基本機能ユーザーズ・ガイド

MERCURY™

Mercury WinRunner

基本機能ユーザーズ・ガイド

Version 8.2

MERCURY™

Mercury WinRunner 基本機能ユーザーズ・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

総合目次

WinRunner のユーザーズ・ガイドは 2 巻で構成されています。

- 『**Mercury WinRunner 基本機能ユーザーズ・ガイド**』は，WinRunner について説明し，機能と自動テストの手順について解説します。
- 『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』は，WinRunner の上級機能について説明し，Mercury のテスト・スクリプト言語（TSL）について紹介し，上級者向けの設定オプションについて説明します。他の Mercury 製品と組み合わせて使用する方法についても説明します。

各ガイドに含まれる章のサマリは下記をご覧ください。

Mercury WinRunner 基本機能ユーザーズ・ガイド

第 1 部 : テスト・プロセスの開始

第 1 章 : はじめに 3

第 2 章 : WinRunner の概要 11

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 27

第 4 章 : GUI マップの基本概念について 35

第 5 章 : グローバル GUI マップ・ファイル・モードでの作業 47

第 6 章 : テスト特有の GUI マップ・ファイル・モードでの作業 67

第 7 章 : GUI マップの編集 73

第 3 部 : テストの作成 ー 基本

第 8 章 : テストの設計 95

第 9 章 : GUI オブジェクトの検査 129
第 10 章 : Web オブジェクトでの作業 181
第 11 章 : ActiveX と Visual Basic のコントロールの使用 223
第 12 章 : PowerBuilder のアプリケーションの検査 243
第 13 章 : テーブル内容の検査 253
第 14 章 : データベースの検査 267
第 15 章 : ビットマップの検査 331
第 16 章 : テキストの検査 341
第 17 章 : データ駆動型テストの作成 359
第 18 章 : テスト実行の同期化 407

第 4 部 : テストの実行 ー 基本
第 19 章 : テスト実行について 423
第 20 章 : テスト結果の分析 443

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 499
第 22 章 : グローバル・テスト・オプションの設定 519

Mercury WinRunner 上級機能ユーザーズ・ガイド

第 1 部 : GUI マップを使った作業
第 1 章 : GUI マップ・ファイルのマージ 3
第 2 章 : GUI マップの構成設定 15
第 3 章 : 仮想オブジェクトの学習 37

第 2 部 : テストの作成 ー 上級
第 4 章 : 回復シナリオの定義と使用 47
第 5 章 : Web 例外処理の定義 89
第 6 章 : 正規表現の使い方 95

第 3 部 : TSL を使ったプログラミング

第 7 章 : プログラミングによるテスト・スクリプトの 機能強化 105
第 8 章 : 関数の生成 125
第 9 章 : テストの呼び出し 135
第 10 章 : ユーザ定義関数の作成 153
第 11 章 : テストでのユーザ定義関数の利用 163
第 12 章 : 外部ライブラリからの関数の呼び出し 185
第 13 章 : 対話形式の入力用ダイアログ・ボックスの作成 193

第 4 部 : テストの実行 ー 上級

第 14 章 : バッチ・テストの実行 203
第 15 章 : コマンドラインからのテストの実行 211

第 5 部 : テストのデバッグ

第 16 章 : テスト実行の制御 233
第 17 章 : ブレークポイントの使用 239
第 18 章 : 変数の監視 249

第 6 部 : 上級者向けの設定

第 19 章 : テスト・スクリプト・エディタのカスタマイズ 259
第 20 章 : WinRunner のユーザ・インタフェースの カスタマイズ .. 269
第 21 章 : テスト・スクリプトからのテスト・オプションの設定 293
第 22 章 : 関数ジェネレータのカスタマイズ 319
第 23 章 : 特殊な構成の初期化 335

第 7 部 : その他の MERCURY 製品を使った作業

第 24 章 : Business Process Testing での作業 341
第 25 章 : QuickTest Professional との統合 379
第 26 章 : テスト工程の管理 385
第 27 章 : 負荷下でのシステムのテスト 419

目次

総合目次 iii
Mercury WinRunner 基本機能ユーザーズ・ガイド iii
Mercury WinRunner 上級機能ユーザーズ・ガイド iv

Mercury WinRunner へようこそ xv
本書の使い方 xv
WinRunner の関連マニュアル xvi
オンライン・リソース xvii
表記規則 xix

第 1 部：テスト・プロセスの開始

第 1 章：はじめに 3
WinRunner テストのモード 4
WinRunner のテスト・プロセス 5
サンプル・アプリケーション 8
他の Mercury 製品との統合 9

第 2 章：WinRunner の概要 11
WinRunner の起動方法 11
WinRunner のメイン・ウィンドウ 14
テスト・エディタ・ウィンドウ 16
WinRunner コマンドの使用方法 17
WinRunner アドインのロード 21

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 27
GUI オブジェクトの識別について 27
テストが GUI オブジェクトを識別する方法 29
物理的記述 29
論理名 31
GUI マップ 32
ウィンドウのコンテキストの設定 33

第 4 章： GUI マップの基本概念について 35
GUI マップの概要 35
GUI オブジェクトのプロパティの表示 36
WinRunner によるアプリケーションの GUI の学習 42
GUI マップでのオブジェクトまたはウィンドウの検索 43
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 43
使用する GUI マップ・ファイル・モードの決定 44

第 5 章： グローバル GUI マップ・ファイル・モードでの作業 47
［グローバルな GUI マップ ファイル］モードについて 47
テスト間での GUI マップ・ファイルの共有 49
アプリケーションの GUI の学習 50
GUI マップの保存 59
GUI マップ・ファイルのロード 61
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 65

第 6 章： テスト特有の GUI マップ・ファイル・モードでの作業 67
［テスト特有の GUI マップ ファイル］モードについて 67
［テスト特有の GUI マップ ファイル］モードの指定 68
［テスト特有の GUI マップ ファイル］モードでの作業 70
［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 71

第 7 章： GUI マップの編集 73
GUI マップの編集について 74
GUI マップ・エディタ 75
実行ウィザード 78
論理名と物理的記述の修正 79
WinRunner のウィンドウ・ラベル変更の対処方法 82
物理的記述の正規表現の使用 84
ファイル間でのオブジェクトのコピーと移動 85
GUI マップ・ファイルでオブジェクトを検索する方法 87
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 88
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 89
GUI マップ・ファイルからのオブジェクトの削除 90
GUI マップ・ファイルの全内容の削除 90
表示されるオブジェクトのフィルタ処理 91
GUI マップへの変更の保存 92

第 3 部 : テストの作成 ー 基本

第 8 章 : テストの設計.....95
テストの作成について.....96
WinRunner のテスト・ウィンドウについて 97
テストの計画.....98
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 99
アナログ記録モードを使ったテストの作成 104
テストの記録のガイドライン.....106
テストへのチェックポイントの追加 108
データ駆動型テストを使った作業.....108
テストへの同期化ポイントの追加.....109
トランザクションの測定 109
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 112
テストのプログラミング 115
テストの編集.....115
テスト・ファイルの管理 116

第 9 章 : GUI オブジェクトの検査.....129
GUI オブジェクトの検査について 130
単数のプロパティ値の検査 132
単数のオブジェクトの検査 134
ウィンドウ内の複数のオブジェクトの検査 137
ウィンドウ内のすべてのオブジェクトの検査.....139
GUI チェックポイント・ステートメントについて 141
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 143
GUI チェックリストの変更 145
GUI チェックポイント・ダイアログ・ボックスについて.....150
プロパティ検査と標準の検査.....163
プロパティ検査への引数の指定 169
プロパティの期待値の編集 175
GUI チェックポイントの期待結果の変更.....177

第 10 章 :Web オブジェクトでの作業.....181
Web オブジェクトの作業について 181
記録済み Web オブジェクトのプロパティ表示.....182
テストでの Web オブジェクト・プロパティの使用 183
Web オブジェクトの検査について 193
テキストの検査 218

第 11 章 :ActiveX と Visual Basic のコントロールの使用 223
ActiveX と Visual Basic のコントロールの使い方について.....223
Visual Basic アプリケーションに対する適切なサポートの選択.....228
ActiveX と Visual Basic コントロールのプロパティの表示.....229
ActiveX と Visual Basic のコントロールのプロパティ値の取得と設定 232
ActiveX コントロール・メソッドのアクティブ化.....235
Visual Basic のラベル・コントロールの使用.....236
ActiveX と Visual Basic のコントロールのサブオブジェクトの検査 238
ActiveX コントロールでの TSL テーブル関数の使用 241

第 12 章 :PowerBuilder のアプリケーションの検査.....243
PowerBuilder のアプリケーションの検査について 243
ドロップダウン・オブジェクトのプロパティ検査.....244
DataWindow のプロパティの検査 247
DataWindow 内のオブジェクトのプロパティの検査 249
DataWindow 内の計算カラムの処理 251

第 13 章 :テーブル内容の検査 253
テーブル内容の検査について 253
標準の検査によるテーブル内容の検査.....255
検査を指定してのテーブル内容の検査.....256
［チェックの編集］ダイアログ・ボックスについて.....259

第 14 章 :データベースの検査 267
データベースの検査について 268
データベース実行時レコード・チェックポイントの作成.....272
実行時データベース・レコード・チェックリストの編集.....281
データベースを対象とする標準の検査の作成.....286
データベースを対象とするユーザ定義の検査の作成.....289
［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.....293
［データベース チェックポイント］ウィザードでの作業.....293
［チェックの編集］ダイアログ・ボックスについて.....300
標準のデータベース・チェックポイントの変更 306
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 317
標準のデータベース・チェックポイントのパラメータ化.....319
データベースの指定 323
TSL 関数を使用してのデータベース作業.....325

第 15 章 : ビットマップの検査 331
ビットマップの検査について 331
ビットマップ・チェックポイントの作成 333
ウィンドウとオブジェクトのビットマップの検査 336
領域ビットマップのチェック 338

第 16 章 : テキストの検査 341
テキストの検査について 341
テキストの読み取り 344
テキストの検索 348
テキストの比較 352
WinRunner によるフォントの学習 352

第 17 章 : データ駆動型テストの作成 359
データ駆動型テストの作成について 360
データ駆動型テストの工程 360
変換用基本テストの作成 361
テストのデータ駆動型テストへの変換 363
データ・テーブルの準備 376
データベースからのデータのインポート 385
データ駆動型テストの実行と分析 390
テストへのメイン・データ・テーブルの割り当て 391
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 393
データ駆動型テストでの TSL 関数の使用 398
データ駆動型テスト作成のガイドライン 404

第 18 章 : テスト実行の同期化 407
テスト実行の同期化について 407
オブジェクトとウィンドウの待機 409
オブジェクトとウィンドウのプロパティ値の待機 410
オブジェクトやウィンドウのビットマップの待機 415
スクリーン領域のビットマップの待機 417
テストの同期化のためのヒント 419

第 4 部 : テストの実行 ー 基本

第 19 章 :テスト実行について 423
テスト実行について 423
WinRunner のテスト実行モード 425
WinRunner の実行コマンド.....429
ソフトキーを使った実行コマンドの選択.....431
アプリケーションを検査するためのテスト実行 432
テスト・スクリプトをデバッグするためのテスト実行 434
期待結果を更新するためのテスト実行.....435
テスト実行時の入力パラメータの値の指定 438
テスト・オプションによるテスト実行の制御.....439
テスト実行に関する一般的な問題の解決法 440

第 20 章 :テスト結果の分析 443
テスト結果の分析について.....444
統一レポート・ビューの結果ウィンドウについて.....445
テスト結果の表示のカスタマイズ.....456
WinRunner レポート・ビューの結果ウィンドウについて.....457
テスト実行の結果の表示 464
チェックポイントの結果の表示 471
シングル・プロパティ検査の結果の分析 473
GUI チェックポイントの結果の分析 473
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 476
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 479
ビットマップ・チェックポイントの結果の分析 483
データベース・チェックポイントの結果の分析 484
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 486
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 489
ファイルの比較の結果の表示.....490
テスト実行中に検出された不具合の報告.....493

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.....499
単独のテストのプロパティの設定について 499
［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.....500
テストの一般情報の文書化 502
テストの説明情報の文書化 504
テスト・パラメータの管理 505
テストへのアドインの関連付け 508
現在のテスト設定の確認 510
起動アプリケーションおよび起動関数の定義.....512

第 22 章 :グローバル・テスト・オプションの設定.....519
グローバル・テスト・オプションの設定について 519
［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 520
一般オプションの設定.....523
フォルダ・オプションの設定.....528
記録オプションの設定.....532
テストの実行オプションの設定 548
通知オプションの設定.....564
概観オプションの設定.....570
適切なタイムアウトと遅延設定の選択 574

索引 579

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。

Mercury WinRunner 基本機能ユーザーズ・ガイドでは，WinRunner の機能と自動化されたテスト手順について詳細に説明します。**Mercury WinRunner 上級機能ユーザーズ・ガイド**では，WinRunner の高度な機能について解説します。**Mercury WinRunner 上級機能ユーザーズ・ガイド**を利用する読者は，**Mercury WinRunner 基本機能ユーザーズ・ガイド**に記載されている情報を熟知していることが望まれます。

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 ー 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。

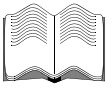
第 4 部　テストの実行 ー 基本

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。

WinRunner の関連マニュアル



WinRunner には，この『基本機能ユーザーズ・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 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』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。

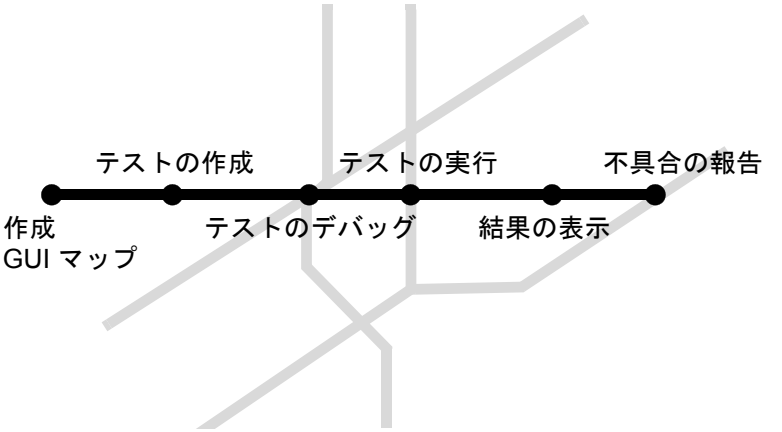
テストを実行するには，テスト・スクリプトを再生します。WinRunner は，アプリケーション上でマウス・ポインタを動かし，オブジェクトを選択し，キーボード入力を行って，ユーザをエミュレートします。WinRunner は，GUI マップでオブジェクトの物理的記述を読み取り，これらの記述に一致するオブジェクトをテスト対象アプリケーションで検索します。ウィンドウ内のオブジェクトの位置が変わっても，オブジェクトを見つけることができます。

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。

QuickTest Professional，Quality Center，ビジネス・プロセス・テスティング，および LoadRunner との統合については，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』を参照してください。

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。

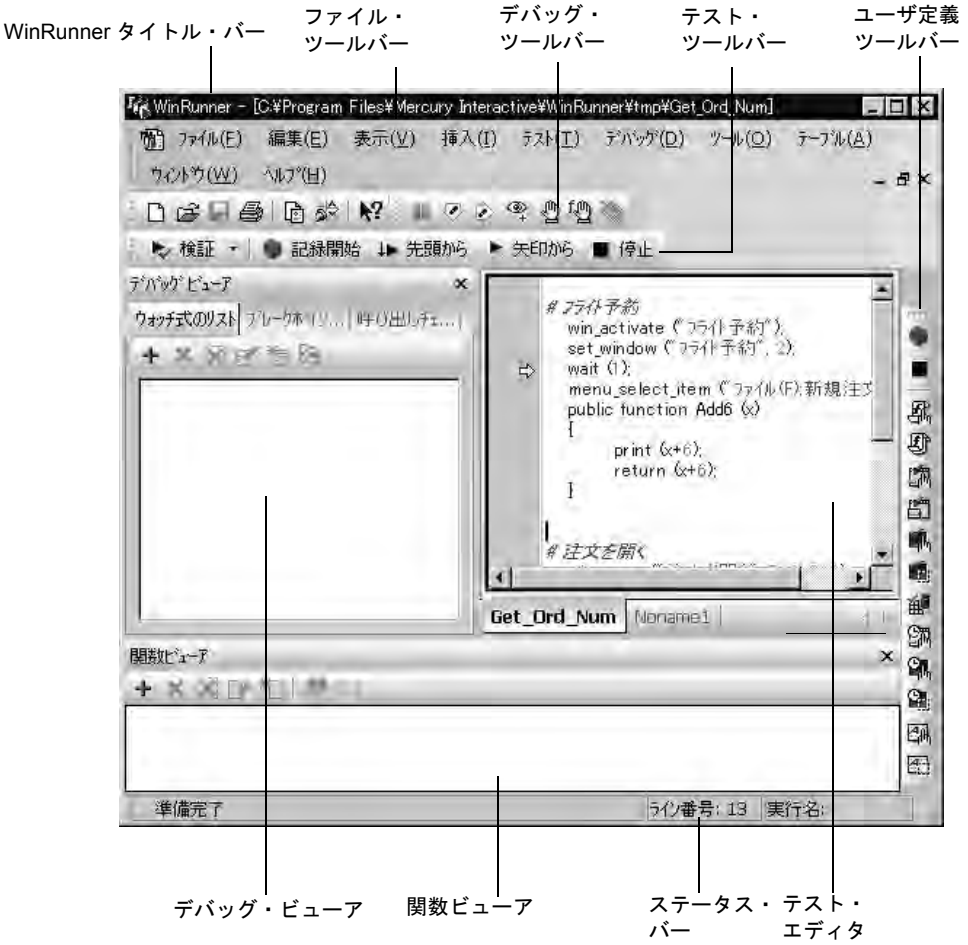
ヒント：次回 WinRunner を起動する際に［WinRunner へようこそ］ウィンドウを表示したくない場合は，［**起動時に表示**］チェック・ボックスをクリアします。このオプションは一度選択すると，後で WinRunner にようこそ画面を表示させることができます。［**ツール**］＞［**一般オプション**］を選択して，［**一般設定**］＞［**起動**］カテゴリを選択し，［**[ようこそ] 画面を起動時に表示する**］チェック・ボックスを選択します。

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。

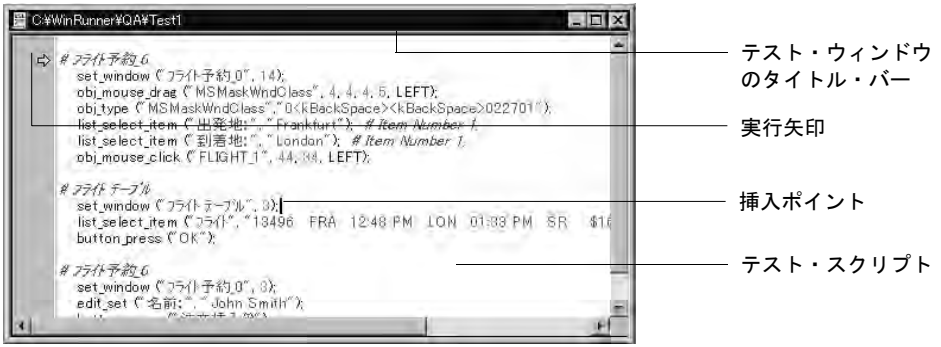


ヒント：標準設定では，各テストがテスト・エディタの異なるタブに表示されます。テストのタブが多く，エディタの底部に収まらない場合は，左矢印ボタンと右矢印ボタンを使用して，テストのタブを左または右にスクロールします。タブが表示されない場合は，［**一般オプション**］ダイアログ・ボックスの［**概観**］カテゴリで［**テスト タブを表示する**］オプションを選択して表示します。

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

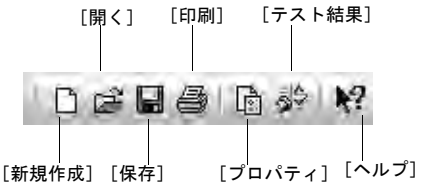
ツールバーはフローティング・ツールバーに変更できます。また，ユーザ定義ツールバーがフローティング・ツールバーになっている場合は，WinRunner を最小化してもユーザ定義ツールバーのコマンドにはアクセスできます。したがって，テスト対象アプリケーションで自由に作業できます。

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。

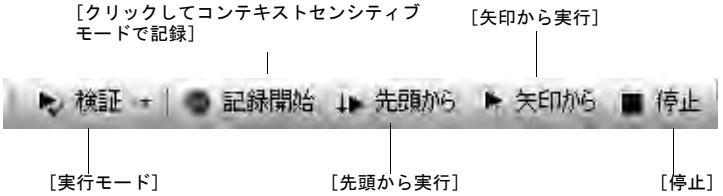


ユーザ定義ツールバーの詳細については，第 8 章「テストの設計」を参照してください。

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。

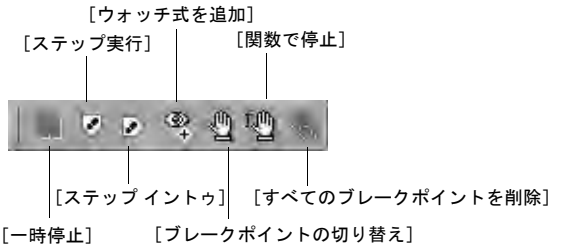


テスト・ツールバーの詳細については，第 19 章「テスト実行について」を参照してください。

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。

——[クリックしてコンテキストセンシティブモードで記録]
——[停止]
——[オブジェクト / ウィンドウの関数を挿入]
——[関数ジェネレータから関数を挿入]
——[オブジェクト / ウィンドウの GUI チェックポイント]
——[複数のオブジェクトの GUI チェックポイント]
——[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]
——[選択範囲のビットマップチェック・ポイント]
——[標準データベースチェックポイント]
——[オブジェクト / ウィンドウプロパティの同期化ポイント]
——[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]
——[選択範囲のビットマップの同期化ポイント]
——[オブジェクト / ウィンドウからのテキスト取得]
——[画面領域からテキスト取得]

ユーザ定義ツールバーのカスタマイズについては，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 20 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。

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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 20 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。

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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 15 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 1 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 2 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 3 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

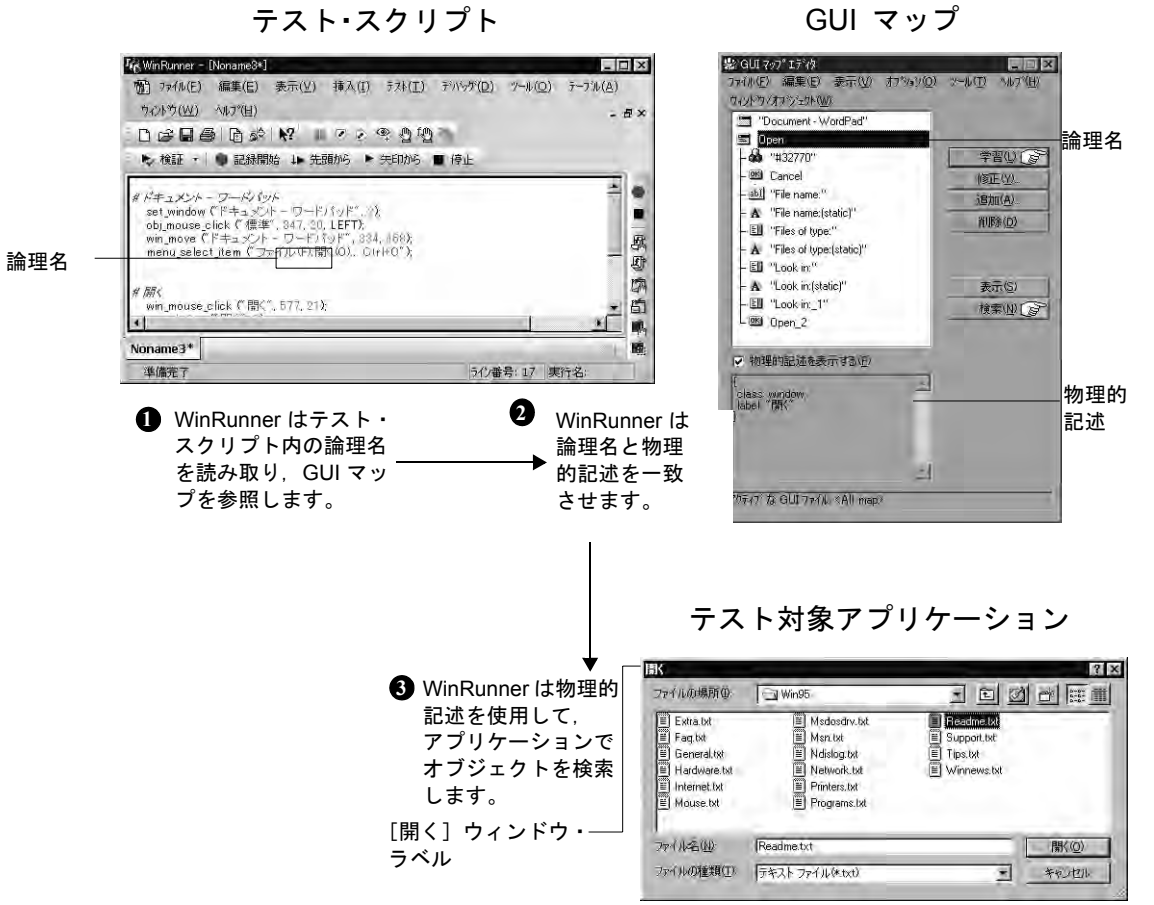
WinRunner は，オブジェクトの「**物理的記述**」に基づいて，テスト対象アプリケーションの各 GUI オブジェクトを識別します。物理的記述とは，一連の物理的プロパティとそれらの値のことです。これらの属性と値の対は，GUI マップの中で以下の形式で記録されます。

{*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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 2 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 33 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。

