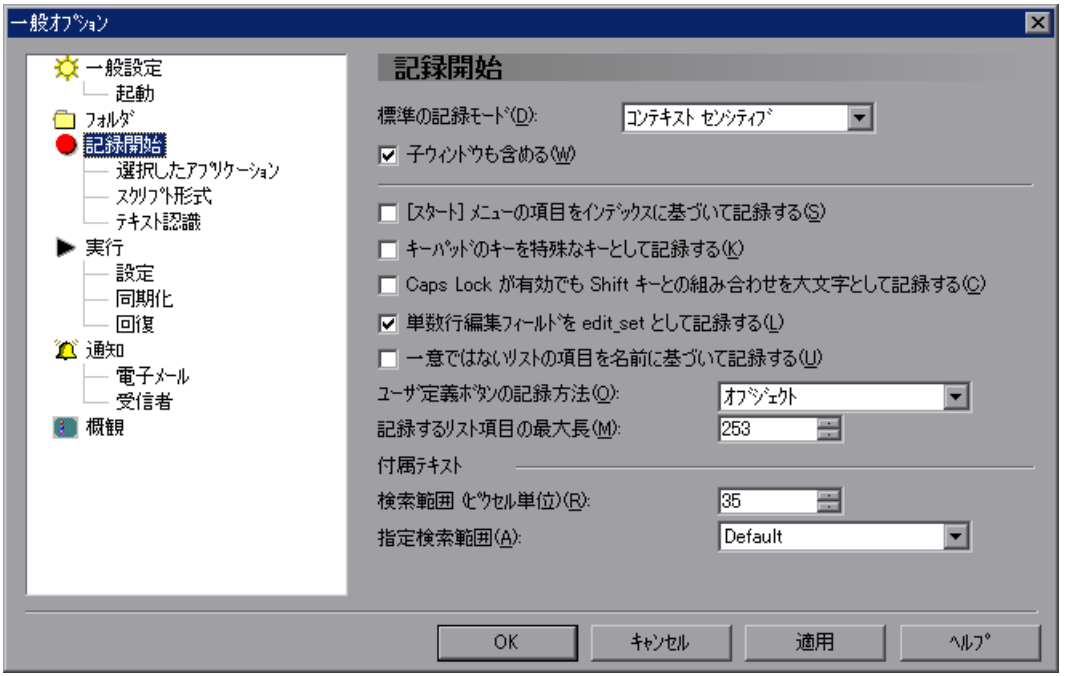
オプション	詳細
［一時ファイル］	一時テストを格納するフォルダ。フォルダを入力または参照します。 **注：** 新しいフォルダを指定した場合は，変更を有効にするために WinRunner を再起動する必要があります。 テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **tempdir** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 **標準設定 = ＜ WinRunner のインストール・フォルダ＞ ¥tmp**

オプション	詳細
[一時 GUI マップ・ファイル]	一時 GUI マップ・ファイル（*temp.gui*）を格納するフォルダ。[一般オプション] ダイアログ・ボックスの［**一般設定**］カテゴリの［**一時 GUI マップ ファイルをロードする**］チェック・ボックスを選択すると，このファイルが WinRunner の起動時に自動的にロードされます。新しいフォルダを入力するには，テキスト・ボックスに入力するか，［**参照**］をクリックして見つけます。 **注：** 新しいフォルダを指定した場合は，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。 標準設定 = **＜ WinRunner のインストール・フォルダ＞ ¥tmp**
[共有チェックリスト]	WinRunner が GUI チェックポイントとデータベース・チェックポイントのための共有チェックリストを格納するフォルダ。テスト・スクリプトでは，共有チェックリスト・ファイルは，**win_check_gui** ステートメント，**obj_check_gui** ステートメント，または **db_check** ステートメントの中で，ファイル名の前に SHARED_CL を付けて示されます。新しいパスを入力するには，テキスト・ボックスに入力するか，［**参照**］をクリックしてフォルダを見つけます。共有 GUI チェックポイントの詳細については，143 ページ「GUI チェックリストの共有フォルダへの保存」を参照してください。共有データベース・チェックポイントの詳細については，305 ページ「データベース・チェックリストを共有フォルダに保存」を参照してください。 **注：** 新しいフォルダを指定した場合は，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。 **getvar** 関数を使用すると，対応する **shared_checklist_dir** テスト・オプションの値をテスト・スクリプトの中から取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 **標準設定 = ＜ WinRunner のインストール・フォルダ＞ ¥chklist**
[ドキュメントファイル]	文書ファイルを格納するフォルダ。新しいパスを入力するには，テキスト・ボックスに入力するか，［参照］をクリックしてフォルダを見つけます。 **標準設定 = ＜ WinRunner のインストール・フォルダ＞ ¥doc**

オプション	詳細
［呼び出すテストの検索パス］	WinRunner が相対パスで指定されたファイルまたはテストを検索するパス。この表示枠で検索パスを定義した場合は，テストを呼び出して他のファイル名を指定するときに相対パスを指定できます。リスト中の検索パスの順序によって，相対パスを使用して指定されたファイルまたはテストを WinRunner が検索する際の検索場所の順序が決まります。 詳細については，第 29 章「テストの呼び出し」を参照してください。 • 検索パスを追加するには，パスをテキスト・ボックスに入力して，［**パスの追加**］をクリックします。  これで，パスがテキスト・ボックスの下のリスト・ボックスに表示されます。 • 検索パスを削除するには，パスを選択して［**パスの削除**］をクリックします。  • リスト中で選択した検索パスの位置を 1 つ上に移動するには，パスを選択して［**項目を上に移動**］をクリックします。  • リスト中で選択した検索パスの位置を 1 つ下に移動するには，パスを選択して［**項目を下に移動**］をクリックします。  WinRunner から Quality Center に接続すると，WinRunner が呼び出し先のテストを検索する際の Quality Center データベース内のパスを指定できます。Quality Center データベース内の検索パスには前に [QC] が付きます。ただし，［**参照**］ボタンを使用して Quality Center データベース内の検索パスを指定することはできません。 **注：** テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **searchpath** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。また，このオプションは対応する **-search_path** コマンド・ライン・オプションを使用して設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

記録オプションの設定

［**記録開始**］カテゴリには，WinRunner でのテストの記録方法を制御するためのオプションがあります。



このカテゴリにあるオプションに加えて，［**選択したアプリケーション**］，［**スクリプト形式**］，および［**テキスト認識**］サブカテゴリにある追加の記録オプションも設定できます。

［**記録**］カテゴリには次のオプションがあります。

オプション	詳細
［標準の記録モード］	標準の記録モードを［**コンテキスト センシティブ**］または［**アナログ**］に指定します。テストを記録している間，記録モードを切り替えることができます。詳細については，第 3 章「WinRunner の GUI オブジェクトの識別方法」を参照してください。 標準設定＝コンテキスト・センシティブ

オプション	詳細
[子ウィンドウも含める]	選択されている場合，WinRunner はすべての MSW_class ウィンドウ，またはこのクラスにマップされるすべてのオブジェクトを，親オブジェクトとして認識します。選択されていない場合，WinRunner は最上位のウィンドウおよび MDI フレームだけを親オブジェクトとして認識します。 テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **enum_descendent_toplevel** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 **標準設定** = 選択されている

オプション	詳細
[[スタート］メニューの項目をインデックスに基づいて記録する]	WinRunner が Windows NT で Windows の［**スタート**］メニューをどのように記録するかを指定します。 このオプションが選択されている場合，WinRunner は選択された各メニュー項目のインデックス ID を記録します。例えば，次のようになります。 button_press ("Start");menu_select_item ("item_2;item_0;item_4"); メニュー項目の位置が一定で，選択しようとするメニューの名前が変更される可能性ある場合に，このオプションを選択します。例えば，メニュー・オプションの名前が動的に生成される場合などがあります。 このオプションが選択されていない場合，WinRunner はスタート・メニュー内のメニュー項目の名前を記録します。例えば，次のようになります。 button_press ("Start");menu_select_item ("Programs;Accessories;Calculator"); **標準設定** = 選択されていない

オプション	詳細
[キーパッドのキーを特殊なキーとして記録する]	WinRunner が数値キーボード上のキーの押下をどのように記録するかを指定します。 このオプションが選択されている場合，WinRunner は NUM LOCK キーの押下を記録します。また，数値キーパッド上の数値キーと制御キーを押した場合は，それらを一意のキーとして，生成する **obj_type** ステートメントに記録します。例えば，次のようになります。 obj_type ("Edit","<kNumLock>") obj_type ("Edit","<kKP7>") このオプションが選択されていない場合，キーボードの数値キーや矢印キーを押した場合でも，数値キーパッドの数値キーや矢印キーを押した場合でも，WinRunner は同じステートメントを生成します。WinRunner は NUM LOCK キーの押下を記録しません。数値キーパッドの数値キーや制御キーを押した場合，それらを一意のキーとして，生成する **obj_type** ステートメントに記録しません。例えば，次のようになります。 obj_type ("Edit","7"); **注：** このオプションは **edit_set** ステートメントがどのように記録されるかには影響を与えません。**edit_set** を使用して記録する場合には，WinRunner はキーパッド・キーを特殊なキーとして区別せずに記録します。 **標準設定** = 選択されていない
[Caps Lock が有効でも Shift キーとの組み合わせを大文字として記録する]	CAPS LOCK が有効になっているときに，WinRunner が押された文字キーと SHIFT キーをまとめて大文字として記録するかどうかを指定します。 このオプションが選択されている場合，CAPS LOCK が有効になっていれば，WinRunner は押された文字キーと SHIFT キーをまとめて大文字として記録します。したがって，テストの記録と実行をしているときには，CAPS LOCK キーの状態は無視されます。 このオプションが選択されていない場合，CAPS LOCK が有効になっていても，WinRunner は押された文字キーと SHIFT キーをまとめて小文字として記録します。 **標準設定** = 選択されていない

オプション	詳細
[単数行編集フィールドを edit_set として記録する]	WinRunner が単一行の編集フィールドへの文字列の入力をどのように記録するかを指定します。 このオプションが選択されている場合，WinRunner は **edit_set** ステートメントを記録します（すべてのキーの押下と解放の最終結果だけが記録されます）。例えば，フライト予約アプリケーションの［**Name**］ボックスに「H」と入力して BACKSPACE を押し，その後で「Jennifer」と入力すると，WinRunner は次のステートメントを生成します。 edit_set ("Name","Jennifer"); このオプションが選択されていない場合，WinRunner は **obj_type** ステートメントを生成します（すべてのキーの押下と解放が記録されます）。上の例と同じキー入力をした場合，WinRunner は次のステートメントを生成します。 obj_type ("Name","H<kBackSpace>Jennifer"); **edit_set** 関数と **obj_type** 関数の詳細については，「**TSL リファレンス**」を参照してください。 **標準設定** = 選択されている
[一意ではないリスト項目を名前に基づいて記録する]	リスト・ボックスおよびコンボ・ボックスの一意でない項目を，WinRunner が名前で記録するか（選択されている場合），インデックスで記録するか（選択されていない場合）を指定します。 **注：** テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **rec_item_name** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。このオプションは対応する **-rec_item_name** コマンド・ライン・オプションを使用して設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。 **標準設定** = 選択されていない

オプション	詳細
[ユーザ定義ボタンの記録方法]	WinRunner はオーナー描画ボタンのクラスを識別できないので，自動的に汎用の「Object」クラスにマップします。このオプションはオーナー描画ボタンをすべて標準のボタン・クラス（push_button，radio_button，あるいは check_button）にマップできるようにします。 テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **rec_owner_drawn** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 **標準設定** = オブジェクト
[記録するリスト項目の最大長]	WinRunner がリスト項目名として記録できる最大文字数を定義します。 リスト・ビュー項目またはツリー・ビュー項目が最大文字数を超えた場合，WinRunner はその項目のインデックス番号を記録します。 リスト・ボックスやコンボ・ボックスで最大文字数を超えた場合，WinRunner はそれらの項目の名前を切り捨てます。最大の長さは 1 文字から 253 文字までです。 **標準設定** = 253 文字
[付属テキスト]	WinRunner が GUI オブジェクトに付属しているテキストをどのように探すかを指定します。GUI オブジェクトからの距離は，検索対象半径と，検索が開始される GUI オブジェクト上の点の，2 つのオプションによって決まります。GUI オブジェクト上の指定された点から，指定された検索半径内にある最も近い静的テキスト・オブジェクトが，そのオブジェクトの付属テキストになります。 ただし，GUI オブジェクトに最も近い静的テキスト・オブジェクトが実際には最も近い静的テキスト・オブジェクトではないこともあります。付属テキスト属性が，自分で選んだ静的テキスト・オブジェクトであることを確認するためには，試行錯誤が必要になるでしょう。 テストを実行する際，付属テキスト・オプションには，テストを記録するときに使用したものと同じ値を使用する必要があります。そのようにしないと，WinRunner は GUI オブジェクトを識別できないことがあります。

オプション	詳細
［検索範囲］	GUI オブジェクト上の指定された点からの半径を指定します。WinRunner はこの範囲で，その付属テキストである静的テキスト・オブジェクトを探します。半径は 3 ～ 300 ピクセルまで指定できます。 テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **attached_text_search_radius** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 **標準設定** = 35 ピクセル
［指定検索範囲］	WinRunner が付属テキストの検索を開始する GUI オブジェクト上の位置を指定します。 テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **attached_text_area** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 WinRunner バージョン 7.01 以前では，指定検索領域を設定することができませんでした。従来のバージョンの WinRunner は付属テキストを検索する際，指定検索領域に対して，現在の［**Default**］設定にあたる設定で検索をしていました。従来バージョンとの互換性が重要である場合は［**Default**］の設定を選んでください。 **標準設定** = Default

選択したアプリケーションの設定

［選択したアプリケーション］表示枠では，選択したプログラムに対する動作だけを記録して，他のプログラムに対する動作を記録しないように WinRunner を設定できます。例えば，テスト記録中に電子メールに対して実行した動作を記録したくない場合などに使用します。

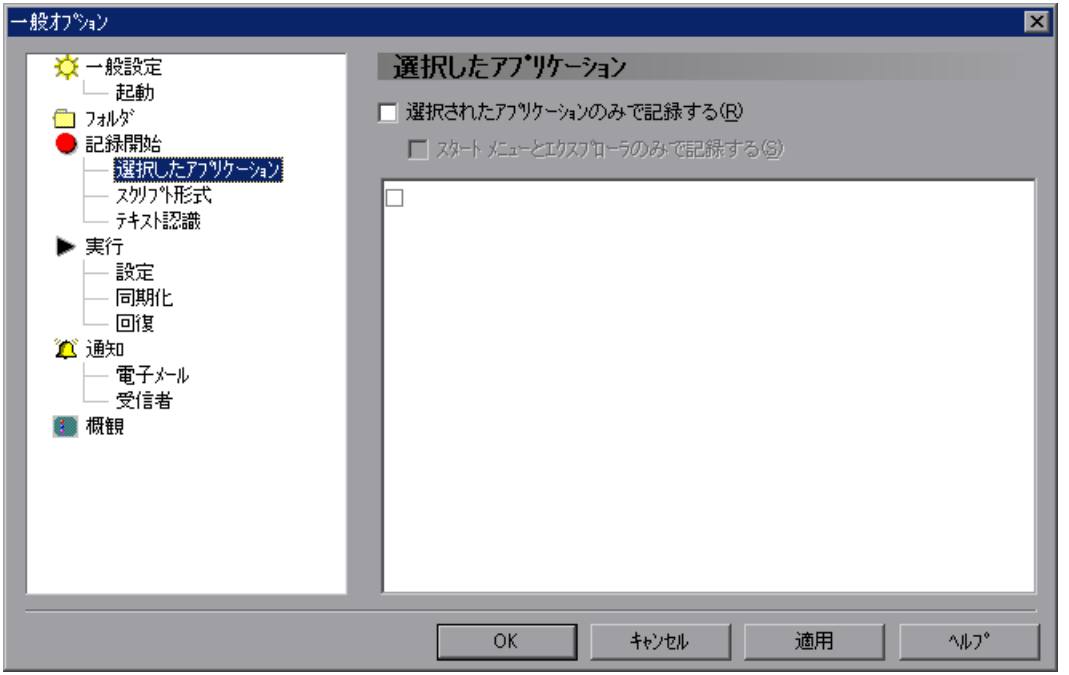
［オプションで記録］を指定すると，選択したプログラムの動作だけを記録できます。

選択したアプリケーションに対する記録だけを行なうように選択していても，チェックポイントを作成し，その他の記録しない操作をすべてのアプリケーションに対して実行することができます。

［オプションで記録］を指定するには，次の手順を実行します。

1 ［**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・ボックスが開きます。

2 ［**選択したアプリケーション**］カテゴリをクリックします。



3 ［**選択されたアプリケーションのみで記録**］を選択します。

4 ［**スタート**］メニューや Windows エクスプローラでの動作を記録する場合は，［**スタート メニューとエクスプローラのみで記録する**］を選択します。関連するファイルが自動的にリストに追加されます。

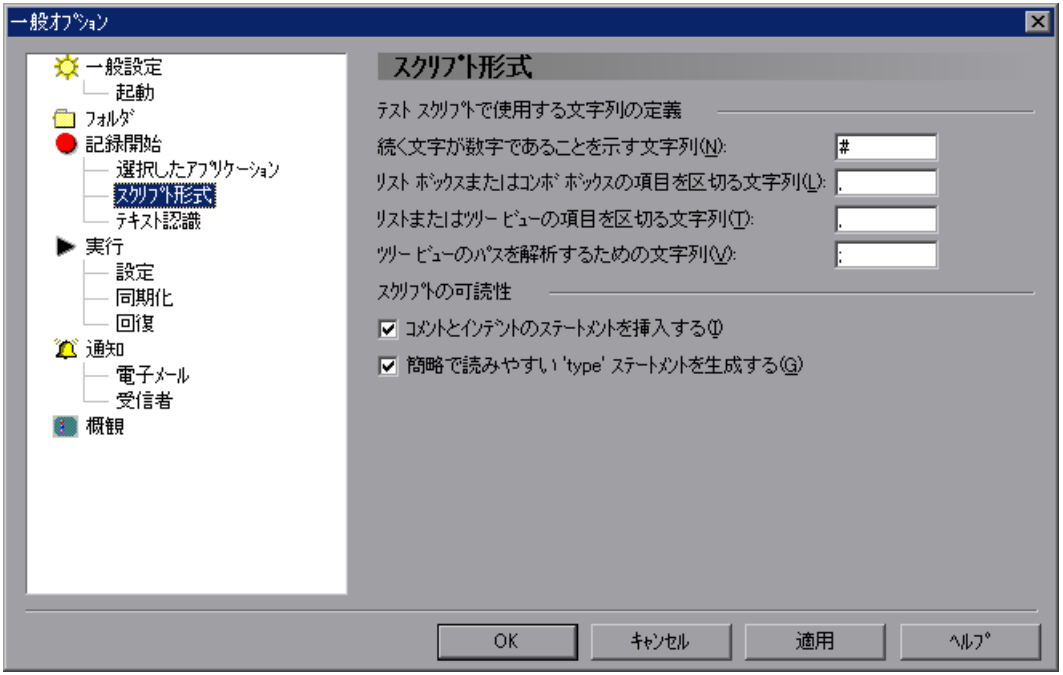
5 Internet Explorer，Netscape，あるいはその両方での動作を記録しない場合は，アプリケーション・リストで，**iexplore.exe**，**netscape.exe**，**netscp6.exe** に対する各オプションをクリアします。

6 新しいアプリケーションをリストに追加するには，空のリスト項目をクリックします。アプリケーション・ファイル名をボックスに入力するか，参照ボタンを使用してアプリケーションを検索し，選択します。

注：記録するアプリケーション・プロセスを必ず入力してください。場合によっては，プロセス・ファイル名が，アプリケーションの実行に使用するファイル名と同じ名前ではないこともあります。

スクリプト形式オプションの設定

［**スクリプト形式**］カテゴリには，スクリプトの表示形式と可読性を制御するためのオプションがあります。



［**スクリプト形式**］カテゴリには次のオプションがあります。

オプション	詳細
［続く文字が数字であることを示す文字列］	リスト項目がインデックス番号で指定されていることを示すために，テスト・スクリプトに記録される文字列。 テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **item_number_seq** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 **標準設定** = #
［リストボックスまたはコンボボックスの項目を区切る文字列］	リスト・ボックスまたはコンボ・ボックスの項目を区切るためにテスト・スクリプトに記録される文字列。 テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **list_item_separator** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 **標準設定** = ,
［リストまたはツリービューの項目を区切る文字列］	リスト・ビューまたはツリー・ビューの項目を区切るためにテスト・スクリプトに記録される文字列。 テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **listview_item_separator** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 **標準設定** = ,
［ツリービューのパスを解析するための文字列］	ツリー表示のパスの項目を区切るためにテスト・スクリプトに記録される文字列。 テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **treeview_path_separator** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 **標準設定** = ;

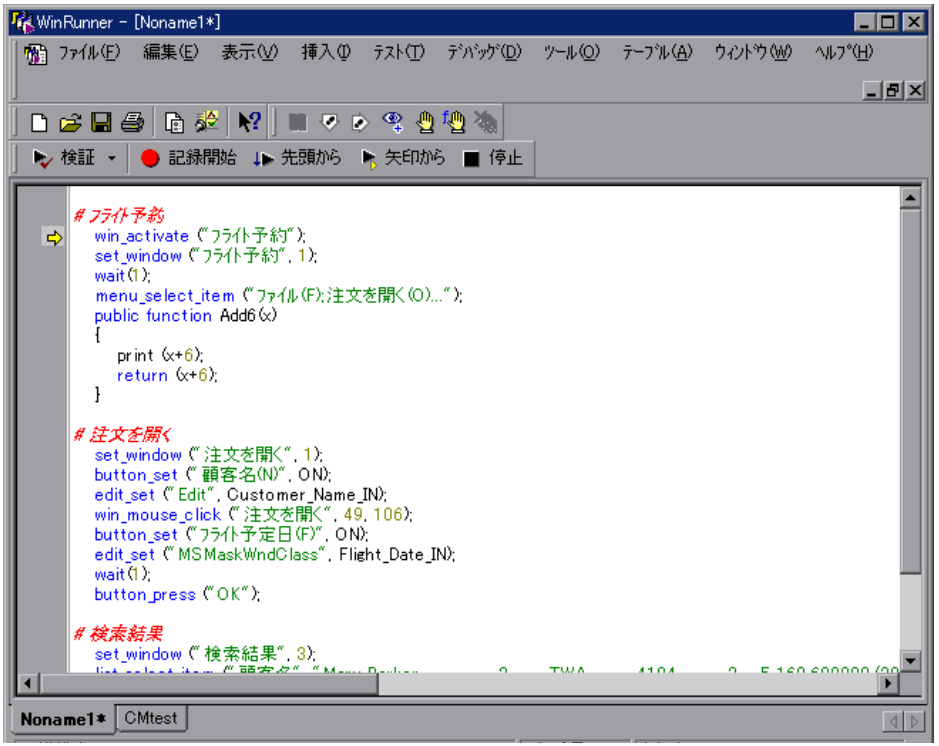
オプション	詳細
[コメントとインデントのステートメントを挿入する]	WinRunner が記録中にテスト・スクリプトを自動的にセクションに分割するかどうかを指定します。 詳細については，次の「コメントとインデントのステートメントの挿入」を参照してください。 **標準設定** = 選択されている
[簡略で読みやすい 'type ' ステートメントを生成する]	WinRunner が **type**，**win_type**，**obj_type** の各ステートメントをテスト・スクリプト内にどのように生成するかを指定します。 このオプションが選択されている場合，WinRunner は，入力キーの押下と解放の最終結果だけを表す，より簡潔な **type**，**win_type**，および **obj_type** ステートメントを生成します。これによってテスト・スクリプトが読みやすくなります。例えば，次のようになります。 obj_type (object, "A"); このオプションが選択されていない場合，WinRunner は，各キーの押下と解放をすべて記録します。例えば，次のようになります。 obj_type (object, "<kShift_L>-a-a+<kShift_L>+"); テストにおいて，キー入力の正確な順番が重要な場合は，このオプションをクリアしてください。 詳細については，「**TSL リファレンス**」の **type**，**win_type**，**obj_type** の各関数を参照してください。 テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **key_editing** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 **標準設定** = 選択されている

コメントとインデントのステートメントの挿入

［**コメントとインデントのステートメントを挿入する**］オプションを選択した場合，WinRunner は以下を自動的に実行します。

- 記録中に，ウィンドウ・フォーカスの変化に応じてテスト・スクリプトをセクションに分割します。
- 現在のウィンドウを記したコメントを挿入します。
- 各コメントの下のステートメントにインデントを付けます。

このオプションによって，同じウィンドウに関連するすべてのステートメントをグループ化できます。

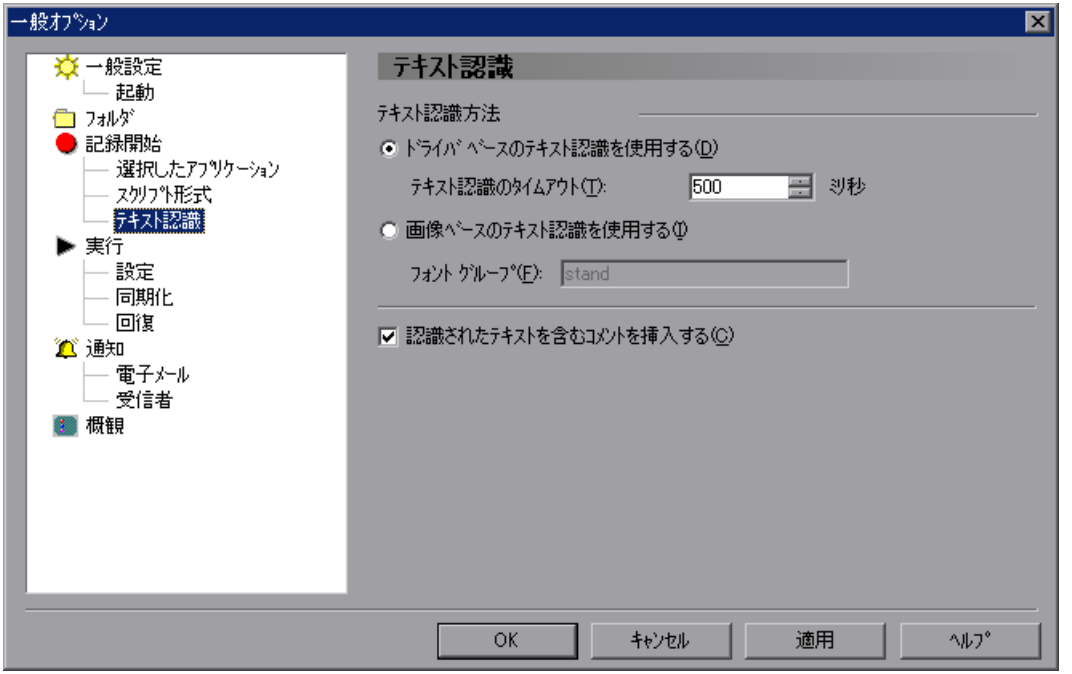


このオプションが選択されている場合，WinRunner は記録中，自動的にテストをセクションに分割します。**set_window** ステートメントも **win_*** ステートメントと同様に，セクションに分割できます。コンテキスト・センシティブからアナログに記録モードを切り換えると新しいセクションが始まります。

WinRunner が作成する新しいセクションにはそれぞれ，ウィンドウ名の記されたコメントが挿入されます。同じウィンドウがフォーカスされている間に記録されたステートメントはすべて，コメントの下に字下げされます。このオプションを選択してアナログ・モードで記録すると，コメントは常に Analog Recording となります。

テキスト認識オプションの設定

［**テキスト認識**］カテゴリのオプションは，WinRunner がアプリケーション内のテキストをどのように認識するかに影響を与えます。



［**テキスト認識**］カテゴリには次のオプションがあります。

オプション	詳細
［ドライバ ベースのテキスト認識を使用する］	グラフィックス・ドライバを使用してテキストを認識します。通常はこの方法で最も信頼性の高いテキスト結果が得られます。テスト対象のアプリケーションでこの方法が有効でない場合にのみ，［**画像ベースのテキスト認識を使用する**］を選択してください。 **標準設定** = 選択されている
［テキスト認識のタイムアウト］	テスト実行中にドライバによるテキスト認識方法を使用してテキスト・チェックポイントを実行するときに，WinRunner がテキストを認識するまで待機する最大間隔（ミリ秒単位）を設定します。 この設定の調整をいつ行うかについては，570 ページ「適切なタイムアウトと遅延設定の選択」を参照してください。 **標準設定** = 500 ミリ秒
［画像ベースのテキスト認識を使用する］	フォントがフォント・グループで定義されているテキストを WinRunner が認識できるようにします。このオプションは，テスト対象のアプリケーションでドライバによるテキスト認識方法が有効でない場合にのみ，選択してください。 **標準設定** = 選択されていない

オプション	詳細
［フォント グループ］	イメージ・テキスト認識のためのアクティブ・フォント・グループを設定します。フォント・グループの詳細については，350 ページ「WinRunner によるフォントの学習」を参照してください。 テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **fontgrp** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 このオプションは対応する **-fontgrp** コマンド・ライン・オプションを使用して設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。 **標準設定** = stand
［認識されたテキストを含むコメントを挿入する］	このオプションは，テキスト・チェックポイントを作成するときに，WinRunner がテスト・スクリプトの中でキャプチャされたテキストをどのように表示するのかを決めます。 選択されている場合，WinRunner はテスト作成中にテキスト・チェック・ポイントによってキャプチャされたテキストをテスト・スクリプトにコメントとして挿入します。例えば，［**挿入**］＞［**テキスト取得**］＞［**オブジェクト / ウィンドウ**］を選択し，「Portland」が選択された状態で［出発地］テキストボックスをクリックすると，以下のステートメントがテスト・スクリプトに記録されます。 obj_get_text(" 出発地：", text); # Portland **標準設定** = 選択されている

Windows アプリケーションに対してテキスト認識を使用する際の注意事項

WinRunner のテキスト認識メカニズムは以下の場合に使用します。

- ［**挿入**］＞［**テキストの取得**］＞［**画面領域から**］および［**挿入**］＞［**テキストの取得**］＞［**オブジェクト / ウィンドウから**］を使用してテキスト・チェックポイントを挿入する場合
- **_get_info** または **_check_info** で終わる関数を使用して GUI オブジェクトの **text** プロパティを取得または確認する場合

- **_get_text** または **_check_text** で終わる関数を使用してテキストを取得または確認する場合
- **_find_text**，**_move_locator_text**，または **_click_on_text** で終わる関数を使用して他のテキストベース処理を実行する場合

Windows アプリケーションに対して WinRunner テキスト認識メカニズムを使用するときは，不要なテキスト情報（隠しテキストや同じ文字列のコピーとして複数回表示される影付きテキストなどが）を取り込んでしまう場合があります。

また，使用しているオペレーティング・システムのバージョン，インストールしているサービス・パックやその他のツールキット，アプリケーションで使用する API などによって，テキスト認識の動作が実行セッションごとに異なることがあります。

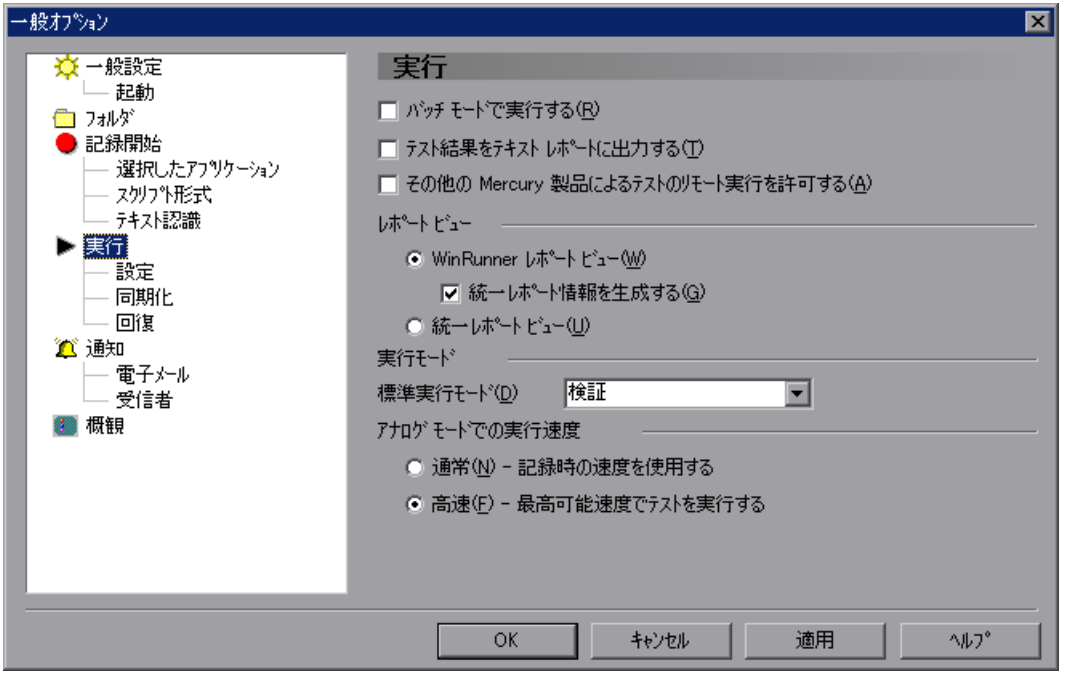
したがって，アプリケーション・ウィンドウからテキストを取得または確認する場合は，可能な限り，標準の GUI チェックポイントを挿入し，オブジェクトの **value**（または同様の）プロパティを調べることを選択することをお勧めします。例えば，次の手順を実行します。

- ［**挿入**］＞［**テキスト取得**］＞［**オブジェクト / ウィンドウ**］を選択する代わりに，［**挿入**］＞［**GUI チェックポイント**］＞［**単数のプロパティ**］を選択し，**value** プロパティを調べることを選択します。
- edit_get_text("Edit", result); や edit_get_info("Edit", "text", result); の代わりに，edit_get_info("Edit", "value", result); を使用します。
- edit_check_text("Edit", exp_val); や edit_check_info("Edit", "text", exp_val); の代わりに，edit_check_info("Edit", "value", expected_result); を使用します。

注：上記の問題は，Web ベースのアプリケーションで作業している場合には，適用されません。

テストの実行オプションの設定

［**実行**］カテゴリのオプションは，WinRunner がどのようにテストを実行するかに影響を与えます。



このカテゴリにあるオプションに加えて，［**設定**］，［**同期化**］，および［**回復**］サブカテゴリにある追加の実行オプションも設定できます。

［**実行**］カテゴリには次のオプションがあります。

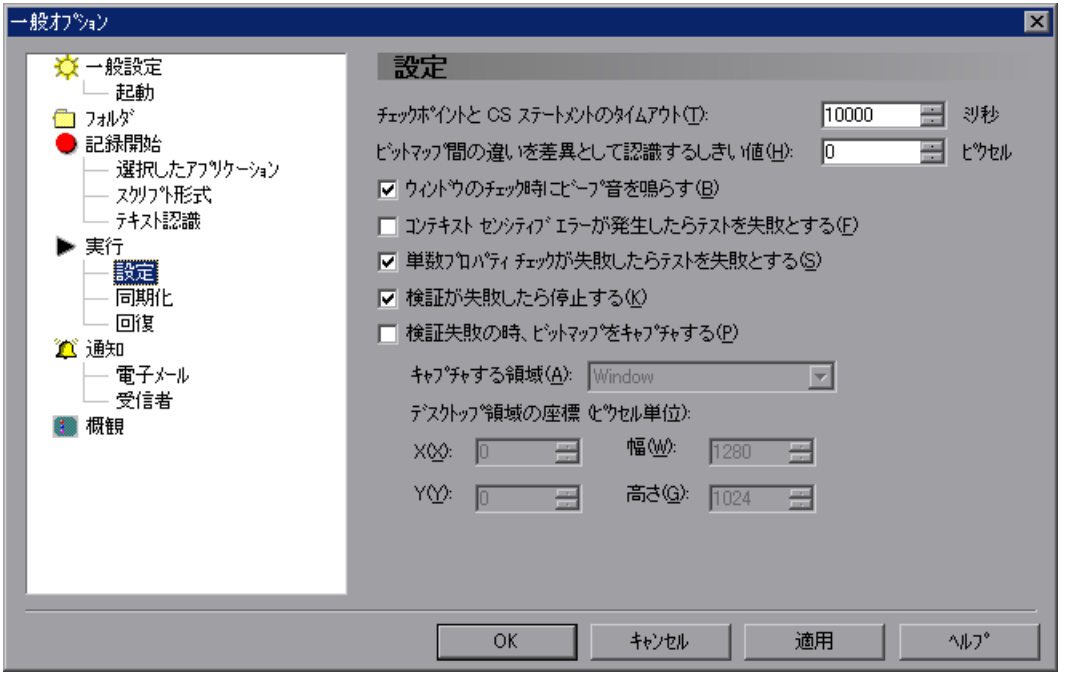
オプション	詳細
［バッチ モードで実行する］	テストを無人で実行できるようにするために，「**検証**」テストの実行中にメッセージを抑制するかどうかを指定します。 例えば，**set_window** ステートメントがテスト・スクリプトにないと，WinRunner は指定されたウィンドウを見つけることができません。テストをバッチ・モードで実行した場合，WinRunner は［テスト結果］ウィンドウにエラーを報告し，テスト・スクリプト内の次のステートメントに進みます。テストをバッチ・モードで実行しない場合，WinRunner はテストを一時停止し，［実行］ウィザードを開いてユーザがそのウィンドウを指定できるようにします。 **注：** バッチ・テストでメッセージが抑制されるのは，「**検証**」テスト実行モードを使用してテストを実行した場合のみです。「**更新**」または「**デバッグ**」実行モードを使用してテストを実行した場合は，［**バッチ モードで実行**］オプションを選択していても，一部のメッセージが表示されることがあります。 選択されている場合，WinRunner は呼び出し先テストのテスト結果を，呼び出し元（メイン・バッチ・テスト）の配下と，すべての第 1 レベルの呼び出し先テストのテスト・フォルダの配下の，両方に保存します。選択されていない場合，呼び出し先テストの結果は呼び出し元テストの配下にのみ保存されます。 テスト実行中のメッセージの抑制の詳細については，第 36 章「バッチ・テストの実行」を参照してください。 **getvar** 関数を使用すると，対応する **batch** テスト・オプションの値をテスト・スクリプトの中から取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 このオプションは対応する **-batch** コマンド・ライン・オプションを使用して設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。 **標準設定** = 選択されていない

オプション	詳細
[テスト結果をテキスト レポートに出力する]	結果フォルダに保存されるテキスト・レポート（**report.txt**）にテスト結果を自動的に書き込むよう，WinRunner を設定します。このオプションは対応する **-create_text_report** コマンド・ライン・オプションを使用して設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。 **注：** テスト結果のテキスト・レポートは，[テスト結果] ウィンドウ（WinRunner レポート・ビュー内）で［**ツール**］＞［**テキスト レポート**］を選択して作成することもできます。 **標準設定** = 選択されていない
[その他の Mercury 製品によるテストのリモート実行を許可する]	他の Mercury 製品で WinRunner のテストを自分のコンピュータ上でリモート・マシンから実行できるようにします。 他の Mercury 製品からリモートに WinRunner のテストを実行する方法については，各製品のマニュアルを参照してください。
[WinRunner レポート ビュー]	WinRunner のテスト結果表示を使用してテスト結果を表示します。 **標準設定** = 選択されている
[統一レポート情報を生成する]	統合レポートの作成に必要な情報を生成し，後でテストの結果を統合レポート・ビューで表示することを選択できるようにします。 （［**WinRunner レポート ビュー**］を選択した場合にのみ有効）。 **標準設定** = 選択されている
[統一レポート ビュー]	テストの実行中に統合レポート情報を生成し，統合レポートのデザインを使用してテスト結果を表示します。この表示では，すべての WinRunner イベントと QuickTest ステップを 1 つのレポートに表示できます。 **注：** このオプションを選択しているときは WinRunner レポートは常に自動的に生成され，後で WinRunner レポート・ビューに切り替えることができます。 **標準設定** = 選択されていない

オプション	詳細
[標準実行モード]	すべてのテストに対して標準で使用される実行モードを選択できます。 • [**更新**]：テストの期待結果を更新する場合や，新しい期待結果フォルダを作成する場合に使用します。 • [**検証**]：アプリケーションをテストする場合に使用します。 • [**デバッグ**]：テスト・スクリプト内の不具合の特定に使用します。 **注：** 検証モードはテストの実行時にのみ関係があり，コンポーネントの実行時には関係ありません。コンポーネントで作業しているときは，コンポーネントが Quality Center 内でビジネス・プロセス・テストの一部として実行されているときに，アプリケーションが検証されます。 実行モードの詳細については，422 ページ「WinRunner のテスト実行モード」を参照してください。 **標準設定** = 検証
[アナログ モードでの実行速度]	アナログ・モードでのテストの標準の実行速度を指定します。 [**通常**]：記録されたときの速度でテストを実行します。 [**高速**]：アプリケーションが入力を受け取れる最高の速さでテストを実行します。 テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **speed** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 このオプションは対応する **-speed** コマンド・ライン・オプションを使用して設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。 **標準設定** = 高速

実行の設定オプションの設定

［**設定**］カテゴリには，テスト実行中における特定の状況を処理するためのオプションがあります。



［**設定**］カテゴリには次のオプションがあります。

オプション	詳細
［チェックポイントと CS ステートメントのタイムアウト］	チェックポイントとコンテキスト・センシティブ・ステートメントを実行しているときに WinRunner が使用するグローバルなタイムアウトをミリ秒単位で設定します。この値は，GUI チェックポイントや同期化ポイントに埋め込まれた **time** パラメータに加算されます。WinRunner が指定されたウィンドウまたはオブジェクトを検索する際，この値が最長時間となります。タイムアウトの数値は，ウィンドウ同期化のための遅延の数値（［**同期化**］カテゴリの［**ウィンドウの同期化のための遅延**］オプションで設定します）よりも大きくする必要があります。 例えば，遅延時間が 2,000 ミリ秒で，タイムアウトが 10,000 ミリ秒の場合，WinRunner はテスト対象アプリケーション内のウィンドウまたはオブジェクトを，希望する検査結果が得られるまで，あるいは 10 秒が経過するまで，検査します。 **注：** 実際のタイムアウトでは，このオプションに設定した値から 20 ～ 30 ミリ秒以内の誤差が生じる場合があります。 この設定の調整をいつ行うかについては，570 ページ「適切なタイムアウトと遅延設定の選択」を参照してください。 テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **timeout_msec** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 このオプションは対応する **-timeout_msec** コマンド・ライン・オプションを使用して設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。 **標準設定** = 10000 ミリ秒

オプション	詳細
[ビットマップ間の違いを差異として認識するしきい値]	ビットマップの不一致のしきい値となるピクセル数を定義します。この値が 0 に設定されている場合は，1 ピクセルでも違いがあればビットマップの不一致と判定されます。 テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **min_diff** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 このオプションは対応する **-min_diff** コマンド・ライン・オプションを使用して設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。 **標準設定** = 0 ピクセル
[ウィンドウのチェック時にビープ音を鳴らす]	テスト実行中にウィンドウを検査するときにビープ音を鳴らすかどうかを指定します。 テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **beep** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 このオプションは対応する **-beep** コマンド・ライン・オプションを使用して設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。 **標準設定** = 選択されている

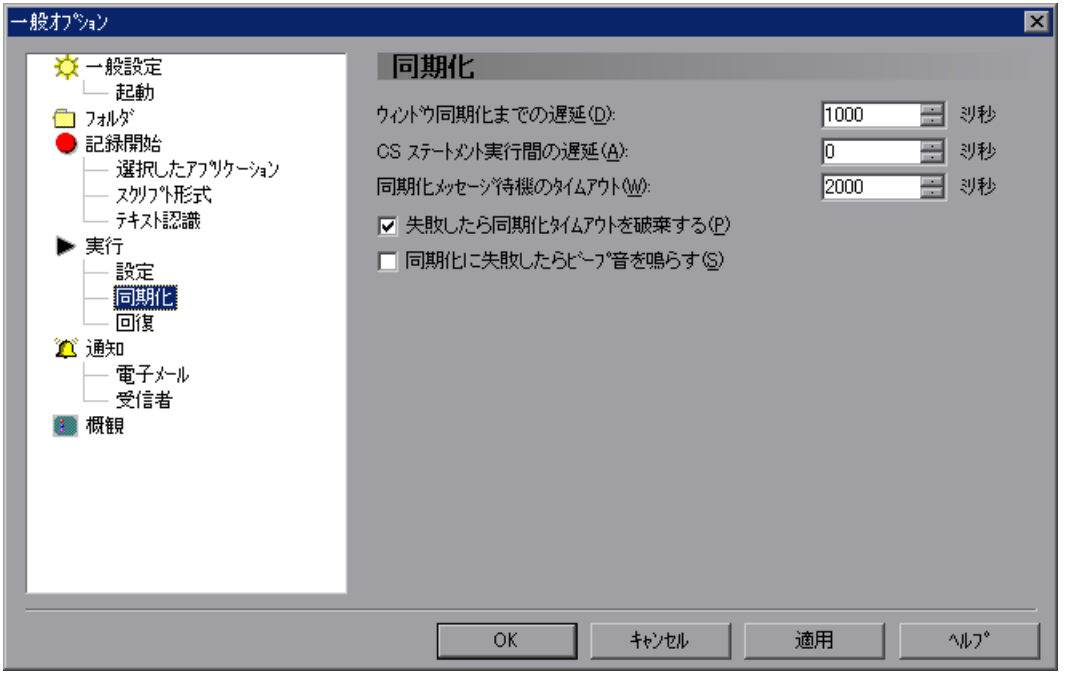
オプション	詳細
[コンテキスト センシティブ エラーが発生したらテストを失敗とする]	コンテキスト・センシティブ・エラーが発生したときに WinRunner のテストを失敗とするかどうかを指定します。テスト実行中のコンテキスト・センシティブ・エラーは，コンテキスト・センシティブ・ステートメントの失敗です。コンテキスト・センシティブ・エラーは，WinRunner が GUI オブジェクトを認識できないときに発生することがあります。 例えば，存在しないウィンドウ名で **set_window** ステートメントが含まれているテストを実行すると，コンテキスト・センシティブ・エラーが発生します。コンテキスト・センシティブ・エラーは，ウィンドウ名が不明確な場合にも発生します。コンテキスト・センシティブ関数の詳細については，「**TSL リファレンス**」を参照してください。 テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **cs_fail** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 このオプションは対応する **-cs_fail** コマンド・ライン・オプションを使用して設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。 **標準設定** = 選択されていない
[単数のプロパティ チェックが失敗したらテストを失敗とする]	**_check_info** ステートメントが失敗したときに WinRunner のテストを失敗とするかどうかを指定します。また，これらのステートメントに対応するイベントを［テスト結果］ウィンドウに書き込みます。 (**_check_info** ステートメントは［**挿入**］＞［**GUI チェックポイント**］＞［**単数のプロパティ**］コマンドを使用して作成できます)。**check_info** 関数の詳細については，「**TSL リファレンス**」を参照してください。 テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **single_prop_check_fail** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 このオプションは対応する **-single_prop_check_fail** コマンド・ライン・オプションを使用して設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。 **標準設定** = 選択されている

オプション	詳細
[検証が失敗したら停止する]	検証に失敗したとき，または，「**検証**」モードで実行中のテストでコンテキスト・センシティブ・ステートメントの結果として何らかのメッセージが生成されたときに，WinRunner がテストの実行を中断してメッセージを表示するかどうかを指定します。このオプションは対話的に作業するときにだけ使用し，バッチ・モードでは使用しないでください。 例えば，**set_window** ステートメントがテスト・スクリプトにないと，WinRunner は指定されたウィンドウを見つけることができません。このオプションが選択されている場合，WinRunner はテストを中断して［実行］ウィザードを開き，ウィンドウを探すことができるようにします。このオプションが選択されていない場合，WinRunner は［テスト結果］ウィンドウにエラーを報告し，テスト・スクリプト内の次のステートメントに進みます。 テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **mismatch_break** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 このオプションは対応する **-mismatch_break** コマンド・ライン・オプションを使用して設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。 **標準設定** = 選択されている
[検証失敗の時，ビットマップをキャプチャする]	チェックポイントが失敗するたびにアプリケーションの画像をキャプチャするよう WinRunner を設定します。ビットマップはテスト結果フォルダに保存されます。 **標準設定** = 選択されていない テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **capture_bitmap** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。
[キャプチャする領域]	チェックポイントが失敗したときにキャプチャする画面の領域を指定します。 ［**Window**］：アクティブ・ウィンドウをキャプチャします。 ［**Desktop**］：デスクトップ全体をキャプチャします。 ［**Desktop Area**］：指定されたデスクトップ領域をキャプチャします。

オプション	詳細
[デスクトップ領域の座標]	[**X**]：キャプチャする四角形領域の左上隅の x 座標。 [**Y**]：キャプチャする四角形領域の左上隅の y 座標。 [**幅**]：キャプチャする四角形の幅。 [**高さ**]：キャプチャする四角形の高さ。 ([**キャプチャする領域**]で[**デスクトップ領域の座標**]を選択した場合にのみ有効)。

実行の同期化オプションの設定

[同期化]カテゴリでは，テスト実行の同期化設定を定義します。



［**同期化**］カテゴリには次のオプションがあります。

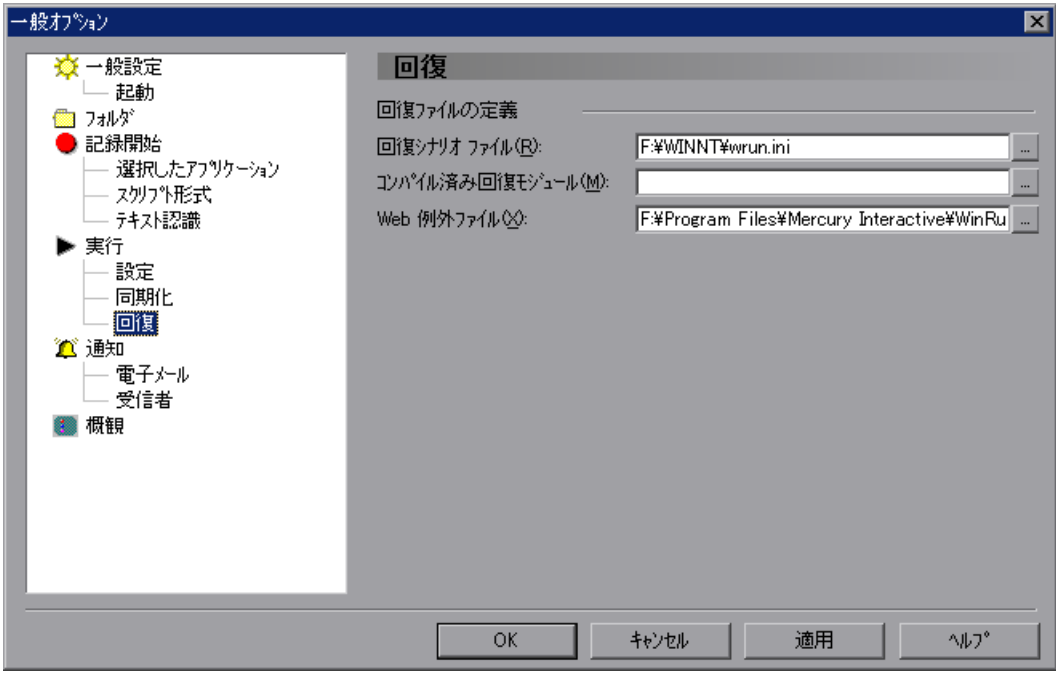
オプション	詳細
［ウィンドウの同期化までの遅延］	コンテキスト・センシティブ・チェックポイントまたは同期ポイントのためのキャプチャを行う前にウィンドウが安定したことを判定するために使われるサンプリングの間隔（単位：ミリ秒）を設定します。安定であると判定されるためには，ウィンドウが 2 回の連続するサンプリングの間で変化がないことが必要になります。このサンプリングは，ウィンドウが安定するか，またはタイムアウト（［**設定**］カテゴリの［**チェックポイントと CS ステートメントのタイムアウト**］で設定します）になるまで，続行されます。 通常は，遅延時間が短いほど，WinRunner はオブジェクトやウィンドウをより早くキャプチャでき，テストを継続できますが，一方でシステムへの負荷が高くなります。 実際のタイムアウトでは，このオプションに設定した値から 20 ～ 30 ミリ秒以内の誤差が生じる場合があります。 この設定の調整をいつ行うかについては，570 ページ「適切なタイムアウトと遅延設定の選択」を参照してください。 テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **delay_msec** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 このオプションは対応する **-delay_msec** コマンド・ライン・オプションを使用して設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。 **標準設定** = 1000 ミリ秒

オプション	詳細
[CS ステートメントの実行間の遅延]	テストの実行時に，WinRunner が各コンテキスト・センシティブ・ステートメントの実行を待機する時間（単位：ミリ秒）を設定します。 この設定の調整をいつ行うかについては，570 ページ「適切なタイムアウトと遅延設定の選択」を参照してください。 テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **cs_run_delay** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 このオプションは対応する **-cs_run_delay** コマンド・ライン・オプションを使用して設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。 **標準設定** = 0 ミリ秒
[同期化メッセージ待機のタイムアウト]	テストの実行中にキーボードまたはマウス入力が正しく行われたことを検証するまで WinRunner が待機するタイムアウト（単位：ミリ秒）を設定します。 テストの実行中に同期が頻繁に失敗する場合には，このオプションの値を大きくしてみてください。 この設定の調整をいつ行うかについては，570 ページ「適切なタイムアウトと遅延設定の選択」を参照してください。 テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **synchronization_timeout** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 **標準設定** = 2000 ミリ秒

オプション	詳細
[失敗したら同期化タイムアウトを破棄する]	最初の同期が失敗した場合に，WinRunner が（上の［**同期メッセージ待機のタイムアウト**］オプションで定義された）同期タイムアウトを最小化するかどうかを指定します。 この設定の調整をいつ行うかについては，570 ページ「適切なタイムアウトと遅延設定の選択」を参照してください。 テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **drop_sync_timeout** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 **標準設定** = 選択されている
[同期化に失敗したらビープ音を鳴らす]	同期メッセージ待機のタイムアウトが失敗するたびにビープ音を鳴らすかどうかを指定します。 このオプションは主にテスト・スクリプトをデバッグする際に使用します。 テスト実行中に同期が頻繁に失敗する場合は，［**同期化メッセージ待機のタイムアウト**］オプションの値を大きくするか，テスト・スクリプトで **setvar** 関数を使用し，対応する **synchronization_timeout** テスト・オプションの値を大きくしてみてください。 この設定の調整をいつ行うかについては，570 ページ「適切なタイムアウトと遅延設定の選択」を参照してください。 テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **sync_fail_beep** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 **標準設定** = 選択されていない

回復オプションの設定

［回復］カテゴリには，WinRunner が回復シナリオ情報および Web 例外情報を取得するために参照するファイルを指定するためのオプションがあります。

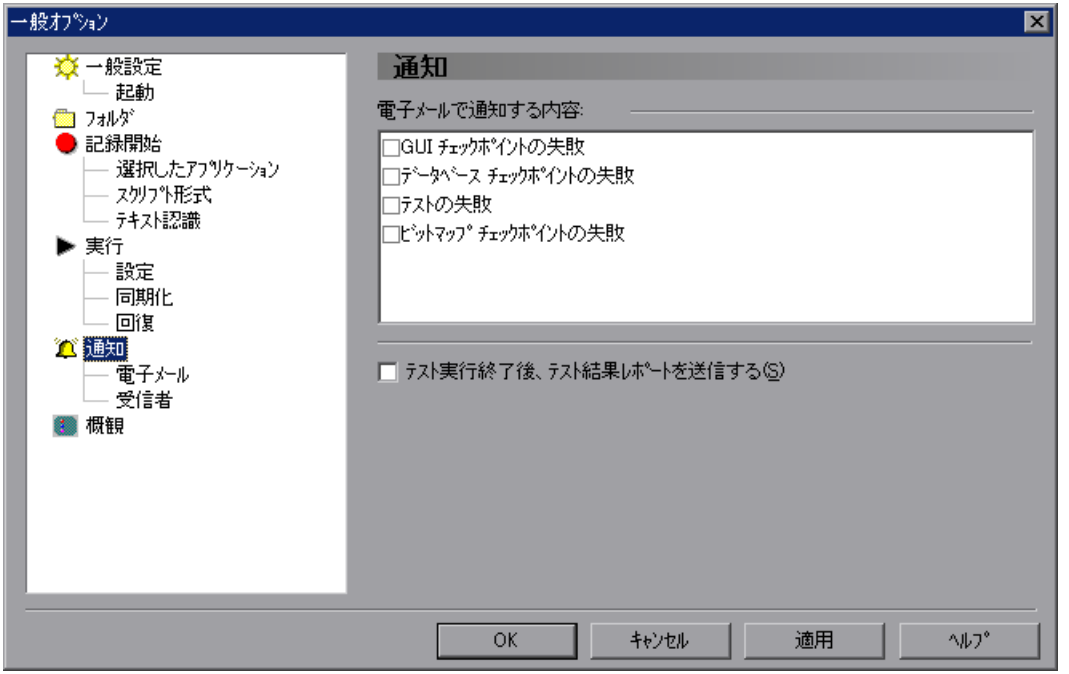


［**回復**］カテゴリには次のオプションがあります。

オプション	詳細
［回復シナリオ ファイル］	回復シナリオ・ファイルの場所を指定します。回復シナリオ・ファイルには，使用可能な回復シナリオの詳細が格納されています。Recovery Manager を使用して回復シナリオを作成または変更するには，**wrun.ini** 以外の回復シナリオ・ファイルを選択する必要があります。 回復シナリオは Recovery Manager で定義および変更します。詳細については，第 26 章「回復シナリオの定義と使用」を参照してください。 **標準設定 = ＜ Windows フォルダ＞ ¥wrun.ini**
［コンパイル済み回復モジュール］	回復コンパイル済みモジュールの場所を指定します。回復コンパイル済みモジュールは WinRunner が開くときに自動的にロードされ，回復シナリオで使用される回復関数および回復後関数が格納されています。新しいモジュール名を入力するか，既存のコンパイル済みモジュールの名前を入力します。詳細については，第 26 章「回復シナリオの定義と使用」を参照してください。 **注：** Quality Center スクリプトを回復コンパイル済みモジュールとして指定することができます。指定する場合は，［Quality Center への接続］ダイアログ・ボックスで［**起動時に再接続する**］が選択されていることを確認してください。詳細については，964 ページ「プロジェクトの接続と接続の解除」を参照してください。
［Web 例外ファイル］	Web 例外ファイルの場所を指定します。Web 例外ファイルには，使用可能な Web 例外処理の定義の詳細が格納されています。 Web 例外は，Web 例外エディタで定義および変更します。詳細については，第 32 章「Web 例外処理の定義」を参照してください。 **標準設定 = ＜ WinRunner のインストール・フォルダ＞ ¥arch¥exception.inf**

通知オプションの設定

［**通知**］カテゴリには，指定された条件に基づいて電子メール通知を送信するための条件があります。



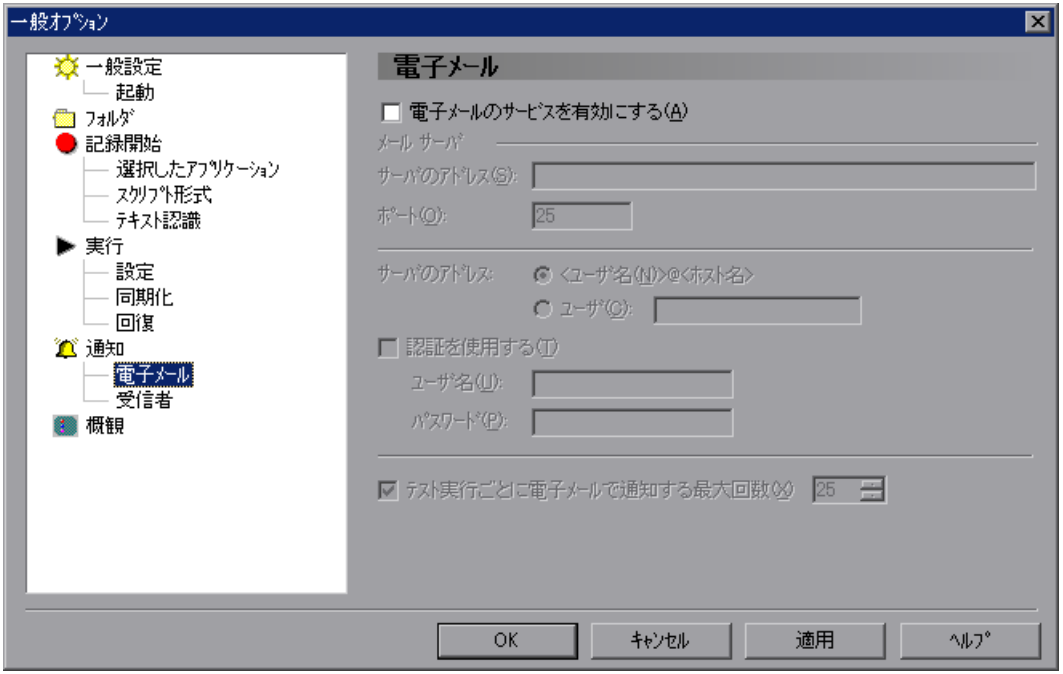
このカテゴリにあるオプションに加えて，［**電子メール**］および［**受信者**］サブカテゴリにある追加の通知オプションも設定できます。

［**通知**］カテゴリには次のオプションがあります。

オプション	詳細
[電子メールで通知する内容]	選択した条件に対して電子メールを送信します。次のいずれか 1 つ以上を選択できます。 • [**ビットマップ チェックポイントの失敗**]：ビットマップ チェックポイントが失敗するたびに，指定された受信者に電子メールを送信します。電子メールには，テストに関するサマリ詳細，チェックポイント，および，期待画像，実際の画像，差異画像の各ファイル名が含まれます。 • [**データベース チェックポイントの失敗**]：データベース チェックポイントが失敗するたびに，指定された受信者に電子メールを送信します。電子メールには，テストに関するサマリ詳細，チェックポイント，および，チェックポイントに使用された接続文字列と SQL クエリに関する詳細が含まれます。 • [**GUI チェックポイントの失敗**]：GUI チェックポイントが失敗するたびに，指定された受信者に電子メールを送信します。電子メールには，テストに関するサマリ詳細，チェックポイント，および，プロパティ検査の期待値と実際の値に関する詳細が含まれます。 • [**テストの失敗**]：テストの実行が失敗するたびに，指定された受信者に電子メールを送信します。電子メールには，テスト結果のサマリがテキスト形式で含まれます。 受信者の指定の詳細については，565 ページ「通知受信者オプションの設定」を参照してください。 **注：** 通知オプションを有効にするには，[**電子メール**] カテゴリで [**電子メールのサービスを有効にする**] オプションを選択する必要があります。 **標準設定** = すべてのチェック・ボックスが選択されていない
[テスト実行終了後，テスト結果レポートを送信する]	テストの実行が終了するたびに，指定された受信者（[**受信者**] カテゴリを参照）に電子メールを送信します。電子メールには，テスト結果のサマリがテキスト形式で含まれます。 **注：** [**テストの失敗**] に対して電子メール通知を送信することも選択していて，テストの実行が失敗した場合は，テストの失敗を知らせる電子メールだけが送信されます。 通知オプションを有効にするには，[**電子メール**] カテゴリで [**電子メールのサービスを有効にする**] オプションを選択する必要があります。 **標準設定** = 選択されている

電子メール通知オプションの設定

［**電子メール**］カテゴリには，使用するメール・サーバその他の電子メール設定を指定するためのオプションがあります。



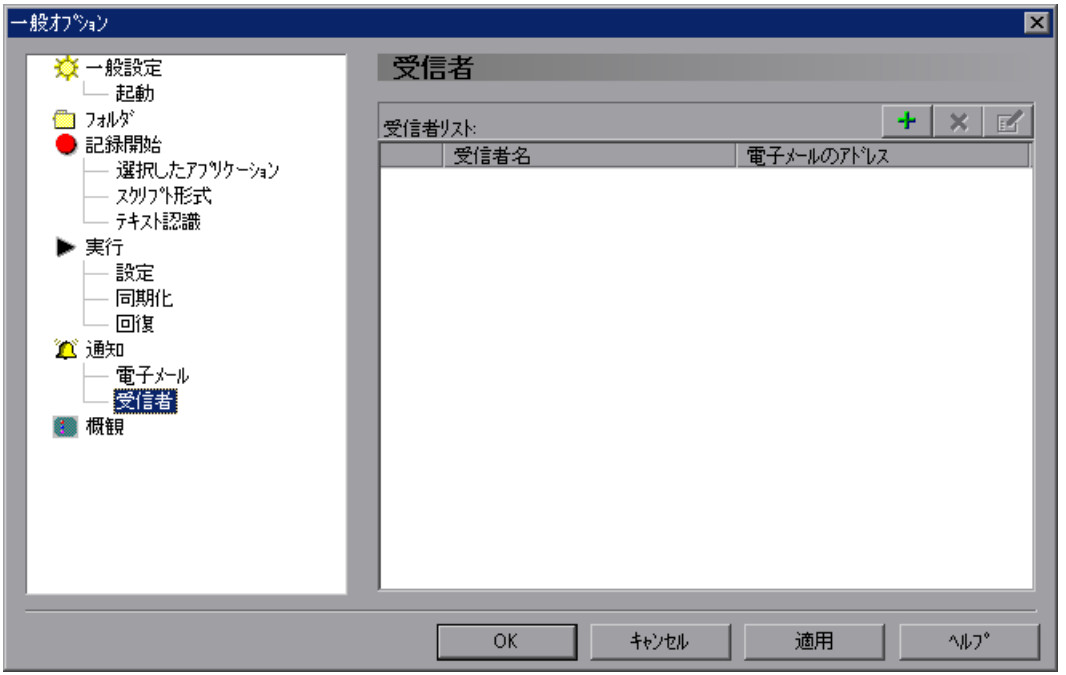
［**電子メール**］カテゴリには次のオプションがあります。

オプション	詳細
［電子メールのサービスを有効にする］	［**通知**］カテゴリで設定されている電子メール通知オプションのほか，テスト・スクリプトで **email_send_msg** 関数を使用して指定した電子メール通知オプションを有効にするよう，WinRunner を設定します。 テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **email_service** テスト・オプションの値を設定および取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 **標準設定** = 選択されていない

オプション	詳細
[サーバの アドレス]	電子メール・メッセージの送信に使用する送信用メール・サーバのアドレス。
[ポート]	使用するメール・サーバ・ポート。 **標準設定** = 25
[サーバのアドレス]	電子メール通知の送信元として表示する電子メール・アドレス。次のどちらかを選択します： • [**<ユーザ名> @ <ホスト名>**]：送信元としてテストが実行された WinRunner コンピュータのログイン名とホスト名を使用します。例えば，次のようになります。Amy@MYCOMPUTER • [**ユーザ**]：任意のテキストまたは電子メール・アドレスを送信元アドレスとして指定できます。 **注：** 多くのメール・サーバでは，送信元の名前が有効な電子メール・アドレスである必要があります。指定した送信用メール・サーバでこの条件が満たされている場合は，[ユーザ] オプションを使用して有効な電子メール・アドレスを指定してください。有効な電子メール・アドレスを必要とするサーバで有効な電子メール・アドレスを指定しなかった場合，WinRunner からは電子メールがメール・サーバに送信されますが，メール・サーバからは電子メールが受信者に送信されません。 **標準設定** = <ユーザ名> @ <ホスト名>
[認証を使用する]	送信用メール・サーバで電子メールを送信するためにログインが必要であることを示します。このオプションを選択するときは，ログインのユーザ名とパスワードを入力する必要があります。 **標準設定** = 選択されていない
[テスト実行ごとの電子メールで通知する最大回数]	テスト実行中に受信者（[**受信者**] カテゴリで指定）に送信する電子メール通知の最大数。 **注：** このオプションは，[**通知**] カテゴリで設定されたオプションに従って WinRunner が送信する電子メール・メッセージの数にのみ，適用されます。**email_send_msg** 関数を使用して送信されるメッセージについては，このオプションとは完全に独立しています。**email_send_msg** 関数の詳細については，「**TSL リファレンス**」を参照してください。 **標準設定** = 25

通知受信者オプションの設定

［**受信者**］カテゴリでは，（［**通知**］カテゴリで選択したオプションに従って）電子メール通知を受信する受信者を指定できます。

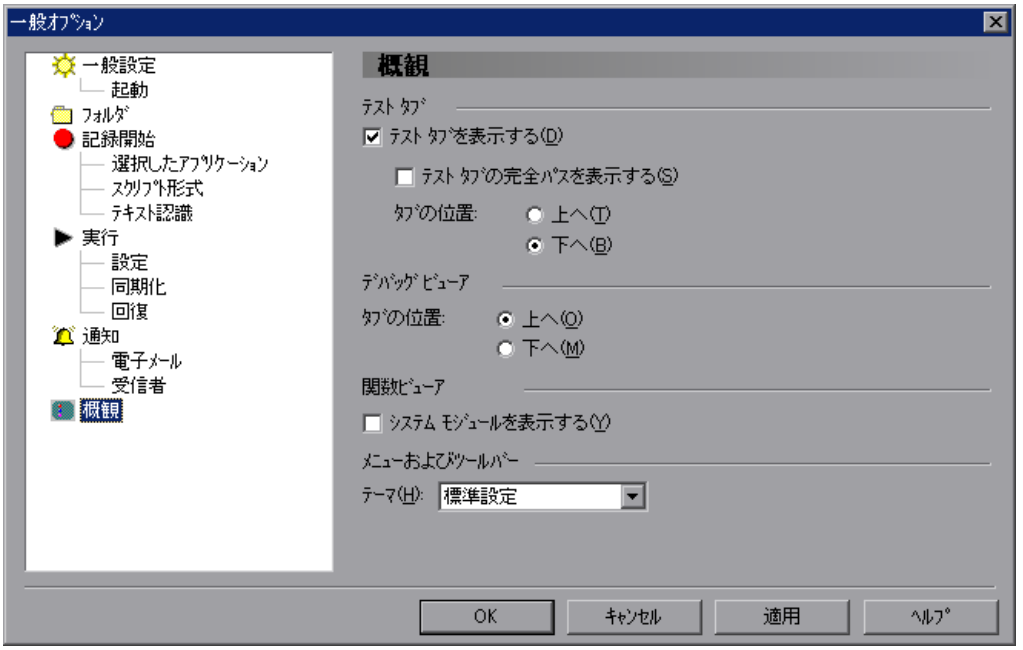


- 新しい受信者をリストに追加するには，［**受信者の追加**］ をクリックします。
- リストから受信者を削除するには，リストから受信者を選択して［**受信者の削除**］ をクリックします。
- リスト内の受信者の詳細を変更するには，リストから受信者を選択して［**受信者詳細の変更**］ をクリックします。

注：Microsoft Exchange などの一部のメール・サーバでは，設定によって，Microsoft Outlook 以外のメール・クライアントから組織外部に電子メールを送信できないことがあります。［**電子メール**］カテゴリで指定した送信用メール・サーバにおいてそのような制限が設定されている場合は，メール・サーバのドメイン名と同じドメイン名を持つ電子メール・アドレスだけを指定していることを確認してください。外部の受信者を指定した場合，WinRunner のメール・クライアントからは電子メール・メッセージがメール・サーバに送信されますが，メール・サーバからはメッセージが受信者に送信されません。ほとんどの場合，このような状況ではメール・サーバからエラー・メッセージが送信元に送られることはありません。

概観オプションの設定

［**概観**］カテゴリには，WinRunner の表示形式を制御するためのオプションがあります。



［**概観**］カテゴリには次のオプションがあります。

オプション	詳細
［テスト タブを表示する］	開いているテストごとにタブを表示します。タブをクリックすると開いているテストを表示できます。 このオプションをクリアした場合は，［**ウィンドウ**］メニュー・コマンドを使用してテストを選択し，表示できます。 **標準設定** = 選択されている
［テスト タブの完全パスを表示する］	このオプションを選択すると，テストのフル・パスが各テスト・タブに表示されます。このオプションをクリアすると，テスト名だけがタブに表示されます。 **標準設定** = 選択されていない
［タブの位置（テスト・タブ）	テスト・タブをページの［**上**］に表示するか［**下**］に表示するかを示します。
［タブの位置（デバッグ・ビューア）］	デバッグ・タブを［デバッグ・ビューア］表示枠の［**上**］に表示するか［**下**］に表示するかを示します。
［システム モジュールを表示する（関数ビューア）］	このオプションを選択すると，読み込まれているシステム・モジュールが関数ビューアに表示されます。
［テーマ］	フレーム用にあらかじめ設定されたスタイルまたは背景の画像を選択できます。詳細については，次の「テーマの選択」を参照してください。

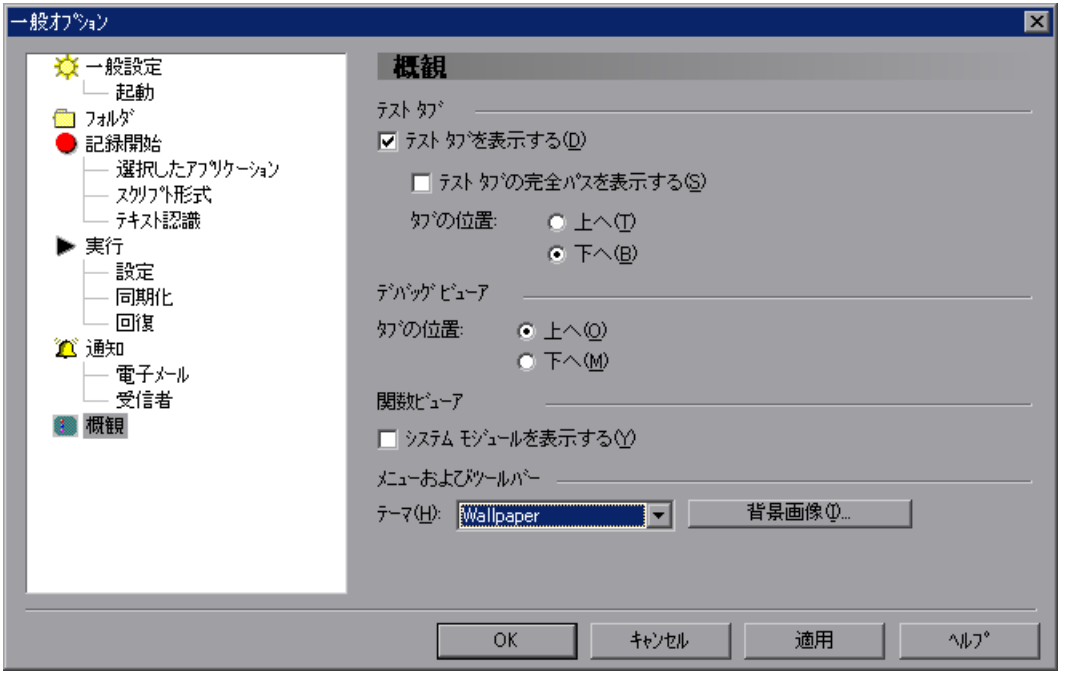
テーマの選択

フレーム用にあらかじめ設定されたスタイルを［**テーマ**］リストから選択できます。または，独自の壁紙をフレームの背景として選択することもできます。選択したテーマは WinRunner ウィンドウと［WinRunner テスト結果］ウィンドウの両方に反映されます。

注：統合レポート・ビュー内のテスト結果には，テーマは表示されません。レポート・ビューの選択の詳細については，545 ページ「テストの実行オプションの設定」を参照してください。統合レポート・ビューの詳細については，443 ページ「統一レポート・ビューの結果ウィンドウについて」を参照してください。

独自の壁紙背景を選択するには，次の手順を実行します。

1 ［**テーマ**］リストから，［**Wallpaper**］を選択します。［**背景画像**］ボタンが表示されます。



2 ［**背景画像**］をクリックします。［背景画像］ダイアログ・ボックスが開きます。

3 標準の WinRunner 背景の画像を使用するには，［**内部設定**］を選択します。独自の画像を使用するには，［**ファイルから**］を選択します。

4 手順 3 で［**ファイルから**］を選択した場合は，ファイル名を入力するか，参照ボタンを使用してビットマップ（**.bmp**）ファイルを選択します。

5 選択した画像の明るさを調整する場合は，［**明るさ**］スライダ・コントロールを使用します。

6 ［**OK**］をクリックして，［背景画像］ダイアログ・ボックスを閉じます。背景の画像は［一般オプション］ダイアログ・ボックスで［**適用**］または［**OK**］をクリックした後に初めて WinRunner のユーザ・インタフェースに表示されます。

適切なタイムアウトと遅延設定の選択

次の表は，［一般オプション］ダイアログ・ボックスで指定できるタイムアウトと遅延の設定の一覧で，各設定をどのようなときに調節するかを説明したものです。

設定	詳細	調節する場合	標準
ウィンドウの同期化のための遅延	WinRunner がウィンドウまたはオブジェクトを見つけるまでに待機する時間。この設定により，ウィンドウが安定します。	遅延時間が短いほど，WinRunner はオブジェクトやウィンドウをより早くキャプチャでき，テストを継続できますが，一方でシステムへの負荷が高くなります。多くの場合，［**チェックポイントと CS ステートメントのタイムアウト**］を変更すると，一定の比率を保持するため，［**ウィンドウの同期化のための遅延**］も変更する必要があります。システムへのオーバーロードを避けるには，タイムアウトと遅延時間の比率を 50 : 1 以上に設定しないでください。	1000 ミリ秒

設定	詳細	調節する場合	標準
チェックポイントと CS ステートメントのタイムアウト	GUI チェックポイントまたは同期化ポイントに含まれる時間パラメータに追加する時間。この時間だけ，WinRunner はオブジェクトまたはウィンドウが表示されるのを待機します。	アプリケーションがオブジェクトまたはウィンドウを正しく表示するのに現在のタイムアウトの値より長くかかる場合に，この値を増やします。ただし，1 つまたは複数のオブジェクトでこの問題が生じる場合は，スクリプトで問題のオブジェクトに同期化ポイントを追加することをお勧めします。	10000 ミリ秒
CS ステートメントの実行の間に入れる遅延	テストの実行時に，WinRunner が各コンテキスト・センシティブ・ステートメントの実行を待機する時間。	同期化に関する問題以外でテスト実行を遅らせる必要がある場合にこの遅延の値を増やします。例えば，テスト実行をステップごとに確認したい場合に，この値を増やします。	0 ミリ秒
同期化メッセージ待機のタイムアウト	テストの実行中にキーボードまたはマウス入力が正しく行われたことを検証するまで WinRunner が待機するタイムアウト（単位：ミリ秒）を設定します。	アプリケーションのステートメント実行よりも速く WinRunner がスクリプトを実行する場合に，この値を増やします。	2000 ミリ秒

設定	詳細	調節する場合	標準
失敗したら同期化タイムアウトを破棄	同期の検証が失敗すると，［**同期化メッセージ待機のタイムアウト**］の設定の長さを自動的に最小化します。これにより，マウスまたはキーボードによる入力が完了しないと，テストがすぐに失敗します。	テストが不正確なマウスまたはキーボード入力による間違ったデータで長い間実行されるのを防ぐためにこのオプションを選択します。	選択
同期化に失敗したらビープ音を鳴らす	WinRunner は，［**同期化メッセージ待機のタイムアウト**］の設定を超えるたびにビープ音を鳴らします。	スクリプトのデバッグ中にこのオプションを選択することをお勧めします。1 回のテスト実行で何回もビープ音が鳴る場合は，［**同期化メッセージ待機のタイムアウト**］の値を増やします。	選択されていない

設定	詳細	調節する場合	標準
テキスト認識のタイムアウト	テストの実行中に通常のテキスト認識機能を使ってテキスト・チェックポイントを実行する場合に，WinRunner がテキストが認識するまでに待機する時間。	通常のテキスト認識機能を使ったテキスト・チェックポイントが失敗した場合は，タイムアウトの値を増やしてみてください。またはイメージ・テキスト認識機能を使ってみてください。あるいは，テキスト認識をまったく使用しない代わりのテキスト検査方法を使用することもできます。詳細については，543 ページ「Windows アプリケーションに対してテキスト認識を使用する際の注意事項」を参照してください。	500 ミリ秒

第 6 部

GUI マップを使った作業

第 23 章

GUI マップ・ファイルのマージ

本章では，GUI マップ・ファイルのマージ方法について説明します。本章は，［**テストごとの GUI マップファイル**］モードから［**グローバル GUI マップファイル**］モードに移行するときに特に役立ちます。また，［**グローバル GUI マップファイル**］モードでの作業中に GUI マップ・ファイルを合成する場合にも役立ちます。

本章では，以下の項目について説明します。

- GUI マップ・ファイルのマージについて
- GUI マップ・ファイルのマージの準備
- GUI マップ・ファイルを自動的にマージするときの競合の解決
- 手動による GUI マップ・ファイルのマージ
- ［テストごとの GUI マップファイル］モードへの変更

GUI マップ・ファイルのマージについて

［**テストごとの GUI マップファイル**］モードで作業しているときは，ユーザが作成した各テストに対して GUI マップ・ファイルが自動的に作成，保存，ロードされます。WinRunner に慣れていない場合は，このモードを使用するほうが簡単です。しかし，これは最も効率的なモードではありません。WinRunner に慣れてきたら，［**グローバル GUI マップファイル**］モードに変更することをお勧めします。このモードでは，複数のテストによって参照される 1 つの GUI マップに，アプリケーションの GUI に関する情報を保存できるので，より効率的です。アプリケーションが変更されたときには，各テストを個別に更新するのではなく，テスト・グループ全体によって参照される GUI マップを更新するだけで済みます。

［GUI マップファイルの結合ツール］を使用すると，複数の GUI マップ・ファイルを 1 つの GUI マップ・ファイルにマージできます。GUI マップ・ファイルをマージする前に，ソースの GUI マップ・ファイルを少なくとも 2 つ指定し，ターゲット・ファイルとして GUI マップ・ファイルを少なくとも 1 つ指定する必要があります。ターゲットの GUI マップ・ファイルは，既存のファイルでも，新しい（空の）ファイルでもかまいません。

このツールは，自動モード，手動モードのどちらでも使用できます。

- 自動モードでは，ファイルが自動的にマージされます。マージされるファイル間で競合が生じた場合は，その競合が強調表示され，それを解決することが求められます。
- 手動モードでは，ターゲット・ファイルに GUI オブジェクトを手作業で追加する必要があります。ファイル間の競合は強調表示されません。

いずれのモードでも，マージ・ツールは，ファイルのマージ中に競合が生じることを防ぎます。

GUI マップ・ファイルをマージしたら，GUI マップ・ファイル・モードを変更し，適切な GUI マップ・ファイルをロードできるように，テストまたは起動テストを修正する必要があります。

GUI マップ・ファイルのマージの準備

GUI マップ・ファイルをマージする前に，ファイルをマージするモードを決定し，ソース・ファイルとターゲット・ファイルを指定する必要があります。

GUI マップ・ファイルのマージを開始するには，次の手順を実行します。

1 ［**ツール**］＞［**GUI マップファイルの結合**］を選択します。

WinRunner のメッセージ・ボックスが表示され，開いているすべての GUI マップが閉じられ，未保存の変更が破棄されることが通知されます。

2 続行するには，［**OK**］をクリックします。

開いている GUI マップに対する変更を保存するには，［**キャンセル**］をクリックし，［GUI マップエディタ］を使って GUI マップを保存します。GUI マップ・ファイルの保存の詳細については，90 ページ「GUI マップへの変更の保存」を参照してください。開いている GUI マップ・ファイルに対する変更を保存したら，手順 1 から再開します。

［GUI マップファイルの結合ツール］が開いたら，結合タイプを選択し，ターゲット・ファイルとソース・ファイルを指定します。

3 ［**結合タイプ**］ボックスでは，［**自動結合**］をそのまま使用するか，［**手動の結合**］を選択します。

- ［**自動結合**］を選択すると，ファイルが自動的にマージされます。マージされるファイルの間で競合が生じた場合は，その競合が強調表示され，それを解決するように求められます。
- ［**手動の結合**］を選択すると，ソース・ファイルの GUI オブジェクトをターゲット・ファイルに手作業で追加できます。ファイル間の競合は強調表示されません。

4 ターゲットの GUI マップ・ファイルを指定するには，［**結合先のファイル**］の横にある参照ボタンをクリックします。［GUI ファイルを保存］ダイアログ・ボックスが開きます。

- 既存の GUI マップ・ファイルを選択するには，ファイルを見つけて強調表示させます。これにより，そのファイル名が［ファイル名］ボックスに表示されます。ファイルを置換するかどうかを尋ねられたら，［**OK**］をクリックします。
- 新しい（空の）GUI マップ・ファイルを作成するには，任意のフォルダを見つけ，［**ファイル名**］ボックスに新しいファイル名を入力します。

5 ソースの GUI マップ・ファイルを指定します。

- フォルダ内のすべての GUI マップ・ファイルをソース・ファイルのリストに追加するには，［**フォルダの参照**］ボタンをクリックします。［フォルダの設定］ダイアログ・ボックスが開きます。任意のフォルダを見つけ，［**OK**］をクリックします。フォルダ内のすべての GUI マップ・ファイルが，ソース・ファイルのリストに追加されます。
- ソース・ファイルのリストに 1 つの GUI マップ・ファイルを追加するには，［**ファイルの追加**］ボタンをクリックします。［GUI ファイルを開く］ダイアログ・ボックスが開きます。任意のファイルを見つけて強調表示させます。これにより，そのファイル名が［ファイル名］ボックスに表示されます。［**OK**］をクリックします。
- リストからソース・ファイルを削除するには，［**ソースファイル**］ボックス内の GUI マップ・ファイルを強調表示させ，［**削除**］をクリックします。

6 ［**OK**］をクリックし，ダイアログ・ボックスを閉じます。

- ［**自動結合**］を選択し，ソースの GUI マップ・ファイルが競合することなく正常にマージされる場合，マージを確認するメッセージが表示されます。
- ［**自動結合**］を選択し，マージするソースの GUI マップ・ファイル間に競合が生じた場合，その問題を警告するメッセージ・ボックスが表示されます。［OK］をクリックすると，メッセージ・ボックスが閉じ，［GUI マップファイルの自動結合ツール］が開きます。詳細については，580 ページ「GUI マップ・ファイルを自動的にマージするときの競合の解決」を参照してください。
- ［**手動の結合**］を選択すると，［GUI マップファイル手動結合ツール］が開きます。詳細については，584 ページ「手動による GUI マップ・ファイルのマージ」を参照してください。