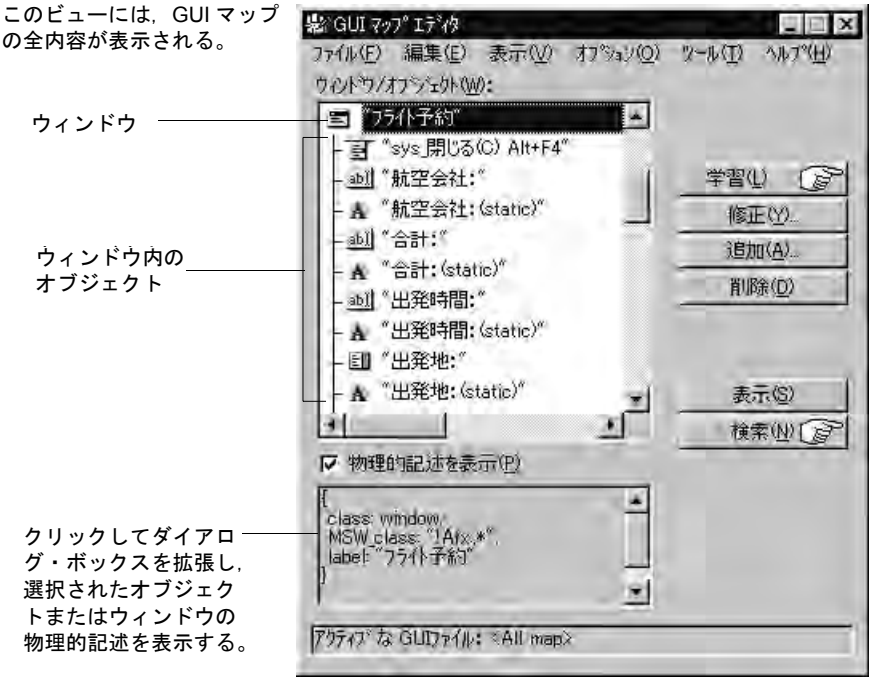
静的テキスト・オブジェクトの場合，テキストと「:(static)」という文字列をつなげたものとなります。例えば，「ファイル名 (N)」という静的テキストの論理名は，「ファイル名 (N):(static)」です。

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，44 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。

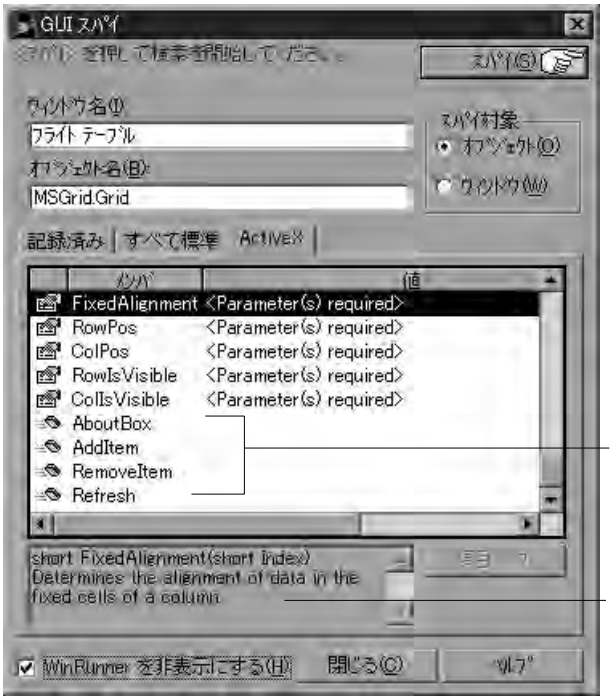
前の例で［コピー］をクリックすると，次の物理的記述がクリップボードに貼り付けられます。

```
{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，91 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，79 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，65 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。

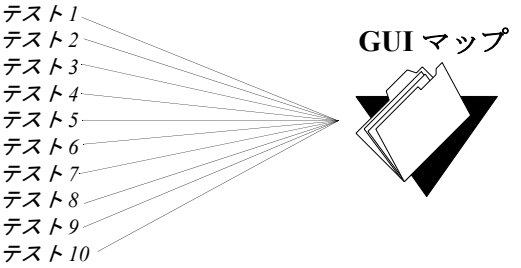
GUI マップ・ファイルはテストに依存しないので，テストを開くときも自動的にはロードされません。したがって，テストを実行する前に，適切な GUI マップ・ファイルをロードしておく必要があります。WinRunner は，これらのファイルを使って，テスト対象アプリケーション内のオブジェクトを識別します。起動テストの中に **GUI_load** ステートメントを入れておくと最も効率的です。WinRunner を起動すると，初期化テストが自動的に実行され，指定された GUI マップ・ファイルがロードされます。初期化テストの詳細については，**『Mercury WinRunner 上級機能ユーザーズ・ガイド』**の第 23 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 2 章「GUI マップの構成設定」を参照してください。

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。

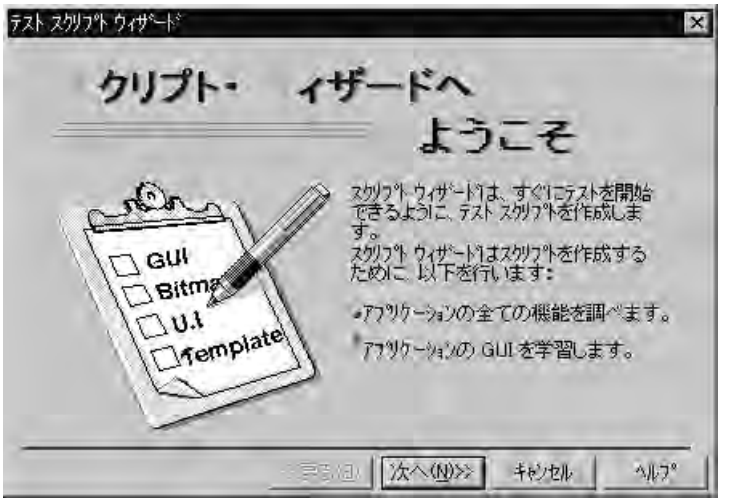
テスト特有の GUI マップ・ファイル・モードで作業する場合，テスト・スクリプト・ウィザードは使用できません。テスト・スクリプト・ウィザードは，WebTest またはその他の特定のアドインがロードされている場合は使用できません。使用しているアドインでテスト・スクリプト・ウィザードを使用できるかどうかについては，アドインのマニュアルを参照してください。

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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 23 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

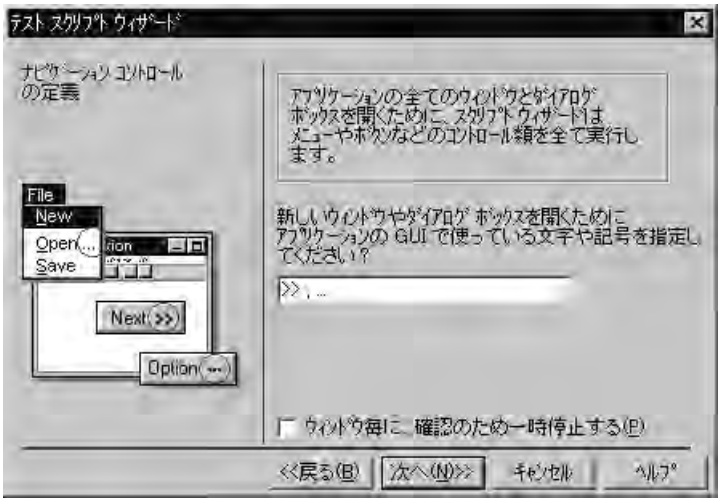
このテストを実行すると，WinRunner はアプリケーションのインタフェースを探し，Microsoft Windows の標準に対応していない事項を報告します。

➤［**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 59 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。

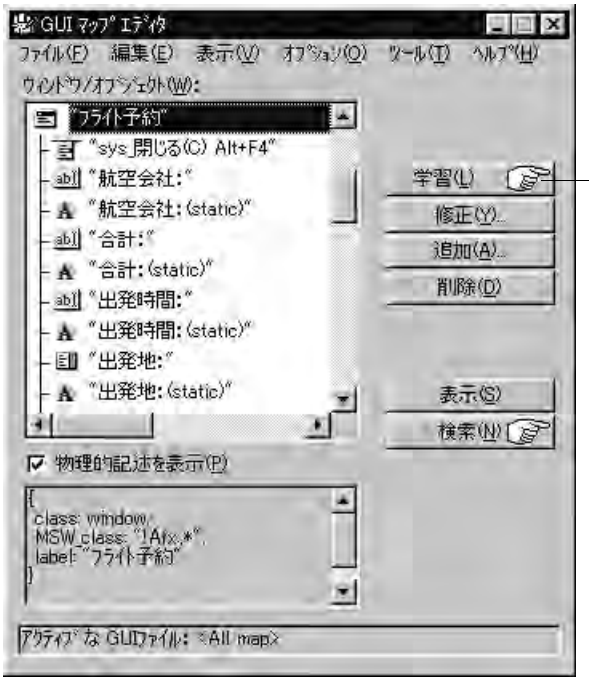
GUI マップ・エディタを使った GUI の学習

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。

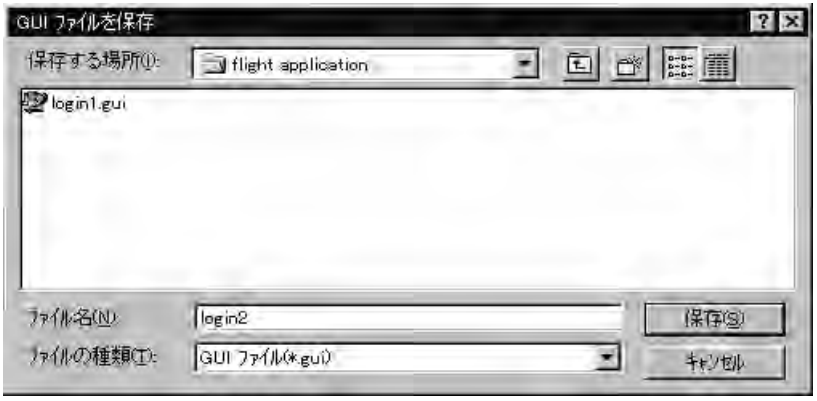
一時 GUI マップ・ファイルの内容を恒久 GUI マップ・ファイルに保存するには，次の手順を実行します。

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，85 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 26 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 23 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 8 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 7 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 26 章「テスト工程の管理」を参照してください。

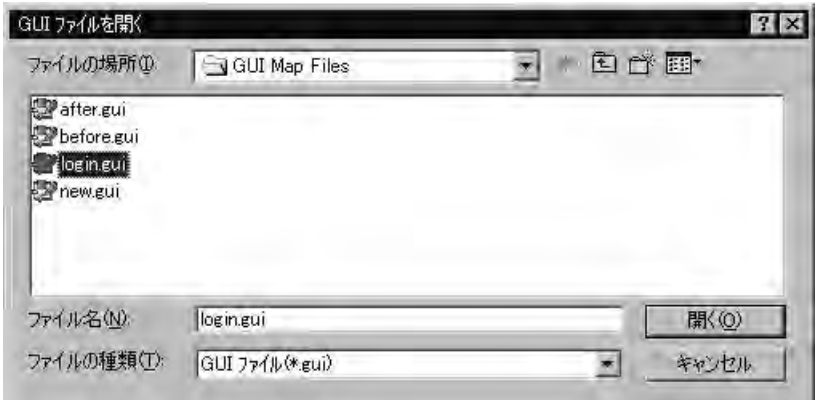
GUI マップ・エディタを使用して，ファイル・システムから GUI マップ・ファイルをロードするには，次の手順を実行します。

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，79 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，43 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 11 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 1 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。

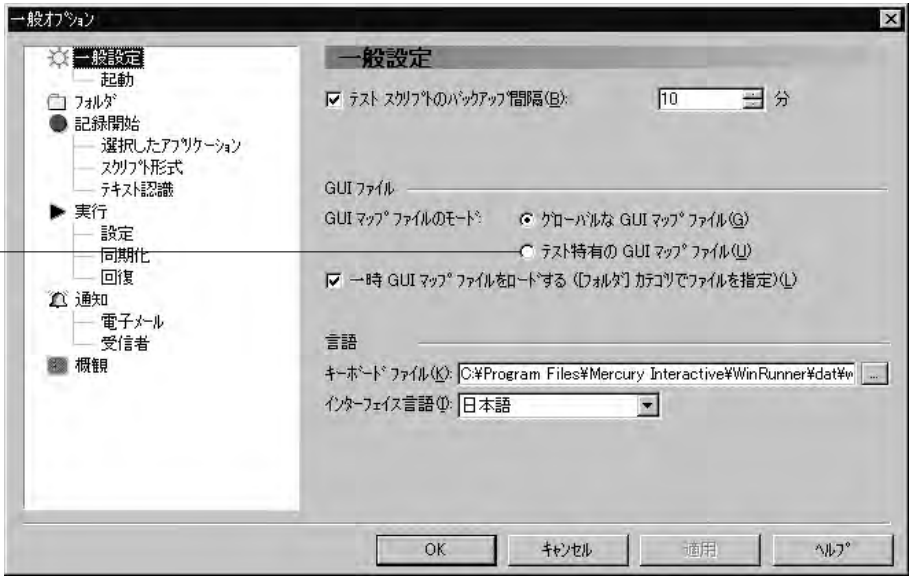
1 ［**ツール**］＞［**一般オプション**］を選択します。

［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，79 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，43 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。

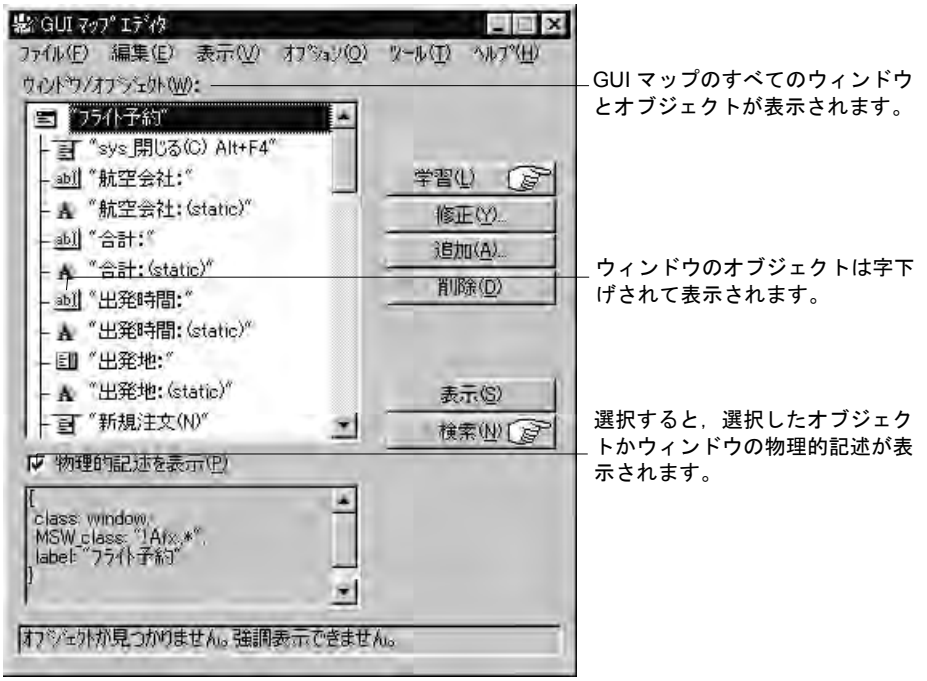
注：［**テスト特有の GUI マップ ファイル**］モードで作業している場合は，GUI マップ・ファイルのロード，アンロードを手作業で行わないでください。

GUI マップ・エディタ

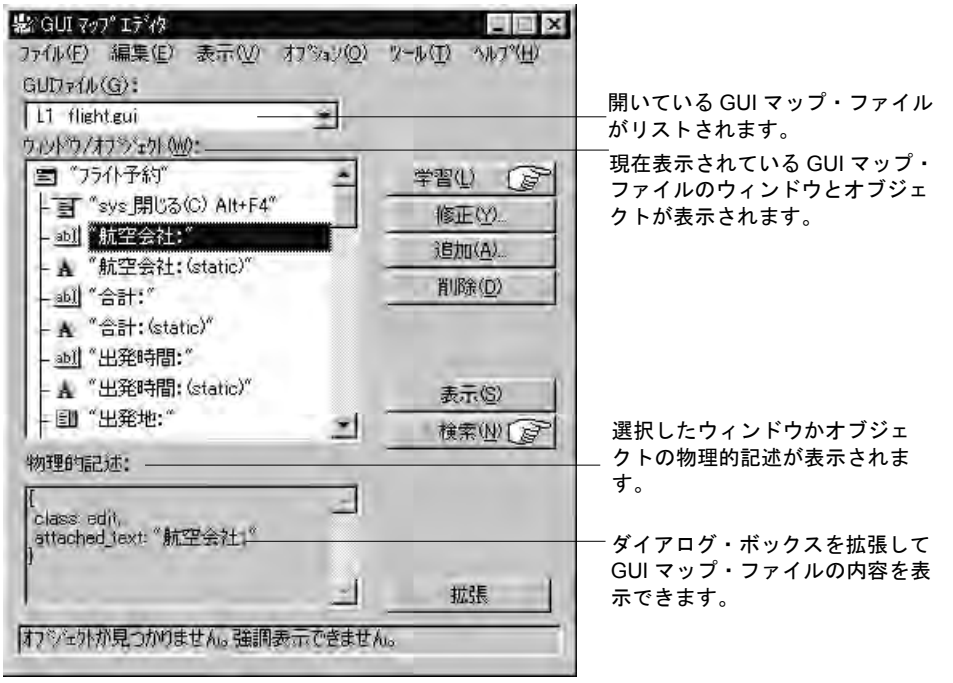
GUI マップ・エディタを使えば，いつでも GUI マップを編集できます。GUI マップ・エディタを開くには，［**ツール**］＞［**GUI マップ エディタ**］を選択します。

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

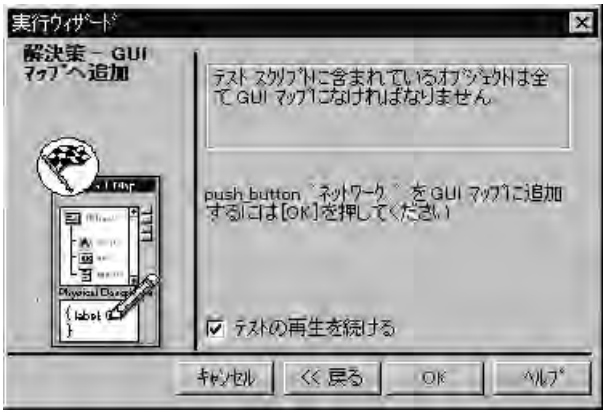
実行ウィザードは，テストの実行の妨げとなる，アプリケーションの GUI における変更を検出します。テスト実行時に WinRunner がオブジェクトを見つけられないと，実行ウィザードが自動的に開きます。実行ウィザードは，アプリケーションのオブジェクトを指すようユーザに指示し，オブジェクトが見つからなかった原因を特定し，解決策を提示します。例えば，実行ウィザードから適切な GUI マップ・ファイルをロードするよう指示がでます。ほとんどの場合，GUI マップに自動的に新しい記述が追加されるか，既存の記述が修正されます。この手順を完了すると，テスト実行が継続されます（この更新作業により，以降のテスト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

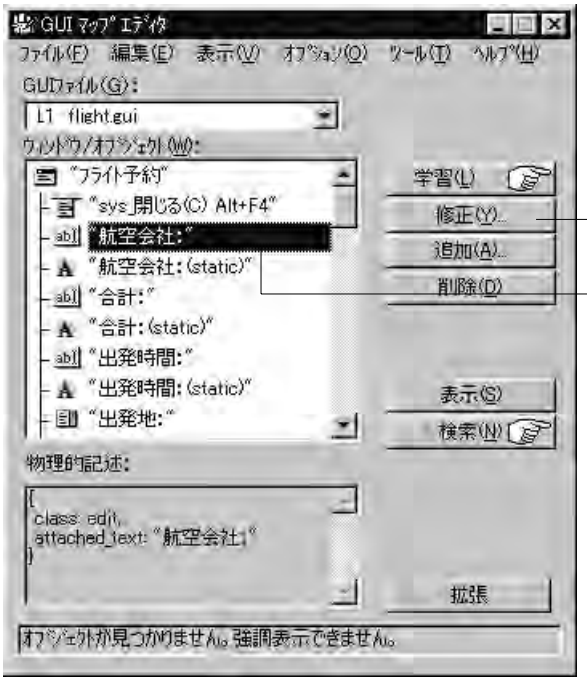
1 ［**ツール**］＞［**GUI マップ エディタ**］を選択します。GUI マップ・エディタが開きます。

2 ［**表示**］＞［**GUI ファイル**］を選択します。

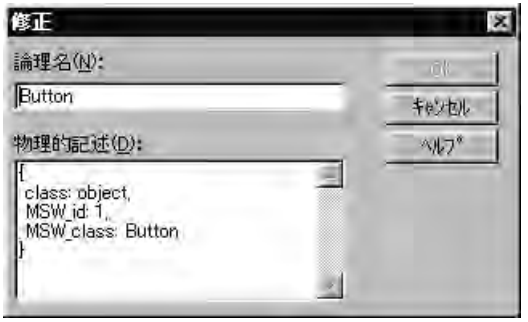
3 適切な GUI マップ・ファイルがロードされない場合は，［**ファイル**］＞［**開く**］を選択してファイルを開きます。

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。

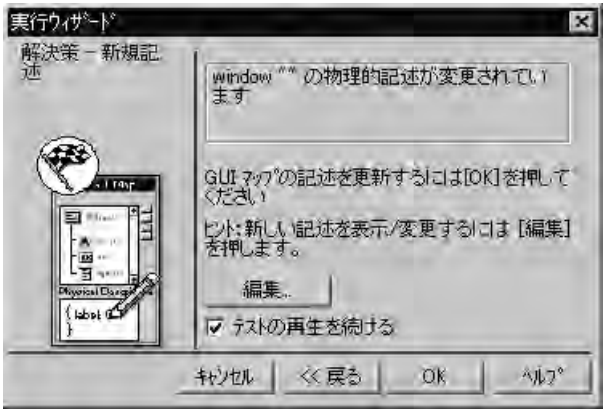
学習したラベルが変わったために WinRunner がウィンドウを認識できない場合，実行ウィザードが開き，不明ウィンドウを特定するようにユーザに求めます。ウィンドウを特定すると，WinRunner はラベルが変更されたことを認識し，その変更に合わせてウィンドウの物理的記述を変更します。

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 －」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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 2 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の 第 6 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 7 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。

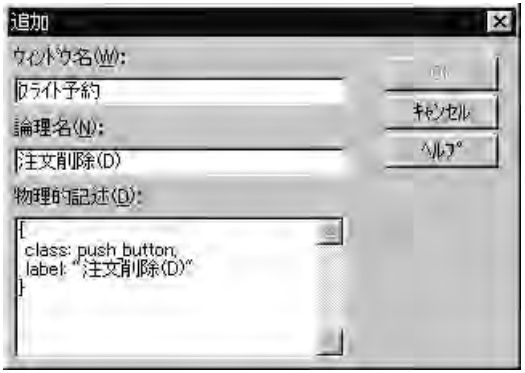
1 ［**ツール**］＞［**GUI マップ エディタ**］を選択して，GUI マップ・エディタを開きます。

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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 20 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。

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，99 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 104 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，108 ページ「テストへのチェックポイントの追加」と 109 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，115 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」（25 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。

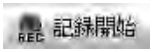
［**ファイル**］**メニュー** : {class:menu_item, label: **ファイル** , parent:None}
［**印刷**］**コマンド** : {class: menu_item, label: " 印刷 ... Ctrl+P", parent: **ファイル** }
［**印刷**］**ウィンドウ** : {class:window, label: 印刷 }
［**OK**］**ボタン** : {class:push_button, label:OK}

（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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 2 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。

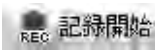
```
# マウスの軌跡
move_locator_track (1);

# 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 7 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 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。

➤ 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 7 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 20 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 22 章「グローバル・テスト・オプションの設定」および第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 20 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 8 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の「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。

編集コマンド	説明
［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。
［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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 19 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 19 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。詳細については，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』を参照してください。

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。『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』を参照してください。

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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 26 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 26 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

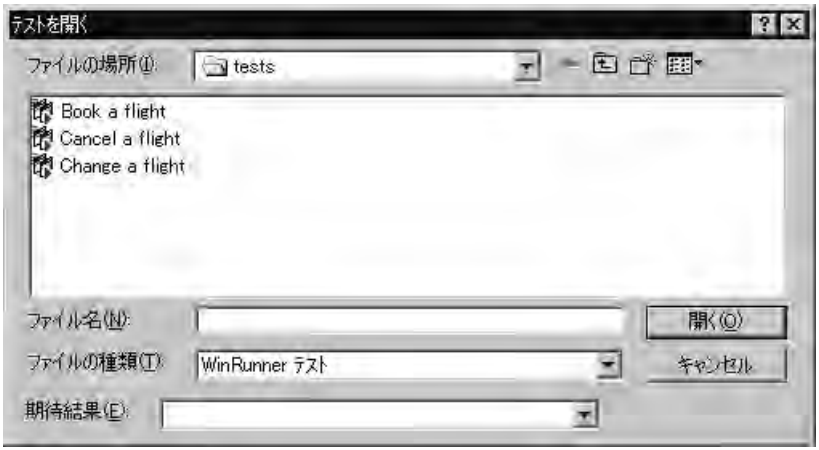
ファイル・システムまたは Quality Center プロジェクトから既存のテストを開くことができます。

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，123 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。



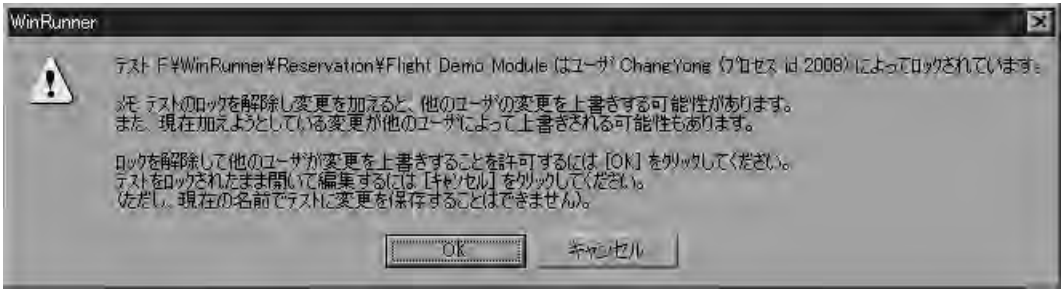
2 ディレクトリ名のボックスで，開きたいテストの場所をクリックします。

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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 26 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 26 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，**『Mercury WinRunner 上級機能ユーザーズ・ガイド』**の第 26 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 26 章「テスト工程の管理」を参照してください。

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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 6 章「正規表現の使い方」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，564 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，548 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。

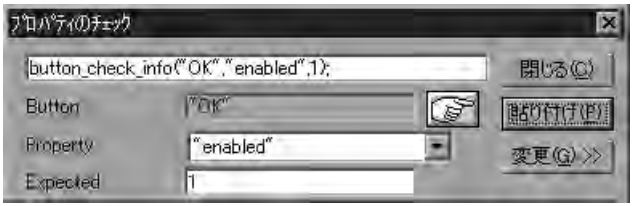
プロパティ値で GUI チェックポイントを作成するには以下のようにします。

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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 8 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，163 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 141 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，175 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，169 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，153 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，141 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，150 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，163 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 チェックポイントー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，139 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，175 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，169 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，156 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，141 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，150 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，163 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。

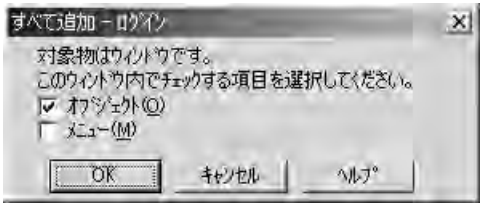
1 つのウィンドウ内のすべての GUI オブジェクトを対象に標準の検査を行う GUI チェックポイントを作成するには，次の手順を実行します。

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 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，153 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，143 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 435 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，61 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，145 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，177 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，306 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，61 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，145 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，160 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，169 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，169 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，150 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，425 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，177 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，175 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，169 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。

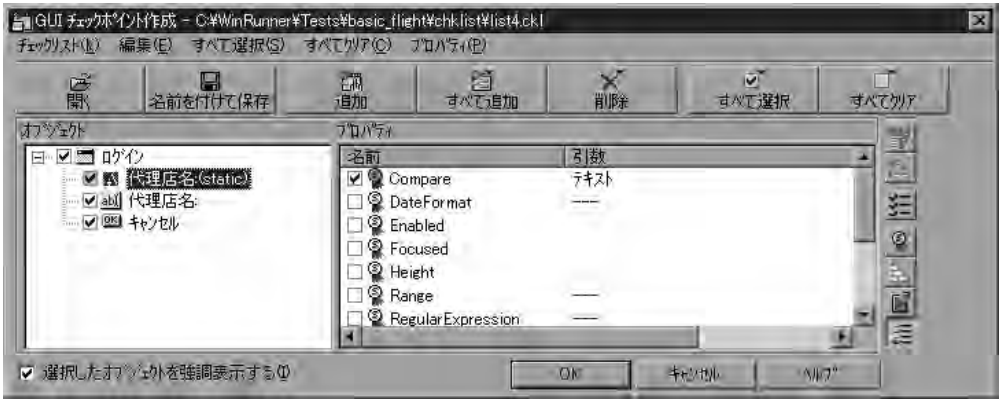
ボタン	説明
	［**プロパティ リスト**］ボタンで，**gui_ver_add_class** 関数を使ってカスタマイズされたクラスに対してのみ定義された **ui_function** parameter を呼び出します。このボタンは，［オブジェクト］表示枠で少なくとも 1 つのオブジェクトが，**gui_ver_add_class** 関数によって **ui_function** parameter が定義されているクラスに属している場合にのみ表示されます。詳細については，『**WinRunner カスタマイズ・ガイド**』を参照してください。
	［**期待結果値を編集**］ボタンで，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，175 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，169 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，169 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，425 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，169 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 171 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 6 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 6 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，177 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。

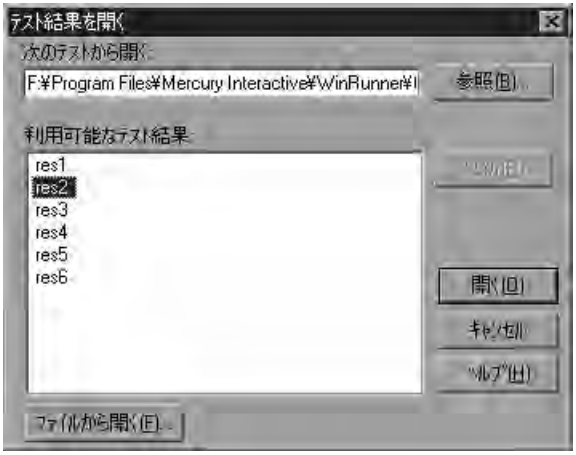
既存の GUI チェックポイントの期待結果を変更するには，次の手順を実行します。

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 175 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，489 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，36 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 2 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，193 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 7 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，185 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。

プロパティ名	説明
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。
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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 7 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

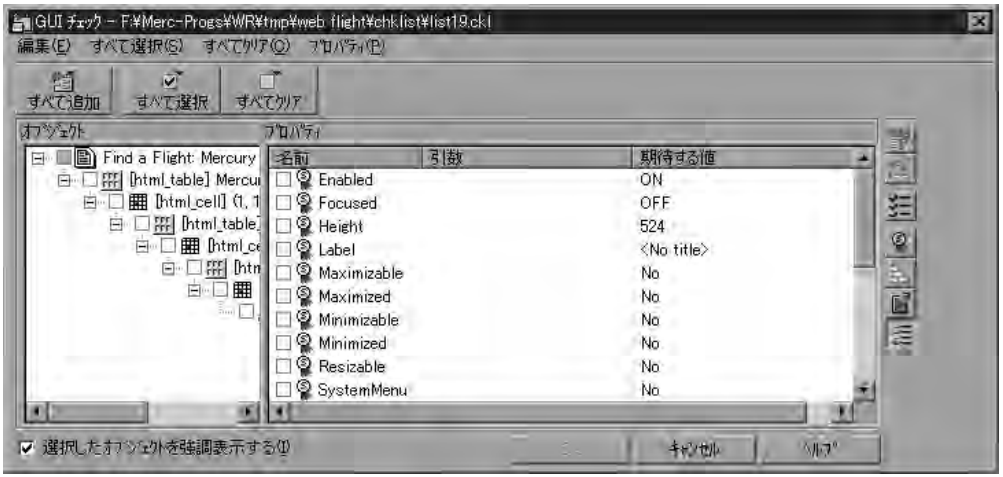
標準的なフレーム・プロパティを検査する GUI チェックポイントを作成できます。

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

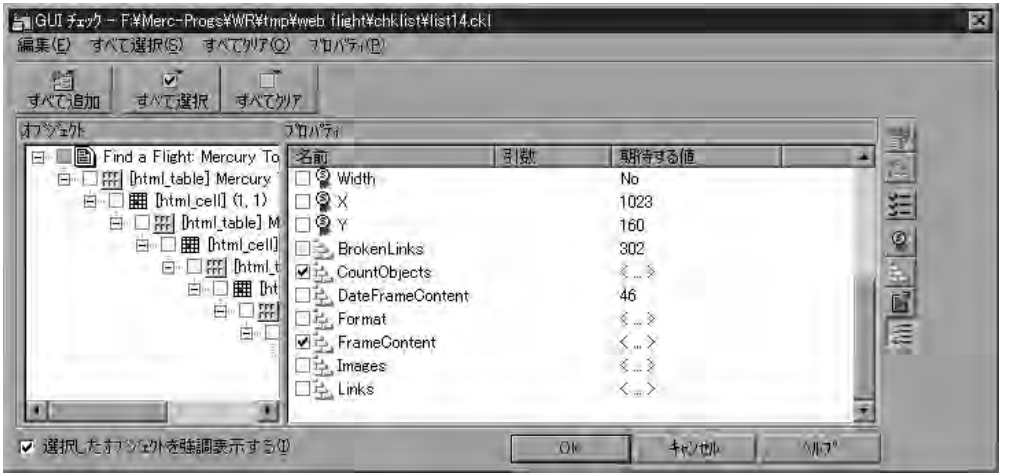
フレーム内のオブジェクトの数を検査する GUI チェックポイントを作成できます。

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。

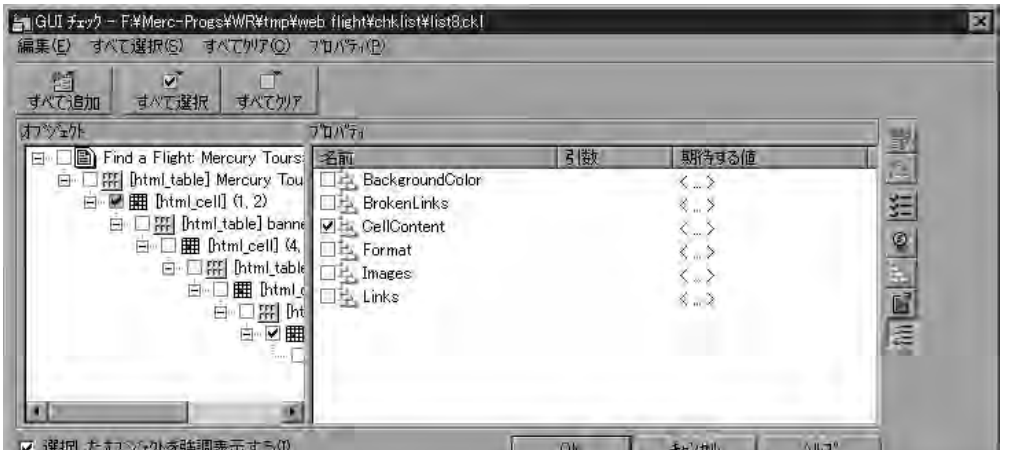
フレーム，テーブル，セルの構造の検査

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

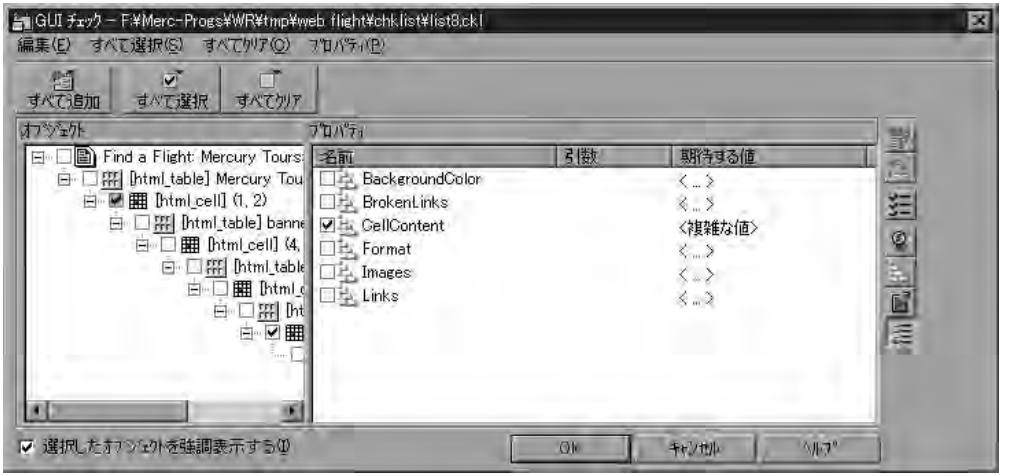
フレーム，セル，テキスト・リンク，画像リンク，画像の内容を検査する GUI チェックポイントを作成できます。テーブルの内容を検査するには，209 ページ「テーブルの内容の検査」を参照してください。

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

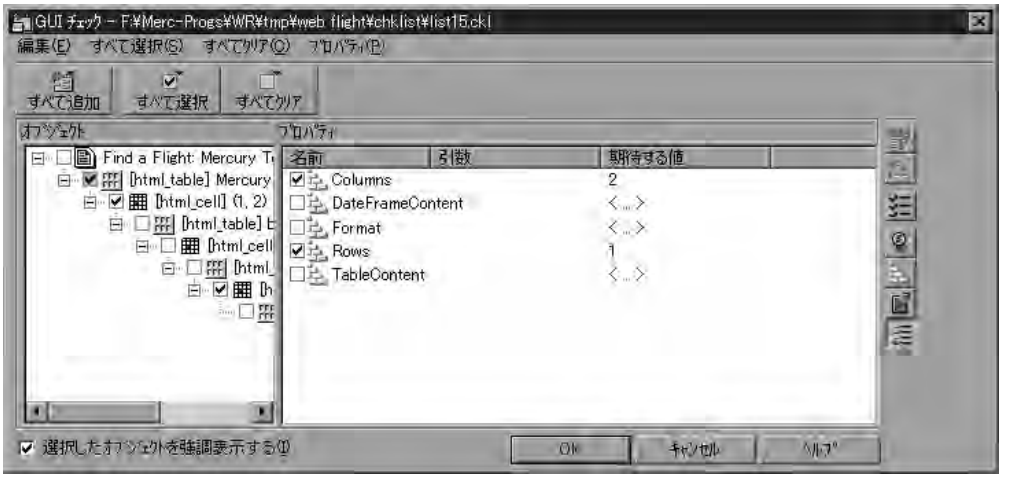
テーブル内の列数と行数を検査する GUI チェックポイントを作成できます。

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

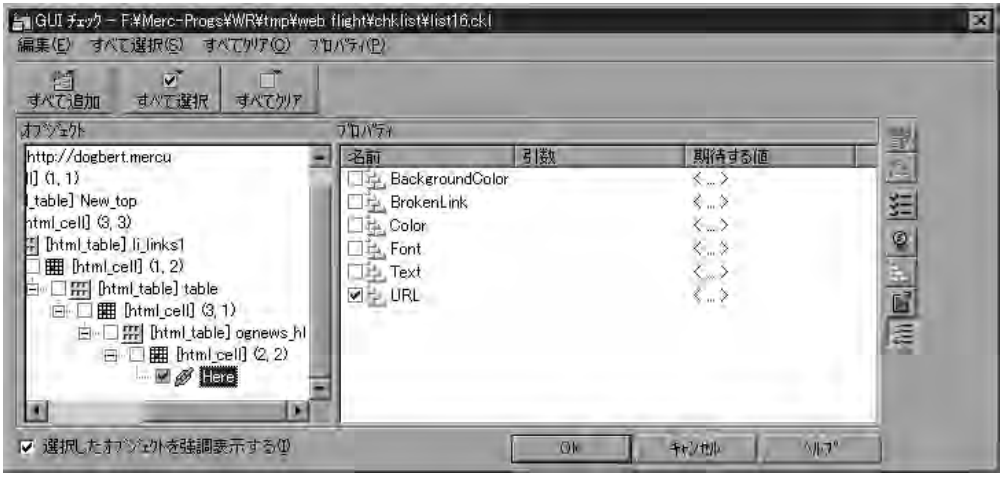
Web ページ内のテキスト・リンクまたは画像リンクの URL を検査する GUI チェックポイントを作成できます。

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

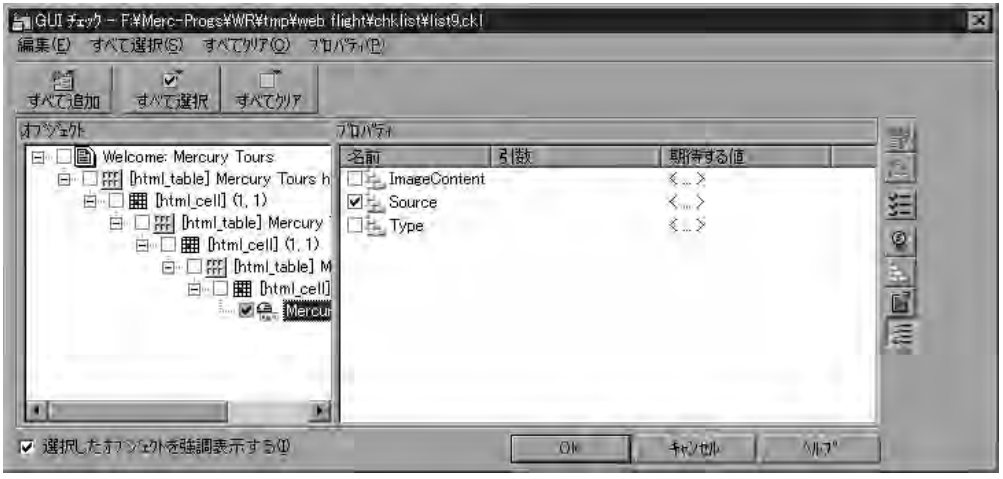
Web ページ内の画像または画像リンクのソースとタイプを検査する GUI チェックポイントを作成できます。

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

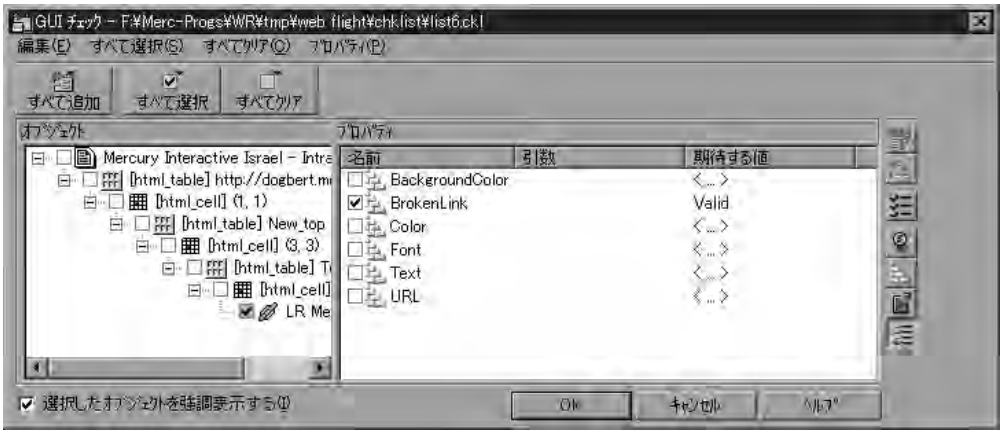
テキスト・リンクまたは画像リンクが有効かどうかを検査するチェックポイントを作成できます。フレーム内の 1 つの壊れたリンクを検査するチェックポイント，またはすべての壊れたリンクを検査するチェックポイントを作成できます。

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。

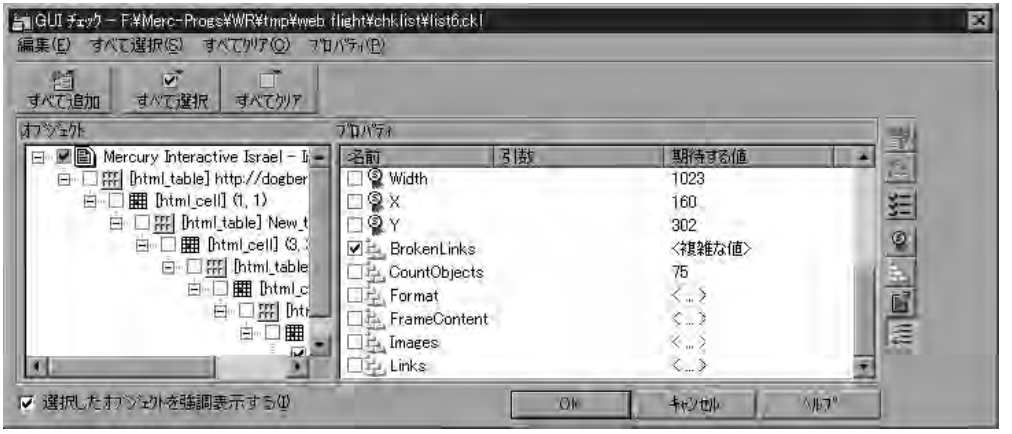
WinRunner はオブジェクトの情報をキャプチャし，それをテストの期待結果フォルダに格納します。画面が WinRunner のウィンドウに戻り，チェックポイントがテスト・スクリプトに **obj_check_gui** または **win_check_gui** ステートメントとして表示されます。**obj_check_gui** および **win_check_gui** 関数の詳細については，「**TSL リファレンス**」を参照してください。

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，211 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，211 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

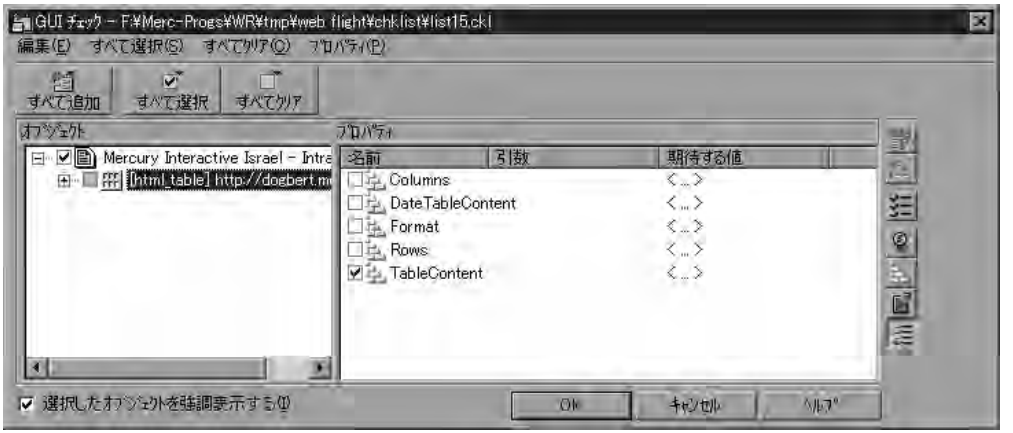
テーブルのテキストの内容を検査するチェックポイントを作成できます。

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，211 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，215 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，216 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，212 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

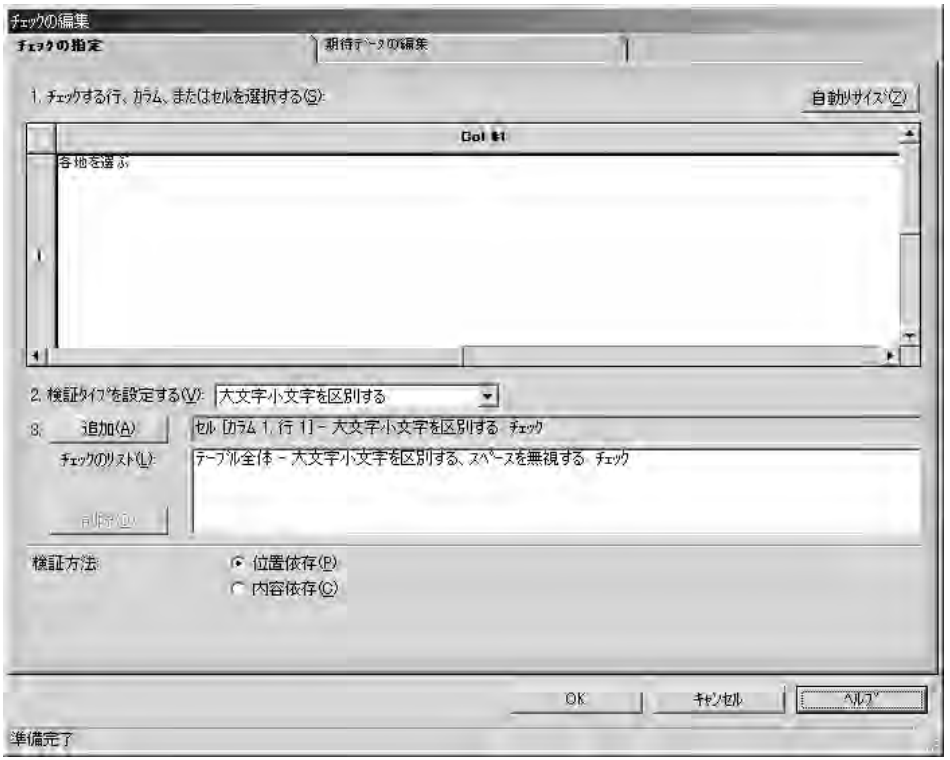
➤ **［キー］**：WinRunner は，［**キーカラムの選択**］リスト・ボックスで指定された列のデータに基づいて，選択された行を検索します。例えば，行の到着時間に基づいて，x ページのテーブルの 2 行目を識別するように指定できます。行の位置が異なっても，不一致とは見なされません。選択されたキーよって，行が一意に識別されない場合，WinRunner は最初に一致する行を検査します。行を一意に識別するために，複数の列をキーとして使用できます。

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。

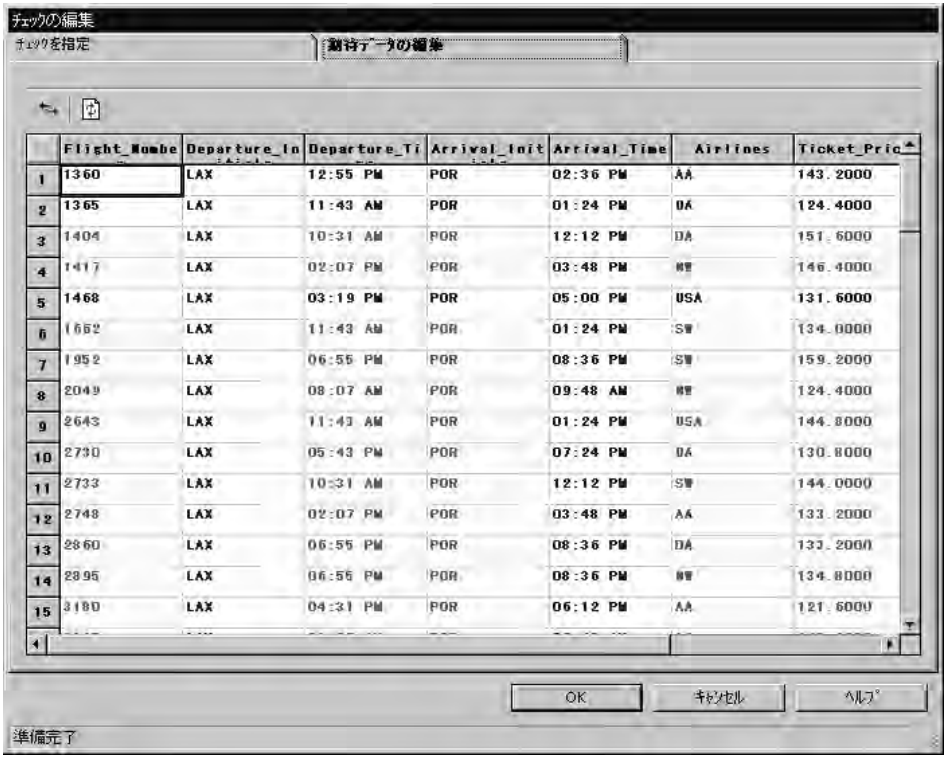
［**OK**］をクリックし，［チェックの編集］ダイアログ・ボックスの両方のタブに対する変更を保存します。ダイアログ・ボックスが閉じ，［GUI チェック］ダイアログ・ボックスに戻ります。