GUI マップ・ファイルを自動的にマージするときの競合の解決

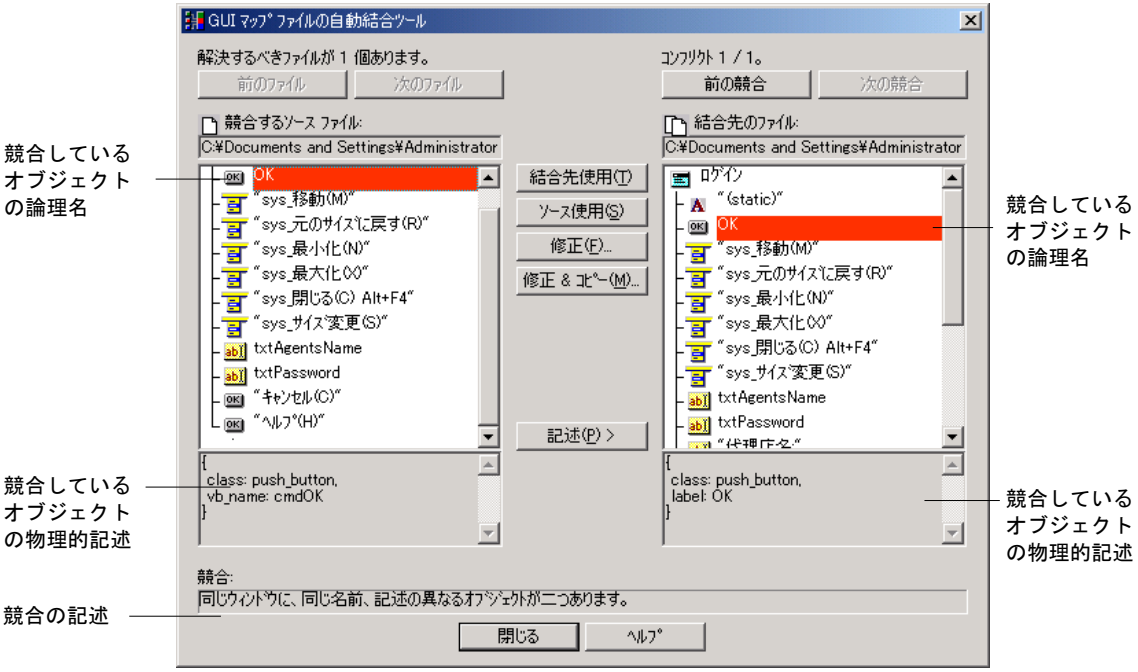
［GUI マップファイルの結合ツール］で［**自動結合**］オプションを選択し，ファイル間で競合が生じなかった場合，マージを確認するメッセージが表示されます。

GUI マップ・ファイルを自動的にマージする場合，次の状況で競合が生じます。

- 2 つのウィンドウが同じ名前を持ち，その物理的記述が異なる場合。

➤ 同じウィンドウにある 2 つのオブジェクトが同じ名前を持ち，その物理的記述が異なる場合。

次の例では，競合している 2 つのソース・ファイル（**before.gui** と **after.gui**）を新しいターゲット・ファイル（**new.gui**）に自動的にマージしています。578 ページ「GUI マップ・ファイルのマージの準備」の説明に従って，競合警告メッセージ・ボックスで［OK］をクリックすると，［GUI マップファイルの自動結合ツール］が開きます。このダイアログ・ボックスでは，競合を解決し，ターゲット・ファイルで新たに競合が生じることを防ぎます。



競合しているオブジェクトは赤で強調表示され，競合の記述がダイアログ・ボックスの下部にある表示枠に表示されます。両方の GUI マップ・ファイルに，同じウィンドウの同名のオブジェクトがあり，その物理的記述が異なっているため，これらのファイルは競合します（最初，*new.gui* ターゲット・ファイルは空だったため，**after.gui** ソース・ファイルのウィンドウとオブジェクトは，競合することなく *new.gui* にコピーされました）。競合しているオブジェクトの名前は，赤で表示されます。

ソース・ファイルは，578 ページ「GUI マップ・ファイルのマージの準備」で説明されているように，［GUI マップファイルの結合ツール］に表示される順序でマージされます。

競合しているオブジェクトまたはウィンドウの物理的記述を表示するには，［**記述**］をクリックします。

強調表示される競合は，次のいずれかのボタンをクリックすることで解決できます。競合しているオブジェクトまたはウィンドウが両方の表示枠で強調表示されている場合にのみ，これらのボタンは有効になります。

競合を解決するための オプション	説明
［**結合先使用**］	ターゲットの GUI マップ・ファイルにあるオブジェクトまたはウィンドウの名前と物理的記述を使用することで，競合を解決します。
［**ソース使用**］	ソースの GUI マップ・ファイルにあるオブジェクトまたはウィンドウの名前と物理的記述を使用することで，競合を解決します。
［**修正**］	ターゲットの GUI マップ・ファイルにあるオブジェクトまたはウィンドウの物理的記述に対して，ターゲットとソースの両方のオブジェクトまたはウィンドウを正確に記述する正規表現を指定することで（可能な場合），競合を解決します。この記述は修正できます。
［**修正＆コピー**］	ソースの GUI マップ・ファイルにあるオブジェクトまたはウィンドウの物理的記述を編集し，ターゲットの GUI マップ・ファイルに貼り付けることで，競合を解決します。 **注：** 物理的記述に対する変更は，ソースの GUI マップ・ファイルには保存されません。

ヒント：

競合しているソース・ファイルが複数ある場合，［**前のファイル**］または［**次のファイル**］をクリックすると，現在の GUI マップ・ファイルを切り替えることができます。

1 つのソース・ファイルに，競合しているオブジェクトが複数ある場合，［**前の競合**］または［**次の競合**］をクリックすると，強調表示されている競合を切り替えることができます。マウスを使って，ターゲット・ファイル内の競合していないオブジェクトを強調表示させた場合（オブジェクトの物理的記述を確認するときなど），およびターゲット・ファイルで競合が強調表示されていない場合は，［**前の競合**］をクリックすると，競合しているオブジェクトが強調表示されます。

現在のソース・ファイルとターゲット・ファイルの間にある競合がすべて解決されると，ソース・ファイルが自動的に閉じ，次の競合しているソース・ファイルが開きます。GUI マップ・ファイル間の競合がすべて解決されると，残りのソース・ファイルとターゲット・ファイルが閉じ，［GUI マップファイルの自動結合ツール］が閉じます。

ヒント：別のソース・ファイル内の競合を解決した結果，現在のソース・ファイルに含まれるすべての競合が解決されることがあります。例えば，現在のオブジェクト以外のオブジェクトとの競合に使用されるターゲット・ファイル内のオブジェクトを変更した場合などです。この場合，［ファイルの削除］ボタンが表示されます。このボタンをクリックすると，ソースの GUI マップ・ファイルのリストから現在のソース・ファイルが削除されます。

注：ターゲットの GUI マップ・ファイルに対する変更は，自動的に保存されます。

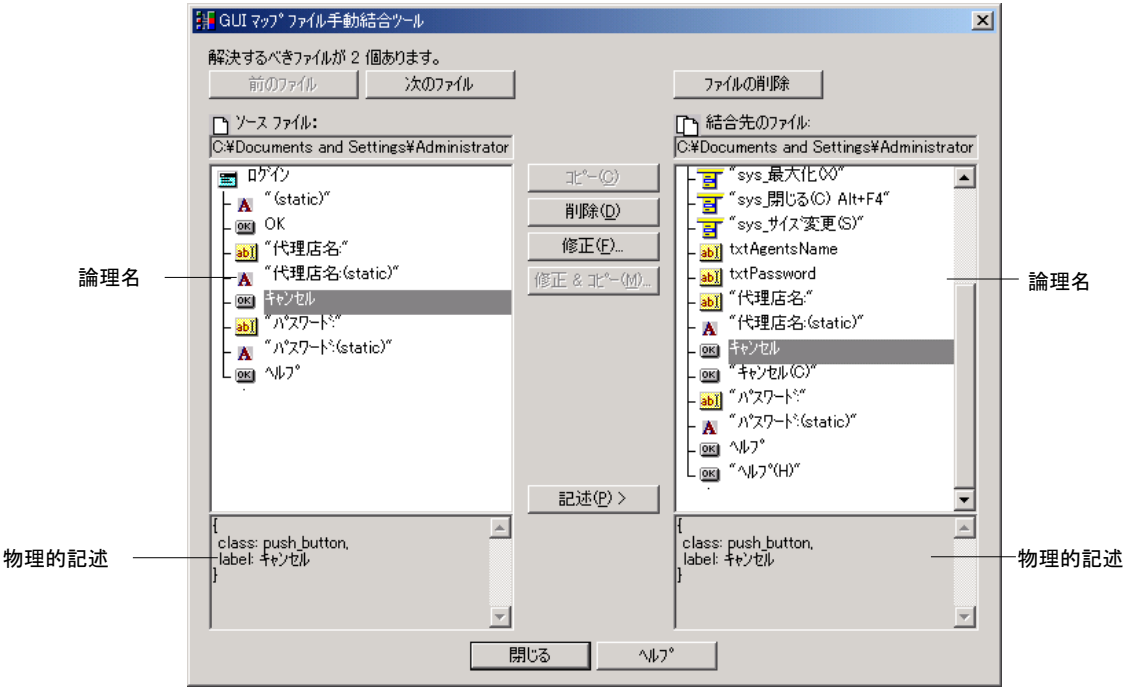
手動による GUI マップ・ファイルのマージ

GUI マップ・ファイルを手動でマージする場合，各ターゲット・ファイルをソース・ファイルとマージします。マージ・ツールが，ファイルのマージ中に競合が生じることを防ぎます。

GUI マップ・ファイルを手動でマージする場合，ターゲットの GUI マップ・ファイルは，次のものを含むことができません。

- 名前が同じで，物理的記述が異なる 2 つのウィンドウ。
- 名前も物理的記述も同じである 2 つのウィンドウ（同一のウィンドウ）。
- 同じウィンドウにあり，名前が同じで，物理的記述が異なる 2 つのオブジェクト。
- 同じウィンドウにあり，名前も物理的記述も同じである 2 つのオブジェクト（同一のオブジェクト）。

次の例では，*after.gui* ソース・ファイルのすべての内容が，*new.gui* ターゲット・ファイルにコピーされ，**before.gui** ソース・ファイルとターゲット・ファイルの間に競合が生じています。



上の例では，両方の表示枠で強調表示されているオブジェクトは同じ論理名を持っていますが，物理的記述は異なります。したがって，これらのオブジェクトはこのままでは，マージされたファイル内に存在できません。

GUI マップ・ファイルを手動でマージするには，次の手順を実行します。

1 578 ページ「GUI マップ・ファイルのマージの準備」で説明されている手順に従って，結合タイプとして［**手動結合**］を選択します。ソース・ファイルとターゲット・ファイルを指定し，［**OK**］をクリックすると，［GUI マップファイル手動結合ツール］が開きます。

ソース・ファイルとターゲット・ファイルの内容が表示されます。

2 マージするウィンドウまたはオブジェクトを検索します。

➤ ウィンドウ内のオブジェクトを表示するには，ウィンドウをダブルクリックします。

- ソース・ファイルが複数ある場合，［**前のファイル**］または［**次のファイル**］をクリックすると，現在の GUI マップ・ファイルを切り替えることができます。
- 強調表示されたオブジェクトまたはウィンドウの物理的記述を表示するには，［**記述**］をクリックします。

3 次のマージ・オプションを使って，ファイルをマージします。

マージ・オプション	説明
［**コピー**］（現在のソース・ファイル内のオブジェクトまたはウィンドウが強調表示されている場合にのみ有効）	ソース・ファイルで強調表示されているオブジェクトまたはウィンドウを，ターゲット・ファイルで強調表示されているウィンドウまたは強調表示されているオブジェクトの親ウィンドウにコピーします。 **注：** ウィンドウをコピーすると，そのウィンドウ内のオブジェクトもすべてコピーされます。
［**削除**］（ターゲット・ファイル内のオブジェクトまたはウィンドウが強調表示されている場合にのみ有効）	ターゲットの GUI マップ・ファイルから，強調表示されているオブジェクトまたはウィンドウを削除します。 **注：** ウィンドウを削除すると，そのウィンドウ内のオブジェクトもすべて削除されます。
［**修正**］（**ターゲット・ファイル内のオブジェクト**またはウィンドウが強調表示されている場合にのみ有効）	［修正］ダイアログ・ボックスが開きます。このダイアログ・ボックスでは，ターゲット・ファイルで強調表示されているオブジェクトまたはウィンドウの論理名と物理的記述を変更できます。
［**修正＆コピー**］（現在のソース・ファイル内のオブジェクトまたはウィンドウが強調表示されている場合にのみ有効）	［修正］ダイアログ・ボックスが開きます。このダイアログ・ボックスでは，ソース・ファイルで強調表示されているオブジェクトまたはウィンドウの論理名と物理的記述を変更し，ターゲット・ファイルで強調表示されているウィンドウまたは強調表示されているオブジェクトの親ウィンドウにコピーできます。 **注：** 物理的記述に対する変更は，ソースの GUI マップ・ファイルには保存されません。

注：ターゲットの GUI マップ・ファイルに対する変更は，自動的に保存されます。

ヒント：ソース・ファイルのマージが終了したら，［**ファイルの削除**］をクリックすると，マージするソース・ファイルのリストから，そのファイルが削除されます。

ソース・ファイルが複数ある場合，［**前のファイル**］または［**次のファイル**］をクリックすると，現在の GUI マップ・ファイルを切り替えることができます。

［テストごとの GUI マップファイル］モードへの変更

［**テストごとの GUI マップファイル**］モードから［**グローバル GUI マップファイル**］モードに変更するための準備作業の中で最も複雑なのは，前述した通り GUI マップ・ファイルのマージです。

また，次の変更も行う必要があります。

- GUI マップ・ファイルをロードできるように，テストまたは起動テストを変更します。GUI マップ・ファイルのロードの詳細については，59 ページ「GUI マップ・ファイルのロード」を参照してください。
- ［一般オプション］ダイアログ・ボックスの［**一般設定**］カテゴリの［**GUI マップ ファイル**］セクションで［**グローバル GUI マップファイル**］を選択する必要があります。

WinRunner を閉じると，設定の変更を保存するかどうかを尋ねられます。［**はい**］をクリックします。

注：この変更を有効にするには，WinRunner を再起動する必要があります。

［一般オプション］ダイアログ・ボックスの詳細については，第 22 章「［一般オプション］ダイアログ・ボックスでのグローバル・テスト・オプションの設定」を参照してください。

- GUI マップ・ファイル・モードを切り替えたら，GUI マップ・ファイルに対して行った変更を保存することを忘れないでください。詳細については，57 ページ「GUI マップの保存」を参照してください。

第 24 章

GUI マップの構成設定

本章では，コンテキスト・センシティブ・モードでのテストにおける WinRunner の GUI オブジェクトの識別方法を変更する方法を解説します。

本章では，以下の項目について説明します。

- GUI マップの構成設定について
- 標準の GUI マップの構成設定について
- 標準クラスへのユーザ定義オブジェクトのマッピング
- 標準またはユーザ定義クラスの構成設定
- 恒久的な GUI マップの構成設定の作成
- ユーザ定義クラスの削除
- WinRunner のオブジェクト・クラスについて
- オブジェクトのプロパティについて
- 学習対象プロパティについて
- Visual Basic のオブジェクトのプロパティ
- PowerBuilder のオブジェクトのプロパティ

GUI マップの構成設定について

テスト対象アプリケーションの各 GUI オブジェクトは，class，label，MSW_class，MSW_id，x（座標軸），y（座標軸），width，height など，複数のプロパティによって定義されます。WinRunner は，コンテキスト・センシティブ・テストの際に，これらのプロパティを使ってアプリケーションの GUI オブジェクトを識別します。

WinRunner は，GUI オブジェクトの記述を学習する際に，すべてのプロパティを学習するわけではありません。オブジェクトを一意に識別するための必要最低限のプロパティを学習します。WinRunner は，各オブジェクト・クラス（push_button，list，window，menu など）ごとに異なる，標準のプロパティのセットを学習します。このセットを GUI マップ構成設定といいます。

例えば，標準のプッシュ・ボタンは，26 のプロパティ（MSW_class，label，text，nchildren，x，y，height，class，focused，enabled など）によって定義されます。しかし，WinRunner はほとんどの場合，*class* と *label* プロパティだけあれば，プッシュ・ボタンを一意に識別できます。オブジェクト・クラスに定義されたプロパティでは，特定のオブジェクトの一意の記述を作成するのに十分でない場合もあります。このような場合，WinRunner は定義済みのプロパティ・セットとセレクタ・プロパティを学習します。セレクタ・プロパティは，各オブジェクトに，一意の記述を持つ他のオブジェクトと比較したオブジェクトの位置に基づいて序数値を割り当てます。

オブジェクト・クラスに学習される標準のプロパティ・セットがアプリケーションに適していない場合は，GUI マップを設定して，そのクラスの別のプロパティ・セットを学習できます。例えば，編集ボックスの標準設定のプロパティの 1 つが *attached_text* プロパティであるとします。アプリケーションに添付テキストのプロパティを持たない編集ボックスが含まれていると，記録の際に WinRunner は編集ボックスの近くの別のオブジェクトの添付テキストのプロパティをキャプチャし，その値をオブジェクト記述の一部として保存する場合があります。このような場合は，学習対象プロパティの標準セットから *attached_text* プロパティを削除し，別のプロパティを代わりに追加します。

クラスに使用されているセレクタの種類または使用されている記録方式を変更することもできます。

多くのアプリケーションには，ユーザ定義の GUI オブジェクトも含まれます。ユーザ定義オブジェクトとは，WinRunner が使用する標準クラスのいずれにも属さない任意のオブジェクトです。したがって，これらのオブジェクトは，「object」という汎用のクラスに割り当てられます。ユーザ定義オブジェクトに対する操作を記録すると，WinRunner はテスト・スクリプトに **obj_mouse_** ステートメントを生成します。

ユーザ定義オブジェクトが標準オブジェクトに似ている場合は，標準クラスにマップできます。コンテキスト・センシティブ・テスト中にカスタム・オブジェクトを識別するために WinRunner が使用するプロパティを設定することもできます。設定したマッピングと構成は，現在の WinRunner セッションにのみ有効です。

マッピングや構成を恒久的なものにしたい場合は，構成用のステートメントを初期化テスト・スクリプトに追加しなくてはなりません。WinRunner を起動するたびに，初期化テストがこの構成を有効にします。

注：アプリケーションにオーナー描画ユーザ定義ボタンがある場合は，各ボタンをそれぞれマッピングしなくても，すべてのボタンを標準ボタン・クラスの 1 つにマッピングできます。これには，［一般オプション］ダイアログ・ボックスの［**記録開始**］カテゴリの［ユーザ定義ボタンの記録方法］ボックスで標準ボタン・クラスを選択する方法とテスト・スクリプト内で setvar 関数を使って rec_owner_drawn テスト・オプションを設定する方法があります。［一般オプション］ダイアログ・ボックスの詳細については，第 22 章「グローバル・テスト・オプションの設定」を，setvar 関数でのテスト・オプションの設定については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

オブジェクトのプロパティは，プロパティごとに移植のしやすさが異なります。MSW_class，MSW_id などは移植できません（特定のプラットフォームに固有のものがあります）。handle，Tookit_class などのように，移植が半ば可能なものもあります（複数のプラットフォームでサポートされているが，値が変わる可能性のあるもの）。他は完全に移植が可能です（label，attached_text，enabled，focused，parent など）。

非標準 Windows オブジェクトの設定について：GUI Map 構成設定ツールを使用して，WinRunner が標準 Windows オブジェクト，ActiveX および Visual Basic コントロール，PowerBuilder オブジェクト，およびその他の Web オブジェクトなど，ウィンドウ・ハンドル（HWND）を持つオブジェクトを認識する方法を変更できます。GUI マップ構成設定ツールをサポートする Web オブジェクトについては，第 10 章「Web オブジェクトでの作業」を参照してください。
WinRunner アドインを使用して他のオブジェクトをテストしている場合は，**set_record_attr** や **set_record_method** などの GUI マップ構成設定関数を使用できます。これらの関数については，「**TSL リファレンス**」を参照してください。
アドインには，WinRunner が特定のツールキットのオブジェクトを認識する方法を設定するための独自のツールを持つものもあります。詳細については，お使いの WinRunner アドインの「**最初にお読みください**」を参照してください。

標準の GUI マップの構成設定について

WinRunner は各クラスごとに 1 組の標準のプロパティを学習します。各標準のプロパティは「必須」または「オプション」のどちらかに分類されます（標準プロパティの一覧は，605 ページ「オブジェクトのプロパティについて」を参照してください）。

- 「*必須*」プロパティは（存在する場合は）必ず学習されるプロパティです。
- 「*オプション*」プロパティは，オブジェクトが必須プロパティによって識別できない場合にだけ使用されます。これらのオプション・プロパティは，リストに格納されています。WinRunner はそのリストからオブジェクトを識別するのに必要最少限の数のプロパティを選び出します。WinRunner はリストの先頭にあるプロパティから選び始めます。そのプロパティだけではオブジェクトが一意に識別できない場合は，オプション・リストの 2 番めのプロパティが記述に追加され，オブジェクトが一意に識別できるまで続けられます。

［GUI スパイ］を使って［OK］ボタンのプロパティを調べてみると，WinRunner が class と label プロパティを学習することがわかります。そのため，このボタンの物理的記述は次のようになります。

```
{class:push_button, label:"OK"}
```

必須プロパティとオプション・プロパティだけでオブジェクトを一意に識別できない場合，WinRunner は「*セレクタ*」を使用します。例えば，あるウィンドウで同じ MSW_id をもつ［OK］ボタンが 2 つ存在する場合，WinRunner は 2 つを識別するためにセレクタを使います。セレクタには，2 つの種類があります。

- 「**位置（location）**」セレクタは，オブジェクトの物理的な配置を利用します。
- 「**インデックス（index）**」セレクタは，ウィンドウ内のオブジェクトを識別するのに一意の番号を使います。

「**位置（location）**」セレクタでは，同じ記述のオブジェクトを識別するためにオブジェクトの配置の順番（左上角から下右角に向かう順番で）を利用します。

「**インデックス（index）**」セレクタでは，ウィンドウのオブジェクトを識別するためにオブジェクトの作成時に割り当てた番号を使います。ウィンドウ内の同じ記述を持つオブジェクトの配置が変わる可能性のある場合は，インデックス・セレクタを使用します。詳細については，596 ページ「標準またはユーザ定義クラスの構成設定」を参照してください。

標準クラスへのユーザ定義オブジェクトのマッピング

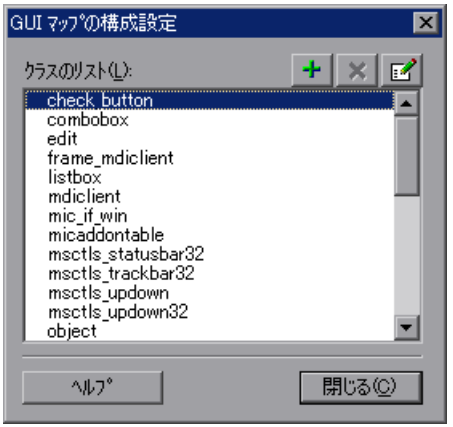
ユーザ定義オブジェクトとは，WinRunner が使用する標準のクラスに属さない GUI オブジェクトを指します。WinRunner は，このようなオブジェクトを汎用の「object」クラスとして学習します。WinRunner は，このようなオブジェクトに対する操作を，**obj_mouse_** ステートメントとして記録します。

［GUI マップの構成設定］ダイアログ・ボックスを使用して，WinRunner にユーザ定義オブジェクトを学習させたり，標準のクラスにマッピングすることもできます。例えば，アプリケーションが WinRunner が識別できないようなユーザ定義ボタンを使用している場合，このボタンをクリックする操作は，**obj_mouse_click** として記録されます。この「SampleCustomButtonClass」ユーザ定義クラスを WinRunner に学習させて，標準の push_button クラスにマッピングできます。そうしておけば，このボタンを押す操作は，**button_press** として記録されるようになります。

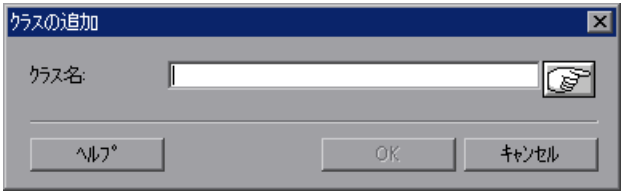
ユーザ定義オブジェクトは，同等の動作をする標準のクラスにしか割り当てられない点に注意してください。例えば，ユーザ定義のプッシュ・ボタンを edit クラスにマッピングすることはできません。

ユーザ定義オブジェクトを標準のクラスにマッピングするには，次の手順を実行します。

1 ［**ツール**］＞［**GUI マップ構成**］を選択して，［GUI マップの構成設定］ダイアログ・ボックスを開きます。［クラスのリスト］には，WinRunner が識別できるすべての標準クラスとユーザ定義クラスが表示されます。



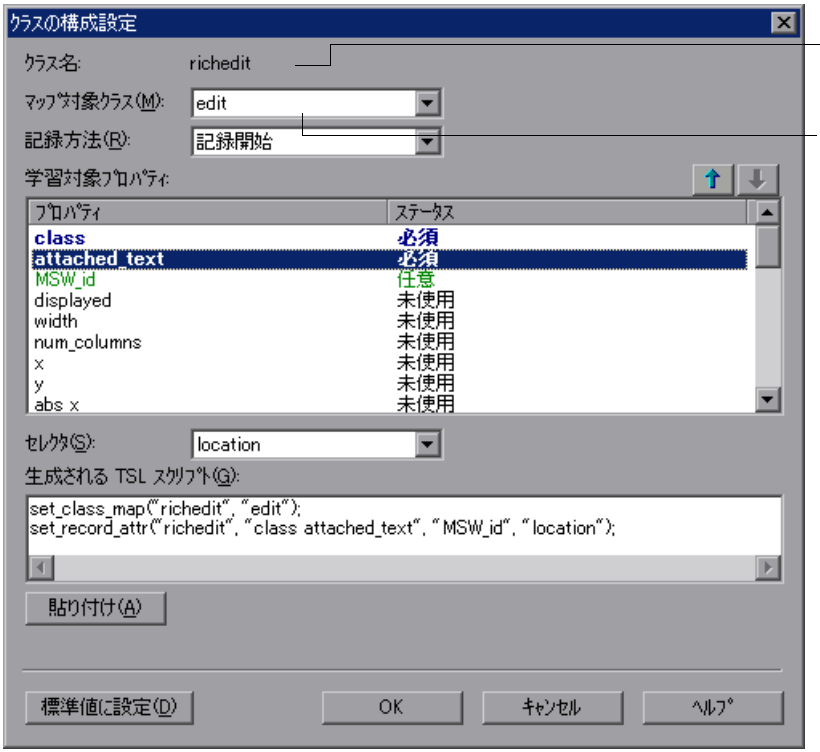
2 ［**追加**］をクリックして［クラスの追加］ダイアログ・ボックスを開きます。



3 指差しボタンをクリックしたら，追加するクラスのオブジェクトをクリックします。ユーザ定義オブジェクトの名前が［クラス名］ボックスに表示されます。この名前は，オブジェクトの MSW_class プロパティの値です。

4 ［OK］をクリックしてダイアログ・ボックスを閉じます。追加した新しいクラスが［クラスのリスト］に選択された状態で表示され，名前の手前に「U」という文字が付加されます（ユーザ定義であることを示します）。

5［**構成設定**］ボタンをクリックして［クラスの構成設定］ダイアログ・ボックスを開きます。



［**マップ対象クラス**］ボックスには，object クラスが表示されます。これは，WinRunner がすべてのユーザ定義オブジェクトの標準のクラスとして使用するクラスです。

6［**マップ対象クラス**］のリストから，ユーザ定義クラスのマップ対象となる標準クラスを選択します。ユーザ定義クラスの動作と同等の動作をする標準クラスにマッピングしなければならない点に注意してください。

対象となる標準クラスを選択すると，そのクラスの GUI マップ構成情報がダイアログ・ボックスに表示されます。

ユーザ定義クラスの GUI マップ構成情報を変更することも可能です（学習対象プロパティ，セレクタ，記録の方式など）。詳細については，596 ページ「標準またはユーザ定義クラスの構成設定」を参照してください。

7［**OK**］をクリックすると，構成が完了します。

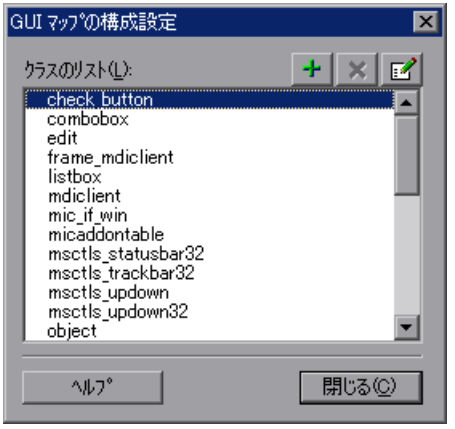
構成は，設定を行ったときのセッションにおいてのみ有効である点に注意してください。構成を恒久的なものとするには，TSL ステートメントを初期化テスト・スクリプトに貼り付けなければなりません。詳細については，601 ページ「恒久的な GUI マップの構成設定の作成」を参照してください。

標準またはユーザ定義クラスの構成設定

任意の標準のクラスあるいはユーザ定義クラスについて，学習対象プロパティ，セレクタ，および記録方式を変更できます。

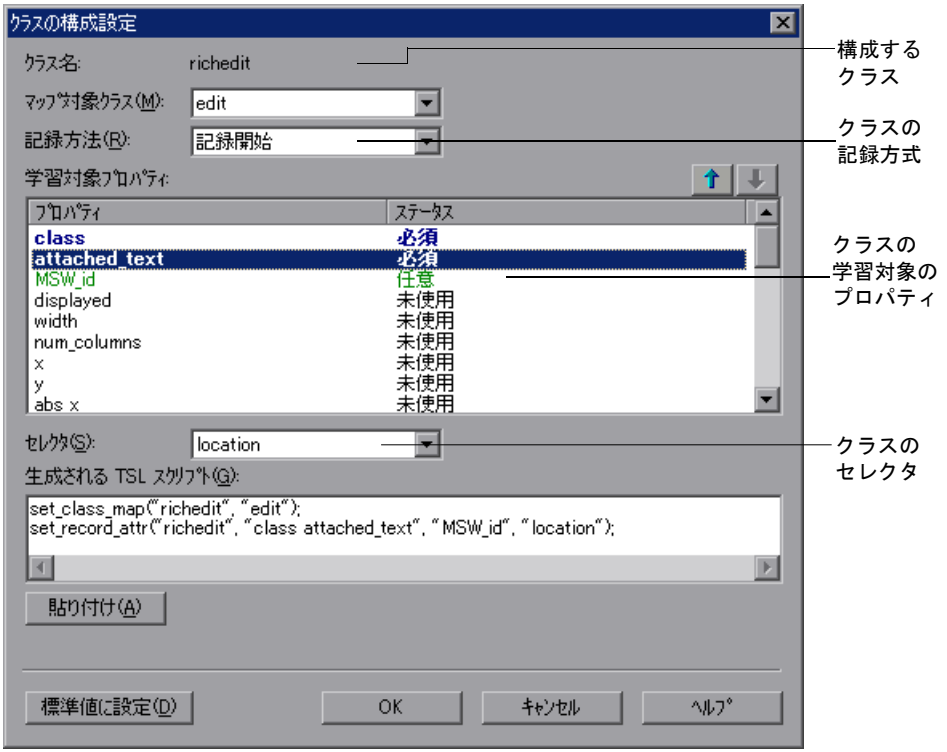
標準またはユーザ定義クラスを構成するには，次の手順を実行します。

1 ［**ツール**］ > ［**GUI マップ構成**］を選択して，［GUI マップの構成設定］ダイアログ・ボックスを開きます。



［クラスのリスト］には，すべての標準クラスのほか，これまでに追加したユーザ定義クラスが表示されます。

2 構成設定を行うクラスを選択して［**構成設定**］ボタンをクリックします。［クラスの構成設定］ダイアログ・ボックスが表示されます。



ダイアログ・ボックスの最上部にある［**クラス名**］フィールドには，構成対象クラスの名前が表示されます。

3 必要に応じて学習対象プロパティ，セレクタ，記録方式を変更します。詳細については，598 ページ「学習対象プロパティの構成設定」，600 ページ「セレクタの構成設定」，または 600 ページ「記録方法の構成」を参照してください。

4 ［**OK**］をクリックします。

構成は，設定を行ったときのセッションにおいてのみ有効である点に注意してください。構成を恒久的なものとするには，TSL ステートメントを初期化テスト・スクリプトに貼り付けなければなりません。詳細については，601 ページ「恒久的な GUI マップの構成設定の作成」を参照してください。

5 ［GUI マップの構成設定］ダイアログ・ボックスで［**OK**］をクリックします。

学習対象プロパティの構成設定

［クラスの構成設定］ダイアログ・ボックスの［学習対象プロパティ］セクションでは，記録あるいは学習の対象となるクラスのプロパティを構成できます。これは，プロパティのステータスを変更して，プロパティが必須，オプション，または未使用のどれであるか指定します。

- 必須プロパティは，常に学習されるプロパティです（それが対象オブジェクトにおいて有効なプロパティである場合）。
- オプション・プロパティは，必須プロパティだけでオブジェクトを一意に識別できない場合にだけ使用されます。WinRunner は記録の際に，オブジェクトを識別するために，リストの先頭から順に，必要な最小限のプロパティしか使いません
- 上記以外のプロパティは，使用されていません。

ダイアログ・ボックスが表示されると，［**学習対象プロパティ**］のリストには，［クラス名］フィールドに表示されたクラスの学習対象プロパティが表示されます。

プロパティが表示される順番は重要です。必須プロパティは常にリストの先頭に表示され，任意のプロパティ，未使用のプロパティと続きます。WinRunner が必須プロパティを使用してオブジェクトを特定できない場合は，リストに表示される順番で任意のプロパティが参照されます。［**項目を上へ移動**］または［**項目を下へ移動**］ボタンをクリックして，プロパティの位置を調整することができます。

プロパティの構成を変更するには，次の手順を実行します。

1. 変更するステータスを持つプロパティの［**ステータス**］セルをクリックします。
2. リストから［必須］，［任意］，［未使用］のどれかを選択します。
3. ［**OK**］ボタンをクリックして変更を保存します。

すべてのプロパティが必ずしも，すべてのクラスに適用されるわけではありません。各プロパティと，そのプロパティの適用が可能なクラスの一覧を以下に示します。

プロパティ	クラス
abs_x	すべてのクラス
abs_y	すべてのクラス
active	すべてのクラス
attached_text	combobox，edit，listbox，scrollbar
class	すべてのクラス
displayed	すべてのクラス
enabled	すべてのクラス
focused	すべてのクラス
handle	すべてのクラス
height	すべてのクラス
label	check_button，push_button，radio_button，static_text，window
maximizable	calendar, window
minimizable	calendar, window
MSW_class	すべてのクラス
MSW_id	window を除くすべてのクラス
nchildren	すべてのクラス
obj_col_name	edit
owner	mdiclient，window
pb_name	check_button，combobox，edit, list，push_button，radio_button，scroll，window（object）
regexp_label	ラベル付きのすべてのクラス
regexp_ MSWclass	すべてのクラス

プロパティ	クラス
text	すべてのクラス
value	calendar，check_button，combobox，edit，listbox，radio_button，scrollbar，static_text
vb_name	すべてのクラス
virtual	list，push_button，radio_button，table，object（仮想オブジェクトのみ）
width	すべてのクラス
x	すべてのクラス
y	すべてのクラス

セレクタの構成設定

必須プロパティとオプションプロパティを使用してもオブジェクトを一意に識別できない場合には，WinRunner は「**位置**」，または「**インデックス**」のいずれかのセレクタを使用します。

位置セレクタの場合，ウィンドウ内でのオブジェクトの位置（上から下，および左から右に向かう順番）に従って識別が行われます。インデックス・セレクタの場合は，アプリケーション開発者がオブジェクトに対して割り当てた一意の番号に従って識別が行われます。セレクタの使用例については，592 ページ「標準の GUI マップの構成設定について」を参照してください。

標準の場合，WinRunner はすべてのクラスに対して，位置セレクタを使用します。セレクタを変更するには，適切なラジオ・ボタンを有効にします。

記録方法の構成

記録方法を設定することで，WinRunner が同一のクラスに属するオブジェクトに対する操作を記録する方法を制御できます。3 つの記録方式があります。

➤ ［**記録開始**］の場合，WinRunner は，GUI オブジェクトに対して行われるすべての操作を記録します（唯一の例外は，static クラス（静的テキスト）で，このクラスの標準の記録方法は「**スキップ**」です）。

- ［**スキップ**］の場合，WinRunner は，このクラスで実行される操作を，オブジェクトを内包している要素に対する操作として記録します。通常，この要素はウィンドウであり，操作は **win_mouse_click** として記録されます。
- ［*オブジェクトとして*］の場合，WinRunner は GUI オブジェクトで実行されたすべての操作を、そのクラスが「object」クラスであるかのように記録します。
- ［*無視*］の場合，WinRunner は，オブジェクトに対するマウス・クリックを無視します。

記録方法を変更するには，適切なラジオ・ボタンを選択します。

恒久的な GUI マップの構成設定の作成

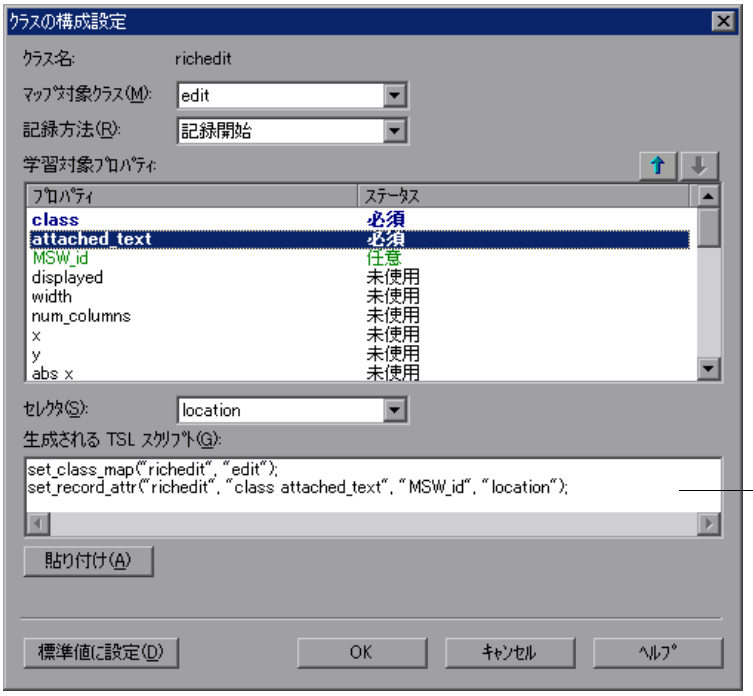
設定した GUI 構成を記述する TSL ステートメントを生成し，それを初期化テストに挿入しておけば，標準およびユーザ定義オブジェクトについて，WinRunner が常に適切な構成を使用するようにできます。

クラスの恒久的な GUI 構成を作成するには，次の手順を実行します。

1 ［**ツール**］＞［**GUI マップ構成**］を選択して，［GUI マップの構成設定］ダイアログ・ボックスを開きます。

2 クラスをクリックし，［**構成設定**］ボタンをクリックします。［クラス構成］ダイアログ・ボックスが開きます。

3 クラスに必要な構成を設定します。ダイアログ・ボックスの下の表示枠に，WinRunner は自動的に構成用の TSL ステートメントを自動的に生成します。

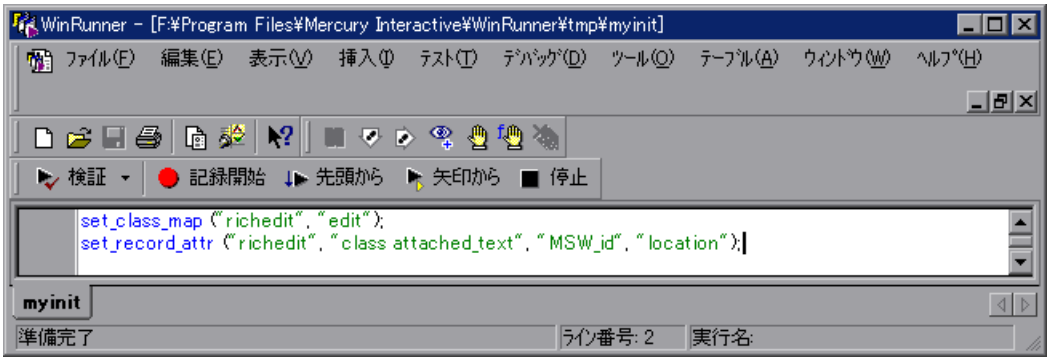


4 [**貼り付け**] ボタンを使って，TSL ステートメントを初期化テストに貼り付けます。

例えば，WinRunner の構成ファイルである wrun.ini（Windows フォルダにあります）に，初期化テストが次のように設定されていたとします。

```
[WrEnv]
XR_TSL_INIT = C:¥tests¥my_init
```

my_init テストを WinRunner ウィンドウ内に開いて，生成された TSL ステートメントを貼り付けます。



初期化テストの詳細については，第 45 章「特殊な構成の初期化」を参照してください。ユーザ定義の GUI マップ構成を定義する TSL 関数（**set_class_map**，**set_record_attr**，**set_record_method**）の詳細については，「**TSL リファレンス**」を参照してください。

ユーザ定義クラスの削除

削除可能なクラスは，ユーザ定義オブジェクトのクラスだけです。WinRunner が使用する標準クラスは削除できません。

ユーザ定義クラスを削除するには，次の手順を実行します。

1 ［**ツール**］＞［**GUI マップ構成**］を選択して，［GUI マップの構成設定］ダイアログ・ボックスを開きます。

2 ［クラスのリスト］で，削除するクラスを選択します。

3 ［**削除**］をクリックします。

WinRunner のオブジェクト・クラスについて

WinRunner は，GUI オブジェクトをオブジェクト・クラスの数にしたがって，次のクラスで分類します。任意のオブジェクトのプロパティを表示する場合，クラスのプロパティはオブジェクトのオブジェクト・プロパティ・クラスを示します。WinRunner は次のオブジェクト・クラスをサポートします。

クラス	説明
calendar	***CDateTimeCtrl*** または ***CMonthCalCtrl*** MSW_class に属する標準カレンダー・オブジェクト。
check_button	チェック・ボックス
edit	編集フィールド
frame_mdiclient	WinRunner がウィンドウを mdiclient オブジェクトとして扱えるようにします。
list	リスト・ボックス。通常のリストかコンボ・ボックスです。
menu_item	メニュー項目
mdiclient	mdiclient オブジェクト
mic_if_win	WinRunner が，ウィンドウ内のオブジェクトに対する記録および実行操作をすべて mic_if ライブラリに送れるようにします。詳細については，『**WinRunner カスタマイズ・ガイド**』を参照してください。
object	この表で説明されているクラスに含まれないあらゆるオブジェクト。
push_button	プッシュ（コマンド）ボタン
radio_button	ラジオ（オプション）ボタン
scroll	スピン・オブジェクト
spin	スクロール・バーまたはスライダ
static_text	GUI オブジェクトの一部ではない，表示専用テキスト。
status bar	ウィンドウのステータス・バー
tab	タブ項目

クラス	説明
toolbar	ツールバー・オブジェクト
window	MDI ウィンドウを含む任意のアプリケーション・ウィンドウ，ダイアログ・ボックスあるいはフォーム。

オブジェクトのプロパティについて

以下の表は，WinRunner がコンテキスト・センシティブ・テストの実行の際に使用する全プロパティを一覧にしたものです。

プロパティ	説明
abs_x	画面の原点（左上隅）からのオブジェクトの左上隅の相対 x 座標。
abs_y	画面の原点（左上隅）からのオブジェクトの左上隅の相対 y 座標。
active	最上位レベルウィンドウに対して入力フォーカスが向けられているかどうかを示すブール値。
attached_text	オブジェクトの近くにある静的テキスト。
class	メニューに含まれているメニュー項目の数。
class_index	オブジェクトが表示されているかどうかを示す論理値。画面上に見える場合は 1 で，見えない場合は 0。
count	オブジェクトが選択可能あるいは有効にできるかどうかを示す論理値。すでに有効あるいは選択されている場合は 1 で，そうでない場合は 0。
displayed	オブジェクトに対してキーボード入力が向けられているかどうかを示す論理値。オブジェクトがキーボード・フォーカスを持っている場合は 1 で，持っていない場合は 0。
enabled	メニューに含まれているメニュー項目の数。
focused	オブジェクトが表示されているかどうかを示す論理値。画面上に見える場合は 1 で，見えない場合は 0。
handle	オブジェクトに対する実行時ポインタ：HWND ハンドル。
height	オブジェクトの高さ（単位：ピクセル）。

プロパティ	説明
label	URL （WebTest のみ）
maximizable	ボタンのラベルなど，オブジェクト上に表示されるテキスト。
minimizable	ウィンドウが最大化可能かどうかを示す論理値。可能な場合は 1 で，不可の場合は 0。
module_name	ウィンドウが最小化可能かどうかを示す論理値。可能な場合は 1 で，不可の場合は 0。
MSW_class	Microsoft Windows のクラス。
MSW_id	Microsoft Windows の ID。
nchildren	オブジェクトの所有する子の数（オブジェクトの子孫の総数）
num_columns	端末エミュレータ・アプリケーションのテーブル・オブジェクトのみ。
num_rows	端末エミュレータ・アプリケーションのテーブル・オブジェクトのみ。
obj_col_name	DataWindow とカラム名を連結したもの。PowerBuilder アドインがサポートされている WinRunner の編集フィールド・オブジェクト用のカラム名を示します。
owner	（ウィンドウ用）ウィンドウが属するアプリケーション（実行形式）の名前。
parent	オブジェクトの親の論理名。
pb_name	開発者によって PowerBuilder オブジェクトに割り当てられたテキスト文字列（このプロパティは，PowerBuilder アドインがサポートされている WinRunner に対してのみ適用します）。
position	メニュー内にあるメニュー項目の位置（上から下へ）。（先頭の項目の position の値は 0 です）。
regexp_label	ラベルが変わるオブジェクトを識別するためのテキスト文字列と正規表現。
regexp_MSWclass	Microsoft Windows のクラスと正規表現の組み合わせ。WinRunner は，MSW_ class が変わるオブジェクトを識別できます。
submenu	メニュー項目にサブメニューがあるかどうかを示す論理値。ある場合は 1 で，ない場合は 0。

プロパティ	説明
sysmenu	メニュー項目がシステム・メニューの一部であるかどうかを示すブール値。
TOOLKIT_class	指定されたツールキット・クラスの値。このプロパティの値は Windows の MSW_class あるいは Motif の X_class の値と同じです。
text	オブジェクトまたはウィンドウに表示されるテキスト。
value	各クラスごとに異なります。 ラジオおよびチェック・ボタン：ボタンがチェックされている場合は 1，そうでなければ 0。 メニュー項目：項目がチェックされている場合は 1，そうでなければ 0。 リスト・オブジェクト：選択された項目の文字列を示します。 編集／静的オブジェクト：テキスト・フィールドの内容を示します。 スクロール・オブジェクト：スクロールの位置を示します。 その他のクラス：value プロパティは NULL 文字列です。
vb_name	開発者によって Visual Basic オブジェクトに割り当てられたテキスト文字列（「*name*」プロパティ）。（このプロパティは，Visual Basic アドインがサポートされている WinRunner にのみ適用します）。
width	オブジェクトの幅（単位：ピクセル）。
x	ウィンドウ原点からのオブジェクトの左上隅の相対 x 座標。
y	ウィンドウ原点からのオブジェクトの左上隅の相対 y 座標。

学習対象プロパティについて

次の表は，それぞれのクラスで学習される標準のプロパティの一覧です（標準のプロパティは，テスト・スクリプト・ウィザード，GUI マップ・エディタ，および記録のすべての学習方法に適用されます）。

クラス	必須プロパティ	オプション・プロパティ	セレクタ
すべての button	class，label	MSW_id	位置
list，edit，scroll，combobox	class,，attached_text	MSW_id	位置

クラス	必須プロパティ	オプション・プロパティ	セレクタ
frame_mdiclient	class, regexp_MSWclass, regexp_label	label, MSW_class	位置
menu_item	class, label, sysmenu	position	位置
object	class, regexp_MSWclass, label	attached_text, MSW_id, MSW_class	位置
mdiclient	class, label	regexp_MSWclass, MSW_class	
static_text	class, MSW_id	label	位置
window	class, regexp_MSWclass, label	attached_text, MSW_id, MSW_class	位置

Visual Basic のオブジェクトのプロパティ

label と vb_ name プロパティは必須のプロパティで，Visual Basic オブジェクトのすべてのクラスに対して学習されます。Visual Basic オブジェクトのテストについては，第 11 章「ActiveX と Visual Basic のコントロールの使用」を参照してください。

注：Visual Basic のアプリケーションをテストするには，Visual Basic サポートをインストールしなければなりません。詳細は，**『WinRunner インストール・ガイド』**を参照してください。

PowerBuilder のオブジェクトのプロパティ

次の表は，各 PowerBuilder オブジェクトについて学習される標準オブジェクト・クラスとプロパティです。PoweBuilder オブジェクトのテストの詳細については，第 12 章「PowerBuilder のアプリケーションの検査」を参照してください。

クラス	必須のプロパティ	オプション・プロパティ	セレクタ
すべての button	class, pb_name	label, MSW_id	位置
list，scroll, combobox	class, pb_name	attached_text, MSW_id	位置
edit	class，pb_name, obj_col_name	attached_text，MSW_id	位置
object	class，pb_name	label，attached_text, MSW_id，MSW_class	位置
window	class，pb_name	label，MSW_id	位置

注：PowerBuilder アプリケーションをテストするには，PowerBuilder サポートをインストールしなければなりません。詳細は，『**WinRunner インストール・ガイド**』を参照してしてください。

第 25 章

仮想オブジェクトの学習

ビットマップを「**仮想オブジェクト**」として定義することで，ウィンドウ内のどのビットマップも GUI オブジェクトとして認識するよう，WinRunner に指示することができます。

本章では，以下の項目について説明します。

- 仮想オブジェクトの学習について
- 仮想オブジェクトの定義
- 仮想オブジェクトの物理的記述について

仮想オブジェクトの学習について

アプリケーションには，GUI オブジェクトのように見えかつ振舞うビットマップが含まれている場合があります。WinRunner は，**win_ mouse_ click** ステートメントを使用してこれらのビットマップに対する操作を記録します。ビットマップを「**仮想オブジェクト**」として定義することによって，テストを記録して実行するときにこうしたビットマップをプッシュ・ボタンなどの GUI オブジェクトのように処理するよう WinRunner に指示できます。こうすることで，テスト・スクリプトが一層読みやすく，理解しやすくなります。

例えば，Windows NT の電卓アプリケーションを対象としたテストを記録するとします。このアプリケーションでは計算を実行するのにボタンをクリックします。WinRunner は電卓のボタンを GUI オブジェクトとして認識できないので，標準で作成されるテスト・スクリプトは次のようなものになります。

```
set_window(" 電卓 ");
win_mouse_click (" 電卓 ", 87, 175);
win_mouse_click (" 電卓 ", 204, 200);
win_mouse_click (" 電卓 ", 121, 163);
```

```
win_mouse_click (" 電卓 ", 242, 201);
```

これは分かりやすいテスト・スクリプトではありません。電卓のボタンを仮想オブジェクトとして定義し，これらを push button クラスと関連付けると，WinRunner は次のようなスクリプトを記録します。

```
set_window (" 電卓 ");
button_press("seven");
button_press("plus");
button_press("four");
button_press("equal");
```

アプリケーション内でのビットマップの動作に応じて，仮想プッシュ・ボタン，ラジオ・ボタン，チェック・ボタン，リスト，テーブルなどの作成が可能です。これらのどれにも当てはまらない場合は，仮想オブジェクトを「object」という汎用のクラスにマップできます。

仮想オブジェクト・ウィザードを使用して，ビットマップを仮想オブジェクトとして定義します。ウィザードから新しいオブジェクトを関連付けたい標準クラスを選択するよう指示するダイアログが表示されます。次に，十字型ポインタを使用して，オブジェクトの領域を定義します。最後に，オブジェクトの論理名を選択します。WinRunner は仮想オブジェクトの論理名と物理的記述を GUI マップに追加します。

仮想オブジェクトの定義

仮想オブジェクト・ウィザードを使用して，ビットマップを標準オブジェクト・クラスに割り当てたり，オブジェクトの座標を定義したり，ビットマップに論理名を割り当てたりできます。

仮想オブジェクト・ウィザードを使用して，仮想オブジェクトを定義するには，次の手順を実行します。

1 ［**ツール**］＞［**仮想オブジェクト ウィザード**］を選択します。仮想オブジェクト・ウィザードが開きます。

［**次へ**］をクリックします。

2 ［クラス］リストで，新しい仮想オブジェクトにクラスを選択します。

list クラスを選択した場合，ウィンドウに表示される可視の行数を選択します。**table** クラスを選択した場合は，可視の行数とカラム数を選択します。［**次へ**］をクリックします。

3 ［**オブジェクトを指定**］ボタンをクリックします。十字型ポインタを使用して，仮想オブジェクトの領域を選択します。矢印キーを使用して，十字型ポインタで定義した領域に正確に合わせます。

注：仮想オブジェクトは，アプリケーション内の GUI オブジェクトと重なってはなりません（これらが「」という汎用のクラスか，「 として」記録されるよう設定されているクラスに属している場合は除きます）。仮想オブジェクトが GUI オブジェクトと重なると，WinRunner は GUI オブジェクトでテストを正しく記録したり実行したりできません。

Enter キーを押すか，マウスを右クリックして，ウィザード内に仮想オブジェクトの座標を表示します。指定されているオブジェクトが画面上で可視なっていれば，［**強調表示**］ボタンをクリックしてこれを表示します。

［**次へ**］をクリックします。

4 仮想オブジェクトに論理名を割り当てます。これは，仮想オブジェクトを記録するときにテスト・スクリプトに表示される名前です。オブジェクトに WinRunner が認識できるテキストが含まれている場合，ウィザードは論理名にこのテキストを使用するよう提案します。さもなければ，WinRunner は *virtual_object*，*virtual_push_button*，*virtual_list* などを提案します。

ウィザードの提案を受け入れることも，別の名前を入力することも可能です。WinRunner は，ユーザの選択を確定する前に，GUI マップ内に同じ名前のオブジェクトがないかどうかを調べます。［**次へ**］をクリックします。

5 仮想オブジェクトの学習を終了します。

➤ 別の仮想オブジェクトを学習したい場合は，［**はい**］を選択し，［**次へ**］をクリックします。

➤ ウィザードを終了するには，［**いいえ**］を選択し，［**完了**］をクリックします。

ウィザードを終了したら，WinRunner はオブジェクトの論理名と物理的記述を GUI マップに追加します。次にその仮想オブジェクトに対する操作を記録する際，WinRunner は **win_ mouse_ click** ステートメントではなく TSL ステートメントを生成します。

仮想オブジェクトの物理的記述について

仮想オブジェクトを作成すると，WinRunner はその物理的記述を GUI マップに追加します。仮想オブジェクトの物理的記述には，「本当の」GUI オブジェクトの物理的記述内にある *label* プロパティが含まれていません。その代わり，仮想オブジェクトの物理的記述には，特別なプロパティ，*virtual* が含まれています。その役割は，仮想オブジェクトを識別することであり，その値はいつも TRUE です。

WinRunner は，仮想オブジェクトをサイズとウィンドウ内の位置によって識別するので，x，y，width，height プロパティは仮想オブジェクトの物理的記述に必ず含まれていなければなりません。

例えば，*virtual_push_button* の物理的記述には，以下のプロパティが含まれます。

```
{
 class: push_button,
 virtual: TRUE,
 x: 82,
 y: 121,
 width: 48,
 height: 28,
}
```

これらのプロパティが変更または削除されると，WinRunner は仮想オブジェクトを認識できなくなります。オブジェクトを移動またはサイズ変更したら，ウィザードを使用して，新しい仮想オブジェクトを作成しなければなりません。

第 7 部

テストの作成 ― 上級

第 26 章

回復シナリオの定義と使用

テスト実行中，テスト環境で生じる予期しないイベントやエラーから回復するよう WinRunner に指示できます。

本章では，以下の項目について説明します。

- 回復シナリオの定義と使用について
- 簡易回復シナリオの定義
- 複合回復シナリオの定義
- 回復シナリオの管理
- 回復シナリオ・ファイルを使った作業
- テスト・スクリプトの回復シナリオ・ファイルを使った作業

回復シナリオの定義と使用について

テスト実行中に，予期しないイベント，エラー，およびアプリケーション・クラッシュが発生すると，テストが妨げられ，正しいテスト結果が得られない可能性があります。これは，バッチ・テストを無人で実行する場合に特に問題になります。回復に必要なアクションを実行するまで，バッチ・テストが一時停止状態になるからです。

回復マネージャでは，「**回復シナリオ**」を定義するプロセスを案内するウィザードが使用できます。回復シナリオとは，予期しないイベントと，テスト実行を回復するために必要な操作のことです。例えば，プリンタの用紙切れメッセージを検出し，［**OK**］ボタンをクリックしてメッセージを閉じることによってテスト実行を回復し，テストの中断した位置からテストを続行するよう WinRunner に指示できます。

回復シナリオには，次の 2 つのタイプがあります。

- **簡易：**（クラッシュ以外の）例外イベントと，イベントを停止してテストを続行できるようにするための単一の操作を定義できます。
- **複合：**例外イベントまたはクラッシュ・イベントと，テストおよび関連するアプリケーションの続行または再開に必要な操作です。

回復シナリオには，次の 2 つの主要な構成要素があります。

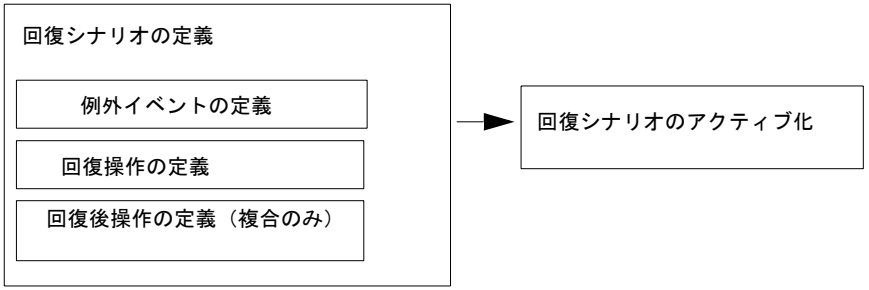
- **例外イベント**：テスト実行を中断するイベントです。
- **回復操作**：中断を終わらせる操作です。

複合回復シナリオには，**回復後操作**も含まれます。回復後操作では，回復操作を実行してから WinRunner をどのように継続するか（継続する前に WinRunner が実行する機能など），および回復操作を実行した場合はテストまたはバッチのどの位置から WinRunner を継続するかについて，指示が提供されます。例えば，特定のアプリケーションを再度開いて適切な状態に設定する関数を実行し，中断したテストを最初から再開する必要が生じる場合もあります。

回復操作や回復後操作のために指定する関数は，通常のコンパイル済みモジュールに含まれる場合と，回復用コンパイル済みモジュールに含まれる場合があります。**回復用コンパイル済みモジュール**は，特別なコンパイル済みモジュールで，WinRunner が開いているときに常にロードされています。これによって，WinRunner が回復シナリオを実行するときはいつでもこのモジュールに含まれる関数にアクセスできます。

テスト実行中に回復シナリオを実行するよう WinRunner に指示するには，回復シナリオをアクティブ化する必要があります。

次の図は，回復シナリオの作成に関する手順をまとめたものです。



回復シナリオは，Windows イベントのみに適用されます。また，Web 例外やハンドラ関数を定義することもできます。詳細については，第 32 章「Web 例外処理の定義」を参照してください。

簡易回復シナリオの定義

簡易回復シナリオは，クラッシュ以外の例外イベントと，イベントを停止してテストを続行できるようにするための単一の操作を定義します。

簡易回復シナリオは，回復マネージャの［**標準**］タブで定義および変更できます。回復ウィザードが，シナリオの作成または変更の手順を示します。

TSL ステートメントを使用して簡易回復シナリオを定義することもできます。詳細については，660 ページ「テスト・スクリプトの回復シナリオ・ファイルを使った作業」を参照してください。

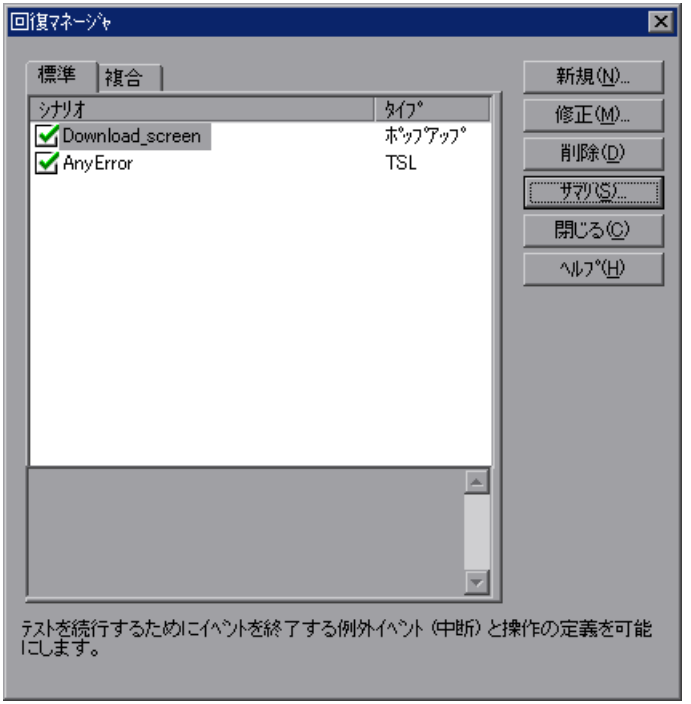
注：

簡易回復シナリオは，これまで例外処理と呼ばれていたものに相当します。WinRunner 7.01 以前の例外ハンドラで作成された例外は，回復マネージャの［**標準**］タブに表示されます。

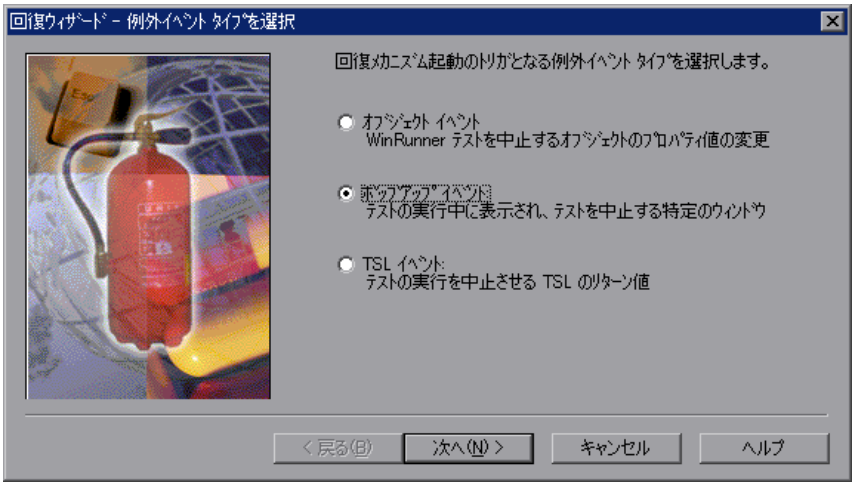
回復マネージャを使用して回復シナリオを初めて追加，変更，または削除すると，新しい回復シナリオ・ファイルを選択するように求められます。詳細については，656 ページ「回復シナリオ・ファイルを使った作業」を参照してください。

簡易回復シナリオを作成するには，次の手順を実行します。

1 ［**ツール**］ > ［**回復マネージャ**］を選択します。回復マネージャが開きます。



2 [**新規**] をクリックします。回復マネージャに [例外イベント タイプを選択] 画面が表示されます。



3 回復メカニズムを呼び出す例外イベントのタイプを選択します。

- [**オブジェクト イベント**]：WinRunner テストを中断させる原因となるオブジェクトのプロパティ値の変化。

  例えば，電気回路が閉じていることを示す緑色のボタンがアプリケーションで使用されていて，回路が故障するとこのボタンが赤色に変わるとします。この場合，回路の故障中はテストを続行できません。

- [**ポップアップ イベント**]：テスト実行中にポップアップしてテストを中断させるウィンドウ。

  例えば，印刷ボタンをクリックして作成されたグラフをプリンタに送信する操作がテストに含まれていて，プリンタの用紙切れを示すメッセージ・ボックスが開いたとします。この場合，メッセージ・ボックスを閉じるまではテストを続行できません。

- [**TSL イベント**]：テスト実行を中断させる原因となる TSL の戻り値。

  例えば，**set_window** ステートメントからエラーが返ったとします。この場合，回復シナリオを使用することにより，ウィンドウを閉じ，初期化し，再度開くことができます。

[**次へ**] をクリックします。

4 ［シナリオ名］画面が開きます。

名前は，英数字とアンダスコアのみを使用し（スペースや特殊文字は使用不可），回復シナリオの内容がわかるように入力します。

［**次へ**］をクリックします。

5 ［例外イベントの定義］画面が開きます。この画面に表示されるオプションは，手順 3 で選択したイベントのタイプによって異なります。

オブジェクト・イベントの定義の詳細については，627 ページを参照してください。

ポップアップ・イベントの定義の詳細については，629 ページを参照してください。

TSL イベントの定義の詳細については，630 ページを参照してください。

手順 3 でオブジェクト・イベントを選択した場合は，次の情報を入力します。

- ［**ウィンドウ名**］：例外の原因となるオブジェクトを含むウィンドウの名前を指定します。ウィンドウの論理名を入力します。または，［オブジェクト名］ボックスの横にある指差しマークを使用して，オブジェクト例外として定義するオブジェクトをクリックすると，［ウィンドウ名］と［オブジェクト名］が自動的に入力されます。

  ウィンドウを例外オブジェクトとして定義する場合は，ウィンドウのタイトル・バーをクリックするか，ウィンドウの論理名を入力して［オブジェクト名］ボックスを空白のままにします。

- ［**オブジェクト名**］：例外の原因となるオブジェクトの名前を指定します。オブジェクトの論理名を入力します。または，［オブジェクト名］ボックスの横にある指差しマークを使用して，オブジェクト例外として定義するオブジェクトを指定すると，［ウィンドウ名］と［オブジェクト名］が自動的に入力されます。

注：定義するオブジェクトは，GUI マップに保存する必要があります。指差しマークを使用して，GUI マップにまだ保存されていないオブジェクトを特定すると，そのオブジェクトはアクティブな GUI マップに自動的に追加されます。オブジェクト名を手動で入力した場合は，そのオブジェクトを GUI マップにも追加する必要があります。GUI マップの詳細については，第 4 章「GUI マップの基本概念について」を参照してください。

➤ ［**オブジェクト プロパティ**］：値をチェックするオブジェクト・プロパティ。例外が発生する可能性があるオブジェクト・プロパティを選択します。例えば，ボタンの状態が有効から無効に変化したことを検出する場合は，enabled プロパティを選択します。

注：オブジェクトの物理的記述を構成するプロパティは，指定できません。

➤ ［**プロパティ値**］：例外が発生したことを示す値。例えば，ボタンの状態が有効から無効に変化したときに回復シナリオをアクティブ化するには，このフィールドに 0 を入力します。

ヒント：プロパティ値のあらゆる変化を検出するには，プロパティ値を空白のままにします。

［**次へ**］をクリックして 631 ページの手順 6 に進みます。

手順 3 でポップアップ・イベントを選択した場合は，次の情報を入力します。

➤ [**ウィンドウ名**]：例外の原因となるポップアップ・ウィンドウの名前を指定します。ウィンドウの論理名を入力します。または，指差しマークを使用して，ポップアップ例外として定義するウィンドウを指定します。

指差しマークを使用して，GUI マップにまだ保存されていないウィンドウを特定すると，そのウィンドウはアクティブな GUI マップに自動的に追加されます。GUI マップにまだ保存されていないウィンドウの名前を手動で入力すると，入力した名前と一致するタイトル・バーを持つポップアップ・ウィンドウが開いたときに，ポップアップ例外が識別されます。

注：[**次のボタンをクリックする**] 回復操作を使用する場合は，定義したポップアップ・ウィンドウを GUI マップに保存する必要があります。ウィンドウ名を手動で入力した場合は，そのウィンドウを GUI マップにも追加する必要があります。回復操作の詳細については，631 ページを参照してください。

ヒント：例外の原因となるポップアップ・ウィンドウのウィンドウ名が動的に生成されたものである場合は，指差しマークを使用してそのウィンドウを GUI マップに追加し，正規表現を使用して GUI マップ内のウィンドウの定義を変更します。

［**次へ**］をクリックして 631 ページの手順 6 に進みます。

手順 3 で TSL イベントを選択した場合は，次の情報を入力します。

➤ ［**TSL 機能**］: 例外イベントを定義する TSL 関数を選択します。リストから TSL 関数を選択します。選択した TSL 関数が［エラー コード］ボックスで選択したコードを返した場合にのみ，例外が検出されます。

ヒント：指定したエラー・コードを返すすべての TSL 関数で例外メカニズムを呼び出すには，［**<< any function >>**］を選択します。

➤ ［**エラー コード**］: 例外メカニズムを呼び出す TSL エラー・コードを選択します。リストからエラー・コードを選択します。テスト実行中に選択した TSL 関

数でこのリターン・コードが検出されると，回復シナリオがアクティブ化されます。

［**次へ**］をクリックします。

6 ［回復作業の定義］画面が開きます。

次のいずれかの回復オプションを選択します。

➤ ［**次のボタンをクリックする**］：例外イベントが発生したときにポップアップ・ウィンドウ上でクリックするボタンの論理名を指定します。標準のボタン名を選択するか，ボタンの論理名を入力するか，または指差しマークを使用してクリックするボタンを指定します。

注：

このオプションはポップアップ例外にのみ適用されます。

回復シナリオのために定義したポップアップ・ウィンドウは，GUI マップに定義する必要があります。ポップアップ回復シナリオを定義するときに，ロード済みの GUI マップにポップアップ・ウィンドウが定義されていない場合は，回復シナリオが自動的に非アクティブに設定されます。その後，ポップアップ・ウィンドウを含む GUI マップをロードすると，回復シナリオをアクティブ化できます。

- ［**起動中のウィンドウを閉じる**］：例外イベントが発生したときにアクティブな（フォーカスがある）ウィンドウを閉じるよう WinRunner に指示します。

注： WinRunner は（TSL の）**win_close** メカニズムを使用してウィンドウを閉じます。**win_close** 関数でウィンドウを閉じることができない場合，回復シナリオはウィンドウを閉じることができません。このような場合は，［**次のボタンをクリックする**］オプションまたは［**次の回復関数を実行する**］オプションを使用してください。

- ［**次の回復関数を実行する**］：例外イベントが発生したときに，指定した関数を実行するよう WinRunner に指示します。既存の関数を指定するか，または［**回復関数の定義**］をクリックして新しい関数を定義します。回復関数の定義の詳細については，649 ページ「回復シナリオ関数の定義」を参照してください。

注： テスト実行時に，この関数を含むコンパイル済みモジュールがロードされている必要があります。関数を回復用コンパイル済みモジュールに保存し，WinRunner が開いているときは関数が常に自動的にロードされるようにします。回復用コンパイル済みモジュールに保存されている関数を選択しない場合は，その関数を使用する回復シナリオがアクティブ化されたときに，その関数を含むコンパイル済みモジュールが必ずロードされるようにしてください。

［**次へ**］をクリックします。

7［完了］画面が開きます。

WinRunner が開いているときに標準設定で回復シナリオをアクティブ化するかどうかを決定します。

- WinRunner が開いているときに標準設定で自動的に回復シナリオをアクティブ化するよう WinRunner に指示する場合は，［**標準設定値で起動する**］を選択します。前の WinRunner セッションの終了時にシナリオが非アクティブに設定された場合でも，シナリオはアクティブ化されます。
- WinRunner が開いているときに標準設定で自動的に回復シナリオをアクティブ化しない場合は，［**標準設定値で起動する**］をクリアします。このチェック・ボックスをクリアすると，［回復マネージャ］ダイアログ・ボックスでこのチェック・ボックスを手動で選択してアクティブ化しない限り，回復シナリオはアクティブ化されません。

回復シナリオをアクティブまたは非アクティブに設定するほかの方法については，654 ページ「回復シナリオのアクティブ化と非アクティブ化」および 660 ページ「テスト・スクリプトの回復シナリオ・ファイルを使った作業」を参照してください。

［**完了**］をクリックします。回復シナリオが［回復マネージャ］ダイアログ・ボックスの［**標準**］タブに追加されます。［標準設定で起動する］を選択した（および必要なオブジェクトがロード済みの GUI マップ・ファイルに見つかっ

た）場合は，回復シナリオがアクティブ化されます。そうでない場合は，回復シナリオは非アクティブのままです。

複合回復シナリオの定義

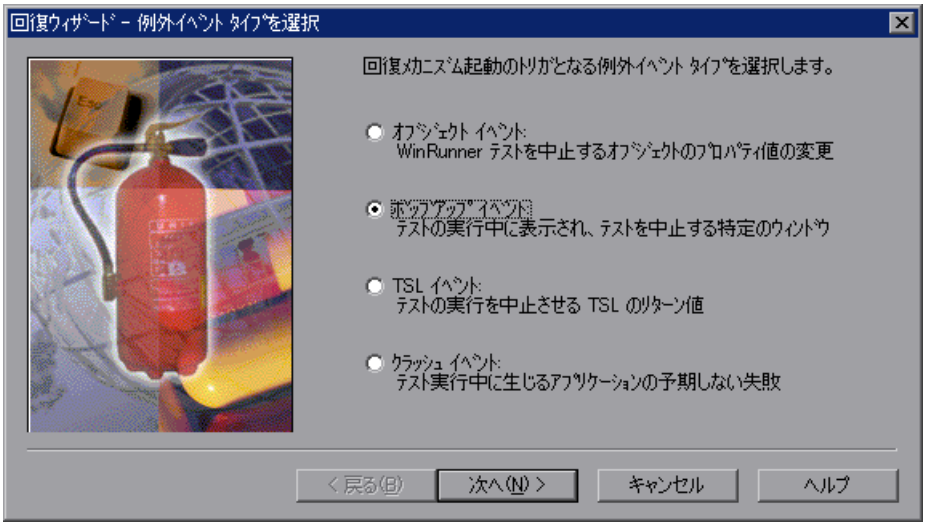
複合回復シナリオは，クラッシュ・イベントまたは例外イベントと，テストおよび関連するアプリケーションの続行または再開に必要な操作を定義します。複合回復シナリオは，回復マネージャの［**複合**］タブで定義および変更できます。回復ウィザードが，シナリオの作成および変更の手順を示します。

複合回復シナリオを作成するには，次の手順を実行します。

1 ［**ツール**］＞［**回復マネージャ**］を選択します。回復マネージャが開きます。

2 ［**複合**］タブをクリックします。

3 [**新規**] をクリックします。回復マネージャに [例外イベント タイプを選択] 画面が表示されます。



4 回復メカニズムを呼び出す例外イベントのタイプを選択します。

- [**オブジェクト イベント**]：WinRunner テストを中断させる原因となるオブジェクトのプロパティ値の変化。

  例えば，電気回路が閉じていることを示す緑色のボタンがアプリケーションで使用されていて，回路が故障するとこのボタンが赤色に変わるとします。この場合，回路の故障中はテストを続行できません。

- [**ポップアップ イベント**]：テスト実行中にポップアップしてテストを中断させるウィンドウ。

  例えば，印刷ボタンをクリックして作成されたグラフをプリンタに送信する操作がテストに含まれていて，プリンタの用紙切れを示すメッセージ・ボックスが開いたとします。この場合，メッセージ・ボックスを閉じるまではテストを続行できません。

- [**TSL イベント**]：テスト実行を中断させる原因となる TSL の戻り値。

- [**クラッシュ イベント**]：テスト実行中のアプリケーションに発生する予期しない障害。

注：

標準設定では，WinRunner は Application Error という文字列を含むウィンドウが開いたときにクラッシュ・イベントを識別します。WinRunner がクラッシュ・ウィンドウを識別するために使用する文字列は，**＜ WinRunner のインストール先フォルダ＞ ¥dat** フォルダにある **excp_str.ini** ファイルで変更できます。詳細については，654 ページ「クラッシュ・イベントのウィンドウ名の変更」を参照してください。

クラッシュ回復シナリオをアクティブ化すると，テストの実行速度が遅くなる場合があります。詳細については，「**WinRunner 最初にお読みください**」を参照してください。

［**次へ**］をクリックします。

5 ［シナリオ名］画面が開きます。

名前は，英数字とアンダスコアのみを使用し（スペースや特殊文字は使用不可），回復シナリオの内容がわかるように入力します。

［**次へ**］をクリックします。

6 手順 4 でオブジェクト・イベント，ポップアップ・イベント，または TSL イベントを選択した場合は，［例外イベントの定義］画面が開きます。イベントの定義に関するオプションは，選択したイベントのタイプによって異なります。

手順 4 でクラッシュ・イベントを選択した場合は，イベントを定義する必要はありません。641 ページの手順 7 に進みます。

オブジェクト・イベントの定義の詳細については，637 ページを参照してください。

ポップアップ・イベントの定義の詳細については，639 ページを参照してください。

TSL イベントの定義の詳細については，640 ページを参照してください。

手順 4 でオブジェクト・イベントを選択した場合は，次の情報を入力します。

➤ ［**ウィンドウ名**］：例外の原因となるオブジェクトを含むウィンドウの名前を指定します。ウィンドウの論理名を入力します。または，［**オブジェクト名**］ボックスの横にある指差しマークを使用して，オブジェクト例外として定義するオブジェクトをクリックすると，［ウィンドウ名］と［オブジェクト名］が自動的に入力されます。

ウィンドウを例外オブジェクトとして定義する場合は，ウィンドウのタイトル・バーをクリックするか，ウィンドウの論理名を入力して［**オブジェクト名**］ボックスを空白のままにします。

- ［**オブジェクト名**］：例外の原因となるオブジェクトの名前を指定します。オブジェクトの論理名を入力します。または，［**オブジェクト名**］ボックスの横にある指差しマークを使用して，オブジェクト例外として定義するオブジェクトを指定すると，［ウィンドウ名］と［オブジェクト名］が自動的に入力されます。

注：定義するオブジェクトは，GUI マップに保存する必要があります。指差しマークを使用して，GUI マップにまだ保存されていないオブジェクトを特定すると，そのオブジェクトはアクティブな GUI マップに自動的に追加されます。オブジェクト名を手動で入力した場合は，そのオブジェクトを GUI マップにも追加する必要があります。GUI マップの詳細については，第 4 章「GUI マップの基本概念について」を参照してください。

- ［**オブジェクト プロパティ**］：値をチェックするオブジェクト・プロパティ。例外が発生する可能性があるオブジェクト・プロパティを選択します。例えば，ボタンの状態が有効から無効に変化したことを検出する場合は，enabled プロパティを選択します。

注：オブジェクトの物理的記述を構成するプロパティは，指定できません。

- ［**プロパティ値**］：例外が発生したことを示す値。例えば，ボタンの状態が有効から無効に変化したときに回復シナリオをアクティブ化するには，このフィールドに 0 を入力します。

ヒント：プロパティ値のあらゆる変化を検出するには，プロパティ値を空白のままにします。

［**次へ**］をクリックして 641 ページの手順 7 に進みます。

手順 4 でポップアップ・イベントを選択した場合は，次の情報を入力します。

➤ [**ウィンドウ名**]：例外の原因となるポップアップ・ウィンドウの名前を指定します。ウィンドウの論理名を入力します。または，指差しマークを使用して，ポップアップ例外として定義するウィンドウを指定します。

指差しマークを使用して，GUI マップにまだ保存されていないウィンドウを特定すると，そのウィンドウはアクティブな GUI マップに自動的に追加されます。GUI マップにまだ保存されていないウィンドウの名前を手動で入力すると，入力した名前と一致するタイトル・バーを持つポップアップ・ウィンドウが開いたときに，ポップアップ例外が識別されます。

注：[**次のボタンをクリックする**] 回復操作を使用する場合は，定義したポップアップ・ウィンドウを GUI マップに保存する必要があります。ウィンドウ名を手動で入力した場合は，そのウィンドウを GUI マップにも追加する必要があります。回復操作の詳細については，641 ページを参照してください。

ヒント：例外の原因となるポップアップ・ウィンドウのウィンドウ名が動的に生成されたものである場合は，指差しマークを使用してそのウィンドウを GUI マップに追加し，正規表現を使用して GUI マップ内のウィンドウの定義を変更します。

［**次へ**］をクリックして 641 ページの手順 7 に進みます。

手順 4 で TSL イベントを選択した場合は，次の情報を入力します。

➤ ［**TSL 機能**］: 例外イベントを定義する TSL 関数を選択します。リストから TSL 関数を選択します。選択した TSL 関数が［エラー コード］ボックスで選択したコードを返した場合にのみ，例外が検出されます。

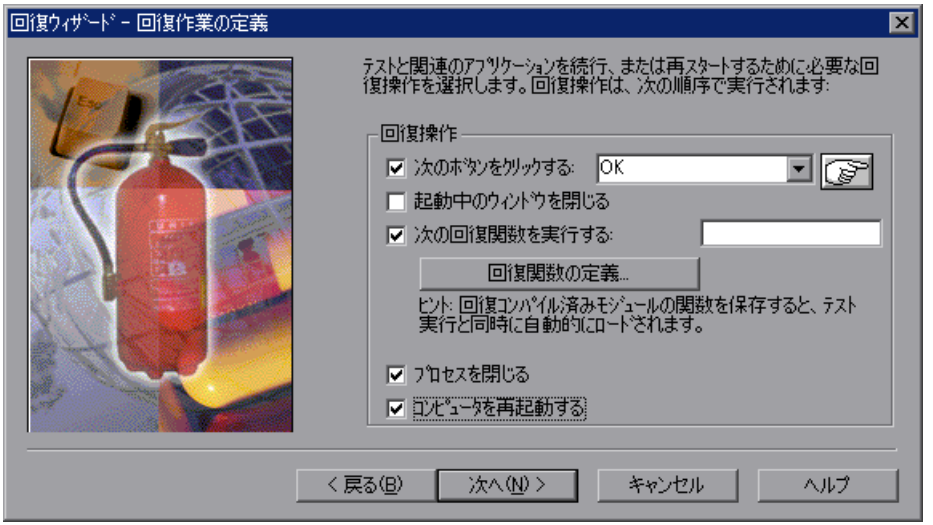
ヒント：指定したエラー・コードを返すすべての TSL 関数で例外メカニズムを呼び出すには，［**<< any function >>**］を選択します。

➤ ［**エラー コード**］: 例外メカニズムを呼び出す TSL エラー・コードを選択します。リストからエラー・コードを選択します。テスト実行中に選択した TSL 関

数でこのリターン・コードが検出されると，回復シナリオがアクティブ化されます。

［**次へ**］をクリックします。

7 ［回復作業の定義］画面が開き，例外発生時に実行できる回復操作が表示されます。



WinRunner は，ダイアログ・ボックス内の表示順序に従って回復操作を実行します。次のいずれかのオプションを選択します。

➤ ［**次のボタンをクリックする**］：例外イベントが発生したときにクリックするボタンの論理名を指定します。標準のボタン名を選択するか，ボタンの論理名を入力するか，または指差しマークを使用してクリックするボタンを指定します。

注：

リストから標準のボタンを選択した場合，WinRunner がボタンを検索するウィンドウは選択した例外イベントのタイプによって異なります。ポップアップ例外イベントを選択した場合，WinRunner は定義したポップアップ・ウィンドウ

上のボタンを検索します。それ以外の例外イベントを選択した場合，WinRunner はアクティブな（フォーカスがある）ウィンドウ上のボタンを検索します。

このオプションをポップアップ例外イベントとともに使用する場合は，回復シナリオのために定義したポップアップ・ウィンドウを GUI マップに定義する必要があります。ポップアップ回復シナリオを定義するときに，ロード済みの GUI マップにポップアップ・ウィンドウが定義されていない場合は，回復シナリオが自動的に非アクティブに設定されます。その後，ポップアップ・ウィンドウを含む GUI マップをロードすると，回復シナリオをアクティブ化できます。

➤ ［**起動中のウィンドウを閉じる**］：例外イベントが発生したときにアクティブな（フォーカスがある）ウィンドウを閉じるよう WinRunner に指示します。

注： WinRunner は（TSL の）**win_close** メカニズムを使用してウィンドウを閉じます。**win_close** 関数でウィンドウを閉じることができない場合，回復シナリオはウィンドウを閉じることができません。

➤ ［**次の回復関数を実行する**］：例外イベントが発生したときに，指定した関数を実行するよう WinRunner に指示します。既存の関数を指定するか，または［**回復関数の定義**］をクリックして新しい関数を定義します。回復関数の定義の詳細については，649 ページ「回復シナリオ関数の定義」を参照してください。

注： テスト実行時に，この関数を含むコンパイル済みモジュールがロードされている必要があります。関数を回復用コンパイル済みモジュールに保存し，WinRunner が開いているときは関数が常に自動的にロードされるようにします。回復用コンパイル済みモジュールに保存されている関数を選択しない場合は，その関数を使用する回復シナリオがアクティブ化されたときに，その関数を含むコンパイル済みモジュールが必ずロードされるようにしてください。

➤ ［**プロセスを閉じる**］：［アプリケーション プロセスを閉じる］画面で指定したアプリケーション・プロセスを閉じるよう WinRunner に指示します。

➤ ［**コンピュータを再起動する**］：回復後操作を実行する前にコンピュータを再起動するよう WinRunner に指示します。

［**コンピュータを再起動する**］を選択する場合は，次の点に注意してください。

- 再起動オプションは，選択されたほかのすべての操作が実行された後でのみ実行されます。
- 再起動処理がスムーズに行われるように，［**次の回復関数を実行する**］オプションを使用し，保存されていないファイルを再起動前にすべて保存するステートメントを関数に追加することをお勧めします。また，自動的にログインするようにコンピュータが設定されていることを確認する必要があります。

注：回復シナリオの一部として再起動が発生すると，WinRunner 内で開いていたテストが自動的に閉じ，変更内容の保存を求めるメッセージが表示されません。

- 再起動オプションを選択した場合は，回復後操作を設定できません。
- 回復シナリオの中で WinRunner がコンピュータを再起動する前に，タイムアウト付きの警告メッセージが表示され，再起動操作を取り消す機会が与えられます。

- 再起動操作が実行されると，テストの最初からテスト実行が開始されます。例外の原因となったテストが別のテストから呼び出された場合は，呼び出しチェーンの最初からテスト実行が開始されます。例えば，テスト A がテスト B を呼び出し，テスト B がテスト C を呼び出し，テスト C の実行時に再起動回復操作を含む回復シナリオが呼び出された場合，WinRunner は再起動の実行後にテスト A の最初からテスト実行を開始します。
- 再起動操作を取り消した場合，WinRunner は例外が発生した位置からテストを続行しようとします。
- 再起動が発生する前にコマンド・ライン・オプションを使用して WinRunner を開いた場合，再起動操作の後で WinRunner が開くときに同じコマンド・ライン・オプションが適用されます。ただし，-t，-exp，および -verify の各オプションは除きます。WinRunner は再起動後に実行したテスト，およびそのテストの期待値と結果フォルダを使用します。