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，225 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，241 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。

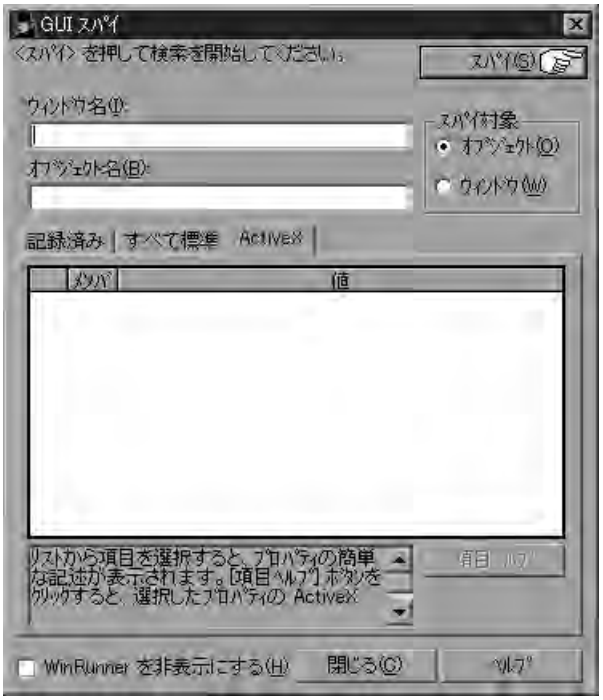
ActiveX と Visual Basic コントロールのプロパティの表示

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，232 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 8 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，229 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，236 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，236 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。

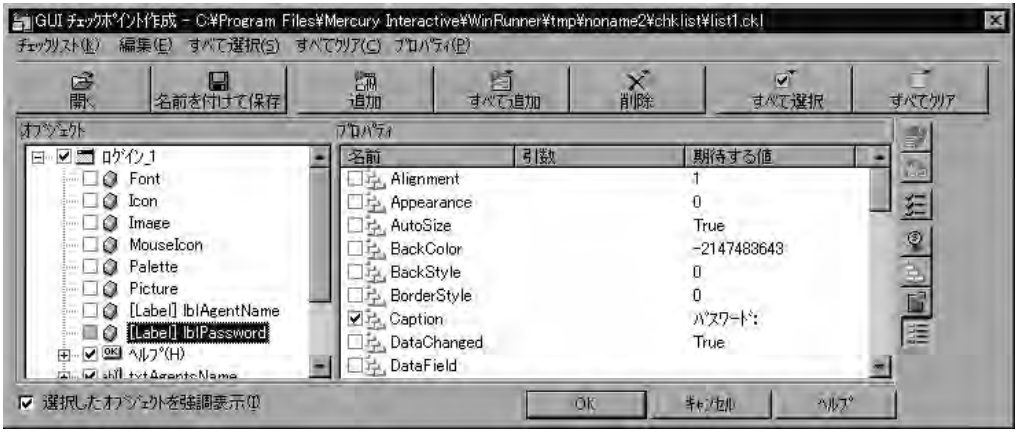
1 ［**挿入**］＞［**GUI チェックポイント**］＞［**複数のオブジェクト**］を選択します。［GUI チェックポイント作成］ダイアログ・ボックスが開きます。

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，232 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，232 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

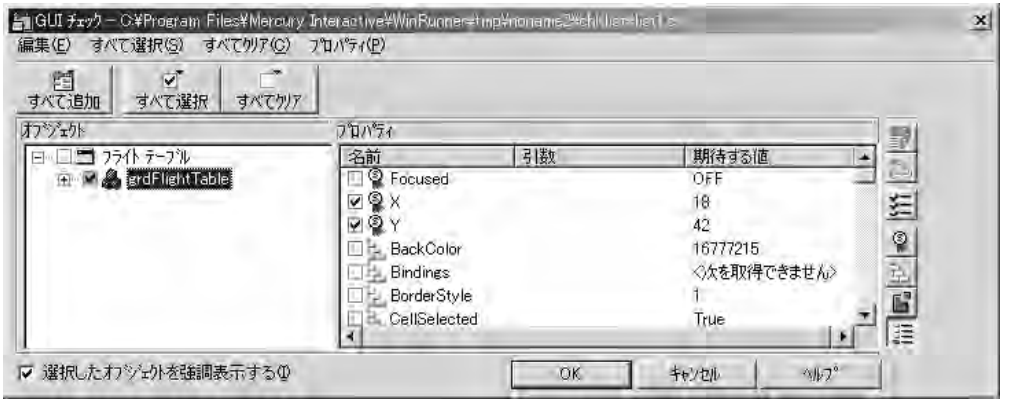
ActiveX と Visual Basic のコントロールには，それぞれのプロパティを含むサブオブジェクトを含めることができます。サブオブジェクトには，例えばフォントがあります。フォントはサブオブジェクトであるため，テスト対象アプリケーションで強調表示することができません。適切なアドイン・サポートをロードすれば，［GUI チェック］ダイアログ・ボックスを使ってサブオブジェクトのプロパティを検査する GUI チェックポイントを作成することができます。GUI チェックポイントの詳細については，第 9 章「GUI オブジェクトの検査」を参照してください。

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。

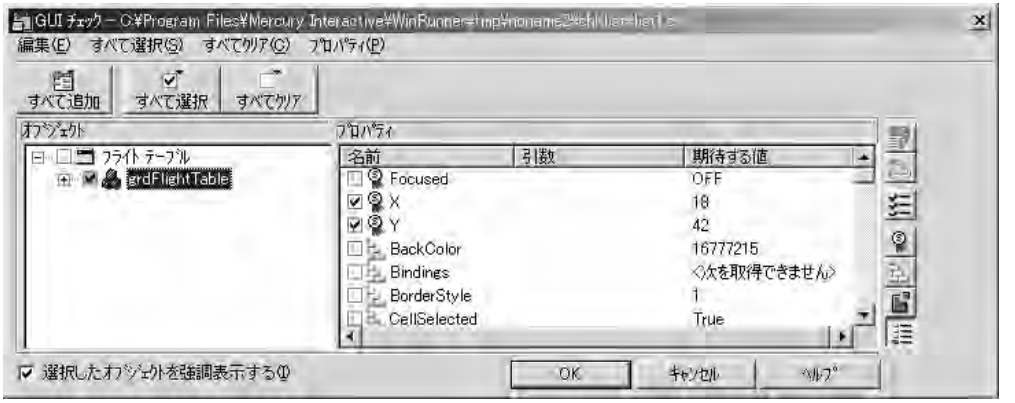
ActiveX または Visual Basic のコントロールのサブオブジェクトを検査するには，次の手順を実行します。

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。

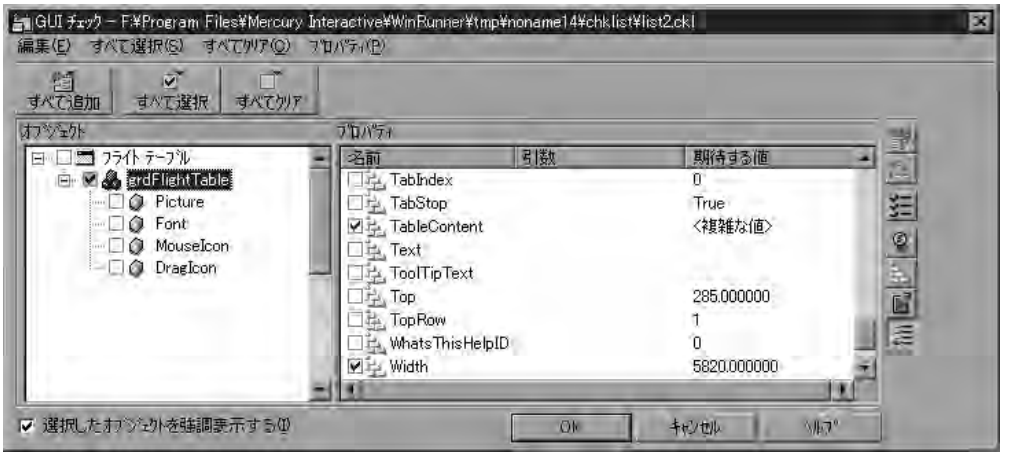


［**オブジェクト**］表示枠にはオブジェクトと，そのサブオブジェクトが表示されます。この例では，サブオブジェクトが「grdFlightTable」オブジェクトの下に表示されています。［**プロパティ**］表示枠には，［**オブジェクト**］表示枠で強調表示されているサブオブジェクトのプロパティが表示されます。各サブオブジェクトには複数の標準のプロパティ検査があることに注目してください。こ

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 チェックポイントーオブジェクト / 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，259 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，259 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。

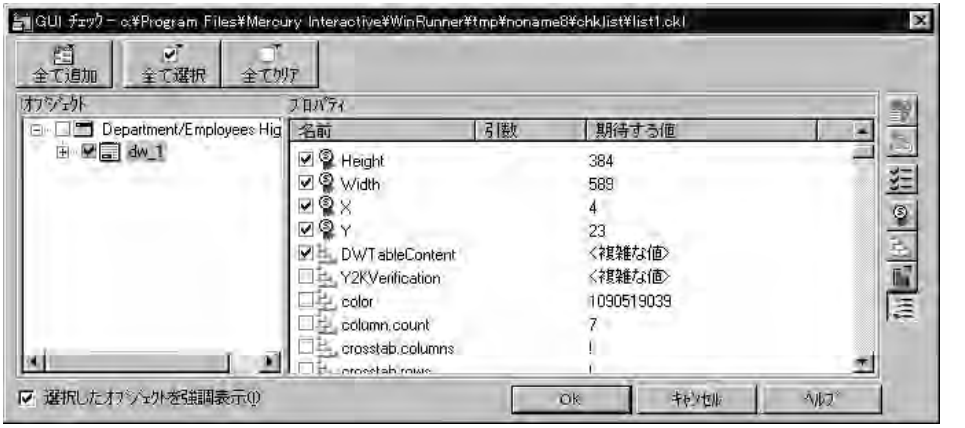
- DataWindow
- DataWindow のカラム
- DataWindow のテキスト
- DataWindow のレポート
- DataWindow のグラフ
- DataWindow の計算カラム

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。

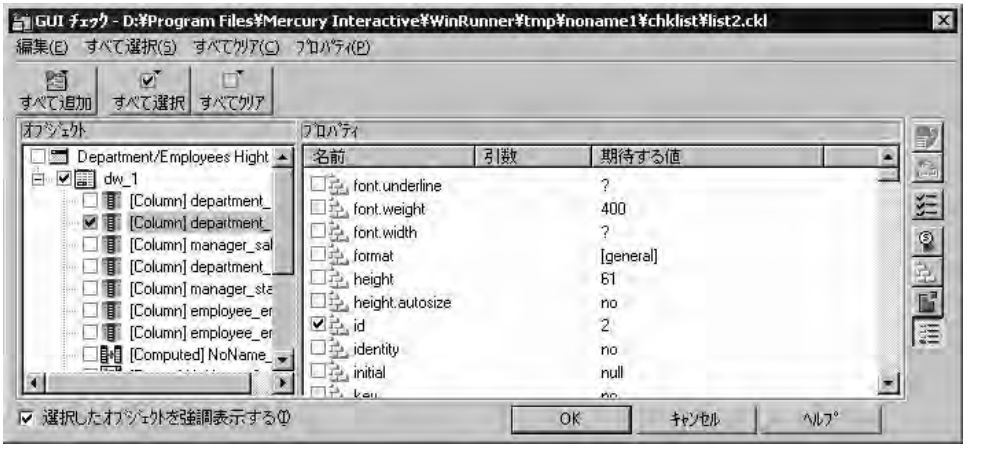
DataWindow 内オブジェクトのプロパティを検査するには，次の手順を実行します。

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，256 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。

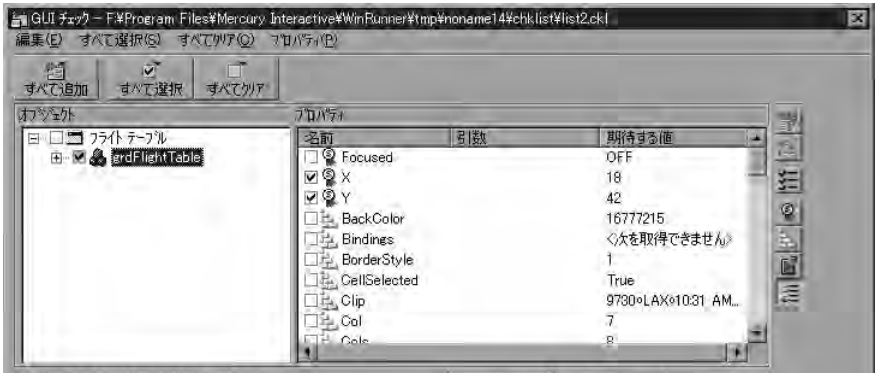
下に示す例では，Visual Basic のアドイン・サポートがインストールされている WinRunner と，サンプルの Visual Basic Flights アプリケーションを使用します。

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。

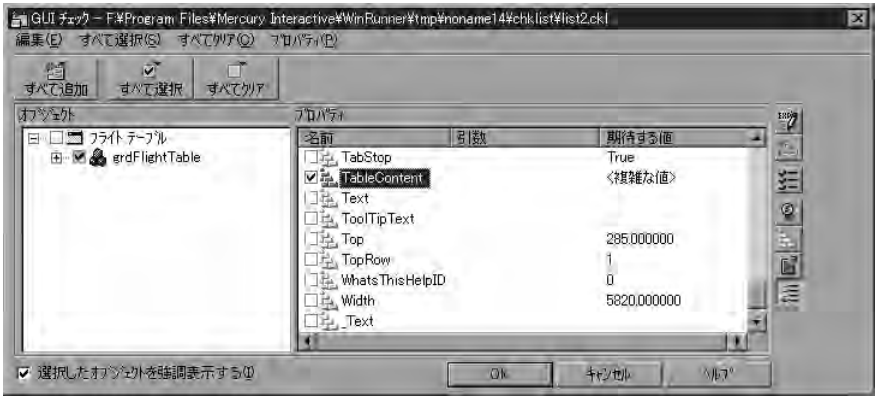
WinRunner は，テーブルに関する情報をキャプチャするのに数秒かかる場合があります。その後，［**GUI チェック**］ダイアログ・ボックスが開きます。



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，259 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，263 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，264 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，260 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

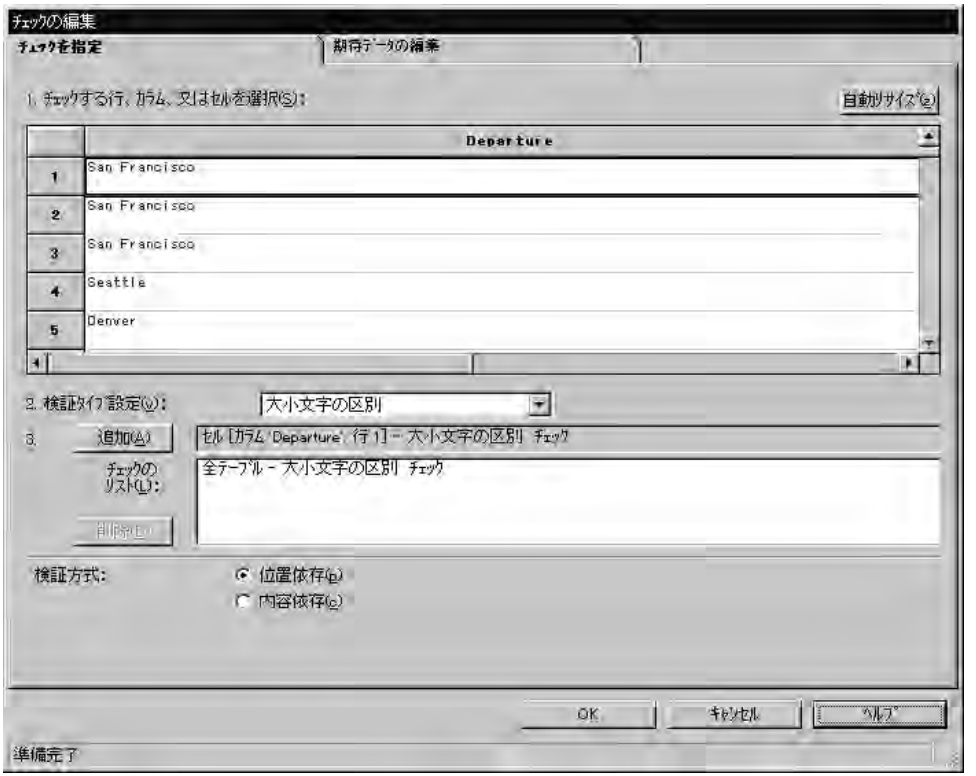
➤ **［キー］**: WinRunner は，テーブル内のすべてのカラムの名前をリストする［**キー選択** ］リスト・ボックスで指定された「キー」カラム内のデータ（複数も可）に基づいて対象の中の行を探します。例えば，WinRunner に秒の到着時間に基づいて，265 ページのテーブル内の 2 行目を特定させることができます。行が移動しても不一致にはなりません。キー選択で行を一意に特定できない場合，WinRunner は最初に一致する行のみ検査します。1 つ以上のキー・カラムを指定して，行を一意に特定することができます。

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。

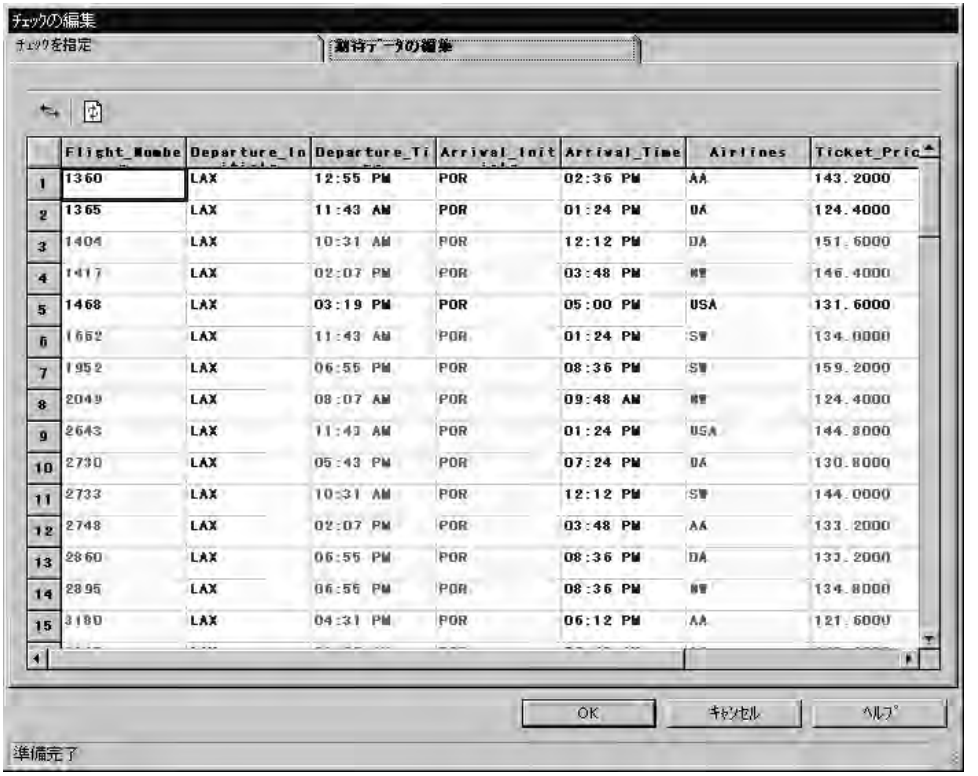
➤［**大文字小文字を区別しない，スペースを無視する**］: WinRunner は，大文字と小文字の違いと空白文字の違いを無視して，内容に基づいてセルの内容を検査します。WinRunner は，内容の違いだけを不一致として報告します。

［**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，278 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，319 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 22 章「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,『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 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，278 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。

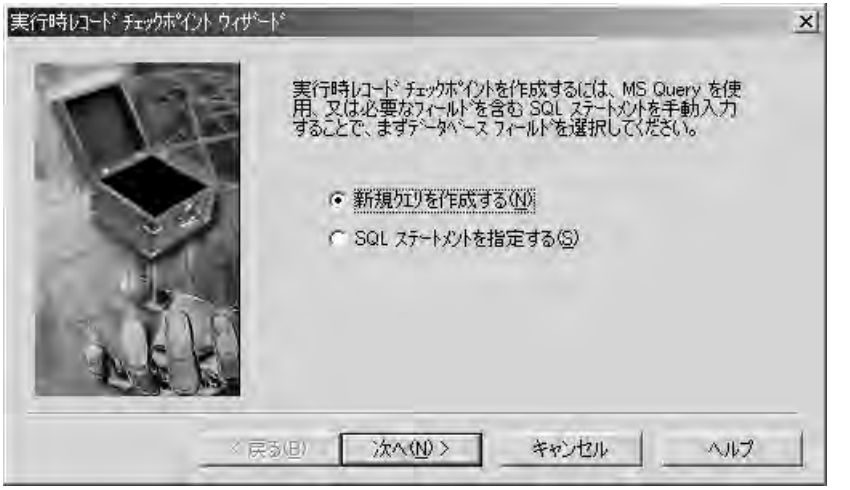
実行時レコード・チェックポイント・ウィザードの使用

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，323 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，297 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 ...]