注：Quality Center からテストを実行する場合は，再起動の発生時に WinRunner が Quality Center から切断されるため，回復操作を使用する回復シナリオをアクティブ化しないでください。

［**次へ**］をクリックします。

［**プロセスを閉じる**］を選択した場合は，手順 8 に進みます。

［**プロセスを閉じる**］も［**コンピュータを再起動する**］も選択しなかった場合は，手順 9 に進みます。

［**プロセスを閉じる**］ではなく［**コンピュータを再起動する**］を選択した場合は，手順 10 に進みます。

8［アプリケーション プロセスを閉じる］画面が開きます。

例外イベントの発生時に WinRunner が閉じるアプリケーション・プロセスを指定します。WinRunner は，指定されたアプリケーション・プロセスのうち，回復シナリオの実行時にすでに（エラーが発生せずに）閉じているアプリケーション・プロセスを無視します。

アプリケーションをリストに追加するには，リスト内の次の空欄をダブルクリックし，アプリケーション名を入力するか，参照して選択します。または，

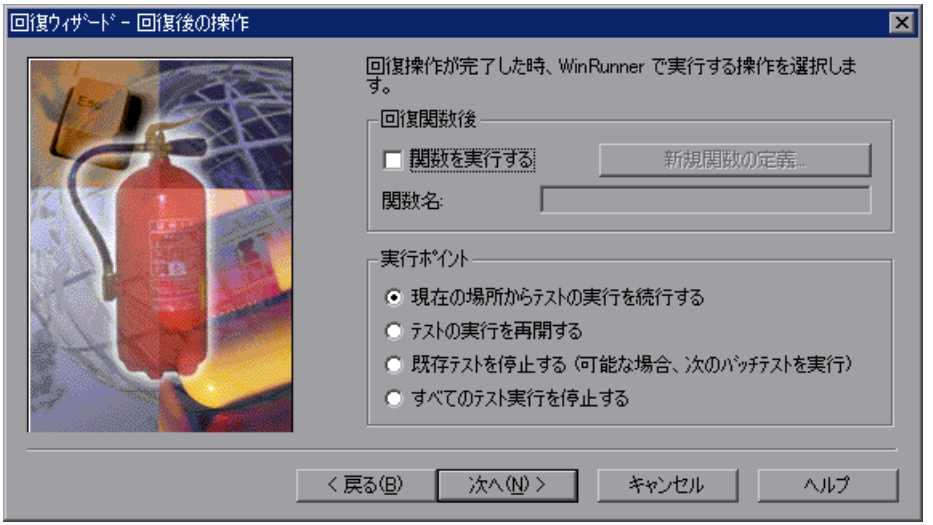
［**処理の選択**］をクリックして［プロセス リスト］を開きます。［プロセス リスト］には，現在実行されているプロセスのリストが表示されます。

このリスト内のプロセスを［アプリケーション プロセスを閉じる］リストに追加するには，該当するプロセスを選択して［**OK**］をクリックします。

注：指定するアプリケーション名には，**.exe** 拡張子を付ける必要があります。

［**次へ**］をクリックします。前の手順で［**コンピュータを再起動する**］を選択した場合は，手順 10 に進みます。それ以外の場合は，手順 9 に進みます。

9［回復後の操作］画面が開きます。



以下のオプションを選択します。

- ［**関数を実行する**］: 回復操作が完了したときに，指定した関数を実行するよう WinRunner に指示します。既存の関数を指定するか，または［**新規関数の定義**］をクリックして新しい関数を定義します。回復後関数の定義の詳細については，649 ページ「回復シナリオ関数の定義」を参照してください。

ヒント：テスト実行時に，この関数を含むコンパイル済みモジュールがロードされている必要があります。関数を回復用コンパイル済みモジュールに保存し，WinRunner が開いているときは関数が常に自動的にロードされるようにします。回復用コンパイル済みモジュールの詳細については，649 ページ「回復シナリオ関数の定義」を参照してください。

回復後関数は，回復処理中に閉じられたアプリケーションを再度開く場合や，アプリケーションを必要な状態に設定する場合に便利です。

- ［**実行ポイント**］: 回復操作および回復後操作（該当する場合）を実行した後の継続方法を WinRunner に指示します。次のいずれかを選択します。
  - ［**現在の場所からテストの実行を続行する**］: 現在のテストを例外が発生した位置から続行します。

- ［**テストの実行を再開する**］：現在のテストを最初から実行し直します。
- ［**既存テストを停止する（可能な場合，次のバッチ テストを実行）**］：現在のテスト実行を停止します。例外イベントが発生したテストがバッチ・テストから呼び出された場合は，バッチ・テストの次の行からテストを続行します。
- ［**すべてのテストの実行を停止する**］：テスト（およびバッチ）実行を停止します。

［**次へ**］をクリックします。

10 ［完了］画面が開きます。

WinRunner が開いているときに標準設定で回復シナリオをアクティブ化するかどうかを決定します。

- WinRunner が開いているときに標準設定で自動的に回復シナリオをアクティブ化するよう WinRunner に指示する場合は，［**標準設定で起動する**］を選択します。前の WinRunner セッションの終了時にシナリオが非アクティブに設定された場合でも，シナリオはアクティブ化されます。
- WinRunner が開いているときに標準設定で自動的に回復シナリオをアクティブ化しない場合は，［**標準設定で起動する**］をクリアします。このチェック・ボックスをクリアすると，［Recovery Manager］ダイアログ・ボックスでこのチェック・ボックスを手動で選択してアクティブ化しない限り，回復シナリオ

はアクティブ化されません。詳細については，654 ページ「回復シナリオのアクティブ化と非アクティブ化」を参照してください。

［**完了**］をクリックします。回復シナリオが［回復マネージャ］ダイアログ・ボックスの［**複合**］タブに追加されます。［標準設定で起動する］を選択した（および必要なオブジェクトがロード済みの GUI マップ・ファイルに見つかった）場合は，回復シナリオがアクティブ化されます。そうでない場合は，回復シナリオは非アクティブのままです。

回復シナリオ関数の定義

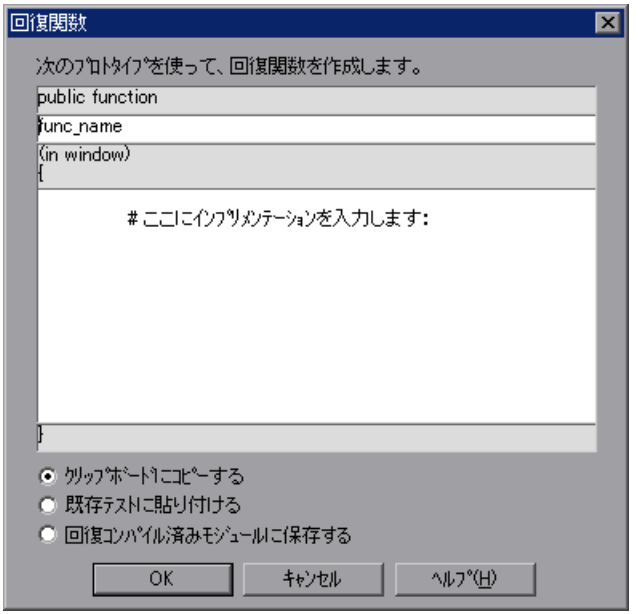
回復関数を定義すると，テストに関する特定のニーズに合わせて例外イベントに対応するよう WinRunner に指示できます。複合回復シナリオの場合は，回復後関数を定義することもできます。これらの関数は，例外発生時や回復処理中に閉じられた可能性があるアプリケーションを再度開く場合や，アプリケーションを必要な状態に設定する場合に便利です。

新しい回復関数および回復後関数を定義するには，回復ウィザードから開く［回復関数］ダイアログ・ボックスまたは［回復後関数］ダイアログ・ボックスを使用します。ダイアログ・ボックスには，選択した例外タイプに対応する構文と関数のプロトタイプが表示されます。

定義した回復関数は，回復用コンパイル済みモジュールに保存したり，現在のテストに貼り付けたり，クリップボードにコピーしたりできます。

回復関数または回復後関数を定義するには，次の手順を実行します。

1 ［回復作業の定義］画面で［**回復関数の定義**］をクリックするか，［回復後の操作］画面で［**新規関数の定義**］をクリックします。［回復関数］画面または［回復後関数］画面が開きます。



2 最初の 3 行に，関数の型（常に public function），関数名，および関数の引数が表示されます。func_name というテキストを新しい関数の名前に置き換えます。

3 実装ボックスに関数の内容を入力します。

4 関数の保存方法を次の中から選択します。

- ［**クリップボードにコピーする**］：関数をクリップボードにコピーします。
- ［**既存テストに貼り付ける**］：関数を現在のテストのカーソル位置に貼り付けます。
- ［**回復コンパイル済みモジュールに保存する**］：関数を回復用コンパイル済みモジュールに保存します。

注：

回復用コンパイル済みモジュールを［一般オプション］ダイアログ・ボックスの［**実行**］＞［**回復**］カテゴリで定義しなかった場合は，［**回復コンパイル済みモジュールに保存する**］オプションが無効になります。詳細については，658 ページ「回復用コンパイル済みモジュールの選択」を参照してください。

回復用コンパイル済みモジュールに関数を保存した場合は，変更された回復用コンパイル済みモジュールをロードするため，新しい関数を必要とする可能性があるテストを実行する前に，WinRunner を再起動するか，該当するコンパイル済みモジュールを手動で実行する必要があります。

関数を回復用コンパイル済みモジュールに保存しない場合は，その関数を使用する回復シナリオがアクティブ化されたときに，その関数を含むコンパイル済みモジュールが必ずロードされるようにしてください。

---

5 ［**OK**］をクリックして，回復ウィザードに戻ります。

回復シナリオの管理

回復シナリオを作成したら，回復マネージャを使用してシナリオを管理できます。回復マネージャによって，次のことが可能になります。

- 各回復シナリオのサマリを表示する
- 回復ウィザードを使用して既存の回復シナリオを変更する
- 既存の回復シナリオをアクティブ化または非アクティブ化する
- 回復シナリオを削除する

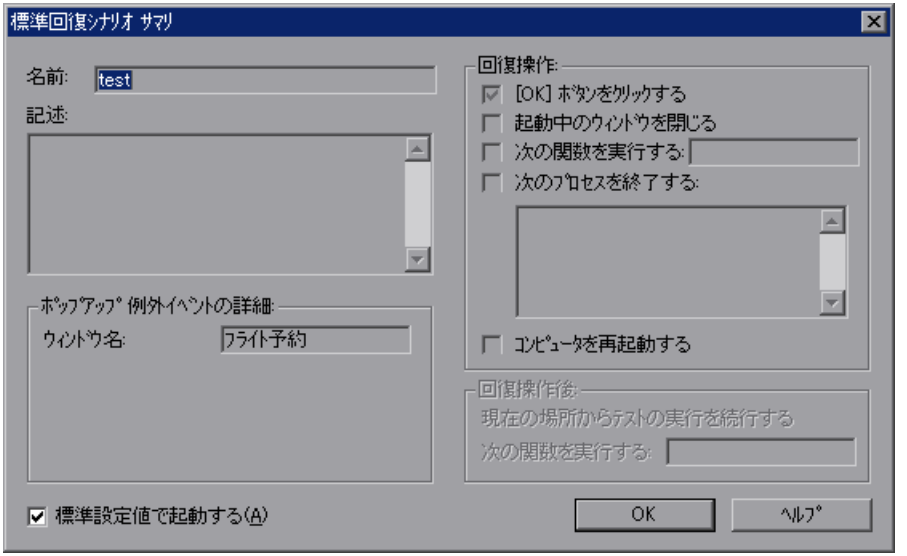
クラッシュ回復シナリオを使用する場合は，WinRunner がクラッシュ・ウィンドウを識別するために使用する文字列を変更することもできます。

回復シナリオの詳細の表示

［回復シナリオ サマリ］ダイアログ・ボックスには選択した回復シナリオの詳細が表示され，ここで［標準設定で起動する］の設定を簡単に変更できます。

[回復シナリオ サマリ] ダイアログ・ボックスを開くには，次の手順を実行します。

1 [回復マネージャ] ダイアログ・ボックスの [**標準**] タブまたは [**複合**] タブで回復シナリオを選択し，[**サマリ**] をクリックするか，回復シナリオ名をダブルクリックします。[(標準または複合) 回復シナリオ サマリ] ダイアログ・ボックスが開きます。



2 回復シナリオの設定を確認します。

3 [**標準設定で起動する**] の設定を変更する場合は，このチェック・ボックスを選択またはクリアします。詳細については，654 ページ「回復シナリオのアクティブ化と非アクティブ化」を参照してください。

回復シナリオの変更

既存の回復シナリオの詳細を変更するには，回復ウィザードの [変更] オプションを使用します。

回復シナリオを変更するには，次の手順を実行します。

1 [回復マネージャ] で，変更する回復シナリオを選択し，[**変更**] をクリックします。

2 回復ウィザードに [シナリオ名] 画面が表示されます。

注：既存の回復シナリオの例外イベント・タイプを変更することはできません。別の例外イベント・タイプを定義する場合は，新しい回復シナリオを作成してください。

3 回復ウィザード内を移動して，必要に応じて詳細を変更します。回復ウィザードのオプションの詳細については，623 ページ「簡易回復シナリオの定義」または 634 ページ「複合回復シナリオの定義」を参照してください。

回復シナリオの削除

既存の回復シナリオを削除するには，回復ウィザードの［削除］オプションを使用します。回復マネージャから回復シナリオを削除すると，回復シナリオ・ファイルからも対応する情報が削除されます。

回復シナリオ・ファイルの詳細については，558 ページ「回復オプションの設定」を参照してください。

回復シナリオを削除するには，次の手順を実行します。

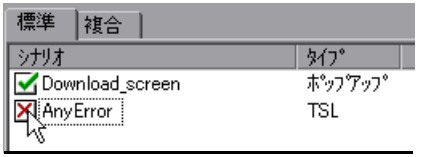
［回復マネージャ］で，削除する回復シナリオを選択し，［**削除**］をクリックします。

注：回復シナリオを削除しても，回復シナリオを定義したときに回復用コンパイル済みモジュールに保存された関数は削除されません。回復用コンパイル済みモジュールのサイズを調整するには，どの回復シナリオからも使用されなくなった関数を回復用コンパイル済みモジュールから削除してください。

回復シナリオのアクティブ化と非アクティブ化

WinRunner による例外イベントの識別と回復操作の実行は，アクティブな回復シナリオだけに対して行われます。回復シナリオは，次に示す複数の方法でアクティブ化または非アクティブ化できます。

- 回復シナリオの作成時に［**標準設定で起動する**］チェック・ボックスを選択またはクリアします。
- 回復シナリオを**一時的に**アクティブ化または非アクティブ化するには，回復マネージャの回復シナリオ名の横にあるアクティブ化チェック・ボックスを（1 回クリックして）切り替えます（［標準設定で起動する］オプションを設定すると，WinRunner が開くたびに回復シナリオがアクティブ状態または非アクティブ状態にリセットされます）。



- 回復マネージャで回復シナリオを選択し，［**サマリ**］をクリックするか回復シナリオをダブルクリックして［回復シナリオ サマリ］ダイアログ・ボックスを開き，［**標準設定で起動する**］チェック・ボックスを選択またはクリアします。
- 回復マネージャで回復シナリオを選択し，［**修正**］をクリックして回復ウィザードを開き，［完了］画面に移動して［**標準設定で起動する**］チェック・ボックスを選択またはクリアします。
- テスト実行中に回復シナリオをアクティブ化するには，TSL コマンドを使用します。これらの関数の詳細については，660 ページ「テスト・スクリプトの回復シナリオ・ファイルを使った作業」を参照してください。

クラッシュ・イベントのウィンドウ名の変更

WinRunner は，アプリケーションのクラッシュを示す文字列が入ったタイトル・バーを持つウィンドウが開いたときにクラッシュ・イベントを識別します。WinRunner がクラッシュ・ウィンドウを識別するために使用する文字列は，**＜ WinRunner のインストール先フォルダ＞ ¥dat** フォルダにある **excp_str.ini** ファイルで変更できます。

excp_str.ini ファイルは，Windows の各種言語に対応するセクションと，それ以外の言語用の標準設定セクションで構成されます。WinRunner は，使用中の Windows の言語に対応する文字列を使用してクラッシュ・イベントを識別します。

クラッシュ・イベントのウィンドウ名を変更するには，使用中の Windows の言語に対応するセクションに設定されたウィンドウ名を変更します。**excp_str.ini** ファイルの言語セクションは，3 文字の LOCALE_SABBREVLANGNAME コードによって識別されます。

使用中の Windows の言語が設定されていない場合は，使用するクラッシュ・イベント文字列を [DEF] セクションに入力します。または，使用中の Windows の言語を示す 3 文字の LOCALE_SABBREVLANGNAME をセクション区切りとして使用してファイルに新しいセクションを追加し，その下にクラッシュ・イベント文字列を引用符で囲んで（" **文字列** " のように）入力します。

次の表は，**excp_str.ini** に標準で含まれる各コードと，それらに対応する Windows の言語の一覧です。すべての言語コードの一覧については，MSDN のドキュメントを参照してください。

言語コード	Windows の言語
ENU	英語（米国）
JPN	日本語
KOR	韓国語
CHS	中国語（中国）
CHT	中国語（台湾）
DEU	ドイツ語（ドイツ）
SVE	スウェーデン語（スウェーデン）
FRA	フランス語（フランス）

回復シナリオ・ファイルを使った作業

回復シナリオを作成，変更，または削除すると，アクティブな回復シナリオ・ファイルに情報が保存されます。WinRunner を開くたびに，アクティブなファイルの情報が読み込まれ，そのファイルで定義された回復シナリオが回復マネージャに取り込まれます。複数の回復シナリオ・ファイルを作成し，必要に応じて WinRunner セッションごとに異なる回復シナリオを選択できます。

注：さまざまな回復シナリオの設定を切り替えることができるように，回復ファイルは回復情報を保存するためだけに使用されます。回復シナリオを作成，変更，または削除するには，回復マネージャと回復ウィザードを使用します。

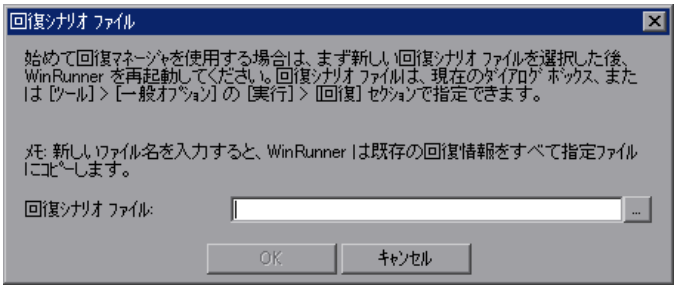
回復マネージャを初めて使用する

WinRunner バージョン 7.01 以前では，すべての「例外処理」の詳細が wrun.ini ファイルに保存されていました。したがって，**wrun.ini** ファイルは標準の回復シナリオ・ファイルです。

回復マネージャを初めて開くと，**wrun.ini** ファイルに定義された例外が回復マネージャの［**標準**］タブに表示され，以前のバージョンの WinRunner と同じように機能します。

ただし，回復マネージャを使用して回復シナリオを作成，変更，または削除するには，新しい回復シナリオ・ファイルを定義する必要があります。

回復シナリオを初めて作成，変更，または削除しようとしたときに開くダイアログ・ボックスで，ファイル名を入力できます。



または，回復マネージャを初めて使用する前に，［一般オプション］ダイアログ・ボックスの［**実行**］＞［**回復**］カテゴリで新しい回復シナリオ・ファイルを定義することもできます。

新しいファイル名を入力すると，新しいファイルが作成され，以前の **wrun.ini** に含まれていた例外情報がそのファイルにコピーされます。これで，回復マネージャを使用して既存の例外処理定義を引き続き操作できるようになります。回復シナリオ・ファイルとその選択方法の詳細については，次の「アクティブな回復シナリオ・ファイルの選択」を参照してください。

アクティブな回復シナリオ・ファイルの選択

アクティブな回復シナリオ・ファイルは，［一般オプション］ダイアログ・ボックスの［**実行**］＞［**回復**］カテゴリで選択します。既存のファイルを選択することも，新しいファイル名を入力することもできます。

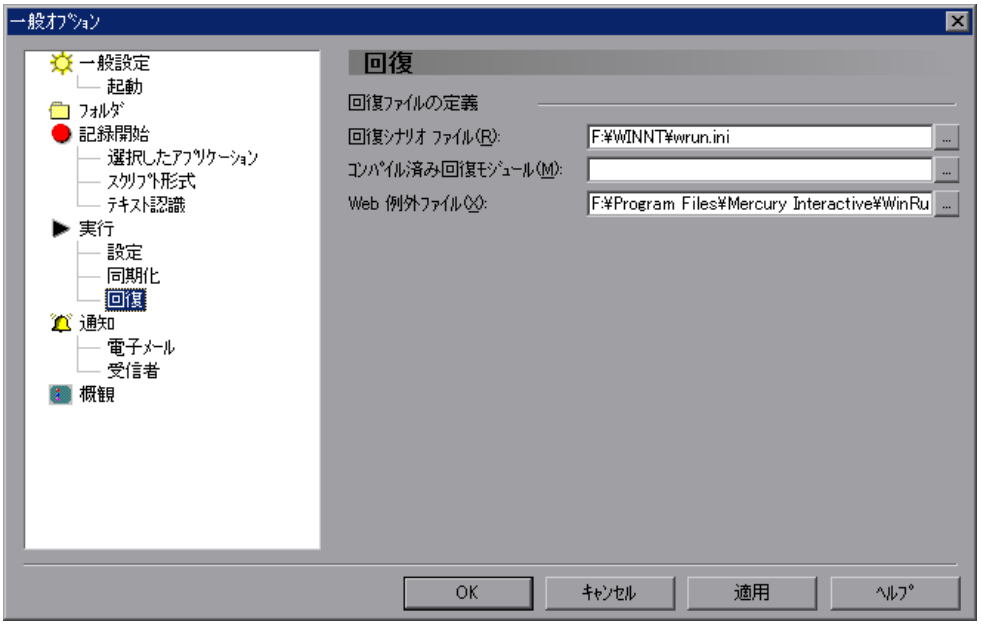
新しいファイル名を入力し，新しいファイルを作成することを確認すると，現在の回復シナリオ・ファイルに含まれるすべての回復シナリオ情報が新しいファイルにコピーされます。

既存の回復シナリオ・ファイルの名前を入力すると，選択したファイルがアクティブな回復シナリオ・ファイルとして設定されますが，直前の回復シナリオ・ファイルの情報はコピーされません。

アクティブな回復シナリオ・ファイルを選択するには，次の手順を実行します。

1 ［**ツール**］＞［**一般オプション**］を選択します。

2 [**実行**] > [**回復**] カテゴリをクリックします。[回復] オプション表示枠が表示されます。



3 [**回復シナリオ ファイル**] ボックスに，使用（または作成）するファイルのパスを入力するか，[参照] をクリックして既存の回復シナリオ・ファイルを選択します。

回復用コンパイル済みモジュールの選択

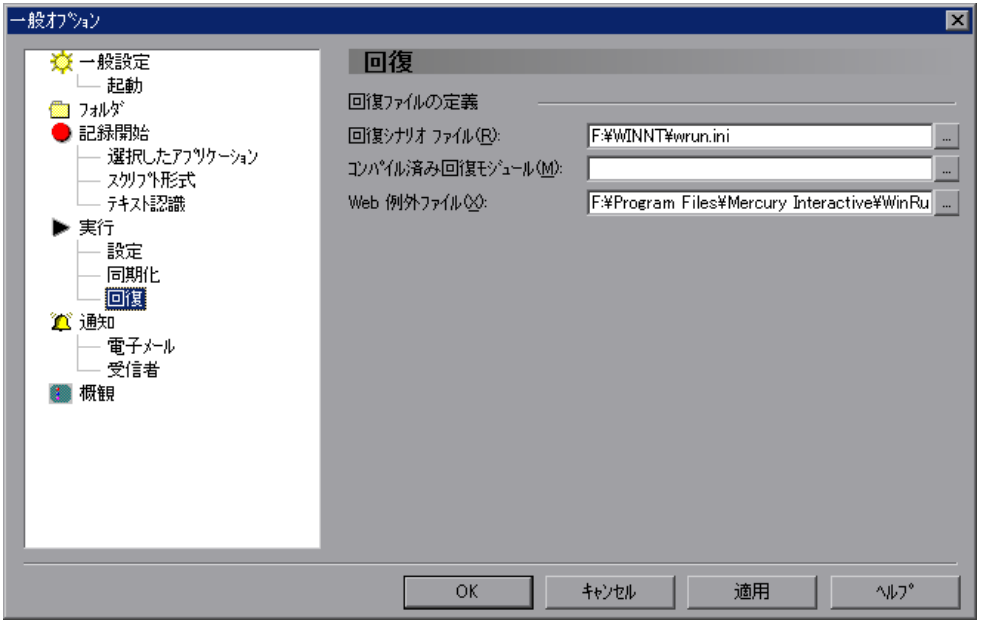
回復用コンパイル済みモジュールは，特別なコンパイル済みモジュールで，WinRunner が開いているときに常にロードされています。これによって，WinRunner が回復シナリオを実行するときはいつでもこのモジュールに含まれる関数にアクセスできます。

回復シナリオの作成時または編集時に [回復関数の定義] ダイアログ・ボックスまたは [回復後関数の定義] ダイアログ・ボックスで定義した関数を回復用コンパイル済みモジュールに直接保存するよう WinRunner に指示できます。また，回復用コンパイル済みモジュールを開き，手動でモジュールに関数を追加することもできます。

アクティブな回復用コンパイル済みモジュールを選択するには，次の手順を実行します。

1 ［**ツール**］＞［**一般オプション**］を選択します。

2 ［**実行**］＞［**回復**］カテゴリをクリックします。［回復］オプション表示枠が表示されます。



3 ［**コンパイル済み回復モジュール**］ボックスに，使用（または作成）するコンパイル済みモジュールのパスを入力するか，参照ボタンをクリックして既存のコンパイル済みモジュールを選択します。新しいファイル名を入力すると，新しいコンパイル済みモジュールが作成されます。

コンパイル済みモジュールの詳細については，第 31 章「テストでのユーザ定義関数の利用」を参照してください。

回復用コンパイル済みモジュール・ファイルの選択の詳細については，558 ページ「回復オプションの設定」を参照してください。

テスト・スクリプトの回復シナリオ・ファイルを使った作業

TSL ステートメントを使用すると，テスト実行中に特定の回復シナリオをアクティブ化または非アクティブ化したり，すべてのアクティブな回復シナリオを非アクティブ化したりできます。TSL を使用して簡易回復シナリオを定義することもできます。

テスト実行中の回復シナリオのアクティブ化と非アクティブ化

回復マネージャを使用すると，テストの設計中に回復シナリオをアクティブ化または非アクティブ化できますが，テストの実行中に回復シナリオのオン / オフを切り替える必要が生じる場合もあります。

例えば，回復関数を実行する回復シナリオを定義したとします。例外イベントによって回復シナリオが呼び出され，そのイベントに対応する例外関数の実行中に再度例外イベントが発生すると，回復シナリオが無限ループに陥る場合があります。このため，回復シナリオの回復関数の先頭で回復シナリオを非アクティブ化し，関数の最後で再度アクティブ化することをお勧めします。

特定の回復シナリオをアクティブ化または非アクティブ化するには，**exception_on** 関数および **exception_off** 関数を使用します。

例えば，次の回復関数は，テスト対象のアプリケーションを再度開くメイン回復スクリプトを実行する前に回復シナリオの処理を無効にします。その後，回復シナリオを再度有効にします。

```
public function label_handler(in win, in obj, in attr, in val)
{
# 回復関数の実行中は，この回復シナリオを無視する
exception_off("label_except");
report_msg("Label has changed");
menu_select_item ("File;Exit");
system ("flights&");
invoke_application ("flights", "", "C:¥¥FRS", "SW_SHOWMAXIMIZED");
#"attr" の値が "val" と一致しない場合
exception_on("label_except");
texit;
}
```

また，テストの実行中にすべての回復シナリオを非アクティブ化することもできます。例えば，特定の条件ステートメントが真である場合は回復シナリオを実行しないようにする必要が生じる場合があります。

すべてのアクティブな回復シナリオを非アクティブ化するには，**exception_off_all** 関数を使用します。

これらの機能の詳細については，「**TSL リファレンス**」を参照してください。

TSL を使った簡易回復シナリオの定義

テスト・スクリプトから，現在の WinRunner セッションでのみアクティブな新しい簡易回復シナリオを定義するには，**define_object_exception**，**define_popup_exception**，および **define_TSL_exception** を使用します。これは，戻り値を回復シナリオの入力として使用する場合に便利です。

上記のいずれかの関数を使用して簡易回復シナリオを定義すると，WinRunner セッション中は回復マネージャに簡易回復シナリオが表示され，回復ウィザードを使用して回復シナリオを変更できます。ただし，これらの回復シナリオは回復シナリオ・ファイルに保存されず，WinRunner を再起動すると回復マネージャに表示されなくなります。

クラッシュ・イベントや複数の回復操作を定義できる複合回復シナリオを作成するには，回復マネージャを使用します。詳細については，634 ページ「複合回復シナリオの定義」を参照してください。

TSL を使った簡易回復シナリオの定義の詳細については，「**TSL リファレンス**」を参照してください。

第 27 章

正規表現の使い方

正規表現を使って，テストの柔軟性と適用性を高めることができます。

本章では，以下の項目について説明します。

- 正規表現について
- どのようなときに正規表現を使うか
- 正規表現の構文について

正規表現について

WinRunner では正規表現を使ってさまざまな名前やタイトルを持つオブジェクトを識別できます。正規表現は TSL ステートメントの中や，GUI マップの中のオブジェクト記述で使うことができます。例えば，プッシュ・ボタンの物理的記述の中で正規表現を定義して，プッシュ・ボタンのラベルが変わっても WinRunner がそれを見つけられるようにできます。

正規表現は，複雑な検索句を指定する文字列です。多くの場合，文字列の前には感嘆符（!）が付けられます。（**match** 関数のように，文字列で表される関数の記述や引数では，感嘆符は必要ありません。）ピリオド（.），アスタリスク（*），キャレット（^），および大括弧（[]）などの特殊文字を使って，検索条件を指定できます。例えば，文字列「!windo.*」は「window」にも「windows」にも一致します。詳しくは，667 ページ「正規表現の構文について」を参照してください。

どのようなときに正規表現を使うか

テストを実行するたびに GUI オブジェクトの名前が変わるようなときに正規表現を使います。例えば，正規表現を次のように使います。

- GUI マップ内のオブジェクトの物理的記述
- GUI チェックポイント。さまざまな名前を持つ編集オブジェクトや静的テキスト・オブジェクトの内容を評価するときに使います。
- テキスト・チェックポイント。**win_find_text** や **obj_find_text** を使用してさまざまなテキストの文字列を見つけます。

GUI マップ内のオブジェクトの物理的記述の正規表現の使用

GUI マップのオブジェクトの物理的記述で正規表現を使うことができます。これによって WinRunner が，オブジェクトのラベルの変化を無視するようにします。例えば，次のような物理的記述を使えば，ラベルが「Start」から「Stop」に変わっても，WinRunner はそのプッシュ・ボタンを識別することができます。

```
{
class: push_button
label: "!St.*"
}
```

GUI チェックポイントでの正規表現の使用

さまざまな名前を持つ編集オブジェクトや静的テキスト・オブジェクトの内容を評価するとき，GUI チェックポイントで正規表現を使うことができます。検査を指定するオブジェクトの GUI チェックポイントを作成することによって，正規表現を定義します。下で，正規表現の使い方について説明します。例えば，［**挿入**］＞［**GUI チェックポイント**］＞［**オブジェクト / ウィンドウ**］を選択して，静的テキスト・オブジェクトをダブルクリックします。［**挿入**］＞［**GUI チェックポイント**］＞［**複数のオブジェクト**］コマンドで正規表現を使用することもできます。GUI チェックポイントの詳細については，第 9 章「GUI オブジェクトの検査」を参照してください。

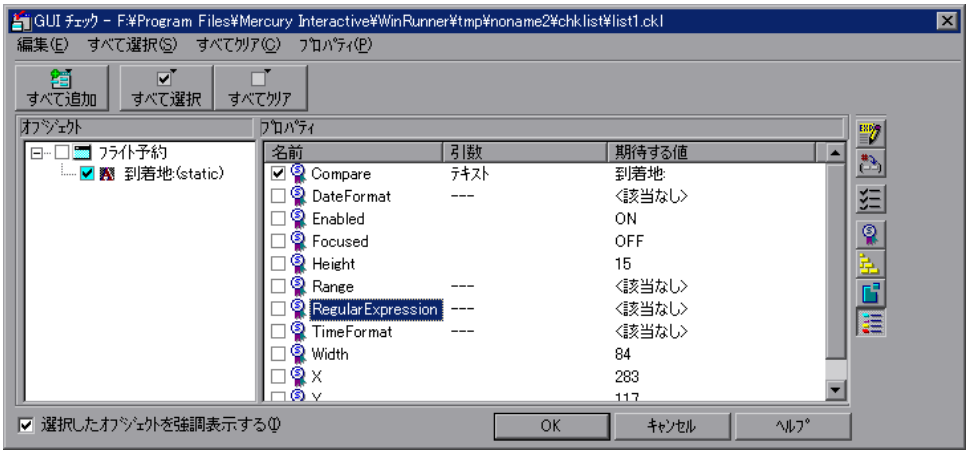
GUI チェックポイントの正規表現を定義するには，次の手順を実行します。

1 検査を指定するオブジェクトの GUI チェックポイントを作成します。この例では，［**挿入**］＞［**GUI チェックポイント**］＞［**オブジェクト / ウィンドウ**］を選択します。

WinRunner のウィンドウが最小化されます。マウス・ポインタが指差し型に変わり，画面にヘルプ・ウィンドウが表示されます。

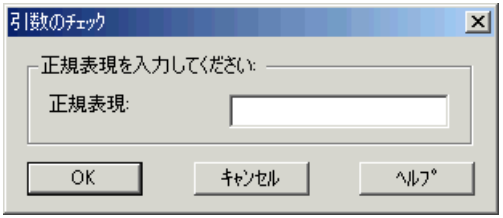
2 静的テキスト・オブジェクトをダブルクリックします。

3 ［**GUI チェック**］ダイアログ・ボックスが表示されます。



4 ［**プロパティ**］**表示枠で**，「Regular Expression」プロパティ・チェックを選択したら，［**引数を指定**］ボタンをクリックします。

［**引数のチェック**］ダイアログ・ボックスが表示されます。



5 ［**正規表現**］ボックスに正規表現を入力して，［**OK**］をクリックします。

注：正規表現を使って静的テキスト・オブジェクトまたは編集オブジェクトの検査を行うときには，その前に感嘆符を付けてはなりません。

---

6 希望どおりに定義されたら，実行する追加の任意の検査を指定し，［**OK**］をクリックして［**GUI チェック**］ダイアログ・ボックスを閉じます。

obj_check_gui ステートメントがテスト・スクリプトに挿入されます。

引数の指定に関する詳細については，167 ページ「プロパティ検査への引数の指定」を参照してください。

テキスト・チェックポイントで正規表現を使用する

テキスト・チェックポイントで，**win_find_text** または **obj_find_text** を使ってさまざまなテキスト文字列を見つけることができます。例えば，次のステートメントによって，WinRunner は「Edit」と名付けられたオブジェクト内の「win」で始まるあらゆるテキストを見つけることができます。

```
obj_find_text ("Edit", "win.*", coord_array, 640, 480, 366, 284);
```

ウィンドウはその時々でさまざまなラベルを持つので，WinRunner はウィンドウの物理的記述で正規表現を定義します。詳しくは第 7 章「GUI マップの編集」を参照してください。

正規表現の構文について

正規表現は，［GUI チェック］ダイアログ・ボックス，テキスト・チェックポイント，あるいは **match**，**obj_find_text**，**win_find_text** のどれかのステートメントで定義するときを除いては，感嘆符（!）で始めなければなりません。正規表現内のすべての文字は文字通りに検索されますが，ピリオド（.），アスタリスク（*），キャレット（^），大括弧（[]）は例外です。これらについては以下で説明します。これらの特殊文字の前にバックスラッシュが付くと，WinRunner はその 1 文字を単なる文字として検索します。例えば，**win_find_text** ステートメントを使用して「Sign up now!」で始まるフレーズを検索する場合は，正規表現を使って「Sign up now¥!*」というように指定します。

以降で説明するオプションを使用して，正規表現を作成できます。

任意の 1 文字と一致

ピリオド（.）は WinRunner に任意の 1 文字を検索するように指示します。例えば，

```
welcome.
```

は，welcomes，welcomed，あるいは後にスペースまたは他の 1 文字が続く welcome に一致します。連続するピリオドは，一連の任意の文字を示します。

範囲内の任意の 1 文字と一致

ある範囲内の 1 文字と一致させるために，大括弧（[]）を使うことができます。例えば，1968 年または 1969 年の日付を探すために，次のように書くことができます。

```
196[89]
```

ハイフン（-）を使って実際の範囲を示すことができます。例えば，1960 年代のすべての年に一致させるには，次のように書きます。

```
196[0-9]
```

大括弧を物理的記述の中で使って変化する静的テキスト・オブジェクトのラベルを指定することができます。

```
{
class: static_text,
```

```
label: "!Quantity[0-9]"
}
```

上の例では，WinRunner はラベルが「Quantity」の static_text オブジェクトを，数が変わっても識別できます。

ハイフンは大括弧内の先頭か末尾，あるいはキャレット（^）の後にある場合には範囲を表しません。

キャレット（^）は，WinRunner に，文字列中で指定されている以外のあらゆる文字との一致を指示します。例えば，

```
[^A-Za-z]
```

はアルファベット以外のすべての文字に一致します。キャレットがこの特別な意味を持つのは，大括弧内の先頭に現れたときだけです。

大括弧内では文字「.」，「*」，「[」，「¥」も単なる文字として扱われます。大括弧内の範囲を表す最初の文字が右括弧である場合には，それもまた単なる文字として扱われます。例えば，

```
[]g-m]
```

は“]”と g から m までの文字に一致します。

注：「¥」文字を続けて 2 つ使用しても（「¥¥」），それは「¥」文字 1 つと判断されます。例えば，GUI マップの物理的記述では，「!D:¥¥.*」は「D:¥」で始まるすべてのラベルを意味しているわけではありません。むしろ，「D:.」で始まるすべてのラベルを参照します。「D:¥」で始まるすべてのラベルを指定するには，正規表現「!D:¥¥¥¥.*」を使います。

特定の文字との一致

アスタリスク（*）は，WinRunner に，その直前の文字の 0 回以上の出現に一致するように指示します。例えば，

```
Q*
```

の場合，WinRunner は Q，QQ，QQQ，などに一致します。

後ろにアスタリスク「*」の付くピリオド「.」は，何らかの文字列が後ろに付く文字列を探すよう WinRunner に指示します。

例えば，次の物理的記述では，正規表現によって WinRunner は「O」で始まる任意のプッシュ・ボタン（例えば，On や Off）を見つけることができます。

```
{
class: push_button
label: "!O.*"
}
```

また，大括弧とアスタリスクの組み合わせを使って，ラベルを数字以外の文字の組み合わせだけに限定することもできます。以下に例を示します。

```
{
class: push_button
label: "!O[a-zA-Z]*"
}
```

第 8 部

TSL を使ったプログラミング

第 28 章

プログラミングによるテスト・スクリプトの機能強化

WinRunner のテスト・スクリプトは，Mercury Interactive 社のテスト・スクリプト言語（TSL）でコーディングされたステートメントで構成されます。本章では，TSL について簡単に紹介し，いくつかの簡単なプログラミング技術を使って，テスト・スクリプトの機能を強化する方法を説明します。

本章では，以下の項目について説明します。

- プログラミングによるテスト・スクリプトの機能強化について
- 記述的プログラミングの使用
- コメントと空白の追加
- 定数と変数について
- 計算の実行
- 負荷条件の作成
- 条件判断ステートメントの組み込み
- テスト結果ウィンドウへのメッセージの送信
- テスト・スクリプトからのアプリケーションの起動
- テスト・ステップの定義
- 2 つのファイルの比較
- TSL スクリプトの構文チェック

プログラミングによるテスト・スクリプトの機能強化について

テストを記録すると，Mercury 独自のテスト・スクリプト言語（TSL）でテスト・スクリプトが生成されます。WinRunner がテスト・スクリプトで生成する各行が「**ステートメント**」です。ステートメントはセミコロンが後に付く任意の式です。テスト・スクリプトの各 TSL ステートメントは，テスト対象アプリケーションへのキーボードやマウスによる入力を示します。1 つのステートメントはテスト・スクリプトの 1 行よりも長い場合があります。

次に例を示します。

```
if (button_check_state("Underline", OFF) == E_OK)
    report_msg("Underline check box is unavailable.");
```

TSL は，テスト・スクリプトを作成するために設計された C 言語に似たプログラミング言語です。この言語は，テスト用に開発された関数と，変数，フロー制御ステートメント，配列，およびユーザ定義関数など，汎用のプログラミング言語機能を組み合わせたものです。テスト・ウィンドウにプログラミング要素を入力して，記録したテスト・スクリプトを機能強化できます。テスト・スクリプトをテスト・ウィンドウに直接入力してプログラミングする場合は，各ステートメントの最後にセミコロンを必ず付けてください。

TSL はコンパイルの必要がないため，簡単に使用できます。テスト・スクリプトを記録または入力するだけで，すぐにテストを実行できます。

TSL には，以下の 4 種類の TSL 関数があります。

- 「*コンテキスト・センシティブ*」関数は，ボタンのクリック，リスト項目の選択など，GUI オブジェクトに対する操作を行います。**button_press**，**list_select_item** などのように，関数名がその機能を示します。
- 「*アナログ*」関数は，マウス・クリック，キーボード入力，およびマウスが移動した正確な座標を再現します。
- 「*標準*」関数は，レポートへのメッセージの送信，計算の実行など，基本的なプログラミング処理を行います。
- 「*カスタマイズ*」関数は，テスト環境に合わせて WinRunner を構成するための関数です。

WinRunner には，TSL 関数をすばやく簡単にテストに追加できるビジュアルなプログラミング・ツールがあります。詳細については，以下のマニュアルを参照してください。を参照してください。

- 「**TSL リファレンス**」には，TSL 言語の概要，各関数，使用例，関数の使用可能範囲，TSL を使って作業するためのガイドラインなどが含まれます。オンライン・リファレンスは，［**ヘルプ**］＞［**TSL オンライン リファレンス**］を選択して開きます。テスト・スクリプトで TSL ステートメントにカーソルを置き，F1 キーを押して関数のヘルプ・トピックを直接表示することもできます。また，コンテキスト・センシティブの［ヘルプ］ボタンをクリックしてから，テスト・スクリプトの TSL ステートメントをクリックして参照することもできます。
- 『**Mercury テスト・スクリプト言語リファレンス**』には，TSL 言語の概要，各関数，関数の使用可能範囲，TSL を使って作業するためのガイドラインなどが含まれています。この本は，WinRunner の関連マニュアルに含まれています。簡単に印刷できる PDF バージョンにアクセスすることもできます。［**ヘルプ**］＞［**印刷用ドキュメント**］を選択して，WinRunner オンライン文書のホームページから［**WinRunner テスト・スクリプト言語リファレンス**］をクリックします。

本章では，プログラミングの基本的な概念を紹介し，いくつかの簡単なプログラミング技術を使って，より強力なテストを作成する方法を説明します。TSL の詳細については，「**TSL リファレンス**」を参照してください。

記述的プログラミングの使用

GUI マップにオブジェクトを追加すると，WinRunner はオブジェクトに論理名を割り当てます。オブジェクトが GUI マップに存在する場合は，そのオブジェクトを対象に実行するテストにステートメントを追加できます。ステートメントを追加するには，通常はオブジェクト記述にオブジェクトの論理名を入力します。

例えば，次のステートメントでは，**フライト予約**がウィンドウの論理名で，**ファイル**；**注文を開く**はメニューの論理名です。

```
set_window (" フライト予約 ", 5);
menu_select_item (" ファイル ; 注文を開く ...");
```

GUI マップの各オブジェクトには一意の論理名 があるので，オブジェクトを記述するのには論理名だけで十分です。テストの実行中，WinRunner は論理名に基づいて GUI マップ内のオブジェクトを検索し，オブジェクトに格納されているその他のプロパティを使用してアプリケーション内のオブジェクトを特定します。

注：GUI マップ内のオブジェクトの論理名を変更して，オブジェクトをテスト内で特定しやすくすることができます。詳細については，77 ページ「論理名と物理的記述の修正」を参照してください。

GUI マップを参照せずにオブジェクトに対して関数を実行するステートメントを追加することもできます。これを行うには，テスト実行中に WinRunner がオブジェクトを特定できるよう，オブジェクトにより多くの情報を入力する必要があります。これを「**記述的プログラミング**」といいます。

例えば，フライト予約アプリケーションで注文を記録したとします。そしてテストを作成後，その注文にラジオ・ボタン・グループが追加されたとします。これらのオブジェクトを GUI マップに追加するために既存のテストに新規ステップを記録する代わりに，選択するラジオ・ボタンを記述するステートメントをスクリプトに追加して，ラジオ・ボタンの状態を ON に設定できます。

オブジェクト・クラスである MSW_class と，オブジェクトを一意に識別するのに必要なプロパティと値の組を必要なだけ定義することによって，オブジェクトを記述します。

一般的な構文は次のとおりです。

function_name("{ class: class_value , MSW_class: MSW_value , property3: value , ... , propertyX: value }" , other_function_parameters) ;

function_name：オブジェクトに対して実行する関数。

property:value：オブジェクトのプロパティと値。プロパティと値の組はカンマで区切る必要があります。

other_function_parameters：オブジェクト・パラメータに論理名を使用する場合に，他の必須または任意のパラメータをステートメントに入力します。

オブジェクト記述全体は，二重引用符で囲む必要があります。

例：**“{description}”.**

例えば，次のステートメントは，ビジネス・クラスの座席を選択するために，**button_set** 関数をラジオ・ボタンに対して実行します。テストの実行時に WinRunner は一致するプロパティ値を持つラジオ・ボタン・オブジェクトを検索しそれを選択します。

```
set_window ("Flight Reservation", 3);
button_set ("{class: radio_button, MSW_class: Button, label: Business}", ON);
```

オブジェクトを識別するのに使用できるプロパティと値が分からない場合は，［GUI スパイ］を使用して，オブジェクトの現在のプロパティと値を表示します。詳細については，34 ページ「GUI オブジェクトのプロパティの表示」を参照してください。

コメントと空白の追加

プログラミングの際には，テスト・スクリプトを読みやすく，分かりやすくするために，コメントや空白を追加できます。

コメントの使用

コメントは，テスト・スクリプトにおいて，シャープ記号（#）から始まる行または行の一部です。テストを実行した場合，TSL のインタープリタは，コメントを処理しません。テスト・スクリプトを読みやすく，アップデートしやすくするためには，テスト・スクリプトのセクションを説明するコメントを追加することをお奨めします。

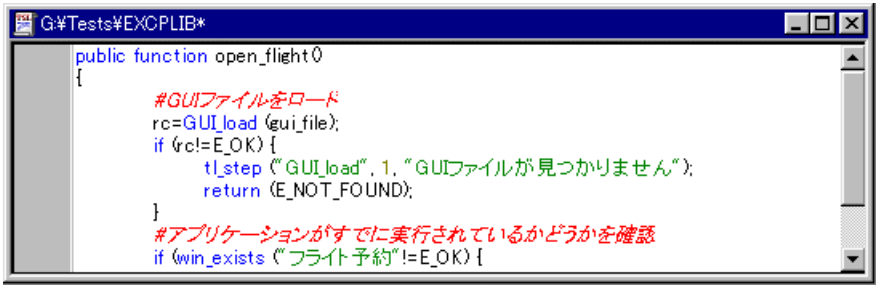
以下に例を示します。

```
# フライト予約アプリケーションの［注文を開く］ウィンドウを開く。
set_window (" フライト予約 ", 10);
menu_select_item (" ファイル ; 注文を開く ");

# James Brown の予約を選択する。
set_window (" 注文を開く _1");
button_set (" 顧客名 ", ON);
edit_set ("Value", "James Brown"); # James Brown と入力する。
button_press ("OK");
```

［一般オプション］ダイアログ・ボックスの［**記録開始**］＞［**スクリプト形式**］カテゴリの［**コメントとインデントのステートメントを挿入する**］オプションを使って，ウィンドウのフォーカスの変更を記録中，テスト・スクリプトを自動的にセクションに分割するように設定できます。このオプションを選択する際，WinRunner は自動的に，ウィンドウの名前のついたコメントをセクションの始まりに挿入し，各コメントの下にステートメントを字下げして表示します。［**コメントとインデントのステートメントを挿入する**］オプションについては，537 ページ「スクリプト形式オプションの設定」を参照してください。

空白の挿入

テスト・スクリプトにおける空白とは，スペース，タブ，空白行を指します。TSL のインタープリタは，空白がリテラル文字列の一部でない限り，空白の数や種類は無視します。空白を使えば，テスト・スクリプトのロジックが明らかになります。



定数と変数について

TSL ステートメントでは，定数と変数を使ってデータを格納します。定数とは，不変の値を指します。数値，文字，文字列などの定数があります。一方変数は，テストを実行するたびに値が変わる可能性があります。

変数および定数の名前には，英数字およびアンダーバー（_）が使えます。先頭の文字は，英字またはアンダーバーでなければなりません。TSL は大文字と小文字を区別します。したがって，y と Y は別の文字と認識されます。いくつかの単語は，TSL によって予約されており，名前として使うことはできません。

関数定義の外で使う変数は，型を決定するために宣言する必要がありません。変数が宣言されない場合，WinRunner はテストの実行時に型（auto，static，public，extern）が決められます。

例えば，以下のステートメントでは，テキスト・ボックスに現れるテキストを格納する変数を使用しています。

```
edit_get_text ("Name:", text);
        report_msg ("The Customer Name is " & text);
```

WinRunner は，［Name］テキスト・ボックスに現れる値を読み取り，それを変数 *text* に格納します。**report_msg** ステートメントを使って，変数 text の値をレポートに書き込んでいます。詳細については，686 ページ「テスト結果ウィンドウへのメッセージの送信」を参照してください。変数と定数の宣言に関する情報については，第 30 章「ユーザ定義関数の作成」を参照してください。

計算の実行

算術演算子を使って，簡単な計算を行うテストを作成できます。例えば，乗算演算子を使って，アプリケーションの 2 つのテキスト・ボックスに表示される値の乗算を行うことができます。TSL は，以下の算術演算子をサポートします。

+	加算
-	減算
-	減算（単項演算子）
*	乗算
/	除算
%	係数
^ または **	指数
++	インクリメント（オペランドに 1 を加算します。単項演算子）
--	デクリメント（オペランドから 1 を減算します。単項演算子）

TSL は，この他に，連結演算子，関係演算子，論理演算子，条件演算子，代入演算子の 5 種類の演算子をサポートします。また，**sin** や **exp** など，複雑な計算を行うための関数も用意されています。詳細については，「**TSL リファレンス**」を参照してください。

以下の例では，フライト予約アプリケーションを使います。WinRunner は，エコノミー・クラスとビジネス・クラスの航空券の価格を読み取ります。その後，それらの価格の差が 100 ドルを超えるかどうかを調べます。

```
# ［エコノミー］ボタンを選択する。
set_window (" フライト予約 ");
button_set (" エコノミー ", ON);

# 価格フィールドからエコノミー・クラスのチケットの金額を取得する。
edit_get_text (" 単価 :", economy_price);

# ［ビジネス］をクリックします。
button_set (" ビジネス ", ON);

# 価格フィールドからビジネス・クラスのチケットの金額を取得する。
edit_get_text (" 単価 :", business_price);

# 価格差が 100 ドル以上かどうか調べる。
if ((business_price - economy_price) > 100)
tl_step ("Price_check", 1, "Price difference is too large.");
```

負荷条件の作成

テスト・スクリプトで負荷条件を作成して，アプリケーションの限界を試すことができます。テスト・スクリプトの一部を指定された回数だけ繰り返し実行するループを作成することで，負荷条件を作成します。TSL では，ループを作成するためのステートメントが 3 つあります。*for*，*while* および *do/while* です。ループ内では定数を定義することはできません。

for ループ

for ループは，WinRunner に対して，1 つまたは複数のステートメントを指定された回数だけ実行するよう指示します。構文は次のとおりです。

```
for ( [ expression1 ]; [ expression2 ]; [ expression3 ] )
    statement
```

まず，*expression1* が実行されます。次に，*expression2* が評価されます。*expression2* が真の場合，*statement* が実行された後，*expression3* が実行されます。このサイクルは，*expression2* が真であり続ける限り繰り返されます。*expression2* が偽の場合，*for* ステートメントが終了し，直後に続くステートメントに実行が移ります。

例えば，次に示す *for* ループでは，［Open］ウィンドウの［File Name］リストから UI_TEST というファイルを選択しています。ファイルを 5 回選択すると，ループが終了します。

```
set_window ("Open")
for (i=0; i<5; i++)
    list_select_item ("File Name:_1", "UI_TEST"); # Item Number 2
```

while ループ

while ループは，指定された条件が真である限り，ステートメントのブロックを実行を繰り返します。構文は次のとおりです。

while (*expression*)
    statement ;

expression が真である限り，statement が実行されます。*expression* が偽になると，ループが終了します。

例えば，以下に示す *while* ステートメントは，上記の *for* ループと同じ動作をします。

```
set_window ("Open");
i=0;
while (i<5)
    {
    i++;
    list_select_item ("File Name:_1", "UI_TEST"); # Item Number 2
    }
```

do/while ループ

do/while ループは，指定された条件が真である限り，ステートメントのブロックの実行を繰り返します。*for* ループおよび *while* ループとは異なり，このループは，ループの先頭ではなく，ループの終わりで条件を調べます。*do/while* ループの構文は，次のとおりです。

```
do
        statement
while (expression);
```

まず，statement が実行され，その後，*expression* が評価されます。expression が真の場合，サイクルが繰り返されます。*expression* が偽になると，サイクルは繰り返されません。

例えば，以下の *do/while* ステートメントは，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・ボックスを 5 回開いて閉じます。

```
set_window (" フライト予約 ");
i=0;
do
    {
    menu_select_item (" ファイル ; 注文を開く ");
    set_window (" 注文を開く ");
    button_press (" キャンセル ");
    i++;
    }
while (i<5);
```

条件判断ステートメントの組み込み

if/else ステートメントまたは *switch* ステートメントを使って，テスト・スクリプトの中で条件判断を行うことができます。

if/else ステートメント

if/else ステートメントは，条件が真の場合，特定のステートメントを実行し，そうでない場合は，別のステートメントを実行します。構文は次のとおりです。

```
if ( expression )
    statement1;
[ else
    statement2; ]
```

まず，*expression* が評価されます。*expression* が真の場合，*statement1* が実行されます。*expression1* が偽の場合，*statement2* が実行されます。

例えば，以下の *if/else* ステートメントは，［フライト予約］ウィンドウの［フライト］ボタンが使用可能かどうかを調べます。その後，レポートに適切なメッセージを送信します。

```
# 新規注文を開く。
set_window (" フライト予約 _1");
menu_select_item (" ファイル ; 新規注文 ");

#［フライト予定日：］ボックスに日付を入力する。
edit_set_insert_pos (" フライト予定日：", 0, 0);
type ("120196");

#［出発地：］ボックスに値を入力する。
list_select_item (" 出発地：", "Portland");

#［到着地：］ボックスに値を入力します。
list_select_item (" 到着地：", "Denver");

#［フライト］ボタンが有効かどうかを調べる。
button_get_state (" フライト ", value);
if (value != ON)
    report_msg ("［フライト］ボタンが有効になりました ");
else
    report_msg ("［フライト］ボタンが有効になりませんでした。
       ［出発地：］と［到着地：］の値が有効かどうか確かめてください。");
```

switch ステートメント

WinRunner は，*switch* ステートメントにより，複数の値を持つことのできる式に基づいて条件判断を行うことができます。構文は次のとおりです。

```
switch (expression)
{
    case case_1:
        statements
    case case_2:
        statements
    case case_n:
        statements
default: statement(s)
}
```

switch ステートメントは，最初の expression と同じ値を持つ式が見つかるまで，各 case 式を順番に評価します。case 式で expression に等しいものがない場合，default ステートメント（statement(s)）が実行されます。default ステートメントは省略可能です。

指定された最初の expression に等しい case 式が見つかると，以降の case 式は評価されません。ただし，*break* ステートメントを使用して，*switch* ステートメントの直後に続く最初のステートメントに実行を渡さない限り，これらの case 式に指定されている以降のステートメントがすべて実行されてしまいます。

以下に示すテストは，フライト予約アプリケーションを対象としています。このテストは，［ファースト］，［ビジネス］，［エコノミー］の各ボタンをランダムに選択します。その後，適切な GUI チェックポイントを使って，航空券の正しい価格が［単価］テキスト・ボックスに表示されているかどうか検証します。

```
arr[1]=" ファースト ";arr[2]=" ビジネス ";arr[3]=" エコノミー ";
while(1)
{
    num=int(rand()*3)+1;

    # クラスのボタンをクリックする。
    set_window (" フライト予約 ");
    button_set (arr[num], ON);

    # 選択したボタンに対する航空券の価格を調べる。
    switch (num)
```

```
{
    case 1: # ファースト
    obj_check_gui(" 単価 :", "list1.ckl", "gui1", 1);
    break;
    case 2: # ビジネス
    obj_check_gui(" 単価 :", "list2.ckl", "gui2", 1);
    break;
    case 3: # エコノミー
    obj_check_gui(" 単価 :", "list3.ckl", "gui3", 1);
    }
}
```

テスト結果ウィンドウへのメッセージの送信

テスト・スクリプトの中でメッセージを定義し，WinRunner にそのメッセージをテスト結果ウィンドウに送信させることができます。テスト結果ウィンドウにメッセージを送信するには，テスト・スクリプトに **report_msg** ステートメントを追加します。構文は次のとおりです。

report_msg (*message* **);**

message には，文字列または変数のいずれか一方または両方を指定できます。

以下の例では，WinRunner は［フライト予約］ウィンドウの label 属性の値を取得し，メッセージと label 属性の値を含むテスト結果にステートメントを送信します。

```
win_get_info(" フライト予約 ", "label", value);
report_msg(" ウィンドウのラベル : " & value);
```

テスト・スクリプトからのアプリケーションの起動

invoke_application 関数を使って，WinRunner テスト・スクリプトからアプリケーションを起動できます。例えば，起動テストに **invoke_application** ステートメントを追加すると，WinRunner を起動するたびにテスト対象アプリケーションも開くことができます。起動テストの詳細については，第 45 章「特殊な構成の初期化」を参照してください。

ヒント : ［テストのプロパティ］ダイアログ・ボックスの［実行］タブを使用して，テスト実行の最初にアプリケーションを起動できます。詳細については，510 ページ「起動アプリケーションおよび起動関数の定義」を参照してください。
system ステートメントを使用して，アプリケーションを起動することもできます。詳細については，『**WinRunner テスト・スクリプト言語リファレンス**』を参照してください。

---

invoke_application 関数の構文は，次のとおりです。

invoke_application (*file, command_option, working_dir, show* **);**

file パラメータには，起動するアプリケーションの完全パス名を指定します。*command_option* パラメータには，適用するコマンドライン・オプションを指定します。*work_dir* パラメータには，アプリケーションの作業ディレクトリを指定し，*show* には，アプリケーションを起動したときのメイン・ウィンドウの状態を指定します。

例えば，以下のステートメントは，フライト予約アプリケーションを起動し，これをアイコンとして表示します。

```
invoke_application("c:¥¥flight1a.exe", "", "", SW_MINIMIZED);
```

テスト・ステップの定義

テストを実行した後，WinRunner はテストの全体の結果（成功 / 失敗）を［レポート］フォームに出力します。テストの特定の部分が成功したか失敗したかを知りたい場合は，テスト・スクリプトに **tl_step** ステートメントを追加します。

tl_step 関数の構文は，次のとおりです。

tl_step (*step_name, status, description* **);**

step_name パラメータには，テスト・ステップの名前を指定します。*status* パラメータは，ステップが成功（0）したか失敗したか（0 以外の値）を示します。*description* パラメータは，ステップを記述します。

例えば，以下に示すテスト・スクリプトの一部では，WinRunner がワードパッドからテキストを読み取ります。**tl_step** 関数を使って，正しいテキストが読み取れたかどうかを示します。

```
win_get_text(" ドキュメント - ワードパッド ", text, 247, 309, 427, 329);
if (text=="100-Percent Compatible")
    tl_step(" テキスト検証 ", 0, " 正しいテキストがワードパッドに見つかりました ");
else
    tl_step(" テキスト検証 ", 1," 間違ったテキストがノートパッドに見つかりました ");
```

テスト実行が完了すると，テスト結果を WinRunner の［レポート］に表示できます。レポートには，**tl_step** で定義した各ステップの結果（成功 / 失敗）が表示されます。

Quality Center を使ってテストの計画と設計を行っている場合は，自動テスト・スクリプトにおいて，**tl_step** 関数を使ってテスト・ステップを作成します。詳細については，『**Mercury Quality Center ユーザーズ・ガイド**』を参照してください。

2 つのファイルの比較

WinRunner を使用すると，テストの実行中に任意の 2 つのファイルを比較し，**file_compare** 関数を使って 2 つのファイル間の違いを見ることができます。

テストの作成中，テスト・スクリプト内に **file_compare** ステートメントを挿入し，検査したいファイルを指定します。テストを実行する際，WinRunner は両方のファイルを開いて，比較します。ファイルが同一でない場合，または開けない場合は，これをテスト・レポートに記録します。ファイルの不一致の場合は，レポートからこれらのファイルを直接表示し，ファイル内の相違のある行を見ることができます。

例えば，ファイルを新しい名前で保存できるアプリケーションがあるとします(名前を付けて保存)。ファイル比較を使用して，正しいファイルが保存されたかどうか，あるいは特に長いファイル名が切り詰められたかどうかを検査できます。

テストの実行中に 2 つのファイルを比較するには，テスト・スクリプトの適切な場所に **file_compare** ステートメントを配置します。この関数の構文は次のとおりです。

file_compare (*file_1*, *file_2* **[**,*save_file* **])**;

file_1 と *file_2* パラメータは，比較されるファイルの名前を示します。ファイルが現在のテスト・フォルダに含まれていない場合は，完全パスを指定しなければなりません。省略可能な *save_file* パラメータは，3 番目のファイルの名前を保存します。このファイルには，最初のファイルと 2 番目のファイルの違いが含まれています。

次に示す例では，WinRunner は，ワードパッド・アプリケーションの名前を付けて保存機能をテストします。このテストはメモ帳で *win.ini* ファイルを開き，これを *win1.ini* という名前で保存します。次に，**file_compare** 関数を使って，1 つのファイルが他のファイルと同一であるかどうか検査し，test ディレクトリに差異ファイルを格納します。

```
# ワードパッドで win.ini ファイルを開く。
system("write.exe c:¥win95¥win.ini");
set_window("win.ini - ワードパッド ",1);

# win.ini を win1.ini として保存する。
menu_select_item(" ファイル (F); 名前を付けて保存 (A)...");
set_window(" 名前を付けて保存 ");
edit_set(" ファイル名 (N):_0","c:¥Win95¥win1.ini");
set_window(" 名前を付けて保存 ", 10);
button_press(" 保存 (S)");

# win.ini を win1.ini と比較して，両方のファイルを「save」に保存する。
file_compare("c:¥¥win95¥¥win.ini","c:¥¥win95¥¥win1.ini","save");
```

ファイルの比較結果を表示する方法については，第 20 章「テスト結果の分析」を参照してください。

TSL スクリプトの構文チェック

WinRunner がテストを実行すると，**If**，**While**，**Switch**，および **For** ステートメントに不正な構文や要素の欠落など基本的な構文エラーがないか，各スクリプト行を検査します。

例えば，WinRunner は次のうちの 1 つが見つかるとテスト実行を停止して失敗とします。

```
# then のない If ステートメント
if()
report_msg("Bad If Structure");

# 終了条件のない while ステートメント
while(1
{
    report_msg("Bad While Structure");
}

# 閉じ括弧のない for ステートメント
for(i=0;i<5;i++)
{
```

［**構文チェック**］オプションを使用して，テストを実行する前にこれらの構文エラーの種類を検査できます。テストの開始時またはテスト内で選択した行の先頭で構文チェックを実行できます。これにより，テストに構文エラーがないかすばやく検査できるので，テスト全体を実行せずにエラーを確認できます。

テスト全体に対して構文チェックを実行する場合は，［**ツール**］＞［**構文チェック**］＞［**先頭から構文チェック**］を選択します。

テスト内で選択した場所から構文チェックを実行する場合は，左側のマージンをクリックして矢印の位置を設定してから，［**ツール**］＞［**構文チェック**］＞［**矢印から構文チェック**］を選択します。

ヒント：左側のマージンが表示されていない場合は，［**ツール**］＞［**エディタオプション**］を選択して，［**オプション**］タブで［**可視のとじしろ**］を選択します。

構文チェック中に構文エラーが見つかると，メッセージ・ボックスにエラーの説明が表示されます。

第 29 章

テストの呼び出し

WinRunner を使って作成したテストは，別のテストを呼び出したり，別のテストから呼び出すことができます。WinRunner でテストを呼び出すと，呼び出し元のテストから呼び出し先のテストにパラメータ値を渡すことができます。

本章では，以下の項目について説明します。

- テストの呼び出しについて
- call ステートメントの使用法
- 呼び出し元のテストへ戻る方法
- 検索パスの設定
- テスト・パラメータを使った作業
- 呼び出しチェーンの表示

テストの呼び出しについて

テスト・スクリプトに **call** ステートメントを追加して，1 つのテスト・スイートをまるごと含むモジュール化したテスト・ツリー構造を構築できます。モジュール化したテスト・ツリーは，他のテストを呼び出してテスト実行を制御するメイン・テストで構成されます。

WinRunner は，テスト・スクリプト内の **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。このテストには，呼び出し元のテストから入力パラメータ値を渡すことができます。呼び出し先のテストが完了すると，WinRunner は制御を呼び出し元のテストに戻し，テスト実行を継続します。呼び出し先のテストが呼び出し元のテストに出力パラメータ値を返すと，呼び出し元のテストはこれらのパラメータを次のステップで使用できます。呼び出し先のテストは，他のテストを呼び出すこともできます。

テスト・スクリプトに条件判断ステートメントを追加すれば，メイン・テストを使用して，呼び出し先のテストを実行する条件を指定できます。

次に例を示します。

```
rc= call login ("Jonathan", "Mercury");
if (rc == E_OK)
{
    call insert_order();
}
else
{
    tl_step ("Call Login", 1, "Login test failed");
    call open_order ();
}
```

このテストは，まず login テストを呼び出します。login の実行結果が成功の場合，WinRunner は insert_order テストを呼び出します。login テストが失敗した場合，open_ordr テストが実行されます。

呼び出し先のテストは，パラメータ化された値を持つことができます。パラメータには次の 2 種類があります。

- **入力：**呼び出し先のテストは，呼び出し元テストからパラメータを受け取り，これらを使用してテスト内のデータを置換します。
- **出力：**呼び出し先のテストは，呼び出し元のテストにパラメータを返します。呼び出し元のテストでこれらのパラメータを使用できるようになります。

通常，**call** ステートメントは，バッチ・テストの中で使います。バッチ・テストでは，一連のテストを呼び出して，それらを無人で実行できます。バッチ・テストでは，ビットマップの不一致などを示すメッセージなど，通常の実行の際に表示されるメッセージが表示されません。詳細については，第 36 章「バッチ・テストの実行」を参照してください。

注：呼び出し先テストが［**テスト特有の GUI マップ ファイル**］モードで作成されていて，GUI オブジェクトを参照している場合は，［**グローバルな GUI マップ ファイル**］モードでは正しく実行されない場合があります。

テスト実行の際，ブレークごとに［ステップ実行］コマンドの後，ブレークポイント，テストの最後などで［デバッグ ビューア］ウィンドウの［呼び出しチェーン］表示枠で呼び出し先のテストの現在のチェーンと関数を参照できます。また，［呼び出しチェーン］表示枠で［**テストの表示**］ボタンをクリックして，現在実行中のテストを表示することもできます。

［QuickTest への呼び出し］ダイアログ・ボックスを使用することも，QuickTest テストを呼び出す **call_ex** ステートメントを挿入することもできます。詳細については，第 25 章「QuickTest Professional との統合」を参照して下さい。

call ステートメントの使用法

2 種類の call ステートメントを使って，あるテストから別のテストを呼び出すことができます。

- **call** ステートメントは，あるテストから別のテストを呼び出します。
- **call_close** ステートメントは，別のテスト内から別のテストを呼び出して，テストが終了したらテストを閉じます。

call ステートメントの構文は，次のとおりです。

call *test_name* **([** *parameter*$_1$, *parameter*$_2$, ...*parameter*$_n$ **]);**

call_close ステートメントの構文は，次のとおりです。

call_close ***test_name*** **([** ***parameter***$_1$, *parameter*$_2$, ... *parameter*$_n$ **]);**

test_name は，呼び出すテスト名です。*parameters* は，呼び出し先のテストで定義されている入出力パラメータです。パラメータの使用法の詳細については，702 ページ「テスト・パラメータを使用した作業のガイドライン」を参照してください。

呼び出しの対象となるすべてのテストは検索パスに指定されているディレクトリになければなりません。そうでない場合は，テストの完全パス名を二重引用符で囲んで指定しなければなりません。

以下に例を示します。

```
call "w:¥¥tests¥¥my_test" ();
```

呼び出し先のテストの実行中に，実行を中断して現在の呼び出しチェーンを表示することができます。詳細については 708 ページ「呼び出しチェーンの表示」を参照してください。

呼び出し元のテストへ戻る方法

呼び出し先のテストの実行を停止して，call ステートメントに値を返すには，**treturn** および **texit** ステートメントを使用します。

呼び出し先のテストの **treturn** または **texit** ステートメントの値は，呼び出し元のテストの call ステートメント全体の値を返します。出力パラメータを使用して，呼び出し元のテストに追加の値を返すことができます。詳細については，700 ページ「テスト・パラメータを使った作業」. を参照して下さい。

- **treturn** ステートメントは現在のテストを停止し，呼び出し元のテストに制御を戻します。
- **texit** ステートメントは，テストがバッチ・テストから呼び出されているのでなければ，テスト実行を完全に停止します。この場合，制御はメインのバッチ・テストに戻ります。

どちらの関数も呼び出し先のテストの戻り値を提供します。**treturn** または **texit** を使用しなかった場合，あるいは値を指定しなかった場合は，**call** ステートメントの戻り値は 0 となります。

treturn

treturn ステートメントは，呼び出し先のテストの実行を終了し，呼び出し元のテストに制御を戻します。構文は次のとおりです。

treturn [(*expression* **)];**

expression は省略可能です。これは，テストを起動した **call** ステートメントに返される値です。

以下に例を示します。

```
# test a
if (call test_b() == "success")
    report_msg("test_b succeeded");

# test_b
if (win_check_bitmap ("Paintbrush - SQUARES.BMP", "Img_2", 1))
    treturn("success");
else
    treturn("failure");
```

この例では，test_a が test_b を呼び出しています。test_b のビットマップ比較が成功すると，文字列「success」が呼び出し元のテストである test_a に返されます。不一致が見つかると，test_b は文字列「failure」を test_a に返します。

texit

テストを対話形式で実行している場合，**texit** ステートメントはテストの実行を中止します。しかし，テストがバッチ・テストから呼び出されている場合，**texit** は現在のテストの実行だけを終了し，制御は呼び出し元のバッチ・テストに戻ります。構文は次のとおりです。

texit [(*expression* **)];**

expression は省略可能です。これは，テストを呼び出したステートメントに返される値です。

以下に例を示します。

```
# batch_test
return_val = call help_test();
report_msg("help returned the value " return_val);

# help_test
call select_menu(help, index);
msg = get_text(4,30,12,100);
if (msg == "Index help is not yet implemented")
    texit("index failure");
...
```

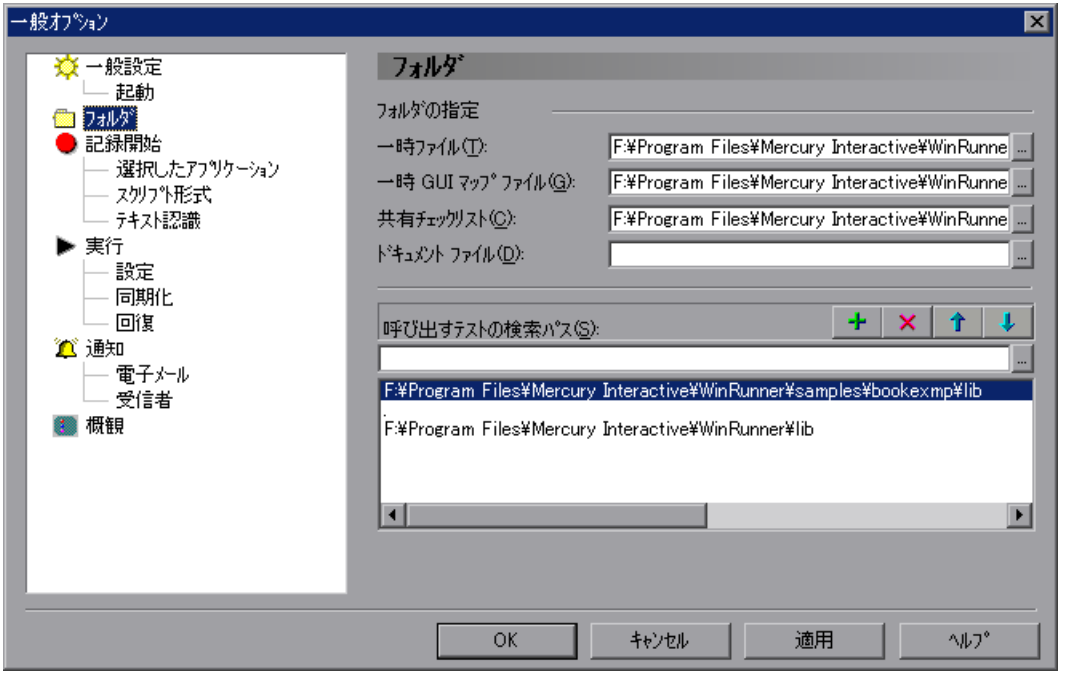
この例では，batch_test が help_test を呼び出しています。help_test では，特定のメッセージが画面に表示されると，実行が停止し，制御がバッチ・テストに戻ります。help_test の戻り値もバッチ・テストに返され，変数 *return_val* に代入されます。**texit** を使わない場合，*return_val* の値は NULL になります。

バッチ・テストの詳細については，第 36 章「バッチ・テストの実行」を参照してください。

検索パスの設定

WinRunner が呼び出し先のテストを検索するディレクトリは，検索パスによって決まります。

検索パスを設定するには，［**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・ボックスが開きます。［**フォルダ**］カテゴリをクリックして［**呼び出すテストの検索パス**］ボックスで検索パスを選択します。WinRunner はボックスに表示されている順番でディレクトリを検索します。定義した検索パスは，以降のテスト・セッションでも有効になります。



➤ 検索パスにフォルダを追加するには，テキスト・ボックスにフォルダ名を入力し，［**追加**］ボタンをクリックします。

➤ ［**項目を上へ移動**］と［**項目を下へ移動**］ボタンを使用して，リストでのフォルダの位置を決めます。

➤ 検索パスを削除するには，リストからフォルダ名を選択し，［**削除**］ボタンをクリックします。

［一般オプション］ダイアログ・ボックスで検索パスを設定する方法の詳細については，第 22 章「グローバル・テスト・オプションの設定」を参照してください。

テスト・スクリプトに **setvar** ステートメントを追加して，検索パスを設定することもできます。**setvar** を使って設定した検索パスは，現在のセッションのすべてのテストにおいてのみ有効です。

以下に例を示します。

```
setvar ("searchpath", "<c:¥¥ui_tests>");
```

このステートメントにより，WinRunner は呼び出し先のテストを ***c:¥¥ui_test*** ディレクトリの中で探します。**setvar** 関数の使用法の詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

注：WinRunner が Quality Center と接続されている場合は，TestDirector データベース内に検索パスを設定することもできます。詳細は，987 ページ「TSL 関数の Quality Center での使用」を参照してください。

テスト・パラメータを使った作業

テストは別のテストを呼び出すと，1 つまたは複数のパラメータを呼び出し先のテストに提供できます。

WinRunner テストは，入力パラメータのデータを受け取り，出力パラメータの値を返します。呼び出し元のテストはこれらの入出力パラメータの値を call ステートメントの引数として提供します。呼び出しチェーンを使用して作業する場合は，パラメータを使用して，データを 1 つのテストから別のテストに渡すことができます。

注：テスト内で定義された入力パラメータを持つテストは，そのテストを呼び出す他のテストを使用せずに実行できます。これは，テストを呼び出しチェーンに含める前のデバッグ時に特に役立ちます。こうしたテストには，実行時に入力パラメータの値を指定します。詳細については，436 ページ「テスト実行時の入力パラメータの値の指定」を参照してください。

パラメータの種類について

テスト・パラメータには，入力パラメータと出力パラメータの 2 種類があります。どちらのパラメータも呼び出すテストで定義でき，テストへの call ステートメントを含むテストに引数として入力することでこれらのパラメータを初期化できます。

テストには入出力パラメータをいくつでも定義できます。テストが受け取ることのできるパラメータは［テストのプロパティ］ダイアログ・ボックスの［パラメータ］タブで定義します。テスト・パラメータの定義の詳細については，503 ページ「テスト・パラメータの管理」を参照してください。

- 入力パラメータとは，呼び出し先のテストの外から値を割り当てられる変数のことです。1 つのテストに，複数の入力パラメータを定義できます。呼び出し元のテストは，これらのパラメータに対して値を提供できます。この値は値そのものでも，値を含む変数でも構いません。呼び出し元のテストによって値が提供されない場合は，定義された値があればそれが使用されます。

  例えば，1 つのテストに 2 つの入力パラメータ *starting_x* と *starting_y* を定義したとします。これらのパラメータは，テストが呼び出されたときに，マウス・ポインタの最初の位置に値を割り当てるために使用するものとします。つまり，呼び出し元のテストが提供する 2 つの値が，マウス・ポインタの x と y 座標を決めることになります。

- 出力パラメータは変数です。その値は（計算またはテストの実行中に値を取得することによって）呼び出し先のテスト内で値が生成され，呼び出し元のテストに返されます。呼び出し元のテストは，call ステートメントの引数として含めることによって，各出力パラメータを初期化します。呼び出し先のテストが実行され，出力パラメータの値を返すと，呼び出し元のテストはこれらの値を，テスト呼び出しで使用した引数を参照することによって使用できます。

例えば，2 つの編集ボックスから情報を読み取るテストがあるとます。このテストに 2 つの出力パラメータを定義します。テスト内のステップは，2 つの編集ボックスから取得されたデータなどのデータをこれらのパラメータに割り当てます。次にこのテストを別のテストから呼び出し，テスト呼び出しの引数として 2 つの変数，*First_Name* および *Last_Name* を含めます。これらの変数は呼び出し先のテストの 2 つの出力パラメータに対応します。呼び出し先のテストの実行後，呼び出し元のテストはスクリプト内で *First_Name* と *Last_Name* を参照でき，呼び出し先のテストによって返された値を使用します。

テスト・パラメータを使用した作業のガイドライン

テスト・パラメータを使用して作業するには，次の点を検討します。

- テストの呼び出しで，出力パラメータの前にすべての入力パラメータを提供しなければなりません。
- 呼び出し先のテストにパラメータが指定されていない場合は，**call** ステートメントに空の括弧を含める必要があります。
- 定義された入力パラメータに値を指定しておらず，呼び出し先のテストで標準の値が定義されている場合は，標準の値が使用されます。標準の値が定義されていない場合は，パラメータは空の値として処理されます。
- 定義された出力パラメータに値を指定しないと，取得されるパラメータの値は呼び出し元のテストに返されません。
- 呼び出し先のテストにテストで実際に定義されているパラメータ数より多いパラメータを渡すと，テストの実行中にエラー・メッセージ「"Warning: Test <path to test>: too many arguments"」が表示されます。
- 出力パラメータは，WinRunner 呼び出しチェーンでの作業時にのみサポートされます。QuickTest Professional または Quality Center を使用している場合は，出力パラメータを含む WinRunner テストを呼び出してはいけません。
- 配列として送られたパラメータは呼び出し先のテストと呼び出し元のテストの両方でスクリプト内の配列として処理されなければなりません。同様に，非配列として送られたパラメータを配列として処理することはできません。
- 定義したパラメータに _IN および _OUT をサフィックス（またはプリフィックス）として追加することをお勧めします。これらのプリフィックスまたはサフィックスを追加することでテストが読みやすくなります。

テスト・パラメータの定義

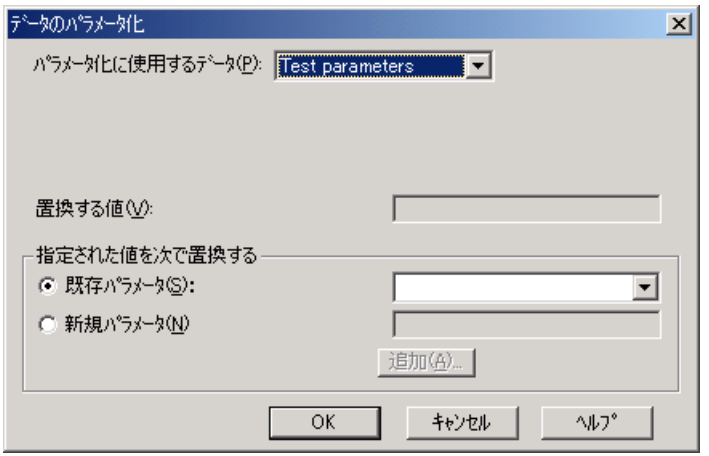
テスト・パラメータは，［テストのプロパティ］ダイアログ・ボックスの［**パラメータ**］タブまたは［データのパラメータ化］ダイアログ・ボックスで定義できます。

- テストのパラメータ・リストの追加，変更，削除を含め，テストのパラメータを管理する場合は，［テストのプロパティ］ダイアログ・ボックスの［パラメータ］タブを使用します。［テストのプロパティ］ダイアログ・ボックスの［**パラメータ**］タブの詳細については，503 ページ「テスト・パラメータの管理」を参照してください。
- テストからの既存のデータを入力パラメータで置換する場合は，［データのパラメータ化］ダイアログ・ボックスを使用します。データを既存の入力パラメータで置換することも，新しい入力パラメータを作成することもできます。

［データのパラメータ化］ダイアログ・ボックスでテストの入力パラメータを定義するには，次の手順を実行します。

1 テスト・スクリプトで，パラメータ化するデータを選択します。

2 ［**テーブル**］＞［**データのパラメータ化**］を選択するか，選択したデータを右クリックして［**データのパラメータ化**］を選択します。［データのパラメータ化］ダイアログ・ボックスが開きます。



3 ［**パラメータ化に使用するデータ**］ボックスで「Test parameters」を選択します。

4 ［**指定された値を次で置換する**］ボックスで［**既存パラメータ**］か［**新規パラメータ**］を選択します。

- ［**既存パラメータ**］を選択する場合は，リストから使用するパラメータを選択します。ここにリストされているパラメータは［テストのプロパティ］ダイアログ・ボックスの［**パラメータ**］タブにリストされているものと同じです。
- ［**新規パラメータ**］を選択する場合は，［**追加**］ボタンをクリックします。［入力パラメータ］ダイアログ・ボックスが開きます。新しいパラメータを追加します。詳細については 503 ページを参照してください。新規パラメータ・フィールドに新規パラメータが表示されます。新規パラメータは，［テストのプロパティ］ダイアログ・ボックスの［**パラメータ**］タブのパラメータ・リストにも表示されます。

5 ［**OK**］をクリックします。

テスト・スクリプトで選択したデータが，作成または選択した入力パラメータで置換されます。

6 パラメータ化する各引数に対してステップ 1 から 5 を繰り返します。

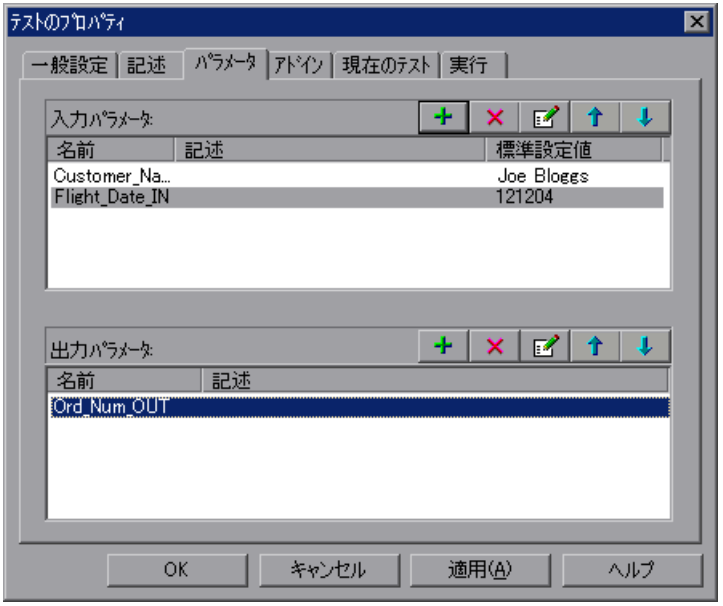
テスト・パラメータの使用 ― 例

次の例では，呼び出し元のテストは顧客が特別割引の対象となるかどうかを確認します。顧客の注文番号を取得するために，指定された顧客とフライト日のデータに基づいて注文番号を返すテストを呼び出しています。

```
# テストの呼び出し
    Cust_Name = “Joe Bloggs”
    Flt_Date = “12122004”
    call “C:¥¥WinRunner Tests¥¥Get_Ord_Num”(Cust_Name, Flt_Date,
    Order_Number);
    if (Order_Number%50==0)
        report_msg(“Prizewinner Discount!”);
    else
        report_msg(“Regular ticket”);

# 呼び出し先テスト Get_Ord_Num
# フライト予約
    win_activate ("Flight Reservation");
    set_window ("Flight Reservation", 1);
    wait(1);
    menu_select_item ("File;Open Order...");
# 注文を開く
    set_window ("Open Order", 1);
```

```
        button_set ("Customer Name", ON);
        edit_set ("Edit", "Customer_Name_IN");
        button_set ("Flight Date", ON);
        obj_type ("MSMaskWndClass","Flight_Date_IN");
        wait(1);
        button_press ("OK");
# 検索結果
        set_window ("Search Results", 1);
        button_press ("OK");
# フライト予約
        set_window ("Flight Reservation", 1);
        win_activate ("Flight Reservation");
        edit_get_text(“Order No:”Ord_Num_OUT);
```

呼び出し先テストでは，Customer_Name_IN，Flight_Date_IN，Order_No_OUT という 3 つのパラメータが定義されます。これらのパラメータ名はパラメータの種類を明確に表しています。