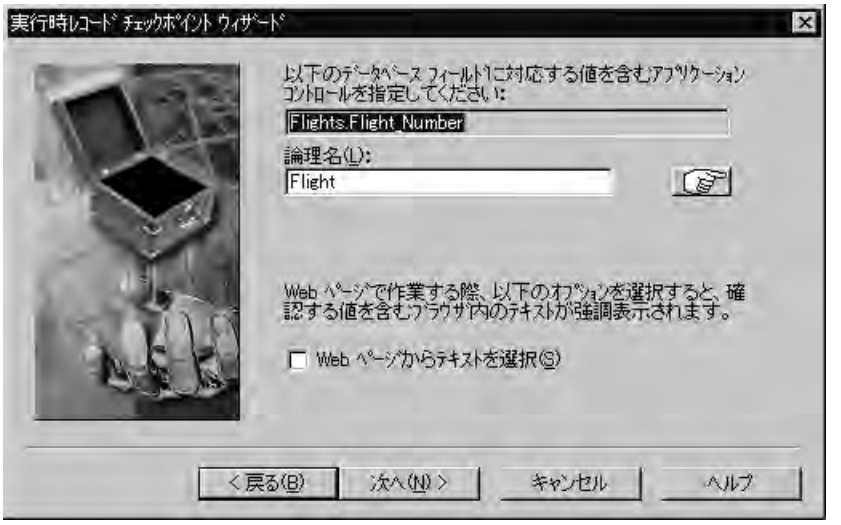
データベース・フィールドとの対応付け画面

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。

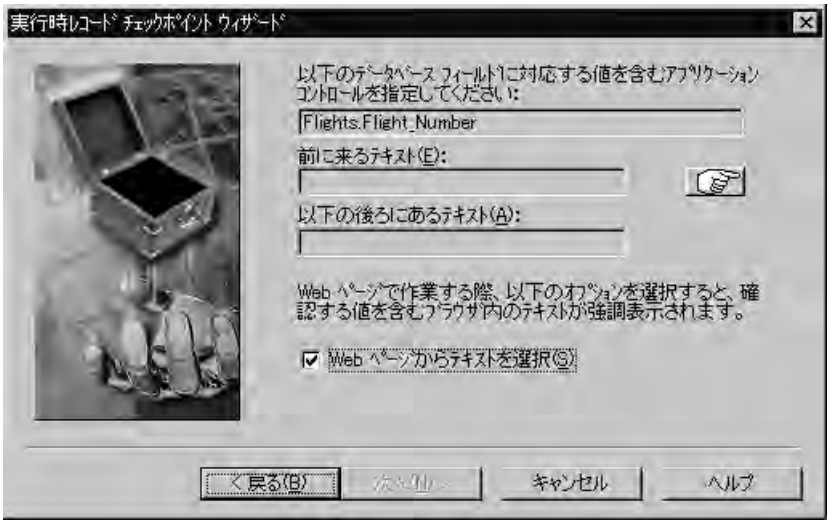


➤ **[前に来るテキスト]**：検査するテキストのすぐ前に現れるテキストが表示されます。

（[**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。

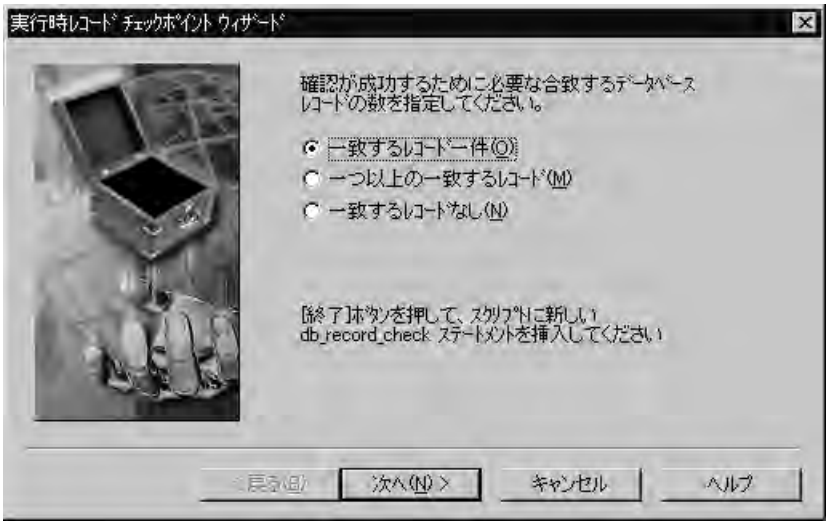
注：

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

6 272 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，279 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。

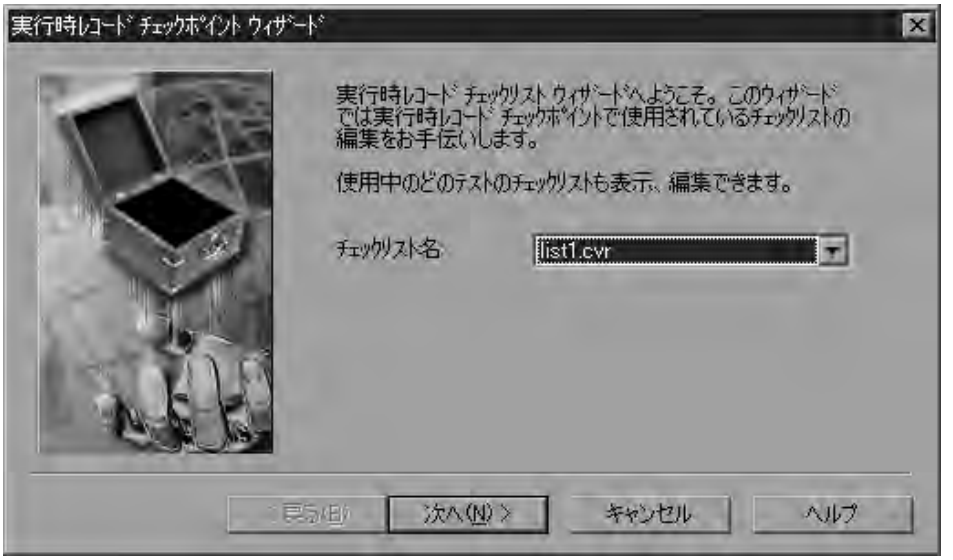
実行時データベース・レコード・チェックリストの編集では，次のことができます。

- 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。

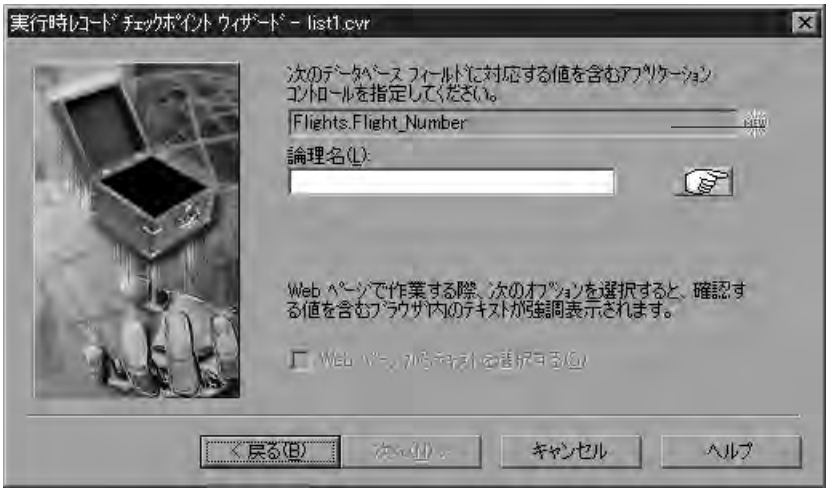
この画面上では，次のことが行えます。

- 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，275 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，289 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，293 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，294 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 323 ページ「ODBC/Microsoft Query でのクエリの作成」か 297 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。322 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，293 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，289 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。

2 293 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，323 ページ「ODBC/Microsoft Query でのクエリの作成」か 297 ページ「SQL ステートメントを指定」を参照してください。

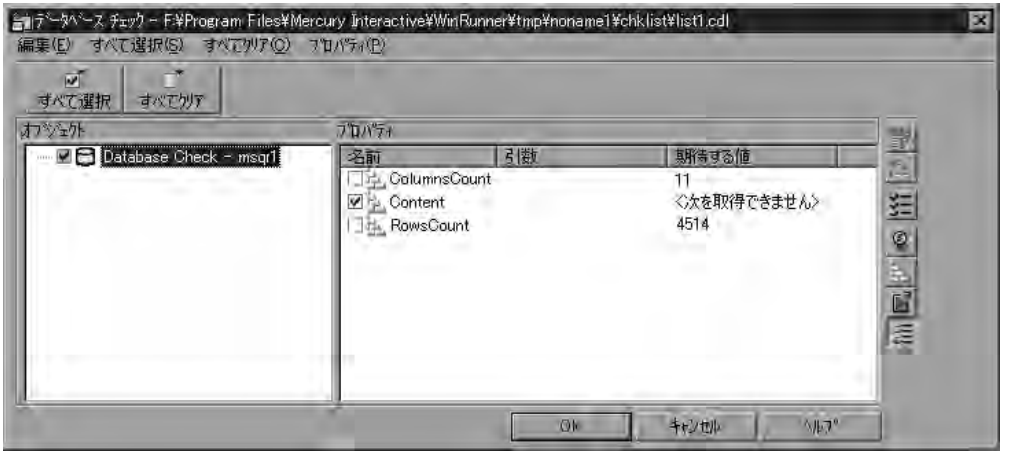
Data Junction を使用する場合は，新しい変換ファイルを作成するか，既存のファイルを使用します。詳細については，324 ページ「Data Junction での変換ファイルの作成」を参照してください。

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。322 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：286 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，300 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，289 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，296 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，297 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，289 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，319 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，289 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，324 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，299 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，299 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，289 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，289 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，303 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，304 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，302 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。

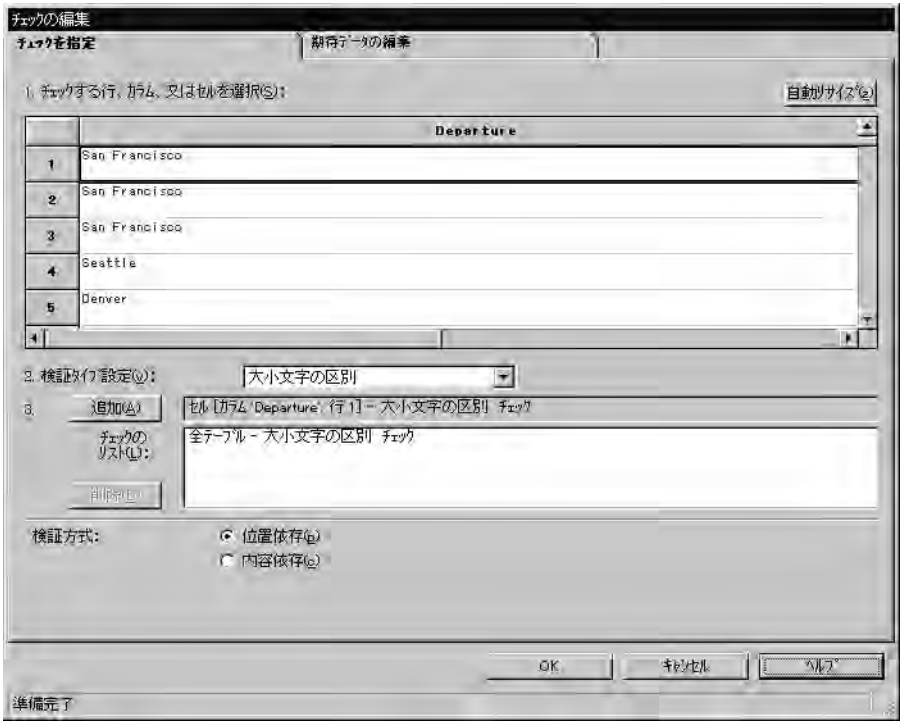
行

- ［**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。

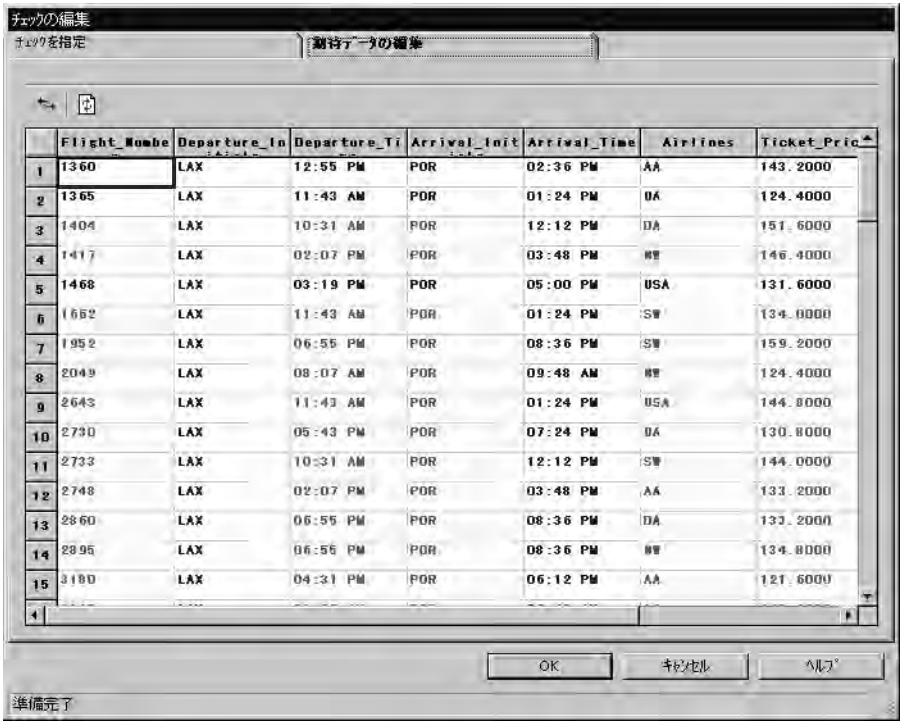
［**OK**］をクリックして，［**チェックの編集**］ダイアログ・ボックスの両方のタブで行った変更を保存します。ダイアログ・ボックスが閉じ，［**データベースチェック**］ダイアログ・ボックスに戻ります。

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。



	Flight_Numbe	Departure_In	Departure_Ti	Arrival_Init	Arrival_Time	Airlines	Ticket_Pric
1	1360	LAX	12:55 PM	POR	02:36 PM	AA	143.2000
2	1365	LAX	11:43 AM	POR	01:24 PM	UA	124.4000
3	1404	LAX	10:31 AM	POR	12:12 PM	UA	151.6000
4	1417	LAX	02:07 PM	POR	03:48 PM	NW	146.4000
5	1468	LAX	03:19 PM	POR	05:00 PM	USA	131.6000
6	1652	LAX	11:43 AM	POR	01:24 PM	SW	134.0000
7	1952	LAX	06:55 PM	POR	08:36 PM	SW	159.2000
8	2049	LAX	08:07 AM	POR	09:48 AM	NW	124.4000
9	2643	LAX	11:43 AM	POR	01:24 PM	USA	144.8000
10	2730	LAX	05:43 PM	POR	07:24 PM	UA	130.8000
11	2733	LAX	10:31 AM	POR	12:12 PM	SW	144.0000
12	2748	LAX	02:07 PM	POR	03:48 PM	AA	133.2000
13	2860	LAX	06:55 PM	POR	08:36 PM	UA	133.2000
14	2895	LAX	06:55 PM	POR	08:36 PM	NW	134.8000
15	3190	LAX	04:31 PM	POR	06:12 PM	AA	121.6000

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，317 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。

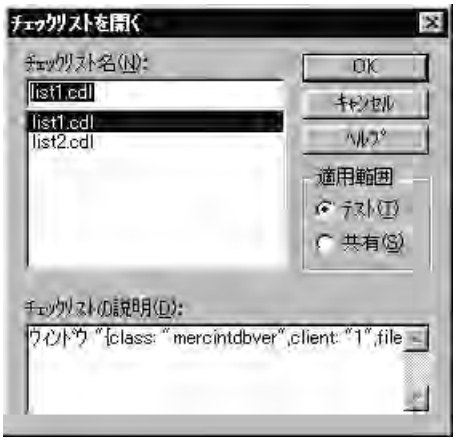
注：**.sql* ファイルは共有データベース・チェックリスト・フォルダには保存されません。

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，145 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。

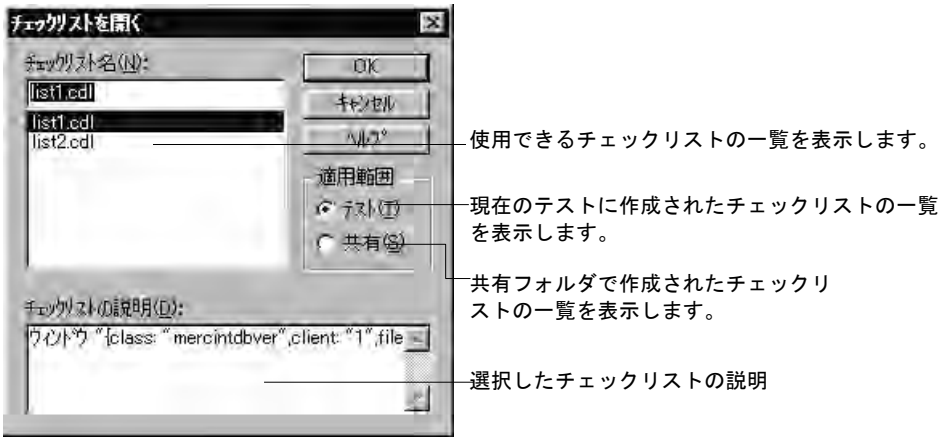
データベース・チェックリストを編集して，検査するデータベース内のプロパティを変更したほうがよいでしょう。

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，307 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。

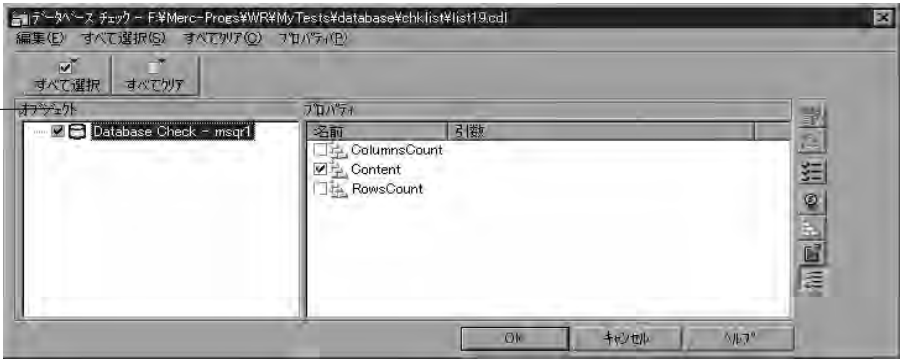


3 データベース・チェックリストを選択します。

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。



***.sql クエリ・ファイルまたは *.djs 変換ファイルの名前**

［**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，311 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，286 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，425 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

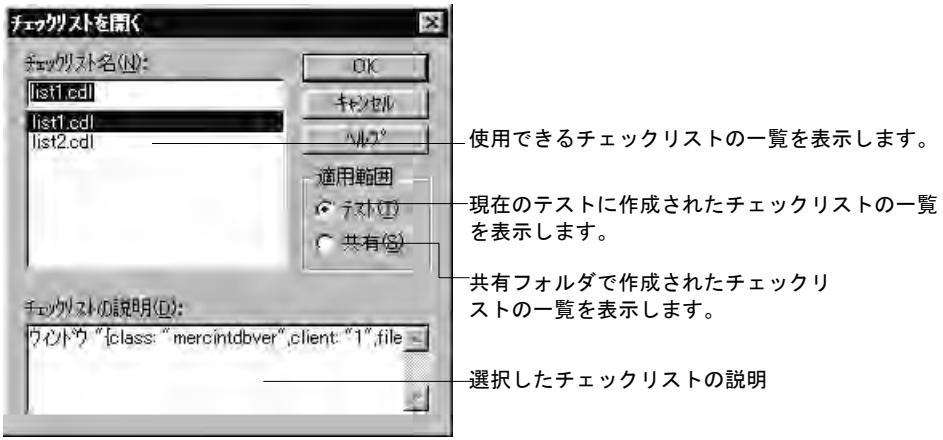
［データベース チェックリストを編集］ダイアログ・ボックスでは，ODBC/Microsoft Query で作成したクエリを変更できます。

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，307 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。

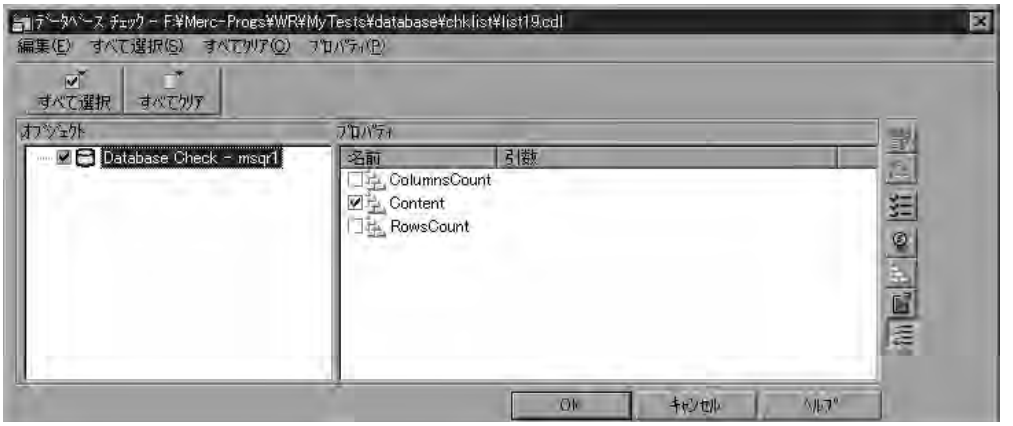
4 **［OK］**をクリックします。

［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

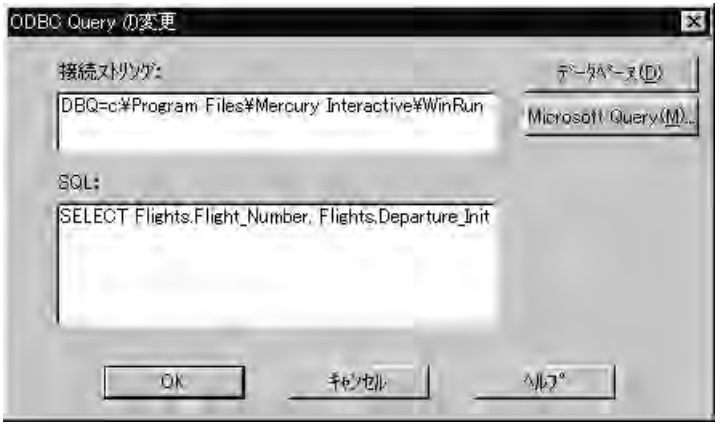
更の詳細については，309 ページ「データベース・チェックリストの編集」を参照してください。



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，435 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

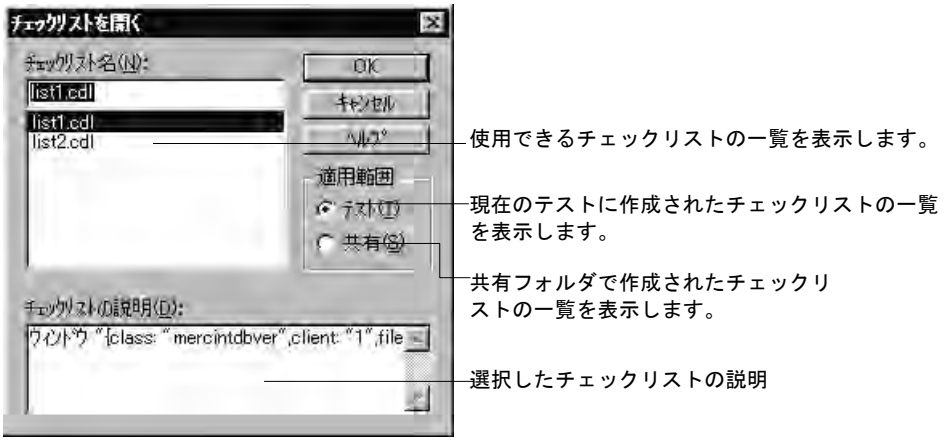
データベース・チェックポイントで使う Data Junction 変換ファイルは，Data Junction で直接変更できます。変換ファイルのパス名を表示するには，以下の手順に従ってください。

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，307 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，309 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，435 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。

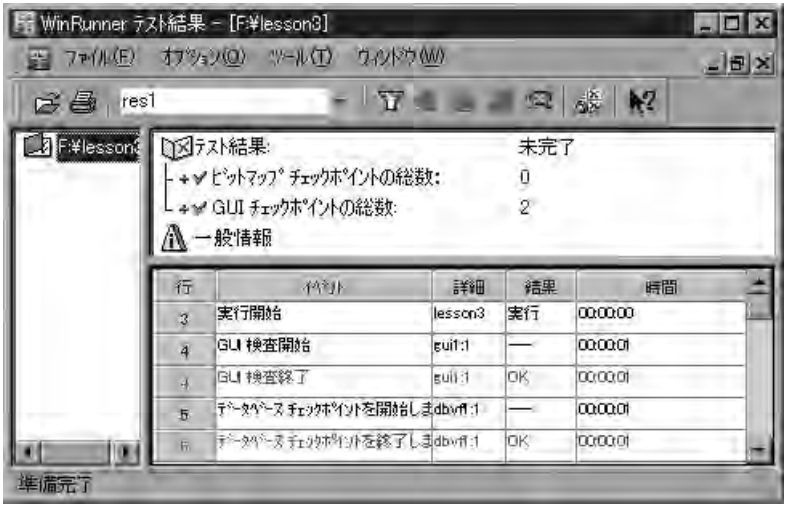
標準のデータベース・チェックポイントの期待結果の変更

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，289 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 489 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，297 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，289 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，293 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，319 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，297 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。322 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，289 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，291 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。299 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，289 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，291 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『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 8 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，286 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 289 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，319 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，272 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。

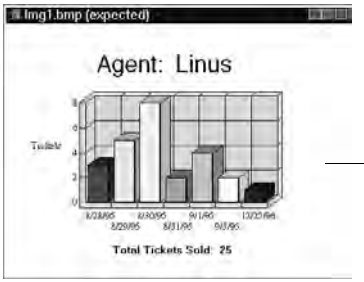
本章では，以下の項目について説明します。

- 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

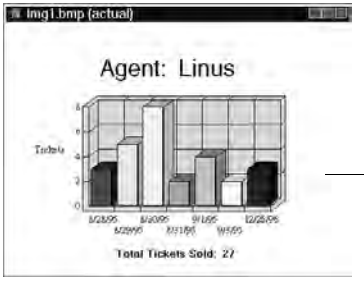
ビットマップの検査について

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，363 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，564 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，548 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，352 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，544 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，546 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，352 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，352 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 8 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* **[** *,x1,y1,x2,y2* **] [** *,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 ***x1***，***y1***，***x2***，***y2*** 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。

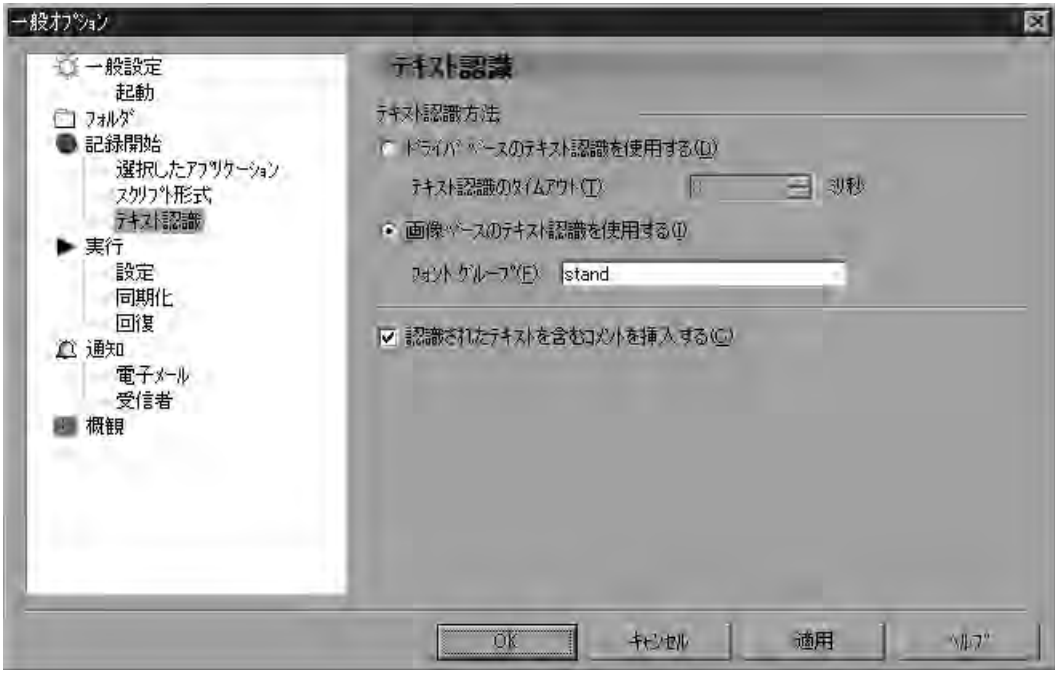
学習済みのフォントを対象としたテストの実行

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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の 第 21 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『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 8 章「関数の生成」を参照してください。

button_check_state	**list_check_selected**
edit_check_selection	**scroll_check_pos**
edit_check_text	**static_check_text**
list_check_item	

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，393 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。

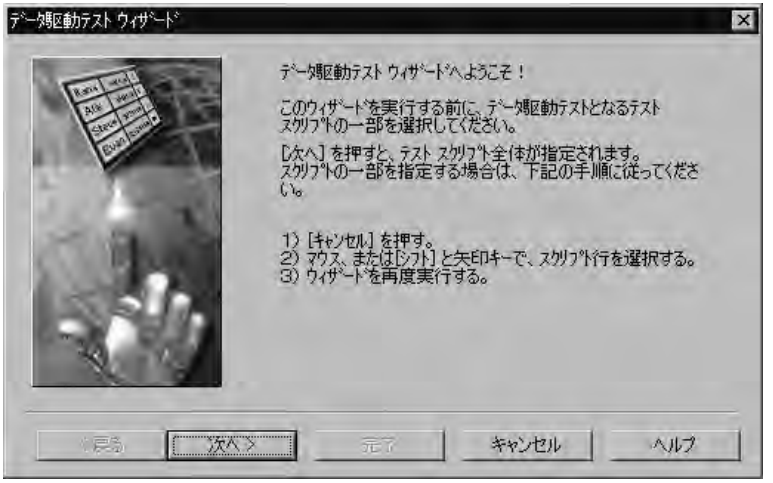
データ駆動型テストを作るには，次の手順を実行します。

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，365 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，373 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，374 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，374 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 10 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，374 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。

361 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。

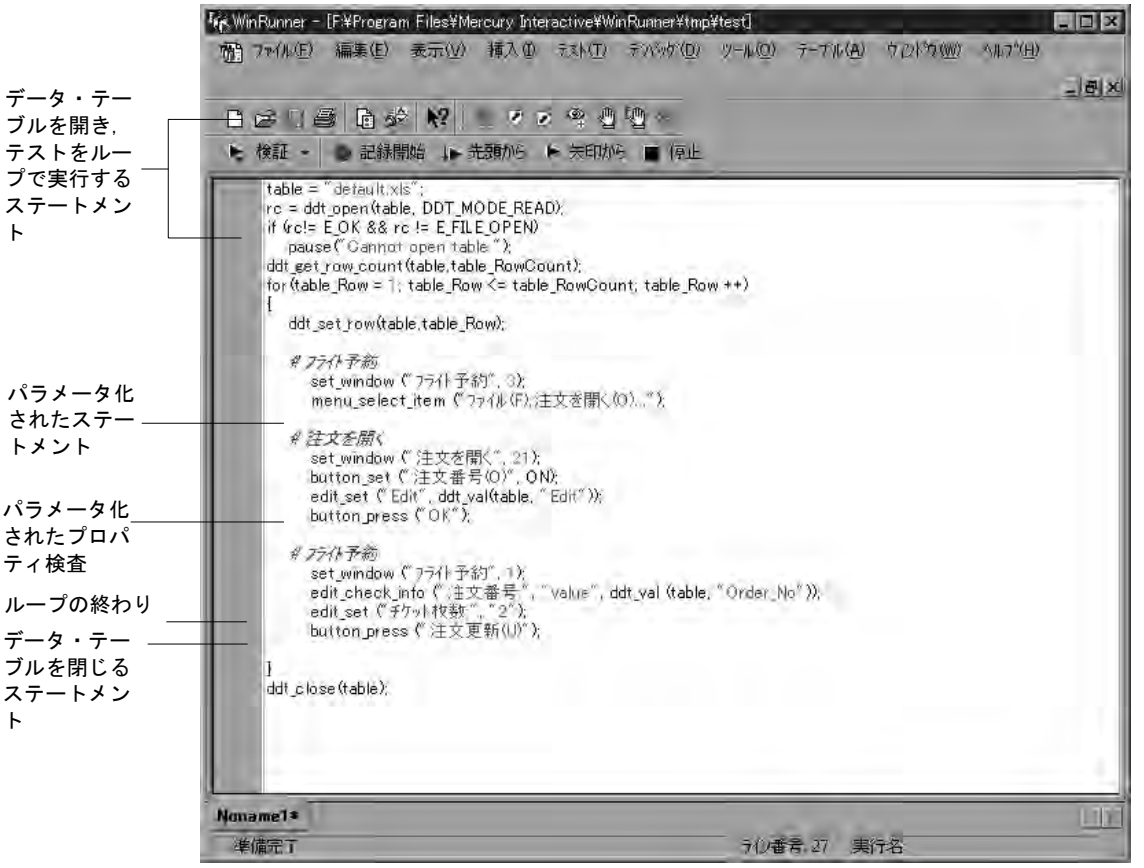
➤ ウィザードを閉じた後にデータ・テーブルを開く場合は，［**データ テーブルを表示する**］を選びます。

➤ 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。

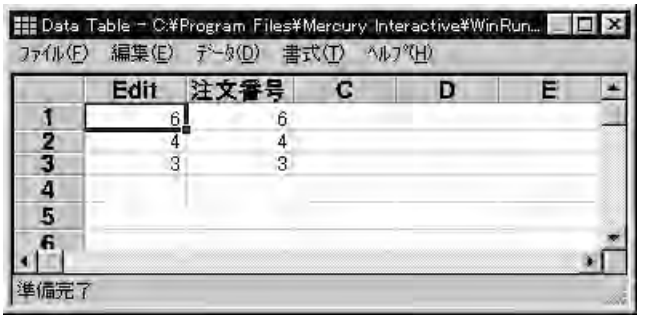
注 : テスト・スクリプトをパラメータ化した後，最後のウィザード画面が表示される前に［**キャンセル**］をクリックすると，データ・テーブルには追加したデータが含まれます。データ・テーブルにデータを保存する場合は，データ・テーブルを開いて，保存します。

データ駆動テスト・ウィザードの実行が終了すると，361 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，404 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，374 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，398 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

361 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);
```

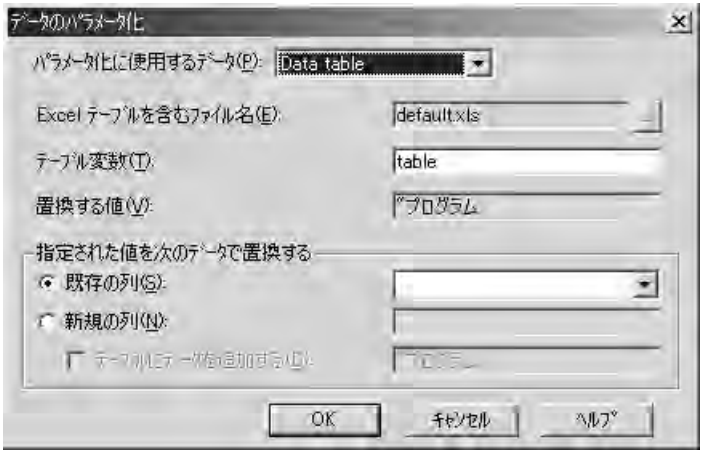
［**データのパラメータ化**］ダイアログ・ボックスを使用して，ステートメントをパラメータ化して，パラメータでデータを置き換えます。

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，398 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

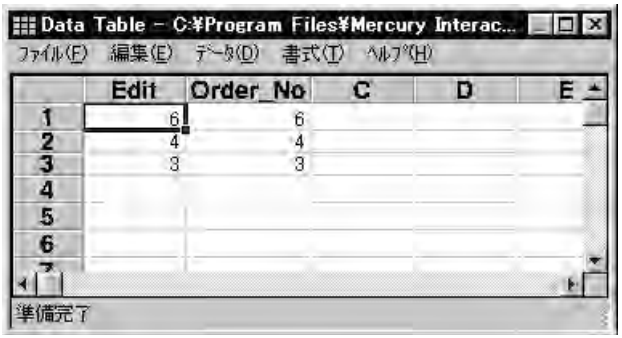
各データ駆動型テストに，少なくとも 1 つはデータ・テーブルを用意しなければなりません。データ・テーブルには，WinRunner がデータ駆動型テストの変数を置換するのに使う値が含まれます。

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，378 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，398 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，385 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。

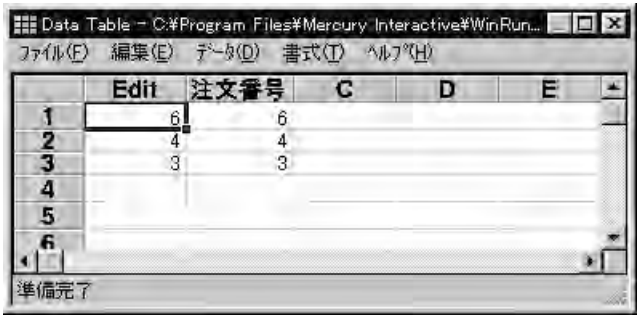
注 : データの形式を変更（例えば日付の変更など）したくない場合は，データ・テーブルに入力する文字列は引用符（'）で始めなくてはなりません。これにより，セル内の文字列の形式を変更しないようエディタに指示します。

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，387 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，388 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，389 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，368 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 368 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，368 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，374 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，398 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，403 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，361 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 ");
```

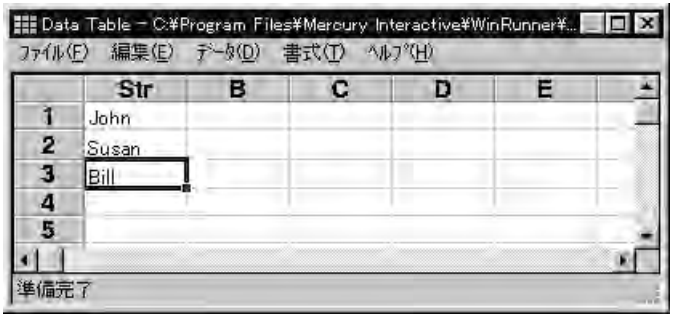
2 データ駆動テスト・ウィザード（［**テーブル**］＞［**データ駆動テスト ウィザード**］）を使ってこのスクリプトをデータ駆動テストにし，テスト・スクリプト内のステートメントのデータの値をパラメータ化します。詳細については，364 ページ「データ駆動テスト・ウィザードを使ったデータ駆動型テストの作成」を参照してください。また，テスト・スクリプトへのこうした変更は手作業でも行えます。詳細については，372 ページ「手作業によるデータ駆動テストの作成」を参照してください。

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。

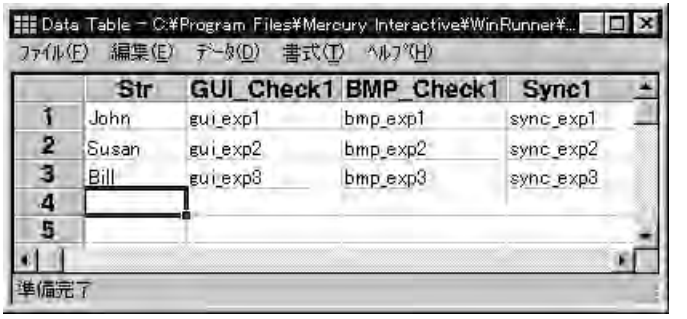


データ駆動型テストの GUI チェックポイントとビットマップ・チェックポイントおよび同期化ポイントは，テスト実行の 2 番目と 3 番目の繰り返しで失敗し

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，374 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 8 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，378 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，398 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，398 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，393 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 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 8 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の 第 6 章「正規表現の使い方」を参照してください。

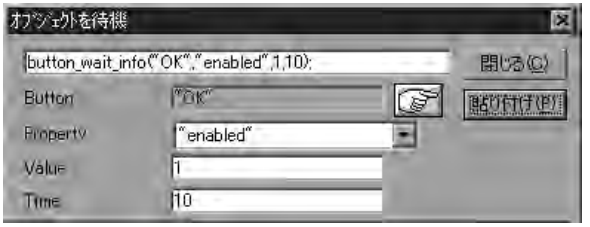
記録の際，アクティブ・ウィンドウ以外のウィンドウのオブジェクトをキャプチャすると，WinRunner は **set_window** ステートメントを自動的に生成します。

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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 8 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，112 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，393 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 14 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，432 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，427 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，**『Mercury WinRunner 上級機能ユーザーズ・ガイド』**の第 16 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，**『Mercury WinRunner 上級機能ユーザーズ・ガイド』**の第 17 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，**『Mercury WinRunner 上級機能ユーザーズ・ガイド』**の第 18 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。**『Mercury WinRunner 上級機能ユーザーズ・ガイド』**の第 9 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，**『Mercury WinRunner 上級機能ユーザーズ・ガイド』**の第 9 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，434 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**『Mercury WinRunner 上級機能ユーザーズ・ガイド』**の第 21 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，435 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。

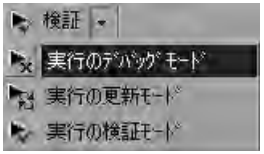
テスト・スクリプトを更新モードで実行する方法の詳細については，435 ページ「期待結果を更新するためのテスト実行」を参照してください。

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

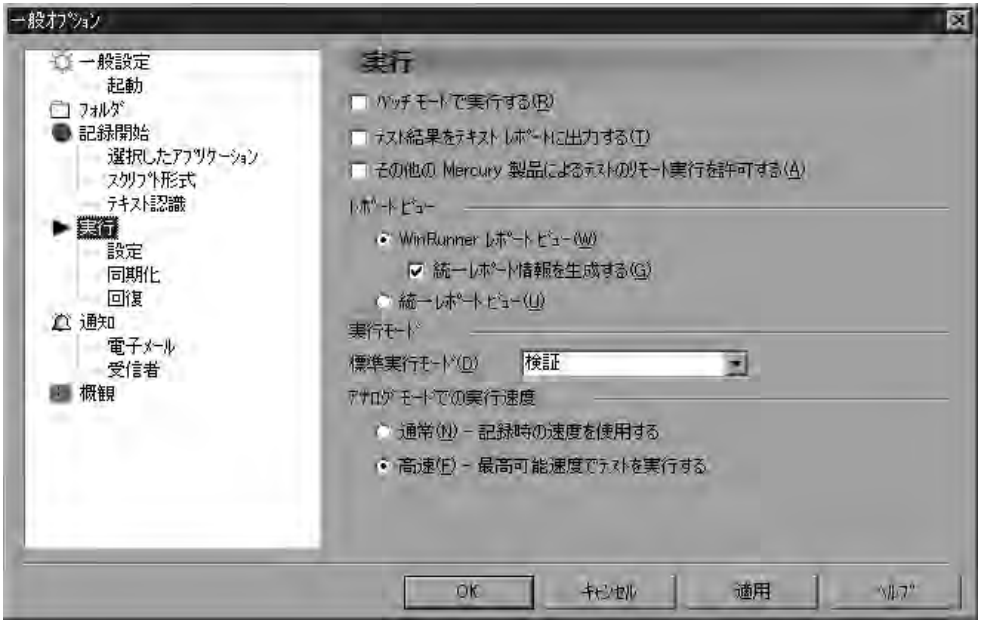
［一般オプション］ダイアログ・ボックスの［実行］カテゴリで標準設定の実行モードを設定できます。ここで設定されるモードによって，テストを開くときのモードが決定されます。

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。



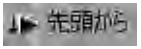
3 ［**標準実行モード**］ボックスで実行モードを選択します。

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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 16 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，439 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 11 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，439 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 20 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。

コマンド	標準ソフトキー組み合わせ	機能
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。

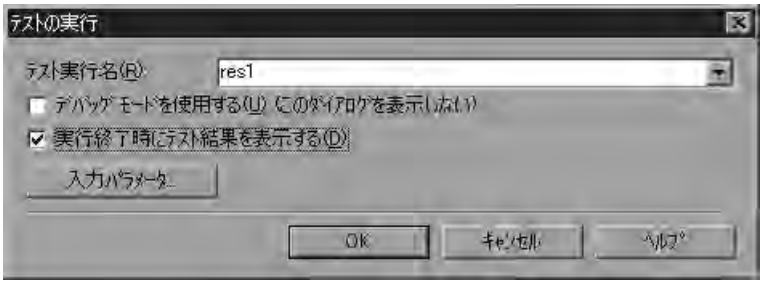
アプリケーションを検査するためにテストを実行するには，次の手順を実行します。

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，438 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

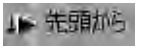
テスト・スクリプトをデバッグするためにテストを実行すると，WinRunner は現在の結果と期待結果を比較します。あらゆる違いがデバッグ結果フォルダに保存されます。WinRunner はデバッグ・モードでテストを実行するたびに，以前のデバッグ結果を上書きします。

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

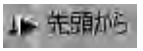
期待結果を更新するためにテストを実行すると，以前に作成した期待結果が，新しい結果に置き換えられ，以降のテスト実行における比較の基準になります。

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**］

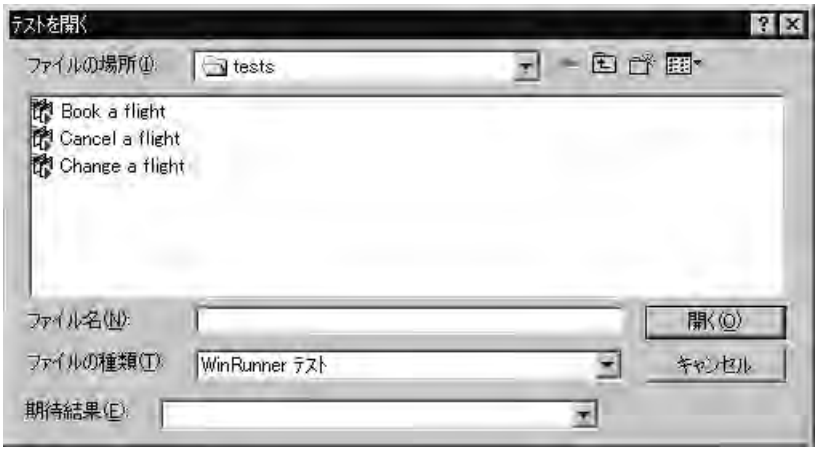
WinRunner は選択された［**テスト**］メニュー・コマンドに従ってテストを実行し，期待結果を更新します。期待結果の標準のフォルダは「*exp*」です。

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。

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。

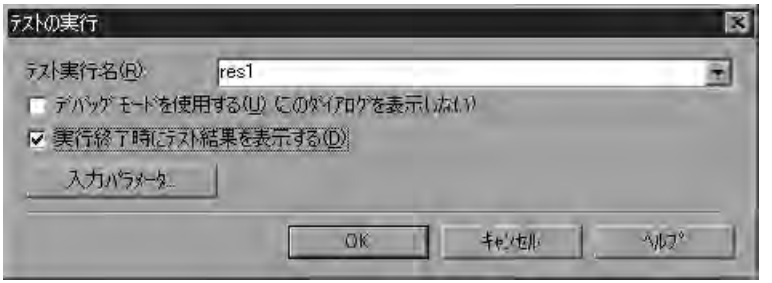
3［**期待結果**］ボックスのリストから，必要な期待結果セットを選択して，［**開く**］をクリックします。［テストを開く］ダイアログ・ボックスが閉じます。

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，438 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，505 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『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 9 章「テストの呼び出し」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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 6 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 2 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，79 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，79 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。

考えられる原因	解決策
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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 2 章「GUI マップの構成設定」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，65 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，425 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，467 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，

『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 25 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，464 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，453 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，454 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，452 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，451 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，456 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，456 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。

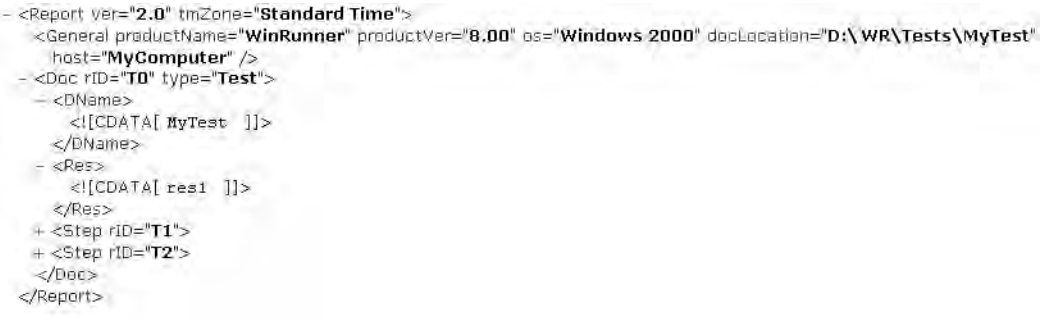
4［**プレビュー**］をクリックし，画面にテスト結果のプレビューを表示します。

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。



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，570 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，464 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，483 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。

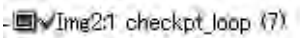
テスト・サマリ

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，483 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，473 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，444 ページ「テスト結果の分析について」および 548 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，467 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 26 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。

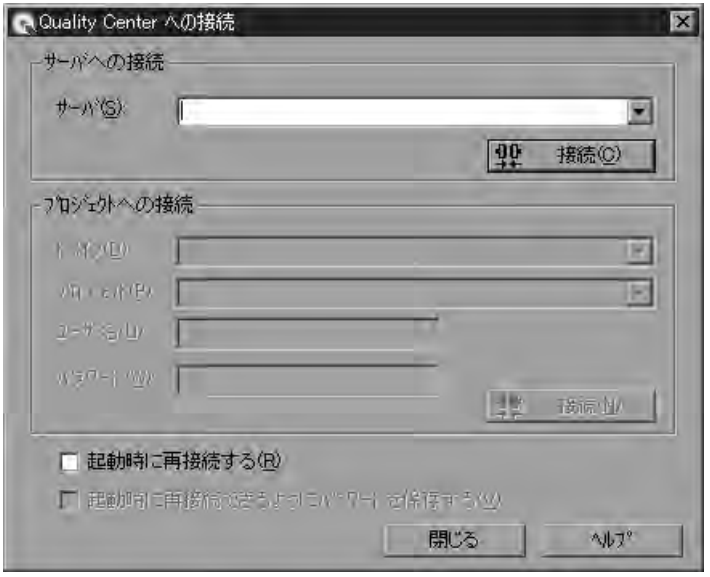
接続プロセスには 2 つの段階があります。まず，WinRunner 統一レポートからローカルまたはリモートの Quality Center Web サーバに接続します。このサーバによって，WinRunner と Quality Center プロジェクトの間の接続が処理されます。

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。

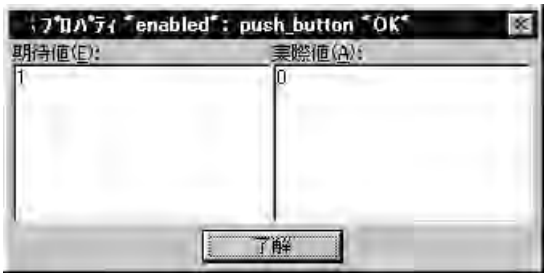
4 ［**OK**］をクリックし，ダイアログ・ボックスを閉じます。

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，489 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，475 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，175 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，169 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。

1 471 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，259 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。

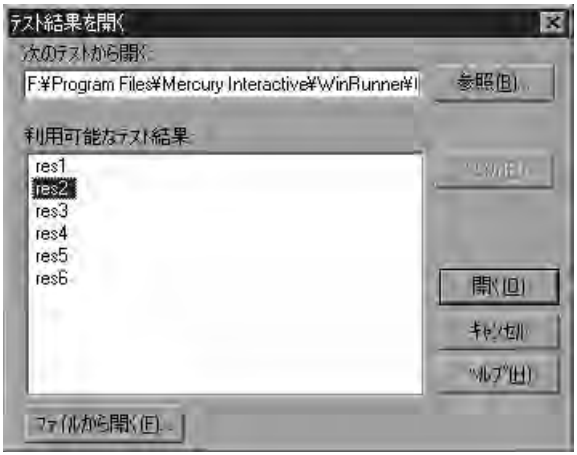
テーブルを対象にする GUI チェックポイントの期待結果は［チェックの編集］ダイアログ・ボックスでも表示できます。詳細については，第 13 章「テーブル内容の検査」を参照してください。

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。

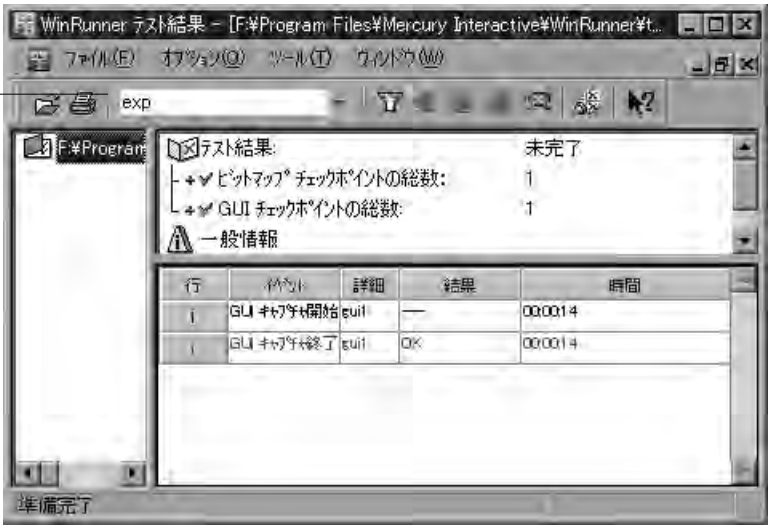
1 ［テスト結果］ウィンドウを開き，期待結果を表示するテストを表示します。詳細については，471 ページ「チェックポイントの結果の表示」を参照してください。