呼び出し元のテストは，顧客名とフライト日を入力パラメータとして提供します。呼び出しの先テストは，対応する注文番号の検索に顧客名とフライト日の入力パラメータを使用します。返される数字は，出力パラメータとして返されます。

呼び出し元のテストはこの出力パラメータをスクリプト内で使用して，値が 50 で割れるものかどうかを確認します。そしてそれを元に顧客が特別割引を受けられるかどうかを決定します。

テスト・パラメータの有効範囲

呼び出し先のテストで定義されるパラメータは引数パラメータといいます。テストのパラメータは，定数，変数，式，配列の要素，あるいは完全な配列のいずれであってもかまいません。

式，変数，または配列要素であるパラメータは，評価された後，呼び出し先のテストに渡されます。つまり，呼び出し先のテストには値のコピーが渡されます。このコピーはローカルです。したがって，呼び出し先のテストでその値が変更されたとしても，呼び出し元のテストの元の値には何の影響もありません。以下に例を示します。

```
# test_1（呼び出し元テスト）
i = 5;
call test_2(i);
pause(i); # 番号「5」を持つメッセージ・ボックスを開きます。

# test_2（呼び出し先テスト test_1）フォーマル・パラメータ x の使用
x = 8;
pause(x); # 番号「8」を持つメッセージ・ボックスを開きます。
```

呼び出し元のテスト（test_1）では，変数 i に値 5 を代入しています。この値が，フォーマル・パラメータ x の値として呼び出し先のテスト（test_2）に渡されます。そして，test_2 で x に新しい値（8）を代入された場合，test_1 の i の値には何の影響も与えません。

完全な配列は参照渡しされます。つまり完全な配列は，配列要素，変数，あるいは式とは異なり，コピーが作成されません。呼び出し先のテストで配列になんらかの変更を加えると，呼び出し元のテストの対応する配列も影響を受けます。以下に例を示します。

```
# test_q
a[1] = 17;
call test_r(a);
pause(a[1]); # 番号「104」を持つメッセージ・ボックスを開きます。

# test_r パラメータ x の使用
x[1] = 104;
```

呼び出し元のテスト（test_q）では，配列 *a* の要素 1 に値 17 を代入しています。その後，配列 *a* は，フォーマル・パラメータ *x* を含む呼び出し先テスト（test_r）に渡されます。test_r では，配列 *x* の要素 1 に値 104 を代入しています。

先述の例と異なり，このパラメータは配列であるため，呼び出し先のテストにおけるこの変更は，呼び出し元のパラメータにも影響を与えます。

呼び出し先のテストで宣言されていない変数のうち，フォーマル・パラメータ・リストに入っていない変数はすべてグローバル変数です。したがって，これらの変数には，呼び出し先のテストや呼び出し元のテストからアクセスすることも，変更することもできます。テストにパラメータ・リストが定義されており，そのテストが別のテストから呼び出されず，直接実行される場合，リストのパラメータは，そのテストにおいてはグローバル変数として機能します。変数の詳細については，WinRunner の「**TSL リファレンス**」を参照してください。

次に示すのは，グローバル変数の使い方について示すテストの一部です。test_a は呼び出されず，直接実行されることに注意してください。

```
# test_a はパラメータ k を含みます。
# アンパサンド（&）はビット単位の AND 演算子です。これは連結を表します。
i = 1;
j = 2;
k = 3;
call test_b(i);
pause(j & k & l); # 番号「256」を持つメッセージ・ボックスを開きます。
# test_b は入力パラメータ j を含みます。
# アンパサンド（&）はビット単位の AND 演算子です。これは連結を表します。
j = 4;
k = 5;
l = 6;
pause(j & k & l); # 番号「456」を持つメッセージ・ボックスを開きます。
```

呼び出しチェーンの表示

テスト実行の際，ブレークごとに［ステップ実行］コマンドの後，ブレークポイント，テストの最後などで［デバッグ ビューア］ウィンドウの［呼び出しチェーン］表示枠で呼び出し先のテストの現在のチェーンと関数を参照できます。

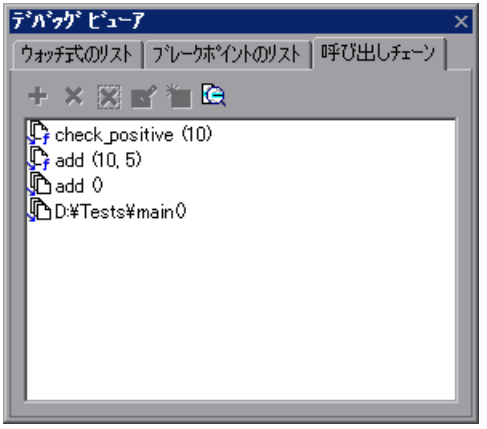
現在の呼び出しチェーンを表示するには，次の手順を実行します。

1 ［デバッグ ビューア］ウィンドウが現在表示されていない場合，またはウィンドウ内で［呼び出しチェーン］表示枠が開いていない場合は［**デバッグ**］＞［**呼び出しチェーン**］を選択して，これを表示します。呼び出しチェーン］表示枠が開いているけれども別の表示枠が現在表示されている場合は，［**呼び出しチェーン**］タブをクリックしてこれを表示します。

2 呼び出し先テストで，呼び出しチェーンを表示する場所にブレークポイントがあることを確認します。または，［ステップ］コマンドを使用して，テストの実行を制御します。

［ステップ］コマンドの詳細については，第 38 章「テスト実行の制御」を参照してください。

3 テストが一時停止すると，［デバッグ ビューア］の［呼び出しチェーン］表示枠に呼び出しチェーンが表示されます。



ヒント：［デバッグ ビューア］ウィンドウは，WinRunner ウィンドウに固定されたウィンドウとして表示することも，画面上の希望の場所にドラッグできるフローティング・ウィンドウとして表示することもできます。

4 呼び出しチェーンでテストのスクリプトを表示するには，テストまたは関数をダブルクリックするか，リスト内でテストまたは関数を選択して，［テストの表示］をクリックします。選択したテストまたは関数が WinRunner のアクティブ・ウィンドウに表示されます。

第 30 章

ユーザ定義関数の作成

ユーザは独自のユーザ定義 TSL 関数を定義することで，WinRunner のテスト機能を拡張できます。ユーザ定義関数は，テストやコンパイル済みモジュールの中で使用できます。

本章では，以下の項目について説明します。

- ユーザ定義関数について
- 関数の構文
- return ステートメントと exit ステートメント
- 変数，定数，および配列の宣言
- ユーザ定義関数の例

ユーザ定義関数について

TSL では，組み込み関数が用意されているほか，ユーザ独自の関数を設計し，実装することができます。以下が可能です。

作成した関数は，それらの関数がが含まれるテストで使用することも，他のテストで使用できるようコンパイル済みモジュールに保存することもできます。コンパイル済みモジュールの詳細については，723 ページ「コンパイル済みモジュールの内容について」を参照してください。

ユーザ定義関数は，テスト・スクリプトの中で同じ操作を繰り返し行いたい場合などに便利です。同じコードを繰り返す代わりに，その処理を行う関数を 1 つ書くだけで済みます。その結果，テスト・スクリプトをモジュール化でき，読みやすくなる上，デバッグとメンテナンスが容易になります。

例えば，GUI マップ・ファイルを読み込み，航空券予約アプリケーションを起動してシステムにログインする，あるいはアプリケーションがすでに開いている場合は，メイン・ウィンドウをリセットする open_flight という関数を作成することもできます。

関数は，テスト・スクリプトのどこからでも呼び出せます。すでにコンパイル済みであるため，実行速度もより高速です。例えば，いくつかのファイルを開き，その内容を調べるテストを作成したとします。ファイルを開く手順を複数回記録あるいはプログラムする代わりに，関数を作成してファイルを開きたいときにはその関数を呼び出すようにできます。

関数の構文

ユーザ定義関数の構造は，次のとおりです。

[*class*] **function** *name* ([*mode*] *parameter*...)
{
declarations;
statements;
}

関数クラス

関数のクラス（class）は，*static* または *public* のいずれかです。

static 関数は，関数が定義されたテストあるいはモジュールの中でしか利用できません。

public 関数を一度でも再生すると，その関数を含んでいるテストが開いている限り，すべてのテストでその関数が利用できます。これは，呼び出し先のテストから関数を利用できるようにしたい場合には便利です。しかし，多くのテストで利用するような関数を作成したい場合は，コンパイル済みモジュールに入れることをお奨めします。コンパイル済みモジュールを一度ロードすると，その関数はそのコンパイル済みモジュールがアンロードされるまですべてのテストに利用できます。コンパイル済みモジュールのロードとアンロードの詳細については，728 ページ「関数またはコンパイル済みモジュールのロード」および 729 ページ「関数またはコンパイル済みモジュールのアンロード」を参照してください。

「**外部関数**」は，宣言はローカル・テストまたはコンパイル済みモジュールにあり，実装コードが外部ソースにある点を除き，public 関数のように振る舞います。最も一般的な例は，DLL で定義されている関数です。関数が定義されている DLL をロードしてから，テストまたはコンパイル済みモジュールの関数を宣言しロードできます。一度ロードされると，テストはその関数を呼び出すことができます。詳細については，第 33 章「外部ライブラリからの関数の呼び出し」を参照してください。

クラスを明示的に宣言しないと，関数のクラスは public クラスとなります。

関数のパラメータ

パラメータは明示的に宣言する必要はありません。*in*，*out*，あるいは *inout* のいずれかのモードとして定義できます。配列以外のパラメータの標準のモードは *in* です。配列パラメータの標準のモードは *inout* です。パラメータの種類のそれぞれの意味は次のとおりです。

in：関数の外から与えられる値が代入されるパラメータです。

out : 関数の中で値が代入されるパラメータです。

inout：関数の外または中で値を代入できるパラメータです。

out または inout と指定されたパラメータは，式ではなく，変数名でなければなりません。*out* または *inout* パラメータを含む関数を呼び出した場合，それらのパラメータに対応する引数は，式ではなく，変数でなければなりません。例えば，以下の構文のユーザ定義関数を考えてみましょう。

```
function get_date (out todays_date) { ... }
```

正しい使い方は，次のとおりです。

```
get_date (todays_date);
```

一方，次の関数呼び出しは，式が含まれているため不正です。

```
get_date (date[i]); または get_date ("Today's date is"& todays_date);
```

「**配列パラメータ**」は大括弧によって表現されます。例えば，以下のユーザ定義関数のパラメータ・リストは，変数 *a* が配列であることを示しています。

```
function my_func (a[], b, c){ ... }
```

配列パラメータのモードは，out または inout のいずれかです。クラスを指定しないと，標準の inout モードであるとみなされます。

注：ユーザ定義関数には 15 までのパラメータを定義できます。

---

return ステートメントと exit ステートメント

return ステートメントは，関数の中だけで使用するステートメントです。構文は次のとおりです。

return ([*expression***]);**

ステートメントは，呼び出し元の関数またはテストに制御を戻します。その際，expression に指定された式を評価し，その値を呼び出し元の関数またはテストに返します。**return** ステートメントに対して式を割り当てなかった場合，空の文字列が返されます。

texit ステートメントは，関数またはテスト実行を停止するためにしようできます。構文は次のとおりです。

texit ([*expression* **]);**

テストが対話的に実行されると，texit はテスト実行を完全に中止します。テストがバッチ・モードで実行されると，ステートメントは現在の主要テストの実行のみを終了し，コントロールが呼び出し元のバッチ・テストに返されます。また，texit 関数は，評価された式の値を呼び出し元の関数またはテストに返します。

注：QuickTest は呼び出し先の関数では **texit** ステートメントをサポートしません。QuickTest が **texit** ステートメントを含む WinRunner 関数を呼び出すと，関数呼び出しは失敗します。

変数，定数，および配列の宣言

通常，TSL では宣言は省略可能です。しかし，関数では変数，定数，および配列をすべて宣言しなければなりません。宣言は，関数自身の中，あるいは関数を含んでいるテスト・スクリプトまたはコンパイル済みモジュールの任意の箇所に記述できます。宣言の詳細については，「**TSL リファレンス**」を参照してください。

変数の宣言

変数宣言の構文は次のとおりです。

class variable [**=** *init_expression*]**;**

宣言した変数に代入される *init_expression* は，有効な式であればどんな式でもかまいません。*init_expression* を設定しないと，変数には空の文字列が代入されます。*class* は，変数の有効範囲を定義します。以下のいずれかを指定できます。

auto：auto 変数は関数の中でしか宣言できず，その有効範囲は関数の内部に限られています。関数の実行が終了すると消滅します。この変数は，関数が呼び出されるたびに新しくコピーが作られます。

static：static 変数の有効範囲は，その変数が宣言されている関数，テスト，またはモジュール内に限られます。この変数は，テストが［停止］コマンドによって終了するまで値を保持します。この変数は関数の定義が実行されるたびに初期化されます。

注：コンパイル済みモジュールでは，**static** 変数はコンパイル済みモジュールがコンパイルされるたびに初期化されます。

---

public：public 変数は，テストまたはモジュールの中でのみ宣言できます。この変数は，どの関数，テストおよびコンパイル済みモジュールからも使用できます。

extern：extern 宣言は，現在のテストまたはモジュールの外で宣言された public 変数への参照を示します。

関数で使う変数はすべて関数内，または関数が含まれているテストまたはモジュールの中で宣言しなければなりません。関数が含まれているテストまたはモジュールの外で宣言されている public 変数を使いたい場合は，それを extern として宣言しなければなりません。

extern 宣言は，テストまたはモジュール内で，関数のコードの前になければなりません。extern 宣言では変数を初期化できません。

例えば，Test 1 で以下の変数を宣言するとします。

```
public window_color=green;
```

そして，別のテストまたはモジュールの Test 2 で window_color 変数を使うユーザ定義関数を作成したとします。その関数を含んでいるテストまたはモジュールでは，window_color を次のように宣言しなければなりません。

```
extern window_color;
```

auto 変数を除くすべての変数は，［停止］コマンドが実行されるまで存在し続けます。

注：コンパイル済みモジュールでは，**auto** および **public** 変数を除くすべての変数は，［停止］コマンドが実行されるまで存在し続けます。（**auto** 変数は，関数を実行している間だけ存在します。**public** 変数は，WinRunner を終了するまで存在し続けます。）

次の表に，それぞれの種類の変数の有効範囲，存続期間，利用可能範囲（宣言可能な箇所）を示します。

宣言	有効範囲	存続期間	利用可能範囲
auto	ローカル	関数の終了まで	関数
static	ローカル	中止まで	関数，テスト，モジュール
public	グローバル	中止まで	テストまたはモジュール
extern	グローバル	中止まで	関数，テスト，モジュール

注：コンパイル済みモジュールでは，［停止］コマンドは static および public 変数を初期化します。詳細については，第 31 章「テストでのユーザ定義関数の利用」を参照してください。

定数の宣言

const 指定子は，宣言した変数の値を変更できないことを示します。この宣言の構文は次のとおりです。

[*class*] **const** *name* [**=** *expression*]**;**

定数のクラス（*class*）は public か static のいずれかです。クラスが明示的に宣言されなかった場合，定数は標準のクラスである public となります。定数は一度定義すると，WinRunner を終了するまで存在し続けます。

例えば，次の宣言を使って定数 TMP_DIR を定義します。

```
const TMP_DIR = "/tmp";
```

この場合，代入された /tmp という値は変更できません（この値を変更するには，TMP_DIR の定数宣言を再度明示的に行う必要があります）。

配列の宣言

配列のクラスと初期化式を定義するには次の構文を使用します。TSL では，配列のサイズを定義する必要はありません。

class array_name [ ] [=*init_expression*]

配列のクラス（class）には，変数宣言で使用する任意のクラス（auto，static，public，extern）を指定できます。

配列は，C 言語と同様の構文を使用して初期化できます。以下に例を示します。

```
public hosts [ ] = {"lithium", "silver", "bronze"};
```

このステートメントは，次の要素を持つ配列を作成します。

```
hosts[0]="lithium"
```

```
hosts[1]="silver"
hosts[2]="bronze"
```

C 言語の場合と同様，*auto* クラスの配列は初期化できません。

また，配列の添字として文字列を使用して各配列要素を初期化することも可能です。文字列の添字は，TSL の有効な式であれば何であってもかまいません。値はコンパイル時に評価されます。

次に例を示します。

```
static gui_item [ ]={
    "class"="push_button",
    "label"="OK",
    "X_class"="XmPushButtonGadget",
    "X"=10,
    "Y"=60
};
```

これにより，次の配列要素が作成されます。

```
gui_item ["class"]="push_button"
gui_item ["label"]="OK"
gui_item ["X_class"]="XmPushButtonGadget"
gui_item ["X"]=10
gui_item ["Y"]=60
```

配列は，関数が最初に実行されたときに一度だけ初期化されます。配列の初期化値を編集しても，新しい値は以降のテスト実行には反映されません。配列を新しい値で初期化するには，［停止］コマンドでテスト実行を中止するか，配列の新しい要素を明示的に定義しなければなりません。以下に例を示します。

通常の初期化	**明示的な定義**

`public number_list[] = {1,2,3};`	`number_list[0] = 1;`
	`number_list[1] = 2;`
	`number_list[2] = 3;`

ステートメント

TSL テスト・スクリプトで使用できる有効なステートメントであれば，関数の中で使用できます。ただし，**treturn** ステートメントだけは使用できません。

ユーザ定義関数の例

以下のユーザ定義関数は，指定されたテキスト・ファイルをエディタで開きます。必要な GUI マップ・ファイルが読み込まれていることが前提となっています。この関数は，操作が完了した後に，ファイルの名前とウィンドウのタイトル・バーに表示されているラベルを比較することで，実際にファイルが開かれたかどうかを調べます。

```
function open_file (file)
{
    auto lbl;
    set_window ("Editor");

    # ［Open］フォームを開く。
    menu_select_item ("File;Open...");

    # 適切なフィールドにファイル名を挿入し、［OK］をクリックして確認する
    set_window ("Open");
    edit_set(“Open Edit”, file);
    button_press ("OK");

    # ウィンドウ・タイトルを読みこむ。
    win_get_info("Editor","label",lbl);

    # ラベルとファイル名を比較する。
    if ( file != lbl)
        return 1;
    else
        return 0;
}
rc=open_file("c:¥¥dash¥¥readme.tx");
pause(rc);
```

第 31 章

テストでのユーザ定義関数の利用

ユーザ定義関数は，それらを定義したテストの中から，または当該関数を読み込んだ他のテストから，あるいは読み込んだコンパイル済みモジュールから，呼び出すことができます。

本章では，次の項目について説明します。

- ユーザ定義関数の利用について
- コンパイル済みモジュールの内容について
- 関数ビューアの使用方法
- テスト内で定義された関数の利用
- コンパイル済みモジュール内で定義された関数の利用

ユーザ定義関数の利用について

ユーザ定義関数は 3 つのうちいずれかの方法で利用できます。

- 関数を定義したテストの中から，関数を呼び出すことができます。関数呼び出しには入力引数と出力引数を含めることができます。

例えば，次のような簡単な関数を定義できます。

```
public function Add6(x)
{
      return(x+6);
}
```

この関数を次のようなコマンドで呼び出すことができます。

```
y=Add6(Ord_Num);
```

- すでに実行した任意のテストで定義されている，非静的関数を呼び出すことができます。テストを実行すると，その中に含まれているすべてのパブリック関数がロードされ，［**停止**］ボタンをクリックするまでは，他のどのテストからも使用できるようになります。［**停止**］ボタンをクリックすると，コンパイル済みモジュールではないすべてのテストから読み込まれていた関数が，アンロードされます。詳細については，732 ページ「テスト内で定義された関数の利用」を参照してください。
- 読み込まれているコンパイル済みモジュールから，非静的関数を呼び出すことができます。コンパイル済みモジュールは，使用頻度の高い関数をライブラリとして収めた特殊なタイプのテストです。コンパイル済みモジュールは，関数ビューアを使用して，またはテスト・スクリプトから，読み込むことができます。関数ビューアの詳細については，725 ページ「関数ビューアの使用方法」を参照してください。テスト・スクリプトからのコンパイル済みモジュールのロードに関する詳細については，733 ページ「コンパイル済みモジュール内で定義された関数の利用」を参照してください。

注：上記のうち最後の 2 つの方法を使用して呼び出すことができるのは，パブリック関数と外部関数のみです。詳細については，712 ページ「関数クラス」を参照してください。

以降，本章では，コンパイル済みモジュールの内容，関数ビューア，および上記のうち最後の 2 つの方法について説明します。

コンパイル済みモジュールの内容について

コンパイル済みモジュールは，TSL で作成する通常のテストと同じように，開き，編集し，保存することができます。［テストのプロパティ］ダイアログ・ボックスの［**一般設定**］タブの［**テストの種類**］ボックスで，「**コンパイルされたモジュール**」を選択することによって，テストがコンパイル済みモジュールであることを指定します。詳細については，724 ページ「コンパイル済みモジュールの作成」を参照してください。

コンパイル済みモジュールの内容は通常のテストの内容とは異なります。例えば，コンパイル済みモジュールにはチェックポイントやマウスの追跡などのアナログ入力を含めることはできません。コンパイル済みモジュールの用途はテストの実行ではなく，頻繁に使用するユーザ定義関数を格納しておき，他のテストから手早く便利にアクセスできるようにすることです。

通常のテストとは異なり，コンパイル済みモジュール内のデータ・オブジェクト（変数，定数，配列）はすべて，使用前に宣言しておく必要があります。コンパイル済みモジュールの構造は，次の要素を含めることができるという点で C のプログラム・ファイルに似ています。

- 関数の定義，変数の宣言，定数の宣言，配列の宣言。詳細については，第 30 章「ユーザ定義関数の作成」を参照してください。
- 外部関数のプロトタイプ。詳細については，第 33 章「外部ライブラリからの関数の呼び出し」を参照してください。
- 他のモジュールを対象とする **load** ステートメント。詳細については，735 ページ「コンパイル済みモジュールのロードとアンロード」を参照してください。

ユーザ定義関数がコンパイル済みモジュール内に出現するときは，次の点に注意してください。

- パブリック関数または外部関数はすべてのモジュールとテストで使用できますが，静的関数はその関数を定義しているモジュールでしか使用できません。
- 読み込まれたモジュールは，テストの実行が中止されてもメモリ上に残ります。ただし，モジュールの中で定義されている変数は，静的，パブリック，または外部にかかわらず，すべて初期化されます。

注：読み込まれたコンパイル済みモジュール内の関数に変更を加えた場合は，変更を有効にするために，コンパイル済みモジュールをいったんアンロードしてから再度読み込む必要があります。

詳細については，741 ページ「コンパイル済みモジュールの例」を参照してください。

コンパイル済みモジュールの作成

コンパイル済みモジュールの作成は通常のテスト・スクリプトの作成と同様に行います。

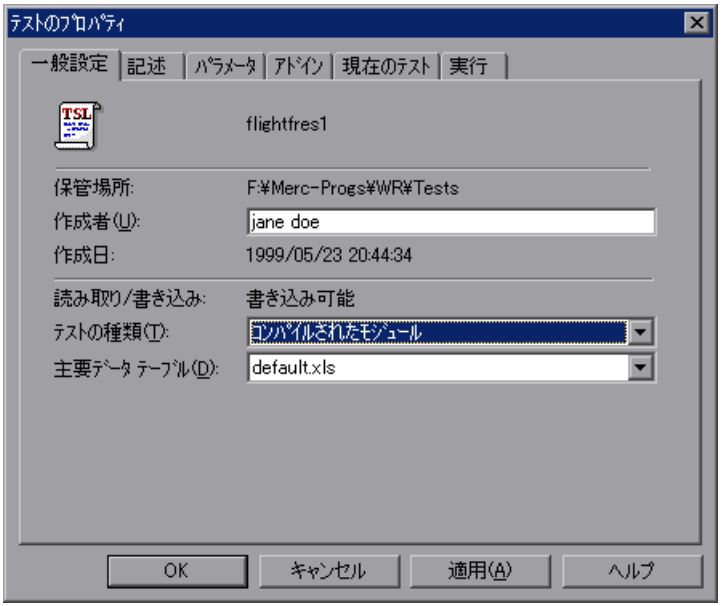
コンパイル済みモジュールを作成するには，次の手順を実行します。

1 ［**ファイル**］＞［**開く**］を選択して，新しいテストを開きます。

2 テスト内にユーザ定義関数を記述します。

3 ［**ファイル**］＞［**テストのプロパティ**］を選択し，［**一般設定**］タブをクリックします。

4 ［**テストの種類**］リストで「**コンパイルされたモジュール**」を選択し，［**OK**］をクリックします。



5 ［**ファイル**］＞［**上書き保存**］を選択します。

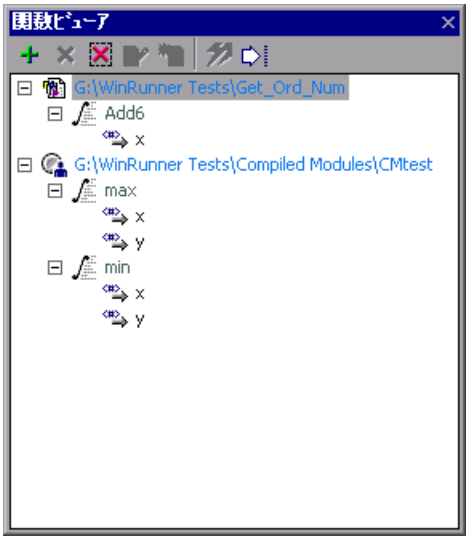
モジュール内の関数を呼び出す可能性のあるすべてのテストから簡単に利用できる場所にモジュールを保存します。モジュールのロード時には，定義されている検索パスに基づいてモジュールが探されます。検索パスの定義の詳細については，699 ページ「検索パスの設定」を参照してください。

6 テスト・スクリプトの中で **load** 関数を使用するか，または関数ビューアの［**ロード**］ボタンを使用して，モジュールをコンパイルします。詳細については，735 ページ「コンパイル済みモジュールのロードとアンロード」を参照してください。

関数ビューアの使用方法

関数ビューアを使用して，コンパイル済みモジュールのロードとアンロードが行えるほか，読み込まれたコンパイル済みモジュールおよびテストの関数のコピー，貼り付け，および実行もできます。また，読み込まれる関数を含んだロード済みのコンパイル済みモジュールおよびテストを開くこともできます。

関数ビューアはいつでも開いたり閉じたりでき，WinRunner ウィンドウに固定することもできるウィンドウです。



関数ビューアは，ツールバーと，関数ツリーを表示する表示枠で構成されています。

関数ツリーには 3 つのレベルがあります。最上位のレベルには，読み込まれたコンパイル済みモジュールのほか，読み込まれた関数を含んでいる，開かれているテストが表示されます。

テスト，または［**実行**］ツールバー・ボタンを使用して読み込まれたコンパイル済みモジュールは，テスト・アイコンで示されます。

テスト内の **load** 関数または関数ビューアの［**ロード**］ボタンによって読み込まれたコンパイル済みモジュールは，コンパイル済みモジュール・アイコンで示されます。

非静的関数は非静的関数アイコンで示され，関数ツリー内でコンパイル済みモジュールおよびテストよりも 1 つ下のレベルに表示されます。

静的関数は静的関数アイコンで示され，関数ツリー内でコンパイル済みモジュールおよびテストよりも 1 つ下のレベルに表示されます。

入力パラメータが存在する場合は入力パラメータ・アイコンによって示され，ビューア内では表示される関数よりも 1 つ下のレベルに表示されます。

出力パラメータおよび inout パラメータが存在する場合は出力パラメータ・アイコンによって示され，ビューア内では表示される関数よりも 1 つ下のレベルに表示されます。

項目の横にある［**展開**］ボタンをクリックするか，またはキーボードのテンキーにあるプラス・キー（+）を押すと，項目を展開できます。

キーボードのテンキーにあるアスタリスク・キー（*）を押すと，項目とその下のすべてのレベルを展開できます。

項目の横にある［**折りたたみ**］ボタンをクリックするか，またはキーボードのテンキーにあるマイナス・キー（-）を押すと，項目とその下のすべてのレベルを折りたたむことができます。

関数ビューアのツールバーには以下のオプションがあります。

- ［**コンパイル済みモジュールのロード**］：コンパイル済みモジュールを読み込むことができます。詳細については，728 ページ「関数またはコンパイル済みモジュールのロード」を参照してください。

- ［**ロード解除**］：現在選択されているコンパイル済みモジュールをアンロードします。

- ［**全モジュールのロード解除**］：すべてのコンパイル済みモジュールをアンロードします。このボタンは，テストからロードした関数に対しては何も実行しません。

- ［**コピー**］：選択した関数プロトタイプをクリップボードにコピーします。詳細については，729 ページ「関数プロトタイプのコピーと貼り付け」を参照してください。

- ［**貼り付け**］：選択した関数プロトタイプをコピーし，テスト内の現在のカーソル位置に貼り付けます。詳細については，729 ページ「関数プロトタイプのコピーと貼り付け」を参照してください。

- ［**実行**］：選択した関数を実行します。詳細については，730 ページ「関数ビューアからの関数の実行」を参照してください。

- ［**移動先番号を指定**］：選択したコンパイル済みモジュール，テスト，または関数をテスト・ウィンドウで開きます。詳細については，731 ページ「読み込まれたコンパイル済みモジュール，テスト，または関数を表示や編集のために開く方法」を参照してください。

関数ビューアの表示

関数ビューアを表示するには，［**ツール**］＞［**関数ビューア**］を選択します。関数ビューアはテスト・ウィンドウの上下左右のどの端にもドッキングできます。関数ビューアを閉じるには，［**閉じる**］ツールバー・ボタンをクリックするか，または［**ツール**］＞［**関数ビューア**］をもう一度選択します。

関数またはコンパイル済みモジュールのロード

呼び出し元のテストで定義されていない関数を呼び出す必要がある場合は，その関数を読み込む必要があります。

関数はテストまたはコンパイル済みモジュールの中で定義できます。

テスト内で定義されている関数は，テストを実行することによってロードします。テストを実行すると，そのテスト内で定義されているすべての関数が読み込まれ，［**停止**］ボタンをクリックするまで使用し続けることができます。関数はテストが閉じた場合でも引き続き読み込まれたままになっています。［**停止**］ボタンをクリックすると，コンパイル済みモジュールではないすべてのテストから読み込まれていたすべての関数が，アンロードされます。詳細については，732 ページ「テスト内で定義された関数の利用」を参照してください。

注：すでに読み込まれているコンパイル済みモジュールに含まれている関数と同じ名前を持つ関数を含んだコンパイル済みモジュールを読み込むと，先に読み込まれていたコンパイル済みモジュールの関数がアンロードされ，後から読み込まれるコンパイル済みモジュールの関数が読み込まれます。標準の TSL 関数と同じ名前を持つ関数を含んだコンパイル済みモジュールを読み込んだ場合は，元の関数がオーバーライドされます。

コンパイル済みモジュール内で定義されている関数はすべて，次の 2 つのいずれかの方法でロードします。

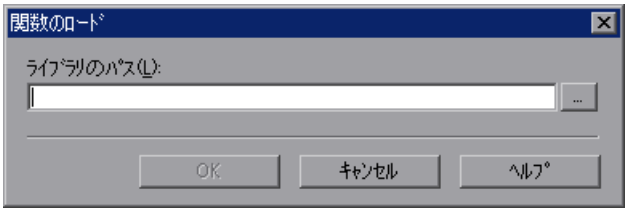
- テスト・スクリプトから。テスト・スクリプトで **load** TSL 関数を使用してコンパイル済みモジュールを読み込むことができます。詳細については，737 ページ「TSL 関数を使用したコンパイル済みモジュールのロードとアンロード」を参照してください。
- 関数ビューアから。関数ビューアの［**ロード**］ツールバー・ボタンを使用してコンパイル済みモジュールを読み込むことができます。コンパイル済みモ

ジュールのロードが終われば，その中で定義されている任意の非静的関数を呼び出すことができるようになります。

関数ビューアからコンパイル済みモジュールを読み込むには，次の手順を実行します。

1 関数ビューアが表示されていることを確認します。詳細については，728 ページ「関数ビューアの表示」を参照してください。

2 ［**ロード**］ボタンをクリックします。［関数のロード］ダイアログ・ボックスが開きます。



3 参照ボタンを使用して，読み込むコンパイル済みモジュールを探します。または，［**ライブラリ パス**］エディット・ボックスにパスを手入力します。

4 ［**OK**］をクリックします。コンパイル済みモジュールが関数ビューア・ツリーに表示され，そのすべての関数が読み込まれます。

関数またはコンパイル済みモジュールのアンロード

コンパイル済みモジュールをアンロードするには，コンパイル済みモジュールを選択して［**ロード解除**］ボタンをクリックします。

読み込まれているすべてのコンパイル済みモジュールをアンロードするには，［**全モジュールのロード解除**］ボタンをクリックします。

［**ロード解除**］ボタンおよび［**全モジュールのロード解除**］ボタンを使用して，テスト内の関数をアンロードすることはできません。これらの関数のロードを解除するには，［**停止**］ボタンをクリックします。

関数プロトタイプのコピーと貼り付け

関数プロトタイプをクリップボードにコピーして，他のアプリケーションに貼り付けることができます。

関数プロトタイプをクリップボードにコピーするには，関数プロトタイプを選択して［**コピー**］ボタンをクリックします。

関数プロトタイプをテスト画面にコピーして貼り付けるには，関数プロトタイプを選択し，テスト画面上の貼り付け位置にカーソルを置いて，［**貼り付け**］ボタンをクリックします。この場合は先にコピーする必要はありません。関数プロトタイプをドラッグしてドロップすることもできます。

関数ビューアからの関数の実行

関数ビューアから直接，非静的関数を実行することができます。この方法は関数をテストする場合に便利です。例えば，多数の関数を収めたコンパイル済みモジュールを作成していて，その中の関数を 1 つだけテストする必要がある場合に，その関数を読み込んで呼び出すテストを記述しなくても，関数を直接実行できます。

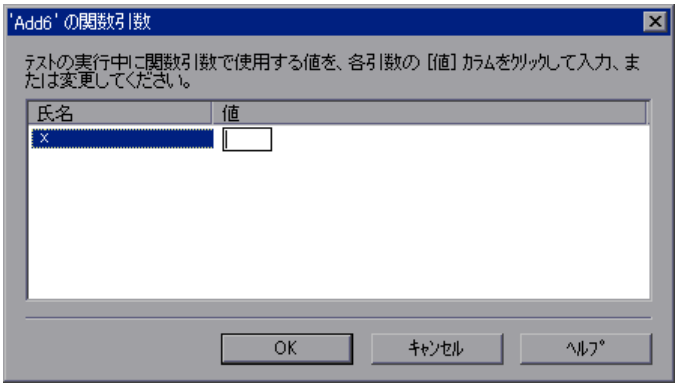
注：静的関数を関数ビューアから実行することはできません。静的関数を実行しようとした場合は，エラー・メッセージが表示されます。

関数ビューアから関数を実行するには，次の手順を実行します。

1 実行する関数を関数ビューア内で選択します。

2 ［**実行**］ツールバー・ボタンをクリックします。入力パラメータを必要としない関数であれば，関数がそのまま実行されます。

3 パラメータを必要とする関数の場合は，［関数引数］ダイアログ・ボックスが開きます。



4 各パラメータの値を入力します。それには，［**値**］カラムの中で，各引数に対応する行をクリックして値を入力します。［**OK**］をクリックします。関数が実行されます。

注：出力パラメータまたは inout パラメータを持つ関数を呼び出すとき，そのパラメータに対応する引数は，式ではなく変数である必要があります。

入力パラメータを持つ関数を呼び出すとき，そのパラメータに対応する引数は変数であってはならず，文字列または数値である必要があります。数字以外の文字はすべて文字列として扱われます。

読み込まれたコンパイル済みモジュール，テスト，または関数を表示や編集のために開く方法

［**移動**］ツールバー・ボタンを使用して，関数ビューア内に表示されている，読み込まれたコンパイル済みモジュール，テスト，または関数を開くことができます。そして，内容を表示および編集することができます。

コンパイル済みモジュールまたはテストをテスト・ウィンドウで開くには，次の手順を実行します。

1 関数ビューア内で，コンパイル済みモジュール，テスト，または関数を選択します。

2［**移動**］ボタンをクリックします。あるいは，コンパイル済みモジュール，テスト，または関数をダブルクリックします。コンパイル済みモジュールまたはテストが，テスト・ウィンドウ内の個別のタブに開きます。関数を開いた場合は，その関数が定義されているテスト全体が開き，関数の最初の行が実行マーカによって示されます。

テスト内で定義された関数の利用

関数は任意のテスト・スクリプトの中で定義できます。テストを実行すると，テストの中で定義されている個々の関数が WinRunner によって読み込まれます。このとき，テストの中で呼び出されていない関数も含め，テストの中で定義されているすべての関数が読み込まれます。読み込まれた関数は関数ビューアに表示されます。

注 : テストの実行時にエラーのために関数の読み取りができなかった場合，その関数は読み込まれず，関数ビューアに表示されません。

テストから読み込まれた関数は，テストを実行した時点から使用できるようになり，以降，WinRunner セッションが終了するまで，または［**停止**］ボタンをクリックするまで，使用できます。

［**停止**］ボタンはいつでもクリックできます。クリックすると，すべてのテストとそれらの関数が関数ビューアに表示されなくなり，他のテストからそれらの関数を呼び出すことができなくなります。

関数が関数ビューアに表示されている間は，関数の呼び出し，関数の実行，関数プロトタイプのコピーと貼り付け，および関数を開く操作が行えます。関数プロトタイプのコピーと貼り付けの詳細については，729 ページ「関数プロトタイプのコピーと貼り付け」を参照してください。読み込まれた関数の実行の詳細については，730 ページ「関数ビューアからの関数の実行」を参照してください。関数を開く方法の詳細については，731 ページ「読み込まれたコンパイル済みモジュール，テスト，または関数を表示や編集のために開く方法」を参照してください。

コンパイル済みモジュール内で定義された関数の利用

コンパイル済みモジュールは，他のテストから頻繁に呼び出す必要のあるユーザ定義関数をライブラリとして収めたスクリプトです。

関数をコンパイル済みモジュールに保存すれば，他のテストからそれらの関数が呼び出しやすくなります。

コンパイル済みモジュールをテストから読み込むときは，システム・モジュールとして，またはユーザ・モジュールとして，読み込むことができます。システム・モジュールは，テスト作業者からは見えないモジュールであり，使用頻度の高い，問題なく動作する完成版の関数が収められます。ユーザ・モジュールは，まだ開発途中のモジュールや使用頻度の比較的低いモジュールです。

コンパイル済みモジュールは起動テストから読み込むことができます。

コンパイル済みモジュールについて

コンパイル済みモジュールを読み込むと，その非静的関数が自動的にコンパイルされ，メモリに常駐します。これらの非静的関数は任意のテストの中から直接呼び出すことができます。

例えば，2 つのファイルのサイズを比較する関数や，システムの現在のメモリ・リソースを調べる関数を収めた，コンパイル済みモジュールを作成できます。

コンパイル済みモジュールによって，テストの編成や性能を向上させることができます。コンパイル済みモジュールは使用する前にデバッグするため，それらのモジュールから関数を呼び出すテストでは，必要になるエラー検査の量が減ります。また，すでにコンパイル済みの関数を呼び出す動作は，テスト・スクリプト内の関数をインタプリタ処理する動作よりもずっと高速です。

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場合は，コンパイル済みモジュールによって GUI マップ・ファイルが読み込まれることはありません。コンパイル済みモジュールから GUI オブジェクトを参照する場合は，そのコンパイル済みモジュールを読み込むテストの中でもそれらのオブジェクトを参照する必要があります。詳細については，第 6 章「テスト特有の GUI マップ・ファイル・モードでの作業」を参照してください。

システム・コンパイル済みモジュールとユーザ・コンパイル済みモジュールについて

コンパイル済みモジュールは，システム・コンパイル済みモジュールまたはユーザ・コンパイル済みモジュールとして読み込むことができます。

- 「システム・コンパイル済みモジュール」は，テスト作業者からは見えない閉じたモジュールです。読み込まれてもテスト・ウィンドウには表示されず，ステップインしたり，一時停止コマンドで停止したりすることはできません。システム・モジュールは，パラメータを指定せずに **unload** ステートメントを実行したとき（グローバル読み込み解除時）にアンロードされません。

  読み込まれているシステム・モジュールを関数ビューアに表示するかどうかを，選択することができます。標準設定では，システム・モジュールは表示されません。システム・モジュールを表示するには，［**ツール**］＞［**一般オプション**］を選択し，［**システム モジュールを表示する**］を選択します。この設定は次回 WinRunner を開いたときに有効になります。

- ユーザ・コンパイル済みモジュールは，ほとんどの点でシステム・モジュールとは正反対のものです。実行時に表示され，WinRunner のすべてのデバッグ・オプションを使用して実行を制御できます。通常，ユーザ・モジュールはまだ開発途中のモジュールです。開発中のモジュールでは，段階的に変更を加えてはコンパイルする，ということをします。

  コンパイル済みモジュールは，テスト・スクリプトの中で **load** TSL 関数を使用して読み込むときに，システム・モジュールかユーザ・モジュールのどちらかとして定義します。詳細については，737 ページ「TSL 関数を使用したコンパイル済みモジュールのロードとアンロード」を参照してください。関数ビューアの［**コンパイル済みモジュールのロード**］ボタンを使用してコンパイル済みモジュールを読み込むときは，常にユーザ・コンパイル済みモジュールとしてロードされます。

WinRunner の起動時にコンパイル済みモジュールを自動的に実行する方法

頻繁に使用する関数を収めたコンパイル済みモジュールを作成した場合に，そのモジュールを起動テストから読み込むことができます。これには，**load** ステートメントを起動テストに追加します。詳細については，第 45 章「特殊な構成の初期化」を参照してください。

「回復コンパイル済みモジュール」を読み込む場合は，**load** ステートメントを起動テストや他のテストに追加する必要はありません。復旧コンパイル済みモジュールは WinRunner の起動時に自動的に読み込まれます。復旧コンパイル済みモジュールの詳細については，第 26 章「回復シナリオの定義と使用」を参照してください。

コンパイル済みモジュールのロードとアンロード

コンパイル済みモジュール内の関数にアクセスするには，モジュールを読み込む必要があります。次の 3 つのうちいずれかの方法でモジュールを読み込むことができます。

- 関数ビューアの［**コンパイル済みモジュールをロード**］ボタンを使用してモジュールをロードします。
- TSL の **load** または **reload** 関数を使用してテスト・スクリプトからモジュールをロードします。**load** または **reload** 関数を使用して任意のテスト・スクリプトからコンパイル済みモジュールを読み込むことができます。
- WinRunner の［実行］コマンドを使用してモジュール・スクリプトを実行します。

  コンパイル済みモジュールを実行すると，モジュールがそのすべての関数と一緒にメモリに読み込まれ，関数ビューアに表示されます。この方法で読み込んだモジュールをアンロードするには，［**停止**］ボタンをクリックします。［**実行**］コマンドを使用して読み込んだモジュールに対しては，［**ロード解除**］および［**全モジュールのロード解除**］ボタンは機能しません。

  モジュールをデバッグする必要がある場合や，モジュールに変更を加える必要がある場合は，［ステップ］コマンドを使用して段階的なコンパイルを実行できます。変更を加えたモジュールは，変更を加えた部分を実行するだけでモジュール全体が更新されます。

以降，この節では上記のうち最初の 2 つの方法について説明します。

関数ビューアを使用したコンパイル済みモジュールのロードとアンロード

関数ビューアを使用してコンパイル済みモジュールのロードまたはアンロードを行うには，［**コンパイル済みモジュールのロード**］または［**ロード解除**］ツールバー・ボタンを使用します。この方法は，通常比較的大きな呼び出しチェーンの一部になっている個々のテストをデバッグする場合に特に便利です。例えば，呼び出しチェーン内の最初のテストで，チェーン内で呼び出されるすべてのテスト用のすべてのコンパイル済みモジュールがロードされるとします。チェーン内のテストを 1 つデバッグする必要がある場合，別のテストを実行してモジュールを読み込むのではなく，関数ビューアを使用してコンパイル済みモジュールを読み込むことができます。

［**コンパイル済みモジュールのロード**］ボタンを使用してコンパイル済みモジュールを読み込むと，ユーザ・コンパイル済みモジュールとしてロードされます。

［**ロード解除**］および［**全モジュールのロード解除**］ボタンを使用してコンパイル済みモジュールを関数ビューアからアンロードするときは，ツールバー・ボタンを 1 回クリックするだけで，1 つまたは複数のコンパイル済みモジュールがメモリから完全に消去されます。読み込まれたモジュールのインスタンスを個別にアンロードするには，**unload** TSL 関数を使用します。詳細については，737 ページ「TSL 関数を使用したコンパイル済みモジュールのロードとアンロード」を参照してください。

TSL 関数を使用したコンパイル済みモジュールのロードとアンロード

load コマンドを使用すると，任意のテスト・スクリプトの中からコンパイル済みモジュールを読み込むことができます。この後，WinRunner を終了するか，コンパイル済みモジュールをアンロードするまで，すべてのテストから関数にアクセスできるようになります。

テストを監視なしで実行できるようにするには，テストに **load** コマンドを挿入します。例えば，テストのデバッグが終了したとします。テストのデバッグ中は，関数ビューアを使用して必要なモジュールのロードとアンロードを行っていました。デバッグが終了したら，テストを監視なしで実行するには，関数を呼び出すテストの中で，または呼び出しチェーン内の前のテストで，**load** ステートメントを追加し，必要なモジュールをプログラムから読み込む必要があります。

TSL 関数を使用してすでに読み込まれたモジュールを，さらに読み込もうとしても，そのモジュールが WinRunner によってもう一度読み込まれることはありません。代わりに，変数が初期化され，「ロード・カウンタ」がインクリメントされます。モジュールがすでに複数回読み込まれている場合は，**unload** ステートメントが実行されてもモジュールはアンロードされず，カウンタがデクリメントされます。

例えば，テスト A でモジュール *math_functions* を読み込んだ後に，テスト B を呼び出すとします。テスト B でも *math_functions* を読み込んだ後，テストの最後にモジュールをアンロードします。テスト B の実行後，*math_functions* 内に定義されている関数を テスト A で呼び出します。仮に，unload 関数で，カウンタをデクリメントするのではなく，コンパイル済みモジュールを完全にアンロードしたとします。そのような場合，テスト B の **unload** 関数によって *math_functions* がメモリから完全にアンロードされるので，その後にテスト A から *math_functions* を呼び出すと失敗します。

このような状況を避けるために，カウンタが設けられています。カウンタにより，テスト B で *math_functions* のアンロードするとカウンタがデクリメントされますが，その後のテスト A からの呼び出しに対応できるように，*math_functions* はメモリに残ったままです。

➤ **load** 関数の構文は，次のとおりです。

load (*＜モジュール名＞* [, *＜モジュール・タイプ＞*] [, *＜オープン・ステータス＞*]);

＜モジュール名＞ は，既存のコンパイル済みモジュールの名前です。

2 つの追加の任意指定パラメータは，モジュールのタイプを示します。最初のパラメータは，関数モジュールがシステム・モジュールかユーザ・モジュールかを示します。1 はシステム・モジュールであることを示し，0 はユーザ・モジュールであることを示します。

（標準設定 = 0）

システム・モジュールとユーザ・モジュールの詳細については，734 ページ「システム・コンパイル済みモジュールとユーザ・コンパイル済みモジュールについて」を参照してください。

続く任意してパラメータは，「ユーザ」モジュールをロードした後も WinRunner ウィンドウ内で開いたままにするか，またはロード後に自動的に閉じるかを示します。1 は，モジュールが自動的に閉じることを示します。0 は，モジュールを開いたままにすることを示します。

（標準設定 = 0）

load 関数の初めての実行時に，モジュールがコンパイルされてメモリに格納されます。コンパイルされたモジュールは任意のテストから使用できる状態になり，再度インタプリタ処理する必要がなくなります。

読み込まれたモジュールは，テストの実行が中止されてもメモリ上に残ります。ただし，モジュールの中で定義されている変数は，静的かパブリックかにかかわらず，すべて初期化されます。

➤ **unload** 関数は，読み込まれたモジュールまたは選択された関数の最後のインスタンスを，メモリから削除します。構文は次のとおりです。

unload ([*＜モジュール名＞* | *＜テスト名＞* [, " *＜関数名＞* "]]);

例えば，次のステートメントは，mem_test というコンパイル済みモジュールの中で読み込まれているすべての関数を削除します。

```
unload ("mem_test");
```

引数を指定せずに **unload** ステートメントを実行した場合は，現在のセッション中にすべてのテストの中で読み込まれた，システム・モジュールを除くすべてのモジュールが削除されます。

異なるスクリプトから同じモジュールを複数回読み込んだ場合は，それらの load ごとに別々の unload ステートメントが必要です。詳細については，737 ページ「TSL 関数を使用したコンパイル済みモジュールのロードとアンロード」を参照してください。

➤ モジュールに変更を加えた場合，モジュールをロードしなおす必要があります。**reload** 関数は，読み込まれていたモジュールをメモリから削除し，ロードしなおします（**unload** 関数と **load** 関数の組み合わせです）。

reload 関数の構文は次のとおりです。

reload (*＜モジュール名＞* [, *＜モジュール・タイプ＞*] [, *＜オープン・ステータス＞*]);

＜モジュール名＞ は，既存のコンパイル済みモジュールの名前です。

2 つの追加の任意指定パラメータは，モジュールのタイプを示します。最初のパラメータは，モジュールがシステム・モジュールかユーザ・モジュールかを示します。1 はシステム・モジュールであることを示し，0 はユーザ・モジュールであることを示します。

（標準設定 = 0）

続く任意してパラメータは，「ユーザ」モジュールをロード後も WinRunner ウィンドウ内で開いたままにするか，またはロード後に自動的に閉じるかを示します。1 は，モジュールが自動的に閉じることを示します。0 は，モジュールを開いたままにすることを示します。

（標準設定 = 0）

注 : モジュールを再コンパイルするためにモジュールを複数回 load することは避けてください。モジュールを再コンパイルするには，**unload** 関数に続いて **load** 関数を使用するか，または **reload** 関数を使用してください。

コンパイル済みモジュールの例

次のモジュールには，任意のテストから呼び出すことのできる 2 つの簡単な汎用関数が含まれています。1 番目の関数は，数値のペアを受け取り，大きい方の値を返します。2 番目の関数は，数値のペアを受け取り，小さい方の値を返します。

```
# 2 つの値のうち大きい方を返す
function max (x,y)
{
    if (x>=y)
        return x;
    else
        return y;
}

# 2 つの値のうち小さい方を返す
function min (x,y)
{
    if (x>=y)
        return y;
    else    return x;
}
```

第 32 章

Web 例外処理の定義

WinRunner では，テスト実行中にテスト環境で起きる予期しないイベントやエラーを処理できます。

本章では，次の項目について説明します。

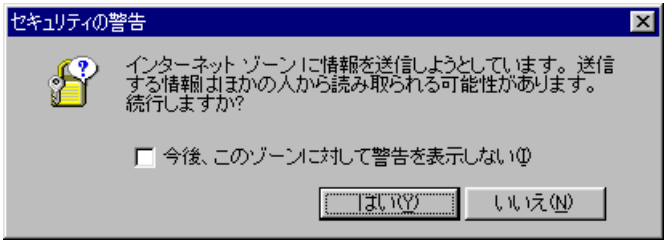
➤Web 例外の定義

➤例外の変更

➤ Web 例外を有効 / 無効にする

Web 例外の処理について

WebTest のアドインをロードする際，テスト実行中，Web サイトで発生する予期しないイベントやエラーを処理するよう WinRunner に指定できます。例えば，テスト実行中に［セキュリティの警告］ダイアログ・ボックスを表示すれば，［**はい**］ボタンをクリックして，テストの実行を元に戻すよう指定できます。



WinRunnerには，Web例外処理エディタでサポートする例外リストが含まれています。リストを変更して，WinRunner にサポートさせる例外を追加作成できます。

WebTest のアドインでの WinRunner のロード方法については，21 ページ「WinRunner アドインのロード」を参照してください。

Web 例外の定義

新規の例外を［Web 例外処理エディタ］のリストに定義できます。

Web 例外を定義するには，次の手順を実行します。

1 ［**ツール**］＞［**Web 例外処理**］を選択します。［Web 例外処理エディタ］が開きます。

2 指差しボタンをクリックして，ダイアログ・ボックスをクリックします。新規の例外がリストに追加されます。

3 例外を分類するには，［**タイプ**］リストで，例外のタイプを選択します。

エディタに例外のタイトル，MSW_Class，メッセージが表示されます。

4 ［**処理**］リストで，テストの実行を回復する処理関数の動作を選択します。

- **Web_exception_handler_dialog_click_default** は，標準のボタンをアクティブにします。
- **Web_exception_handler_fail_retry** は，標準のボタンをアクティブにし，Web ページを再ロードします。
- **Web_exception_enter_username_password** は，指定したユーザ名とパスワードを使用します。

- **Web_exception_handler_dialog_click_yes** は，［**はい**］ボタンをアクティブにします。
- **Web_exception_handler_dialog_click_no** は，［**いいえ**］ボタンをアクティブにします。

5 ［**適用**］をクリックして，［設定詳細を保存］ダイアログ・ボックスが開きます。

6 ［**OK**］をクリックして，構成ファイルに変更を保存します。

7 ［**エディタの終了**］をクリックして，［Web 例外処理エディタ］を終了します。

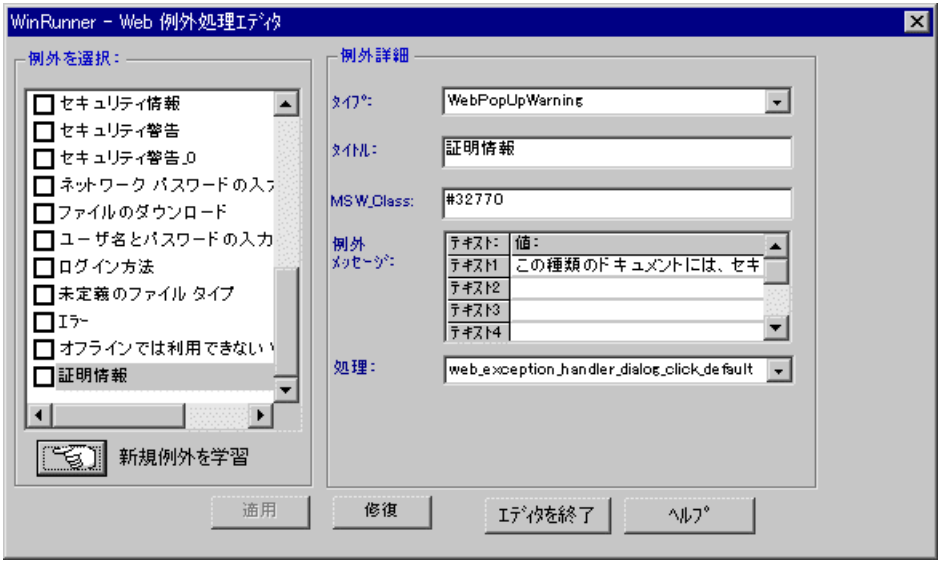
例外の変更

Web 例外処理エディタにリストされている例外の詳細を変更できます。

例外の詳細を変更するには，次の手順を実行します。

1 ［**ツール**］ > ［**Web 例外処理**］を選択します。［Web 例外処理エディタ］が開きます。

2 ［**例外を選択**］リストで，例外をクリックします。



例外が強調表示されます。例外の現在の記述が［例外詳細］に表示されます。

3 ダイアログ・ボックスのタイトルを変更するには，［**タイトル**］ボックスに新しいタイトルを入力します。

4 例外ダイアログ・ボックスに表示されるテキストを変更するには，テキストの列をクリックしてテキストを編集します。

5 テストの実行を回復する動作を変更するには，［**処理**］リストから動作を選択します。

➤ **Web_exception_handler_dialog_click_default** は，標準ボタンをアクティブにします。

- **Web_exception_handler_fail_retry** は，標準ボタンをアクティブにし，Web ページを再ロードします。
- **Web_exception_enter_username_password** は，指定したユーザ名とパスワードを使用します。
- **Web_exception_handler_dialog_click_yes** は，［**はい**］ボタンをアクティブにします。
- **Web_exception_handler_dialog_click_no** は，［**いいえ**］ボタンをアクティブにします。

6 ［**適用**］をクリックします。［設定詳細を保存］ダイアログ・ボックスが開きます。

7 ［**OK**］をクリックして，構成ファイルに変更を保存します。

8 ［**エディタの終了**］をクリックして，［Web 例外処理エディタ］を終了します。

Web 例外を有効 / 無効にする

Web 例外処理エディタには，使用できる例外すべてのリストが含まれています。リストの任意の例外を有効にするか無効にするか選択できます。

例外のステータスを変更するには，次の手順を実行します。

1 ［**ツール**］＞［**Web 例外処理**］を選択します。［Web 例外処理エディタ］が開きます。

2 ［**例外を選択**］リストで，例外をクリックします。例外が強調表示されます。例外の現在の記述が［例外詳細］に表示されます。

3 例外を有効にするには，そのチェック・ボックスを選択します。例外を無効にするには，チェック・ボックスをクリアします。

4 ［**適用**］をクリックします。［設定詳細を保存］ダイアログ・ボックスが開きます。

5 ［**OK**］をクリックして，構成ファイルに変更を保存します。

6 ［**エディタの終了**］をクリックして，Web 例外処理エディタを終了します。

第 33 章

外部ライブラリからの関数の呼び出し

WinRunner では，Windows API や任意の外部 DLL（ダイナミック・リンク・ライブラリ）の関数を呼び出すことができます。

本章では，以下の項目について説明します。

- 外部ライブラリからの関数の呼び出しについて
- 外部ライブラリの動的なロード
- TSL における外部関数の宣言
- Windows API の使用例

外部ライブラリからの関数の呼び出しについて

Windows API または任意の外部 DLL から関数を呼び出すことによって，自動テストの機能を強化できます。例えば，Windows API の関数を使用すれば，以下のようなことができます。

- *MessageBox* 関数を使用して，標準の Windows メッセージ・ボックスをテストで使用できます。
- *SendMessage* 関数を使用して，テスト対象アプリケーションに WM（Windows Message）メッセージを送信できます。
- *GetWindow* 関数を使用して，アプリケーションのウィンドウに関する情報を取得できます。
- *MessageBeep* 関数を使用して，システム・ビープ音をテストに組み込めます。
- *ShellExecute* 関数を使用して，任意の Windows プログラムを実行し，作業ディレクトリやウィンドウのサイズなど追加のパラメータを定義できます。

- *GetTextColor* 関数を使用して，テスト対象アプリケーションのフィールドのテキストの色を調べることができます。これは，テキストの色が操作のステータスを示すような場合に重要な機能です。
- *GetClipboard* 関数を使用して，Windows のクリップボードにアクセスできます。

_ _ stdcall 呼び出し規則を使えば，DLL からエクスポートされている任意の関数を呼び出すことができます。テスト対象アプリケーションの一部である DLL をロードして，エクスポートされている関数にアクセスすることもできます。

load_dll 関数を使用して，必要な関数を含んでいるライブラリを動的にロードします。実際に関数を呼び出す前に，*extern* 宣言を書いて，関数が外部ライブラリにあることをインタプリタに知らせなければなりません。

注：特定の Windows API 関数は『**Windows API リファレンス**』を参照してください。WinRunner のテスト・スクリプトでの Windows API 関数の使用方法の詳細については，Windows インストール・フォルダの **¥lib¥win32api** フォルダの *read.me* ファイルを参照してください。

外部ライブラリの動的なロード

呼び出したい関数を含む外部 DLL（ダイナミック・リンク・ライブラリ）をロードするには，TSL 関数 **load_dll** を使います。この関数は，32 ビット DLL を実行時に読み込みます。この関数の構文は次のとおりです。

load_dll (*pathname* **);**

pathname は，読み込む DLL の完全パス名です。

次に例を示します。

```
load_dll ("h:¥¥qa_libs¥¥os_check.dll");
```

読み込んだ外部 DLL をアンロードするには, TSL 関数 **unload_dll** を使用します。この関数の構文は次のとおりです。

unload_dll (*pathname* **);**

次に例を示します。

```
unload_dll ("h:¥¥qa_libs¥¥os_check.dll");
```

pathname は，削除する 32 ビット DLL の完全パス名です。

メモリからすべての読み込み済みの DLL を削除するには，次の構文を使用します。

```
unload_dll ("");
```

詳細については，「**TSL リファレンス**」を参照してください。

TSL における外部関数の宣言

外部ライブラリから関数を呼び出すには，呼び出したい各関数について，*extern* 宣言を行わなければなりません。extern 宣言は，関数呼び出しの前になければなりません。これらの宣言は，起動テストに格納することをお奨めします（起動テストの詳細については，第 45 章「特殊な構成の初期化」を参照してください)。

extern 宣言の構文は，次のとおりです。

extern *type function_name* **(** *parameter1*, *parameter2*,.. **);**

type は，この関数の戻り値です。type は次のいずれかになります。

char（signed と unsigned）	*float*
short（signed と unsigned）	**double**
int（signed と unsigned）	*string*（C の char* と同等）

各パラメータ（*parameter*）には，次の情報が含まれなければなりません。

[mode] *type* [name] [<*size*>]

mode は，*in*，*out* または *inout* のいずれかです。標準では *in* です。これらは小文字で指定しなければなりません。

type は，先述のいずれかです。

省略可能な *name* は，可読性を高めるためにパラメータに指定する名前です。

<*size*> は，*string* 型の *out* または *inout* パラメータにのみ必要です（下記を参照してください)。

例えば，時計アプリケーションの時間を設定する set_clock という関数を呼び出したいとします。この関数は，**load_dll** ステートメントを使用して読み込んだ外部 DLL に含まれています。この関数を宣言するには次のように記述します。

```
extern int set_clock (string name, int time);
```

set_clock 関数はパラメータを 2 つ受け取ります。どちらも入力パラメータであるため，モード（mode）は指定されていません。文字列である最初のパラメータは，時計ウィンドウの名前です。2 つめのパラメータは，時計に設定する時間を指定します。この関数は，操作が成功したかどうかを示す整数値を返します。

extern 宣言を解釈させたら，TSL 関数を呼び出すのと同じように，set_clock 関数を呼び出すことができます。

```
result = set_clock ("clock v. 3.0", 3);
```

extern 宣言に，*string* 型である *out* または *inout* パラメータが含まれている場合，パラメータの型（*type*）または（省略可能な）名前（*name*）の後ろに整数値 <*size*> を指定して，文字列の最大長を予測しなければなりません。例えば，次のステートメントは，get_clock_string という関数を宣言しています。この関数は，時計アプリケーションに表示されている時間を，「The time is ...」という形式の文字列値として返します。

```
extern int get_clock_string (string clock, out string time <20>);
```

size にはオーバーフローが起こらないような大きさを指定しなければなりません。*size* に何も指定しなかった場合，標準の大きさは 100 です。

TSL は，コード内の関数を名前だけで識別します。TSL から正しいパラメータ情報を関数に渡さなければなりません。TSL は，パラメータの検査を行いません。情報に間違いがあると，操作は失敗します。

注：外部 DLL で関数から文字列値を返す場合は，戻り値よりも出力パラメータを使用することを推奨します。

例えば，DLL が次のようになっているとします。

```
int foo(char* szRetString)
{
  ...
  strcpy(szRetString, "hi");
  return nErrCode;
}
```

対応する **extern** ステートメントは次のようになります。

```
extern int foo(out string);
```

また，エクスポートされる関数は，以下の規則に従わなければなりません。

- TSL で *string* として指定されたパラメータは，*char** 型のパラメータに対応しなければなりません。
- TSL で *out* または *inout* モードと指定されたパラメータは，エクスポートされた関数のポインタに対応しなければなりません。例えば，TSL でパラメータを *out int* と指定した場合，そのパラメータはエクスポートされた関数では *int** 型のパラメータに対応しなければなりません。
- 外部関数は，標準の Pascal 呼び出し規則である *export far Pascal* に準拠しなければなりません。

例えば，次の TSL の宣言を見てください。

```
extern int set_clock (string name, inout int time);
```

これは，外部関数では次のように表示されます。

```
int set_clock(
        char* name,
        int* time
        );
```

Windows API の使用例

次のサンプル・テストでは，Windows API の関数を呼び出しています。

ウィンドウのニーモニックの検査

次のテストは，オブジェクトのラベルのニーモニック（キーボード・ショートカットを示す下線の付いた文字）を検査する TSL 関数に *GetWindowText* という API 関数を統合します。この TSL 関数は，パラメータ（オブジェクトの論理名）を 1 つ取得します。オブジェクトのラベルにニーモニックがない場合，メッセージがレポートに送信されます。

```
# 適切な DLL を読み込む（Windows ディレクトリから）
load (“win32api”);

# ユーザ定義関数「check_labels」を定義する。
public function check_labels(in obj)
{
    auto hWnd,title,pos,win;
    win = GUI_get_window();
    obj_get_info(obj,"handle",hWnd);
    GetWindowTextA(hWnd,title,128);
     pos = index(title,"&");
    if (pos == 0)
        report_msg("No mnemonic for object: "& obj & "in window: "& win);
}

# メモ帳アプリケーションを起動する。
invoke_application("notepad.exe","","",SW_SHOW);

# ［検索］ウィンドウを開く。
set_window (" メモ帳 ");
menu_select_item (" 検索 (S); 検索 (F)...");

# 「Up」ラジオ・ボタンと「キャンセル」プッシュボタンのニーモニックを検査する。
set_window (" 検索 ");
check_labels (" 上へ (U)");
check_labels (" キャンセル ");
```

DLL と外部関数の読み込み

以下に，crk_w.dll を使用して，デバッグ対象アプリケーションの記録を行わないようにするテストの一部を示します。記録を行わないようにするには，外部関数 *set_debugger_pid* を呼び出します。

```
# 適切な DLL を読み込む。
load_dll("crk_w.dll");

# 関数を宣言する。
extern int set_debugger_pid(long);

# システムの DLL を（Windows ディレクトリから）ロードする。
load ("win32api");

# デバッガ・プロセスの ID を検索する。
win_get_info("Debugger","handle",hwnd);
GetWindowThreadProcessId(hwnd,Proc);

# WinRunner にデバッガ・プロセスの ID を通知する。
set_debugger_pid(Proc);
```

第 34 章

関数の生成

ビジュアル・プログラミング機能を使えば，テスト・スクリプトに TSL ステートメントをすばやく簡単に追加できます。

本章では、以下の項目について説明します。

- 関数の生成について
- GUI オブジェクトを対象とする関数の生成
- リストからの関数の選択
- 引数値の割り当て
- カテゴリの標準関数の変更

関数の生成について

テストを記録すると，GUI オブジェクトをクリックしたり，キーボードを使うたびに，WinRunner はテスト・スクリプトに TSL ステートメントを生成します。TSL には，記録可能な関数の他に，テストの機能と柔軟性を高める関数が数多く用意されています。WinRunner のビジュアル・プログラミング・ツールである関数ジェネレータを使って，テスト・スクリプトに簡単に関数を追加できます。

関数ジェネレータを使えば，スクリプトをすばやく誤りなくプログラミングできます。以下のことが可能です。

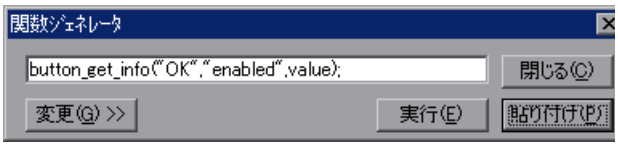
- GUI オブジェクトで操作を行ったり，テスト対象アプリケーションから情報を取得するコンテキスト・センシティブ関数を追加できます。
- テスト実行の同期化やユーザ定義メッセージのレポートへの出力など，非コンテキスト・センシティブな操作を行う標準関数やアナログ関数を追加できます。

➤ テスト環境に合わせて WinRunner を構成設定するためのカスタマイズ関数を追加できます。

関数ジェネレータを使ってテスト・スクリプトに TSL ステートメントを追加する方法は 2 つあります。1 つは GUI オブジェクトを指す方法で，もう 1 つはリストから関数を選択する方法です。［関数挿入］コマンドを選択して，GUI オブジェクトをポイントすると，WinRunner は適切なコンテキスト・センシティブ関数を提示し，その引数に値を割り当てます。提示されたものをそのまま受け入れるか，引数の値を変えるか，全く別の関数を選択できます。

標準で WinRunner はオブジェクトに対して標準の関数を提示します。提示する関数は多くの場合「get」系の関数か、あるいはオブジェクトに関する情報を返す関数です。例えば，［**挿入**］＞［**関数**］＞［**オブジェクト / ウィンドウ**］を選択して，［OK］ボタンをクリックすると，WinRunner は［関数ジェネレータ］ダイアログ・ボックスを開き，次のステートメントを生成します。

```
button_get_info("OK",”enabled”, value);
```



このステートメントは，［OK］ボタンの enabled プロパティを調べ，このプロパティの現在の値を *value* 変数に保存します。*value* の値は，1（使用可能）または 0（使用不能）のいずれかとなります。

他のオブジェクトの関数を選択するには，［**変更**］をクリックします。ステートメントを生成すると、次の 2 つのオプションのいずれかまたは両方を実行できます。

➤ テスト・スクリプトにステートメントを「**貼り付け**」る方法。必要であれば，生成されたステートメントの直前に，**set_window** ステートメントが自動的に挿入されます。

➤ 関数ジェネレータからステートメントを「**実行**」する方法。

GUI マップに含まれていないオブジェクトを指すと，生成されたステートメントを実行したとき，またはテスト・スクリプトに貼り付けたとき，そのオブジェクトが自動的に仮 GUI マップ・ファイルに追加されます。

注：関数ジェネレータをカスタマイズして，テスト・スクリプトでもっとも頻繁に使用するユーザ定義の関数を含めることができます。新しい関数，および新しいカテゴリとサブ・カテゴリを関数ジェネレータに追加できます。新しいカテゴリに対して標準の関数を設定することもできます。詳細は，第 44 章「関数ジェネレータのカスタマイズ」を参照してください。既存のカテゴリの標準の関数を変更することもできます。詳細は，765 ページ「カテゴリの標準関数の変更」を参照してください。

GUI オブジェクトを対象とする関数の生成

関数ジェネレータを使えば，アプリケーションの GUI オブジェクトをポイントするだけでコンテキスト・センシティブ関数を生成できます。WinRunner は対象オブジェクトを調べ，クラスを判断し，適切な関数を提示します。この標準の関数をそのまま受け入れたり，リストから別の関数を選択したりできます。

GUI オブジェクトに対する標準の関数の使用

アプリケーションの GUI オブジェクトを指して関数の生成を行う場合，WinRunner はオブジェクトのクラスを判断し，関数を提示します。大半のクラスでは，「**get**」系の関数が標準の関数として提示されます。例えば，リストをクリックした場合，WinRunner は，**list_get_selected** 関数を提示します。

GUI オブジェクトの標準の関数を使うには，次の手順を実行します。

1 ［**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。WinRunner はアイコン化され，マウス・ポインタが指差し型に変わります。

2 テスト対象アプリケーションの GUI オブジェクトをポイントします。指差しポインタがオブジェクトの上を通ると，オブジェクトが点滅します。

3 対象オブジェクトの上でマウスの左ボタンをクリックします。［**関数ジェネレータ**］ダイアログ・ボックスが開き，選択したオブジェクトに対する標準の関数が表示されます。WinRunner は，自動的に関数に引数の値を割り当てます。

オブジェクトを選択せずに操作を取り消したい場合は，マウスの右ボタンをクリックします。

4 テスト・スクリプトにステートメントを「**貼り付ける**」には，［**貼り付け**］をクリックします。関数が，テスト・スクリプトの挿入ポイントの位置に貼り付けされ，［関数ジェネレータ］ダイアログ・ボックスが閉じます。

関数を「**実行**」するには，［**実行**］をクリックします。関数は実行されますが，［**実行**］のクリックではスクリプトには貼り付けられません。

5 ［**閉じる**］ボタンをクリックしてダイアログ・ボックスを閉じます。

GUI オブジェクトの標準の関数の選択

WinRunner が提示する標準の関数を使いたくない場合には，リストから他の関数を選択できます。

GUI の非標準の関数を選択するには次の手順を実行します。

1 ［**挿入**］＞［**関数**］＞［**オブジェクト / ウィンドウ**］を選択するか，［**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。WinRunner は最小化され，マウス・ポインタが指差し型に変わります。

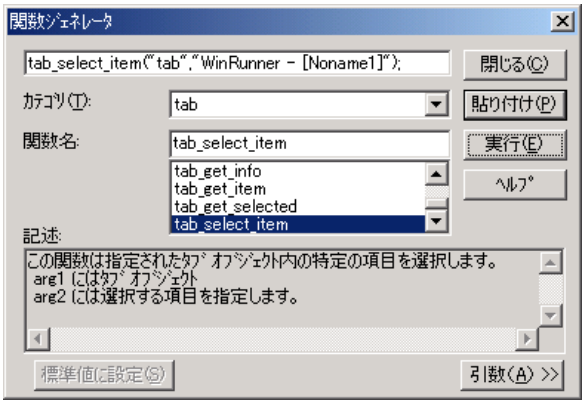
2 テスト対象アプリケーションの GUI オブジェクトを指します。指差しポインタがオブジェクトの上を通ると，オブジェクトが点滅します。

3 対象オブジェクトの上でマウスの左ボタンをクリックします。［**関数ジェネレータ**］ダイアログ・ボックスが開き，選択したオブジェクトに対する標準の関数が表示されます。WinRunner は，自動的に関数に引数の値を割り当てます。

オブジェクトを選択せずに操作を取り消したい場合は，マウスの右ボタンをクリックします。

4 ［関数ジェネレータ］ダイアログ・ボックスで，［**変更**］ボタンをクリックします。ダイアログ・ボックスが拡張し，関数のリストが表示されます。リストには，選択した GUI オブジェクトに対して使用できる関数だけが含まれています。例えば，プッシュ・ボタンを選択した場合，リストには，**button_get_info** や **button_press** などの関数が表示されます。

5 ［**関数名**］リストから関数を選択します。生成されたステートメントがダイアログ・ボックスの最上部に表示されます。WinRunner は自動的に引数の値を埋めます。関数の説明がダイアログ・ボックスの最下部に表示されます。



6 引数の値を変更する場合は，［**引数**］ボタンをクリックします。ダイアログ・ボックスが拡張し，各引数に対応するテキスト・ボックスが表示されます。引数のテキスト・ボックスに入力を行う方法の詳細については，763 ページ「引数値の割り当て」を参照してください。

7 テスト・スクリプトにステートメントを「**貼り付ける**」には，［**貼り付け**］をクリックします。関数が，テスト・スクリプトの挿入ポイントの位置に貼り付けます。

関数を「**実行**」するには，［**実行**］をクリックします。関数はすぐに実行されますが，テスト・スクリプトには貼り付けられません。

8 ダイアログ・ボックスを閉じずに上記の手順を繰り返して，同じオブジェクトに対する関数ステートメントを続けて生成できます。選択したオブジェクトがつねに対象のオブジェクトとなり，どの関数を選択しても引数が自動的に埋められます。

9 ダイアログ・ボックスを閉じるには［**閉じる**］を選択します。

リストからの関数の選択

テストをプログラミングする際に，テストに特定の処理を行わせたいが，それを行うための関数が分からないことがあるかもしれません。関数ジェネレータでは，必要な関数をすばやく見つけ出し，それをテスト・スクリプトに挿入できます。関数は，カテゴリに分類されています。適切なカテゴリを選択し，リストから必要な関数を選択します。関数の説明が，そのパラメータとともに表示されます。

リストから関数を選択するには，次の手順を実行します。

1 ［**挿入**］＞［**関数**］＞［**関数ジェネレータから**］を選択するか，ユーザ定義ツールバーの［**関数ジェネレータから関数を挿入**］**ボタン**をクリックして，［関数ジェネレータ］ダイアログ・ボックスを開きます。

2 ［**カテゴリ**］リストから関数カテゴリを選択します。例えば，メニュー関数を表示したい場合は，「menu」を選択します。必要なカテゴリが分からない場合は，標準の「**すべての関数**」を選択します。アルファベット順にすべての関数が表示されます。

3 ［**関数名**］リストから関数を選択します。特定のカテゴリを選択した場合は，そのカテゴリの関数だけがリストに表示されます。生成されたステートメントがダイアログ・ボックスの最上部に表示されます。WinRunner は，自動的に標準の引数の値を埋めます。関数の説明がダイアログ・ボックスの最下部に表示されます。

4 引数の値を定義または変更するには，［**引数**］をクリックします。ダイアログ・ボックスが拡張し，各引数に対応するテキスト・ボックスが表示されます。引数のテキスト・ボックスに入力する方法については，763 ページ「引数値の割り当て」を参照してください。

5 テスト・スクリプトにステートメントを「**貼り付ける**」には，［**貼り付け**］をクリックします。関数がテスト・スクリプトの挿入ポイントの位置に貼り付けられます。

関数を「**実行**」するには，［**実行**］をクリックします。関数はすぐに実行されますが，テスト・スクリプトには貼り付けられません。

6 ダイアログ・ボックスを閉じずに上記の手順を繰り返して，追加の関数ステートメントを続けて生成できます。

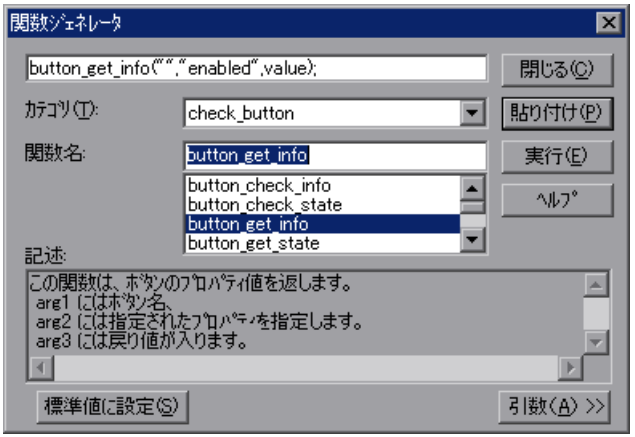
7 ダイアログ・ボックスを閉じるには［**閉じる**］を選択します。

引数値の割り当て

［関数ジェネレータ］を使って関数を生成すると，WinRunner は自動的に関数の引数に値を割り当てます。GUI オブジェクトをクリックして関数を生成した場合，WinRunner は対象オブジェクトを評価し，適切な引数値を割り当てます。リストから関数を選択した場合，WinRunner は可能であれば標準の値を割り当て，残りはユーザが入力します。

生成された関数の引数値の割り当てまたは変更を行うには，次の手順を実行します。

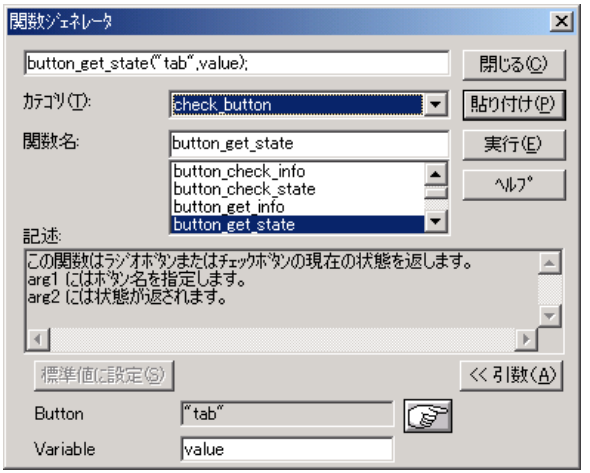
1 ［**挿入**］＞［**関数**］＞［**関数ジェネレータから**］を選択するか，ユーザ定義ツールバーの［**関数ジェネレータから関数を挿入**］ボタンをクリックして，［関数ジェネレータ］ダイアログ・ボックスを開きます。



2 ［**カテゴリ**］リストから関数カテゴリを選択します。例えば，メニュー関数を表示したい場合は，「menu」を選択します。どのカテゴリが必要かが分からない場合は，標準の「**すべての関数**」を選択します。アルファベット順にすべての関数が表示されます。

3 ［**関数名**］リストから関数を選択します。特定のカテゴリを選択した場合は，そのカテゴリの関数だけがリストに表示されます。生成されたステートメントがダイアログ・ボックスの最上部に表示されます。WinRunner は，自動的に標準の引数値を埋めます。関数の説明がダイアログ・ボックスの最下部に表示されます。

4 ［**引数**］をクリックします。ダイアログ・ボックスに関数の引数が表示されます。



5 引数に値を割り当てます。値の割り当ては手作業で行うことも，自動的に行うこともできます。

値を「**手作業で**」割り当てるには，引数のテキスト・ボックスに値を直接入力します。テキスト・ボックスによっては，値をリストから選択できる場合もあります。

値を「**自動的に**」割り当てるには，指差しボタンをクリックした後，アプリケーションのオブジェクトをクリックします。適切な値が引数テキスト・ボックスに表示されます。

選択した関数を適用できないオブジェクトをクリックした場合，選択したオブジェクトにこの関数は適用されないという内容のメッセージが表示されます。［**OK**］をクリックしてメッセージを消去し，［関数ジェネレータ］に戻ります。

カテゴリの標準関数の変更

関数ジェネレータでは，各関数カテゴリに標準の関数が定義されています。アプリケーションのオブジェクトをクリックして関数を生成した場合，WinRunner はオブジェクトに対応する適切なカテゴリを判断し，標準の関数を提示します。大半のコンテキスト・センシティブ関数のカテゴリでは，標準の関数は「**get**」系の関数です。例えば，テキスト・ボックスをクリックした場合，標準の関数は **edit_get_text** です。アナログ，標準，およびカスタマイズ関数カテゴリの場合，そのカテゴリで最も頻繁に使われる関数が標準の関数です。例えば，system カテゴリの標準の関数は，**invoke_application** です。

カテゴリの標準の関数とは別の関数を使うことが多い場合は，その関数を標準の関数に設定することもできます。

カテゴリの標準の関数を変更するには，次の手順を実行します。

1 ［**挿入**］＞［**関数**］＞［**関数ジェネレータから**］を選択するか，ユーザ定義ツールバーの［**関数ジェネレータから関数を挿入**］**ボタン**をクリックして，［関数ジェネレータ］ダイアログ・ボックスを開きます。

2 ［**カテゴリ**］リストから関数カテゴリを選択します。例えば，メニュー関数を表示したい場合は，「メニュー」を選択します。

3 ［**関数名**］リストから標準に設定したい関数を選択します。

4 ［**標準値に設定**］をクリックします。

5 ［**閉じる**］**をクリックします。**

このように設定した標準の関数は，別の関数に変更するか，WinRunner を終了するまでカテゴリの標準の関数になります。

標準の関数の設定を恒久的に保存する場合は，初期化テストに **generator_set_default_function** ステートメントを追加します。初期化テストの詳細については，第 45 章「特殊な構成の初期化」を参照してください。

generator_set_default_function 関数の構文は，次のとおりです。

generator_set_default_function (*category_name*, *function_name* **);**

例えば，以下は **button_press** 関数を push_button カテゴリの標準の関数として設定します。

```
generator_set_default_function ("push_button", "button_press");
```

第 35 章

対話形式の入力用ダイアログ・ボックスの作成

WinRunner を使用すると，対話型のテストの実行中にテストに入力を渡す際に使用できるダイアログ・ボックスを作成できます。

本章では，以下の項目について説明します。

- 対話形式の入力用ダイアログ・ボックスの作成について
- 入力ダイアログ・ボックスの作成
- リスト・ダイアログ・ボックスの作成
- ユーザ定義ダイアログ・ボックスの作成
- 参照ダイアログ・ボックスの作成
- パスワード・ダイアログ・ボックスの作成

対話形式の入力用ダイアログ・ボックスの作成について

対話型のテストの実行中にポップアップして，テキストを入力したりリストから項目を選んだりといったユーザのアクションを求めるダイアログ・ボックスを作成できます。これは，ユーザが実行時のテスト対象アプリケーション（AUT）の動作に基づいて決断を下して，それに応じて入力を行わなければならないときに便利です。例えば，ダイアログ・ボックスに入力されたユーザ名に従って特定のテスト群を実行するように WinRunner に命令できます。

このようなダイアログ・ボックスを作成するには，テスト・スクリプトの適切な場所に TSL ステートメントを入力します。対話型のテスト実行中にそのステートメントが実行されるとダイアログ・ボックスが開きます。制御フロー・ステートメントを使用すれば，それぞれの場合にユーザの入力に対して WinRunner がどのように対応するか指定できます。

次の TSL 関数を使用して，5 種類のダイアログ・ボックスを作成できます。

- **create_input_dialog** では，自分が指定したメッセージと編集フィールドを含むダイアログ・ボックスを作成できます。この関数は対話型のテストの実行中に編集フィールドに入力されたすべての内容を含む文字列を返します。
- **create_list_dialog** では項目のリストとメッセージを含むダイアログ・ボックスを作成できます。この関数は対話型のテストの実行中に選択した項目を含む文字列を返します。
- **create_custom_dialog** では，編集フィールド，チェック・ボックス，「実行」ボタン，ボタンが含まれるダイアログ・ボックスを作成できます。「実行」ボタンがクリックされると，**create_custom_dialog** 関数は指定された機能を実行します。
- **create_browse_file_dialog** では，ユーザがファイルを選択できる参照ダイアログ・ボックスが表示されます。対話実行中，この関数は選択されたファイルの名前を含む文字列を返します。
- **create_password_dialog** では，ログイン名の入力用とパスワードの入力用の 2 つの編集フィールドを持つダイアログ・ボックスを作成します。パスワードのダイアログ・ボックスを使って，テストやテストの一部に対するユーザのアクセスを制限できます。

どのダイアログ・ボックスも，それを作成するステートメントがテストの実行中に実行されると開き，ダイアログ・ボックス内のいずれかのボタンがクリックされると閉じます。

入力ダイアログ・ボックスの作成

入力ダイアログ・ボックスには，ユーザが指定した 1 行メッセージと編集フィールド，［**OK**］ボタン，［**キャンセル**］ボタンが含まれます。テストの実行中にユーザが編集フィールドに入力したテキストは，文字列として返されます。

TSL 関数 **create_input_dialog** を使用して，入力ダイアログ・ボックスを作成します。この関数の構文は次のとおりです。

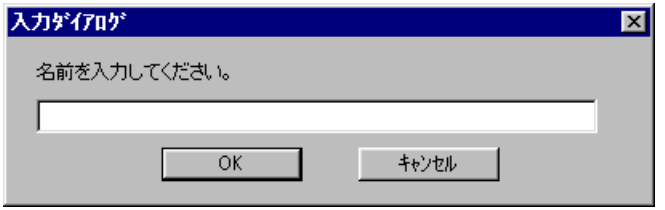
create_input_dialog (*message* **);**

(*message***)** はどのような式のものでも構いません。このテキストはダイアログ・ボックス内に 1 行で表示されます。

例えば，ユーザ名の入力を求める入力ダイアログ・ボックスを作成できます。この名前は変数に渡され，ユーザの名前に応じて特定のテスト・スイートを呼び出せるよう，**if** ステートメント中で使用されます。

こうしたダイアログ・ボックスを作成するには，次のステートメントをプログラミングします。

```
name = create_input_dialog (" 名前を入力してください。");
```



テスト実行中のダイアログ・ボックスへの入力は，［**OK**］ボタンをクリックすると変数 *name* に渡されます。［**キャンセル**］ボタンをクリックすると，空の文字列（空の引用符）が変数 *name* に渡されます。

また，message パラメータの前には感嘆符を付けることができます。この場合，ユーザが編集フィールドに入力すると，入力された各文字はアスタリスクで表されます。感嘆符は，秘密の情報を他人に見られないようにしたいときに使用します。

リスト・ダイアログ・ボックスの作成

リスト・ダイアログ・ボックスには，タイトルと選択できる項目のリストが含まれます。ユーザがリストから選択した項目は，変数に文字列として渡されます。

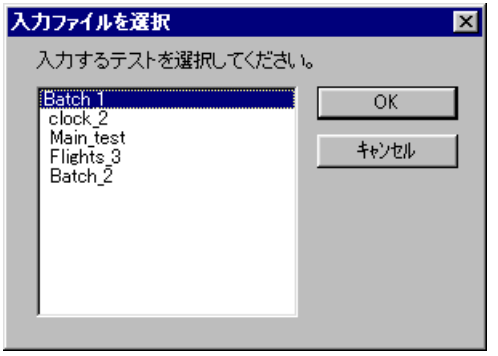
TSL 関数 **create_list_dialog** を使用して，リスト・ダイアログ・ボックスを作成します。この関数の構文は次のとおりです。

create_list_dialog (*title*, ***message***, *list_items* **);**

- *title* はダイアログ・ボックス・ウィンドウのタイトル・バーに表示される文字列です。
- *message* はダイアログ・ボックス内に表示される 1 行のテキストです。
- *list_items* にはダイアログ・ボックスに表示されるオプションが含まれます。項目はカンマで区切られ，リスト全体が 1 つの文字列として扱われます。

例えば，開くテストを選択できるダイアログ・ボックスを作成できます。これを行うには，次のように入力します。

```
filename = create_list_dialog (" 入力ファイルを選択 ", " 入力するテストを選択してください。", "Batch_1, clock_2, Main_test, Flights_3, Batch_2");
```



［**OK**］ボタンをクリックすると，テストの実行中にリストから選択した項目が変数 *filename* に渡されます。［**キャンセル**］ボタンをクリックすると，空の文字列（空の引用符）が変数 *filename* に渡されます。

ユーザ定義ダイアログ・ボックスの作成

ユーザ定義ダイアログ・ボックスには，1 つのユーザ定義のタイトル，上限 10 個の編集フィールド，上限 10 個のチェック・ボックス，1 つの「実行」ボタン，1 つの［キャンセル］ボタンが含まれます。ユーザは「実行」ボタンにラベルを指定できます。「実行」ボタンをクリックすると，指定した関数が実行されます。関数には，TSL 関数とユーザ定義関数のどちらも使用できます。

TSL 関数 **create_custom_dialog** を使用して，ユーザ定義ダイアログ・ボックスを作成します。この関数の構文は次のとおりです。

create_custom_dialog (*function_name*, *title, button_name, edit_name*$_{1\text{-}n}$, *check_name*$_{1\text{-}m}$ **);**

- *function_name* は，「実行」ボタンをクリックすると実行される関数名です。
- *title* はダイアログ・ボックスのタイトル・バーに表示される文字列です。
- *button_name* は「実行」ボタンに表示されるラベルです。このボタンをクリックして，指定した関数を実行します。
- *edit_name* は，ダイアログ・ボックスの編集フィールドのラベルです。編集フィールドの複数のラベルはカンマで区切って指定し，ラベルはまとめて単独の文字列として扱われます。ダイアログ・ボックスに編集フィールドを含まない場合は，このパラメータは空の文字列（空の引用符）になります。
- *check_name* はダイアログ・ボックスのチェック・ボックスのラベルです。複数のチェック・ボックスのラベルはカンマで区切って指定し，ラベルはまとめて単独の文字列として扱われます。ダイアログ・ボックスにチェック・ボックスを含まない場合は，このパラメータは空の文字列（空の引用符）になります。

「実行」ボタンがクリックされると，ユーザが入力した値は指定された関数に次の順番でパラメータとして渡されます。

edit_name$_1$,... *edit_name*$_n$,*check_name*$_1$,... *check_name*$_m$

次の例で，ユーザはユーザ定義ダイアログ・ボックスでアプリケーションの初期パラメータを指定できます。ユーザが［**実行**］ボタンをクリックすると，ユーザ定義の関数である run_application1 はユーザが指定した初期条件で特定の Windows アプリケーションを呼び出します。

```
res = create_custom_dialog ("run_application1", " 初期条件 ", " 実行 ",
    " アプリケーション :, 形状 :, 背景 :, 前景 :, フォント :", " 音 , 速度 ");
```

指定された関数が値を返すと，この値は変数 *res* に渡されます。［**キャンセル**］ボタンがクリックされると，空の文字列（空の引用符）が変数 *res* に渡されます。

編集フィールドのラベルの前には感嘆符を付けることができます。この場合，ユーザが編集フィールドに入力をすると，入力された各文字はアスタリスクで表されます。感嘆符は，パスワードなど秘密の情報を他人に見られないようにしたいときに使用します。

参照ダイアログ・ボックスの作成

参照ダイアログ・ボックスを使用すると，ファイルのリストからファイルを選択でき，選択されたファイル名が文字列として返されます。

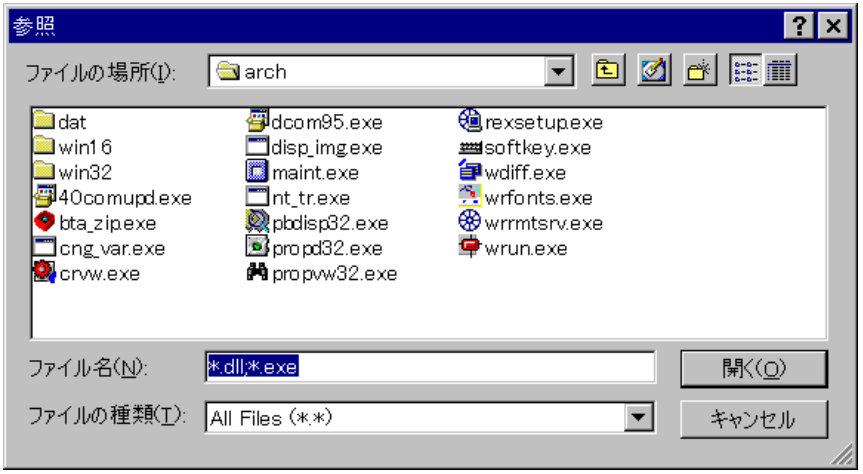
TSL 関数 **create_browse_file_dialog** を使用して，参照ダイアログ・ボックスを作成します。この関数の構文は次のとおりです。

create_browse_file_dialog (*filter* **);**

filter には，参照ダイアログ・ボックスに表示するファイルのフィルタを設定します。ワイルド・カードを使用してすべてのファイル「*.*」を表示することも，特定のファイル「*.exe や *.txt など」だけを表示することもできます。

次の例では , 参照ダイアログ・ボックスに .dll または .exe という拡張子の付いたファイルがすべて表示されます。

```
filename = create_browse_file_dialog( "*.dll;*.exe"  );
```



［**開く**］ボタンがクリックされると , 選択されたファイル名とパスが変数 *filename* に渡されます。［**キャンセル**］ボタンがクリックされると , 空の文字列（空の引用符）が変数 *filename* に渡されます。

パスワード・ダイアログ・ボックスの作成

パスワード用のダイアログ・ボックスには 2 つの編集フィールド，［OK］ボタン , ［キャンセル］ボタンが含まれます。編集フィールドにはラベルを指定できます。対話型のテストの実行中にユーザが編集フィールドに入力したテキストは，分析用として変数に保存されます。

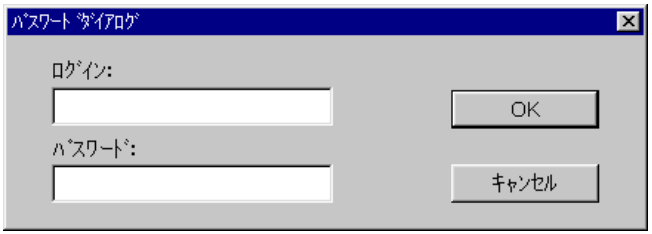
TSL 関数 **create_password_dialog** を使用して , パスワード・ダイアログ・ボックスを作成します。この関数の構文は次のとおりです。

create_password_dialog (*login, password, login_out, password_out* **);**

- *login* は，ユーザ名を入力するための最初の編集フィールドのラベルです。空の文字列（空の引用符）を指定すると，標準のラベル「ログイン：」が表示されます。
- *password* は，パスワードを入力するための 2 番目の編集フィールドのラベルです。空の文字列（空の引用符）を指定すると，標準時のラベル「パスワード：」が表示されます。ユーザがこの編集フィールドに入力をすると，画面には入力した文字ではなくアスタリスクが表示されます。
- *login_out* は，1 番目の編集フィールド「ログイン」の内容が渡されるパラメータの名前です。このパラメータを使用して，［ログイン：］編集フィールドの内容を検証します。
- *password_out* は，2 番目の編集フィールド「パスワード」の内容が渡されるパラメータの名前です。このパラメータを使用して，［パスワード：］編集フィールドの内容を検証します。

次に，編集フィールドに標準時のラベルを使ったパスワード用のダイアログ・ボックスを示します。

```
status = create_password_dialog ("",  "", user_name, password);
```



［**OK**］ボタンがクリックされると，値 1 が変数 *status* に渡されます。［**キャンセル**］ボタンがクリックされると，値 0 が変数 *status* に渡され，*user_name* と *password* パラメータに空の文字列が割り当てられます。

第 9 部

テストの実行 ー 上級

第 36 章

バッチ・テストの実行

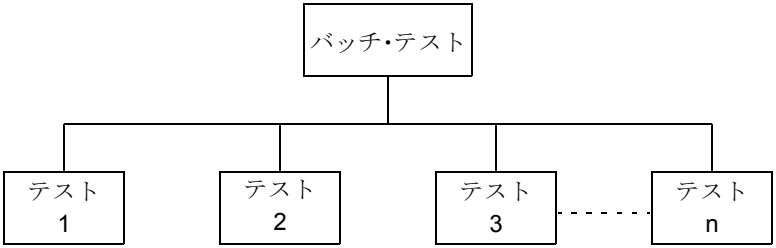
WinRunner では，一連のテストをまとめて無人で実行できます。この機能は特に，大規模なテストを夜間あるいは他の業務時間外に実行したい場合に便利です。

本章では，以下の項目を説明します。

- バッチ・テストの実行について
- バッチ・テストの作成
- バッチ・テストの実行
- バッチ・テストの結果の格納
- バッチ・テストの結果の表示

バッチ・テストの実行について

バッチ・テストを 1 つ作成して実行すれば，一連のテストをまとめて無人で実行できます。バッチ・テストは，他のテストを呼び出す call ステートメントを含んでいるテスト・スクリプトです。



バッチ・テストは，一見 call ステートメントを含んでいる通常のテストのように見えます。しかし，テストは実行の前に［一般オプション］ダイアログ・ボックスの［**実行**］カテゴリで［**バッチ モードで実行する**］オプションを選択した場合にのみ，「バッチ・テスト」となります。

テストをバッチ・モードで［**検証**］実行オプションを使用して実行するには，WinRunner はビットマップの不一致を報告するメッセージなど，通常のテスト実行の際に表示されるメッセージの表示を抑制します。また，WinRunner は，すべての **pause** ステートメントと，実行時エラーによる実行の停止も抑制します。

WinRunner は，すべてのメッセージを抑制することで，バッチ・テストを無人で実行できます。これに対し，通常の対話型のテスト実行の場合，メッセージが画面に表示されたらボタンをクリックしないとテストを再開できません。バッチ・テストにより，夜間や負荷の少ない時間帯にテストを実行できるため，アプリケーションのテスト期間を短縮できます。

テスト実行の際，ステップ・コマンドの後や停止ポイント，あるいはテストの終了時など，テスト実行中，ブレークごとに，［デバッグ ビューア］の［呼び出しチェーン］表示枠で現在の呼び出し先テストのチェーンを表示できます。詳細については，バッチ・テストの実行が完了した後，その結果を［テスト結果］ウィンドウで見ることができます。［テスト結果］ウィンドウには，実行の際に生じたすべての主要なイベントが表示されます。

一連のテストをまとめてコマンド行からも実行できます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

バッチ・テストの作成

バッチ・テストは，他のテストを呼び出すテスト・スクリプトです。バッチ・テストは，テスト・ウィンドウに call ステートメントを直接入力し，テストを実行する前に［一般オプション］ダイアログ・ボックスの［**実行**］カテゴリで［**バッチ モードで実行する**］オプションを選択することで作成します。

バッチ・テストには，ループや条件判断ステートメントなどのプログラミング要素を含めることができます。ループを使って，呼び出し先のテストを特定の回数だけ実行させることができます。*if/else* や *switch* などの条件判断ステートメントを使って，同じバッチ・スクリプト内でそれまでに呼び出されたテストの結果に基づいてテストを実行する条件を設定できます。詳細については，第 28 章「プログラミングによるテスト・スクリプトの 機能強化」を参照してください。

例えば，以下のバッチ・テストは，3 つのテストを連続して実行し，ループにより先頭に戻って，3 つのテストを再度呼び出します。ループの条件により，呼び出し先のテストは 10 回呼び出されます。

```
for (i=0; i<10; i++)
    {
    call "c:¥¥pbtests¥¥open" ();
    call "c:¥¥pbtests¥¥setup" ();
    call "c:¥¥pbtests¥¥save" ();
    }
```

バッチ・テストを実行するには，次の手順を実行します。

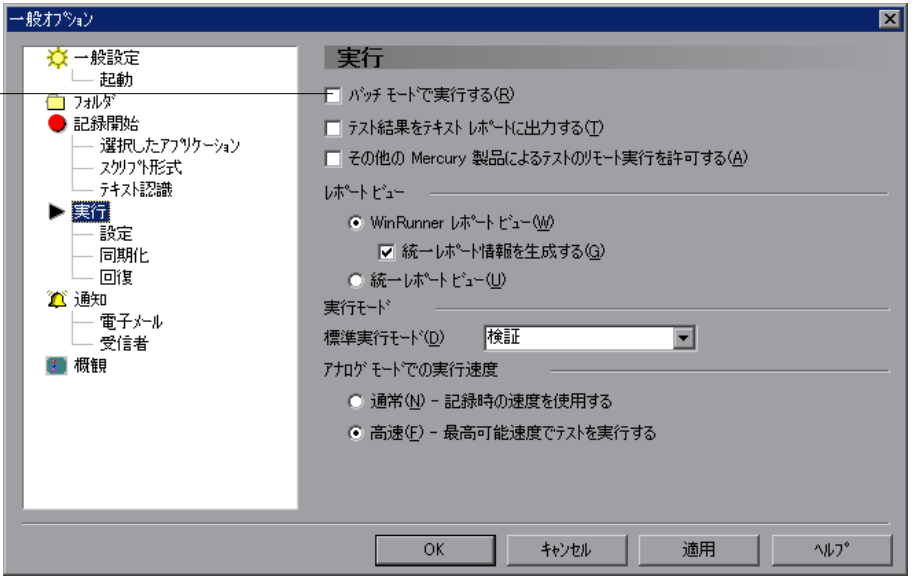
1 ［**ツール**］＞［**一般オプション**］を選択します。

［一般オプション］ダイアログ・ボックスが開きます。

2 ［**実行**］カテゴリをクリックします。

3 ［**バッチ モードで実行する**］チェック・ボックスを選択します。

［バッチ モードで実行する］チェック・ボックス



4 ［**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。

［一般オプション］ダイアログ・ボックスでのバッチ・オプションの設定の詳細については，第 22 章「グローバル・テスト・オプションの設定」を参照してください。

バッチ・テストの実行

バッチ・テストの実行は，通常のテストと同じように実行します。ツール・バー のリストからモード（[検証]，［更新]，［デバッグ]）を選択し，［**テスト**］＞［**先頭から実行**］を選択します。詳細については，第 19 章「テスト実行について」を参照してください。

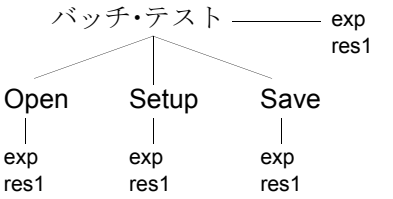
バッチ・テストを実行すると，WinRunner は各呼び出し先のテストを開いて実行します。テストが中断せずに実行されるように，メッセージはすべて抑制されます。バッチ・テストを検証モードで実行すると，現在のテスト結果が，先に保存した期待結果と比較されます。既存の期待結果を更新するためにバッチ・テストを実行すると，各テストの期待結果フォルダに新しい期待結果が作成されます。詳細については次，を参照してください。バッチ・テストの実行が完了したら，テスト結果を［テスト結果］ウィンドウで見ることができます。

テストに TSL の **texit** ステートメントが含まれている場合，WinRunner のこのステートメントの解釈は，バッチ・テスト実行と通常のテスト実行とで異なります。通常のテスト実行の場合，**texit** はテスト実行を終了します。バッチ・テスト実行の場合，**texit** は現在のテストの実行を停止するだけで，制御はバッチ・テストに戻されます。

バッチ・テストの結果の格納

通常の対話型のテストを実行すると，結果はテストのサブフォルダに格納されます。［一般オプション］ダイアログ・ボックスの［**実行**］カテゴリで［**バッチ モードで実行する**］が選択されている場合は，WinRunner は各（最上位の）呼び出し先のテストの結果をそれぞれのテストのサブフォルダに個別に保存します。バッチ・テスト用に，バッチ・テスト全体の結果（すべての呼び出し先テストを含む）を格納するサブフォルダも作成されます。

例えば，*Open*, *Setup*, および *Save* という 3 つのテストを作成したとします。各テストのテスト・ディレクトリの下の *exp* サブフォルダに，それぞれのテストの期待結果が保存されています。この 3 つのテストを呼び出すバッチ・テストを作成したとします。このバッチ・テストを検証モードで実行する前に，WinRunner に対して，結果を *res1* という名前の呼び出し先テストのサブフォルダに保存するよう指示します。バッチ・テストを実行すると，現在のテスト結果が，先に保存した期待結果と比較されます。WinRunner は，各テスト・フォルダの下に *res1* という名前のサブフォルダを作成し，そこにテストの検証結果を保存します。*res1* フォルダは，バッチ・テストのディレクトリの下にも作成され，実行全体の検証結果が格納されます。



期待結果を更新するためにバッチ・テストを［**更新**］モードで実行した場合，WinRunner は各テストと，バッチ・テストの下にある *exp* サブフォルダの期待結果を上書きします。

［バッチ モードで実行する］チェック・ボックス（［ツール］ ＞［一般オプション］ ＞［実行］）を選択せずにバッチ・テストを実行すると，WinRunner は結果をバッチ・テストの下のサブフォルダだけに保存します。その場合，呼び出し先テストを個別に実行することにしたときに，WinRunner は以前に保存した期待結果と検証結果がどこにあるかが分からないため，後で問題になることがあります。

バッチ・テストの結果の表示

バッチ・テストが完了した後，実行の際に生じたイベントに関する情報を［テスト結果］ウィンドウに表示できます。呼び出し先のテストがどれか 1 つ失敗すると，バッチ・テストは失敗と記されます。

［テスト結果］ウィンドウには，バッチ・テスト実行中に生じたすべてのイベントがリストされます。テストが呼び出されるたびに，*call_test* エントリがリストされます。第 22 章「グローバル・テスト・オプションの設定」［テスト結果］ウィンドウのテスト・ログ・セクションには，バッチ・テスト実行中に生じたすべてのイベントがリストされます。テストが呼び出されるたびに，*call_test* エントリがログにリストされます。呼び出し先のテストの結果を表示するには，対応する *call_test* エントリをダブルクリックします。［テスト結果］ウィンドウでテスト結果を表示する詳細については，第 2 章「統一レポート・ビューでのテスト結果の分析」を参照してください。

第 37 章

コマンドラインからのテストの実行

Windows のコマンドラインから直接テストを実行できます。

本章では，以下の項目について説明します。

- コマンドラインからのテストの実行について
- Windows のコマンドラインの使用法
- コマンドライン・オプション

コマンドラインからのテストの実行について

Windows の［名前を指定して実行］コマンドを使用して，WinRunner を起動し，あらかじめ定義したオプションに従ってテストを実行できます。また，ユーザ定義の WinRunner ショートカットを作成して，起動オプションを保存できます。ショートカットを作成すれば，ショートカット・アイコンをダブルクリックするだけで，起動オプションで WinRunner を起動できます。コマンドラインを使用して，次のことができます。

- アプリケーションの起動
- WinRunner の起動
- 関連付けたテストのロード
- テストの実行
- テスト・オプションの指定
- テストの結果ディレクトリの指定

WinRunner で設定できる機能オプションのほとんどは，コマンドラインからでも設定できます。これらの機能オプションには，テスト実行オプションや，テスト結果を格納するディレクトリの指定などが含まれます。

また，このような設定や他の環境変数，システム・パラメータを含むユーザ定義の「*.ini*」ファイルを指定できます。

例えば，以下のコマンドは WinRunner を起動して，バッチ・テストをロードし，テストを［**検証**］モードで実行します。

```
C:¥Program Files¥Mercury Interactive¥WinRunner¥WRUN.EXE -t
c:¥batch¥newclock -batch on -verify -run_minimized -dont_quit -run
```

最小化した WinRunner に「*newclock*」というテストがロードされ，バッチ・モードで実行されます。テスト実行が完了しても WinRunner は開いたままです。

注：AT コマンドを使用して（特に SU.EXE コマンド）を WinRunner で使用できます。AT コマンドは Microsoft Windows NT オペレーティング・システムに含まれています。AT コマンドに関する情報は，NT のリソース・キットに含まれます。このコマンドを使用することで，ユーザの介入なしに，完全に自動化されたスクリプトを実行できます。

Windows のコマンドラインの使用法

Windows のコマンドラインから，あらかじめ定義したオプションを指定して WinRunner を起動できます。WinRunner を起動するたびに同じオプションを使用する場合は，ユーザ定義の WinRunner ショートカットを作成するとよいでしょう。

コマンドラインからの WinRunner の起動

ここでは，WinRunner をコマンドラインから起動する手順を説明します。

［ファイル名を指定して実行］コマンドで WinRunner を起動するには，次の手順を実行します。

1 ［**スタート**］メニューから［**ファイル名を指定して実行**］を選択します。［**ファイル名を指定して実行**］ダイアログ・ボックスが開きます。

2 WinRunner の実行ファイルの *wrun.exe* のパスに続けて，使用するコマンドライン・オプションを入力します。

3 ［**OK**］をクリックしてダイアログ・ボックスを閉じると，WinRunner が起動します。

注：コマンド・ライン・オプションを，スペースが含まれるパスに追加する場合は，wrun.exe のパスを引用符で囲んで指定しなければなりません。次に例を示します。

"D:¥Program Files¥Mercury Interactive¥WinRunner¥arch¥wrun.exe" -addins WebTest

---

ユーザ定義の WinRunner ショートカットの追加

ユーザ定義の WinRunner ショートカットを作成して，指定したオプションを恒久的なものにできます。

ユーザ定義の WinRunner ショートカットを作成するには，次の手順を実行します。

1 Windows の［エクスプローラ］または［マイコンピュータ］で，wrun.exe ファイルのショートカットを作成します。

2 ショートカットの上でマウスの右ボタンをクリックして，［**プロパティ**］を選択します。

3 ［**ショートカット**］タブをクリックします。

4 ［**リンク先**］ボックスに，WinRunner の wrun.exe ファイルのパスと，使用するコマンドライン・オプションを入力します。

5 ［**OK**］をクリックします。

コマンドライン・オプション

以下では，各コマンドライン・オプションについて説明します。

-addins *ロードするアドインの一覧*

WinRunner に特定のアドインをロードさせます。リストで，アドインをカンマで区切ります（スペースは入れない）。このコマンドラインは，**-addins_select_timeout** コマンドライン・オプションと併せて使用できます。

（前のバージョンでは **-addons**）

注：インストールされたアドインはすべて，次のように，レジストリにリストされます。
HKEY_LOCAL_MACHINE¥SOFTWARE¥Mercury Interactive¥WinRunner¥CurrentVersion¥Installed Components¥

ロードするアドインを指定するときは，このブランチの下に表示されるキー名をそのまま入力します。指定したアドイン名の大文字・小文字は区別されません。

例えば次の行は，WinRunner に含まれている 4 つのアドインをロードします。

<WinRunner のインストールフォルダ >¥arch¥wrun.exe -addins ActiveX,pb,vb,WebTest

---

-addins_select_timeout タイムアウト

WinRunner を起動してから［**アドイン マネージャ**］ダイアログ・ボックスを閉じるまで，指定した時間（秒）WinRunner を待機させます。タイムアウトが 0（ゼロ）の場合，ダイアログ・ボックスは表示されません。このコマンドラインは，

-addins コマンドライン・オプションと併せて使用できます。

（前のバージョンでは **-addons_select_timeout**）

-animate

実行矢印が実行中のテストの行を表示している間に，ロードしたテストを WinRunner に実行させます。

-app *path*

WinRunner を実行する前に指定したアプリケーションを実行します。これは，**-app_params, -app_open_win,** および **-WR_wait_time** コマンドライン・オプションと組み合わせて使用することができます。

［テストのプロパティ］の［実行］タブでも起動アプリケーションを定義することができます。詳細については，第 21 章「単独のテストのプロパティの設定」を参照してください。

-app_params *param1[,param2,...,paramN]*

-app で指定されたアプリケーションに指定されたパラメータを渡します。

注：このコマンドライン・オプションは，**-app** コマンドライン・オプションも使用している場合のみ使用できます。

-app_open_win *setting*

アプリケーション・ウィンドウが開いたときの外観を決定します。

次の表に，*setting* に指定可能な値を示します。

オプション	説明
SW_HIDE	ウィンドウを非表示にし，別のウィンドウをアクティブにします。
SW_SHOWNORMAL	ウィンドウをアクティブにして表示します。ウィンドウが最小化または最大化されている場合は，Windows はオリジナルのサイズと位置に戻します。初めてウィンドウを表示したときに，このフラグを指定します。
SW_SHOWMINIMIZED	ウィンドウをアクティブにして，最小化ウィンドウとして表示します。
SW_SHOWMAXIMIZED	ウィンドウをアクティブにして，最大化ウィンドウとして表示します。
SW_SHOWNOACTIVATE	ウィンドウを最新のサイズと位置に表示します。アクティブなウィンドウはアクティブなままになります。

オプション	説明
SW_SHOW	ウィンドウをアクティブにして，現在のサイズと位置に表示します。
SW_MINIMIZE	指定したウィンドウを最大化して，次の最上位レベルのウィンドウを手前から順番にアクティブにします。
SW_SHOWMINNOACTIVE	ウィンドウを最小化ウィドウとして表示します。アクティブ・ウィドウはアクティブなままになります。
SW_SHOWNA	ウィンドウを現在の状態で表示します。アクティブ・ウィンドウはアクティブなままになります。
SW_RESTORE	ウィンドウをアクティブにして表示します。ウィンドウが最小化または最大化されると，Windows はオリジナルのサイズと位置に戻します。最小化ウィンドウに戻す場合にこのフラグを指定します。

注：このコマンドライン・オプションは，**-app** コマンドライン・オプションも使用している場合のみ使用できます。

-auto_load {on | off}

仮 GUI マップ・ファイルの自動ロードを有効または無効にします。

（標準設定 = **on**）

-auto_load_dir *パス*

仮 GUI マップ・ファイル（*temp.gui*）が格納されているフォルダを指定します。このオプションは auto_load が on の場合にのみ有効です。

（標準設定 = **M_Home¥dat**）

-batch {on | off}

ロードしたテストをバッチ・モードで実行します。

（標準設定 = **off**）

［一般オプション］ダイアログ・ボックスの［**実行**］カテゴリにある［**バッチモードで実行**］チェック・ボックスを使ってこのオプションを設定することもできます。詳細については，545 ページ「テストの実行オプションの設定」を参照してください。

batch テスト・オプションの値は，**getvar** 関数を使ってテスト・スクリプトから取得できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

ヒント：エラー・メッセージが表示されてテスト実行が一時停止しないよう，**-batch** オプションを **-verify** オプションと組み合わせて使用します。**-verify** オプションの詳細については，804 ページを参照してください。

-beep {on | off}

WinRunner システムによるビープ音を有効または無効にします。

（標準設定 = **on**）

［一般オプション］ダイアログ・ボックスの［**実行**］＞［**設定**］カテゴリにある［**ウィンドウのチェック時にビープ音を鳴らす**］チェック・ボックスを使ってこのオプションを設定することもできます。詳細については，549 ページ「実行の設定オプションの設定」を参照してください。

テスト・スクリプトで **setvar** 関数と **getvar** 関数を使って，**beep** テスト・オプションの値を設定したり，取得したりできます。これについては，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

-capture_bitmap {on | off}

チェックポイントが失敗するたびに WinRunner がビットマップをキャプチャするかどうかを決定します。このオプションが on （1）の場合，WinRunner は［一般オプション］ダイアログ・ボックスの［実行］＞［設定］カテゴリから設定を使用して，ビットマップのキャプチャ領域を決定します。

（標準設定 = **off**）

このオプションは［一般オプション］ダイアログ・ボックスの［**実行**］＞［**設定**］カテゴリの［**検証失敗の時，ビットマップをキャプチャする**］チェックボックスを使用して設定することもできます。詳細については，549 ページ「実行の設定オプションの設定」を参照してください。

setvar および **getvar** 関数を使用して *capture_bitmap* テスト・オプションに対応するの値をテスト・スクリプト内から取得することができます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

-create_text_report {on | off}

WinRunner に，テスト結果を「*report.txt*」というテキスト・レポートに書き込ませます。このファイルは結果フォルダに保存されます。

-create_unirep_info {on | off}

統一レポートを作成するのに必要な情報を生成し（WinRunner レポート・ビューが選択されている場合），後でテストの統一レポートを表示できるようにします。

（標準設定 = **on**）

このオプションは，［一般オプション］ダイアログ・ボックスの［実行］カテゴリにある［**統一レポート情報を生成する**］オプションを使用して設定することもできます。詳細については，549 ページ「実行の設定オプションの設定」を参照してください。

-cs_fail {on | off}

コンテキスト・センシティブ・エラーが起きたときに WinRunner がテストを失敗したとするかどうかを指定します。コンテキスト・センシティブ・エラーとは，テスト中のコンテキスト・センシティブ・ステートメントの失敗です。コンテキスト・センシティブのエラーのほとんどは，WinRunner が GUI オブジェクトの特定に失敗したのが原因です。

例えば，コンテキスト・センシティブ・エラーは，存在しないウィンドウ名が指定された **set_window** ステートメントを含むテストを実行した場合に起きます。また，ウィンドウ名が一意でない場合にも，コンテキスト・センシティブ・エラーが発生することがあります。コンテキスト・センシティブ関数についての詳細は，「**TSL リファレンス**」を参照してください。

（標準設定 = **off**）

［一般オプション］ダイアログ・ボックスの［実行］タブの［**コンテキストセンシティブエラーが発生したらテストを失敗とする**］チェック・ボックスを使ってこのオプションを設定することもできます。詳細については，549 ページ「実行の設定オプションの設定」を参照してください。

テスト・スクリプトで **setvar** 関数と **getvar** 関数を使って，**cs_fail** テスト・オプションの値を設定したり，取得したりできます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

-cs_run_delay *正の整数*

テスト実行中に，WinRunner がコンテキスト・センシティブ・ステートメントの実行を待機する間隔（単位：ミリ秒）を指定します。

（標準設定 = **0**［ミリ秒］）

［一般オプション］ダイアログ・ボックスの［**実行**］>［**同期化**］カテゴリにある［**CS ステートメント実行間の遅延**］ボックスを使ってこのオプションを設定することもできます。これについては，554 ページ「実行の同期化オプションの設定」を参照してください。

setvar 関数と **getvar** 関数を使って，**cs_run_delay** テスト・オプションに対する値をテスト・スクリプトから設定し，取得できます。これについては，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

-def_replay_mode {verify | debug | update}

すべてのテストに標準設定で使用できる実行モードを設定します。

注：「**検証**」モードはテストの実行時のみ使用できます（コンポーネントの実行時には使用できません）。コンポーネントを使って作業する場合は，コンポーネントが Quality Center でビジネス・プロセス・テストの一部として実行されるときに，アプリケーションが検証されなければなりません。

指定可能な値：

- **Update**：テストの期待結果を更新する，または新しい期待結果フォルダを作成する場合に使用します。
- **Verify**：アプリケーションの検査に使用します。

➤ **Debug**：テスト・スクリプトでバグを特定するのに使用します。

（標準設定 = **Verify**）

このオプションは，［一般オプション］ダイアログ・ボックスの［**実行**］カテゴリにある［**標準実行モード**］オプションを使用して設定することもできます。詳細については，545 ページ「テストの実行オプションの設定」を参照してください。

-delay *正の整数*

ウィンドウ / オブジェクトをビットマップ・チェックポイントや同期化ポイントにキャプチャする前に，そのウィンドウ / オブジェクトが安定しているかどうかを WinRunner に判定させます。WinRunner が続けて画面をサンプリングする間隔（単位：ミリ秒）を指定します。連続した 2 回の検査結果が一致する場合，WinRunner はウィンドウ / オブジェクトをキャプチャします（前のバージョンでは **-delay** は秒単位で計測していました）。

（標準設定 = **1000**［ミリ秒］）

（前のバージョンでは **-delay** です）

注：このパラメータの誤差範囲は 20 ～ 30 ミリ秒です。

［一般オプション］ダイアログ・ボックスの［**実行**］＞［**同期化**］カテゴリにある［**ウィンドウ同期までの遅延**］ボックスを使ってこのオプションを設定することもできます。これについては，554 ページ「実行の同期化オプションの設定」を参照してください。

テスト・スクリプトで **setvar** 関数と **getvar** 関数を使って，**delay_msec** テスト・オプションの値を設定したり，取得したりできます。これについては，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

-dont_connect

［Quality Center に接続］ダイアログ・ボックスで［**起動時に再接続する**］オプションが選択されている場合に，Quality Center に接続せずに WinRunner を開くことができます。

［**起動時に再接続する**］オプションを無効にするには，［**ツール**］＞［**Quality Center への接続**］を選択して［**起動時に再接続する**］チェック・ボックスのチェックを外します。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

-dont_quit

テストの完了後に WinRunner がテストを閉じないようにします。

-dont_show_welcome

WinRunner 起動時に［ようこそ］ウィンドウが表示されないようにします。

-email_service

WinRunner が電子メール送信オプション（チェックポイントの失敗，テストの失敗，テストの完了レポートのメール通知，およびテスト内の **email_send_msg** ステートメントを含む）をアクティブにするかどうかを決定します。

（標準設定 = **off**）

このオプションは，［一般オプション］ダイアログ・ボックスの［**通知**］＞［**電子メール**］カテゴリで［**電子メールのサービスを有効にする**］を使用して設定することもできます。詳細については，563 ページ「電子メール通知オプションの設定」を参照してください。

setvar および **getvar** 関数を使用して，**email_service** テスト・オプションの値ををテスト・スクリプト内から設定し取得することができます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

-exp *期待結果フォルダ名*

期待結果を格納するサブフォルダの名前を指定します。検証実行の場合には，検証比較のベースとなる期待結果のセットを格納するサブフォルダを指定します。

（標準設定 = **exp**）

［テストのプロパティ］ダイアログ・ボックスの［**現在のテスト**］タブにある［**期待結果フォルダ**］ボックスを使ってこのオプションを設定することもできます。これについては，508 ページ「現在のテスト設定の確認」を参照してください。

getvar 関数を使って，**exp** テスト・オプションの値をテスト・スクリプトから取得できます。これについては，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

-fast_replay {on | off}

アナログ・モードで記録されたテストのテスト実行の速さを設定します。**on** に設定するとテストが最高の速さで実行され，**off** に設定すると，記録された速さでテストが実行されます。

アナログ実行の速度は，［一般オプション］ダイアログ・ボックスの［実行］カテゴリにある［アナログ モードでの実行速度］オプションを使用しても指定できます。詳細については，545 ページ「テストの実行オプションの設定」を参照してください。

（標準設定 = **on**）

-f *ファイル名*

コマンドライン・オプションを含むテキスト・ファイルを指定します。ファイルのオプションの指定は，1 行に並べて，あるいは複数の行に分けて指定できます。このオプションを使用して，Windows の［プロパティ］ダイアログ・ボックスの［ショートカット］タブにある［リンク先］ボックスに入力できる文字数の制限を回避できます。

注 : 同じコマンドライン・オプションがコマンドラインとファイルの両方にある場合，WinRunner はファイルに指定されているオプションの設定を優先させます。

-fontgrp *グループ名*

WinRunner 起動時に有効になるフォント・グループを指定します。

［一般オプション］ダイアログ・ボックスの［記録］＞［テキスト認識］カテゴリにある［**フォント グループ**］ボックスを使ってこのオプションを設定することもできます。詳細については，541 ページ「テキスト認識オプションの設定」を参照してください。

テスト・スクリプトで **setvar** 関数と **getvar** 関数を使って，**fontgrp** テスト・オプションの値を設定したり，取得したりできます。これについては，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

-ini *パス名* wrun.ini ファイル

WinRunner 起動時に使用する **wrun.ini** ファイルを定義します。このファイルは，**-update_ini** コマンドライン・オプションと一緒に使用しない限り，読み取り専用です。

-min_diff *正の整数*

イメージの不一致が起きるしきい値をピクセルの数で定義します。

(標準設定 = **0**［ピクセル］です)。

［一般オプション］ダイアログ・ボックスの［**実行**］＞［**設定**］カテゴリにある［**ビットマップ間の違いを差異として認識するしきい値**］ボックスを使ってこのオプションを設定することもできます。詳細については，541 ページ「テキスト認識オプションの設定」を参照してください。

テスト・スクリプトで **setvar** 関数と **getvar** 関数を使って，**min_diff** テスト・オプションの値を設定したり，取得したりできます。第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

-mismatch_break {on | off}

検証実行の前に［検証が失敗したら停止する］を有効または無効にします。［検証が失敗したら停止する］の動作は，テストを対話形式で実行しているときとは異なります。対話形式で実行している場合には，テストが停止します。コマンドラインから開始したテストの場合，最初の比較の不一致でテスト実行が終了します。

［検証が失敗したら停止する］は，［検証］モードでテストを実行中に，検証が失敗したら WinRunner がテストの実行を一時停止してメッセージを表示するか，コンテキスト・センシティブ・ステートメントの結果として任意のメッセージを生成するかを指定します。

例えば，テスト・スクリプトに **set_window** ステートメントがない場合，WinRunner は指定されたウィンドウを見つけられません。このオプションが on の場合，WinRunner はテストを一時停止して［実行］ウィザードを開いて，ユーザがウィンドウを指定できるようにします。このオプションが off の場合，WinRunner は［テスト結果］ウィンドウでエラーを報告して，テスト・スクリプトの次のステートメントの実行を続行します。

（標準設定 = **on**）

このオプションは，［一般オプション］ダイアログ・ボックスの［**実行**］＞［**設定**］カテゴリにある［**検証が失敗したら停止する**］ボックスを使って設定することもできます。これについては，549 ページ「実行の設定オプションの設定」を参照してください。

テスト・スクリプトで **setvar** 関数と **getvar** 関数を使って，**mismatch_break テスト・オプションの値を**設定したり，取得したりできます。これについては，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

-qc_connection {on | off}

on の場合に，WinRunner の Quality Center への接続を有効にします。

（標準設定 = **off**）

（以前の **-td_connection** または **-test_director**）

［Quality Center に接続］ダイアログ・ボックスから Quality Center に接続できます。ダイアログ・ボックスは［**ツール**］＞［**Quality Center への接続**］を選択して開きます。Quality Center 接続の詳細については，第 48 章「テスト工程の管理」を参照してください。

注：［Quality Center に接続］ダイアログ・ボックスで［起動時に再接続する］オプションを選択している場合は，**qc_connection** を off に設定しても Quality Center への接続は妨げられません。このような場合に Quality Center に接続されないようにするには，**-dont_connect** コマンドを使用します。詳細については，794 ページ「-dont_connect」を参照してください。

-qc_cycle_name *cycle name*

現在のテスト・サイクルの名前を指定します。このオプションは，WinRunner が Quality Center に接続されている場合にのみ有効です。

qc_cycle_name テスト・オプションを使って，現在のテスト・サイクルの名前を指定できます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

（以前の **-td_cycle_name** または **-cycle**）

-qc_database_name *database path*

アクティブな Quality Center データベースを指定します。WinRunner はデータベース内のテストを開いたり，実行したり，保存したりできます。このオプションは，WinRunner が Quality Center に接続されている場合にのみ有効です。

このオプションを使用する時には次の構文を使用します。

<database_name>.<domain>

次に例を示します。

Mercury.Wrun

qc_database_name テスト・オプションを使用して，アクティブな Quality Center データベースを使用することもできます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

WinRunner を TestDirector に接続する際，［**Quality Center への接続**］ダイアログ・ボックスからアクティブな Quality Center プロジェクト・データベースを指定できます。ダイアログ・ボックスは［**ツール**］＞［**Quality Center への接続**］を選択して開きます。詳細については，第 48 章「テスト工程の管理」を参照してください。

（以前の **-td_database_name** または **-database**）

-qc_password *password*

Quality Center サーバのデータベースに接続するためのパスワードを指定します。

［Quality Center に接続］ダイアログ・ボックスから Quality Center に接続するためのパスワードを指定できます。ダイアログ・ボックスは［**ツール**］＞［**Quality Center への接続**］を選択して開きます。Quality Center 接続の詳細については，第 48 章「テスト工程の管理」を参照してください。

（以前の **-td_password**）

-qc_server_name *server name*

WinRunner を接続する Quality Center サーバ名を指定します。

qc_server_name テスト・オプションを使用して，WinRunner が接続する Quality Center サーバの名前を指定します。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

サーバに接続するためには **qc_connection** オプションを使用します。

（以前の **-td_server_name** または **-td_server**）

-qc_user_name *ユーザ名*

テスト・サイクルを実行している現在のユーザ名を指定します。

qc_user_name テスト・オプションを使用して，ユーザを指定することもできます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

［**Quality Center に接続**］ダイアログ・ボックスから Quality Center に接続するためのパスワードを指定できます。ダイアログ・ボックスは［**ツール**］＞［**Quality Center への接続**］を選択して開きます。Quality Center 接続の詳細については，第 48 章「テスト工程の管理」を参照してください。

（以前の **-td_user_name**, **-user_name,** または **-user**）

-rec_item_name {0 | 1}

WinRunner が，一意でないリスト・ボックスやコンボ・ボックスの項目を名前で記録するかインデックスで記録するかを指定します。

（標準設定 = **0**）

［一般オプション］ダイアログ・ボックスの［記録開始］カテゴリにある［**一意ではないリストの項目を名前に基づいて記録する**］チェック・ボックスを使ってこのオプションを設定することもできます。これについては，529 ページ「記録オプションの設定」を参照してください。

テスト・スクリプトで **setvar** 関数と **getvar** 関数を使って，**rec_item_name** テスト・オプションの値を設定したり，取得したりできます。これについては，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

-run

ロードしたテストを WinRunner に実行させます。WinRunner ウィンドウにテストをロードするには，***-t*** コマンドライン・オプションを使います。

-run_minimized

WinRunner を起動し WinRunner でテストを実行して，テストをアイコンに最小化します。このオプションだけを指定してもテストは実行されません。***-t*** コマンドライン・オプションを使用してテストをロードし，***-run*** コマンドライン・オプションを使ってロードしたテストを実行します。

-search_path *パス*

開いたり呼び出したりするテストを検索するディレクトリを定義します。検索パスは文字列として指定します

（標準設定 = **起動フォルダ** と，**インストール・フォルダ**の下の **lib** サブフォルダです）。

［一般オプション］ダイアログ・ボックスの［フォルダ］カテゴリにある［**呼び出し先のテストの検索パス**］ボックスを使ってこのオプションを設定することもできます。これについては，526 ページ「フォルダ・オプションの設定」を参照してください。

テスト・スクリプトで **setvar** 関数と **getvar** 関数を使って，**searchpath** テスト・オプションの値を設定したり，取得したりできます。これについては，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

-single_prop_check_fail {on | off}

_check_info ステートメントが失敗するとテストの実行が失敗となり，失敗したステートメントに対するイベントを［テスト結果］ウィンドウに書き込みます（［**挿入**］＞［**GUI チェックポイント**］＞［**単数プロパティ**］コマンドを使って **_check_info** ステートメントを作成できます。

setvar 関数と **getvar** 関数でこのオプションを使用できます。

（標準設定 = **on**）

check_info 関数の詳細については，「**TSL リファレンス**」を参照してください。

［一般オプション］ダイアログ・ボックスの［**実行**］＞［**設定**］タブにある［**単数のプロパティチェックが失敗したらテストを失敗とする**］チェック・ボックスを使ってこのオプションを設定することもできます。これについては，526 ページ「フォルダ・オプションの設定」を参照してください。

テスト・スクリプトで **setvar** 関数と **getvar** 関数を使って，**single_prop_check_fail** テスト・オプションの値を設定したり，取得したりできます。これについては，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

-speed {normal | fast}

ロードしたテストの実行速度を設定します。

（標準設定 = **fast**）

［一般オプション］ダイアログ・ボックスの［**実行**］カテゴリにある［**アナログモードでの実行速度**］オプションを使ってこのオプションを設定することもできます。これについては，526 ページ「フォルダ・オプションの設定」を参照してください。

テスト・スクリプトで **setvar** 関数と **getvar** 関数を使って，**speed** テスト・オプションの値を設定したり，取得したりできます。これについては，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

（前のバージョンでは **-run_speed**）

-t *テスト名*

WinRunner ウィンドウにロードするテストの名前を指定します。検索パスで指定されたフォルダに格納されているテストの名前か，システム内に格納されている任意のテストの完全パス名を指定できます。

timeout_msec *正の整数*

WinRunner がチェックポイントやコンテキスト・センシティブ・ステートメントを実行するときに使用するグローバル・タイムアウト（単位：ミリ秒）を設定します。この値は，GUI チェックポイントまたは同期化ポイント・ステートメントに含まれる *time* パラメータに追加され，WinRunner が指定されたウィンドウまたはオブジェクトを検索する最大時間が決定されます（前のバージョンでは *timeout* で，秒単位でした）。

（標準設定 = **10,000** ［ミリ秒］）

（前のバージョンでは **-timeout**）

注：このオプションの誤差範囲は 20 ～ 30 ミリ秒です。

［一般オプション］ダイアログ・ボックスの［実行］＞［設定］カテゴリにある［**チェックポイントと CS ステートメントのタイムアウト**］ボックスを使ってこのオプションを設定することもできます。これについては，549 ページ「実行の設定オプションの設定」を参照してください。

setvar 関数と **getvar** 関数を使って，**timeout_msec** テスト・オプションに対する値をテスト・スクリプトから設定し，取得できます。これについては，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

-tslinit_exp *期待結果フォルダ*

tslinit スクリプトの実行中に，WinRunner で期待フォルダを使用します。

-update_ini

wrun.ini ファイルを **-ini** コマンドライン・オプションで指定する際，WinRunner のセッションで加えられた構成設定の変更を保存します。

注：-ini コマンドライン・オプションを使用する際も，このコマンドライン・オプションだけが使用できます。

-verify *検証結果フォルダ名*

テストを検証モードで実行するように指定し，テスト結果を格納するサブフォルダの名前を指定します。

-WR_wait_time *正の整数*

アプリケーションを起動して，WinRunner を起動する間に待機する時間をミリ秒単位で指定します。

（標準設定 = **0** [ミリ秒]）

［テストのプロパティ］ダイアログ・ボックスの［**実行**］タブの［**テスト実行の前にアプリケーションを実行する**］ボックスを使用して設定することもできます。詳細については，第 21 章「単独のテストのプロパティの設定」第 21 章「単独のテストのプロパティの設定」を参照してください。

注：このコマンドライン・オプションは，**-app** コマンドライン・オプションも使用している場合のみ使用できます。

第 10 部

テストのデバッグ

第 38 章

テスト実行の制御

テスト実行を制御することで，テスト・スクリプト内の不具合を特定し，修正できます。

本章では，以下の項目について説明します。

- テスト実行の制御について
- テスト・スクリプトを 1 行だけ実行する方法
- テスト・スクリプトの一部を実行する方法
- テスト実行の一時停止

テスト実行の制御について

テスト・スクリプトを作成したら，構文やロジックに誤りがなく，スムーズに実行できるかどうかを確かめなければなりません。スクリプト中の不具合を検出するには，［ステップ］コマンドと［一時停止］コマンドを使って，テスト実行を制御します。

以下のステップ・コマンドがあります。

- ［ステップ］コマンドは，テスト・スクリプトを 1 行だけ実行します。
- ［ステップイントゥ］コマンドは，別のテストまたはユーザ定義関数を呼び出して，表示します。
- ［ステップアウト］コマンドは，［ステップイントゥ］コマンドと組み合わせて使います。呼び出し先のテストまたはユーザ定義関数を最後まで実行します。
- ［カーソル行にステップ］コマンドは，テスト・スクリプトの特定の部分を実行します。

［一時停止］コマンドあるいは **pause** 関数を使って，テスト実行を一時的に停止することもできます。

また，ブレークポイントを設定してテスト実行を制御することも可能です。ブレークポイントは，あらかじめ定義した箇所でテスト実行を一時停止し，アプリケーションのテストの効果を検査できます。［デバッグ ビューア］の［ブレークポイント リスト］表示枠ですべてのブレークポイントを表示できます。詳細については，第 39 章「ブレークポイントの使用」を参照してください。

テストをデバッグしやすいように，WinRunner ではテスト・スクリプトの変数を監視できます。監視する変数を監視リストに定義します。テストを実行すると，変数に代入された値が表示されます。［デバッグ ビューア］の［ウォッチ リスト］で監視対象変数の現在の値を表示できます。詳細については，第 40 章「変数の監視」を参照してください。

呼び出しチェーンを使用して，テスト・フローを追跡し移動できます。テスト実行中，ブレークごとに（［ステップ実行］コマンドの後，ブレークポイント，テストの最後などで），［デバッグ ビューア］の［呼び出しチェーン］表示枠で呼び出し先テストの現在のチェーンと関数を表示できます。詳細については，第 29 章「テストの呼び出し」を参照してください。

テスト・スクリプトをデバッグする際には，テストをデバッグ・モードで実行します。テストの結果は *debug* フォルダに保存されます。テストを実行するたびに，以前のデバッグ結果は上書きされます。検証モードでテストを実行する準備が整うまでは，デバッグ・モードでテストを実行してください。デバッグ・モードの詳細については，第 19 章「テスト実行について」を参照してください。

テスト・スクリプトを 1 行だけ実行する方法

［ステップ］，［ステップイントゥ］，および［ステップアウト］コマンドを使って，テスト・スクリプトの単独の行を実行できます。

［ステップ］コマンド

［**ステップ**］コマンドを選択するか，［**ステップ実行**］ボタンをクリックして，アクティブなテスト・スクリプトの現在の行（実行矢印によって示される行）だけを実行します。

現在の行が別のテストまたはユーザ定義関数を呼び出す場合は，呼び出し先のテストまたは関数がまるごと実行されますが，呼び出し先のテスト・スクリプトは［WinRunner］ウィンドウには表示されません。起動アプリケーションまたは起動関数を使用している場合は，それらも実行されます。

［ステップ イントゥ］コマンド

［**ステップ イントゥ**］コマンドを選択するか，［**ステップ イントゥ**］ボタンをクリックして，アクティブなテスト・スクリプトの現在の行だけを実行します。ただし，［ステップ］コマンドとは異なり，現在の行が別のテストまたはコンパイルしたモードでユーザ定義関数を呼び出していると，以下のことが行われます。

- 呼び出し先のテストまたは関数のテスト・スクリプトが［WinRunner］ウィンドウに表示されます。
- 起動アプリケーションおよび関数設定（［テストのプロパティ］ダイアログ・ボックスの［**実行**］タブ）は実装されません。
- テスト実行を継続するには，［ステップ］または［ステップアウト］コマンドを使います。

［ステップ アウト］コマンド

［**ステップ アウト**］コマンドは，［**ステップ イントゥ**］コマンドを使ってテストまたはユーザ定義関数に入った場合にのみ使います。［ステップ アウト］コマンドは呼び出し先のテストまたはユーザ定義関数を最後まで実行し，呼び出し元のテストに戻って，テストの実行を一時的に停止します。

テスト・スクリプトの一部を実行する方法

［カーソル位置まで実行］コマンドを使って，テスト・スクリプトの特定の部分を実行することができます。

［カーソル位置まで実行］コマンドを使うには，次の手順を実行します。

1 テスト・スクリプト内で，テストの実行を開始したい行に実行矢印を移動します。実行矢印を移動するには，マージンでテスト・スクリプト内の希望の行の横をクリックします。

2 テスト・スクリプトの中で，テスト実行を終了する行でマウスをクリックして，カーソルをそこに移動します。

3 ［**デバッグ**］＞［**カーソル位置まで実行**］を選択するか，STEP TO CURSOR ソフトキーを押します。WinRunner は挿入ポイントで示された行までテストを実行します。

テスト実行の一時停止

［一時停止］コマンドを選択するか，テスト・スクリプトに **pause** ステートメントを追加して，テスト実行を一時的に停止できます。

［一時停止］コマンド

［**テスト**］＞［**一時停止**］コマンドを選択するか，［**一時停止**］ボタンあるい［一時停止］ソフトキーを押して，テストの実行を一時的に停止できます。一時停止したテストは，それまでに解釈された TSL ステートメントがすべて実行されると実行を停止します。［停止］コマンドとは異なり，［一時停止］コマンドでは，テストの変数や配列は初期化されません。

一時停止したテストの実行を再開するには，［テスト］メニューから適切な実行コマンドを選択します。テストは，［一時停止］コマンドを実行した場所，または，テストが一時停止している間に実行矢印を移動した場合はその場所から実行を再開します。

pause 関数

WinRunner は，テスト・スクリプトで **pause** ステートメントを処理すると，テスト実行が一時停止し，メッセージ・ボックスが表示されます。**pause** ステートメントに式が含まれている場合，式の結果がメッセージ・ボックスに表示されます。**pause** 関数の構文は，次のとおりです。

pause ([*expression*]**);**

次の例では，**pause** はテスト実行を一時停止し，2 点の間で経過した時間を表示します。

```
t1=get_time();
t2=get_time();
pause ("Time elapsed" is & t2-t1);
```

注：WinRunner は，テストをバッチ・モードで実行中は **pause** ステートメントを無視します。

---

pause 関数の詳細については，「**TSL リファレンス**」を参照してください。

第 39 章

ブレークポイントの使用

ブレークポイントは，テスト・スクリプトの中で，テスト実行を一時停止する場所を示します。ブレークポイントを使えば，スクリプトの不具合を特定しやすくなります。

本章では，以下の項目について説明します。

- ブレークポイントの使用について
- ブレークポイントのタイプの選択
- 「停止位置」ブレークポイントの設定
- 「関数で停止」ブレークポイントの設定
- ブレークポイントの変更
- ブレークポイントの削除

ブレークポイントの使用について

ブレークポイントを設定することで，テスト・スクリプトの特定の場所でテスト実行を停止できます。ブレークポイントは，テスト・ウィンドウの左マージンで，ブレークポイント・マーカによって示されます。

WinRunner は，ブレークポイントに達するとテスト実行を一時停止します。その場所まで実行したテストの影響を調べたり，変数の現在値を表示したり，必要な変更を行ったりした後，そのブレークポイントからテストを再開できます。テスト実行を再開するには，［**矢印から実行**］コマンドを使います。WinRunner はテストが再開されると，次のブレークポイントに達するか，テストが完了するまで実行を続けます。

注：WinRunner はバッチ・モードで実行中の場合以外に一時停止します。バッチ・モードでテストを実行中は，WinRunner はブレークポイントを無視します。

ブレークポイントは次の場合に役立ちます。

- テストの実行を中断して，アプリケーションの状態を検査する。
- 監視リストのエントリを監視する。詳細については，第 40 章「変数の監視」を参照してください。
- ステップ・コマンドを使ってテストを 1 行ずつ実行する。詳細については，第 38 章「テスト実行の制御」を参照してください。

ブレークポイントは 2 種類あります。1 つは，「停止位置」ブレークポイントで，もう 1 つは，「関数で停止」ブレークポイントです。「停止位置」ブレークポイントは，テスト・スクリプトにおいて，指定された行番号でテストを停止します。「関数で停止」ブレークポイントは，テストがロードされているコンパイル済みモジュールの指定されたユーザ定義関数を呼び出すと停止します。

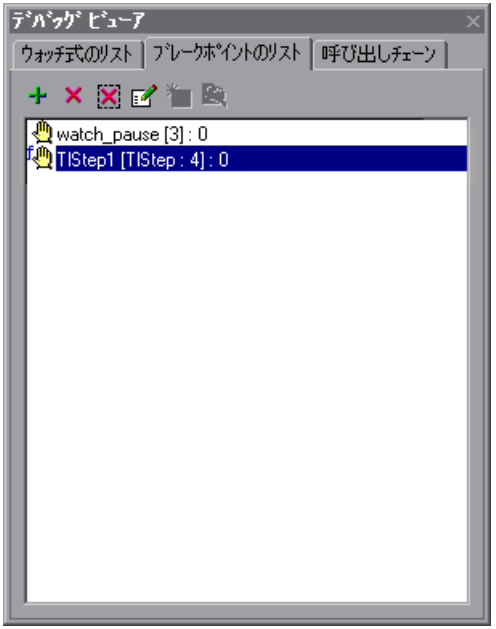
定義するブレークポイントごとにパス数を設定します。テストは，パス数に設定された回数だけブレークポイントを通過してから停止します。例えば，コマンドを 50 回実行するループを作成したとします。標準では，パス数がゼロに設定されているため，テスト実行はループを 1 回実行するごとに停止します。しかし，パス数を 25 に設定すると，テスト実行は，ループを 25 回実行したところで停止します。

注：定義したブレークポイントは，現在の WinRunner セッションにおいてのみ有効です。WinRunner セッションを終了して，別のセッションでスクリプトのデバッグを続けるには，ブレークポイントを再定義しなければなりません。

［デバッグ ビューア］の［ブレークポイントのリスト］の表示

［デバッグ ビューア］ウィンドウの［ブレークポイントのリスト］表示枠で変数の値を表示します。［デバッグ ビューア］ウィンドウが現在表示されていない場合，または［ブレークポイントのリスト］表示枠がウィンドウで開いてい

ない場合は，［**デバッグ**］＞［**ブレークポイントのリスト**］を選択して表示します。［ブレークポイントのリスト］表示枠は開いているけれども，現在別の表示枠が表示されている場合は，［**ブレークポイントのリスト**］タブをクリックして［ブレークポイントのリスト］表示枠を表示します。



ヒント：［デバッグ ビューア］ウィンドウは，WinRunner ウィンドウに固定されたウィンドウとして表示することも，画面上の希望の場所にドラッグできるフローティング・ウィンドウとして表示することもできます。標準設定では，［デバッグ ビューア］は WinRunner 画面の左側に固定されたウィンドウとして表示されます。［デバッグ ビューア］ウィンドウを別の場所に移動するには，［デバッグ ビューア］のタイトルバーをドラッグします。

ブレークポイントのタイプの選択

WinRunner では，2 種類のブレークポイントが使用できます。1 つは，「停止位置」ブレークポイントで，もう 1 つは，「関数で停止」ブレークポイントです。

停止位置

「停止位置」ブレークポイントは，テスト名とテスト・スクリプトの行番号によって定義されます。ブレークポイント・マーカが，テスト・スクリプトの左にあるマージンの中で，対象となる行の横に表示されます。「停止位置」ブレークポイントは，［ブレークポイントのリスト］表示枠に次のように表示されます。

```
ui_test[137] : 0
```

これは，ブレークポイント・マーカが *ui_test* という名前のテストの 137 行目にあることを意味します。コロンの右隣ある数字はパス数を示し，ここでは 0（標準値）に設定されています。つまり，WinRunner はブレークポイントを通過するたびにテストの実行を停止します。

関数で停止

「関数で停止」ブレークポイントは、テストがロードされたコンパイル済みのモジュール内の指定されたユーザ定義関数を呼び出すとテストを停止します。このブレークポイントは，ユーザ定義関数の名前と，その関数が含まれているコンパイル済みモジュールの名前によって定義されます。「関数で停止」ブレークポイントを定義すると，ブレークポイント・マーカが，WinRunner ウィンドウの左マージンの中で，対象となる関数が指定されている行の横に表示されます。WinRunner は，指定された関数が呼び出されるたびに実行を停止します。「関数で停止」ブレークポイントは，［ブレークポイントのリスト］表示枠に次のように表示されます。

```
ui_func [ui_test : 25] : 10
```

これは，*ui_test* というテストの中で，*ui_func* という関数を含んでいる行（ここでは 25 行目）にブレークポイントが設定されていることを示します。パス数は 10 に設定されています。これは，関数が 10 回呼び出されるたびに WinRunner がテストを停止することを示します。

「停止位置」ブレークポイントの設定

「停止位置」ブレークポイントは，［デバッグ ビューア］の［ブレークポイントのリスト］表示枠，マウス，または［ブレークポイントの切り替え］コマンドを使って設定します。

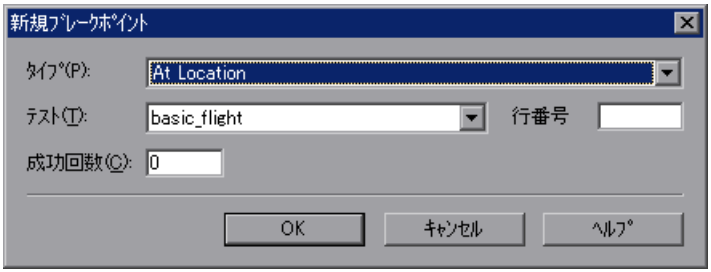
注：WinRunner に関数が読み込まれている（関数が最低 1 回実行された）場合のみ，関数でブレークポイントを設定できます。

［ブレークポイントのリスト］表示枠を使って「停止位置」ブレークポイントを設定するには，次の手順を実行します。

1 814 ページ「［デバッグ ビューア］の［ブレークポイントのリスト］の表示」の説明を参照して，［ブレークポイントのリスト］を表示します。

2 ［**エントリの追加**］を押して，［**新規ブレークポイント**］ダイアログ・ボックスを開きます。

3 ［**タイプ**］ボックスで「**At Location**」を選択します。



4 ［**テスト**］ボックスにアクティブなテストの名前が表示されます。別のテストのブレークポイントを挿入するには，［**テスト**］リストから名前を選択します。

5 ［**行番号**］ボックスに，ブレークポイントを追加する行番号を入力します。

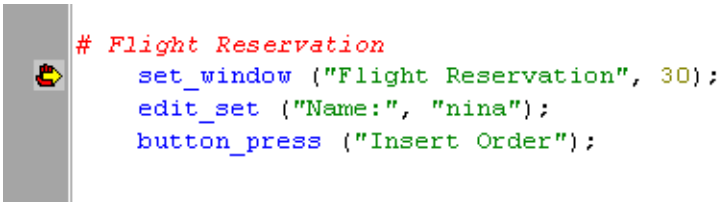
6 ブレークポイントに達するたびにテストをブレークするには，［**成功回数**］で標準設定の 0 を指定します。指定した回数に達してからのみテストをブレークするには，［**成功回数**］ボックスに数字を入力します。

7 [**OK**] を押すと，ブレークポイントが設定され，ダイアログ・ボックスが閉じます。新しいブレークポイントが［ブレークポイントのリスト］表示枠に表示されます。

ブレークポイント・マーカが，テスト・スクリプトの左にあるマージンの内の指定された行の横に表示されます。

マウスを使って「停止位置」ブレークポイントを設定するには，次の手順を実行します。

1 WinRunner ウィンドウで，ブレークポイントを追加する行の左側のマージン（グレー部分）を右ボタンをクリックします。ブレークポイント記号が WinRunner ウィンドウの左マージンに表示されます。



ヒント：グレーのマージンが表示されていない場合は，［**ツール**］＞［**エディタ オプション**］をクリックして，［**オプション**］タブをクリックします。次に［**可視のとじしろ**］オプションを選択します。

---

2 ブレークポイントをこの方法で追加すると，成功回数には自動的に 0 が使用されます。異なる成功回数を使用するには，821 ページ「ブレークポイントの変更」を参照して，ブレークポイントを変更します。

［ブレークポイントの切り替え］コマンドで「停止位置」ブレークポイントを設定するには，次の手順を実行します。

1 挿入ポイントをテスト・スクリプトの中で，テスト実行を停止する行の位置まで移動します。

2 ［**デバッグ**］から［**ブレークポイントの切り替え**］を選択するか，［**ブレークポイントの切り替え**］ボタンをクリックします。WinRunner ウィンドウの左マージンにブレークポイント・マーカが表示され，［ブレークポイントのリスト］にも表示されます。

3 ブレークポイントをこの方法で追加すると，成功回数には自動的に 0 が使用されます。異なる成功回数を使用するには，821 ページ「ブレークポイントの変更」を参照して，ブレークポイントを変更します。

「位置停止」ブレークポイントを削除するには，次の手順を実行します。

マウスの右ボタンでブレークポイントをクリックします。

または

［**デバッグ**］から［**ブレークポイントの切り替え**］を選択するか，［**ブレークポイントの切り替え**］ボタンをクリックします。

「関数で停止」ブレークポイントの設定

「関数で停止」ブレークポイントは，指定したユーザ定義関数でテスト実行を停止します。「関数で停止」ブレークポイントは，［デバッグ ビューア］の［ブレークポイントのリスト］表示枠または，［**関数で停止**］コマンドを使って設定します。

注：WinRunner に関数がロードされた（関数が最低 1 回実行された）場合のみ関数でブレークポイントを設定できます。

「関数で停止」ブレークポイントを設定するには，次の手順を実行します。

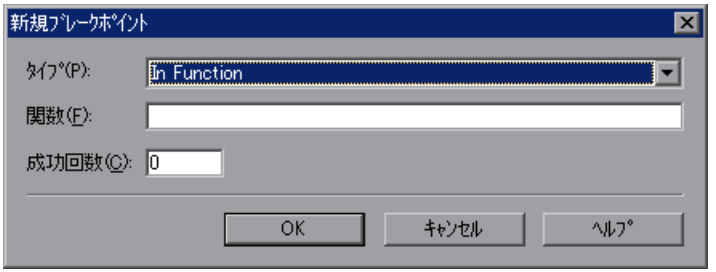
1 すでにテストに含まれている関数の関数ブレークポイントにブレークを設定するには，関数名に挿入ポイントを配置します。

2 ［**デバッグ**］＞［**関数で停止**］を選択します。［新規ブレークポイント］ダイアログ・ボックスが開きます。手順 5 にスキップします。

3 あるいは，［ブレークポイントのリスト］表示枠から［新規ブレークポイント］を開くことができます。814 ページ「[デバッグ ビューア］の［ブレークポイントのリスト］の表示」の説明を参照して，［ブレークポイントのリスト］を表示します。

4 ［**エントリの追加**］をクリックします。

5 ［新規ブレークポイント］ダイアログ・ボックスが開きます。



ブレークポイントのタイプは「**In Function**」を選択します。

6 標準設定では［**関数**］ボックスには，挿入ポイントが現在配置されている関数の名前（またはテキスト）が表示されます。関数名を受け入れるか，有効な関数名を入力します。指定した関数名は WinRunner でコンパイルできなければなりません。詳細については，第 30 章「ユーザ定義関数の作成」および第 31 章「テストでのユーザ定義関数の利用」を参照してください。

7 ［**成功回数**］ボックスに値を入力します。

8 ［**OK**］ボタンをクリックしてブレークポイントを設定して，［新規ブレークポイント］ダイアログ・ボックスを閉じます。

新しいブレークポイントが［ブレークポイントのリスト］表示枠に表示されます。

ブレークポイント記号が，コンパイル済みモジュールの関数の最初の行の左マージンに表示されます。

ブレークポイントの変更

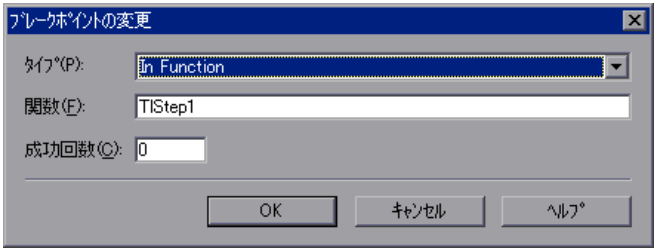
［ブレークポイントの変更］ダイアログ・ボックスを使って，ブレークポイントの定義を修正することができます。ブレークポイントのタイプ，対象となっているテストまたは行番号，パス数の値を修正できます。

ブレークポインタを修正するには，次の手順を実行します。

1 814 ページ「[デバッグ ビューア］の［ブレークポイントのリスト］の表示」の説明を参照して，［ブレークポイントのリスト］を表示します。

2 ［**ブレークポイントのリスト**］表示枠でブレークポイントを選択します。

3 ［**エントリの変更**］をクリックして［ブレークポイントの変更］ダイアログ・ボックスを開きます。



4 ブレークポイントのタイプを修正するには，［**タイプ**］ボックスからブレークポイントのタイプを選択します。

5 必要に応じて設定を変更します。

6 ［**OK**］をクリックして，ダイアログ・ボックスを閉じます。

ブレークポイントの削除

［ブレークポイント］ダイアログ・ボックスを使って，単独のブレークポイントまたは現在のテストで定義されている全てのブレークポイントを削除できます。

単独のブレークポイントを削除するには，次の手順を実行します。

1 814 ページ「［デバッグ ビューア］の［ブレークポイントのリスト］の表示」の説明を参照して，［ブレークポイントのリスト］を表示します。

2 リストからブレークポイントを選択します。

3 ［**エントリの削除**］をクリックします。ブレークポイントがリストから削除され，テストの左マージンからブレークポイント記号が削除されます。

［すべてのブレークポイントを削除］コマンドを使用してすべてのブレークポイントを削除するには，次の手順を実行します。

［**デバッグ**］＞［**すべてのブレークポイントを削除**］を選択するか，［**すべてのブレークポイントを削除**］ツールバー・ボタンをクリックします。

［デバッグ ビューア］を使用して，すべてのブレークポイントを削除するには，次の手順を実行します。

1 814 ページ「［デバッグ ビューア］の［ブレークポイントのリスト］の表示」の説明を参照して，［ブレークポイントのリスト］を表示します。

2 ［**すべてのブレークポイントを削除**］をクリックします。すべてのブレークポイントがリストから削除され，テストの左マージンからすべてのブレークポイント記号が削除されます。

第 40 章

変数の監視

［ウォッチ式のリスト］には，テスト実行の際に，変数，式および配列要素の値が表示されます。［ウォッチ式のリスト］を利用してデバッグ作業を効率よく進めることができます。

本章では，以下の項目について説明します。

- ［ウォッチ式のリスト］への変数の追加
- ［ウォッチ式のリスト］での変数の表示
- ［ウォッチ式のリスト］の変数の変更
- ［ウォッチ式のリスト］の変数への値の割り当て
- ［ウォッチ式のリスト］からの変数の削除

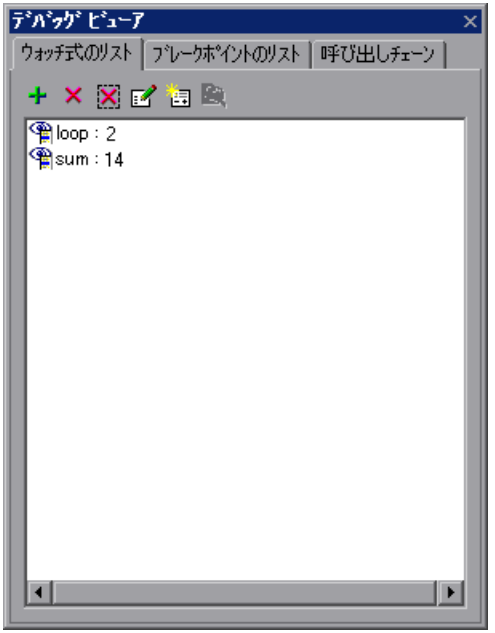
変数の監視について

［ウォッチ式のリスト］を使って，テスト・スクリプトをデバッグする際に，変数，式，および配列要素の値を監視できます。監視したい要素を［ウォッチ式のリスト］に追加します。テスト実行の際，ブレークごとに（［ステップ実行］コマンドの後，ブレークポイント，テストの最後などで）［ウォッチ式のリスト］で項目の現在値を表示できます。

［デバッグ ビューア］での［ウォッチ式のリスト］の表示

［デバッグ ビューア］ウィンドウの［ウォッチ式のリスト］表示枠で変数の値を参照します。［デバッグ ビューア］ウィンドウが現在表示されていない場合，またはウィンドウで［ウォッチ式のリスト］表示枠が表示されていない場合

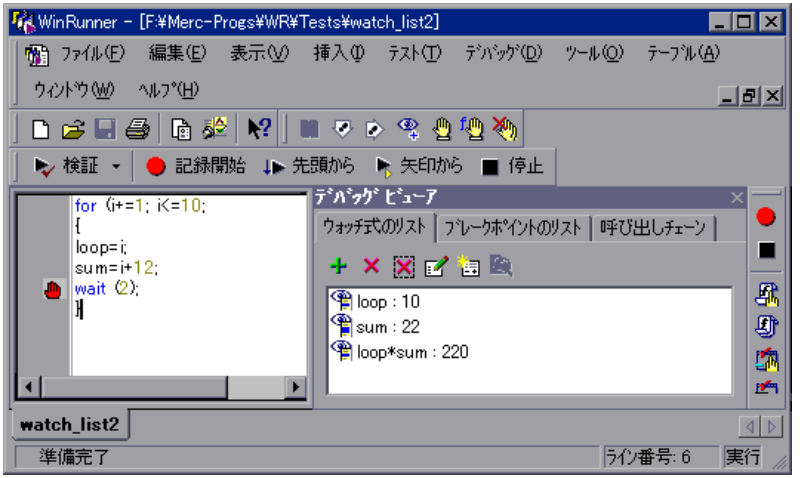
は，［**デバッグ**］＞［**ウォッチ式のリスト**］を選択して表示します。［ウォッチ式のリスト］表示枠が開いているけれども別の表示枠が現在表示されている場合は，［**ウォッチ式のリスト**］タブをクリックしてこれを表示します。



ヒント：［デバッグ ビューア］ウィンドウは，WinRunner ウィンドウ内に固定されたウィンドウとして表示することも，画面上のどこにでもドラッグ可能なフローティング・ウィンドウにすることも可能です。標準設定では，［デバッグ ビューア］は WinRunner 画面の右側に固定されて表示されます。ウィンドウを別の位置に移動するには，［デバッグ ビューア］のタイトル・バーをドラッグします。

変数値の監視－例

例えば，次のテストでは，［ウォッチ式のリスト］を使って，変数 *loop*（現在のループ）と *sum* の値を測定し監視します。各ループの最後のステップで，テストはブレークポイントで一時停止するので，現在の値を見ることができます。



WinRunner が最初のループを実行すると，テストは一時停止します。［ウォッチ式のリスト］には変数が表示され，それらの値が更新されます。WinRunner がテスト実行を完了すると，［ウォッチ式のリスト］には次のような結果が表示されます。

```
loop:10
sum:22
loop*sum:220
```

テスト・スクリプトに，スコープは異なるが同じ名前の変数がある場合，それらの変数は，インタープリタにおける現在のスコープに応じて評価されます。例えば，*test_a* と *test_b* の両方に，*x* という static 変数があり *test_a* が *test_b* を呼び出したとします。［ウォッチ式のリスト］に変数 *x* を含めると，表示される *x* の値は，その時点で WinRunner が test_a または test_b のどちらを解釈しているかによります。

［呼び出しチェーン］のリスト（［デバッグ］＞［呼び出しチェーン］）からテストまたは関数を選択した場合，［ウォッチ式のリスト］における変数の対象範囲と式が変わります。WinRunner は，［ウォッチ式のリスト］の変数の値を自動的に更新します。

［ウォッチ式のリスト］への変数の追加

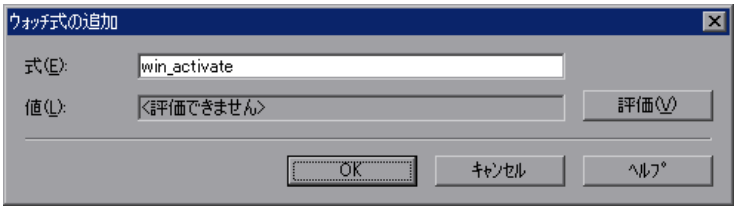
［ウォッチ式のリスト］に変数，式，配列を追加するには，［ウォッチ式の追加］ダイアログ・ボックスを使用します。項目は，テストの実行前，テストが［停止］コマンドでブレークしたとき，テストが一時停止したときまたはブレークポイントにおいて追加できます。

［ウォッチ式のリスト］に変数，式，配列を追加するには，次の手順を実行します。

1 ［**デバッグ**］＞［**ウォッチ式の追加**］を選択するか［**ウォッチ式を追加**］ボタンをクリックします。

または，823 ページ「[デバッグ ビューア］での［ウォッチ式のリスト］の表示」で説明する方法で［ウォッチ式のリスト］を表示して，［**エントリの追加**］をクリックします。

2 ［ウォッチ式の追加］ダイアログ・ボックスを開きます。



3 ［**式**］ボックスに，［ウォッチ式のリスト］に追加する変数，式または配列を入力します。

4 ［**評価**］をクリックして，新しいエントリの現在の値を見ることができます。新しいエントリの変数または配列がまだ初期化されていない場合，［**値**］フィールドに「< 評価できません >」というメッセージが表示されます。誤りがある式を指定した場合も同じメッセージが表示されます。

5 ［**OK**］をクリックして ［**ウォッチ式の追加**］ダイアログ・ボックスを閉じます。新しいエントリが［**ウォッチ式のリスト**］に表示されます。

注 : ［ウォッチ式のリスト］に変数の値を代入したりインクリメントするような式を追加してはなりません。テストの実行に影響を与える可能性があります。

［ウォッチ式のリスト］での変数の表示

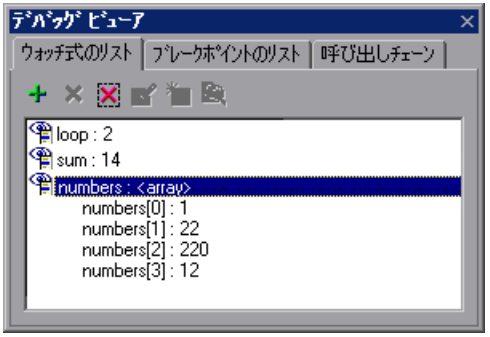
［ウォッチ式のリスト］に変数，式，配列を追加したら，それらを［ウォッチ式のリスト］に表示できます。

［ウォッチ式のリスト］に変数，式，配列を表示するには，次の手順を実行します。

1 823 ページ「[デバッグ ビューア］での［ウォッチ式のリスト］の表示」で説明する方法で，［ウォッチ式のリスト］を表示します。

変数，式，配列が表示されます。現在の値は，コロンに続いて表示されます。

2 配列要素の値を見るには，配列名をダブルクリックします。配列の要素とその値が配列名の下に表示されます。配列名をダブルクリックすると，要素が隠れます。



3 ［**閉じる**］をクリックしてダイアログ・ボックスを閉じます。

［ウォッチ式のリスト］の変数の変更

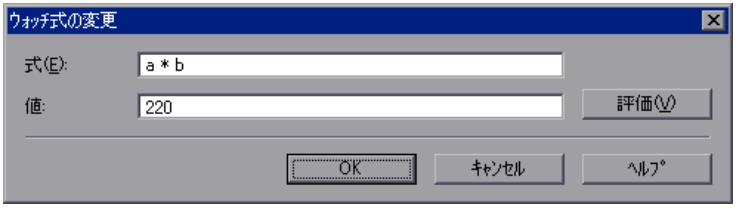
［ウォッチ式のリスト］の変数や式は，［ウォッチ式の変更］ダイアログ・ボックスを使って変更できます。例えば，変数 *b* を *b* + *1* という式に変えたり，*b* + *1* という式を *b* * *10* という式に変更することもできます。［ウォッチ式の変更］ダイアログ・ボックスを閉じると，［ウォッチ式のリスト］が更新されて式の新しい値が反映されます。

［ウォッチ式のリスト］の式を変更するには，次の手順を実行します。

1 823 ページ「［デバッグ ビューア］での［ウォッチ式のリスト］の表示」で説明する方法で，［ウォッチ式のリスト］を表示します。

2 変更する変数または式を選択します。

3 ［**エントリの変更**］をクリックして［**ウォッチ式の変更**］ダイアログ・ボックスを開きます。



4 必要に応じて［**式**］フィールドの式を変更します。

5 ［**評価**］ボタンをクリックします。式の新しい値が［**値**］フィールドに表示されます。

6 ［**OK**］をクリックして［**ウォッチ式の変更**］ダイアログ・ボックスを閉じます。変更した式とその新しい値が［ウォッチ式のリスト］に表示されます。

［ウォッチ式のリスト］の変数への値の割り当て

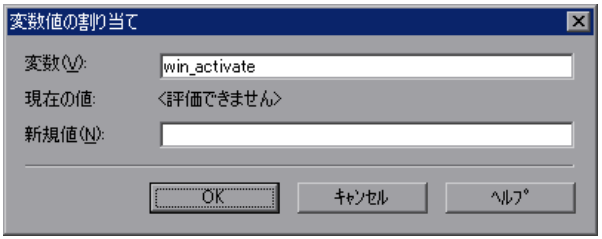
［ウォッチ式のリスト］では，変数や配列の要素に新しい値を割り当てることができます。値は，変数と配列の要素にのみ代入できます。式に値を代入することはできません。

変数または配列の要素に値を代入するには，次の手順を実行します。

1 823 ページ「[デバッグ ビューア] での [ウォッチ式のリスト] の表示」で説明する方法で，[ウォッチ式のリスト] を表示します。

2 値を割り当てる変数または配列の要素を選択します。

3 [**変数値の割り当て**] をクリックして，[**変数値の割り当て**] ダイアログ・ボックスを開きます。



4 [**新規値**] フィールドに変数または配列要素に割り当てる新しい値を入力します。

5 [**OK**] をクリックしてフォームを閉じます。新しい値が [ウォッチ式のリスト] に表示されます。

[ウォッチ式のリスト] からの変数の削除

[ウォッチ式のリスト] の特定の変数，式または配列を削除することができます。また，[ウォッチ式のリスト] の全てのエントリを一度に削除することもできます。

変数，式または配列を削除するには，次の手順を実行します。

1 823 ページ「[デバッグ ビューア] での [ウォッチ式のリスト] の表示」で説明する方法で，[ウォッチ式のリスト] を表示します。

2 削除する変数，式または配列をクリックします。

注：配列は，その要素が隠れている場合にのみ削除できます。配列の要素を隠すには，[ウォッチ式のリスト] で配列名をダブルクリックします。

3 ［**エントリの削除**］をクリックすると，リストからエントリが削除されます。

4 ［**閉じる**］をクリックしてダイアログ・ボックスを閉じます。

［ウォッチ式のリスト］のすべてのエントリを削除するには，次の手順を実行します。

1 823 ページ「[デバッグ ビューア］での［ウォッチ式のリスト］の表示」で説明する方法で，［ウォッチ式のリスト］を表示します。

2 ［**すべてのエントリを削除**］を選択します。すべてのエントリが削除されます。

3 ［**閉じる**］をクリックしてダイアログ・ボックスを閉じます。

第 11 部

上級者向けの設定

第 41 章

テスト・スクリプト・エディタのカスタマイズ

WinRunner にはカスタマイズが可能で強力なスクリプト・エディタがあります。これにより，テスト・ウィンドウのマージン・サイズの設定や，テスト・スクリプト要素の表示形式の変更，また WinRunner によって自動的に修正される入力ミスのリストの作成などが行えます。

本章では，以下の項目について説明します。

- テスト・スクリプト・エディタのカスタマイズについて
- 表示オプションの設定
- 編集コマンドのユーザ設定

テスト・スクリプト・エディタのカスタマイズについて

WinRunner のスクリプト・エディタを使って，表示オプションを設定したり，スクリプト編集コマンドを個人向けに定義したりできます。

表示オプションの設定

表示オプションを設定することで，WinRunner のテスト・ウィンドウやテスト・スクリプトの表示方法を設定できます。例えば，テスト・ウィンドウのマージン・サイズの設定や行の折り返しの有効化 / 無効化などが行えます。

表示オプションはまた異なるスクリプト要素ごとに色と表示を変更できるようにします。スクリプト要素にはコメント，文字列，WinRunner の予約語，演算子，数値があります。こうした個々の要素に対して，色，テキスト属性（太字，斜体，下線），フォント，フォント・サイズを設定できます。例えば，すべての文字列を赤で表示することもできます。

それ以外にも表示オプションによってスクリプトを紙に出力したときの出力形式も制御できます。

スクリプト編集コマンドのユーザ定義化

WinRunner にはカーソル移動，文字の削除，クリップボードの情報の切り取り，コピー，貼り付けを実行する標準設定のキーボード・コマンドのリストがあります。これらのコマンドは自分の好みのコマンドと置き換えることもできます。例えば，［Set Bookmark ［#］］コマンドを標準設定の CTRL+K+［#］から CTRL+B+［#］に変更できます。

表示オプションの設定

WinRunner の表示オプションを使って，テスト・ウィンドウ内のテスト・スクリプトの表示方法，テスト・スクリプトの様々な要素の表示方法，テスト・スクリプトの印刷形式を制御できます。

テスト・スクリプトとウィンドウのカスタマイズ

WinRunner のテスト・ウィンドウの外観とスクリプトの表示をカスタマイズできます。例えば，テスト・ウィンドウのマージン・サイズの設定，スクリプト要素の強調表示，テキスト記号の表示 / 非表示などが行えます。

スクリプトの表示形式をカスタマイズするには，次の手順を実行します。

1 ［**ツール**］ > ［**エディタ オプション**］を選択します。［エディタ オプション］ダイアログ・ボックスが開きます。

2 ［**オプション**］タブをクリックします。

3 ［**一般オプション**］の下で，以下のオプションを選びます。

オプション	説明
自動インデント	字下げされた行に合わせて，以降の行が自動的に字下げされます。キーボードの Home キーをクリックすると，カーソルが左マージンに戻ります。
スマート タブ	Tab キーを 1 度押すと，上の行のテキストに合うように，適切な数のタブとスペースが挿入されます。