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。



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。

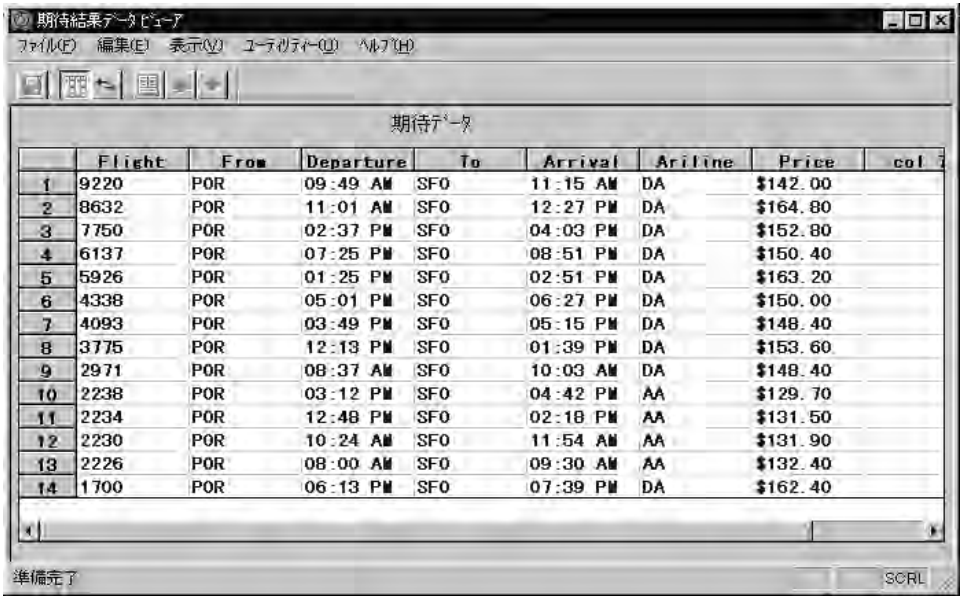
［**GUI 検証結果**］ダイアログ・ボックスが開き，選択した GUI チェックポイントの期待結果が表示されます。

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，259 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。

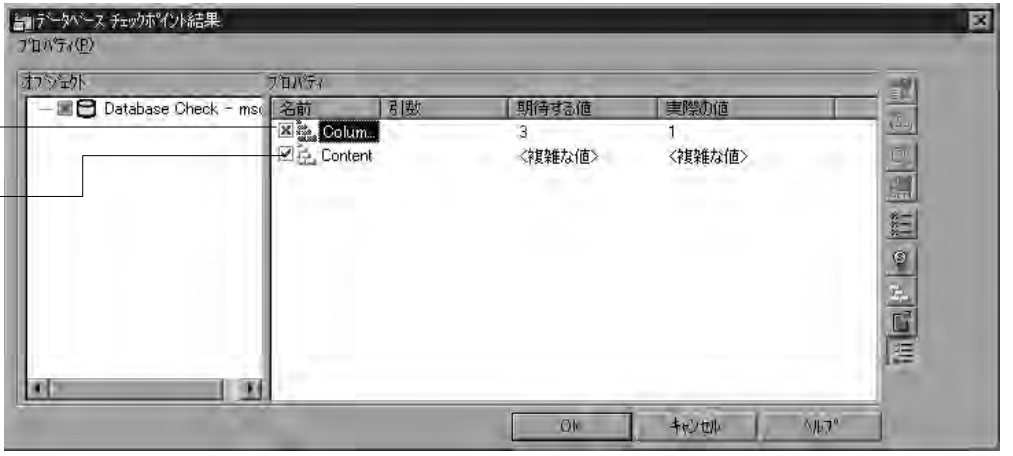
期待 **実際** **差異**

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，334 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，489 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，485 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，300 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，289 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，471 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。

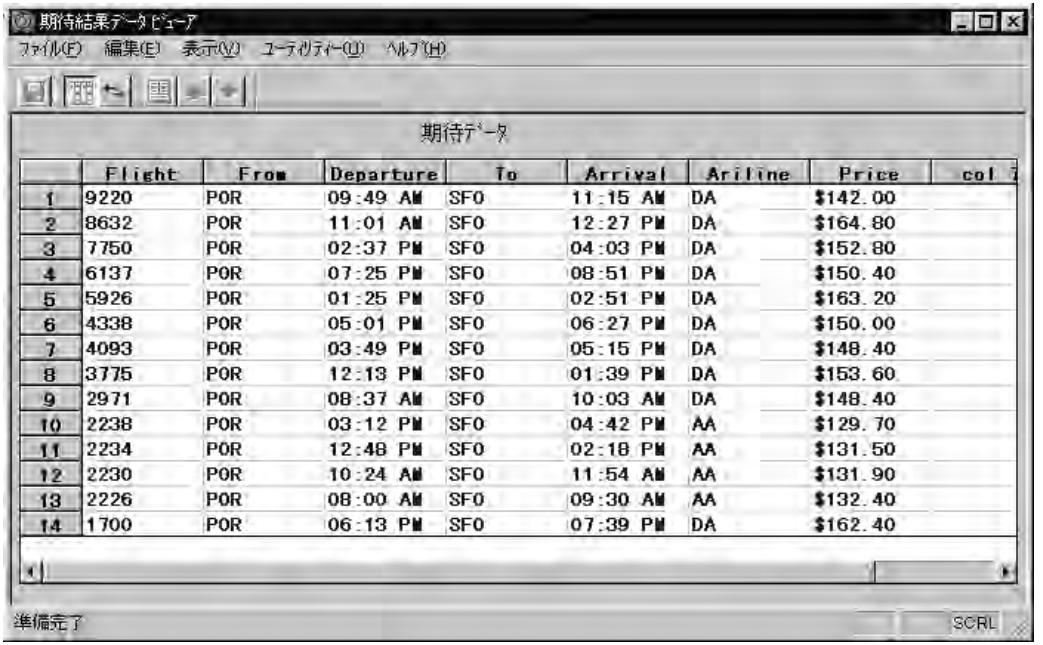
➤ WinRunner レポート・ビューでは，テスト・ログにあるテーブル検査の［**GUI キャプチャ終了**］エントリをダブルクリックします。または，エントリを強調表示して，［**オプション**］＞［**表示**］を選択するか［**表示**］ボタンをクリックします。

［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	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，［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，300 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。

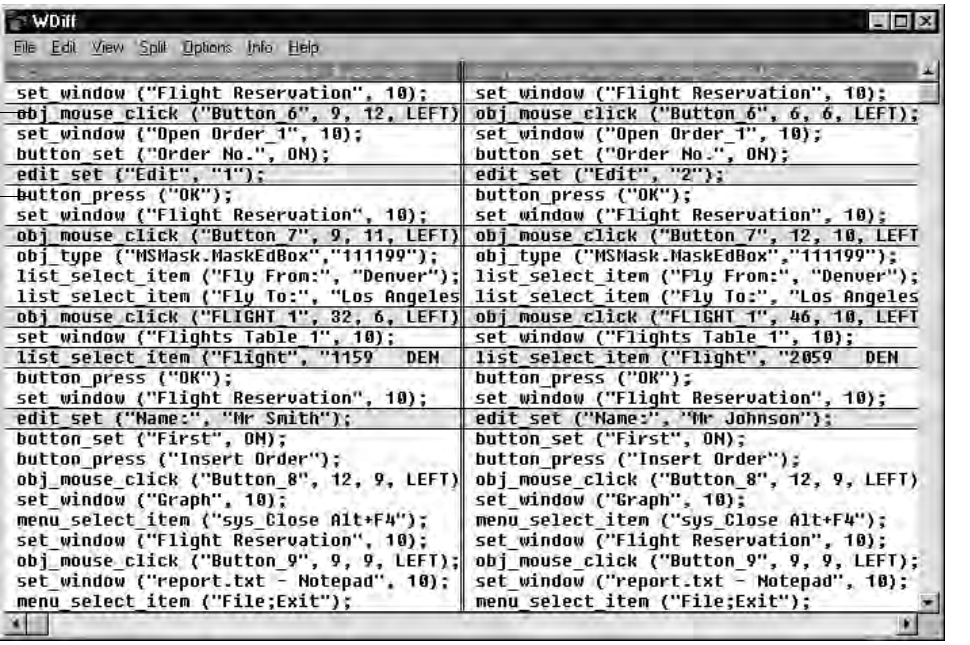
1 ［テスト結果］ウィンドウを開き，ファイル比較結果を表示するテストを表示します。詳細については，471 ページ「チェックポイントの結果の表示」を参照してください。

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。

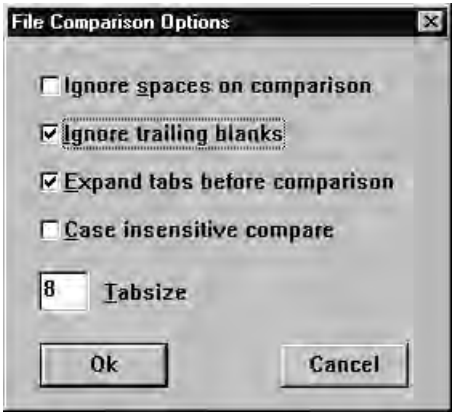
WDiff ユーティリティ・ウィンドウが開きます。



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，469 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，469 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，471 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。469 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。

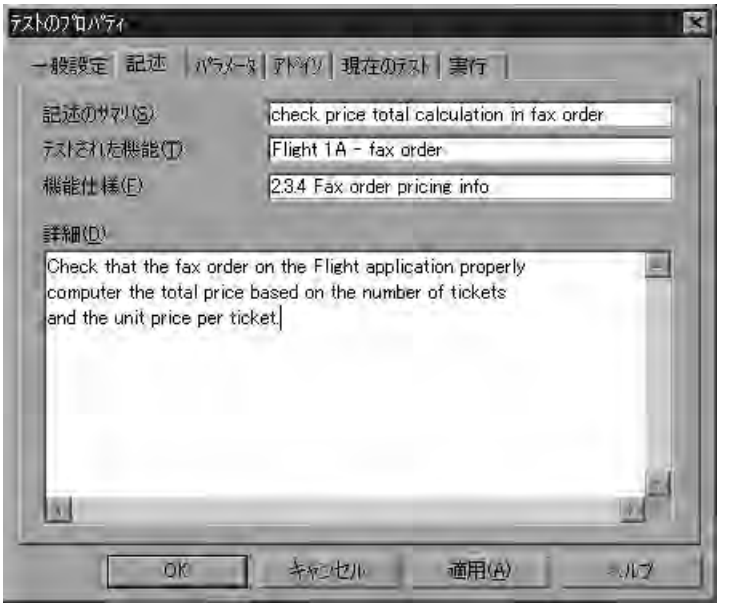
このタブには以下の情報が表示されます。

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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 11 章「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，391 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 9 章「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。

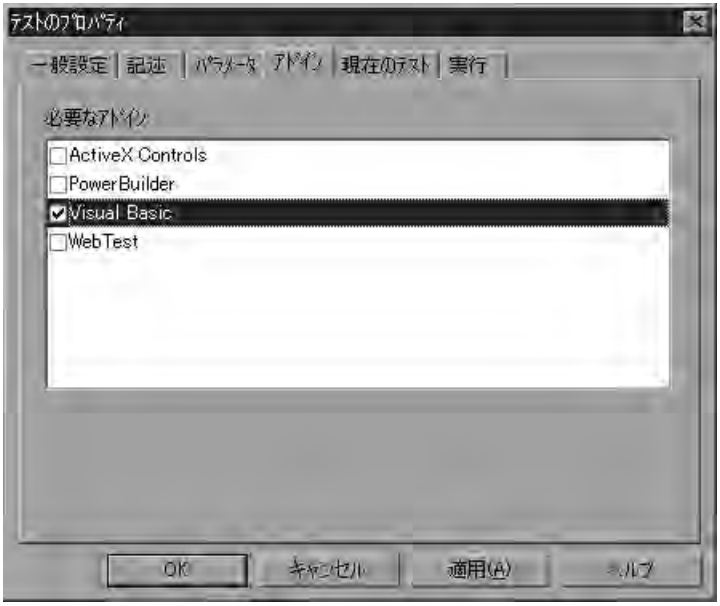
2 ［**パラメータの変更**］ボタン をクリックします。または，パラメータ名をダブルクリックします。［パラメータのプロパティ］ダイアログ・ボックスが開き，パラメータの現在の名前と説明が表示されます。

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，509 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 15 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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]

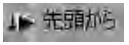
このボックスには，現在の実行モード（検証，デバッグ，または更新）が表示されます。

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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，429 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 9 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 16 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 7 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。
- 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 15 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。
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。

オプション	説明
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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 15 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 8 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 10 章「ユーザ定義関数の作成」および第 11 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 11 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。

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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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。

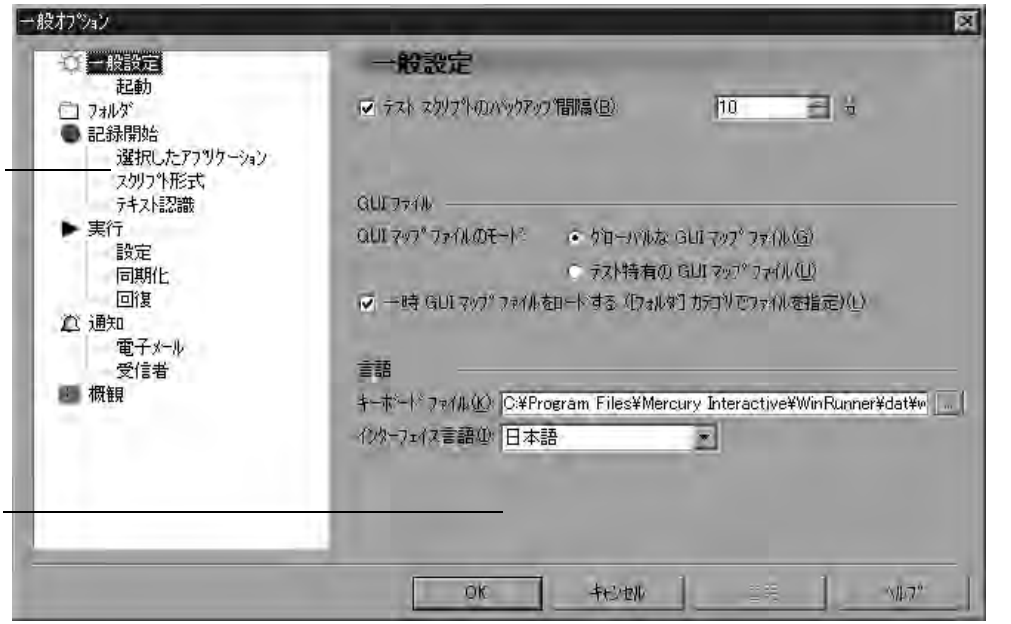
［一般オプション］ダイアログ・ボックスには，次のカテゴリおよびサブカテゴリがあります。

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。

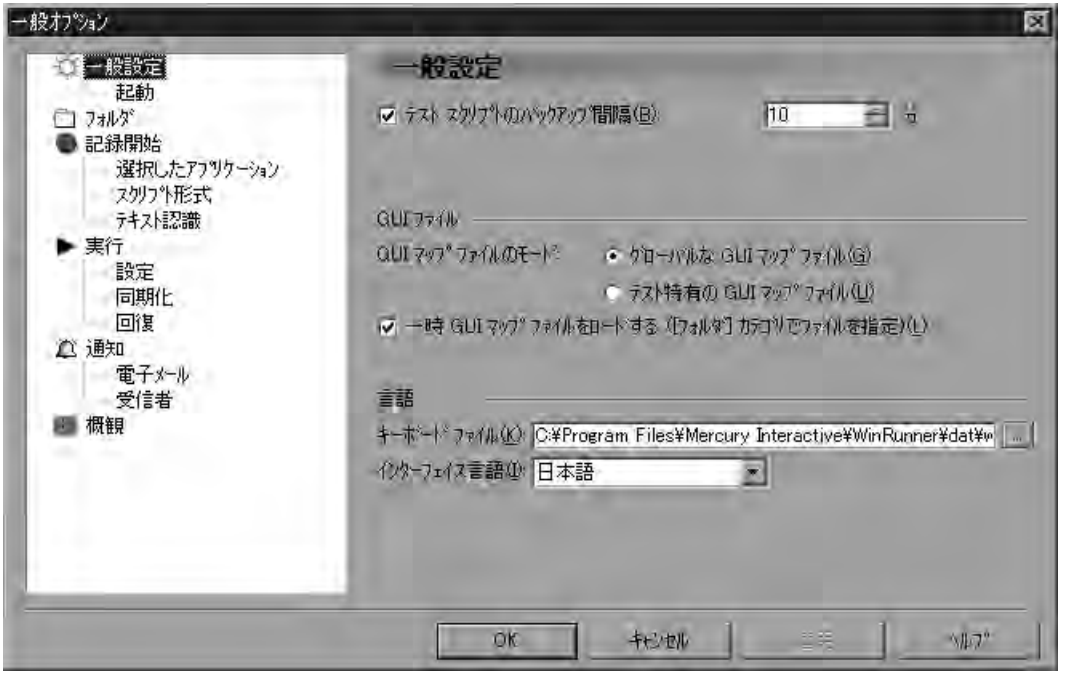
3 以降の項で説明する手順に従って必要なオプションを設定します。

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。

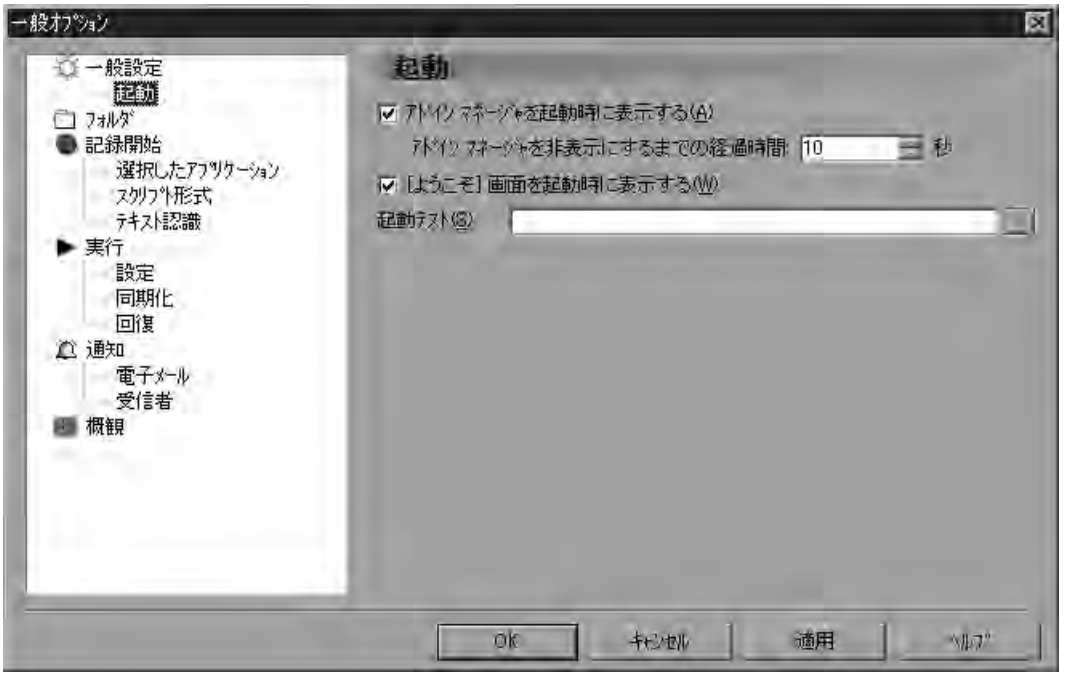
オプション	詳細
［テスト スクリプトのバックアップ間隔］	指定したインターバルで定期的にスクリプトのバックアップ・ファイルを作成するよう指定します。このオプションを選択すると，**script.sav** という名前のテスト・フォルダにバックアップ・ファイルが作成されます。これはスクリプトの単純なテキスト・ファイルです。バックアップを保存するたびに，前の **script.sav** ファイルに上書きされます。 **標準設定** = 選択されている，10（分）
［タイプ保存ダイアログを表示する］	このオプションは，WinRunner が Quality Center サーバに接続されていると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。

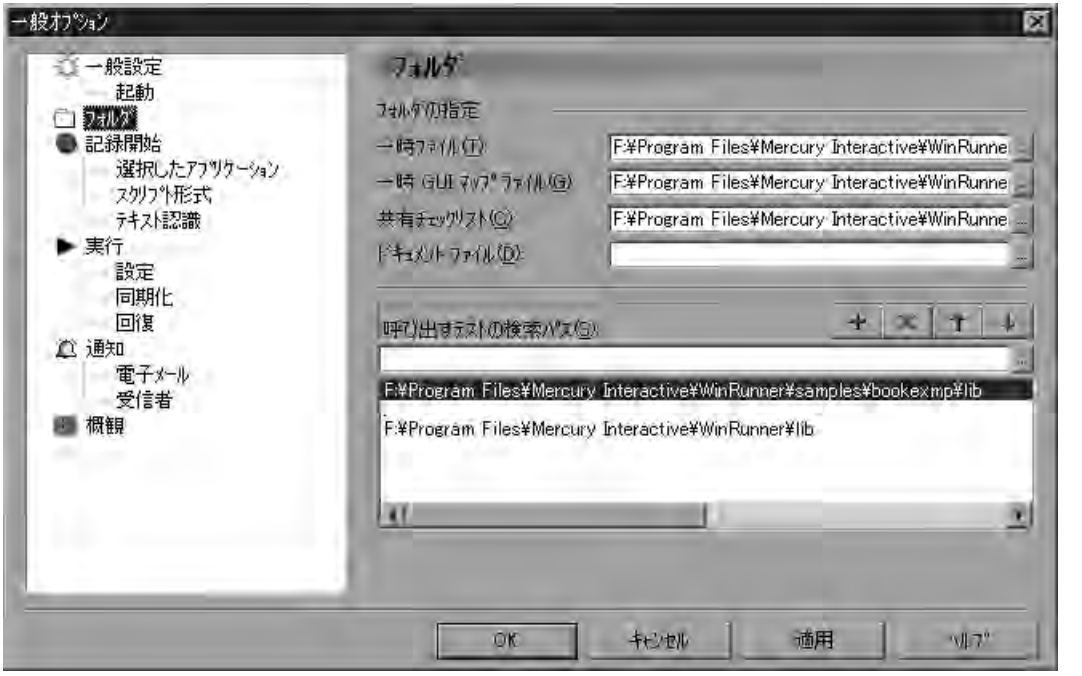


［**起動**］カテゴリには次のオプションがあります。

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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 23 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 26 章「テスト工程の管理」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。

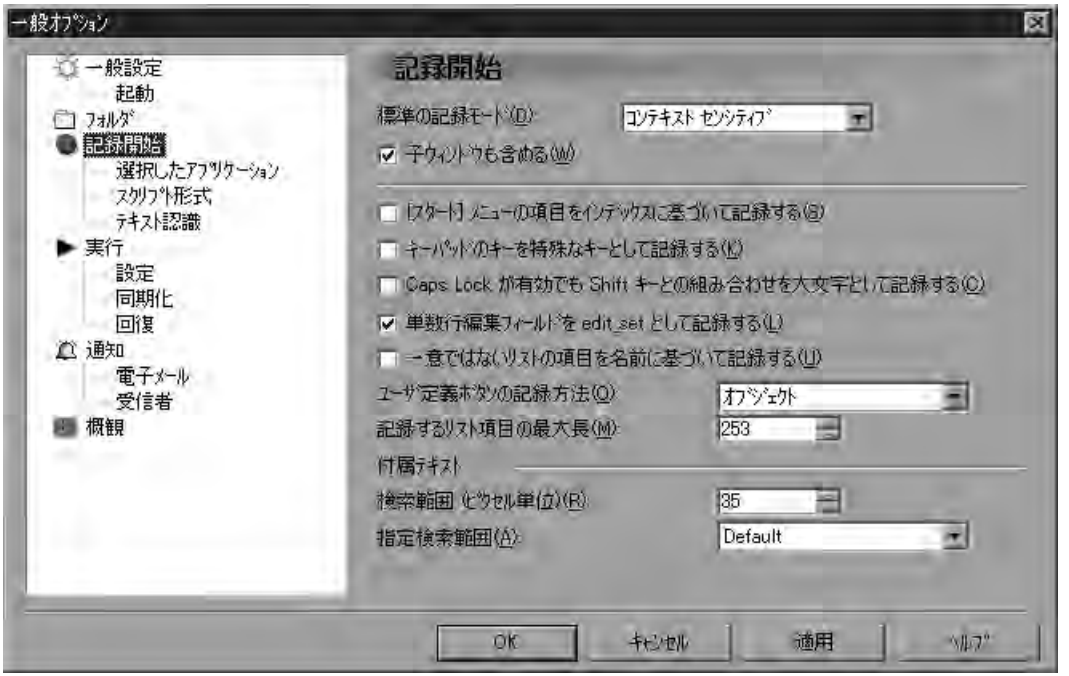
オプション	詳細
［一時ファイル］	一時テストを格納するフォルダ。フォルダを入力または参照します。 **注：** 新しいフォルダを指定した場合は，変更を有効にするために WinRunner を再起動する必要があります。 テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **tempdir** テスト・オプションの値を設定および取得できます。詳細については，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 **標準設定 = ＜ WinRunner のインストール・フォルダ＞ ¥tmp**
［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**

オプション	詳細
[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，145 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，307 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 9 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 15 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，**『Mercury WinRunner 上級機能ユーザーズ・ガイド』**の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 15 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド** .』の第 21 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。

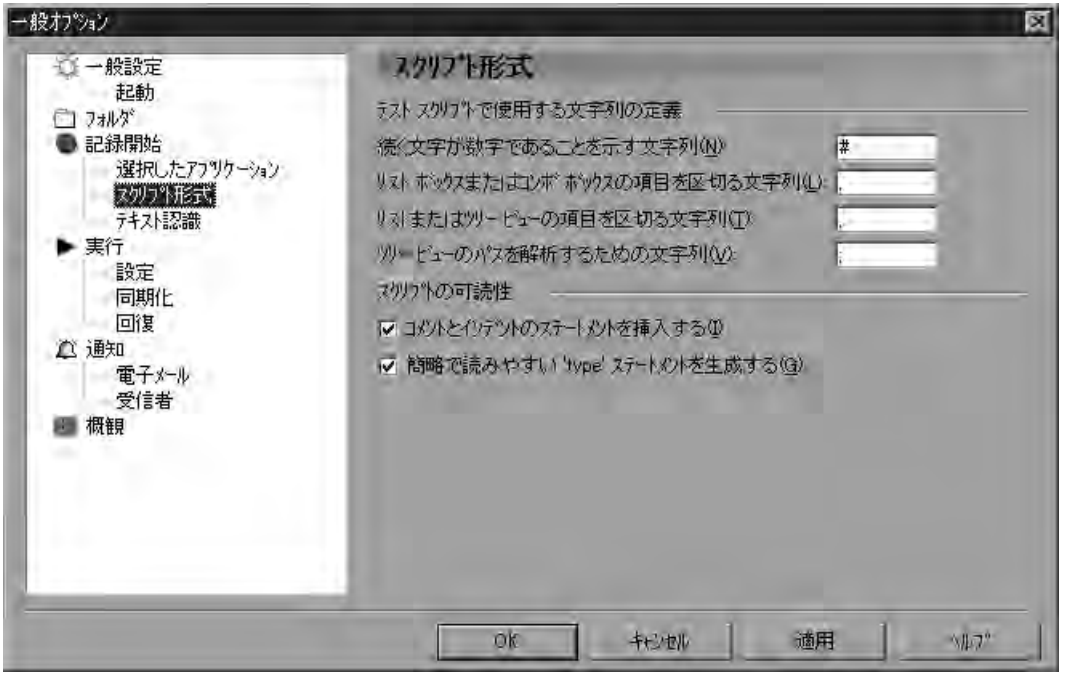
5 Internet Explorer，Netscape，あるいはその両方での動作を記録しない場合は，アプリケーション・リストで，**iexplore.exe**，**netscape.exe**，**netscp6.exe** に対する各オプションをクリアします。