オプション	説明
スマート フィル	［自動インデント］オプションが適用されるよう，適切な数のタブとスペースを挿入します。このオプションが選択されていないと，［自動インデント］を適用するのにスペースだけが使用されます。 **注：**このオプションを適用するには，［**自動インデント**］と［**タブ文字を使用**］が選択されていなければなりません。
タブ文字を使用	キーボードの Tab キーが使用されるとタブ文字を挿入します。このオプションが無効になっているときは，適切な数のスペースが挿入されます。
とじしろの行数	スクリプトの各行の横に行番号を表示します。行番号はテスト・スクリプト・ウィンドウのマージンに表示されます。
ステートメントの完了	Ctrl と Space キーを同時に押すか，アンダスコア・キーを押すと，関数の当該接頭辞に一致するすべての関数を表示するリスト・ボックスが開きます。リストから項目を選択して，入力した文字列を置換します。リスト・ボックスを閉じるには，Esc キーを押します。エディタに入力されたことのある完全な関数には，その関数のパラメータを示すツールチップが表示されます。
すべての文字を表示	タブ記号や段落記号などのテキストの表示記号をすべて表示します。
上書きのブロックカーソル	上書きモードを選択したときに，標準カーソルの代わりにブロック・カーソルを表示します。
単語の選択	テスト・ウィンドウをダブルクリックしたときに，1 番近い単語を選びます。
構文の強調表示	コメント，文字列，予約語などのスクリプト要素を強調表示します。予約語の詳細については，839 ページ「予約語」を参照してください。
可視の右余白	テスト・ウィンドウの右マージンを示す線を表示します。
右余白	テスト・ウィンドウの右マージンの位置を文字数で設定します。

オプション	説明
可視のとじしろ	テスト・ウィンドウの左マージンの余白部分を表示します。
とじしろの幅	マージンの幅を文字数単位で設定します。
ブロック インデントのサイズ	［INDENT SELECTED BLOCK］ソフトキーを使用すると，選択した TSL ステートメントのブロックが何文字分移動（字下げ）されるかを設定します。エディタ オプションの詳細については，841 ページ「編集コマンドのユーザ設定」を参照してください。
タブ ストップ	各タブ・ストップ間の文字数を設定します。

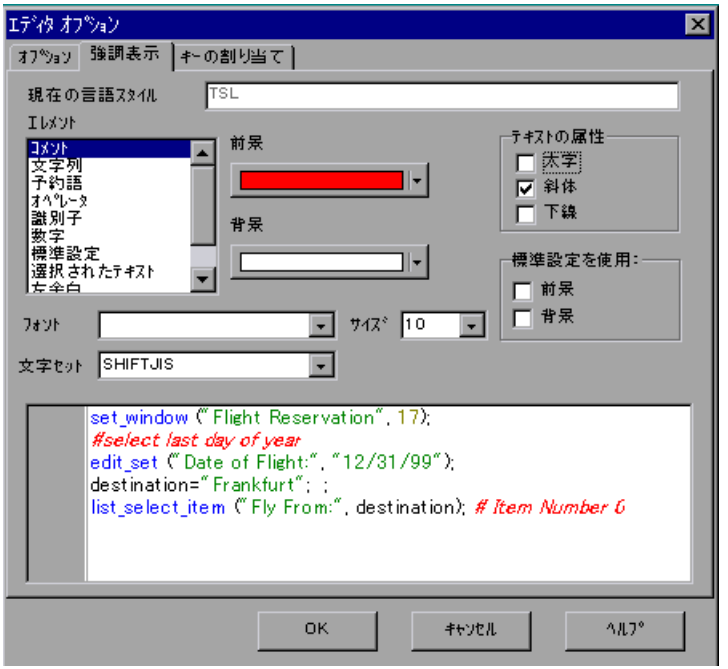
スクリプト要素の強調表示

WinRunner のスクリプトにはコメント，文字列，WinRunner 予約語，演算子，数値などの多くの異なる要素が含まれています。WinRunner のスクリプトの各要素は異なる色とスタイルで表示されます。ユーザは個々のスクリプト要素に対して独自の配色とスタイルを作ることができます。例えば，スクリプト内のすべてのコメントを斜体，青文字，黄色の背景で表示できます。

スクリプト要素を編集するには，次の手順を実行します。

1 ［**ツール**］＞［**エディタ オプション**］を選択します。［エディタ オプション］ダイアログ・ボックスが開きます。

2 [**強調表示**] タブをクリックします。



3 [**エレメント**] リストからスクリプト要素を選択します。

4 次のオプションから選びます。

オプション	説明
前景	スクリプト要素のテキストに適用する色を設定します。
背景	スクリプト要素の背景に適用する色を設定します。
テキストの属性	スクリプト要素に適用するテキスト属性を設定します。太字，斜体，下線を選択できます。これらは組み合わせて使用することも可能です。
標準設定を使用	選択したスタイルに「標準の」スタイルのフォントと色を適用します。
フォント	すべてのスクリプト要素の活字を設定します。

オプション	説明
サイズ	すべてのスクリプト要素のサイズをポイント単位で設定します。
文字セット	選択したフォントの文字サブセットを設定します。

適用する個々の変更例はダイアログ・ボックスの下部の表示枠内に表示されます。

5 ［OK］をクリックして，変更内容を適用します。

予約語

WinRunner には「予約語」があります。予約語にはすべての TSL 関数名と auto，break，char，close，continue，int，function などの TSL 言語のキーワードを含みます。WinRunner のすべての予約語の完全なリストについては「**TSL リファレンス**」を参照してください。WinRunner のインスタレーション・ディレクトリ内の dat フォルダにある reserved_words.ini ファイルの [ct_KEYWORD_USER] セクションにユーザ独自の予約語を追加できます。テキスト・エディタ（メモ帳アプリケーションなど）を使ってファイルを開きます。ただし，リストを編集した後は，WinRunner を再起動して，更新済みのリストを読み込ませなくてはなりません。

印刷オプションのカスタマイズ

スクリプトをプリンタに送信する際，紙への出力形式を設定できます。例えば，印刷されたスクリプトに行番号，ファイル名，印刷日を含めることができます。

印刷オプションをカスタマイズするには次の手順を実行します。

1 ［**ツール**］＞［**エディタ オプション**］を選択します。［エディタ オプション］ダイアログ・ボックスが開きます。

2［**オプション**］タブをクリックします。

3 以下の印刷オプションから選びます。

オプション	説明
長い行を折り返す	現在のプリンタのページ設定よりもテキストの行が長い場合，自動的に次の行に折り返します。
行番号	スクリプトの各行の横に行番号を印刷します。
ヘッダ内のタイトル	印刷されるスクリプトのヘッダにファイル名を挿入します。
ヘッダの日付	印刷されるスクリプトのヘッダに今日の日付を挿入します。
ページ番号	スクリプトの各ページに番号を振ります。

4［**OK**］をクリックして，変更を適用します。

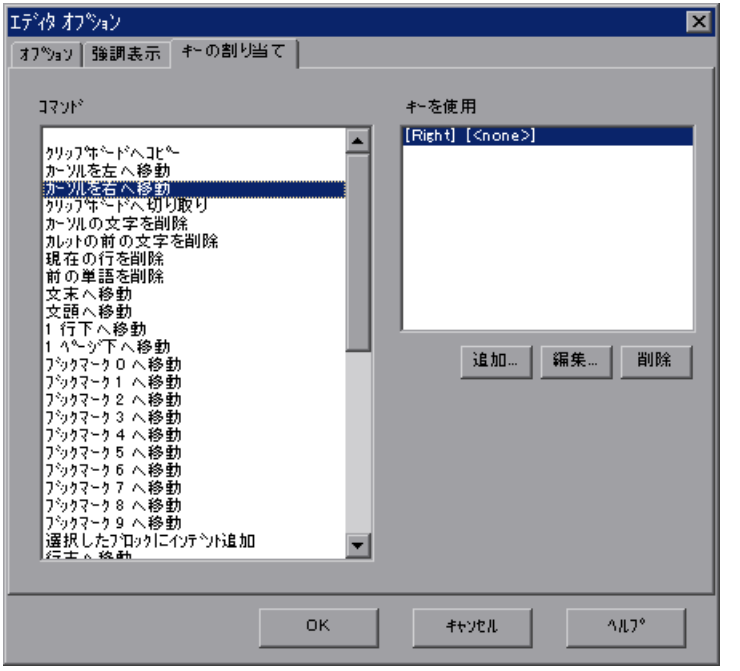
編集コマンドのユーザ設定

テスト・スクリプトの編集に使用する標準のキーボード・コマンドは個人向けに設定できます。WinRunner にはカーソルの移動，文字の削除，クリップボードの情報の切り取り，コピー，貼り付けを可能にする標準設定のキーボード・コマンドがあります。これらのコマンドを自分の好みのコマンドと置き換えることができます。例えば，［貼り付け］コマンドを標準設定の CTRL+V から CTRL+P に変更できます。

編集コマンドを個人向けにユーザ設定するには，次の手順を実行します。

1 ［**ツール**］＞［**エディタ オプション**］を選択します。［エディタ オプション］ダイアログ・ボックスが開きます。

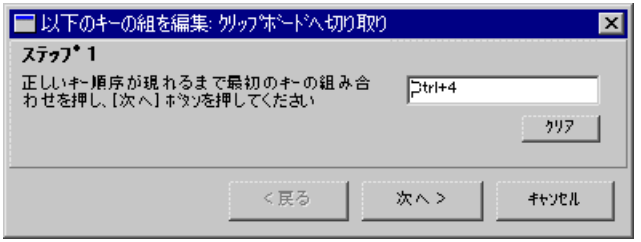
2 ［**キーの割り当て**］タブをクリックします。



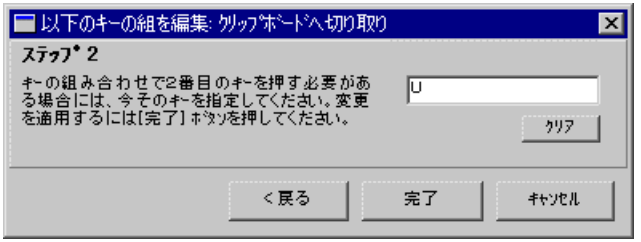
3 ［**コマンド**］リストからコマンドを選びます。

4 ［**追加**］をクリックして新しいキーの割り当てを作成するか，［**編集**］をクリックして既存のキーの割り当てを変更します。［以下のキーの組を編集］ダ

イアログ・ボックスが開きます。使いたいキーを押します。例えば CTRL + 4 などです。



5 ［**次へ**］をクリックします。組み合わせるキーを追加したい場合には，使用するキーを押します。例えば U などです。



6 ［**完了**］をクリックして，キーの組み合わせを［**キーを使用**］リストに追加します。

リストからキーの組み合わせを削除したい場合には，［**キーを使用**］リストの中からキーを強調表示して，［**削除**］をクリックします。

7 ［**OK**］をクリックして，変更を適用します。

第 42 章

WinRunner のユーザ・インタフェースのカスタマイズ

WinRunner のユーザ・インタフェースをカスタマイズして，テストのニーズおよびテスト対象のアプリケーションに合わせることができます。

本章では，以下の項目について説明します。

- WinRunner のユーザ・インタフェースのカスタマイズについて
- ファイル・ツールバー，デバッグ・ツールバー，ユーザ定義ツールバーのカスタマイズ
- ユーザ定義ツールバーのカスタマイズ
- ユーザ定義ツールバーの使い方
- WinRunner ソフトキーの構成設定

WinRunner のユーザ・インタフェースのカスタマイズについて

WinRunner のコマンドへのアクセス方法を変えることによって，WinRunner のユーザ・インタフェースをテストのニーズに合わせることができます。

テストを作成して実行しているときに，しばしば同じ WinRunner メニュー・コマンドを使って同じ TSL ステートメントをテスト・スクリプトに挿入していることに気づくことがあります。WinRunner ツールバーをカスタマイズすることによって，これらのコマンドと TSL ステートメントのショートカットを作成できます。

テスト対象アプリケーションによって，WinRunner コマンド用にあらかじめ設定されているソフトキーが使われていることがあります。その場合には，WinRunner の Softkey ユーティリティを使って衝突する WinRunner ソフトキーを設定し直すことによって，WinRunner のユーザ・インタフェースをそのアプリケーションに対応させることができます。

ファイル・ツールバー，デバッグ・ツールバー，ユーザ定義ツールバーのカスタマイズ

［ツールバーのカスタマイズ］オプションを使用して，ユーザ定義ツールバーを作成したり，ファイル・ツールバー，デバッグ・ツールバー，ユーザ定義ツールバーの外観や内容をカスタマイズできます。

注：ユーザ定義ツールバーをカスタマイズすることもできます。詳細については，852 ページ「ユーザ定義ツールバーのカスタマイズ」を参照してください。

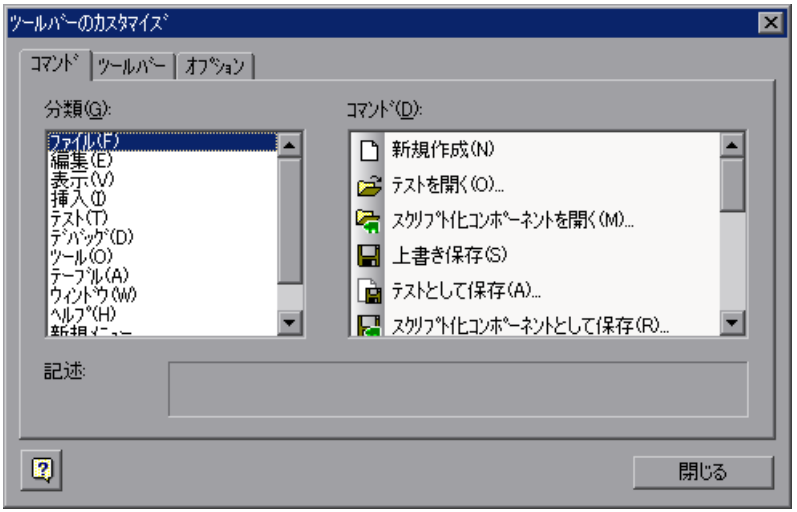
メニュー・コマンドを実行するツールバー・ボタンの追加と削除

［ツールバーのカスタマイズ］ダイアログ・ボックスの［**コマンド**］タブを使用して，よく使用するメニュー・コマンドを実行するツールバー・ボタンをファイル・ツールバー，デバッグ・ツールバー，既存のユーザ定義ツールバーに追加できます。これらのツールバーからツールバー・ボタンを削除することもできます。

ヒント：選択したツールバーまたはすべてのツールバーのボタンを標準設定に戻すには，［**ツールバー**］タブで［**元に戻す**］または［**すべて元に戻す**］を選択します。詳細については，846 ページ「ツールバーの表示の制御」を参照してください。

ファイル・ツールバー，デバッグ・ツールバー，ユーザ定義ツールバーにボタンを追加するには，次の手順を実行します。

1 ［**表示**］＞［**ツールバーのカスタマイズ**］を選択します。［ツールバーのカスタマイズ］ダイアログ・ボックスが開き，［**コマンド**］タブが表示されます。



2 ［**分類**］リストで，ツールバーに追加するコマンドを含むメニュー名を見つけて選択します。

3 ［**コマンド**］リストで追加したいコマンドを選択し，ファイル・ツールバー，デバッグ・ツールバー，ユーザ定義ツールバーにドラッグします。

4 これらのツールバー上にボタンを移動すると，マウス・ポインタが I 型のカーソルに変わり，ボタンが配置される場所を示します。I 型のカーソルをボタンを追加したい場所にドラッグして，マウス・ボタンを放します。

ヒント：［ツールバーのカスタマイズ］ダイアログ・ボックスが開いている間に，ツールバー・ボタンをあるツールバーから別のツールバーにドラッグすることもできます。

ファイル・ツールバー，デバッグ・ツールバー，ユーザ定義ツールバーからボタンを削除するには，次の手順を実行します。

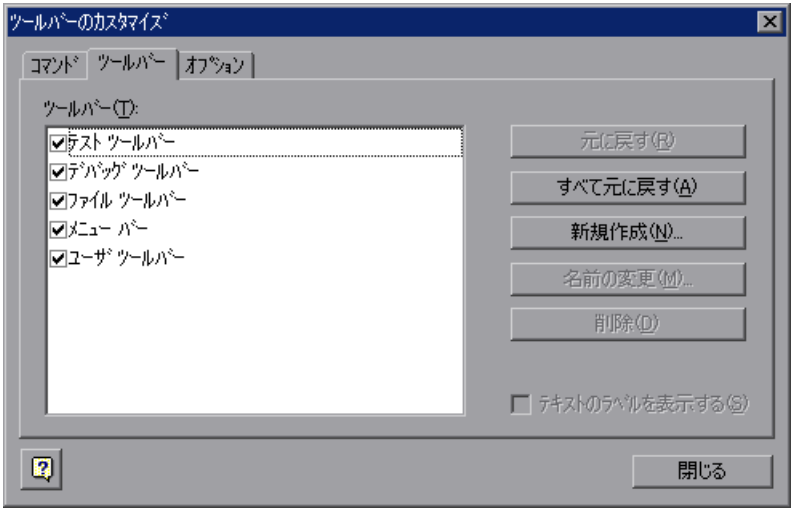
1 ［**表示**］＞［**ツールバーのカスタマイズ**］を選択します。［ツールバーのカスタマイズ］ダイアログ・ボックスが開きます。

2 ツールバーから削除するツールバー・ボタンをツールバーの外にドラッグすると，そのボタンが削除されます。

ツールバーの表示の制御

［ツールバーのカスタマイズ］ダイアログ・ボックスの［**ツールバー**］タブを使用して，ツールバーの表示・非表示，ツールバーへの標準設定のボタンの回復，ユーザ定義ツールバーの作成・名前の変更・削除，個々のツールバーの外観の制御が行えます。

ヒント：［**表示**］メニューからオプションを選択して，WinRunner ツールバーの表示・非表示を切り替えることもできます。



ツールバーの表示・非表示を切り替えるには，次の手順を実行します。

1 ［**表示**］＞［**ツールバーのカスタマイズ**］を選択します。［ツールバーのカスタマイズ］ダイアログ・ボックスが開き，［**コマンド**］タブが表示されます。

2 ［**ツールバー**］タブをクリックします。

3 WinRunner ツールバーまたはユーザ定義ツールバーの横のチェック・ボックスを選択またはクリアして，表示・非表示を切り替えます。

注：［**メニュー**］バーを非表示にすることはできません。

選択したツールバーまたはすべてのツールバーのボタンを標準設定に戻すには，次の手順を実行します。

1 ［**表示**］＞［**ツールバーのカスタマイズ**］を選択します。［ツールバーのカスタマイズ］ダイアログ・ボックスが開き，［**コマンド**］タブが表示されます。

2 ［**ツールバー**］タブをクリックします。

3 特定のツールバーのボタンを標準設定に戻すには，ツールバー・リストからツールバーを選択し，［**元に戻す**］をクリックします。

注：ユーザ定義ツールバーが選択されているときは［**元に戻す**］ボタンは無効になります。

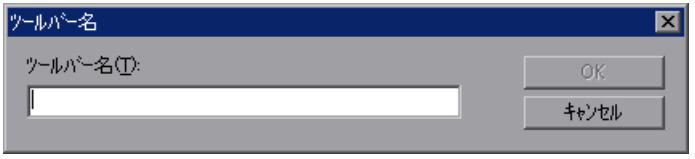
すべての WinRunner ツールバーのボタンを標準設定に戻すには，［**すべて元に戻す**］をクリックします。

ユーザ定義ツールバーを作成するには，次の手順を実行します。

1 ［**表示**］＞［**ツールバーのカスタマイズ**］を選択します。［ツールバーのカスタマイズ］ダイアログ・ボックスが開き，［**コマンド**］タブが表示されます。

2 ［**ツールバー**］タブをクリックします。

3 ［**新規作成**］をクリックします。［ツールバー名］ダイアログ・ボックスが開きます。



4 ツールバーの一意の名前を入力して［**OK**］をクリックします。新規ツールバーの名前がツールバー・リストに追加されます。空のツールバーが**フローティング・ツールバー**として画面の中央に表示されます。

5 ツールバーを希望の位置にドラッグします。ツールバーを右上のツールバー領域にドラッグすると，WinRunner ウィンドウに固定されます（タイトルバーがツールバー・ハンドルに変わります)。

ヒント：ツールバーは，タイトルバーをダブルクリックしても上部のツールバー領域の標準設定の位置に固定されます。

---

6 ［ツールバーのカスタマイズ］ダイアログ・ボックスの［**コマンド**］タブを選択して，ツールバー・ボタンを新規ツールバーに追加します。詳細については，844 ページ「メニュー・コマンドを実行するツールバー・ボタンの追加と削除」を参照して下さい。

ユーザ定義ツールバーの名前を変更するには，次の手順を実行します。

1 ［**表示**］＞［**ツールバーのカスタマイズ**］を選択します。［ツールバーのカスタマイズ］ダイアログ・ボックスが開き，［**コマンド**］タブが表示されます。

2 ［**ツールバー**］タブをクリックします。

3 名前を変更するユーザ定義ツールバーを選択します。

注：［**名前の変更**］オプションは，ユーザ定義ツールバーが選択されている場合にだけ有効になります。

---

4 ［**名前の変更**］をクリックします。［ツールバー名］ダイアログ・ボックスが開き，選択されたツールバーの現在の名前が表示されます。

5 新しい名前を入力して，［**OK**］をクリックします。

ユーザ定義ツールバーを削除するには，次の手順を実行します。

1 ［**表示**］＞［**ツールバーのカスタマイズ**］を選択します。［ツールバーのカスタマイズ］ダイアログ・ボックスが開き，［**コマンド**］タブが表示されます。

2 ［**ツールバー**］タブをクリックします。

3 名前を変更するユーザ定義ツールバーを選択します。

注 :［**削除**］オプションは，ユーザ定義ツールバーが選択されている場合だけ有効になります。

---

4 ［**削除**］をクリックします。

5 ［**はい**］をクリックします。選択したツールバーを削除することを確認します。ツールバーが，ツールバー・リストと WinRunner ウィンドウから削除されます。

デバッグ・ツールバー，ファイル・ツールバー，テスト・ツールバーにテキスト・ラベルを表示するには，次の手順を実行します。

1 ［**表示**］＞［**ツールバーのカスタマイズ**］を選択します。［ツールバーのカスタマイズ］ダイアログ・ボックスが開き，［**コマンド**］タブが表示されます。

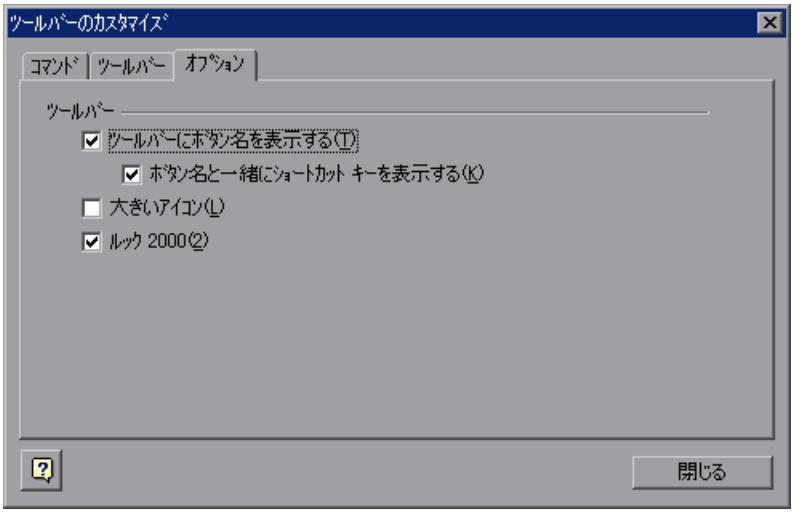
2 ［**ツールバー**］タブをクリックします。

3 ツールバー・リストから［**デバッグ ツールバー**］，［**ファイル ツールバー**］，または［**テスト ツールバー**］を選択します。

4 ［**テキストのラベルを表示する**］チェック・ボックスを選択します。

ツールバー・オプションの設定

［ツールバーのカスタマイズ］ダイアログ・ボックスの［**オプション**］タブを使用して，すべてのツールバーに適用するオプションを設定できます。



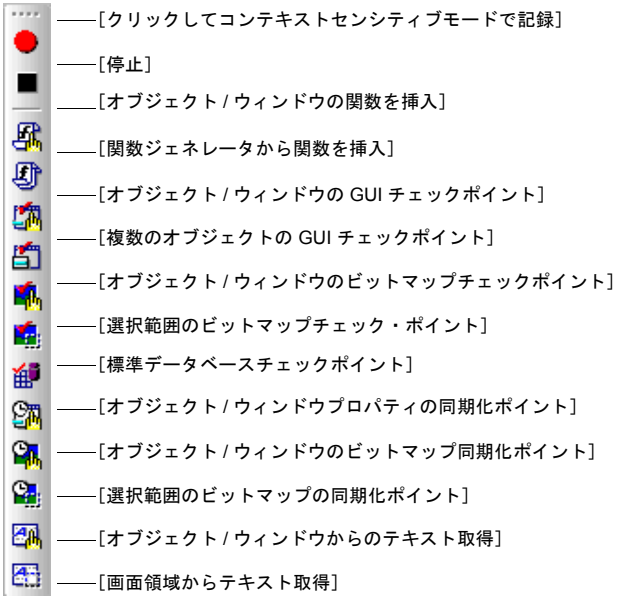
［**オプション**］タブには，次のオプションが含まれます。

オプション	説明
ツールバーにボタン名を表示する	マウスでツールバー・ボタンを指すと，そのボタンで表されるコマンド名を含むツール・チップが表示されます。
ボタン名と一緒にショートカット キーを表示する	画面上のヒントにツールバー・ボタンで表されるコマンドのショートカット・キーが表示されます。［**ツールバーにボタン名を表示する**］が選択されている場合だけ有効になります。

オプション	説明
大きいアイコン	すべてのツールバー・ボタンが大きいアイコンで表示されます。
ルック 2000	選択すると，ツールバーのハンドルが Windows 2000 形式の 1 つのバーに表示されます。クリアすると，ツールバー・ハンドルが 2 つのバーに表示されます。このオプションは，［一般オプション］ダイアログ・ボックスの［**概観**］カテゴリで［**標準設定**］のテーマが選択された場合にのみ使用できます。

ユーザ定義ツールバーのカスタマイズ

ユーザ定義ツールバーにはテストを作成する際に使用するコマンドのボタンがあります。標準の設定では，以下の WinRunner のコマンドに簡単にアクセスできるように設定されています。



標準設定ではユーザ定義ツールバーは非表示です。表示するには，［表示］＞［ユーザ定義ツールバー］を選択するか，［ツールバーのカスタマイズ］ダイアログ・ボックスの［ツールバー］タブで［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，標準では WinRunner ウィンドウの右端に固定されて表示されます。

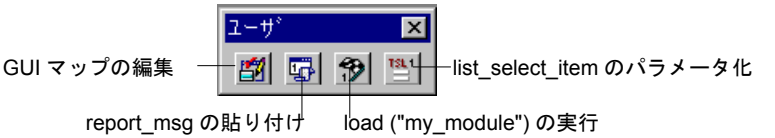
ユーザ定義ツールバーはカスタマイズ可能なツールバーです。ボタンを追加したり削除したりして，アプリケーションをテストするときにもっとも頻繁に使うコマンドにアクセスしやすくできます。ユーザ定義ツールバーを使って次のことができます。

- 追加の WinRunner メニュー・コマンドの実行。例えば，［GUI マップ エディタ］を開くボタンをユーザ定義ツールバーに追加することができます。

- テスト・スクリプトへの TSL ステートメントの貼り付け。例えば，TSL ステートメント **report_msg** をテスト・スクリプトに貼り付けるボタンをユーザ定義ツールバーに追加することができます。
- TSL ステートメントの実行。例えば，次の TSL ステートメントを実行するボタンをユーザ定義ツールバーに追加できます。

```
load ("my_module");
```

- テスト・スクリプトへの貼り付けまたは実行する前の TSL ステートメントのパラメータ化。例えば，TSL ステートメント **list_select_item** にパラメータを追加できるようにし，それをテスト・スクリプトに貼り付けたり，実行したりするためのボタンをユーザ定義ツールバーに追加できます。



注：上の図に表示されているボタンはいずれも，標準でユーザ定義ツールバーに表示されるボタンではありません。

メニュー・コマンドを実行するボタンのユーザ定義ツールバーへの追加

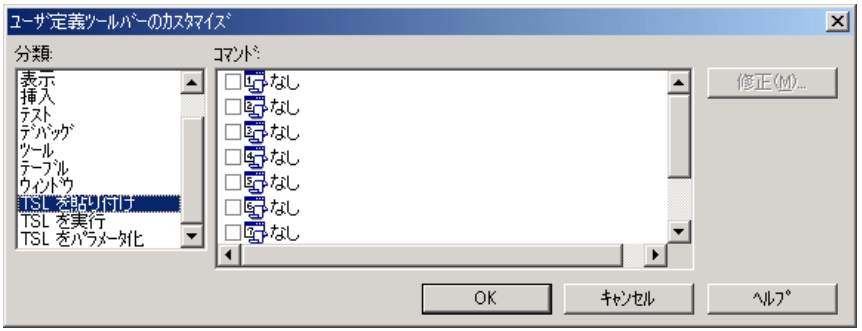
［ツールバーのカスタマイズ］ダイアログ・ボックスを使用して，ユーザ定義ツールバーに頻繁に使うメニュー・コマンドを実行するボタンを追加できます。

注：ファイル・ツールバーおよび**デバッグ・ツールバー**にボタンを追加することも，ユーザー定義ツールバーを作成することもできます。詳細については，844 ページ「ファイル・ツールバー，デバッグ・ツールバー，ユーザ定義ツールバーのカスタマイズ」を参照してください。

メニュー・コマンドを実行するボタンを追加するには，次の手順を実行します。

1 ［**表示**］＞［**ユーザ定義ツールバーのカスタマイズ**］を選択します。

［ユーザ定義ツールバーのカスタマイズ］ダイアログ・ボックスが開きます。



メニュー・バーの各メニューは，［ユーザ定義ツールバーのカスタマイズ］ダイアログ・ボックスの［分類］枠内のカテゴリに相当します。

2 ［**分類**］枠でメニューを選択します。

3 ［**コマンド**］枠でメニュー・コマンドの横のチェック・ボックスを選択します。

4 ［**OK**］をクリックして［ユーザ定義ツールバーのカスタマイズ］ダイアログ・ボックスを閉じます。

これで選択したメニュー・コマンド・ボタンがユーザ定義ツールバーに追加されます。

ユーザ定義ツールバーからメニュー・コマンドを削除するには，次の手順を実行します。

1 ［**表示**］＞［**ユーザ定義ツールバーのカスタマイズ**］を選んで，［ユーザ定義ツールバーのカスタマイズ］ダイアログ・ボックスを開きます。

2 ［**分類**］枠でメニューを選択します。

3 ［**コマンド**］枠でメニュー・コマンドの横のチェック・ボックスをクリアします。

4 ［**OK**］をクリックして［**ユーザ定義ツールバーのカスタマイズ**］ダイアログ・ボックスを閉じます。

これで選択したメニュー・コマンド・ボタンがユーザ定義ツールバーから削除されます。

ヒント：［ツールバーのカスタマイズ］ダイアログ・ボックスの［**ツールバー**］タブで［**元に戻す**］または［**すべて元に戻す**］ボタンを使用して，ユーザ定義ツールバーのボタンを標準設定に戻すことができます。詳細については，846 ページ「ツールバーの表示の制御」を参照してください。

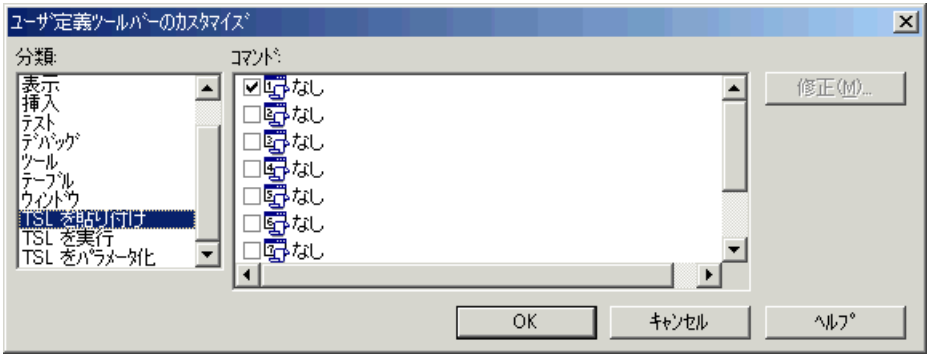
TSL ステートメントを貼り付けるボタンの追加

テスト・スクリプトに TSL ステートメントを貼り付けるボタンをユーザ定義ツールバーに追加することができます。1 つのボタンで 1 つの TSL ステートメント，またはステートメントのグループを貼り付けることができます。

TSL ステートメントを貼り付けるボタンを追加するには，次の手順を実行します。

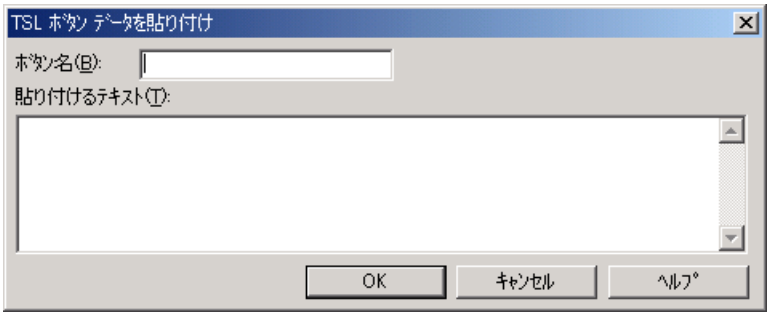
1 ［**表示**］＞［**ツールバーのカスタマイズ**］を選択します。［ユーザ定義ツールバーのカスタマイズ］ダイアログ・ボックスが開きます。

2 ［**分類**］枠で［**TSL を貼り付け**］を選択します。



3 ［**コマンド**］枠でボタンの横のチェック・ボックスを選択し，ボタンを選択します。

4 [**修正**] をクリックして、[TSL ボタン データを貼り付け] ダイアログ・ボックスを開きます。



5 [**ボタン名**] ボックスで，ボタンの名前を入力します。

6 [**貼り付けるテキスト**] 枠に TSL ステートメントを入力します。

7 [**OK**] をクリックして，[TSL ボタンデータを貼り付け] ダイアログ・ボックスを閉じます。

ボタンの名前が [コマンド] 枠内の対応するボタンの横に表示されます。

8 [**OK**] をクリックして，[ユーザ定義ツールバーのカスタマイズ] ダイアログ・ボックスを閉じます。

これでユーザ定義ツールバーにボタンが追加されます。

TSL ステートメントを貼り付けるユーザ定義ツールバーのボタンを変更するには，次の手順を実行します。

1 [**表示**] > [**ツールバーのカスタマイズ**] を選んで，[ユーザ定義ツールバーのカスタマイズ] ダイアログ・ボックスを開きます。

2 [**分類**] 枠で，[**TSL を貼り付け**] を選択します。

3 [**コマンド**] 枠で，内容を変更したいボタンを選択します。

4 [**修正**] をクリックします。[TSL ボタンデータを貼り付け] ダイアログ・ボックスが開きます。

5 [**ボタン名**] ボックスと [**貼り付けるテキスト**] ボックスの一方または両方を変更します。

6 [**OK**] をクリックして，[TSL ボタンデータを貼り付け] ダイアログ・ボックスを閉じます。

7 [**OK**] をクリックして，[ユーザ定義ツールバーのカスタマイズ] ダイアログ・ボックスを閉じます。ユーザ定義ツールバーのボタンが変更されます。

TSL ステートメントを貼り付けるユーザ定義ツールバーのボタンを削除するには，次の手順を実行します。

1 [**表示**] > [**ツールバーのカスタマイズ**] を選択して，[ユーザ定義ツールバーのカスタマイズ] ダイアログ・ボックスを開きます。

2 [**分類**] 枠で，[**TSL を貼り付け**] を選択します。

3 [**コマンド**] 枠で，ボタンの横のチェック・ボックスをクリアします。

4 [**OK**] をクリックして [ユーザ定義ツールバーのカスタマイズ] ダイアログ・ボックスを閉じます。ユーザ定義ツールバーからボタンが削除されます。

TSL ステートメントを実行するボタンの追加

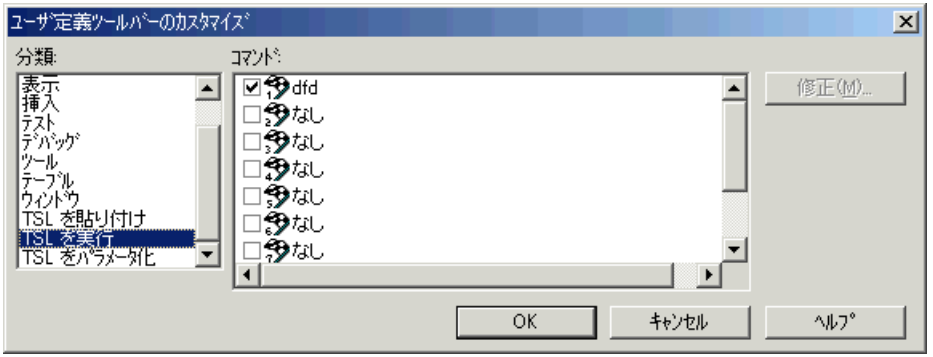
頻繁に使う TSL ステートメントを実行するボタンを [ユーザ定義ツールバー] に追加できます。

TSL ステートメントを実行するボタンを [ユーザ定義ツールバー] に追加するには，次の手順を実行します。

1 [**表示**] > [**ツールバーのカスタマイズ**] を選択します。

[ユーザ定義ツールバーのカスタマイズ] ダイアログ・ボックスが開きます。

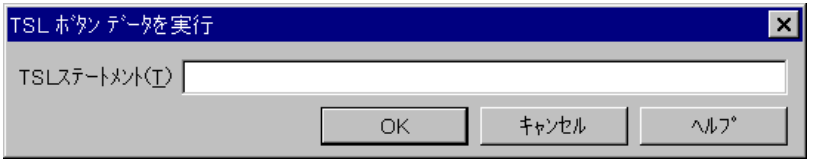
2 [**分類**] 枠で [**TSL を実行**] を選択します。



3 [**コマンド**] 枠で，ボタンの横のチェック・ボックスを選択し，ボタンを選択します。

4 [**修正**] をクリックします。

[TSL ボタン データを実行] ダイアログ・ボックスが開きます。



5 [**TSL ステートメント**] ボックスに，TSL ステートメントを入力します。

6 [**OK**] をクリックして，[TSL ボ実行ボタンのデータ] ダイアログ・ボックスを閉じます。

TSL ステートメントが [コマンド] 枠の中の対応するボタンの横に表示されます。

7 [**OK**] をクリックして，[ユーザ定義ツールバーのカスタマイズ] ダイアログ・ボックスを閉じます。ボタンがユーザ定義ツールバーに追加されます。

ユーザ定義ツールバー上の TSL ステートメントを実行するボタンを変更するには，次の手順を実行します。

1 [**表示**] > [**ツールバーのカスタマイズ**] を選択して，[ユーザ定義ツールバーのカスタマイズ] ダイアログ・ボックスを開きます。

2 [**分類**] 枠で [**TSL を実行**] を選択します。

3 [**コマンド**] 枠で，内容を変更したいボタンを選択します。

4 [**修正**] をクリックします。[TSL ボタン データを実行] ダイアログ・ボックスが開きます。

5 [**TSL ステートメント**] ボックスに望みの変更を入力します。

6 [**OK**] をクリックして [TSL ボタン データを実行] ダイアログ・ボックスを閉じます。

7 [**OK**] をクリックして [ユーザ定義ツールバーのカスタマイズ] ダイアログ・ボックスを閉じます。ユーザ定義ツールバーのボタンが変更されます。

ユーザ定義ツールバーから TSL ステートメントを実行するボタンを削除するには，次の手順を実行します。

1 ［**表示**］＞［**ツールバーのカスタマイズ**］を選択して，［ユーザ定義ツールバーのカスタマイズ］ダイアログ・ボックスを開きます。

2 ［**分類**］枠で［**TSL を実行**］を選択します。

3 ［**コマンド**］枠で，ボタンの横のチェック・ボックスをクリアします。

4 ［**OK**］をクリックして，［ユーザ定義ツールバーのカスタマイズ］ダイアログ・ボックスを閉じます。ボタンがユーザ定義ツールバーから削除されます。

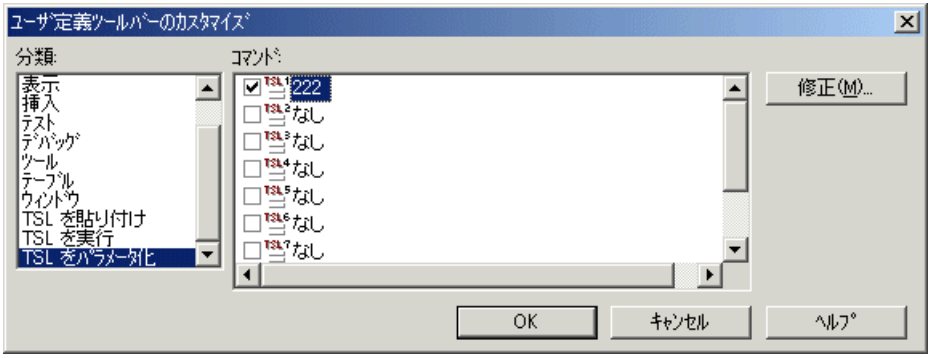
TSL ステートメントをパラメータ化するボタンの追加

頻繁に使う TSL ステートメントを簡単にパラメータ化できるようにするボタンをユーザ定義ツールバーに追加し，それらをテスト・スクリプトに貼り付けたり，実行することができます。

TSL ステートメントをパラメータ化するボタンをユーザ定義ツールバーに追加するには，次の手順を実行します。

1 ［**表示**］＞［**ツールバーのカスタマイズ**］を選択して，［ユーザ定義ツールバーのカスタマイズ］ダイアログ・ボックスを開きます。

2 ［**分類**］枠で［**TSL をパラメータ化**］を選択します。



3 ［**コマンド**］枠でボタンの横のチェック・ボックスを選択し，ボタンを選択します。

4 ［**修正**］をクリックします。

［TSL ボタン データをパラメータ化］ダイアログ・ボックスが開きます。

5 ［TSL ステートメント］ボックスに TSL 関数の名前を入力します。パラメータを入力する必要はありません。例えば，**list_select_item** のように入力します。

6 ［**OK**］をクリックして，［TSL ボタン データをパラメータ化］ダイアログ・ボックスを閉じます。TSL ステートメントは［コマンド］表示枠の中の対応するボタンの横に表示されます。

7 ［**OK**］をクリックして［ユーザ定義ツールバーのカスタマイズ］ダイアログ・ボックスを閉じます。ボタンがユーザ定義ツールバーに追加されます。

ユーザ定義ツールバー上の TSL ステートメントをパラメータ化するボタンを変更するには，次の手順を実行します。

1 ［**表示**］＞［**ユーザ定義ツールバーのカスタマイズ**］を選択して，［ユーザ定義ツールバーのカスタマイズ］ダイアログ・ボックスを開きます。

2 ［**分類**］枠で，［**TSL をパラメータ化**］を選択します。

3 ［**コマンド**］枠で，内容を変更したいボタンを選択します。

4 ［**修正**］をクリックします。［TSL ボタン データをパラメータ化］ダイアログ・ボックスが開きます。

5 ［**TSL ステートメント**］ボックスに望みの変更を入力します。

6 ［**OK**］をクリックして，［TSL ボタン データをパラメータ化］ダイアログ・ボックスを閉じます。

7 ［**OK**］をクリックして，［ユーザ定義ツールバーのカスタマイズ］ダイアログ・ボックスを閉じます。ユーザ定義ツールバー上のボタンが変更されます。

ユーザ定義ツールバー上の TSL ステートメントをパラメータ化するボタンを削除するには，次の手順を実行します。

1 ［**表示**］＞［**ツールバーのカスタマイズ**］を選択して，［ユーザ定義ツールバーのカスタマイズ］ダイアログ・ボックスを開きます。

2 ［**分類**］枠で［**TSL をパラメータ化**］を選択します。

3 ［**コマンド**］枠で，ボタンの横のチェック・ボックスをクリアします。

4 ［**OK**］をクリックして，［ユーザ定義ツールバーのカスタマイズ］ダイアログ・ボックスを閉じます。ユーザ定義ツールバーからボタンが削除されます。

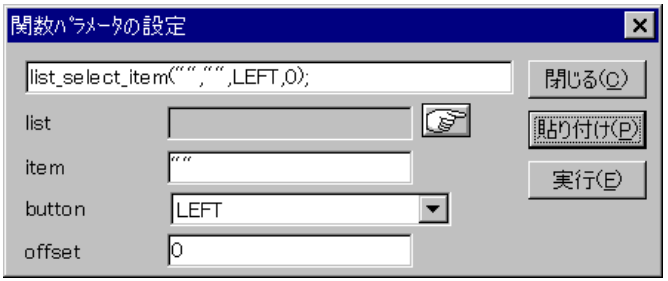
ユーザ定義ツールバーの使い方

標準ではユーザ定義ツールバーは非表示になっています。［**表示**］メニューで選択して，表示させることができます。ユーザ定義ツールバー上のコマンドを実行するには，そのコマンドに対応するボタンをクリックします。また，ユーザ定義ツールバー上にあるのと同じ TSL ベースのコマンドには，［**挿入**］メニューで選択してもアクセスできます。

ユーザ定義ツールバーが「フローティング」ツールバーのときは，テストの記録中に WinRunner を最小化していても，ツールバーは開いたままになります。詳細については，を参照してください。

TSL ステートメントのパラメータ化

ユーザ定義ツールバー上の，パラメータ化される TSL ステートメントを表しているボタンをクリックすると，［関数パラメータの設定］ダイアログ・ボックスが開きます。



［関数パラメータの設定］ダイアログ・ボックスの外見は，特定の TSL 関数によって要求するパラメータによって異なります。例えば，**list_select_item** 関数には 4 つのパラメータ，*List*，*Item*，*Button*，および *Offset* があります。各パラメータごとに以下のように値を定義します。

- *List* パラメータの値を定義するには，指差し型ボタンをクリックします。WinRunner が最小化されアイコンになり，ヘルプ・ウィンドウが開き，マウス・ポインタが指差し型になります。アプリケーションの中のリストをクリックします。
- *Item* パラメータの値を定義するには，対応するボックスに入力します。
- *Button* パラメータの値を定義するには，リストから選択します。
- *Offset* パラメータの値を定義するには，対応するボックスに入力します。

メニュー・バー上の TSL ステートメントへのアクセス

ユーザ定義ツールバーに追加したすべての TSL ステートメントには［**挿入**］メニューからアクセスできます。

メニューから TSL ステートメントを変更するには，次の手順を実行します。

- TSL ステートメントを貼り付けるには［**挿入**］＞［**TSL の貼り付け**］＞［(**TSL ステートメント**)］とクリックします。
- TSL ステートメントを実行するには［**挿入**］＞［**TSL を実行**］＞［(**TSL ステートメント**)］とクリックします。
- TSL ステートメントをパラメータ化するには［**挿入**］＞［**TSL のパラメータ化**］＞［(**TSL ステートメント**)］とクリックします。

WinRunner ソフトキーの構成設定

WinRunner のいくつかのコマンドは，ソフトキーを使って実行できます。WinRunner は WinRunner ウィンドウが画面上のアクティブ・ウィンドウでない場合や，最小化されているときでも，ソフトキー・コマンドを実行できます。

テストしているアプリケーションがあらかじめ WinRunner のために定義されていたソフトキーの組み合わせを使っている場合，WinRunner のソフトキー構成設定ユーティリティを使って，WinRunner のソフトキーの組み合わせを再定義できます。

WinRunner のソフトキーの標準の設定

次の表では標準のソフトキーの構成とそれらの機能を示します。

コマンド	標準のソフトキーの組み合わせ	機能
RECORD	F2	テストの記録を開始します。記録中，このソフトキーはコンテクスト・センシティブ・モードとアナログ・モードを切り換えます。
CHECK GUI FOR SINGLE PROPERTY	右 Alt + F12	GUI オブジェクトの単数のプロパティを検査します。
CHECK GUI FOR OBJECT/WINDOW	右 Ctrl+ F12	オブジェクトまたはウィンドウの GUI チェックポイントを作成します。
CHECK GUI FOR MULTIPLE OBJECTS	F12	［GUI チェックポイント作成］ダイアログ・ボックスを開きます。
CHECK BITMAP OF OBJECT/WINDOW	左 Ctrl+ F12	オブジェクトまたはウィンドウのビットマップをキャプチャします。
CHECK BITMAP OF SCREEN AREA	左 Alt + F12	領域のビットマップをキャプチャします。
CHECK DATABASE (DEFAULT)	右 Ctrl + F9	データベースの全ての内容を対象にチェックを作成します。
CHECK DATABASE (CUSTOM)	右 Alt + F9	データベースのカラム数，行数および指定した情報を検査します。
SYNCHRONIZE OBJECT/WINDOW PROPERTY	左 Ctrl + F11	WinRunner に特定のオブジェクトまたはウィンドウのビットマップが現れるのを待つように指示します。
SYNCHRONIZE BITMAP OF OBJECT/WINDOW	左 Alt + F11	WinRunner に特定の領域のビットマップが現れるのを待つように指示します。

コマンド	標準のソフトキーの組み合わせ	機能
GET TEXT FROM OBJECT/WINDOW	F11	オブジェクトまたはウィンドウのテキストをキャプチャします。
GET TEXT FROM SCREEN AREA	左 Ctrl + F11	指定された領域からテキストをキャプチャし，テスト・スクリプトに **get_text** ステートメントを追加します。
INSERT FUNCTION FOR OBJECT/WINDOW	F8	GUI オブジェクトの TSL 関数を挿入します。
INSERT FUNCTION FROM FUNCTION GENERATOR	F7	［関数ジェネレータ］ダイアログ・ボックスを開きます。
RUN FROM TOP	左 Ctrl + F5	テストを先頭から実行します。
RUN FROM ARROW	左 Ctrl + F7	スクリプトの矢印によって示された行からテストを実行します。
STEP	F6	テスト・スクリプトの現在の行だけを実行します。
STEP INTO	左 Ctrl + F8	［ステップ］と似ています。ただし，現在の行がテストまたは関数を呼び出すと，呼び出し先のテストまたは関数が WinRunner ウィンドウに現れますが，実行はされません。
STEP TO CURSOR	左 Ctrl + F9	テストの矢印によって示された行から挿入ポイントによって示された行までテストを実行します
PAUSE	PAUSE	あらかじめ解釈されている TSL ステートメントがすべて実行されるとテストを停止します。この場所から［矢印から実行］コマンドまたは［矢印から実行］ソフトキーを使って実行を再開できます。

コマンド	標準のソフトキーの組み合わせ	機能
STOP	左 Ctrl + F3	テストの記録または実行を停止します。
MOVE LOCATOR	左 Alt + F6	move_locator_abs ステートメントを画面ポインタの現在の位置（単位：ピクセル）で記録します。

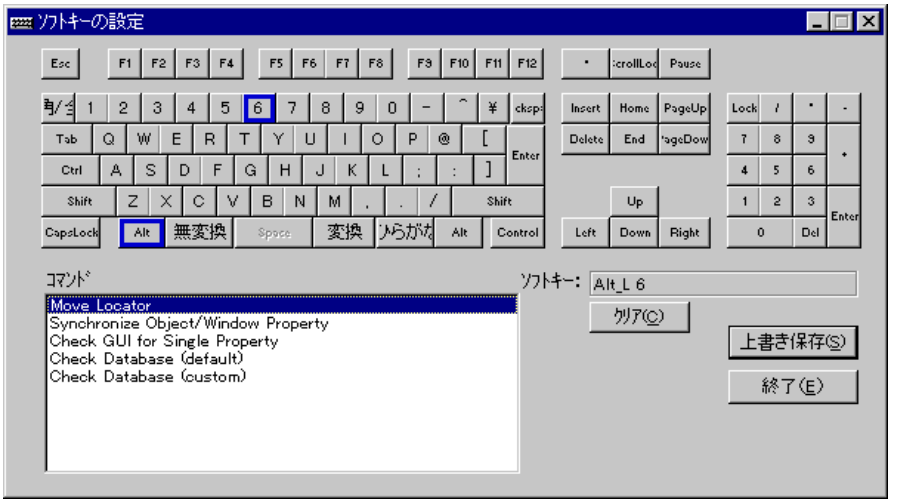
WinRunner のソフトキーの再定義

［ソフトキーの設定］ダイアログ・ボックスに現在のソフトキーの割り当てのリストが表示され，キーボードのイメージが表示されます。ソフトキーの設定を変更するには，ダイアログ・ボックスに表示された新しいキーの組み合わせをクリックします。

WinRunner のソフトキーの設定を変更するには，次の手順を実行します。

1 ［**スタート**］＞［**プログラム**］＞［**WinRunner**］＞［**Softkey Configuration**］を選択します。［ソフトキーの設定］ダイアログ・ボックスが開きます。

［コマンド］表示枠のリストに WinRunner のすべてのソフトキー・コマンドが表示されます。



2 変更したいコマンドをクリックします。現在のソフトキーの定義が［**ソフトキー**］ボックスに現れます。それらのキーはキーボード上で強調表示されます。

3 定義したい新しいキーまたは組み合わせをクリックします。これで，新しい定義が［**ソフトキー**］フィールドに表示されます。

すでに使われている定義や不正なキーの組み合わせを選択した場合にはエラー・メッセージが表示されます。別のキーまたは組み合わせをクリックしてください。

4 ［**上書き保存**］をクリックして，変更を保存しダイアログ・ボックスを閉じます。新しいソフトキー構成設定は WinRunner を起動すると有効になります。

第 43 章

テスト・スクリプトからのテスト・オプションの設定

WinRunner では，テスト・スクリプトでテスト・オプションを設定または取得し，テストの記録や実行の方法を制御することができます。

本章では，以下の項目について説明します。

- テスト・スクリプトからのテスト・オプションの設定について
- getvar を使ったテスト・オプションの取得
- setvar と getvar によるテスト実行の制御
- テスト・スクリプトのテスト・オプションの使用

テスト・スクリプトからのテスト・オプションの設定について

WinRunner のテスト・オプションは，テスト・スクリプトの記録法方や，テストの実行方法に影響を与えます。例えば，WinRunner のテストの実行速度やキーボード入力の記録方法などを設定できます。

テスト・スクリプト内でテスト・オプションの値の設定と取得を行えます。テスト・オプションの値を設定するには **setvar** 関数を使用します。現在のテスト・オプションの値を取得するには **getvar** 関数を使用します。テスト・スクリプトで **setvar** と **getvar** のステートメントを組み合わせて使い，WinRunner のテストの実行を制御できます。これらの関数を使って全部のテストや単独のテスト，またはテスト内の特定部分のテスト・オプションを表示したり設定したりできます。また，起動テスト・スクリプト内でこれらの関数を使って環境変数を設定することもできます。

ほとんどのテスト・オプションは［一般オプション］ダイアログ・ボックスを使って設定することもできます。［一般オプション］ダイアログ・ボックスを使ったテスト・オプションの設定については，第 22 章「グローバル・テスト・オプションの設定」を参照してください。

setvar を使ったテスト・オプションの設定

テスト・スクリプト内で，テスト・オプションの値を設定するには，**setvar** 関数を使用します。この関数の構文は次のとおりです。

setvar ("*testing_option***", "***value***");**

この関数の *testing_option* に，次のいずれかを指定できます。

attached_text_area	enum_descendent_toplevel	searchpath
attached_text_search_radius	fontgrp	silent_mode
beep	item_number_seq	single_prop_check_fail
capture_bitmap	List_item_separator	speed
cs_run_delay	Listview_item_separator	sync_fail_beep
cs_fail	min_diff	synchronization_timeout
delay_msec	mismatch_break	tempdir
drop_sync_timeout	rec_item_name	timeout_msec
email_service	rec_owner_drawn	Treeview_path_separator

例えば，以下の **setvar** ステートメントを実行したとします。

```
setvar ("mismatch_break", "off");
```

WinRunner は ***mismatch_break*** テスト・オプションを無効にします。オプションの設定は，**setvar** ステートメントまたは［一般オプション］ダイアログ・ボックスの［**実行**］カテゴリにある［**検証が失敗したら停止する**］チェック・ボックスを使って変えるまでは，現在のテスト・セッションを通して有効です。

setvar 関数を使うとテスト・オプションはグローバルに変更され，［一般オプション］ダイアログ・ボックスの中の設定にも反映されます。また，**setvar** 関数を使って特定のテスト，または特定のテストの一部分に対するテスト・オプションを設定することもできます。

グローバルな値を変更せずに，**setvar** 関数を使って現在のテストの変数を変更するには，次のようにします。変数の既存の値を別に保存しておいて，後で元に戻します。

例えば，**delay_msec** テスト・オプションを一時的に 20,000 に変更する場合，テスト・スクリプトの先頭に以下のステートメントを挿入します。

```
# 元の「delay_msec」テスト・オプションの値を保存する。
old_delay = getvar ("delay_msec") ;
setvar ("delay_msec", "20,000") ;
```

テストの終わりに ***delay*** テスト・オプションを元の値に戻すには，テスト・スクリプトの末尾に以下を挿入します。

```
# 「delay_msec」テスト・オプションを元の値に戻す。
setvar (“delay_msec”, old_delay);
```

注：テスト・オプションには，WinRunner によって設定され，**setvar** でも［一般オプション］ダイアログ・ボックスでも変更できないものがあります。例えば，testname オプションの値は常に現在のテストの名前になります。**getvar** を使用して読み取り専用の値を取得することはできます。詳細については，869 ページ「getvar を使ったテスト・オプションの取得」を参照してください。

---

getvar を使ったテスト・オプションの取得

特定のテスト・オプションの現在の値を取得するには，**getvar** 関数を使用します。**getvar** 関数は読み取り専用の関数のため，取得したテスト・オプションの値を変更できません（テスト・スクリプト内でテスト・オプションの値を変更する場合は先に述べた方法で **setvar** を使用します）。このステートメントの構文は次のとおりです。

user_variable = **getvar (**"*testing_option*"**);**

この関数の *testing_option* に対しては，次のいずれかを指定できます。

attached_text_area	key_editing	sync_fail_beep
attached_text_search_radius	line_no	synchronization_timeout
batch	List_item_separator	qc_connection
beep	Listview_item_separator	qc_cycle_name
capture_bitmap	min_diff	qc_database_name
cs_fail	mismatch_break	qc_log_dirname
cs_run_delay	rec_item_name	qc_log_dirname
curr_dir	rec_owner_drawn	qc_server_name
delay_msec	result	qc_test_instance
drop_sync_timeout	runmode	qc_test_run_id
email_service	searchpath	qc_user_name
enum_descendent_toplevel	shared_checklist_dir	tempdir
exp	single_prop_check_fail	testname
fontgrp	silent_mode	timeout_msec
item_number_seq	speed	Treeview_path_separator

以下に例を示します。

```
currspeed = getvar ("speed");
```

これは，現在の実行速度の値をユーザ定義変数 currspeed に代入しています。

setvar と getvar によるテスト実行の制御

getvar ステートメントと **setvar** ステートメントを組み合わせて，グローバルな設定を変更せずにテストの実行を制御できます。次に示すテスト・スクリプトの一部では，WinRunner ビットマップ Img1 を検査しています。**getvar** ステートメントで **timeout_msec** および **delay_msec** の値を取得し，次に実行する win_check_bitmap ステートメントのために **setvar** を使用してこれらの変数に値を代入しています。ウィンドウの検査が終わると **setvar** を使用して，timeout と delay の値を元に戻します。

```
t = getvar ("timeout_msec");
d = getvar ("delay_msec");
setvar ("timeout_msec", 30000);
setvar ("delay_msec", 3000);
win_check_bitmap ("calculator", Img1, 2, 261,269,93,42);
```

```
setvar ("timeout_msec", t);
setvar ("delay_msec", d);
```

注：起動のテスト・スクリプト内で **setvar** 関数と **getvar** 関数を使用して，特定の WinRunner セッションに対して環境変数を変更することができます。詳細については，第 45 章「特殊な構成の初期化」を参照してください。

テスト・スクリプトのテスト・オプションの使用

この節ではテスト・スクリプト内で **setvar** 関数や **getvar** 関数で使える WinRunner のテスト・オプションについて説明します。設定を使用できる場合，あるいはダイアログ・ボックスから対応するオプションを表示できる場合は，その旨を示します。

attached_text_area

このオプションは WinRunner が付属テキストの検索を開始する GUI オブジェクト上の位置を指定します。

指定可能な値：

値	GUI オブジェクト上の点
Default	一般的な（英語スタイル）ウィンドウの場合は左上角。RTL スタイル（WS_EX_BIDI_CAPTION）ではウィンドウの右上角。
Top-Left	左上角
Top	上側の 2 つの角の間の中点
Top-Right	右上角
Right	右側の 2 つの角の間の中点
Bottom-Right	右下角
Bottom	下側の 2 つの角の間の中点

値	GUI オブジェクト上の点
Bottom-Left	左下角
Left	左側の 2 つの角の間の中点

注：上記の指定可能な値はすべてテキスト文字列です。

このオプションは **setvar** 関数と **getvar** 関数で使えます。

このオプションは，［一般オプション］ダイアログ・ボックスの［**記録開始**］カテゴリの［**付属テキスト**］の［**指定検索範囲**］ボックスでも設定できます。これについては，529 ページ「記録オプションの設定」を参照してください。

注：テストを実行するときは，記録したときと同じ付属テキスト・オプションの値を使用してください。同じ値を使用しないと WinRunner が GUI オブジェクトを認識できないことがあります。

WinRunner の従来のバージョンでは検索範囲を設定できませんでした。従来のバージョンの WinRunner は付属テキストを検索する際，［Default］を元に検索していました。従来バージョンとの互換性が重要である場合は［Default］を選んでください。

---

attached_text_search_radius

このオプションは GUI オブジェクト上の指定された位置から WinRunner が静的テキスト・オブジェクトを探す半径を指定します。

指定可能な値：3 - 300（ピクセル）

このオプションは **setvar** 関数と **getvar** 関数で使えます。

このオプションは，［一般オプション］ダイアログ・ボックスの［**記録開始**］カテゴリにある［**付属テキスト**］の［**検索範囲**］ボックスでも設定できます。これについては，529 ページ「記録オプションの設定」を参照してください。

注：テストを実行するときは，同じ付属テキスト・オプションに対して記録したときと値を使用してください。同じ値を使用しないと WinRunner が GUI オブジェクトを認識できないことがあります。

---

batch

このオプションは WinRunner がバッチ・モードで実行しているかどうかを示します。バッチ・モードでは，テストを無人で実行できるよう，テスト実行中のメッセージ表示が抑制されます。WinRunner はバッチ・モードで実行したテストの期待結果や実際の結果を単独のディレクトリに保存します。それらの結果は単独の［テスト結果］ウィンドウに表示されます。バッチ・テストのオプションについては，第 36 章「バッチ・テストの実行」を参照してください。

例えば，**set_window** ステートメントがテスト・スクリプト内にない場合，WinRunner は指定されたウィンドウを発見できません。このオプションがオンでテストがバッチ・モードで実行された場合，WinRunner は［テスト結果］ウィンドウにエラーを表示して，テスト・スクリプト内の次のステートメントを実行します。このオプションがオフであり，テストがバッチ・モードではなく通常のモードで実行されると WinRunner はテストを一時停止し，［実行］ウィザードを開いてユーザがそのウィンドウを指定できるようにします。

このオプションは **getvar** 関数で使えます。

指定可能な値：on，off（テスト文字列）

このオプションは，［一般オプション］ダイアログ・ボックスの［実行］タブ内の［**バッチ モードで実行**］チェック・ボックスを使って設定することもできます。これについては，529 ページ「記録オプションの設定」を参照してください。

このオプションは，**-batch** コマンドライン・オプションを使用して設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

注：バッチ・モードでテストを実行すると，テストは自動的にサイレント・モードで実行されます。***silent_mode*** テスト・オプションの詳細については，885 ページを参照してください。

---

beep

このオプションはウィンドウを検査するたびに WinRunner がビープ音を鳴らすかどうかを指定します。

このオプションは **setvar** 関数と **getvar** 関数で使えます。

指定可能な値：on，off（テスト文字列）

このオプションは［一般オプション］ダイアログ・ボックスの［**実行**］＞［**設定**］カテゴリにある［**ウィンドウのチェック時にビープ音を鳴らす**］チェック・ボックスを使用して設定することもできます。詳細については，549 ページ「実行の設定オプションの設定」を参照してください。

このオプションは，*-beep* コマンドライン・オプションを使用して設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

capture_bitmap

このオプションは，チェックポイントの失敗時に WinRunner がビットマップをキャプチャするかどうかを決定します。このオプションが on の場合，WinRunner は［一般オプション］ダイアログ・ボックスの［**実行**］＞［**設定**］カテゴリの設定を使用して，ビットマップのキャプチャ領域を決定します。

このオプションは，**setvar** および **getvar** 関数と一緒に使用できます。

指定可能な値：on, off（テキスト文字列）

このオプションは，［一般オプション］ダイアログ・ボックスの［**実行**］＞［**設定**］カテゴリにある［**検証失敗の時，ビットマップをキャプチャする**］チェックボックスを使用して設定することもできます。詳細については，549 ページ「実行の設定オプションの設定」を参照してください。

このオプションは，*-capture_bitmap* コマンドライン・オプションを使用して設定することもできます。詳細については第 37 章「コマンドラインからのテストの実行」を参照してください。

cs_fail

このオプションはコンテキスト・センシティブ・エラーが発生した時に，WinRunner がテストを失敗するかどうかを決定します。コンテキスト・センシティブ・エラーは，WinRunner が GUI オブジェクトを特定できないことが原因で発生します。

例えば，存在しないウィンドウを指定した **set_window** ステートメントを含むテストを実行すると，コンテキスト・センシティブ・エラーが発生します。またコンテキスト・センシティブ・エラーはウィンドウの名前が明確でない場合にも発生します。コンテキスト・センシティブ関数の詳細については，「**TSL リファレンス**」を参照してください。

このオプションは **setvar** 関数と **getvar** 関数で使えます。

指定可能な値：1,0

このオプションは，［一般オプション］ダイアログ・ボックス［実行］＞［設定］カテゴリにある［**コンテキスト センシティブ エラーが発生したらテストを失敗とする**］ボックスを使って設定することもできます。これについては，529 ページ「記録オプションの設定」を参照してください。

このオプションは，*-cs_fail* コマンドライン・オプションを使って設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

cs_run_delay

このオプションはテスト実行中に WinRunner が次のコンテキスト・センシティブ・ステートメントを実行するまでの待ち時間（単位：ミリ秒）を設定します。

このオプションは **setvar** 関数と **getvar** 関数で使えます。

指定可能な値：0 以上の数値

このオプションは，［一般オプション］ダイアログ・ボックスの［**実行**］＞［**同期化**］カテゴリにある［**CS ステートメント実行間の遅延**］ボックスを使って設定することもできます。これについては，529 ページ「記録オプションの設定」を参照してください。

このオプションは，*- cs_run_delay* コマンドライン・オプションを使って設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

curr_dir

このオプションはテストの現在の作業フォルダを示します。

このオプションは **getvar** 関数で使えます。

テストの現在の作業フォルダの位置は，［テストのプロパティ］ダイアログ・ボックスの［現在のテスト］タブにある［**現在のフォルダ**］ボックスでも参照できます。これについては，529 ページ「記録オプションの設定」を参照してください。

delay_msec

このオプションは，コンテキスト・センシティブ・チェックポイントまたは同期ポイントのためのキャプチャを行う前にウィンドウが安定したことを判定するために使われるサンプリングの間隔（単位：ミリ秒）を設定します。安定したと宣言するためには，ウィンドウは 2 回の連続するサンプリングの間で変化があってはなりません。このサンプリングは，ウィンドウが安定するかタイムアウトになるまで（**timeout_msec** テスト・オプションで設定します）続行されます（このオプションは，秒単位で設定されていた従来の「**delay**」に相当します）。

例えば，遅延が 2 秒でタイムアウトが 10 秒の場合，WinRunner はテストの際にアプリケーションのウィンドウを，2 つの検査が同じ結果を生成するか，10 秒経過するまで 2 秒ごとに検査します。値を 0 に設定すると，すべてのビットマップ・チェックが行われなくなります。

このオプションは **setvar** 関数と **getvar** 関数で使えます。

指定可能な値：0 以上の数値

注：このオプションは 20 ～ 30 ミリ秒以内の精度です。

このオプションは，［一般オプション］ダイアログ・ボックスの［**実行**］＞［**同期化**］カテゴリにある［**ウィンドウ同期までの遅延**］ボックスでも設定できます。これについては，529 ページ「記録オプションの設定」を参照してください。

また，このオプションは，*- delay_msec* コマンドライン・オプションを使って設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

drop_sync_timeout

このオプションは，最初の同期化が失敗した場合，WinRunner に対して同期化タイムアウト（**timeout_msec** オプションで定義されます）を最小化するかどうかを指定します。

指定可能な値：on, off (text strings)

このオプションは **getvar** 関数と **setvar** 関数で使えます。

このオプションは［一般オプション］ダイアログ・ボックスの［実行］＞［同期化］カテゴリにある［**失敗したら同期化タイムアウトを破棄する**］チェック・ボックスを使って設定することもできます。詳細については，529 ページ「記録オプションの設定」を参照してください。

email_service

WinRunner が電子メール送信オプション（チェックポイントの失敗，テストの失敗，テストの完了レポートのメール通知，およびテスト内の **email_send_msg** ステートメントを含む）をアクティブにするかどうかを決定します。

指定可能な値：on, off (text strings)

このオプションは，**getvar** および **setvar** 関数を使用して設定することができます。

このオプションは，［一般オプション］ダイアログ・ボックスの［**通知**］＞［**電子メール**］カテゴリで［**電子メールのサービスを有効にする**］を使用して設定することもできます。詳細については，563 ページ「電子メール通知オプションの設定」を参照してください。

-email_service コマンドライン・オプションを使用してもこのオプションを設定できます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

enum_descendent_toplevel

このオプションは WinRunner が，親がウィンドウ以外のオブジェクトである子オブジェクトのコントロール（オブジェクト）を記録して，テストを実行するときにこれらのコントロールを特定するかどうかを決定します。

指定可能な値：1,0

このオプションは **setvar** 関数と **getvar** 関数で使えます。

このオプションは，［一般オプション］ダイアログ・ボックスの［記録開始］カテゴリにある［**子ウィンドウも含める**］チェック・ボックスを使って設定することもできます。詳細については，529 ページ「記録オプションの設定」を参照してください。

exp

このオプションは，現在実行中のテストで使用する期待結果フォルダの完全パス名を示します。

このオプションは **getvar** 関数で使えます。

期待結果フォルダの完全パスは，［テストのプロパティ］ダイアログ・ボックスの［現在のテスト］タブにある［**期待結果フォルダ**］ボックスでも参照できます。これについては，529 ページ「記録オプションの設定」を参照してください。

また，このオプションは，**-exp** コマンドライン・オプションを使って設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

fontgrp

（標準設定のテキスト認識ではなく）イメージ・テキスト認識（541 ページ「テキスト認識オプションの設定」で解説されています）を使うために，アクティブ・フォント・グループを選択しなければなりません。フォント・グループの詳細については，350 ページ「WinRunner によるフォントの学習」を参照してください。

このオプションは **setvar** 関数と **getvar** 関数で使えます。

指定可能な値：任意のテキスト文字列

このオプションは，［一般オプション］ダイアログ・ボックスの［**記録**］＞［**テキスト認識**］タブにある［**フォント グループ**］ボックスを使って設定することもできます。これについては，541 ページ「テキスト認識オプションの設定」を参照してください。

また，このオプションは，**-fontgrp** コマンドライン・オプションを使って設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

item_number_seq

このオプションは，テスト・スクリプトの中で，リスト，リスト・ビュー，またはツリー・ビュー項目をインデックス番号で指定するときに使う文字列を定義します。

このオプションは **setvar** 関数と **getvar** 関数で使えます。

指定可能な値：任意のテキスト文字列

このオプションは，［一般オプション］ダイアログ・ボックスの［**記録開始**］＞［**スクリプト形式**］カテゴリにある［**続く文字が数字であることを示す文字列**］ボックスを使って設定することもできます。これについては，537 ページ「スクリプト形式オプションの設定」を参照してください。

key_editing

このオプションは，テスト・スクリプト内で WinRunner に簡潔な **type**，**win_type**，や **obj_type** ステートメントを生成させるかどうかを指定します。

このオプションが有効な場合，入力キーを押下したり，離したりした結果だけを表す，より簡潔な **type**，**win_type**，**obj_type** ステートメントが生成されます。これによりテスト・スクリプトが読みやすくなります。

以下に例を示します。

```
obj_type (object, "A");
```

このオプションが無効な場合，WinRunner は各キーを押したり離したりするのをすべて記録します。以下に例を示します。

```
obj_type (object, "<kShift_L>-a-a+<kShift_L>+");
```

テストにおいて，キー入力の正確な順番が重要な場合は，key_editing を無効にします。

詳細については，「**TSL リファレンス**」の **type** 関数の説明を参照してください。

このオプションは **setvar** 関数と **getvar** 関数で使えます。

指定可能な値：on, off (text strings)

このオプションは，［一般オプション］ダイアログ・ボックスの［**記録開始**］＞［**スクリプト形式**］カテゴリで［**簡潔で読みやすい type ステートメントを生成する**］チェック・ボックスを選択して設定することもできます。これについては，537 ページ「スクリプト形式オプションの設定」を参照してください。

line_no

このオプションは，テスト・スクリプト内の実行矢印の現在の位置を行番号で示します。

このオプションは **getvar** 関数で使えます。

テスト・スクリプト内の現在の行番号は，［テストのプロパティ］ダイアログ・ボックスの［**現在のテスト**］タブにある［**現在の行番号**］ボックスを使って表示することもできます。これについては，508 ページ「現在のテスト設定の確認」を参照してください。

List_item_separator

このオプションは，リスト・ボックスまたはコンボ・ボックスの中の項目を区切るためにテスト・スクリプトに記録される文字列を定義します。

このオプションは **setvar** 関数と **getvar** 関数で使えます。

指定可能な値：任意のテキスト文字列

このオプションは，［一般オプション］ダイアログ・ボックスの［記録開始］＞［スクリプト形式］タブにある［**リストボックスまたはコンボ ボックスの項目を区切る文字列**］ボックスを使って設定することもできます。これについては，537 ページ「スクリプト形式オプションの設定」を参照してください。

Listview_item_separator

このオプションは，リスト・ビューまたはツリー・ビューの中の項目を区切るためにテスト・スクリプトに記録される文字列を定義します。

このオプションは **setvar** 関数と **getvar** 関数で使えます。

指定可能な値：任意のテキスト文字列

このオプションは，［一般オプション］ダイアログ・ボックスの［**記録開始**］＞［**スクリプト形式**］タブにある［**リストまたはツリー ビューの項目を区切る文字列**］ボックスを使って設定することもできます。これについては，537 ページ「スクリプト形式オプションの設定」を参照してください。

min_diff

このオプションはビットマップを不一致と判定するピクセル数のしきい値を定義します。この値が 0 に設定されている場合には，1 ピクセルでも違いがあれば，ビットマップは不一致であると判定されます。

このオプションは **setvar** 関数と **getvar** 関数で使えます。

指定可能な値：0 以上の数値

このオプションは，［一般オプション］ダイアログ・ボックスの［実行］タブにある［**ビットマップ間の違いを差異として認識するしきい値**］ボックスを使って設定することもできます。これについては，529 ページ「記録オプションの設定」を参照してください。

また，このオプションは，**-min_diff** コマンドライン・オプションを使って設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

mismatch_break

このオプションは，検証に失敗したり，検証モードで実行中のテストでコンテキスト・センシティブ・ステートメントの結果としてなんらかのメッセージが生成されたときに，WinRunner がテストの実行を中断してメッセージを表示するかどうかを指定します。このオプションは対話形式で作業するときにのみ使用します。

例えば，**set_window** ステートメントがテスト・スクリプトにない場合，WinRunner は指定されたウィンドウを見つけることができません。このオプションが選択されている場合，WinRunner はテストを中断して，［実行］ウィザードを開き，ユーザがウィンドウを見つけられるようにします。このオプションが選択されていない場合には，WinRunner は［テスト結果］ウィンドウにエラーを報告し，テスト・スクリプト内の次のステートメントに進みます。

このオプションは **setvar** 関数と **getvar** 関数で使えます。

指定可能な値：on，off (text strings)

このオプションは，［一般オプション］ダイアログ・ボックスの［**実行**］＞［**設定**］カテゴリにある［**検証が失敗したら停止する**］チェック・ボックスを使って設定することもできます。これについては，549 ページ「実行の設定オプションの設定」を参照してください。

また，このオプションは，**-mismatch_break** コマンドライン・オプションを使って設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

rec_item_name

このオプションは，WinRunner がリスト・ボックスやコンボ・ボックスの一意でない項目名を名前とインデックスのどちらに基づいて記録するかを指定します。

このオプションは **setvar** 関数と **getvar** 関数で使えます。

指定可能な値：1，0

このオプションは，［一般オプション］ダイアログ・ボックスの［記録開始］カテゴリにある［**一意ではないリストの項目を名前に基づいて記録する**］チェック・ボックスを選択して設定することもできます。これについては，529 ページ「記録オプションの設定」を参照してください。

また，このオプションは，**-rec_item_name** コマンドライン・オプションを使って設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

rec_owner_drawn

WinRunner はオーナー描画のボタンのクラスを識別できないので，自動的に汎用の「object」クラスに割り当てます。このオプションはすべてのオーナー描画ボタンをすべて標準ボタン・クラス（push_button，radio_button，あるいは check_button）に割り当てるようにします。

このオプションは **setvar** 関数や **getvar** 関数で使えます。

指定可能な値：object，push_button，radio_button，check_button（テキスト文字列）

このオプションは，［一般オプション］ダイアログ・ボックスの［記録開始］カテゴリにある［**ユーザ定義ボタンの記録方法**］ボックスで設定することもできます。これについては，529 ページ「記録オプションの設定」を参照してください。

result

このオプションは現在実行しているテストの検証結果フォルダの完全パス名を示します。

このオプションは **getvar** 関数で使えます。

検証結果フォルダの完全パスは，［テストのプロパティ］ダイアログ・ボックスの［**現在のテスト**］タブにある［**検証結果フォルダ**］ボックスでも参照できます。これについては，508 ページ「現在のテスト設定の確認」を参照してください。

runmode

このオプションは現在のテスト実行モードを示します。

このオプションは **getvar** 関数で使えます。

指定可能な値：verify, debug, update (text strings)

現在の実行モードは，［テストのプロパティ］ダイアログ・ボックスの［**現在のテスト**］タブにある［**実行モード**］ボックスでも参照できます。これについては，508 ページ「現在のテスト設定の確認」を参照してください。

searchpath

このオプションは WinRunner が呼び出し先のテストを検索するパスを示します。検索パスを定義すれば，call ステートメント内にテストの完全パスを指定する必要がありません。1 つのステートメントに複数の検索パスを設定することもできます。その場合は，検索パスの間にスペースを挿入します。長いファイル名の検索パスを複数設定する場合は，各パスの左右を山括弧 <> で囲みます。WinRunner は **getvar** ステートメントと **setvar** ステートメントで指定されているパスが複数ある場合には，指定されている順序に従って呼び出し先のテストを検索します。

このオプションは **setvar** 関数や **getvar** 関数で使えます。

このオプションは，［一般オプション］ダイアログ・ボックスの［フォルダ］カテゴリにある［**呼び出すテストの検索パス**］ボックスを使って設定することもできます。これについては，526 ページ「フォルダ・オプションの設定」を参照してください。

また，このオプションは，**-search_path** コマンドライン・オプションを使って設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

注：WinRunner を Quality Center と接続すると，WinRunner が呼び出し先のテストを検索する際のパスを Quality Center データベース内に指定できます。Quality Center データベース内の検索パスには前に [TD] を付けることができます。

---

shared_checklist_dir

このオプションは，WinRunner が GUI チェックポイントとデータベース・チェックポイントのための共有チェックリストを格納するフォルダを示します。テスト・スクリプトの中で，共有チェックリスト・ファイルは，**win_check_gui** ステートメント，**obj_check_gui** ステートメント，または **check_gui** ステートメントの中で，ファイル名の前に SHARED_CL をつけて示されます。共有 GUI チェック・リストの詳細については，143 ページ「GUI チェックリストの共有フォルダへの保存」を参照してください。共有データベース・チェック・リストの詳細については，305 ページ「データベース・チェックリストを共有フォルダに保存」を参照してください。新しいフォルダを指定した場合，変更内容を有効にするために WinRunner を再起動する必要があります。

このオプションは **getvar** 関数で使えます。

WinRunner が共有チェックリストを格納するフォルダの位置は，［一般オプション］ダイアログ・ボックスの［フォルダ］カテゴリにある［**共有チェックリスト**］ボックスを使って表示することもできます。これについては，526 ページ「フォルダ・オプションの設定」を参照してください。

silent_mode

このオプションは WinRunner がサイレント・モードで実行されているかどうかを示します。サイレント・モードでは，テストを無人で実行できるよう，テスト実行中のメッセージ表示が抑制されます。Quality Center からリモートでテストを実行するときは，テストが実行されているコンピュータで表示されるメッセージを監視する人が誰もいないので，サイレント・モードで実行しなければなりません。Quality Center からリモートでテストを実行する詳細については，第 48 章「テスト工程の管理」を参照してください。

このオプションは **setvar** および **getvar** 関数で使えます。

指定可能な値：on，off（テキスト文字列）

注：バッチ・モードでテストを実行すると，自動的にサイレント・モードで実行されます。バッチ・モードでのテストの実行の詳細については，第 36 章「バッチ・テストの実行」を参照してください。

---

single_prop_check_fail

このオプションは，**_check_info** ステートメントが失敗するとテスト実行を失敗させ，これらのステートメントに関するイベントを［テスト結果］ウィンドウに書き込みます（［**挿入**］＞［**GUI チェックポイント**］＞［**単数プロパティ**］コマンドを使用して **_check_info** ステートメントを作成できます）。

このオプションは **setvar** 関数と **getvar** 関数で使えます。

指定可能な値：1，0

check_info 関数の詳細については「**TSL リファレンス」を参照してください。**

このオプションは，［一般オプション］ダイアログ・ボックスの［実行］＞［設定］カテゴリにある［**単数プロパティチェックが失敗したらテストを失敗とする**］ボックスを使って設定することもできます。これについては，529 ページ「記録オプションの設定」を参照してください。

また，このオプションは，**-single_prop_check_fail** コマンドライン・オプションを使って設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

speed

このオプションはアナログ・モードでテストを実行する際の速度を設定します。

指定可能な値：normal，fast（テキスト文字列）

速度を［**通常**］に設定した場合，記録したときと同じ速度でテストを実行します。

速度を［**高速**］に設定した場合，アプリケーションが入力を受け取れる最大の速さで実行されます。

このオプションは **setvar** 関数と **getvar** 関数で使えます。

このオプションは，［一般オプション］ダイアログ・ボックスの［実行］カテゴリにある［**アナログ モードの実行速度**］オプションを使って設定することもできます。これについては，545 ページ「テストの実行オプションの設定」を参照してください。

また，このオプションは，**-speed** コマンドライン・オプションを使って設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

sync_fail_beep

このオプションは WinRunner が同期化に失敗したときにビープ音を鳴らすかどうかを指定します。

このオプションは **setvar** 関数と **getvar** 関数で使えます。

指定可能な値：1,0

このオプションは，［一般オプション］ダイアログ・ボックスの［**実行**］＞［**同期化**］カテゴリにある［**同期化に失敗したらビープ音を鳴らす**］チェック・ボックスを使って設定することもできます。これについては，554 ページ「実行の同期化オプションの設定」を参照してください。

注：このオプションは，主にテスト・スクリプトのデバッグを行う際に役立ちます。

注：テスト実行中に同期が頻繁に失敗する場合には，［一般オプション］ダイアログ・ボックスの［実行］＞［同期化］カテゴリにある［**同期メッセージ待機のタイムアウト**］オプションか，このオプションに対応する setvar 関数を使ってテスト・スクリプトの *synchronization_timeout* テスト・オプションの値を大きくすることを考えてみてください。

---

synchronization_timeout

このオプションは，WinRunner がテストの実行中にキーボードやマウスの入力が正しく行われたことを検証するまで待機するタイムアウト（単位：ミリ秒）を設定します。

このオプションは **setvar** 関数と **getvar** 関数で使えます。

指定可能な値：0 以上の数値

このオプションは，［一般オプション］ダイアログ・ボックスの［［実行］＞［同期化］カテゴリにある［**同期化メッセージ待機のタイムアウト**］ボックスを使って設定することもできます。これについては，554 ページ「実行の同期化オプションの設定」を参照してください。

注：テスト実行中に頻繁に同期化に失敗する場合は，このオプションの値を上げてください。

---

qc_connection

このオプションは，WinRunner が現在 Quality Center（以前の *td_connection* または ***test_directory***）に接続しているかどうかを示します。

このオプションは **getvar** 関数で使えます。

指定可能な値：on, off (text strings)

Quality Center には［**Quality Center への接続**］ダイアログ・ボックスまたは *-qc_connection* コマンドライン・オプションを使用して接続することができます。Quality Center への接続の詳細については，第 48 章「テスト工程の管理」を参照してください。

qc_cycle_name

このオプションは，テスト（以前の「cycle」）に対応する Quality Center テスト・セットの名前を表示します（以前の *td_cycle_name* または *cycle*）。

このオプションは **getvar** 関数で使えます。

WinRunner が Quality Center に接続されている間に WinRunner からテスト・セットを実行する際，［テスト実行］ダイアログ・ボックスを使って設定することもできます。詳細については，982 ページ「テスト・セット内のテストの実行」を参照してください。Quality Center からこのオプションを設定することもできます。詳細については，『**Mercury Quality Center ユーザーズ・ガイド**』を参照してください。

また，このオプションは，*qc_cycle_name* コマンドライン・オプションを使って設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

qc_database_name

このオプションは，WinRunner が現在接続している Quality Center プロジェクト・データベースの名前を示します（以前の *td_database_name*）。

このオプションは **getvar** 関数で使えます。

このオプションは［Quality Center への接続］ダイアログ・ボックスの［**プロジェクトへの接続**］を使って設定することもできます。このダイアログ・ボックスは［**ツール**］＞［**Quality Center への接続**］を選択して開きます。詳細については，第 48 章「テスト工程の管理」を参照してください。

また，このオプションは，*qc_database_name* コマンドライン・オプションを使って設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

qc_server_name

このオプションは，WinRunner が現在接続している Quality Center サーバの名前を示します（以前の *td_server_name*）。

このオプションは **getvar** 関数で使えます。

このオプションは［Quality Center への接続］ダイアログ・ボックスの［**サーバ**］ボックスを使って設定することもできます。このダイアログ・ボックスは［**ツール**］＞［**Quality Center への接続**］を選択して開きます。詳細については，第 48 章「テスト工程の管理」を参照してください。

また，このオプションは，*qc_server_name* コマンドライン・オプションを使って設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

qc_test_instance

このオプションは，Quality Center テスト・セットで現在開いており，実行されているインスタンスを表示します。

このオプションは **getvar** 関数と一緒に使用できます。

この値は，Quality Center に接続されている場合には，［テスト実行］ダイアログ・ボックスを使用して値を設定できます。詳細については，982 ページ「テスト・セット内のテストの実行」を参照してください。

qc_test_run_id

このオプションは，Quality Center テスト・セットで現在開いており，実行されているテストの実行名を表示します。

このオプションは **getvar** 関数と一緒に使用できます。

この値は，Quality Center に接続されている場合には，［テストの実行名］ボックスを使用して値を設定できます。詳細については，982 ページ「テスト・セット内のテストの実行」を参照してください。

qc_user_name

このオプションは，選択した Quality Center データ・ベースを開く際にユーザ名を示します（以前の *td_user_name* または *user*）。

このオプションは **getvar** 関数で使えます。

また，このオプションは，*qc_user_name* コマンドライン・オプションを使って設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

このオプションは［Quality Center への接続］ダイアログ・ボックスの［**ユーザ名**］を使って設定することもできます。このダイアログ・ボックスは［**ツール**］＞［**Quality Center への接続**］を選択して開きます。詳細については，第 48 章「テスト工程の管理」を参照してください。

tempdir

このオプションは一時ファイルを格納するフォルダを指定します。新しいフォルダを指定した場合，変更内容を有効にするために WinRunner を再起動してください。

このオプションは **setvar** 関数と **getvar** 関数で使えます。

このオプションは，［一般オプション］ダイアログ・ボックスの［フォルダ］カテゴリにある［**一時ファイル**］ボックスを使って設定することもできます。これについては，第 22 章「グローバル・テスト・オプションの設定」を参照してください。

testname

このオプションは現在のテストの完全パスを示します。

このオプションは **getvar** 関数で使えます。

現在のテストの完全パスは，［テストのプロパティ］ダイアログ・ボックスの［**一般設定**］タブで位置とテスト名を表示することもできます。これについては，500 ページ「テストの一般情報の文書化」を参照してください。

timeout_msec

このオプションはチェックポイントとコンテキスト・センシティブ・ステートメントを実行する際，WinRunner が使用するグローバルなタイムアウト（単位：ミリ秒）を設定します。この値は，GUI チェックポイントや同期化ポイントに埋め込まれた time パラメータに加算されます。これが WinRunner が指定されたウィンドウを検索する際の最長時間となります。timeout の数値は，ウィンドウ同期化のための遅延（*delay* テスト・オプションで設定します）の数値よりも大きいものでなければいけません（このオプションは，秒単位で測定されていた従来の「**timeout**」に相当します）。

例えば，次のステートメントを見てください。

```
win_check_bitmap ("calculator", Img1, 2, 261,269,93,42);
```

timeout_msec オプションが 10,000 ミリ秒に設定されている場合，この処理は最大で 12,000（2,000+10,000）ミリ秒になります。

このオプションは **setvar** 関数と **getvar** 関数で使えます。

指定可能な値：0 以上の数値

注：このオプションは 20 ～ 30 ミリ秒以内の精度です。

このオプションは，［一般オプション］ダイアログ・ボックスの［実行］＞［設定］カテゴリにある［**チェック ポイントと CS ステートメントのタイムアウト**］ボックスを使って設定することもできます。これについては，549 ページ「実行の設定オプションの設定」を参照してください。

また，このオプションは，**-timeout_msec** コマンドライン・オプションを使って設定することもできます。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

Treeview_path_separator

このオプションは，ツリー・ビュー・パスの項目を区切るためにテスト・スクリプトに記録される文字列を定義します。

このオプションは **setvar** 関数と **getvar** 関数で使えます。

指定可能な値：任意のテキスト文字列

このオプションは，［一般オプション］ダイアログ・ボックスの［**記録開始**］＞［**スクリプト形式**］タブにある［**ツリー ビューのパスを解析するための文字列**］ボックスを使って参照することもできます。これについては，537 ページ「スクリプト形式オプションの設定」を参照してください。

第 44 章

関数ジェネレータのカスタマイズ

ユーザ定義関数のうち，テスト・スクリプトの中で頻繁に使うものを関数ジェネレータに組み込んでおくことができます。これによりテストのプログラミング作業がより簡単になり，エラーの可能性も少なくなります。

本章では，以下の項目について説明します。

- 関数ジェネレータのカスタマイズについて
- 関数ジェネレータへのカテゴリの追加
- 関数ジェネレータへの関数の追加
- 関数とカテゴリの関連付け
- カテゴリへのサブカテゴリの追加
- カテゴリの標準の関数を設定する方法

関数ジェネレータのカスタマイズについて

関数ジェネレータに変更を加えて，頻繁に使用するユーザ定義関数を組み込むことができます。こうすることにより，よく使う関数を素早く生成して，これらをテスト・スクリプトにすぐに挿入できます。さらに関数ジェネレータでユーザ定義のカテゴリを作り，ユーザ定義関数を分類，整理することもできます。例えば，という名前のカテゴリを作り，ユーザ定義クラス専用の関数をすべて入れておくことができます。新しいカテゴリに対して標準の関数を設定したり，既定のカテゴリの標準の関数を変更したりできます。

新しいカテゴリとその関数を関数ジェネレータに追加するには，次の手順を実行します。

1 関数ジェネレータに新しいカテゴリを追加します。

2 関数ジェネレータに新しい関数を追加します。

3 新しい関数と新しいカテゴリを関連付けます。

4 新しいカテゴリに対して標準の関数を設定します。

5 新しいカテゴリにサブカテゴリを追加します（省略可能）。

関数ジェネレータをカスタマイズするときに必要な関数は，関数ジェネレータの「Function Table」カテゴリの中にすべて揃っています。起動テストにこれらの関数を挿入することにより，正しい構成設定で確実に WinRunnerWinRunner を起動できます。

関数ジェネレータへのカテゴリの追加

関数ジェネレータに新しいカテゴリを追加するには，TSL 関数の **generator_add_category** を使います。この関数の構文は次のとおりです。

generator_add_category (category_name **);**

category_name は，関数ジェネレータに追加するカテゴリの名前です。

次の例では，**generator_add_category** 関数は，「my_button」というカテゴリを関数ジェネレータに追加します。

generator_add_category ("my_button"**);**

注 :［**挿入**］＞［**関数**］＞［**オブジェクト / ウィンドウ**］コマンドでオブジェクトを選択するときに，カテゴリの標準の関数を表示させたい場合は，カテゴリ名を GUI オブジェクト・クラスの名前と同じにしなければなりません。

関数ジェネレータにカテゴリを追加するには，次の手順を実行します。

1［**関数ジェネレータ**］を開きます（［**挿入**］＞［**関数**］＞［**関数ジェネレータから**］を選ぶか，ユーザ定義ツールバーの［**関数ジェネレータから関数を挿入**］ボタンをクリックするか，INSERT FUNCTION FROM FUNCTION GENERATOR ソフトキーを押します）。

2［**カテゴリ**］ボックスで「**Function Table**」をクリックします。

3 [**関数名**] ボックスで **generator_add_category** を選択します。

4 [**引数**] ボタンをクリックします。[関数ジェネレータ] が拡張します。

5 [**カテゴリ**] ボックスに，新しいカテゴリの名前を二重引用符で囲んで入力します。[**貼り付け**] をクリックして，TSL ステートメントをテスト・スクリプトに貼り付けます。

6 [**閉じる**] をクリックして [関数ジェネレータ] を閉じます。

generator_add_category ステートメントがテスト・スクリプトに挿入されます。

注：関数ジェネレータに新しいカテゴリを挿入するためには，テスト・スクリプトを実行しなければなりません。

関数ジェネレータへの関数の追加

関数ジェネレータに関数を追加するときには，以下を指定します。

- 関数の引数の値をユーザが指定する方法
- 関数ジェネレータに表示する関数の説明

関数ジェネレータに関数を追加した後は，関数とカテゴリを関連付けなければなりません。詳細については 903 ページ「関数とカテゴリの関連付け」を参照してください。

関数ジェネレータにユーザ定義関数を追加するには，TSL 関数の **generator_add_function** を使います。

関数ジェネレータに関数を追加するには，次の手順を実行します。

1 [**関数ジェネレータ**] を開きます（[**挿入**] > [**関数**] > [**関数ジェネレータから**] を選ぶか，ユーザ定義ツールバーの [**関数ジェネレータから関数を挿入**] ボタンを選択するか，INSERT FUNCTION FROM FUNCTION GENERATOR ソフトキーを押します）。

2 [**カテゴリ**] ボックスで「**Function Table**」を選択します。

3 ［**関数名**］ボックスで **generator_add_function** を選択します。

4 ［**引数**］ボタンをクリックします。［関数ジェネレータ］が拡張します。

5 ［関数ジェネレータ］で，引数 *function_name*，*description*，*arg_number* を定義します。

- ［*Function Name*］ボックスには，新しい関数の名前を二重引用符で囲んで入力します。関数名にスペースや大文字を入れても構いません。
- ［*Description*］ボックスには，関数の説明を二重引用符で囲んで入力します。ここには，任意の文字を 180 字まで入力できます。
- ［*Arg Number*］ボックスでは必ず 1 を選びます。引数をさらに追加するには（新しい関数 1 つにつき最大 8 個まで可能），自動生成される **generator_add_function** ステートメントをテスト・スクリプトに一度追加した後で，手作業で修正しなければなりません。

6 関数の第 1 引数について，*arg_name*，*arg_type*，*default_va*lue（該当する場合）を定義します。

- ［**Arg Name**］ボックスには，引数の名前を二重引用符で囲んで入力します。引数名にスペースや大文字を入れても構いません。
- ［**Arg Type**］ボックスでは，***browse***, , ***point_window***, , または ***type_edit*** のいずれかを選択して，ユーザが［関数ジェネレータ］で引数の値をどのように指定するかを選びます。詳細については 897 ページ「関数の引数の定義」を参照してください。
- ［**Default Value**］ボックスには，（もしあれば）引数の標準の値を指定します。
- ［関数ジェネレータ］からは，新しい関数に対してさらに引数を追加することはできません。テスト・スクリプトの中で，手作業で *arg_name*，*arg_type*，*default_value* 引数を **generator_add_function** ステートメントに追加しなければなりません。

7 ［**貼り付け**］をクリックして TSL ステートメントをテスト・スクリプトに貼り付けます。

8 ［**閉じる**］をクリックして［関数ジェネレータ］を閉じます。

注 : ［関数ジェネレータ］に新しい関数を挿入するためには，テスト・スクリプトを実行しなければなりません。

関数の引数の定義

generator_add_function 関数の構文は次のとおりです。

generator_add_function (function_name, description, arg_number,
arg_name_1, arg_type_1, default_value_1,
...
arg_name_n, arg_type_n, default_value_n **);**

- *function_name* は，追加する関数の名前です。
- *description* は，関数についての簡単な説明です。これは関数が選択されると［関数ジェネレータ］の［説明］ボックスに表示されます。ここには任意の文字を 180 字まで入力できます。
- *arg_number* は，関数の引数の数です。0 から 8 までの数を指定できます。

  関数内で定義する引数ごとに，引数の名前，指定方法，（もしあれば）標準の値を指定します。新しい関数を定義するときは，関数の各引数に対して *arg_name*，*arg_type*，*default_value* を繰り返し設定します。
- *arg_name* は［関数ジェネレータ］に表示される引数の名前です。
- *arg_type* defines は，ユーザが引数の値をどのような方法で［関数ジェネレータ］に記入するかを定義します。引数の種類は 5 つあります。

  browse: ファイル参照ダイアログ・ボックスで指定したファイルが引数の値として評価されます。引数にファイルを指定した場合に browse を使います。特定のファイル拡張子の付くファイルだけを選択するには，一連の拡張子を指定します。このとき，拡張子と拡張子の間はスペースかタブで区切らなければなりません。これを指定して新しい関数を定義し終わると，［参照］ボタンを使って［関数ジェネレータ］で *browse* 引数の値を指定するときに［参照］ボタンを使うことになります。

: GUI オブジェクト（ウィンドウ以外）をポイントすることによって，引数の値が評価されます。引数がオブジェクトの論理名の場合は *point_object* を使います。新しい関数を定義し終わると，指差しボタンを使用することによって［関数ジェネレータ］で *point_object* 引数の値が定義されます。

: ウィンドウをポイントすることによって，引数の値が評価されます。引数がウィンドウの論理名の場合は point_window を使います。新しい関数を定義し終わると，指差しボタンを使用することによって［関数ジェネレータ］で *point_window* 引数の値が定義されます。

: リストから選択した値が引数になります。引数の値の数が限られている場合は *select_list* を使うと，すべての値を指定できます。これを使って新しい関数を定義し終わると，［関数ジェネレータ］で *select_list* 引数を指定するときにコンボ・ボックスを使うことになります。

type_edit: キー入力が引数の値になります。引数の値の範囲を完全に提示できないときにこの *type_edit* を使います。これを使って新しい関数を定義し終わると，［関数ジェネレータ］の中で *type_edit* 引数を指定するときに編集フィールドを使うことになります。

➤ *default_value* は，引数の既定値です。**select_list** と **type_edit** 引数に既定値を割り当てることができます。***select_list*** と ***type_edit*** 引数に対する既定値は，リストに含まれている値のうちの 1 つでなければなりません。**point_window** と **point_object** の引数には，既定値は指定できません。

以下に，**generator_add_function** ステートメントに記述できる引数定義の例を示します。引数定義の構文と，それが［関数ジェネレータ］にどのように表示されるかを例示するとともに，各定義を簡単に説明していきます。

例 1

```
generator_add_function ("window_name"," この関数は ...",1,
    "Window Name","point_window","");
```

「**関数名**（*function_name*）」は window_name，「**記述**（description）」は「この関数は ...」，「**引数の数**（*arg_number*）」は 1 個，「**引数名**（*arg_name*）」は Window Name，「**引数の種類**（*arg_type*）」は point_window です。「*標準の値*（*default_value*）」はありません。引数はウィンドウを指示することによって指定されるため，この引数は空文字列になっています。

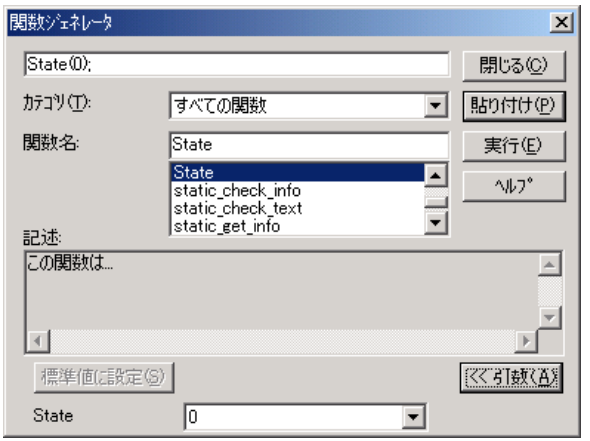
［関数ジェネレータ］で関数 を選択して［引数］ボタンをクリックすると，［関数ジェネレータ］は次のように表示されます。

例 2

```
generator_add_function("state"," この関数は ...",1,"State","select_list (0 1)",0);
```

「**関数名**（*function_name*）」は state，「**記述**（*description*）」は「この関数は ...」，「**引数の数**（*arg_number*）」は 1 個，「**引数名**（*arg_name*）」は State，「**引数の種類**（*arg_type*）」は select_list，「**標準の値**（*default_value*）」は 0 です。

［関数ジェネレータ］で関数 を選択して［引数］ボタンをクリックすると，［関数ジェネレータ］は次のように表示されます。

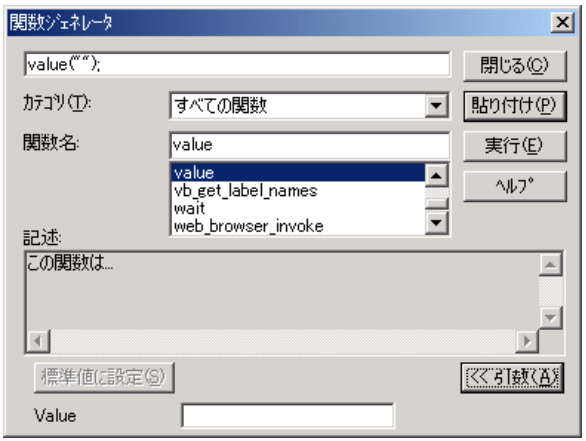


例 3

```
generator_add_function("value"," この関数は ...",1,"Value","type_edit","");
```

「**関数名**（*function_name*）」は value，「**記述**（*description*）」は「この関数は ...」，「**引数の数**（*arg_number*）」は 1 個，「**引数名**（*arg_name*）」は Value，「**引数の種類**（*arg_type*）」は type_edit です。「**標準の値**（*default_value*）」はありません。

［関数ジェネレータ］で関数 を選択して［引数］ボタンをクリックすると，［関数ジェネレータ］は次のように表示されます。



プロパティ引数の定義

プッシュボタンのラベルやチェック・ボックスの幅など，コンテキスト・センシティブ・プロパティを使う引数を受け取る関数を定義できます。この場合，引数に標準の値を割り当てることはできませんが，**attr_val** 関数を使って，選択したウィンドウまたは GUI オブジェクトのプロパティの値を指定できます。**generator_add_function** 関数への呼び出しに **attr_val** 関数を組み込みます。

attr_val 関数の構文は次のとおりです。

attr_val (object_name, "property" **);**

➤ *object_name* は，プロパティ取得対象ウィンドウまたは GUI オブジェクトです。これは **generator_add_function** 関数の引数の中で先に定義されている *arg_name* と同一でなければなりません。

➤ *property* は，コンテキスト・センシティブ・テストのときに使う，高さ，幅，ラベル，値などの任意のプロパティです。MSW_class，MSW_id のようなプラットフォーム固有のプロパティも指定できます。

特定のプロパティを定義するか，**generator_add_function** 関数への同じ呼び出しの中で先に指定されている引数で定義されているパラメータを指定することができます。これについては下記の例 2 を参照してください。

例 1

この例では，「check_my_button_label」という名前の関数を［関数ジェネレータ］に追加します。この関数は，ボタンのラベルをチェックします。

```
generator_add_function("check_my_button_label", "This function checks the
label of a button.", 2,
    "button_name", "point_object"," ",
    "label", "type_edit", "attr_val(button_name, ¥"label¥")");
```

「check_my_button_label」関数には引数が 2 つあります。第 1 引数はボタンの名前です。引数の選択方法は *point_object* です。したがって標準の値はありません。第 2 引数は，指定されたボタンの label プロパティで，*type_edit* 引数です。**attr_val** 関数は，選択された GUI オブジェクトの label プロパティをプロパティ（property）の標準の値として返します。

例 2

次の例は，「check_my_property」という名前の関数を［関数ジェネレータ］に追加します。この関数は，オブジェクトの *class*，*label*，*active* プロパティを検査します。どのプロパティの値が標準の値として戻されるかは，リストからどのプロパティを選択するかによって異なります。

```
generator_add_function ("check_my_property"," この関数はオブジェクトのプロパティを検査します。",3,
    "object_name", "point_object", " ",
    "property", "select_list(¥"class¥"¥"label¥"¥"active¥")", "¥"class¥"",
    "value:", "type_edit", "attr_val(object_name, property)");
```

generator_add_function の最初の 3 つの引数は，以下を定義しています。

➤ 新しい関数の名前（check_my_property）。

➤ ［関数ジェネレータ］の［説明］フィールドに表示される説明（「この関数はオブジェクトのプロパティを検査します。」）。

➤ 引数の数（3）

「check_my_property」の第 1 引数は，どのオブジェクトのプロパティを検査するかを特定します。この引数の第 1 パラメータは，オブジェクト名であり，種類は *point_object* です。したがって引数の第 3 パラメータの NULL 値からも分かるように，標準の値はありません。

第 2 引数は，検査対象となるプロパティであり，種類は *select_list* です。リスト内の項目は括弧で囲まれています。1 つ 1 つ引用符で囲まれ，項目同士はフィールド区切り文字で区切られています。標準の値は class プロパティです。

第 3 引数の value は，*type_edit* 引数です。ここで **attr_val** 関数を呼び出します。この関数は，その第 1 引数で定義されたオブジェクトと，第 2 引数で定義されたプロパティ（class，label，active のいずれか）を取得します。

関数とカテゴリの関連付け

関数ジェネレータに追加する関数はすべて，既存のカテゴリに関連付けなければなりません。TSL 関数の **generator_add_function_to_category** を使って関連付けをします。関連付ける関数とカテゴリはどちらも既存のものでなければなりません。

この関数の構文は次のとおりです。

generator_add_function_to_category (*category_name*, *function_name* **);**

➤ *category_name* は関数ジェネレータのカテゴリの名前です。既定のカテゴリでも，**generator_add_category** 関数を使って定義したユーザ定義のカテゴリでもどちらでも構いません。

➤ *function_name* はユーザ定義関数の名前です。その関数は，**generator_add_category** 関数で関数ジェネレータにすでに追加されていなければなりません。

関数とカテゴリを関連付けるには，次の手順を実行します。

1 ［**関数ジェネレータ**］を開きます（［**挿入**］＞［**関数**］＞［**関数ジェネレータから**］を選ぶか，ユーザ定義ツールバーで［**関数ジェネレータから関数を挿入**］ボタンをクリックするか，INSERT FUNCTION FROM FUNCTION GENERATOR ソフトキーを押します）。

2 ［**カテゴリ**］ボックスで「**Function Table**」をクリックします。

3 ［**関数名**］ボックスで **generator_add_function_to_category** を選択します。

4 ［**引数**］ボタンをクリックします。［関数ジェネレータ］が拡張します。

5 ［**カテゴリ**］ボックスに，［**関数ジェネレータ**］に表示されている既存のカテゴリ名を入力します。

6 ［**関数名**］ボックスに，［関数ジェネレータ］に表示されている既存の関数名を入力します。

7 ［**貼り付け**］をクリックして，TSL ステートメントをテスト・スクリプトに貼り付けます。

8 ［**閉じる**］をクリックして［関数ジェネレータ］を閉じます。

generator_add_function_to_category 関数がテスト・スクリプトに挿入されます。次の例では，「check_my_button_label」関数を「my_button」というカテゴリと関連付けます。この例は，「my_button」カテゴリと「check_my_button_label」関数を関数ジェネレータに追加してあることを前提としています。

```
generator_add_function_to_category ("my_button", "check_my_button_label");
```

注：関数とカテゴリを関連付けるためには，テスト・スクリプトを実行しなければなりません。

カテゴリへのサブカテゴリの追加

TSL 関数の **generator_add_subcategory** を使って，あるカテゴリを別のカテゴリのサブカテゴリにできます。どちらのカテゴリも既存のものでなければなりません。**generator_add_subcategory** 関数は，サブカテゴリに属す関数すべてを，親カテゴリの関数リストに追加します。

新しい関数のために独立のカテゴリを作成するときに，関数 **generator_add_subcategory** を使って，関連のあるコンテキスト・センシティブ・カテゴリのサブカテゴリとして，新しいカテゴリを追加できます。

generator_add_subcategory の構文は次のとおりです。

generator_add_subcategory (*category_name*, *subcategory_name* **);**

- *category_name* は，関数ジェネレータにある既存のカテゴリの名前です。
- *subcategory_name* は，関数ジェネレータにある既存のカテゴリの名前です。

あるカテゴリへサブカテゴリを追加するには，次の手順を実行します。

1. ［**関数ジェネレータ**］を開きます（［**挿入**］＞［**関数**］＞［**関数ジェネレータから**］を選ぶか，ユーザ定義ツールバーの［**関数ジェネレータから関数を挿入**］ボタンをクリックするか，INSERT FUNCTION FROM FUNCTION GENERATOR ソフトキーを押します）。
2. ［**カテゴリ**］ボックスで「**Function Table**」を選択します。
3. ［**関数名**］ボックスで **generator_add_subcategory** を選択します。
4. ［**引数**］ボタンをクリックします。［関数ジェネレータ］が拡張します。
5. ［**Category Name**］ボックスに，［関数ジェネレータ］にある既存のカテゴリ名を入力します。
6. ［**Sub-category**］ボックスに，［関数ジェネレータ］にある既存のカテゴリをサブカテゴリ名として入力します。
7. ［**貼り付け**］をクリックして TSL ステートメントをテスト・スクリプトに貼り付けます。
8. ［**閉じる**］をクリックして［関数ジェネレータ］を閉じます。

generator_add_subcategory ステートメントがテスト・スクリプトに挿入されます。次の例では，generator_add_subcategory 関数は，「my_button」カテゴリを「push_button」カテゴリのサブカテゴリとして追加します。つまり「my_button」に属すすべての関数が，push_button カテゴリ用に定義された関数のリストに追加されます。

```
generator_add_subcategory ("push_button", "my_button");
```

注：あるカテゴリにサブカテゴリを追加するためには，テスト・スクリプトを実行しなければなりません。

カテゴリの標準の関数を設定する方法

あるカテゴリに対して標準の関数を設定するには，TSL 関数の **generator_set_default_function** を使います。この関数の構文は次のとおりです。

generator_set_default_function (*category_name*, *function_name* **);**

- *category_name* は既存のカテゴリです。
- *function_name* は既存の関数です。

既定のカテゴリとユーザ定義カテゴリ（**generator_add_category** 関数で定義）のどちらに対しても，標準の関数を設定できます。ユーザ定義カテゴリに対して標準の関数を指定しなければ，WinRunner はリスト内の最初の関数を標準の関数とします。

generator_set_default_function 関数は，［関数ジェネレータ］ダイアログ・ボックスの［標準値に設定］ボタンと同じ働きをします。ただし［標準として設定］チェック・ボックスを通じて設定した標準の関数は，現在実行中の WinRunner セッションの間しか通用しません。起動テストに **generator_set_default_function** ステートメントを書き加えることにより，標準の関数は恒久的に設定されます。

カテゴリの標準の関数を設定するには、次の手順を実行します。

1 ［**関数ジェネレータ**］を開きます（［**挿入**］＞［**関数**］＞［**関数ジェネレータから**］を選ぶか，ユーザ定義ツールバーの［**関数ジェネレータから関数を挿入**］ボタンをクリックするか，INSERT FUNCTION FROM FUNCTION GENERATOR ソフトキーを押します）。

2 ［**カテゴリ**］ボックスで「**Function Table**」を選択します。

3 ［**関数名**］ボックスで **generator_set_default_function** を選択します。

4 ［**引数**］ボタンをクリックします。［関数ジェネレータ］が拡張します。

5 ［**Category Name**］ボックスに，［関数ジェネレータ］にある既存のカテゴリ名を入力します。

6 ［**Default Function**］ボックスに，［関数ジェネレータ］にある既存の関数名を入力します。

7 ［**貼り付け**］をクリックして TSL ステートメントをテスト・スクリプトに貼り付けます。

8 ［**閉じる**］をクリックして［関数ジェネレータ］を閉じます。

generator_set_default_function ステートメントがテスト・スクリプトに挿入されます。次の例では，**generator_set_default_function** を使って，push_button カテゴリの標準の関数を，**button_check_enabled** からユーザ定義関数「check_my_button_label」に変更します。

```
generator_set_default_function ("push_button", "check_my_button_label");
```

注：あるカテゴリの標準の関数を設定するためには，テスト・スクリプトを実行しなければなりません。

第 45 章

特殊な構成の初期化

「**起動テスト**」を作成することで，WinRunner を起動するたびに特別なテスト構成を自動的に初期化できます。

本章では，次の項目について説明します。

- 特殊な構成の初期化について
- 起動テストの作成
- 起動テストの例

特殊な構成の初期化について

起動テストは，WinRunner を起動するたびに自動的に実行されるテスト・スクリプトです。GUI マップ・ファイルとコンパイル済みモジュールの読み込み，記録の設定，テスト対象アプリケーションの起動などを行う起動テスト・スクリプトを作成できます。

テストを起動テストにするには，［一般オプション］ダイアログ・ボックスの［**一般設定**］＞［**起動**］カテゴリにある［起動テスト］ボックス内にテストの場所を入力します。［一般オプション］ダイアログ・ボックスの使用方法の詳細については，を参照してください。

起動テストの作成

起動テストには，以下の種類のステートメントを追加することをお勧めします。

- **load** ステートメント。テスト・スクリプトから頻繁に呼び出すユーザ定義関数を含む，コンパイル済みモジュールを読み込みます。

- **GUI_load** ステートメント。1 つまたは複数の GUI マップ・ファイルを読み込みます。WinRunner にアプリケーションの GUI オブジェクトを確実に認識させることができます。
- **set_record_attr** や **set_class_map** など, WinRunner がアプリケーションの GUI オブジェクトを記録する方法を設定するステートメント。
- **invoke_application** ステートメント。テスト対象アプリケーションを起動します。
- **add_cust_record_class** など，カスタム・オブジェクトを対象に操作を行う場合に，WinRunner がカスタムの記録用 TSL 関数を生成できるようにするステートメント。

起動テストに上記の要素を含めることで，WinRunner は指定された関数をすべて自動的にコンパイルし，必要な GUI マップ・ファイルをすべて読み込み，GUI オブジェクトの記録を設定して，テスト対象アプリケーションを起動します。

注 : テスト・スクリプト・ウィザードを使えば，GUI マップ・ファイルとテスト対象アプリケーションを読み込む，*myinit* という名前の基本的な起動テストを作成できます。[**GUI テストごとの GUI マップ ファイル**] モードで作業している場合は（第 6 章「テスト特有の GUI マップ・ファイル・モードでの作業」参照) , **myinit** テストは GUI マップ・ファイルをロードしません。

起動テストの例

以下は，起動テストに含まれるステートメントの一例です。

```
# フライト予約アプリケーションが画面に表示されていなければこれを起動する。
if ((rc=win_exists("Flight")) == E_NOT_FOUND)
    invoke_application("w:¥¥flight_app¥¥flight.exe", "", "w:¥¥flight_app",
SW_SHOW);

# コンパイル済みモジュール「qa_funcs」を読み込む。
load("qa_funcs", 1, 1);

# GUI マップ・ファイル「flight.gui」を読み込む。
GUI_load ("w:¥¥qa¥¥gui¥¥flight.gui");
```

```
# カスタムの「borbtn」クラスを標準の「push_button」クラスにマップする。
set_class_map (“borbtn”, “push_button”);
```

第 12 部

その他の Mercury 製品を使った作業

第 46 章

Business Process Testing での作業

Business Process Testing は，ビジネス・プロセス・テスト内の**コンポーネント**の作成と実装に基づいて，新しい品質保証テストの方法を使用する Mercury Quality Center のモジュールです。この方法を使用することで，技術者でない SME（各分野のエキスパート）は，スクリプト不要の環境において開発サイクルの初期の段階でビジネス・プロセス・テストを設計できます。

WinRunner が Business Process Testing と統合されたことで，既存の WinRunner スクリプトへの投資を活用し，Business Process Testing フレームワークを使用してテストの自動化プロセスを改善することができます。

本章では，Business Process Testing で使用されるスクリプトによるコンポーネントを WinRunner で作成および管理する方法について説明します。コンポーネントとテストの両方で共通または類似の WinRunner のオプションと機能については，本ユーザーズ・ガイドの関連する章で説明します。

本章では，以下の項目について説明します。

- Business Process Testing について
- ビジネス・プロセスのテスト方法について
- WinRunner でのスクリプトによるコンポーネントを使った作業の開始
- Quality Center プロジェクトへの接続
- スクリプトによるコンポーネントを使った作業
- スクリプトによるコンポーネントの新規作成
- スクリプトによるコンポーネントのプロパティの定義
- スクリプトによるコンポーネントの保存
- スクリプトによるコンポーネントの変更

Business Process Testing について

ビジネス・プロセス・テストは，コンポーネントの一連のフローで構成されるシナリオです。コンポーネントは，テスト対象のアプリケーションで特定のタスクを実行する，保守が簡単で再利用可能なスクリプトです。

通常，コンポーネントは SME によって Quality Center で作成され，変更されますが，Business Process Testing を使って WinRunner を Quality Center プロジェクトに接続すると，WinRunner でコンポーネントの作成，表示，変更，およびデバッグを行うことができます。コンポーネントは，スクリプトによるコンポーネントの形式で Quality Center のプロジェクトに保存できます。Quality Center で作業する SME は，個々のコンポーネントを 1 つ以上のビジネス・プロセス・テストに組み込むことができます。

SME は，QuickTest などのほかのテスト・ツールのコンポーネントをビジネス・プロセス・テストの WinRunner コンポーネントと組み合わせることができます。これにより，テスト対象アプリケーションの準備が整う前でも，アプリケーションのあらゆる面をカバーできるようになります。

既存の WinRunner テスト・スクリプトを，SME がビジネス・プロセス・テストで使用できるように，スクリプトによるコンポーネントとして保存することもできます。

注：Business Process Testing を WinRunner と統合するには，Business Process Testing のライセンスを購入します。また，WinRunner から Business Process Testing を使って作業するためには，Business Process Testing のサポートが組み込まれた Quality Center プロジェクトに接続する必要があります。

WinRunner コンポーネントには，コンポーネント内のステップについて説明する特別な形式のコメントを追加できます（ステップは，ビジネス・プロセス・テストの実行中に実行される操作を表します）。SME は，これらのコンポーネントを Quality Center で参照することにより，スクリプトのフローを確認できます。

SME は，Quality Center モジュール内のスクリプトによるコンポーネントを表示および操作できますが，スクリプトによるコンポーネントのステップを変更することはできません。

本章の残りの項では，WinRunner のスクリプトによるコンポーネントを使った作業に特有のオプションと機能について説明します。スクリプトによるコンポーネントの詳細については，『**Business Process Testing ユーザーズ・ガイド**』を参照してください。

ビジネス・プロセスのテスト方法について

コンポーネントは，細分化されたビジネス・プロセスの一部です。コンポーネントは効果的なビジネス・プロセス・テスト構造を作成するための基本要素になります。

例えば，ほとんどのアプリケーションのユーザは，何かを行う前にログインする必要があります。WinRunner エキスパートは，アプリケーションへのログイン手続きをテストする 1 つのスクリプトによるコンポーネントを作成できます。このコンポーネントは複数のビジネス・プロセス・テストで再利用できるため，保守，更新，およびテスト管理がより簡単になります。

コンポーネントはステップで構成されています。例えば，ログイン・コンポーネントの最初のステップではアプリケーションを開きます。第 2 ステップはユーザ名の入力です。第 3 ステップはパスワードの入力，そして第 4 ステップは［**Enter**］ボタンのクリックです。

スクリプトによるコンポーネントは，サポートされている環境で設計されたアプリケーションを対象とするステップを，WinRunner で記録または手動で入力することにより作成します。チェックポイントと出力値を追加し，選択した項目をパラメータ化し，フロー・ステートメントやその他のテスト関数を使ってテストを拡張できます。スクリプトによるコンポーネントは，Quality Center のプロジェクトに保存します。Quality Center で Business Process Testing を使用する SME は，保存されたコンポーネントを組み合わせて 1 つ以上のビジネス・プロセス・テストを作成し，それらのテストを使用してアプリケーションが予想どおりに動作することを確認します。

コンポーネントの作成は，コンポーネント・シェルの作成とコンポーネント実装の作成の 2 つのプロセスに分けられます。

- 「**コンポーネント・シェル**」はコンポーネントの外層です。シェル内の情報は，テスト・レベルで表示または使用できます。コンポーネント・シェルの情報は，WinRunner または Quality Center で作成できます。

  コンポーネント・シェルを作成すると，SME はスクリプトがまだ実装されていなくてもコンポーネント・シェルを使用してビジネス・プロセス・テストを作成できます。

- 「**コンポーネント実装**」はコンポーネントの内層です。ここには，実際のスクリプトやコンポーネントの固有の設定が含まれています。情報はコンポーネント・レベルでのみ見ることができます。コンポーネント実装は WinRunner を使用して作成します。

コンポーネントとテストの違いについて

すでに WinRunner を使用したテストの作成に慣れている場合は，その手順がスクリプトによるコンポーネントの作成や編集の手順とよく似ていることに気付くでしょう。ただし，コンポーネント・モデルの設計や目的によっては，コンポーネントの設計，編集，および実行の方法に多少の違いが生じます。次の項目のガイドラインでは，これらの違いの概要について説明します。

コンポーネントとテストの一般的な違い

コンポーネントとテストの違いに関するガイドラインおよび詳細は次のとおりです。

- コンポーネントはほかの WinRunner コンポーネントを呼び出すことができますが，WinRunner テスト，QuickTest テスト，ビジネス・プロセス・テスト，または QuickTest コンポーネントを呼び出すことはできません。
- WinRunner テストはほかのテストを呼び出すことができますが，WinRunner コンポーネント，キーワード駆動型コンポーネント，またはビジネス・プロセス・テストを呼び出すことはできません。
- コンポーネントを使って作業をすると，外部ファイルはすべて現在接続している Quality Center プロジェクトに格納されます。
- コンポーネントを使った作業では，［**デバッグ**］実行モードと［**更新**］実行モードのみがサポートされます。［**検証**］実行モードでコンポーネントを実行することはできません。これは，コンポーネントが Quality Center でビジネス・プロセス・テストの一部として実行されているときに，アプリケーションの動作の検証が実行されるためです。

- テストやコンポーネントを作成するとき，および開くときは，特定のツールバーとメニュー・コマンドを使用します。テストを作成するとき，および開くときは，［**新規作成**］および［**開く**］ツールバー・ボタンを使用します。コンポーネントを作成するとき，および開くときは，メニュー・コマンドを使用します。
- 標準設定では，［**保存**］コマンドおよびツールバー・ボタンによって無題のドキュメントがテストとして保存されます。コンポーネントを使った作業を行う場合は，［**保存**］コマンドおよびツールバー・ボタンによって無題のドキュメントがコンポーネントとして保存されるように，この標準設定を変更できます。詳細については，948 ページ「WinRunner のスクリプトによるコンポーネントを標準の保存タイプとして設定する」を参照してください。

コンポーネントを使ったデータ・テーブルを使用する場合の違い

コンポーネントを使ったデータ・テーブルを使用する場合は，次のことを考慮する必要があります。

- データ・テーブルの複数の行にテスト・データの値を入力してパラメータ化すると，（コンポーネント・パラメータに設定されたデータに基づいてコンポーネントが反復されるだけでなく）コンポーネントの実行が反復されるたびに該当するデータ・テーブルのループがデータ・テーブル内の行数に基づいて実行されます。
- コンポーネント（または Quality Center に保存されたテスト）は，指定された Quality Center のパスに従って Quality Center のデータ・テーブルを使用できます。データ・テーブルがテストとともに保存されていない場合は，ユーザがデータ・テーブルを Quality Center プロジェクトにアップロードする必要があります。

  データ・テーブルの詳細については，第 17 章「データ駆動型テストの作成」を参照してください。

Business Process Testing の役割について

Business Process Testing モデルは，役割に基づいています。このモデルでは，技術者でない SME がコンポーネントとビジネス・プロセス・テストを定義して文書化できます。WinRunner などのテスト・ツール・アプリケーションを扱う自動化エンジニアは，コンポーネントの個々のステップを記録，プログラミング，およびデバッグします。SME はそれらを各自のビジネス・プロセス・テストに取り込むことができます。

注：役割の構造と，組織における役割ごとに実行されるタスクは，組織で採用されている方法によって，ここに記述されているものとは異なる場合があります。例えば，SME の作業とテスト・ツール・エンジニアの作業を同じ人物が行う場合もあります。

WinRunner を使用する場合の Business Process Testing モデルでは，次に示す基本的なユーザの役割が決められています。

- **各分野のエキスパート（SME）**：SME は，アプリケーション・ロジックに関する具体的な知識を持ち，システム全体を高いレベルで理解し，テスト対象アプリケーションの基礎となる個々の要素やタスクの詳細を理解しています。

  これにより，SME はテストが必要な運用シナリオやビジネス・プロセスを決定したり，複数のビジネス・プロセスに共通の重要なビジネス・アクティビティを特定したりできます。また，SME は Quality Center 内で作成された各コンポーネントのテスト・ステップの保守も担当します。

  Business Process Testing モデルの大きな利点の 1 つは，SME の作業が自動化エンジニアによるコンポーネント実装の完了に依存しないことです。したがって，Business Process Testing を使用することにより，テスト対象のアプリケーションが自動化されたコンポーネントを記録できる状態になる前の，開発サイクルの初期の段階でテスト・プロセスを始めることができます。

  SME は，ビジネス・プロセス・テストに使用される値を設定し，それらをテスト・セットで実行して，結果を検討します。

- **自動化エンジニア**：自動化エンジニアは，WinRunner などの自動テスト・ツールのエキスパートです。自動化エンジニアは，WinRunner 内で作成されたスクリプトによるコンポーネントの実装，管理，およびデバッグを担当します。

注：本章は，主に自動化エンジニアと，自動化エンジニアが WinRunner で実行できる作業を対象としています。SME が Quality Center で使用できるオプションの詳細については，『**Mercury Quality Center ユーザーズ・ガイド**』を参照してください。

Business Process Testing のワークフローについて

次の例は，一般的な Business Process Testing のワークフローです。組織の実際のワークフローは，別のプロジェクトまたは製品開発ライフ・サイクルの別の段階では異なる場合があります。

Business Process Testing の用語について

本章では，Business Process Testing に固有の次の用語を使用しています。

ステップ：ステップは，ビジネス・プロセス・テストの実行中に実行される操作を表します。

スクリプトによるコンポーネント（または**コンポーネント**）：特定のタスクを実行する 1 つ以上のステップで構成された，保守が簡単で再利用可能な単位。スクリプトによるコンポーネントは，外部ソースやほかのコンポーネントからの入力値を必要とする場合があります。また，ほかのコンポーネントに出力値を返すことができます。

ビジネス・プロセス・テスト：アプリケーションの特定のビジネス・プロセスをテストするように設計された，コンポーネントの一連のフローで構成されるシナリオ。

コンポーネント入力パラメータ：コンポーネントが受け取り，コンポーネントの特定のパラメータ化ステップの値として使用する変数値。コンポーネント・パラメータには，Quality Center プロジェクトの任意のコンポーネントからアクセスできます。

コンポーネント出力パラメータ：コンポーネントが返す値。この値は，ビジネス・プロセス・テスト結果で表示できます。また，テスト内の後続のコンポーネントの入力値としても使用できます。コンポーネント・パラメータには，Quality Center プロジェクトの任意のコンポーネントからアクセスできます。

役割：Business Process Testing に関係するユーザのさまざまなタイプ。

各分野のエキスパート（SME）：アプリケーション・ロジックに関する具体的な知識を持ち，システム全体を高いレベルで理解し，テスト対象アプリケーションの基礎となる個々の要素やタスクの詳細を理解している人。各分野のエキスパートは，Quality Center を使用してコンポーネントとビジネス・プロセス・テストを作成します。

自動化エンジニア：WinRunner などの自動テスト・ツールのエキスパート。

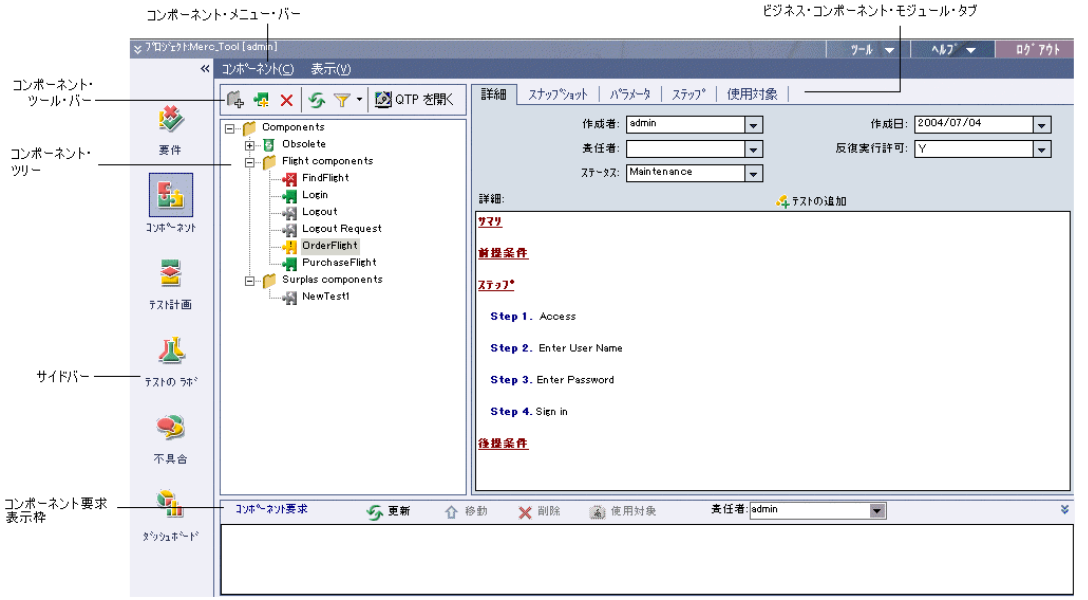
Quality Center ビジネス・コンポーネント・モジュールでのコンポーネントの作成

SME は，新規のコンポーネントを作成し，Quality Center のビジネス・コンポーネント・モジュールでコンポーネント・シェルと自動化されていないステップを定義できます。

SME は，ステップを自動化することにより，コンポーネントを WinRunner コンポーネントに変換できます。自動化されていないコンポーネントを WinRunner コンポーネントに変換すると，コンポーネント内の自動化されていないステップは先頭に #"# が付いたコメントとして WinRunner に表示されます。

注：WinRunner のコメントは最大 500 文字までなので，SME は Quality Center で自動化されてないステップを作成するときに 500 文字を超えないようにすることが重要です。

Quality Center プロジェクトのコンポーネントの例を次に示します。



コンポーネント・シェルには，次の要素が含まれています。

- **説明**：コンポーネントの目的や内容のサマリ，反復が許可されるかどうか，およびコンポーネントの前提条件と事後条件を定義するための詳細な指示です。
- **画面ショット**：コンポーネントの目的や操作について，視覚的な手掛かりや説明を提供するイメージです。
- **入力パラメータ**：コンポーネントが受け取ることのできるデータの名前，標準設定値，および説明です。
- **出力パラメータ**：コンポーネントがビジネス・プロセス・テストに返すことのできる値の名前と説明です。
- **ステータス**：コンポーネントの現在のステータスです。例えば，コンポーネントは完全に実装され実行する準備が整っている，または修正の必要なエラーがあるなどです。最も厳しいコンポーネント・ステータスは，ビジネス・プロセス・テスト全体のステータスを定義したものです。

注：上記の要素は，すべて WinRunner の［スクリプト化コンポーネントのプロパティ］ダイアログ・ボックスを使用して作成または変更できます。詳細については，932 ページ「スクリプトによるコンポーネントのプロパティの定義」を参照してください。

WinRunner でのコンポーネントの実装

スクリプトによるコンポーネントは，WinRunner で実装します。SME によってシェルが定義された既存のコンポーネントを開いて実装できます。また，WinRunner テストを Quality Center プロジェクトのスクリプトによるコンポーネントとして保存することもできます。

WinRunner では，サポートされている任意の環境でステップを記録するか，TSL（WinRunner のテスト・スクリプト言語）を使って手動でステップをプログラミングすることにより，コンポーネントを実装します。チェックポイントと出力値を追加し，選択した項目をパラメータ化し，フロー・ステートメントやユーザ定義関数を使ってテストを拡張します。

また，WinRunner から，コンポーネント固有のオプションの表示や設定を行うこともできます。例えば，コンポーネントの説明を表示したり，コンポーネントのスクリーンショットを変更したり，コンポーネントの反復が適用可能かどうかを決定したりできます。

スクリプトによるコンポーネントの実装が完了すると，SME は Quality Center のビジネス・コンポーネント・モジュールでコンポーネントを開いて，自動化エンジニアが WinRunner で入力した特別なラベルを持つコメントを参照することができます。これらのコメントは，［**ステップ**］タブにコンポーネントのステップのサマリとして表示されます。SME や自動化エンジニアは，Quality Center を使用して WinRunner を起動し，コンポーネントを実行することにより，コンポーネントを表示またはデバッグすることもできます。

Quality Center テスト計画モジュールでのビジネス・プロセス・テストの作成

SME は，ビジネス・プロセス・テストを作成するため，ビジネス・プロセス・テストに適したコンポーネントを選択（ドラッグ・アンド・ドロップ）し，それらの実行設定を設定します。

各コンポーネントは，異なるビジネス・プロセス・テストで別々に使用できます。例えば，テストごとに異なる入力パラメータ値を使用したり，異なる反復回数で実行されるようにコンポーネントを設定できます。

ビジネス・プロセス・テストの実行と結果の分析

WinRunner で［実行］オプションや［デバッグ］オプションを使用して，個々のスクリプトによるコンポーネントを実行およびデバッグできます。Quality Center のテスト計画モジュールからコンポーネントを実行することにより，ビジネス・プロセス・テスト内のスクリプトによるコンポーネントをデバッグすることもできます。Quality Center は，WinRunner を起動し，コンポーネントを実行します。

ビジネス・プロセス・テストがデバッグされ，通常のテスト実行の準備ができると，ほかのテストを Quality Center で実行するのと同じように，テストのラボ・モジュールからテストを実行できます。テストを実行する前に，テスト担当者は実行時パラメータの値と反復を定義できます。

SME は，テストのラボ・モジュールからビジネス・プロセス・テスト全体の実行結果を確認できます。この結果には，各パラメータの値や，WinRunner によって報告された個々のイベントの結果が含まれています。

スクリプトによるコンポーネント・ビュー概要

スクリプトによるコンポーネントは，Quality Center のビジネス・コンポーネント・モジュールの［ステップ］タブにステップ・コメントのリストとして表示されます。各コメントは個別の行に表示されます。

詳細については，『**Business Process Testing ユーザーズ・ガイド**』を参照してください。

WinRunner でのスクリプトによるコンポーネントを使った作業の開始

Quality Center に Business Process Testing のライセンスが含まれている場合は，WinRunner を Quality Center プロジェクトに接続できます。これにより，Quality Center のビジネス・プロセス・テストに取り込むことができるスクリプトによるコンポーネントを作成，表示，およびデバッグできます。

自動化エンジニアが WinRunner のコンポーネント・スクリプトにわかりやすいコメントを追加しておくと，SME はコンポーネントの詳細を参照し，プログラミングの知識がなくてもコンポーネントをビジネス・プロセス・テストに取り込むことができます。

Quality Center プロジェクトへの接続

スクリプトによるコンポーネントを使って作業するには，まず WinRunner を Quality Center サーバに接続する必要があります。このサーバによって，WinRunner と Quality Center プロジェクト間の接続が処理されます。次に，WinRunner からアクセスする Quality Center プロジェクトを選択します。このプロジェクトには，テスト対象アプリケーションのコンポーネントと実行セッションの情報が保存されます。Quality Center プロジェクトはパスワードで保護されるため，ユーザ名とパスワードを指定する必要があります。

注：WinRunner でスクリプトによるコンポーネントを使った作業を行う前に，WinRunner と Quality Center プロジェクトの統合を有効にする必要があります。WinRunner のメイン・ウィンドウで，［**ツール**］＞［**一般オプション**］を選択します。［**実行**］カテゴリを選択します。［**その他の Mercury 製品によるテストのリモート実行を許可する**］チェック・ボックスを選択します。

Quality Center への接続方法の詳細については，964 ページ「WinRunner の Quality Center への接続」を参照してください。

Quality Center からの切断方法の詳細については，967 ページ「Quality Center からの切断」を参照してください。

スクリプトによるコンポーネントを使った作業

スクリプトによるコンポーネントを使って作業するときは，WinRunner のツールやオプションを最大限に利用できます。例えば，スクリプトによるコンポーネントに関数を追加する手順を示す関数ジェネレータを使用できます。また，コンパイル済みモジュールからユーザ定義関数を呼び出したり，選択した項目をパラメータ化したり，スクリプトによるコンポーネントにチェックポイントと出力値を追加したりすることもできます。

本章では，スクリプトによるコンポーネントの作成方法についてのみ説明します。スクリプト内のステップの入力方法や拡張方法については，本書の関連する章を参照してください。

スクリプトによるコンポーネントを作成した後，SME は Quality Center プロジェクトのビジネス・コンポーネント・モジュールで，コンポーネント内の特別なラベルを持つ（読み取り専用の）コメントを参照できます。各分野のエキスパートはスクリプトによるコンポーネントを実行し，それをビジネス・プロセス・テストに追加できますが，変更が必要になった場合に WinRunner でスクリプトによるコンポーネントの保守を行うのは引き続き QuickTest エンジニアです。スクリプトによるコンポーネントは Quality Center では変更できません。

スクリプトによるコンポーネントは，テストと同じように保存できます。詳細については，943 ページ「スクリプトによるコンポーネントの保存」を参照してください。

スクリプトによるコンポーネントの新規作成

本項では，WinRunner でスクリプトによるコンポーネントを新規作成する方法について説明します。

スクリプトによるコンポーネントを作成するか開く前に，WinRunner をスクリプトによるコンポーネントが保存されている Quality Center プロジェクトに接続する必要があります。

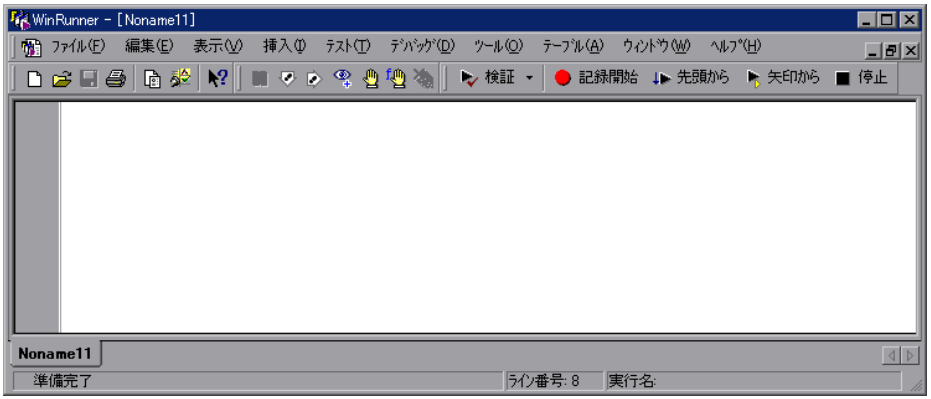
注：スクリプトによるコンポーネントを削除する場合は，WinRunner と Quality Center のどちらで作成されたスクリプトであっても，Quality Center でしか削除できません。詳細については，『**Business Process Testing ユーザーズ・ガイド**』を参照してください。

スクリプトによるコンポーネントを新規作成するには，次の手順を実行します。

1 スクリプトによるコンポーネントを保存する Quality Center プロジェクトに接続します。詳細については，928 ページ「Quality Center プロジェクトへの接続」を参照してください。

2 ［**ファイル**］＞［**新規作成**］を選択するか，CTRL キーを押しながら N キーを押します。

新しい無題のテストが開きます。



3 テストの場合と同じように，WinRunner で提供されている機能やオプションを使用してテストにステップを追加します。例えば，ステップ・ジェネレータを使って，プログラミング・ロジックを含むステップを追加できます。また，スクリプトによるコンポーネントには，チェックポイントや出力値も追加できます。

注：スクリプトによるコンポーネントで作業するときは，テストの場合に比べて，テスト・オプションの中に使用できないものや機能が異なるものがいくつかあります。詳細については，918 ページ「コンポーネントとテストの違いについて」を参照してください。

注：各分野のエキスパートは，Quality Center のビジネス・コンポーネント・モジュールでスクリプトによるコンポーネントの読み取り専用のコメントを参照し，テスト計画モジュールを使用してビジネス・プロセス・テストにスクリプトによるコンポーネントを取り込むことができます。スクリプトによるコンポーネントのステップは変更できません。

---

4 ［スクリプト化コンポーネントのプロパティ］ダイアログ・ボックスで，コンポーネントのプロパティ（説明，入出力パラメータ，ステータス，反復を許可

するかどうかなど）を定義します。詳細については，932 ページ「スクリプトによるコンポーネントのプロパティの定義」を参照してください。

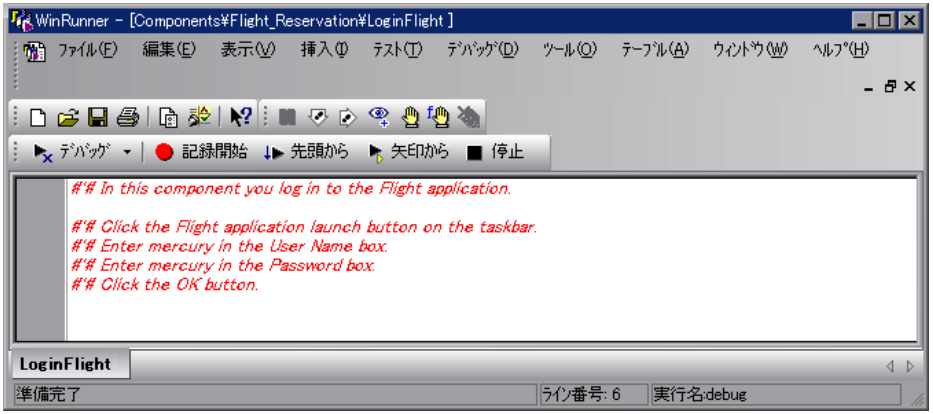
5 テストをスクリプトによるコンポーネントとして保存します。詳細については，943 ページ「スクリプトによるコンポーネントの保存」を参照してください。

SME 向けのコメントの追加

WinRunner では，SME が Quality Center のビジネス・コンポーネント・モジュールで参照できる，HTML 形式による記号付きのコメントを追加できます。

Quality Center で SME に対して表示する各コメントの行の先頭に，**#&srq;#** を追加します。

WinRunner で追加するコメントは，例えば次のようなものです。



これらのコメントは，Quality Center では次のように表示されます。

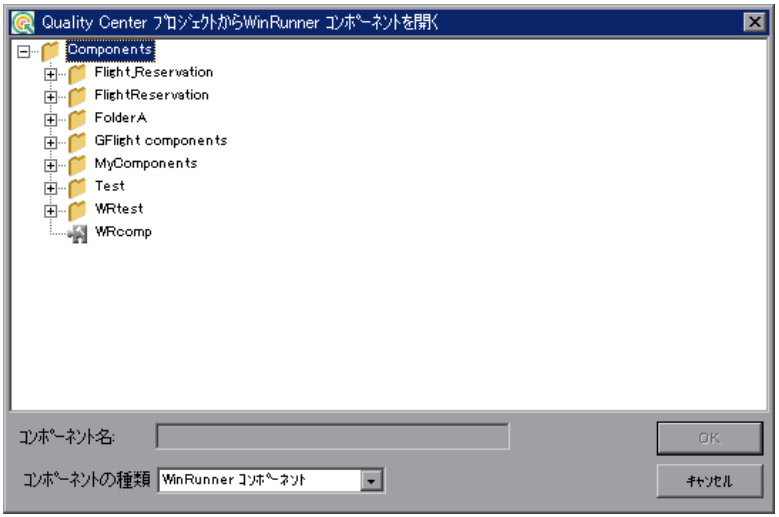
スクリプトによるコンポーネントのプロパティの定義

スクリプトによるコンポーネントのプロパティを WinRunner で定義できます。SME はこれらのプロパティを Quality Center に表示し，必要に応じて変更できます。逆に，Quality Center で SME によって最初に定義されたスクリプトによるコンポーネントのプロパティを表示または変更することもできます。

スクリプトによるコンポーネントのプロパティを定義するには，次の手順で行います。

1 プロパティを定義するスクリプトによるコンポーネントを含む Quality Center プロジェクトに接続します。詳細については，928 ページ「Quality Center プロジェクトへの接続」を参照してください。

2 ［**ファイル**］＞［**スクリプト化コンポーネントを開く**］を選択するか，CTRL キーを押しながら H キーを押します。［Quality Center プロジェクトから WinRunner コンポーネントを開く］ダイアログ・ボックスが開きます。



各コンポーネントのステータスがアイコンで表示されます。コンポーネントのステータスとステータスのアイコンの詳細については，『**Business Process Testing ユーザーズ・ガイド**』を参照してください。

ヒント：最近使用したコンポーネントを開くことができます。その場合は，［**ファイル**］メニューの**最近使用したファイル**のリストから選択します。

---

3 コンポーネント・ツリーで，該当するフォルダをクリックします。ツリーを展開してコンポーネントを表示するには，閉じているフォルダをダブルクリックします。ツリーを折りたたむには，開いているフォルダをダブルクリックします。

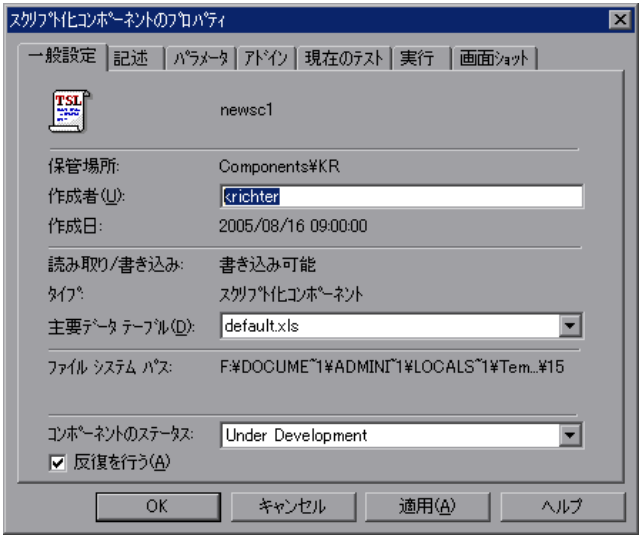
4 コンポーネントを選択します。コンポーネントの名前が読み取り専用の［**コンポーネント名**］ボックスに表示されます。

5 ［**OK**］をクリックしてコンポーネントを開きます。

コンポーネントを開くと，WinRunner のタイトル・バーに Quality Center のフルパスとコンポーネント名が表示されます。例えば，login コンポーネントのタイトル・バーは次のようになります。

Components¥Flights¥login

6 ［**ファイル**］メニューの［**スクリプト化コンポーネントのプロパティ**］を選択します。［スクリプト化コンポーネントのプロパティ］ダイアログ・ボックスが開きます。このダイアログ・ボックスは，内容ごとに 7 つのタブに分かれています。



注 : [ファイル] メニューの [**スクリプト化コンポーネントのプロパティ**] コマンドを使用できるのは，Business Process Testing を使って Quality Center に接続しているときだけです。

---

7 スクリプトによるコンポーネントのプロパティを設定するには，後の各項の説明に従って，該当するタブを選択し，オプションを設定します。

8 変更を適用し，[スクリプト化コンポーネントのプロパティ] ダイアログ・ボックスを開いたままにする場合は，[**適用**] をクリックします。

9 終わったら，[**OK**] をクリックしてダイアログ・ボックスを閉じます。

10 コンポーネントを閉じます。

注 : パラメータの変更内容が Quality Center に保存されるのは，WinRunner でコンポーネントを閉じたときだけです。[スクリプト化コンポーネントのプロパティ] におけるほかのすべての変更内容は，[**適用**] をクリックしたとき，または [スクリプト化コンポーネントのプロパティ] ダイアログ・ボックスを閉じたときに保存されます。

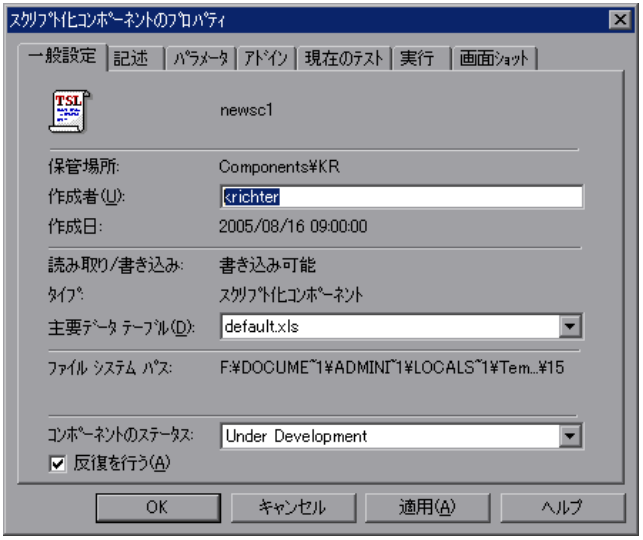
[スクリプト化コンポーネントのプロパティ] ダイアログ・ボックスには，スクリプトによるコンポーネントに関して特別なオプションと意味を持った以下のタブが表示されます。

タブの見出し	説明
一般設定	スクリプトによるコンポーネントに関する一般的な詳細を設定できます。
記述	スクリプトによるコンポーネントの説明テキストを入力できます。
パラメータ	スクリプトによるコンポーネントの入出力パラメータを定義できます。
画面ショット	スクリプトによるコンポーネントに関連する画面ショットを添付できます。

［アドイン］タブ，［現在のテスト］タブ，および［実行］タブは，テストとコンポーネントに同じように対応しています。これらのタブの詳細については，498 ページ「［テストのプロパティ］ダイアログ・ボックスからのテストのプロパティの設定」を参照してください。

一般的な詳細の定義

［スクリプト化コンポーネントのプロパティ］ダイアログ・ボックスの［一般］タブでは，スクリプトによるコンポーネントに関する一般的な詳細を記述および参照できます。作成者の名前の変更，コンポーネントのステータスの定義，およびビジネス・プロセス・テストでスクリプトによるコンポーネントを反復できるかどうかの選択を行うことができます。



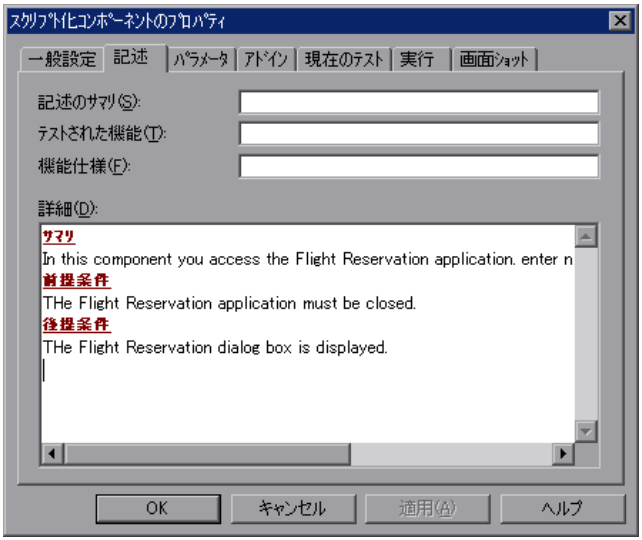
［一般設定］タブには，スクリプトによるコンポーネントに関する次の情報が表示されます。

オプション	説明
	スクリプトによるコンポーネントの名前が表示されます。
保管場所	Quality Center のコンポーネント・ツリーにおけるスクリプトによるコンポーネントの位置が表示されます。

オプション	説明
作成者	スクリプトによるコンポーネントの作成者の名前を入力または変更できます。
作成日	スクリプトによるコンポーネントが作成された日時が表示されます。
読み取り / 書き込み	スクリプトによるコンポーネントが書き込み可能かどうかを示します。
タイプ	コンポーネントの種類がスクリプトによるコンポーネントであることを示します。
主要データ テーブル	データ・テーブル・ファイルを選択できます。**default.xls** または Quality Center に保存されているデータ・テーブルの Quality Center パスのいずれかです。
ファイル システム パス	WinRunner で開いているコンポーネントが保存されるファイル・システム内の一時パスが表示されます。このパスは Quality Center のキャッシュ・パスです。
コンポーネントのステータス	コンポーネントの準備の状態を定義できます。このステータスは，［**Under Development**］，［**Ready**］，［**Maintenance**］，または［**Error**］に設定できます。このステータスは，Quality Center のコンポーネントの［詳細］タブで設定することもできます。 コンポーネントのステータスの詳細については，『**Business Process Testing ユーザーズ・ガイド**』の「**ビジネス・コンポーネント・モジュールでの作業**」の章を参照してください。
反復を行う	ビジネス・プロセス・テスト内でスクリプトによるコンポーネントに複数の反復を設定できるかどうかを定義できます。このオプションは，Quality Center のコンポーネントの［詳細］タブで設定することもできます。

コンポーネントの説明の定義

［スクリプト化コンポーネントのプロパティ］ダイアログ・ボックスの［説明］タブでは，スクリプトによるコンポーネントの説明を参照または入力できます。



［説明］タブには，スクリプトによるコンポーネントに関する次の情報が表示されます。

オプション	説明
記述のサマリ	コンポーネントの簡単なサマリを指定できます。
テストされた機能	テストされるアプリケーション機能の説明を指定できます。
機能仕様	テストされる機能に関するアプリケーション機能仕様への参照を指定できます。
詳細	スクリプトによるコンポーネントの説明テキスト（前提条件と事後条件を含む）が表示されます。必要な場合は，この領域で情報を入力または変更できます。

コンポーネント・パラメータの定義

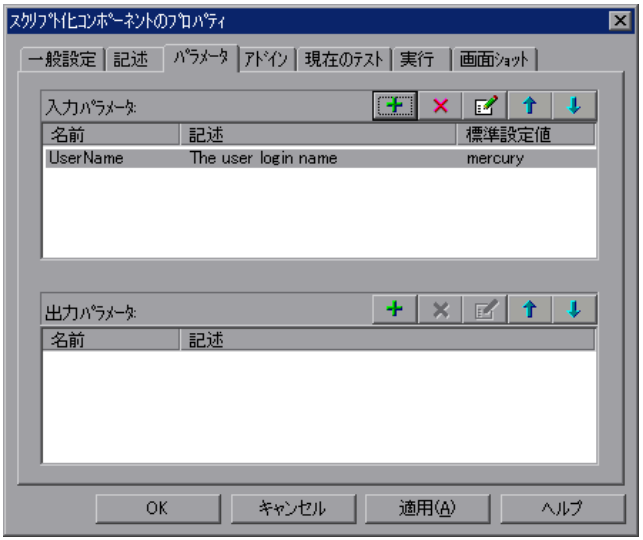
入出力コンポーネント・パラメータを使用して，ビジネス・プロセス・テスト内で 1 つのコンポーネントから以降のコンポーネントにデータを送ることができます。

「**コンポーネント・パラメータ**」は，さまざまな値を割り当てることが可能なビジネス・プロセス・テストまたはコンポーネント内の要素です。入出力コンポーネント・パラメータを使用することで，ビジネス・プロセス・テスト内のコンポーネント間で値を渡すための変数値をコンポーネントで使用できるようになります。コンポーネント・パラメータに指定する値によって，テスト結果が左右されることもあります。この処理によって，コンポーネントやビジネス・プロセス・テストの機能と柔軟性が大幅に向上します。

［**スクリプト化コンポーネントのプロパティ**］ダイアログ・ボックスの［パラメータ］タブでは，スクリプトによるコンポーネントのパラメータを定義，編集および削除できます。

［スクリプト化コンポーネントのプロパティ］ダイアログ・ボックスの［パラメータ］タブでは，コンポーネントに値を渡す入力コンポーネント・パラメータと，コンポーネントからほかのコンポーネントに値を渡す出力コンポーネント・パラメータを定義できます。

また，［パラメータ］タブを使用して既存のコンポーネント・パラメータを変更または削除することもできます。

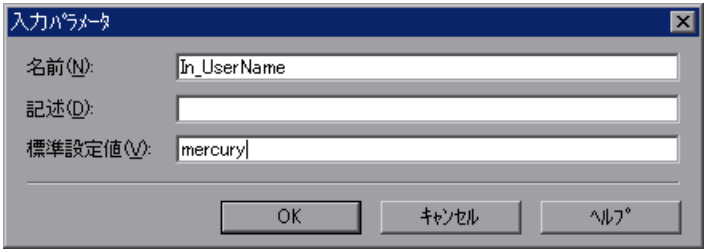


［パラメータ］タブには，スクリプトによるコンポーネントの既存のパラメータの詳細が表示されます。コンポーネント・パラメータの詳細については，『**Business Process Testing ユーザーズ・ガイド**』を参照してください。

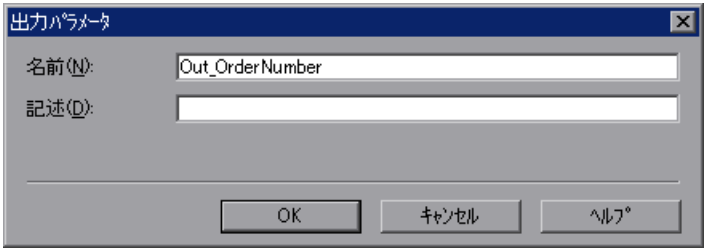
注：パラメータの変更内容が Quality Center に保存されるのは，WinRunner でコンポーネントを閉じたときだけです。［スクリプト化コンポーネントのプロパティ］におけるほかのすべての変更内容は，［**適用**］をクリックしたとき，または［スクリプト化コンポーネントのプロパティ］ダイアログ・ボックスを閉じたときに保存されます。

新しい入力パラメータまたは出力パラメータを定義するには，次の手順を実行します。

1 ［スクリプト化コンポーネントのプロパティ］ダイアログ・ボックスの［**パラメータ**］タブで，パラメータを追加するパラメータ・リスト（［**入力**］または［**出力**］）に対応する［**追加**］ボタンをクリックします。［入力パラメータ］ダイアログ・ボックスまたは［出力パラメータ］ダイアログ・ボックスが開きます。



入力パラメータの場合は，ダイアログ・ボックスに［**標準設定値**］を入力するためのテキスト・ボックスが表示されます。出力パラメータの場合は，ダイアログ・ボックスに［**標準設定値**］エディット・ボックスは表示されません。



2 パラメータの［**名前**］と［**記述**］を入力します。入力パラメータの場合は，パラメータの［**標準設定値**］を指定できます。標準設定値は，コンポーネントの実行時にビジネス・プロセス・テストによってほかの値が提供されない場合に使用されます。

ヒント：パラメータのタイプを示す IN や OUT などの文字列をパラメータ名の先頭または末尾に付けることをお勧めします。これにより，コンポーネントのステップが読みやすくなります。

---

3 [**OK**] をクリックします。該当するパラメータ・リストにパラメータが追加されます。

4 必要な場合は，[**項目を上へ移動**] 矢印ボタンと [**項目を下へ移動**] 矢印ボタンを使用してパラメータの順序を変更します。

注：パラメータの値は順番に割り当てられるため，コンポーネントの反復時にパラメータに割り当てられる値は，[パラメータ] タブにおけるパラメータの表示順序によって決まります。

また，ビジネス・プロセス・テストが使用するコンポーネント内のパラメータの順序を変更すると，テストが失敗することがあります。

---

5 [**OK**] をクリックしてダイアログ・ボックスを閉じます。パラメータの詳細が [パラメータ] タブに表示されます。

6 コンポーネントを閉じて，定義したパラメータを保存します。

パラメータをパラメータ・リストから削除するには，次の手順を実行します。

1 [スクリプト化コンポーネントのプロパティ] ダイアログ・ボックスの [**パラメータ**] タブで，削除するパラメータの名前を選択します。

2 削除するパラメータに対応する [**削除**] ボタンをクリックします。

3 [**OK**] をクリックしてダイアログ・ボックスを閉じます。

パラメータ・リスト内のパラメータを変更するには，次の手順を実行します。

1 [スクリプト化コンポーネントのプロパティ] ダイアログ・ボックスの [**パラメータ**] タブで，変更するパラメータの名前を選択します。

2 [**パラメータの変更**] ボタンをクリックするか，パラメータ名をダブルクリックします。[入力パラメータ] ダイアログ・ボックスまたは [出力パラメータ] ダイアログ・ボックスが開き，パラメータの現在の名前と説明（および，該当する場合は標準設定値）が表示されます。

3 必要に応じて，パラメータを変更します。

4 [**OK**] をクリックしてダイアログ・ボックスを閉じます。変更されたパラメータがパラメータ・リストに表示されます。

5 コンポーネントを閉じて，変更内容を保存します。

画面ショットの添付

［スクリプト化コンポーネントのプロパティ］ダイアログ・ボックスの［**画面ショット**］タブでは，スクリプトによるコンポーネントに関連する画像を添付できます。

Quality Center では，コンポーネントを使用するビジネス・プロセス・テストにこの画像が表示されます。この画像は，各分野のエキスパートに対してコンポーネントの主な目的を視覚的に示します。該当するサムネイル画像をクリックすると，ビジネス・プロセス・テスト内の各コンポーネントの画像がテスト計画モジュールの［テスト スクリプト］タブに表示されます。SME は，これらの画像の一連の流れを見ることで，ビジネス・プロセス・テストの流れを容易に把握できます。

［画面ショット］タブでは，画面ショットをキャプチャできます。また，画像を含む保存済みの **.png** ファイルをロードできます。

注：画面ショット画像は，Quality Center での作業時にキャプチャし，スクリプトによるコンポーネントとともに保存することもできます。

画面ショットをキャプチャして添付するには，次の手順を実行します。

1 ［スクリプト化コンポーネントのプロパティ］ダイアログ・ボックスの［画面ショット］タブで，［**アプリケーションで画面ショットをキャプチャする**］を選択し，［**キャプチャ**］をクリックします。カーソルの形状が十字形のポインタに変わります。

2 ポインタをドラッグしてキャプチャするアプリケーションの領域を選択し，マウスの右ボタンをクリックします。キャプチャした画像が保存され，［画面ショット］タブに表示されます。

3 ［**OK**］をクリックします。

既存の画面ショットをロードするには，次の手順を実行します。

1 ［スクリプト化コンポーネントのプロパティ］ダイアログ・ボックスの［画面ショット］タブで，［**ファイルからロードする**］を選択し，参照ボタンをクリックします。

2 ［Choose Picture File］ダイアログ・ボックスが開きます。

3 必要な **.png** ファイルを参照し，［**開く**］をクリックします。キャプチャした画像が保存され，コンポーネントの［画面ショット］タブに表示されます。

4 ［**OK**］をクリックします。

スクリプトによるコンポーネントの保存

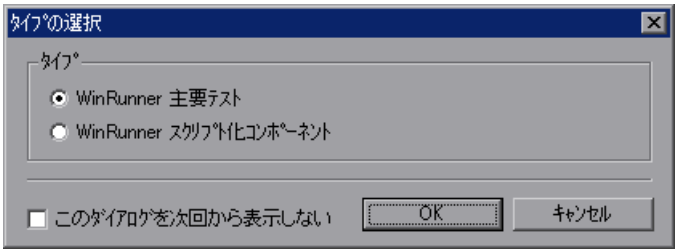
新しいコンポーネントを作成するには，既存または新規のテストをスクリプトによるコンポーネントとして保存します。コンポーネントまたはテストを変更した後は，更新されたコンポーネントを Quality Center プロジェクトに保存できます。ビジネス・コンポーネントを保存するときには，わかりやすい名前を付けて，Quality Center プロジェクト（ビジネス・コンポーネント・モジュール）のコンポーネント・ツリー内の適切なフォルダに保存してください。

また，既存のコンポーネントのコピーを，同じ Quality Center プロジェクト内の任意のフォルダに保存することもできます。すべてのユーザがさまざまなコンポーネントを区別できるように，コンポーネントのコピーの名前は，たとえ別のフォルダに保存する場合でも変更する必要があります。

コンポーネントを Quality Center プロジェクトに保存するには，次の手順を実行します。

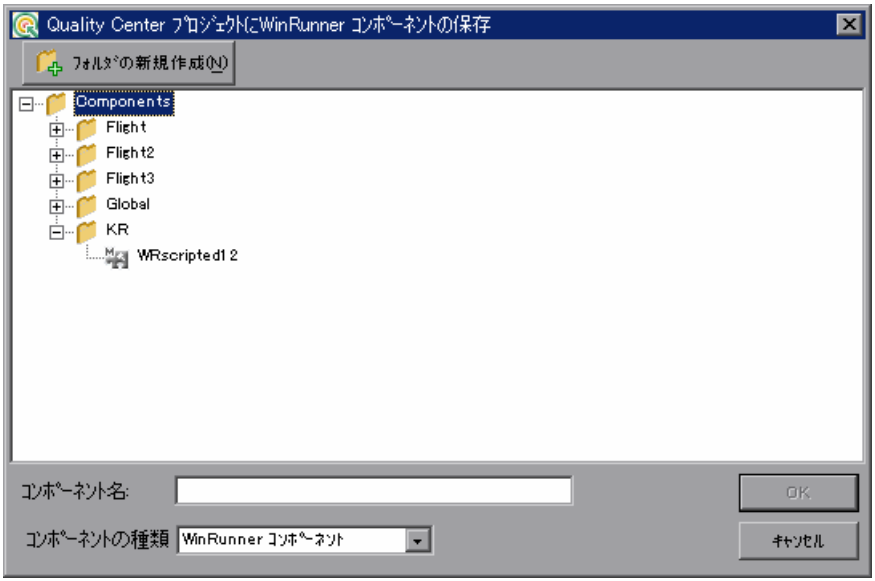
1 次のどれかの方法でコンポーネントを保存します。

- 既存の WinRunner テストをコンポーネントとして保存するには，［**ファイル**］＞［**スクリプト化コンポーネントとして保存**］を選択します。
- 既存のコンポーネントを変更して保存する場合は，［**保存**］をクリックします。
- 既存のコンポーネントに対する変更を保存する場合や，既存のコンポーネントのコピーを保存する場合は，［**ファイル**］＞［**スクリプト化コンポーネントとして保存**］を選択します。手順 2 に進みます。
- 無題のテストをスクリプトによるコンポーネントとして保存するには，［**保存**］をクリックするか，［**ファイル**］＞［**上書き保存**］を選択するか，または CTRL キーを押しながら S キーを押します。コンポーネントを一度も保存したことがない場合は，［タイプの選択］ダイアログ・ボックスが開きます。

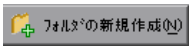


［**WinRunner スクリプト化コンポーネント**］を選択し，［**OK**］をクリックします。

2 ［**Quality Center プロジェクトに WinRunner コンポーネントの保存**］ダイアログ・ボックスが開き，コンポーネント・ツリーが表示されます。



コンポーネントを保存するフォルダを選択します。ツリーを展開して下位レベルを表示するには，閉じているフォルダをダブルクリックします。サブレベルの表示を折りたたむには，開いたフォルダをダブルクリックします。



コンポーネントは，Quality Center プロジェクトの既存のフォルダに保存することも，［**フォルダの新規作成**］ボタンをクリックして新しいフォルダを作成し，そのフォルダに保存することもできます。既存のコンポーネントのコピーを同じ名前で保存するには，別のフォルダに保存する必要があります。

注：コンポーネント・フォルダ名には，「¥」，「^」，「*」の文字は使用できません。

---

3 ［**コンポーネント名**］ボックスで，コンポーネントの名前を入力します。誰もがコンポーネントを識別しやすいよう，わかりやすい名前を付けます。

注：スクリプトによるコンポーネントの名前の先頭および末尾にスペースを使うことはできません。また，次の文字は使用できません。

! @ # $ % ^ & * ( ) - + = { } [ ] | ¥ " ' : ; ? / < > . , ~ ’

---

4 ［**コンポーネントの種類**］については，［**WinRunner コンポーネント**］をそのまま使用します。

5 ［**OK**］をクリックし，コンポーネントを保存してダイアログ・ボックスを閉じます。WinRunner がコンポーネントを保存している間，実行中の操作がステータス・バーに表示されます。

コンポーネントが Quality Center プロジェクトに保存されます。これで，ビジネス・コンポーネントを WinRunner で表示，変更できるようになります。

テストをスクリプトによるコンポーネントとして保存する

Business Process Testing を使って，既存のテストを Quality Center のスクリプトによるコンポーネントとして保存できます。これにより，各分野のエキスパート（SME）は 1 つ以上のビジネス・プロセス・テストにスクリプトによるコンポーネントを取り込むことができます。

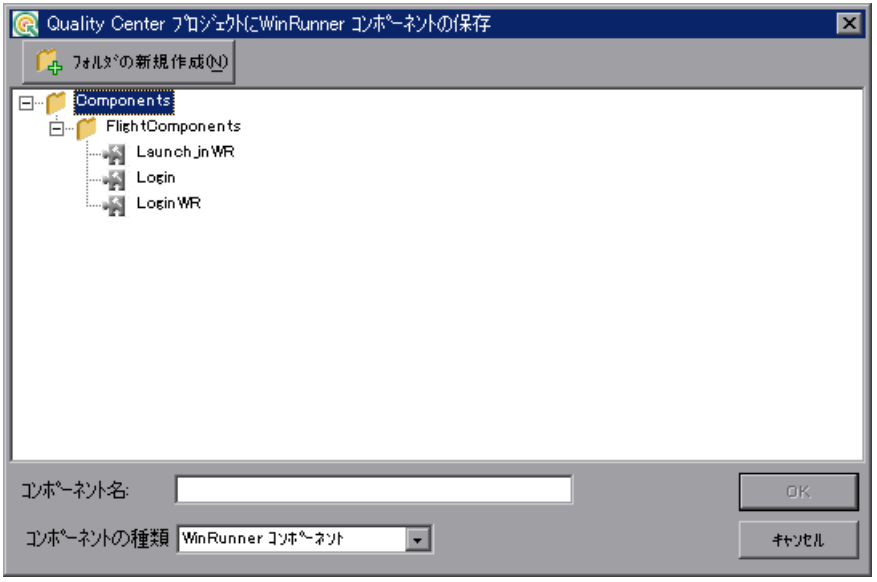
スクリプトによるコンポーネントを Quality Center データベースに保存できるのは，Quality Center プロジェクトに接続している場合だけです。

既存のテストをスクリプトによるコンポーネントとして保存するには，次の手順を実行します。

1 Quality Center プロジェクトに接続していることを確認します。詳細については，928 ページ「Quality Center プロジェクトへの接続」を参照してください。

2 新規または既存の WinRunner テストを開きます。

3 ［**ファイル**］＞［**スクリプト化コンポーネントとして保存**］を選択します。［Quality Center プロジェクトに WinRunner コンポーネントの保存］ダイアログ・ボックスが開きます。



Quality Center のビジネス・コンポーネント・モジュールのコンポーネント・ツリーが表示されます。

注 :［ファイル］メニューの［**スクリプト化コンポーネントのプロパティ**］コマンドを使用できるのは，Business Process Testing を使って Quality Center に接続しているときだけです。

4 コンポーネント・ツリー内の該当するフォルダを選択するか，［**フォルダの新規作成**］ボタンをクリックして新しいフォルダを作成します。ツリーを展開するには，閉じたフォルダのアイコンをダブルクリックします。サブレベルの表示を折りたたむには，開いたフォルダのアイコンをダブルクリックします。

5 ［**コンポーネント名**］テキスト・ボックスで，スクリプトによるコンポーネントの名前を入力します。自分と Quality Center を使用する SME がコンポーネントを識別しやすいように，わかりやすい名前を付けます。

6 ［**OK**］をクリックし，コンポーネントを保存してダイアログ・ボックスを閉じます。

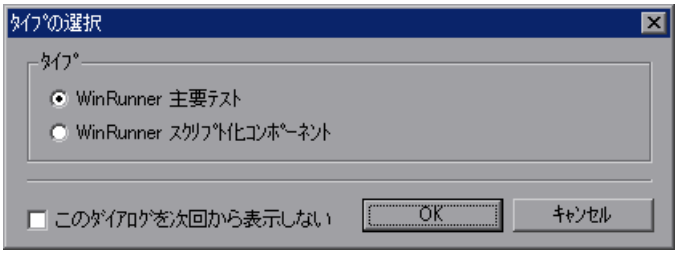
次回 Quality Center を起動したとき，またはビジネス・コンポーネント・モジュールのコンポーネント・ツリーを更新したときに，新しいスクリプトによるコンポーネントがツリーに表示されます。詳細については，『**Business Process Testing ユーザーズ・ガイド**』を参照してください。Quality Center プロジェクトへのテストの保存の詳細については，第 48 章「テスト工程の管理」を参照してください。

WinRunner のスクリプトによるコンポーネントを標準の保存タイプとして設定する

標準設定では，［**保存**］コマンドおよびツールバー・ボタンによって無題のドキュメントがテストとして保存されます。［**保存**］コマンドおよびツールバー・ボタンによって無題のドキュメントが WinRunner のスクリプトによるコンポーネントとして保存されるように，この標準設定を変更できます。

スクリプトによるコンポーネントを標準の保存タイプとして設定するには，次の手順を実行します。

1 新しいテストをスクリプトによるコンポーネントとして保存します。［タイプの選択］ダイアログ・ボックスが開きます。



2 ［**WinRunner スクリプト化コンポーネント**］を選択し，［**このダイアログを次回から表示しない**］チェック・ボックスを選択します。

次回新しいスクリプトを保存するときに，［**上書き保存**］コマンドまたはツールバー・ボタンをクリックすると，［Quality Center プロジェクトに WinRunner コンポーネントの保存］ダイアログ・ボックスが開きます。

ヒント：［**このダイアログを次回から表示しない**］チェック・ボックスを選択すると，今後新しいスクリプトによるコンポーネントまたはテストを保存するときに［タイプの選択］ダイアログ・ボックスが表示されず，［**保存**］ツールバー・ボタンまたはコマンドによる保存が常にダイアログ・ボックスで選択したタイプで行われるようになります。

［**このダイアログを次回から表示しない**］チェック・ボックスを選択した後で，再度［タイプの選択］ダイアログ・ボックスを表示する場合は，［**ツール**］＞［**一般オプション**］を選択し，［**タイプ保存ダイアログを表示する**］チェック・ボックスを選択します。

また，［**ファイル**］＞［**テストとして保存**］または［**ファイル**］＞［**スクリプト化コンポーネントとして保存**］を使用してコンポーネントまたはテストを保存することもできます。

スクリプトによるコンポーネントの変更

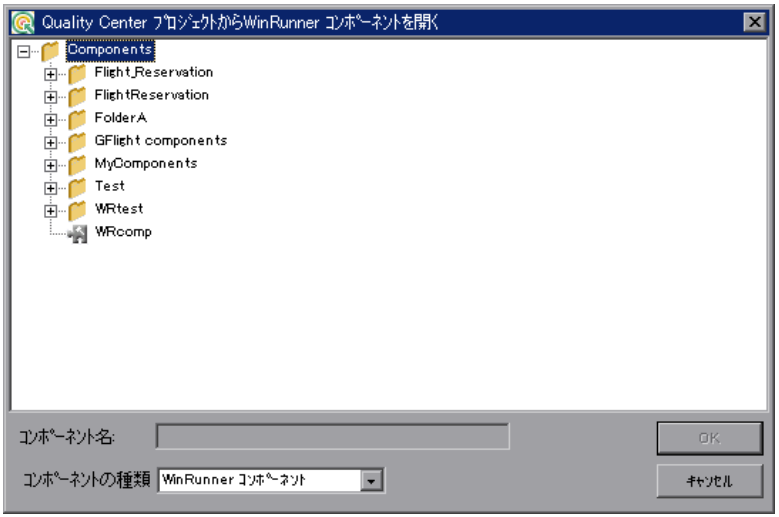
WinRunner が Quality Center プロジェクトに接続されているときは，プロジェクトに保存されているスクリプトによるコンポーネントを開き，コンポーネントの表示，変更，デバッグ，および実行が行えます。コンポーネントは，コンポーネント・ツリーの場所に従って見つけます。

注：Quality Center やほかの WinRunner セッションで現在開いているコンポーネントはロックされ，読み取り専用形式でのみ開くことができます。

スクリプトによるコンポーネントを変更するには，次の手順を実行します。

1 スクリプトによるコンポーネントが保存されている Quality Center プロジェクトに接続します。

2 ［**ファイル**］＞［**スクリプト化コンポーネントを開く**］を選択するか，CTRL キーを押しながら H キーを押します。［Quality Center プロジェクトから WinRunner コンポーネントを開く］ダイアログ・ボックスが開きます。



3 コンポーネント・ツリーで，該当するフォルダをクリックします。

4 ［**コンポーネントの種類**］を次の中から選択します。

- ［**WinRunner コンポーネント**］：選択したフォルダ内にあり，すでに WinRunner のスクリプトによるコンポーネントとして保存されたコンポーネントが表示されます。
- ［**非自動化コンポーネント**］：選択したフォルダ内にあり，Quality Center で作成されたが，マーキュリー・インタラクティブのテスト・ツールでまだ自動化されていないコンポーネントが表示されます。自動化されていないコンポーネントを WinRunner で開くと，そのコンポーネントは WinRunner コンポーネントに恒久的に変換されます。
- ［**すべてのコンポーネント**］：選択したフォルダ内にある WinRunner コンポーネントと自動化されていないコンポーネントがすべて表示されます。

5 コンポーネントを選択します。コンポーネントの名前が読み取り専用の［**コンポーネント名**］ボックスに表示されます。

6 ［**OK**］をクリックしてコンポーネントを開きます。自動化されていないコンポーネントを選択すると，そのコンポーネントは WinRunner コンポーネントに変換されます。

第 47 章

QuickTest Professional との統合

QuickTest Professional でテストを設計し，QuickTest テストから WinRunner テストと関数を呼び出すことによって，既存の WinRunner スクリプト・ライブラリを無駄にすることなく活用できます。WinRunner から QuickTest テストを呼び出すこともできます。

本章では，WinRunner から QuickTest テストを呼び出す方法について説明します。QuickTest から WinRunner テストと関数を呼び出す方法の詳細については，『**QuickTest Professional ユーザーズ・ガイド**』を参照してください。

本章では，次の内容について説明します。

- QuickTest Professional との統合について
- QuickTest テストの呼び出し
- 呼び出された QuickTest テストの結果の表示

QuickTest Professional との統合について

お使いのコンピュータに QuickTest Professional 6.0 以降がインストールされている場合は，WinRunner テストから QuickTest テストへの呼び出しを含めることができます。QuickTest Professional 6.5 がインストールされている場合は，QuickTest テストを呼び出し，テスト呼び出しの結果の詳細を表示できます。

WinRunner のテスト結果ウィンドウの統一レポート・ビューで QuickTest テスト実行の詳細結果を表示できます。

WinRunner は呼び出し先の QuickTest テストを実行すると，［QuickTest テストの設定］ダイアログ・ボックスの［**プロパティ**］タブで指定された関連アドインに応じて，テストに必要な QuickTest アドインが自動的にロードされます。

注：8.0 以前のバージョンの QuickTest を使用している場合は，WebTest アドインがロードされているときに QuickTest の Web アドインを使用する QuickTest テストを WinRunner テストから呼び出すことはできません。

QuickTest アドインを使った作業の詳細については，『**QuickTest Professional ユーザーズ・ガイド**』を参照してください。

WinRunner が QuickTest テストを含む Quality Center プロジェクトに接続されている場合，Quality Center プロジェクトに格納されている QuickTest テストを呼び出すことができます。

QuickTest テストの作成方法の詳細については，QuickTest Professional のマニュアルを参照してください。

QuickTest テストの呼び出し

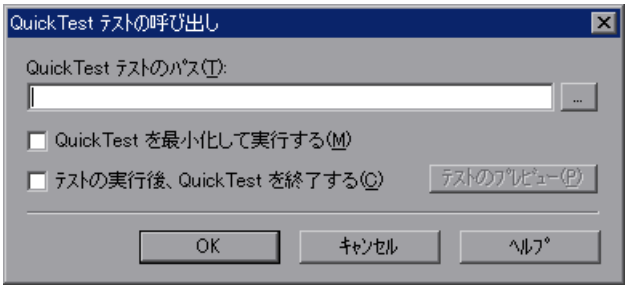
テストを実行するために，QuickTest から WinRunner に接続すると，WinRunner が起動されテストが開かれ（最小化または表示モードで），実行されます。WinRunner のテスト結果ウィンドウの統一レポート・ビューに QuickTest テスト実行の詳細結果が表示されます。

［QuickTest テストへの呼び出し］ダイアログ・ボックスを使用するか，手作業で **call_ex** ステートメントを入力することによって QuickTest テストへの呼び出しを挿入できます。

注：QuickTest テストへの呼び出しを含む WinRunner テストを呼び出すことはできません。

［QuickTest テストの呼び出し］ダイアログ・ボックスを使用して QuickTest テストへの呼び出しを挿入するには，次の手順を実行します。

1 ［**挿入**］＞［**QuickTest テストの呼び出し**］を選択します。［QuickTest テストの呼び出し］ダイアログ・ボックスが開きます。



2 ［**QuickTest テストのパス**］ボックスに QuickTest テストのパスを入力するか，参照ボタンを使って QuickTest テストを指定します。

Quality Center に接続している場合，参照ボタンをクリックすると，Quality Center プロジェクトからテストを選択するための，［Quality Center プロジェクトからテストを開く］ダイアログ・ボックスが開きます。このダイアログ・ボックスの詳細については，第 48 章「テスト工程の管理」を参照してください。

3 QuickTest ウィンドウがテストの実行中に表示されないようにするには，［**QuickTest を最小化して実行する**］を選択します （このオプションのサポート対象は，QuickTest バージョン 6.5 以降です)。

4 QuickTest テストを呼び出すステップが完了したときに QuickTest アプリケーションを終了させるには，［**テスト終了後、QuickTest を閉じる**］を選択します。

5 ［**OK**］をクリックし，ダイアログ・ボックスを閉じます。次に示すような **call_ex** ステートメントがテストに挿入されます。

```
call_ex("F:¥¥Merc_Progs¥¥QTP¥¥Tests¥¥web¥¥short_flight",1,1);
```

call_ex 関数の構文は次のとおりです。

```
call_ex ( QT_test_path [ , run_minimized, close_QT] );
```

注：WinRunner 7.5 で提供された **call_ex** ステートメントは，この関数の 7.6 バージョンで返される値とは異なる値を返します。WinRunner 7.5 で作成された，この関数の戻り値を使うテストがある場合，新しい戻り値を反映するにはテストの変更が必要になる場合があります。これらのメソッドの詳細については，「**TSL リファレンス**」を参照してください。

---

call_ex 関数の詳細とその使用例については，「**TSL リファレンス**」を参照してください。

呼び出された QuickTest テストの結果の表示

WinRunner テストの実行結果は，WinRunner レポート・ビューまたは統一レポート・ビューで表示できます。ただし，呼び出された テスト（バージョン 6.5 以降）に関する詳細情報を表示するには，テストの実行前に，統一レポート情報を生成してテスト結果を表示するときに統一レポートが表示されるよう WinRunner を設定しておく必要があります。

統一レポート情報を作成し，統一レポートが表示されるようにするには，次の手順を実行します。

1 テストを実行する前に（またはテスト結果が表示される前に），［**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・ボックスが開きます。

2 ［**実行**］カテゴリをクリックします。

3 テスト実行前に統一レポート情報が作成されるようにするには，［**統一レポート ビュー**］を選択するか，［**WinRunner レポート ビュー**］を選択して［**統一レポート情報を生成する**］オプションを選択します。

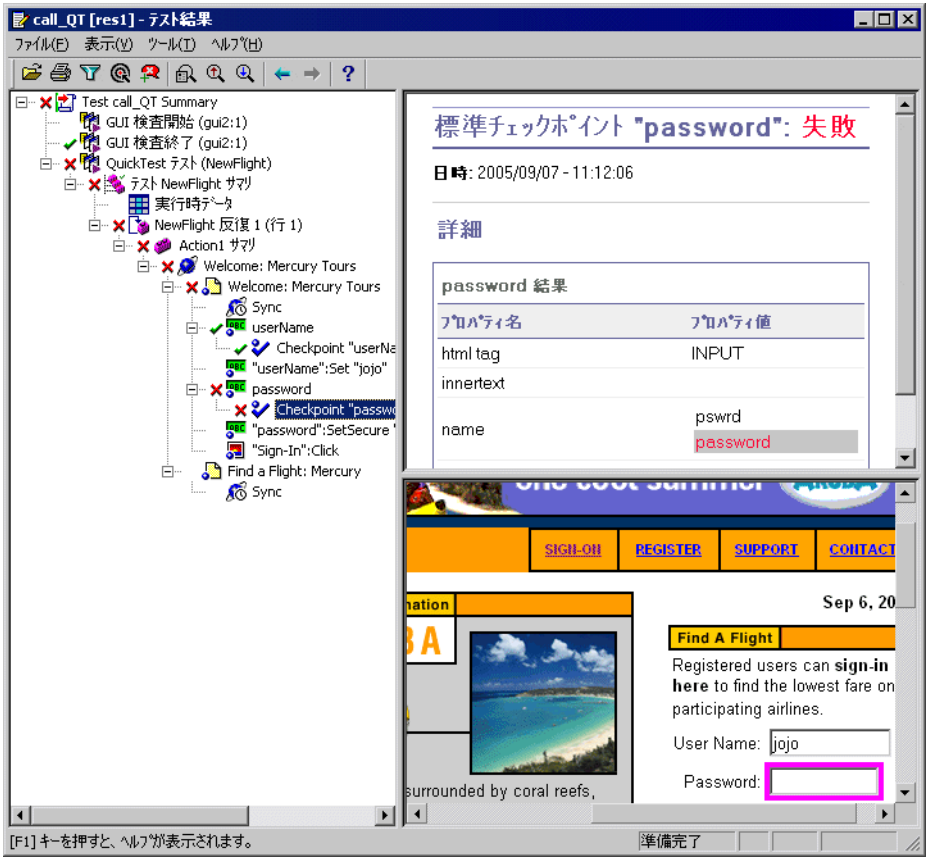
統一レポート情報を表示するには，［テスト結果］ウィンドウを開く前に［**統一レポート ビュー**］を選択します。

詳細については，第 20 章「テスト結果の分析」を参照してください。

呼び出された QuickTest テストの結果の分析

WinRunner のテスト結果ウィンドウの統一レポート・ビューには，WinRuner テストの各イベントのノードと呼び出された QuickTest テストの各ステップのノードが含まれます。

QuickTest ステップに対応するノードを選択すると，右側の表示枠にステップの詳細が表示されます。ステップが実行されたときのアプリケーションの画面ショットが含まれることもあります。



注：呼び出された QuickTest テストの結果は，WinRunner テストの結果フォルダから WinRunner 統一レポート・ビューでのみ表示できます。QuickTest テストの結果は，呼び出された QuickTest テスト・フォルダの下に保存されます。

QuickTest テストへの呼び出しを含む WinRunner テストの結果を分析する場合は，次を参照できます。

➤ WinRunner テストのサマリ結果を表示するには，**start run** ノードを選択します。このサマリはテストの実行のステータスを示しますが，テスト内の WinRunner ステップのサマリ・チェックポイント情報だけが含まれます。

➤ WinRunner イベントの結果を表示するには，他の WinRunner テストと同様，［**WinRunner**］ノードを選択します。

➤ 呼び出された QuickTest テストのサマリ結果を表示するには，QuickTest **Test** ノードを選択します。このサマリには，QuickTest テスト実行のステータス，QuickTest テストに含まれるチェックポイントに関する統計情報が含まれます。

➤ QuickTest テストの結果のデータ・テーブルを表示するには，［QuickTest **Run-Time Data**］ノードを選択します。データ・テーブル・パラメータで使用されたデータや，テストの実行中にテスト内の出力値によってテーブルに格納されたデータなどが含まれます。

➤ テスト反復のサマリ情報を表示するには，**iteration** ノードを選択します。

➤ アクションのサマリ情報を表示するには，**action** ノードを選択します。

➤ 選択したステップの結果に関する情報を表示するには，QuickTest ステップ・ノードを選択します。選択したステップにキャプチャされた画面がある場合は，キャプチャされた画面が［テスト結果ウィンドウ］の右側の表示枠に表示されます。

標準設定では，失敗したステップの画面だけがキャプチャされます。［QuickTest オプション］ダイアログ・ボックスの［**実行**］タブで［**ステップ画面キャプチャの保存先テスト結果**］オプションを変更できます。

様々な QuickTest テスト・ステップに提供されるデータの詳細については，『**QuickTest Professional ユーザーズ・ガイド**』を参照してください。

WinRunner テスト結果の分析の詳細については，441 ページ「テスト結果の分析」を参照してください。

第 48 章

テスト工程の管理

ソフトウェアのテストは一般に，何千ものテストを作成して実行するという作業が伴います。TestDirector を基盤とする Quality Center は，Mercury の品質管理ソリューションです。Quality Center を WinRunner と一緒に使用することで，テスト工程を整理して制御できます。

本章では，以下の項目について説明します。

- テスト工程の管理について
- テスト工程の統合
- Quality Center から WinRunner のテストへのアクセス
- プロジェクトの接続と接続の解除
- プロジェクトへのテストの保存
- テストのプロジェクトへのスクリプト化コンポーネントとしての保存
- プロジェクトのテストを開く
- GUI マップ・ファイルのプロジェクトへの保存
- GUI マップ・ファイルのプロジェクトへの保存
- プロジェクトの GUI マップ・ファイルを開く
- テスト・セット内のテストの実行
- リモート・ホストでのテストの実行
- テスト結果のプロジェクトからの表示
- TSL 関数の Quality Center での使用
- Quality Center で使用するコマンドライン・オプション

テスト工程の管理について

Quality Center は，テスト工程を系統立てて制御できる強力なテスト管理ツールです。テスト工程の枠組みや基盤を作成できます。

Quality Center では，アプリケーション機能の特徴すべてを補うテスト・プロジェクトを管理できます。すべてのテストが，アプリケーションで指定されたテスト要件を満たすように設計されています。プロジェクトの目的を達成するには，プロジェクトのテストを一意のグループに整理します。Quality Center は，テストのスケジューリングや実行，テスト結果の収集，結果の分析などにおいて，直感的で効果的な方法を提供します。

既存の WinRunner テストをスクリプト化コンポーネントとして保存したり，ビジネス・プロセス・テスティングで使用される新規スクリプト化コンポーネントを作成したりすることができます。ビジネス・プロセス・テスティングは，SME（各分野のエキスパート）が開発サイクルの初期の段階で，またスクリプト記述を必要としない環境で，アプリケーションの品質保証テストを設計できるようにする，Quality Center のモジュールです。ビジネス・プロセス・テスティングはテストに新しい方法論を使用します。また WinRunner と一緒に使用することで，より優れた自動テスト環境において様々な利点を提供します。

不具合の追跡を行うシステムも特徴の 1 つで，不具合の検出から解決まで監視することができます。

WinRunner は，TestDirector 7.x および 8.0，および Quality Center と一緒に使用できます。

TestDirector のバージョン 7.5 以降および Quality Center では，バージョン・コントロール・サポートが提供されます。バージョン・コントロールは，各テストの古いバージョンを保持しつつ，自動テスト・スクリプトを更新し，改訂できる機能です。これにより，テスト・スクリプトに行われた変更を追跡でき，スクリプト間で変更された点を確認したり，テスト・スクリプトの前のバージョンに戻したりすることができます。バージョン・コントロール・サポートの詳細については，979 ページ「GUI マップ・ファイルのプロジェクトへの保存」を参照してください。

注：本章では，WinRunner と Quality Center の統合について説明します。Quality Center を使った作業の詳細については，『**Mercury Quality Center ユーザーズ・ガイド**』を参照してください。スクリプト化コンポーネントを使った作業の詳細については，『**Business Process Testing ユーザーズ・ガイド**』を参照してください。

テスト工程の統合

Quality Center と WinRunner を併用して，テスト工程のあらゆる作業を統合できます。WinRunner ではスクリプト化コンポーネントとテストを作成して，Quality Center プロジェクトに保存できます。コンポーネントは，ビジネス・プロセス・テストに含めることができるようになります。テストの実行後，Quality Center で結果を表示したり分析したりできます。

Quality Center は，テストと不具合情報をプロジェクト・データベースに格納します。Quality Center プロジェクトは，Microsoft Access，Oracle，Sybase，Microsoft SQL のいずれかに作成できます。これらのプロジェクトには現在のテスト・プロジェクト（テスト，テストの実行結果，報告された不具合など）に関する情報が格納されます。

プロジェクトに WinRunner からアクセスできるようにするには，WinRunner を Quality Centerがインストールされている Webサーバに接続する必要があります。

WinRunner が Quality Center に接続されている場合，テストをテスト計画マネージャと関連付けて保存することができます。ローカル・ホストまたはリモート・ホスでのテスト実行のスケジュールを立てることができます。テストの実行結果は，Quality Center プロジェクトに直接送信されます。

注：リモート・マシンから Quality Center に WinRunner テストを実行させるには，WinRunner の［**Quality Center でテストをリモート実行する**］オプションを有効にする必要があります。標準設定では，このオプションは無効になっています。このオプションは［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**］カテゴリで有効にできます。この設定の詳細については，23「Setting Global Testing Options」を参照してください。

Quality Center から WinRunner のテストへのアクセス

Quality Center から WinRunner のテストにアクセスする際，テストがプロジェクトのデータベースからローカルの一時ディレクトリにダウンロードされ，そこが現在の作業用ディレクトリとなります。テストが別のファイル（例えば，モジュールやテスト）を呼び出して，呼び出されたファイルの完全パス名が指定されないと，現在の作業ディレクトリは，参照されたファイルの相対パスとなります。そのため，WinRunner は呼び出されたテストを開くことができません。

例えば，テストが flt_lib file を呼び出した場合，次のようになります。

```
static lib_path = getvar("testname") & "¥¥..¥¥flt_lib";
reload(lib_path);
```

WinRunner は，相対パスから呼び出されたテストを探します。WinRunner が正しいパス名を検索できるようにするには，次のようにします。

- WinRunner が呼び出したファイルのパス名を変更します。
- WinRunner のテストすべてに対して，直接編成ファイルを設定します（LAN のみ）。

ファイルのパス名の変更

WinRunner が，テストから呼び出されたファイルにアクセスできるようにするには，Quality Center プロジェクトにファイルを保存してから，WinRunner のテスト・スクリプトのパス名を変更します。

例えば，subject¥¥module の下の Quality Center プロジェクトに flt_lib file を保存するとします。Quality Center は次のステートメントを使ってファイルを呼び出します。

```
static lib_path = "[QC]¥¥Subject¥¥module¥¥flt_lib";
```

Quality Center プロジェクトへのテストの保存については，968 ページ「プロジェクトへのテストの保存」を参照してください。

WinRunner のテスト・ディレクトリへのアクセス（LAN のみ）

ローカル・エリア・ネットワーク（LAN）環境で作業している場合，ディレクトリ・パスに関係なく，WinRunner のすべてのテストにアクセスする直接編成ファイルを提供するよう，コンピュータに設定できます。これにより，呼び出されたテストのディレクトリ・パスを変更しないで，Quality Center から WinRunner のテストを実行できます。

直接編成ファイルのアクセス・オプションを設定するには，次の手順を実行します。

1 WinRunner がインストールされているコンピュータの［**スタート**］メニューから［**実行**］をクリックします。［実行］ダイアログ・ボックスが開きます。

2 regedit と入力して［**OK**］をクリックします。［レジストリ エディタ］が開きます。

3 次のフォルダを指定します。

［マイ コンピュータ］＞［HKEY_LOCAL_MACHINE］＞［ソフトウェア］＞［Mercury Interactive］＞［TestDirector］＞［Testing Tools］＞［WinRunner］

4 **WinRunner フォルダで，［DirectFileAccess］**をダブルクリックします。［Value Data］ボックスの値を Y に変更します。

ヒント：ダイレクト・アクセスのオプションを設定すると，Quality Center から WinRunner のテストにアクセスする際の Web アクセス・パフォーマンスが向上します。

プロジェクトの接続と接続の解除

WinRunner と Quality Center の両方を使っている場合，WinRunner は Quality Center プロジェクトと通信を行うことができます。テスト工程のどの時点でも WinRunner と Quality Center プロジェクトの接続を有効にしたり，無効にしたりできます。ただし，Quality Center から WinRunner のテストを実行中は接続を無効にしないでください。

接続工程には 2 つの段階があります。まず，WinRunner を Quality Center サーバに接続します。このサーバは，WinRunner と Quality Center のプロジェクト間の接続を処理します。次に，WinRunner でアクセスしたいプロジェクトを選択します。プロジェクトにはテスト対象のアプリケーションのテストとテスト実行情報が格納されます。Quality Center プロジェクトは，パスワードで保護されているのでユーザ名とパスワードを入力しなければなりません。

WinRunner の Quality Center への接続

WinRunner をプロジェクトに接続する前に，WinRunner をサーバに接続しなくてはなりません。詳細については，961 ページ「テスト工程の統合」を参照してください。

WinRunner を Quality Center に接続するには，次の手順を実行します。

1 ［**ツール**］＞［**Quality Center への接続**］を選択します。

［Quality Center への接続］ダイアログ・ボックスが開きます。

2 ［**サーバ**］ボックスに，Quality Center がインストールされている Web サーバの URL を入力します。

注：LAN（ローカル・エリア・ネットワーク）または WAN（広域エリア・ネットワーク）経由でアクセス可能な Web サーバを選択できます。

---

3 ［**サーバ接続**］セクションで，［**接続**］をクリックします。

サーバへの接続が確立されると，［**サーバ**］ボックスにサーバ名が読み取り専用形式で表示されます。

4 ［**ドメイン**］ボックスで，Quality Center プロジェクトを含むドメインを入力するか選択します（TestDirector のバージョン 7.5 以前を使用している場合は，［**ドメイン**］ボックスは表示されません。次のステップに進んでください）。

5 ［**プロジェクト**］ボックスで，Quality Center プロジェクト名を入力するか，リストから選択します。

6 ［**ユーザ名**］ボックスに，選択したプロジェクトを開くユーザ名を入力します。

7 ［**パスワード**］ボックスに，選択したプロジェクトのパスワードを入力します。

8 ［**プロジェクト接続**］セクションで，［**接続**］をクリックして，選択したプロジェクトに WinRunner を接続します。

選択したプロジェクトとの接続が確立すると，サーバとプロジェクト接続の詳細が読み取り専用で表示されます。

次に WinRunner を起動したときに自動的に Quality Center サーバと選択したプロジェクトに再接続するには，［**起動時に再接続する**］チェック・ボックスを選択します。

［**起動時に再接続する**］チェック・ボックスが選択されていない場合は，次にスクリプト化コンポーネントを作成しようとしたときに Quality Center プロジェクトに接続するかどうかメッセージが表示されます。

［**起動時に再接続する**］チェック・ボックスが選択されている場合は，［**起動時に再接続できるようにパスワードを保存する**］チェック・ボックスが有効になります。次に WinRunner を起動したときに再接続する際のパスワードを保存するには，［起動時に再接続するためにパスワードを保存］チェック・ボックスを選択します。

パスワードを保存していない場合は，起動時に WinRunner が Quality Center に接続する際にパスワードの入力を促されます。

注：［起動時に再接続する］が選択されていても Quality Center に接続せずに WinRunner を開く場合は，**dont_connect** コマンドライン・オプションを使います。詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

---

9 ［**閉じる**］をクリックして，［Quality Center に接続］ダイアログ・ボックスを閉じます。

注：- *qc_connection*, - *qc_database_name*, - *qc_password*, - *qc_server_name*, - *qc_user_name* などのコマンドライン・オプションを使って，WinRunner を Quality Center サーバおよびプロジェクトに接続することもできます。これらのオプションの詳細については，989 ページ「Quality Center で使用するコマンドライン・オプション」. を参照してください。コマンドライン・オプションの使用の詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

Quality Center からの切断

Quality Center プロジェクトまたはサーバから切断できます。WinRunner をまずプロジェクトから切断せずにサーバから切断すると，WinRunner のデータベースへの接続は自動的に切断されます。

注：Quality Center からの切断時にテストまたはスクリプト化コンポーネントが開いている場合は，WinRunner によって閉じられます。

Quality Center から WinRunner を切断するには，次の手順を実行します。

1 ［**ツール**］＞［**Quality Center への接続**］を選択します。

［Quality Center への接続］ダイアログ・ボックスが開きます。

2 ［**プロジェクトへの接続**］セクションで，［**切断**］をクリックして，選択したプロジェクトと WinRunner との接続を解除します。同じサーバを使用して別のプロジェクトを開く場合は，965 ページの手順 5 に示すようにしてプロジェクトを選択します。

3 Quality Center サーバから切断するには，［**サーバへの接続**］セクションで，［**切断**］をクリックします。

4 ［**閉じる**］をクリックして，［Quality Center への接続］ダイアログ・ボックスを閉じます。

プロジェクトへのテストの保存

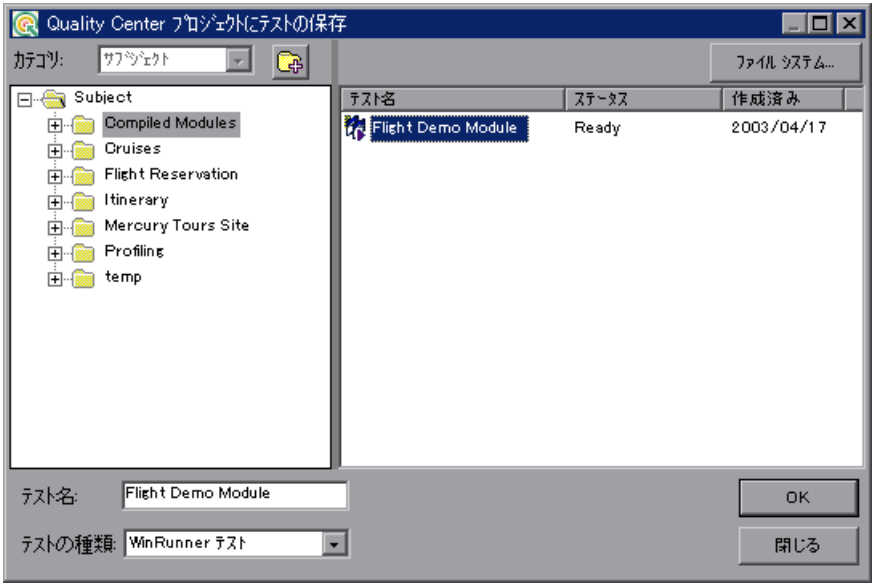
WinRunner が Quality Center プロジェクトに接続されている場合，WinRunner で新しいテストを作成して，それをプロジェクトに直接保存することができます。テストを保存するには，分かりやすい名前を与え，テスト計画ツリーの関連するサブジェクトに関連付けます。こうすることで，各サブジェクトに作成されたテストを追跡したり，テストの計画と作成の進捗をすばやく表示したりするのが簡単になります。

注：Quality Center にテストをスクリプト化コンポーネントとして保存できます。詳細については，971 ページ「テストのプロジェクトへのスクリプト化コンポーネントとしての保存」を参照してください。

Quality Center プロジェクトにテストを保存するには，次の手順を実行します。

1 ［**ファイル**］＞［**テストとして保存**］を選択するか，［**保存**］ボタンをクリックします。ファイル・システムにすでに保存されているテストの場合は，［**ファイル**］＞［**上書き保存**］を選択します。

WinRunner が Quality Center プロジェクトに接続されている場合，［Quality Center プロジェクトにテストの保存］ダイアログ・ボックスが開き，テスト計画ツリーが表示されます。



［Quality Center プロジェクトへテストを保存］ダイアログ・ボックスは，WinRunner が Quality Center プロジェクトに接続されているときのみ表示されます。

注：テストをファイル・システムに直接保存するには，［**ファイルシステム**］ボタンをクリックします。これにより，［**テストを保存**］ダイアログ・ボックスが開きます（［Quality Center］ボタンをクリックすると，［**テストを保存**］ダイアログ・ボックスから［Quality Center プロジェクトへテストを保存］ダイアログ・ボックスに戻ります）。

ファイル・システムにテストを直接保存すると，テストは Quality Center プロジェクトに保存されません。

---

2 対象となるサブジェクトを，テスト計画ツリーから選択します。ツリーを広げて下位レベルを表示するには，閉じているフォルダをダブルクリックします。下位レベルを閉じるには，開いているフォルダをダブルクリックします。

3 ［**テスト名**］ボックスに，テストの名前を入力します。テストを識別しやすいように，分かりやすい名前を指定しましょう。

4 ［**OK**］をクリックしてテストを保存し，ダイアログ・ボックスを閉じます。

注：バッチ・テストを保存する場合は，［**テストの種類**］ボックス［**WinRunner バッチ テスト**］を選択します。

次回 Quality Center を起動すると，Quality Center のテスト計画ツリーに新しいテストが表示されます。詳細については『**Mercury Quality Center ユーザーズ・ガイド**』を参照してください。

テストのプロジェクトへのスクリプト化コンポーネントとしての保存

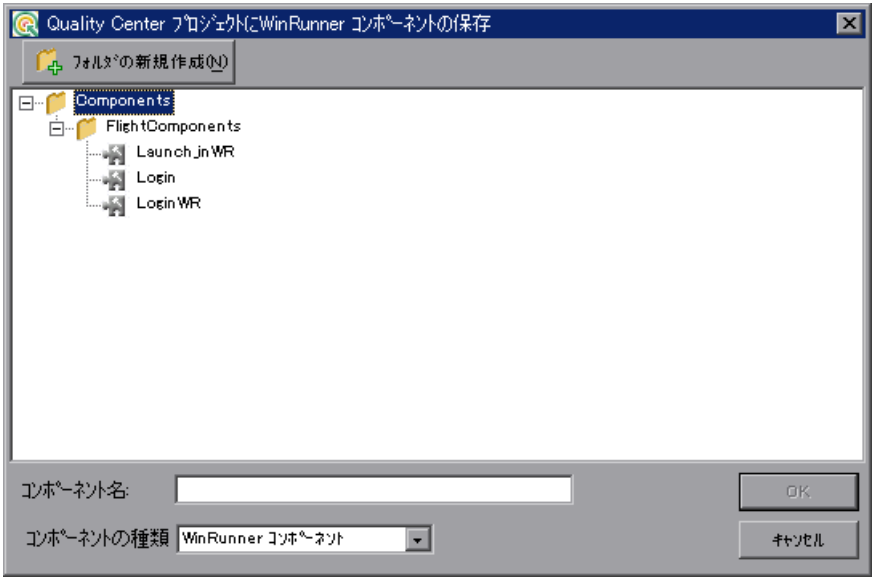
Quality Center で作業する場合，WinRunner で作成した新規または既存のテストをスクリプト化コンポーネントとして直接プロジェクトに保存できます。コンポーネントは，Quality Center の 1 つまたは複数のビジネス・プロセス・テストに含めることができます。コンポーネント，入力および出力パラメータの詳細の定義および画面ショットの添付も行えます。

テストのスクリプト化コンポーネントとしての保存

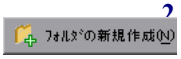
WinRunner テストをスクリプト化コンポーネントとして Quality Center のビジネス・コンポーネント・モジュールに保存できます。

テストを Quality Center プロジェクトにスクリプト化コンポーネントとして保存するには，次の手順を実行します。

1 Quality Center プロジェクトに接続した後，［**ファイル**］＞［**スクリプト化コンポーネントとして保存**］を選択します。［Quality Center プロジェクトに WinRunner コンポーネントの保存］ダイアログ・ボックスが開き，コンポーネント・ツリーが表示されます。



注 : ［ファイル］メニューの［スクリプト化コンポーネントとして保存］オプションは，ビジネス・プロセス・テスティングをサポートする Quality Center に接続している場合のみ可視になります。



2 コンポーネント・ツリーで関連するフォルダを選択するか，［**フォルダの新規作成**］ボタンをクリックして，新しいフォルダを作成します。ツリーを展開するには閉じているアイコンをダブルクリックします。サブレベルを折りたたむには，開いているフォルダ・アイコンをダブルクリックします。

3 ［**コンポーネント名**］テキスト・ボックスにスクリプト化コンポーネントの名前を入力します。内容の分かりやすい名前を使用すると，コンポーネントを特定しやすくなります。

4 ［**OK**］をクリックして，コンポーネントを保存し，ダイアログ・ボックスを閉じます。

次回 Quality Center を起動すると，またはビジネス・コンポーネント・モジュールでコンポーネント・ツリーを更新すると，新規スクリプト化コンポーネントがツリーに表示されます。詳細については『**Business Process Testing ユーザーズ・ガイド**』を参照してください。

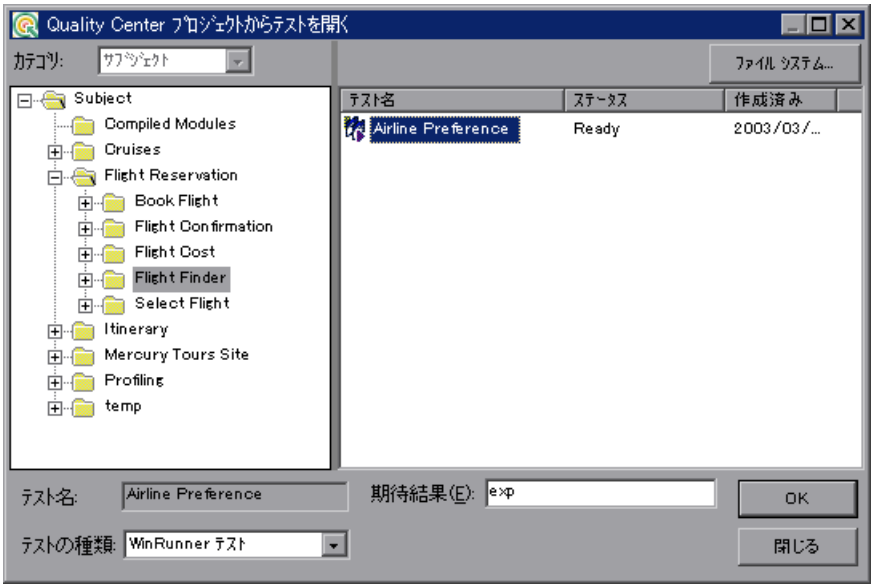
プロジェクトのテストを開く

WinRunner が Quality Center プロジェクトに接続されている場合，プロジェクトに含まれる自動テストを開くことができます。テストを探すには，ファイル・システムにおけるテストの実際の格納場所ではなく，テスト計画ツリーの中での位置を探します。

Quality Center プロジェクトに保存されているテストを開くには，次の手順を実行します。

1 ［**ファイル**］＞［**テストを開く**］を選択するか［**開く**］ボタンをクリックします。

［Quality Center プロジェクトからテストを開く］ダイアログ・ボックスが開き，テスト計画ツリーが表示されます。



［Quality Center データベースからテストを開く］ダイアログ・ボックスは，WinRunner が Quality Center プロジェクトに接続されているときだけ開きます。

注：テストをファイル・システムから直接開くには，［**ファイル システム**］ボタンをクリックします。これにより，［**テストを開く**］ダイアログ・ボックスが開きます（［Quality Center］ボタンをクリックすると，［**テストを開く**］ダイアログ・ボックスから［Quality Center データベースからテストを開く］ダイアログ・ボックスに戻ります）。

テストをファイル・システムから開く場合，そのテストを実行してもテスト実行のイベントは Quality Center プロジェクトには書き込まれません。

---

2 対象となるサブジェクトを，テスト計画ツリーから選択します。ツリーを広げて下位レベルを表示するには，閉じているフォルダをダブルクリックします。下位レベルを閉じるには，開いているフォルダをダブルクリックします。

サブジェクトを選択すると，そのサブジェクトに属するテストが［**テスト名**］リストに表示されます。

3 右側の枠の［**テスト名**］リストからテストを選択します。テストが［**テスト名**］ボックスに読み取り専用で表示されます。

4 ［**OK**］をクリックして，テストを開きます。WinRunner のウィンドウにテストが表示されます。テスト・ウィンドウのタイトル・バーには，サブジェクトの完全パスが表示されます。

注：バッチ・テストを保存する場合は，［**テストの種類**］ボックス［**WinRunner バッチテスト**］を選択します。バッチ・テストの詳細については，第 36 章「バッチ・テストの実行」を参照してください。

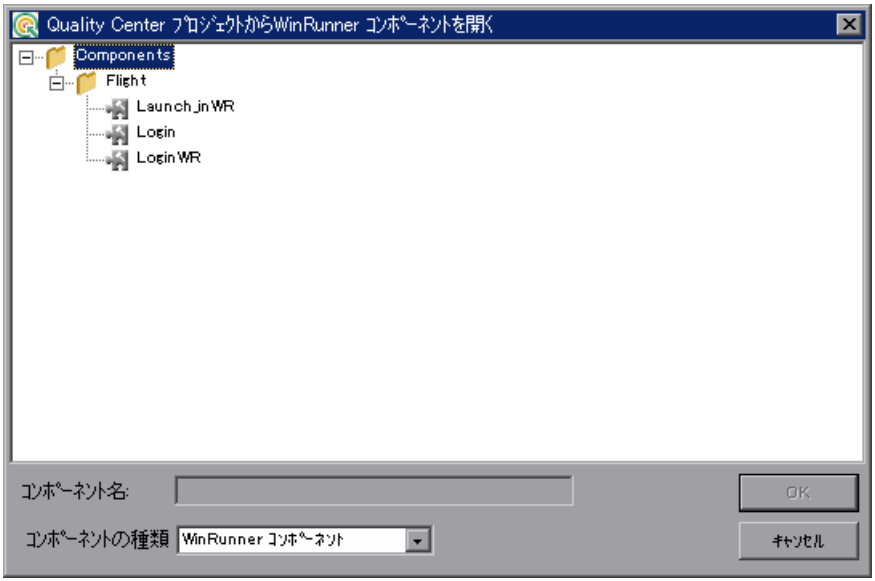
プロジェクト内でスクリプト化コンポーネントを開く

WinRunner が Quality Center プロジェクトに接続されていれば，プロジェクトに含まれる既存のスクリプト化コンポーネントを開くことができます。

Quality Center プロジェクトからスクリプト化コンポーネントを開くには，次の手順を実行します。

1 Quality Center プロジェクトに接続されている場合は，［**ファイル**］＞［**スクリプト化コンポーネントを開く**］を選択します。［Quality Center プロジェクトか

ら WinRunner コンポーネントを開く］ダイアログ・ボックスが開き，コンポーネント・ツリーが表示されます。



注：［ファイル］メニューの［スクリプト化コンポーネントを開く］オプションは，Business Process Testing サポート付きの Quality Center に接続されている場合のみ表示されます。

---

2 コンポーネント・ツリーから関連するコンポーネントを選択します。ツリーを展開してサブレベルを表示するには，閉じているフォルダをダブルクリックします。ツリーを折りたたむには，開いているフォルダをダブルクリックします。スクリプト化コンポーネントが読み取り専用の［**コンポーネント名**］ボックスに表示されます。

3 ［**OK**］をクリックして，スクリプト化コンポーネントを開きます。WinRunner. のウィンドウでコンポーネントが開きます。WinRunner のタイトル・バーにスクリプト化コンポーネントの完全サブジェクト・パスが表示されます。

注：この手順でマニュアル・コンポーネントを開くこともできます。マニュアル・コンポーネントとは，特定のテスト・ツール形式に変換されていない，Quality Center で作成されたコンポーネントです。マニュアル・コンポーネントを WinRunner で開くと，WinRunner コンポーネントとして恒久的に設定されます。この操作はコンポーネントが WinRunner で保存されていなくても，元に戻すことはできません。

WinRunner でのテストのバージョン管理

WinRunner がバージョン管理サポート付きの Quality Center プロジェクトに接続されている場合は，各テストの旧バージョンを保持しながら自動テスト・スクリプトを更新したり改訂したりできます。これにより，各テスト・スクリプトの変更を追跡したり，スクリプトのバージョン間の変更を確認したり，テスト・スクリプトの前のバージョンに戻したりすることができます。

注：バージョン管理サポート付きの Quality Center プロジェクトには，Quality Center のバージョン管理ソフトウェア・コンポーネントのほかにバージョン管理ソフトウェアをインストールする必要があります。Quality Center バージョン管理アドインの詳細については，『**Quality Center インストール・ガイド**』を参照してください。

テストのバージョンは，バージョン管理データベースにチェック・インしたり，バージョン管理データベースからチェック・アウトすることによって管理します。

バージョン管理データベースへのテストの追加

バージョン管理データベースに初めてテストを追加すると，そのテストが「**作業中のテスト**」となり，恒久的なバージョン番号が割り当てられます。

作業中のテストはテスト・リポジトリに含まれ，Quality Center のすべてのテスト実行で使用されます。

注：通常，テストの最新バージョンは，作業中のテストになりますが，Quality Center で任意のバージョンを作業中のテストとして指定できます。詳細については，Quality Center のマニュアルを参照してください。

新規テストをバージョン管理データベースに追加するには，次の手順を実行します。

1 ［**ファイル**］＞［**チェック イン**］を選択します。

注：［ファイル］メニューの［チェック イン］および［チェック アウト］オプションは，バージョン管理サポート付きの Quality Center プロジェクトに現在接続されている場合のみ表示されます。［チェック イン］オプションは，アクティブ・スクリプトがプロジェクト・データベースに保存された場合のみ有効になります。

---

2 ［**OK**］をクリックして，バージョン管理データベースにテストを追加します。

3 ［**OK**］をクリックして，チェック・インしたテストを再度開きます。テストが閉じ，読み取り専用ファイルとして再度開きます。

アクティブ・テストに未保存の変更がある場合は，テストを保存するようメッセージが表示されます。

チェック・インしたテストを検討します。テストを実行して，結果を見ることもできます。ただし，テストがチェック・インされている間は，スクリプトに変更はできません。

変更しようとすると，スクリプトがチェック・アウトされていないため変更できないという内容のメッセージが表示されます。

バージョン管理データベースからのテストのチェック・アウト

現在バージョン管理データベースにチェック・インされているテストを開いた場合は，スクリプトに変更することはできません。このスクリプトを変更するには，スクリプトをチェック・アウトしなければなりません。

テストをチェック・アウトすると，Quality Center はテストの**最新バージョン**を一意のチェックアウト・ディレクトリにコピーし（テストを始めてチェック・アウトしたときには自動的に作成されます），プロジェクト・データベースにテストをロックします。これにより，Quality Center プロジェクトの他のユーザが同じテストに変更を上書きできないようになります。

テストをチェック・アウトするには，次の手順を実行します。

1 ［**ファイル**］＞［**チェック アウト**］を選択します。

2 ［**OK**］をクリックします。読み取り専用のテストが閉じ，書き込み可能なスクリプトとして自動的に再度開きます。

注：チェック・アウト・オプションは，アクティブ・スクリプトがプロジェクトのバージョン管理データベースで現在チェック・インされている場合のみ有効になります。

スクリプトは，スクリプトに変更する場合，あるいはスクリプトの実行可能性をテストする場合のみ，バージョン管理データベースからチェック・アウトします。

バージョン管理データベースへのテストのチェック・イン

テストの変更が完了したら，テストをバージョン管理データベースにチェック・インしてこれを新しい**最新バージョン**とし，**作業中テスト**として割り当てます。

テストをバージョン管理データベースに再度チェック・インすると，Quality Center はチェックアウト・ディレクトリからテストのコピーを削除し，そのバージョンを Quality Center プロジェクトの他のユーザが使用できるように，データベースでテストのロックを解除します。

テストをチェック・インするには，次の手順を実行します。

1 ［**ファイル**］＞［**チェックイン**］を選択します。

2 ［**OK**］をクリックします。ファイルが閉じ，読み取り専用スクリプトとして自動的に再度開きます。

スクリプトをチェック・インした後で，テストを実行すると，結果は Quality Center プロジェクト・データベースに保存されます。

ヒント：Quality Center を使用してテストのチェック・イン / チェック・アウトのステータスを変更せずに WinRunner でテストを閉じます。テストが WinRunner で開いているときに Quality Center を使用してテスト・ステータスに変更を行うと，WinRunner はこれらの変更を反映しません。詳細については，Quality Center のマニュアルを参照してください。

GUI マップ・ファイルのプロジェクトへの保存

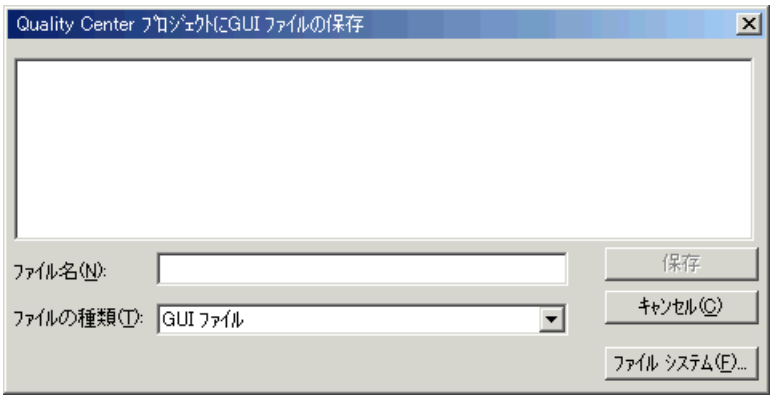
WinRunner と Quality Center プロジェクトが接続されている場合は，［GUI マップエディタ］で［ファイル］＞［上書き保存］を選択して GUI マップ・ファイルを開いているデータベースに保存します。Quality Center プロジェクトに保存されているすべてのテストで使用されるすべての GUI マップ・ファイルは一緒に格納されます。こうすることで，プロジェクトのテストに関連付けられた GUI マップ・ファイルの追跡が簡単になります。

GUI マップ・ファイルを Quality Center プロジェクトに保存するには，次の手順を実行します。

1 ［**ツール**］＞［**GUI マップエディタ**］を選択して，［**GUI マップエディタ**］を開きます。

2 仮 GUI マップ・ファイルで，［**ファイル**］＞［**名前を付けて保存**］を選択します。既存の GUI マップ・ファイルで，［**ファイル**］＞［**上書き保存**］を選択します。

［Quality Center プロジェクトに GUI ファイルを保存］ダイアログ・ボックスが開きます。GUI マップが開いているプロジェクトにすでに保存されている場合，これらはダイアログ・ボックスにリストされます。



［Quality Center プロジェクトに GUI ファイルの保存］ダイアログ・ボックスは，WinRunner が Quality Center プロジェクトに接続されている場合のみ開くことに注意してください。

GUI マップ・ファイルをファイル・システムに直接保存するには，［**ファイル システム**］ボタンをクリックします。これにより，［**GUI ファイルを保存**］ダイアログ・ボックスが開きます（［Quality Center］ボタンをクリックすると，［**GUI ファイルを保存**］ダイアログ・ボックスから［Quality Center プロジェクトに GUI ファイルを保存］ダイアログ・ボックスに戻ります）。

注：GUI マップ・ファイルを直接ファイル・システムに保存する場合は，GUI マップ・ファイルは Quality Center プロジェクトには保存されません。

---

3 ［**ファイル名**］テキスト・ボックスに，GUI マップ・ファイルの名前を入力します。GUI マップ・ファイルを識別しやすいように分かりやすい名前を使用します。

4 ［**保存**］をクリックして GUI マップ・ファイルを保存し，ダイアログ・ボックスを閉じます。

注：GUI マップ・ファイルを Quality Center プロジェクトに保存するよう選択すると，GUI マップ・ファイルはプロジェクトにすぐにアップロードされます。

プロジェクトの GUI マップ・ファイルを開く

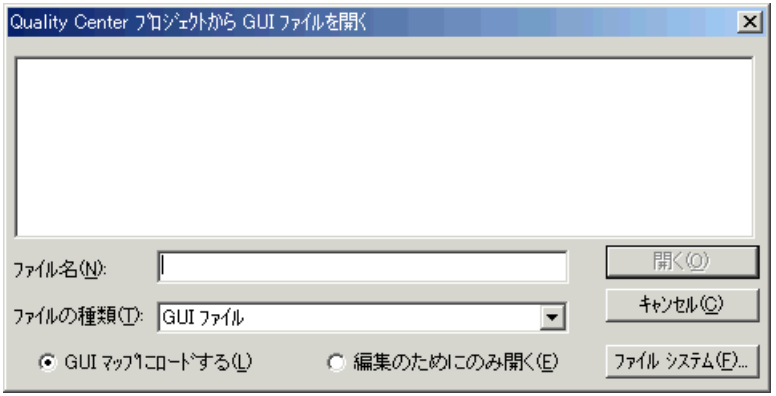
WinRunner が Quality Center プロジェクトに接続されている場合は，［**GUI マップ エディタ**］を使って，Quality Center プロジェクトに保存されている GUI マップ・ファイルを開けます。

Quality Center プロジェクトに保存されている GUI マップ・ファイルを開くには，次の手順を実行します。

1 ［**ツール**］＞［**GUI マップエディタ**］を選択し，［GUI マップ エディタ］を開きます。

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

［Quality Center **プロジェクトから GUI ファイルを開く**］ダイアログ・ボックスが開きます。開いているプロジェクトに保存されているすべての GUI マップ・ファイルがこのダイアログ・ボックスにリストされます。



［Quality Center プロジェクトから GUI ファイルを開く］ダイアログ・ボックスは，WinRunner が Quality Center プロジェクトに接続されている場合にのみ開きます。

GUI マップ・ファイルをファイル・システムから直接開くには，［**ファイルシステム**］ボタンをクリックして［**GUI ファイルを開く**］ダイアログ・ボックスを開きます（［Quality Center］ボタンをクリックすると，［**GUI ファイルを開く**］ダイアログ・ボックスから［Quality Center プロジェクトから GUI ファイルを開く］ダイアログ・ボックスに戻ることができます）。

3 開いているプロジェクト内の GUI マップ・ファイルのリストから GUI マップ・ファイルを選択します。GUI マップ・ファイルの名前が［**ファイル名**］テキスト・ボックスに表示されます。

4 開く GUI マップ・ファイルを GUI マップ・エディタに読み込むには，［**GUI マップにロードする**］をクリックします。これは標準の設定です。GUI マップ・ファイルだけを編集したい場合は，［**編集のためにのみ開く**］をクリックします。詳細については，第 7 章「GUI マップの編集」を参照してください。

5 ［**開く**］をクリックして，GUI マップ・ファイルを開きます。GUI マップ・ファイルは，GUI ファイル・リストに追加されます。「L」という字はファイルがロードされたことを示します。

テスト・セット内のテストの実行

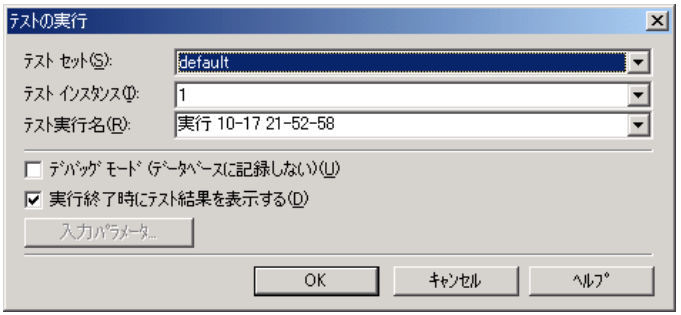
テスト・セットは特定のテスト目的を達成するために選択された一連のテストです。例えば，アプリケーションのユーザ・インタフェースや負荷のかかった状態でのアプリケーションのパフォーマンスをテストするテスト・セットを作成できます。テスト・セットは，Quality Center のテスト実行モードで作業しているときに定義します。

WinRunner がプロジェクトに接続されており，WinRunner からプロジェクト内のテストを実行したい場合は，テストを実行する前に現在のテスト・セットの名前を指定します。テスト実行が完了したら，指定したテスト・セットに応じてテストが Quality Center に格納されます。

テスト・セットとユーザ名を指定するには，次の手順を実行します。

1 ［**テスト**］メニューから［**実行**］コマンドを選択します。

［**テストの実行**］ダイアログ・ボックスが開きます。



2 ［**テスト セット**］ボックスで，リストからテスト・セットを選択します。リストには，Quality Center で作成されたテスト・セットが含まれます。

3 テスト・セットにテストのインスタンスが複数含まれている場合は，［**テスト インスタンス**］を選択します。TestDirector のバージョン 7.5 以前を使用している場合は，［テスト インスタンス］は常に 1 です。

4 ［**テスト実行名**］ボックスで，このテスト実行の名前を選択するか，新しい名前を入力します。

5 テストを「**デバッグ**」モードで実行するには，［**デバッグ モード**］チェック・ボックスを選択します。このオプションが選択されていると，このテスト実行の結果は Quality Center プロジェクトには書き込まれません。

6 テスト実行の最後に WinRunner でテスト結果を表示するには，［**実行終了時にテスト結果を表示する**］チェック・ボックスを選択します。

7 入力パラメータに値を提供するには，［**入力パラメータ**］をクリックして［入力パラメータ］ダイアログ・ボックスにこのテストに使用する値を入力します。詳細については，436 ページ「テスト実行時の入力パラメータの値の指定」を参照してください。

8 ［**OK**］をクリックして，パラメータを保存し，テストを実行します。

リモート・ホストでのテストの実行

複数のリモート・ホストで WinRunner テストを実行できます。他の Mercury 製品を 1 台のコンピュータをリモート・ホストとして使用できるようにするには，［**その他の Mercury 製品によるテストのリモート実行を許可する**］オプションをアクティブにしなければなりません。リモート・ホストでテストを実行すると，テストをテスト実行中に WinRunner メッセージが表示されないようにするサイレント・モードで実行しなければなりません。サイレント・モードの詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

リモート・マシンで他の Mercury 製品を使用して WinRunner のテストを実行できるようにするには，次の手順を実行します。

1 ［**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

2 ［**実行**］カテゴリをクリックします。

3 ［**その他の Mercury 製品によるテストのリモート実行を許可する**］チェック・ボックスを選択します。

注 :［**その他の Mercury 製品によるテストのリモート実行を許可する**］チェック・ボックスが選択されていないと，WinRunner テストはローカルでしか実行できません。

［一般オプション］ダイアログ・ボックスでテストオプションを設定する方法の詳細については，第 22 章「グローバル・テスト・オプションの設定」を参照してください。

テスト結果のプロジェクトからの表示

テスト・セット内のテストを実行すると，Quality Center プロジェクトからテスト結果を表示できます。テスト・セットを「**検証**」モードで実行すると，テスト実行の最後に自動的に［WinRunner テスト結果］ウィンドウが開きます。開かない場合は，［**ツール**］＞［**テスト結果**］を選択して，［テスト結果］ウィンドウを開きます。標準では，［テスト結果］ウィンドウには現在アクティブなテストの最新の実行結果が表示されます。別のテストのテスト結果，または現在アクティブなテストより前のテスト実行の結果を表示するには，［WinRunner テスト結果］ウィンドウで［**ファイル**］＞［**開く**］を選択します。

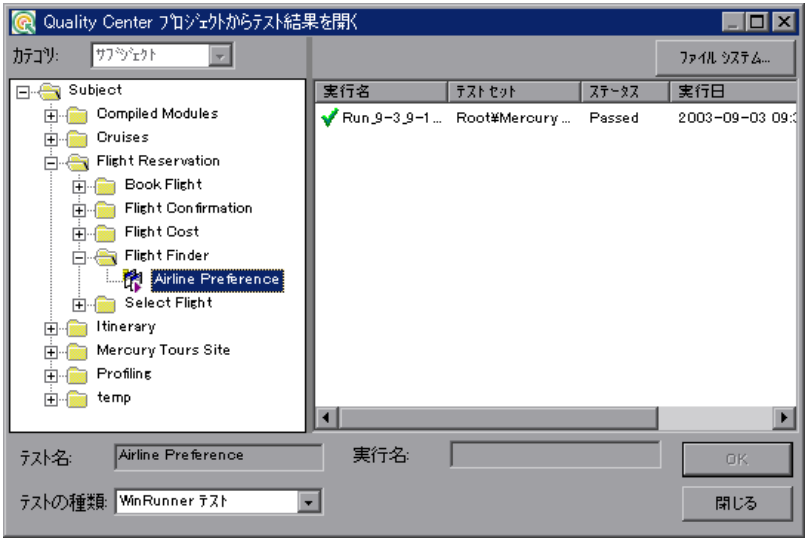
Quality Center プロジェクトからテスト結果を表示するには，次の手順を実行します。

1 ［**ツール**］＞［**テスト結果**］を選択します。

［テスト結果］ウィンドウが開き，アクティブなテストの最新のテスト実行のテスト結果が表示されます。

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

［Quality Center **プロジェクトからテスト結果を開く**］ダイアログ・ボックスが開き，テスト計画ツリーが表示されます。



［**Quality Center プロジェクトからテストを開く**］ダイアログ・ボックスは，WinRunner が Quality Center プロジェクトに接続されている場合のみ開きます。

ファイル・システムでテスト結果を直接開くには，［**ファイル システム**］ボタンをクリックして［テスト結果を開く］ダイアログ・ボックスを開きます（［Quality Center］ボタンをクリックすると，［テスト結果を開く］ダイアログ・ボックスから［Quality Center プロジェクトからテストを開く］ダイアログ・ボックスに戻ります）。

3 ［**テストの種類**］ボックスで，ダイアログ・ボックスに表示するテストの種類を，すべてのテスト（標準設定），WinRunner テスト，WinRunner バッチ・テストから選択します。

4 テスト計画ツリーで該当するサブジェクトを選択します。ツリーを広げて下位のレベルを表示するには，閉じているフォルダをダブルクリックします。ツリーを閉じるには，開いているフォルダをダブルクリックします。

5 表示するテスト実行を選択します。右の枠には以下が表示されます。

- ［**テスト名**］カラムには，テストが成功か失敗かが表示され，テスト実行の名前が含まれます。
- ［**テストセット**］カラムには，テスト・セット名が含まれます。
- ［**ステータス**］カラムのエントリは，テストが成功したか失敗したかを示します。
- ［**実行日**］カラムには，テスト・セットが実行された日付と時刻が表示されます。

6 ［**OK**］をクリックして，選択されたテストの結果を表示します。

テスト結果で，アプリケーションに不具合があることが分かった場合は，［テスト結果］ウィンドウから直接 Quality Center の不具合データベースに不具合を報告できます。詳細については，43 ÉyÅ[ÉWÅuÉeÉXÉgé¿çsíÜÇ…åüèoÇŠÇÍÇ¾ïsãÔçáÇÃïÒçêÅv を参照してください。

［WinRunner テスト結果］ウィンドウのオプションについては，ëÊ 2 èÕÅuiùàÍÉåÉ|Å[ÉgÅEÉrÉÖÅ[ÇÝÇÃÉeÉXÉgåââþÇÃï™ÊÕÅv を参照してください。

TSL 関数の Quality Center での使用

TSL 関数には，Quality Center プロジェクトのフィールドの値を返すことによって，Quality Center プロジェクトを使用した作業を簡易化するものもあります。さらに，Quality Center を使用して作業すると，TSL 関数を使用した作業が簡単になります。WinRunner が Quality Center に接続されている場合は，ファイル・システムの完全パスを使用しなくても，TSL ステートメントで Quality Center プロジェクト内のパスを指定できます。

Quality Center プロジェクト関数

TSL 関数には，Quality Center プロジェクトから情報を取得できるものもあります。

qcdb_add_defect	WinRunner が接続されているプロジェクトの Quality Center 不具合データベースに新規不具合を追加します。
qcdb_get_step_value	Quality Center プロジェクトの「test」テーブル内のフィールドの値を返します。
qcdb_get_testset_value	Quality Center プロジェクトの「testcycle」テーブル内のフィールドの値を返します。
qcdb_load_attachment	テストのファイルの添付をローカル・キャッシュにダウンロードして，その場所を返します。

これらの関数は，関数ジェネレータを使用してテスト・スクリプトに挿入できます。また，これらの関数を使用するステートメントを手作業でプログラムすることもできます。

これらの関数の詳細については，「**TSL リファレンス**」を参照してください。

call ステートメントとコンパイル済みモジュール関数

WinRunner が Quality Center に接続されている場合，**call**，**call_close**，**load**，**reload**，および **unload** 関数を使用するときに，Quality Center プロジェクトに保存されているテストのパスとコンパイル済みモジュール関数を指定できます。

例えば，Quality Center プロジェクトに Subject¥Sub1¥My_test というパスのテストがある場合，これを次のステートメントでテスト・スクリプトから呼び出すことができます。

```
call "[QC]¥¥Subject¥¥Sub1¥¥My_test"();
```

あるいは，［一般オプション］ダイアログ・ボックスで “[QC]¥Subject¥Sub1” という検索パスを指定している場合は，次のステートメントでテストをテスト・スクリプトから呼び出すことができます。

```
call "My_test" ();
```

[QC] という接頭辞は Quality Center プロジェクトのテストまたはコンパイル済みモジュールを指定する場合のオプションです。

注：Quality Center プロジェクトから WinRunner を実行する際は，テストから呼び出し先のテストにパラメータを渡すのに **call** ステートメントを使わなくても，Quality Center 内で「In」パラメータを指定できます。Quality Center を使用して，“Out” パラメータを含む WinRunner テストを呼び出すことはできません。Quality Center から WinRunner テストのパラメータを指定する詳細については，『**Quality Center ユーザーズ・ガイド**』を参照してください。「In」および「Out」パラメータの詳細については，693 ページ「テストの呼び出しについて」を参照してください。

特定の call ステートメントとコンパイル済みモジュール関数を使用する方法については，「**TSL リファレンス**」**を参照してください。**

GUI マップ・エディタ関数

WinRunner が Quality Center に接続されている場合は，テスト・スクリプト内の GUI マップ・エディタ関数を使用する際に Quality Center プロジェクトに保存されている GUI マップ・ファイルの名前を指定できます。

WinRunner が Quality Center プロジェクトに接続されている場合，WinRunner はデータベース内の GUI レポジトリに GUI マップ・ファイルを格納します。[QC] という接頭辞は，Quality Center プロジェクト内の GUI マップ・ファイルを指定する場合は省略可能です。

例えば，My_gui.gui という GUI マップ・ファイルが Quality Center プロジェクト（My_project_database¥GUI）内に格納されている 場合，次のステートメントでこの GUI マップ・ファイルをロードできます。

```
GUI_load ("My_gui.gui");
```

GUI マップ・エディタ関数の使用法の詳細については，「**TSL リファレンス**」を参照してください。

Quality Center から呼び出されたテストへの検索パスの指定

Quality Center プロジェクト内のパスに基づいて検索パスを使用するよう WinRunner を設定できます。

次の例で，**setvar** ステートメントは Quality Center プロジェクト内の検索パスを指定します。

```
setvar ( "searchpath", "[QC]¥¥My_project_database¥¥Subject¥¥Sub1" );
```

［一般オプション］ダイアログ・ボックスを使って検索パスを指定する方法の詳細については，第 22 章「グローバル・テスト・オプションの設定」を参照してください。**setvar** ステートメントを使って検索パスを指定する方法の詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

Quality Center で使用するコマンドライン・オプション

Windows の［ファイル名を指定して実行］コマンドを使用して，Quality Center で使用するパラメータを設定できます。ユーザ定義の WinRunner ショートカットを作成してスタートアップ・パラメータを保存することもできます。次に，スタートアップ・パラメータを使って WinRunner を起動する場合は，アイコンをダブルクリックするだけです。

次のコマンドライン・オプションを使用して，Quality Center で使用するパラメータを設定できます。コマンドライン・オプションの使用法については，第 37 章「コマンドラインからのテストの実行」を参照してください。

-dont_connect

［Quality Center に接続］ダイアログ・ボックスで［**起動時に再接続する**］チェックボックスが選択されている場合，このコマンドライン・オプションを使うと Quality Center に接続せずに WinRunner を開くことができます。

-qc_connection {on | off}

on の場合に，WinRunner の Quality Center への接続を有効にします。

（標準設定 = **off**）

（以前の **-td_connection** または **-test_director**）

注：［Quality Center に接続］ダイアログ・ボックスで［起動時に再接続する］オプションを選択している場合は，**qc_connection** を off に設定しても Quality Center への接続は妨げられません。このような場合に Quality Center に接続されないようにするには，**-dont_connect** コマンドを使用します。詳細については，990 ページ「-dont_connect」を参照してください。

-qc_cycle_name *cycle_name*

現在のテスト・サイクルの名前を指定します。このオプションは，WinRunner が Quality Center に接続されている場合にのみ使用できます。

現在のテスト・サイクルの名前は，**qc_cycle_name** テスト・オプションを使って指定することもできます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

-qc_database_name *database_pathname*

アクティブな Quality Center プロジェクトを指定します。WinRunner はこのプロジェクトでテストを開いたり，実行したり，保存したりできます。このオプションは，WinRunner が Quality Center に接続されている場合にのみ使用できます。

アクティブな Quality Center データベースを指定するには，**qc_database_name** テスト・オプションを使用することもできます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

WinRunner が Quality Center に接続されている場合，［Quality Center に接続］ダイアログ・ボックスでアクティブな TestDirector プロジェクトを指定できます。このダイアログ・ボックスは［**ツール**］＞［**Quality Center への接続**］を選んで開きます。Quality Center への接続の詳細については，964 ページ「プロジェクトの接続と接続の解除」を参照してください。

-qc_password

Quality Centerサーバのプロジェクトに接続するためのパスワードを指定します。

Quality Center に接続するためのパスワードを［Quality Center に接続］ダイアログ・ボックスで指定できます。このダイアログ・ボックスは［**ツール**］＞［**Quality Center への接続**］を選んで開きます。

Quality Center への接続の詳細については，964 ページ「プロジェクトの接続と接続の解除」を参照してください。

-qc_server_name

WinRunner が接続する Quality Center サーバの名前を指定します。

WinRunner が接続する Quality Center サーバの名前を指定するには，**td_server_name** テスト・オプションを使用することもできます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

WinRunner が接続する Quality Center サーバの名前は［Quality Center に接続］ダイアログ・ボックスで指定できます。このダイアログ・ボックスは［**ツール**］＞［**Quality Center への接続**］を選んで開きます。Quality Center への接続の詳細については，964 ページ「プロジェクトの接続と接続の解除」を参照してください。

-qc_user_name *user_name*

現在テスト・サイクルを実行しているユーザの名前を指定します（以前は *user* でした）。

ユーザを指定するには，**qc_user_name** テスト・オプションを使用することもできます。詳細については，第 43 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。

WinRunner が Quality Center と接続している場合，［Quality Center に接続］ダイアログ・ボックスでユーザの名前を指定できます。このダイアログ・ボックスは［**ツール**］＞［**Quality Center への接続**］を選んで開きます。Quality Center への接続の詳細については，964 ページ「プロジェクトの接続と接続の解除」を参照してください。

コマンドライン・オプションの使用法の詳細については，第 37 章「コマンドラインからのテストの実行」を参照してください。

第 49 章

負荷下でのシステムのテスト

今日のアプリケーションは，複雑なアーキテクチャ上で複数のユーザによって実行されます。Mercury の自動化されたパフォーマンス・テスト・ソリューションである LoadRunner では，システム全体のパフォーマンスと信頼性をテストできます。

本章では，以下の項目について説明します。

- 負荷のかかった状態でのシステムのテストについて
- 複数のユーザのエミュレート
- 仮想ユーザ技術
- シナリオの開発と実行
- GUI 仮想ユーザ・スクリプトの作成
- サーバのパフォーマンスの測定
- 仮想ユーザのトランザクションの同期化
- ランデブー・ポイントの作成
- 仮想ユーザ・スクリプトのサンプル

負荷のかかった状態でのシステムのテストについて

ソフトウェアのテストは，もはや単独のスタンドアロン PC で動作するアプリケーションのテストに限りません。アプリケーションは，複数の PC が中央のサーバとやり取りをするネットワーク環境で実行されます。Web ベースのアプリケーションも同様です。

今日のアーキテクチャは複雑です。このアーキテクチャにより，これまでにない処理能力と柔軟性が得られますが，これらのシステムをテストするのは困難です。LoadRunner は，負荷をエミュレートし，パフォーマンスと機能を正確に測定，分析します。本章では，WinRunner を LoadRunner と併用して，システムをテストする方法の概要を解説します。アプリケーションの負荷テストの方法の詳細については，LoadRunner のマニュアルを参照してください。

複数のユーザのエミュレート

LoadRunner では，「**シナリオ**」を作成して，複数のユーザとのやり取りをエミュレートします。シナリオは，各負荷テストのセッションにおいて，ユーザ数，ユーザが実行するアクション，ユーザが使用するマシンなどのイベントを定義します。シナリオの詳細については，『**LoadRunner コントローラ・ユーザーズ・ガイド**』を参照してください。

LoadRunner は，シナリオにおいて，人間の代わりに「**仮想ユーザ**」（**Vuser**）を使います。仮想ユーザは，アプリケーションを使用する人間のユーザによる操作をエミュレートします。シナリオには，何十，何百，あるいは何千もの仮想ユーザを含むことができます。

仮想ユーザ技術

LoadRunner はさまざまな仮想ユーザ技術を提供します。これらの仮想ユーザ技術を使用して，異なった種類のシステムのアーキテクチャを使用する際の負荷を生成できます。それぞれの仮想ユーザ技術は特定のアーキテクチャに適しており，特定のタイプの仮想ユーザを結果として生成します。例えば，GUI 仮想ユーザを使用して，Microsoft Windows などの環境でグラフィカル・ユーザ・インタフェース・アプリケーションを操作したり，Web 仮想ユーザを使用して，Web ブラウザでのユーザの操作をエミュレートしたり，RTF 仮想ユーザを使用して，端末エミュレータを操作したり，データベース仮想ユーザを使用して，データベース・アプリケーション・サーバと通信するデータベース・クライアントをエミュレートしたりできます。

様々な仮想ユーザ技術を，単独でまたは組み合わせて使用し，効果的な負荷テストのシナリオを作成することができます。

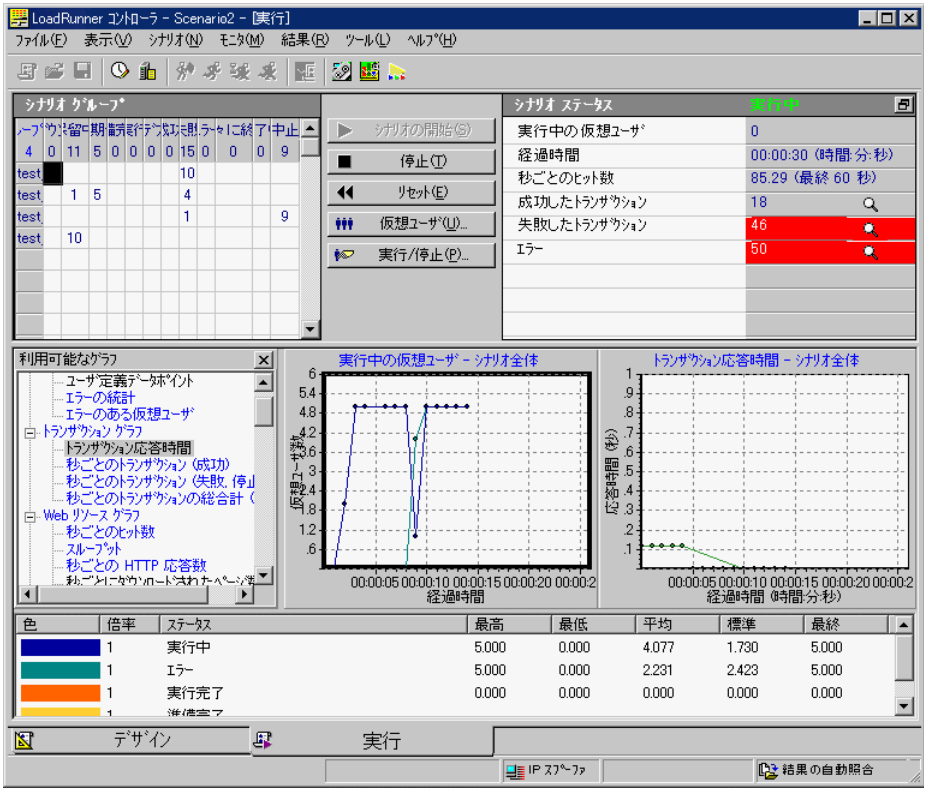
GUI 仮想ユーザ

Microsoft Windows などの環境で，グラフィカル・ユーザ・インタフェース・アプリケーションを操作します。各 GUI 仮想ユーザは，クライアント・アプリケーションに対する入力を行ったり，クライアント・アプリケーションに対する出力を受け取ったりする実際のユーザをエミュレートします。

GUI 仮想ユーザは，1 つの WinRunner とクライアント・アプリケーションから成ります。クライアント・アプリケーションは，データベース・クライアントなど，サーバにアクセスする任意のアプリケーションを対象にできます。WinRunner は人間のユーザの代わりにクライアント・アプリケーションを操作します。各 GUI 仮想ユーザは仮想ユーザ・スクリプトを実行します。「仮想ユーザ・スクリプト」は，WinRunner がシナリオの中で行うべき操作を記述したテストです。サーバのパフォーマンスを測定，記録するステートメントも含まれています。詳細については，『**仮想ユーザ・スクリプトの作成**』を参照してください。

シナリオの開発と実行

LoadRunner コントローラを使って，シナリオを開発，実行します。コントローラは，ネットワークに接続されている任意の PC で動作するアプリケーションです。



次に，LoadRunner コントローラを使って，シナリオの作成，実行および分析を行う大まかな手順を紹介します。詳細については，『**LoadRunner コントローラ・ユーザーズ・ガイド**』を参照してください。

1 **コントローラを起動します。**

2 **シナリオを作成します。**

シナリオは，各負荷テストのセッションで生じる，参加する仮想ユーザ，仮想ユーザが実行するスクリプト，スクリプトを実行するマシン（負荷生成マシン）などのイベントを記述します。

3 **シナリオを実行します。**

シナリオを実行すると，LoadRunner は指定された負荷生成マシンに仮想ユーザを分散します。負荷生成マシンは準備ができると，スクリプトの実行を開始します。シナリオ実行の間，LoadRunner はサーバのパフォーマンスに関するデータの測定，記録を行います。また，LoadRunner はオンライン・ネットワークを提供し，サーバの監視も行います。

4 **サーバのパフォーマンスを分析します。**

シナリオを実行した後，LoadRunner のグラフやレポート機能を使って，シナリオ実行の間にキャプチャしたサーバのパフォーマンス・データを分析できます。

以降では，GUI 仮想ユーザ・スクリプトを作成する方法を解説します。仮想ユーザ・スクリプトは，PC クライアントで動作するアプリケーションを通してサーバにアクセスする人間のユーザが実行したアクションを記述します。

GUI 仮想ユーザ・スクリプトの作成

GUI 仮想ユーザ・スクリプトは，LoadRunner シナリオにおける GUI 仮想ユーザのアクションを記述します。GUI 仮想ユーザ・スクリプトは，WinRunner を使って作成します。以下は，基本的なスクリプトを作成するための大まかな手順です。詳細については，『**仮想ユーザ・スクリプトの作成**』を参照してください。

GUI 仮想ユーザ・スクリプトを作成するには，次の手順を実行します。

1 WinRunner を開きます。

2 クライアント・アプリケーションを起動します。

3 クライアント・アプリケーションに対する操作を記録します。

4 WinRunner を使って仮想ユーザ・スクリプトを編集し，追加の TSL ステートメントなどをプログラミングします。必要に応じてフロー制御構造などを追加します。

5 サーバのパフォーマンスを測定するために，スクリプトに記述したアクションをトランザクションとして編集します。

6 スクリプトに同期化ポイントを追加します。

7 「**ランデブー**」ポイントを追加して，複数の仮想ユーザによるアクションを調整します。

8 スクリプトを保存し，WinRunner を終了します。

サーバのパフォーマンスの測定

トランザクションを使って，複数のユーザの負荷下でサーバのパフォーマンスを測定します。トランザクションは，テキスト・フィールドにテキストを入力する単純な作業から，複数の作業を行う完全なテストまで定義できます。LoadRunner は，さまざまな負荷のもとで，トランザクションのパフォーマンスを測定します。単独のユーザと数百のユーザが同じトランザクションを行った場合の時間を測定できます。

トランザクションの作成は，まず仮想ユーザ・スクリプトの先頭でトランザクションの名前を宣言するところから始まります。仮想ユーザ・スクリプトを仮想ユーザに割り当てると，コントローラは仮想ユーザ・スクリプトをスキャンして，トランザクション宣言ステートメントを探します。スクリプトにトランザクション宣言が含まれている場合，LoadRunner は，トランザクションの名前を読み取って，それを［トランザクション］ウィンドウに表示します。

トランザクションを宣言するには，**declare_transaction** 関数を使います。この関数の構文は，次のとおりです。

declare_transaction ("*transaction_name***");**

transaction_name は，変数や式ではなく，文字列定数でなければなりません。この文字列は，128 文字まで含むことができます。空白文字は使用してはなりません。

次に，LoadRunner がトランザクションの測定を開始する場所を示します。仮想ユーザ・スクリプトの中で，測定するアクションの直前に，**start_transaction** ステートメントを挿入します。

この関数の構文は，次のとおりです。

start_transaction ("*transaction_name***");**

transaction_name は，**declare_transaction** ステートメントで定義した名前です。

スクリプトに，トランザクションの終了を示す **end_transaction** ステートメントを挿入します。テスト全体が単独のトランザクションの場合，このステートメントをスクリプトの最終行に挿入します。この関数の構文は，次のとおりです。

end_transaction ("*transaction_name*" **[**, *status* **])**;

transaction_name は，**declare_transaction** ステートメントで定義した名前です。*status* は，LoadRunner に対して，トランザクションが成功（PASS）または失敗（FAIL）した場合のみトランザクションを終了するよう通知します。

仮想ユーザのトランザクションの同期化

トランザクションを使ってサーバのパフォーマンスを正確に測定するには，サーバがユーザのリクエストに応答するのに要する時間を考慮しなければなりません。人間のユーザならば，メッセージなどの視覚的なきっかけを認識して，サーバが処理を完了したことを知ることができます。例えば，データベース・サーバがユーザの照会に応答するのに要する時間を測定したいとします。人間のユーザなら，照会に対する応答が画面に表示されれば，サーバがデータベース照会の処理を完了したことが分かります。仮想ユーザ・テストでは，同期化ポイントを挿入して，仮想ユーザが特定のきっかけを待つよう指示します。

同期化ポイントは，オブジェクトにメッセージが表示されるなど，特定のイベントが生じるまで仮想ユーザを待機させます。イベントが生じたら，仮想ユーザはスクリプトの実行を再開します。オブジェクトが現れない場合，仮想ユーザはオブジェクトが現れるか，時間制限が過ぎるまで待機し続けます。トランザクションは，WinRunner の任意の同期化関数あるいはオブジェクト関数を使ってどう同期化できます。WinRunner の同期化関数の詳細については，第 18 章「テスト実行の同期化」を参照してください。

ランデブー・ポイントの作成

シナリオ実行の際に，複数のユーザが同時に作業を行うようにするには，ランデブー・ポイントを作成します。これにより，以下のことを実現できます。

- 高負荷をエミュレートする
- トランザクションが複数の仮想ユーザの負荷のもとで測定されるようにする

ランデブー・ポイントとは，仮想ユーザの待ち合わせ場所です。待ち合わせ場所を指定するには，仮想ユーザ・スクリプトにランデブー・ステートメントを挿入します。ランデブー・ステートメントが解釈されると，コントローラは，ランデブー・ポイントの全メンバが到着するまで，仮想ユーザを待機させます。仮想ユーザが全員到着すると（またはタイムリミットが経過すると），全員が一度に解放され，それぞれが仮想ユーザ・スクリプトの次のタスクを実行します。

ランデブー・ポイントの作成は，まず仮想ユーザ・スクリプトの先頭でランデブー・ポイントの名前を宣言します。仮想ユーザ・スクリプトを仮想ユーザに割り当てると，LoadRunner はスクリプトをスキャンして，ランデブー宣言ステートメントを探します。スクリプトにランデブー・ポイント宣言が含まれている場合，LoadRunner はランデブーの名前を読み取って，ランデブーを作成します。同じスクリプトを実行する別の仮想ユーザを作成すると，コントローラはその仮想ユーザをランデブーに追加します。

ランデブーを宣言するには，**decalre_rendezvous** 関数を使います。この関数の構文は，次のとおりです。

declare_rendezvous ("rendezvous_name");

rendezvous_name は，ランデブーの名前です。*rendezvous_name* は，変数や式ではなく，文字列定数でなければなりません。この文字列は，128 文字まで含むことができます。空白文字は使用してはなりません

次に，仮想ユーザ・テストのどこでランデブーが生じるかを示すため，**rendezvous** ステートメントを挿入します。ランデブー・ステートメントは，LoadRunner に対して，他の仮想ユーザがそろうまで，当該仮想ユーザをランデブー地点で待機させるよう指示します。この関数の構文は，次のとおりです。

rendezvous ("rendezvous_name");

rendezvous_name は，ランデブーの名前です。

仮想ユーザ・スクリプトのサンプル

以下に示す仮想ユーザ・テストのサンプルでは，「Ready」というトランザクションで，ユーザのリクエストに対してサーバが応答するのに要する時間を測定します。ユーザはリクエストを入力して［OK］をクリックします。ユーザは，クライアント・アプリケーションの［Status］フィールドに「Ready」という文字列が現れたら，リクエストの処理が完了したと判断します。

仮想ユーザ・スクリプトの最初の部分では，**declare_transaction** および **declare_rendezvous** 関数を使って，トランザクションとランデブー・ポイントの名前を宣言しています。このスクリプトでは，「Ready」というトランザクションと，「wait」というランデブーが宣言されています。この宣言ステートメントにより，LoadRunner コントローラはトランザクションとランデブーに関する情報を表示できます。

```
# トランザクション名を宣言する。
declare_transaction ("Ready");

# ランデブー名を定義する。
declare_rendezvous ("wait");
```

次に，サーバに高負荷を与えるために，**rendezvous** ステートメントを使って，全仮想ユーザが同時に［OK］ボタンをクリックするようにします。

```
# ランデブー・ポイントを定義する。
rendezvous ("wait");
```

続く部分では，仮想ユーザが［OK］ボタンをクリックする直前に，**start_transaction** ステートメントを挿入しています。これにより，LoadRunner は「Ready」トランザクションの記録を開始します。「Ready」トランザクションは，仮想ユーザが送ったリクエストをサーバが処理するのに要する時間を測定します。

```
# トランザクションの記録開始
start_transaction ( "Ready" );
button_press ( "OK" );
```

LoadRunner が，トランザクションに要した時間を記録するためには，サーバによるリクエストの処理が完了したことを示すきっかけを待たなければなりません。人間のユーザなら，［Status］フィールドに「Ready」と表示されれば，リクエストの処理が完了したことを知ることができます。仮想ユーザ・スクリプトでは，**obj_wait_info** ステートメントを使って，メッセージが表示されるまで待機します。

タイムアウト時間を 30 秒に設定することで，メッセージが現れるまで仮想ユーザに 30 秒間待機させてから，テストの実行を継続させることができます。

```
# メッセージが現れるのを待機する。
rc = obj_wait_info("Status","value","Ready.",30);
```

テストの最後の部分では，トランザクションに要した時間を記録します。if ステートメントを使って，**obj_wait_info** 関数の結果を処理しています。

タイムアウト時間内にメッセージがフィールドに表示された場合，最初の **end_transaction** ステートメントによって，トランザクションに要した時間と，トランザクションが成功したことが記録されます。メッセージがタイムアウト時間内に現れなかった場合，トランザクションは失敗します。

```
# トランザクションの終了
if (rc == 0)
    end_transaction ( "OK", PASS );
else
    end_transaction ( "OK" , FAIL );
```

索引

E

R

S

U

V

W

X

Y

あ

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

ひ

ふ