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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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** = ,

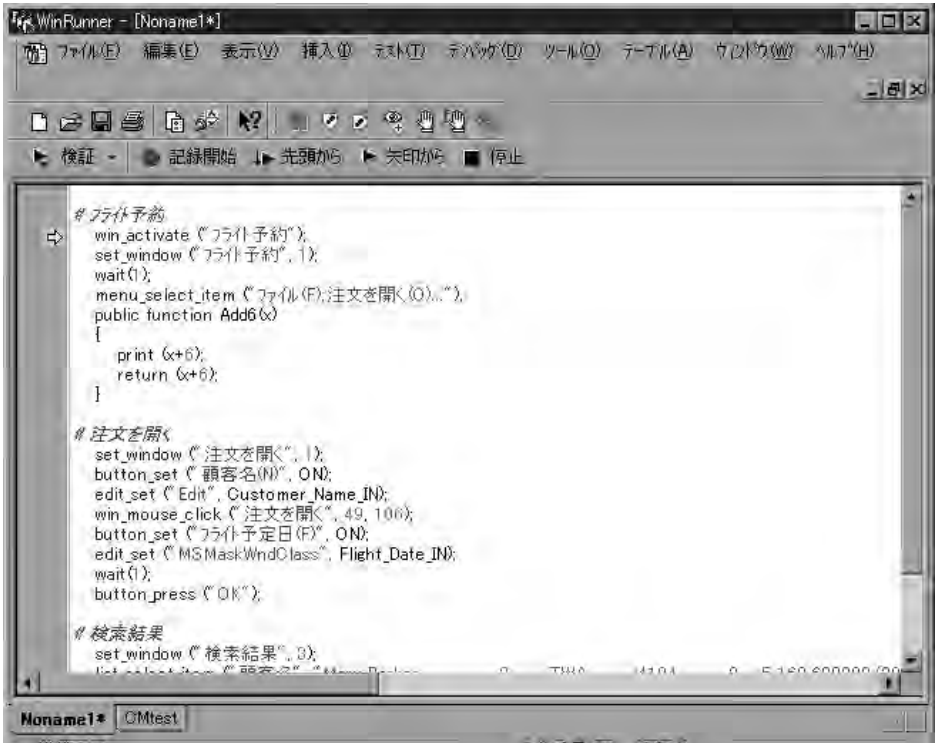
オプション	詳細
[ツリービューのパスを解析するための文字列]	ツリー表示のパスの項目を区切るためにテスト・スクリプトに記録される文字列。 テスト・スクリプトで **setvar** 関数および **getvar** 関数を使用して，このオプションに対応する **treeview_path_separator** テスト・オプションの値を設定および取得できます。詳細については，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 章「テスト・スクリプトからのテスト・オプションの設定」を参照してください。 **標準設定** = ;
[コメントとインデントのステートメントを挿入する]	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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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。

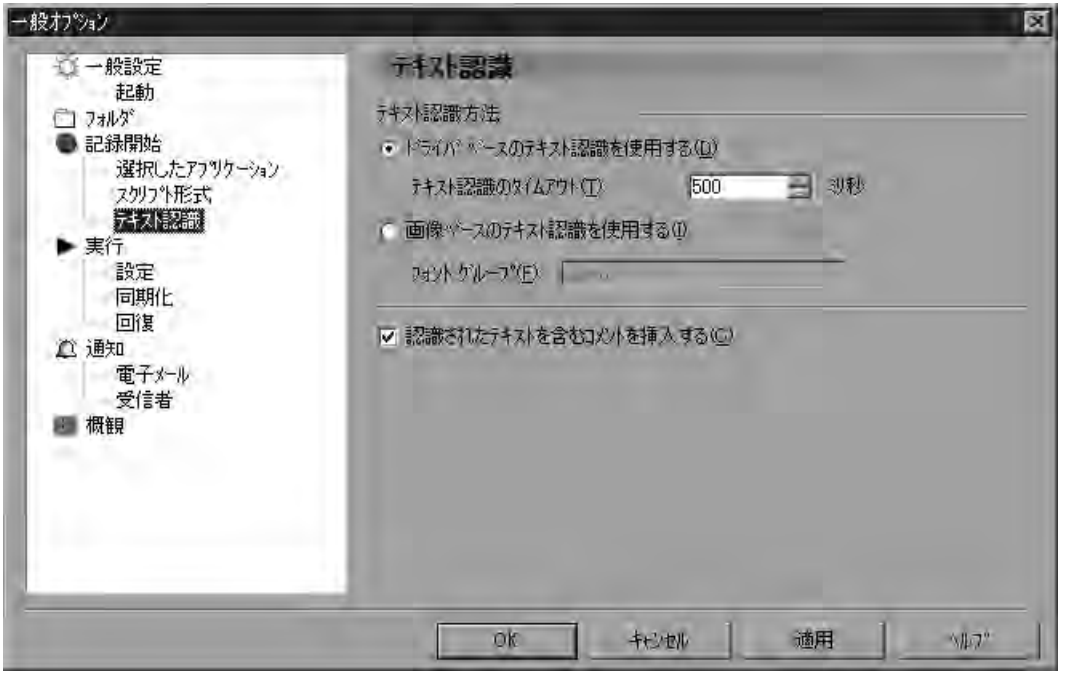


このオプションが選択されている場合，WinRunner は記録中，自動的にテストをセクションに分割します。**set_window** ステートメントも **win_*** ステートメントと同様に，セクションに分割できます。コンテキスト・センシティブからアナログに記録モードを切り換えると新しいセクションが始まります。

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，574 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，352 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 15 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。

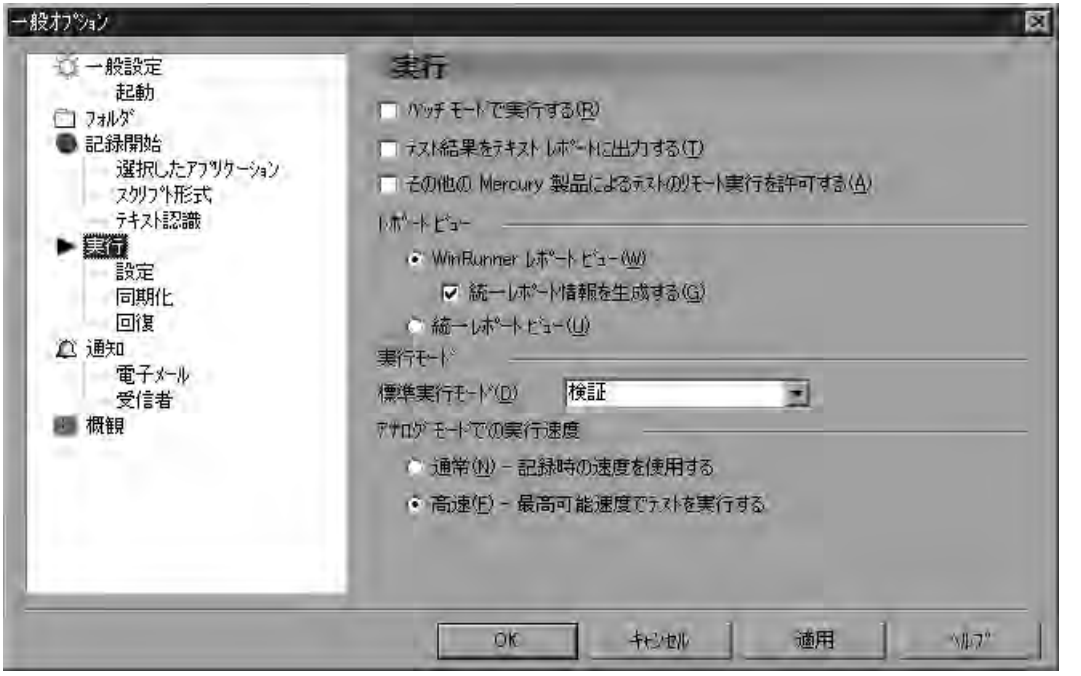
したがって，アプリケーション・ウィンドウからテキストを取得または確認する場合は，可能な限り，標準の GUI チェックポイントを挿入し，オブジェクトの **value**（または同様の）プロパティを調べることを選択することをお勧めします。例えば，次の手順を実行します。

- ［**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。

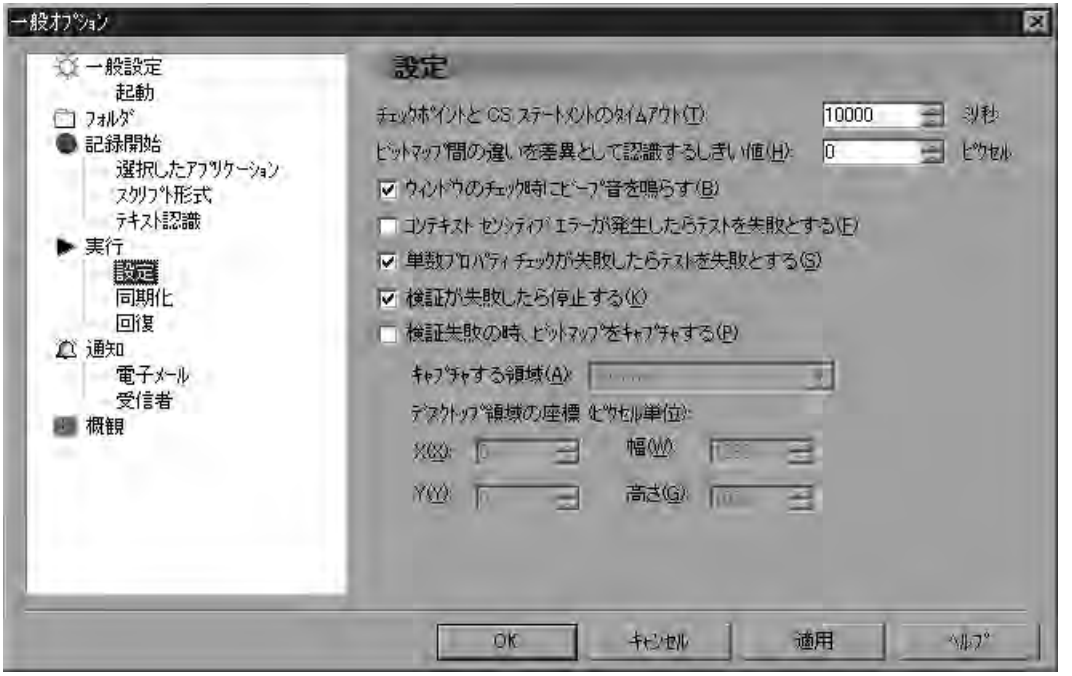
オプション	詳細
［バッチ モードで実行する］	テストを無人で実行できるようにするために，「**検証**」テストの実行中にメッセージを抑制するかどうかを指定します。 例えば，**set_window** ステートメントがテスト・スクリプトにないと，WinRunner は指定されたウィンドウを見つけることができません。テストをバッチ・モードで実行した場合，WinRunner は［テスト結果］ウィンドウにエラーを報告し，テスト・スクリプト内の次のステートメントに進みます。テストをバッチ・モードで実行しない場合，WinRunner はテストを一時停止し，［実行］ウィザードを開いてユーザがそのウィンド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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 14 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 15 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 15 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，425 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 15 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，574 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 15 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 15 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 15 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 15 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

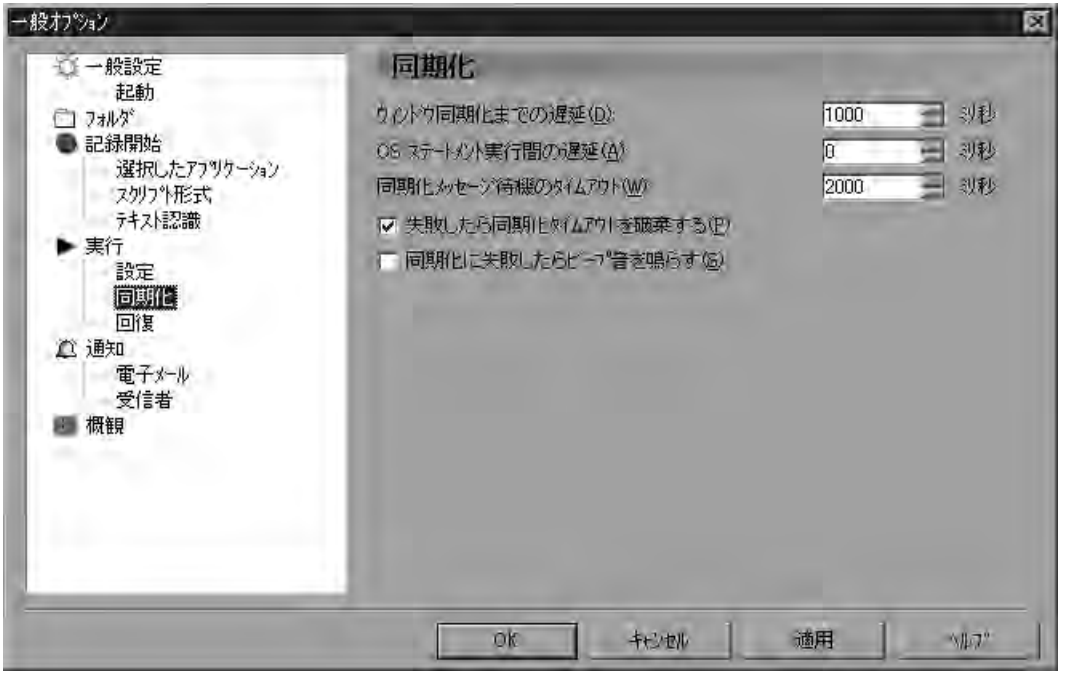
オプション	詳細
[単数のプロパティ チェックが失敗したらテストを失敗とする]	**_check_info** ステートメントが失敗したときに WinRunner のテストを失敗とするかどうかを指定します。また，これらのステートメントに対応するイベントを［テスト結果］ウィンドウに書き込みます。 (**_check_info** ステートメントは［**挿入**］＞［**GUI チェックポイント**］＞［**単数のプロパティ**］コマンドを使用して作成できます)。**check_info** 関数の詳細については，「**TSL リファレンス**」を参照してください。 テスト・スクリプトで **setvar** 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 15 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 15 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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。

オプション	詳細
[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。
[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。

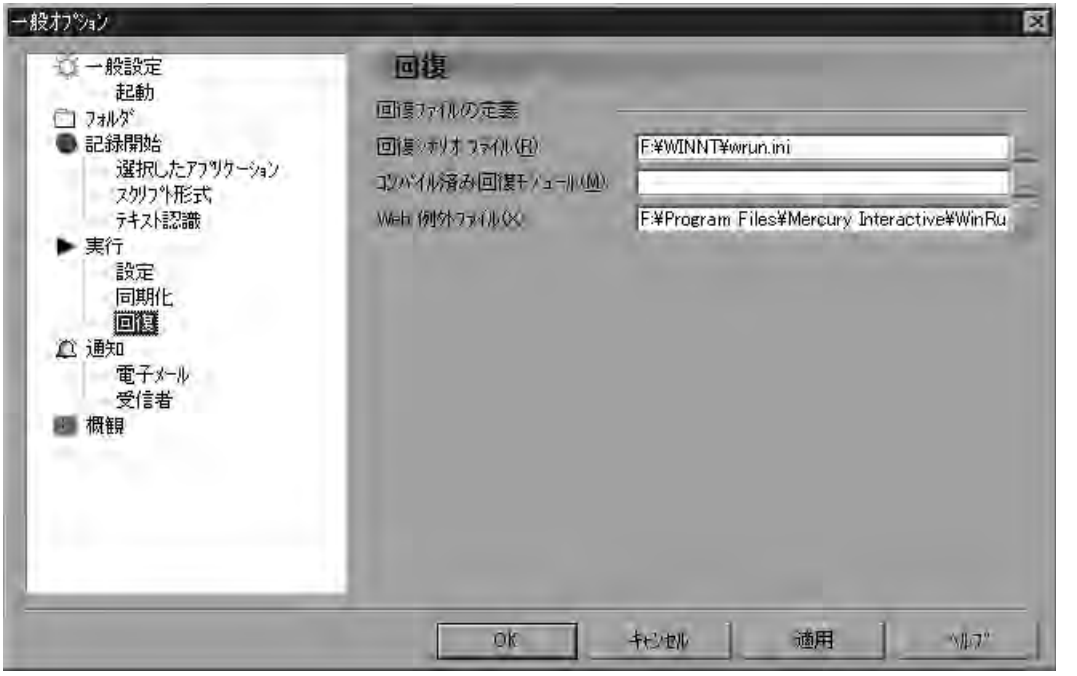
オプション	詳細
［ウィンドウの同期化までの遅延］	コンテキスト・センシティブ・チェックポイントまたは同期ポイントのためのキャプチャを行う前にウィンドウが安定したことを判定するために使われるサンプリングの間隔（単位：ミリ秒）を設定します。安定であると判定されるためには，ウィンドウが 2 回の連続するサンプリングの間で変化がないことが必要になります。このサンプリングは，ウィンドウが安定するか，またはタイムアウト（［**設定**］カテゴリの［**チェックポイントと CS ステートメントのタイムアウト**］で設定します）になるまで，続行されます。 通常は，遅延時間が短いほど，WinRunner はオ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，574 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 15 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，574 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，**『Mercury WinRunner 上級機能ユーザーズ・ガイド』**の第 21 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，**『Mercury WinRunner 上級機能ユーザーズ・ガイド』**の第 15 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，574 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，**『Mercury WinRunner 上級機能ユーザーズ・ガイド』**の第 21 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，574 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，574 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 21 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 4 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 4 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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 26 章「テスト工程の管理」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，『**Mercury WinRunner 上級機能ユーザーズ・ガイド**』の第 5 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。

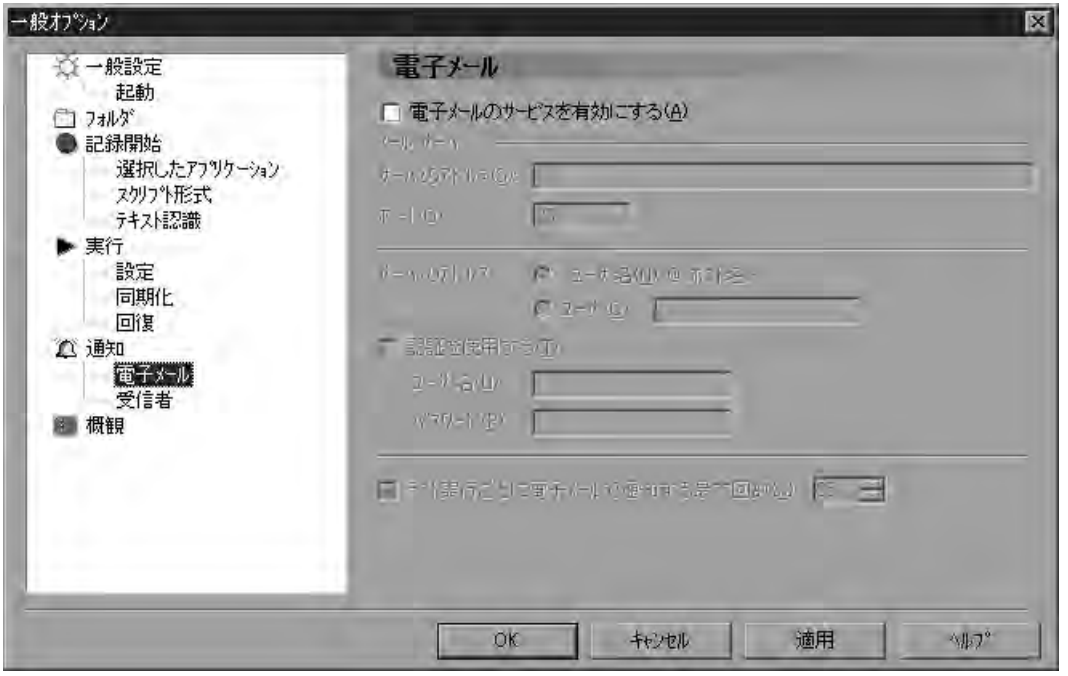
このカテゴリにあるオプションに加えて，［**電子メール**］および［**受信者**］サブカテゴリにある追加の通知オプションも設定できます。

［**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，569 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。



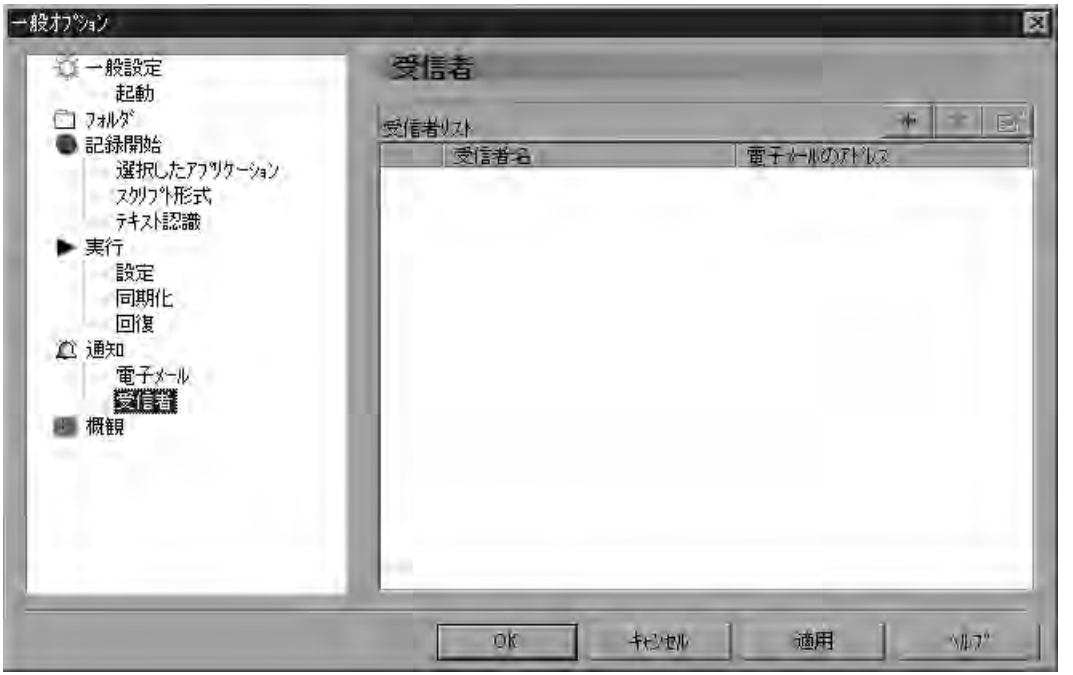
［**電子メール**］カテゴリには次のオプションがあります。

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，**『Mercury WinRunner 上級機能ユーザーズ・ガイド』**の第 21 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。

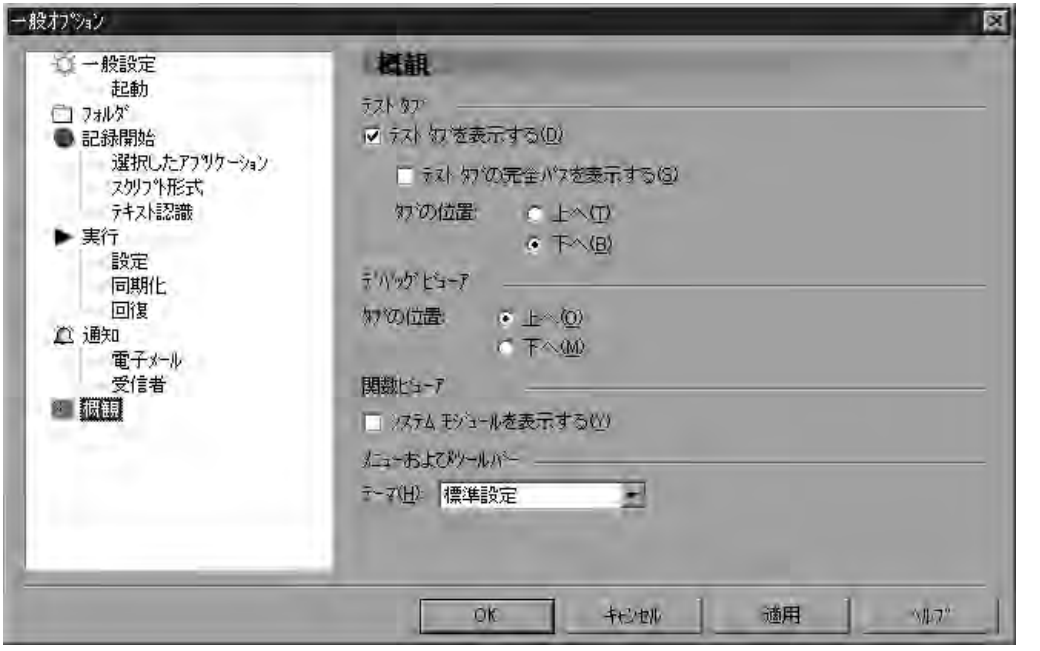


- 新しい受信者をリストに追加するには，［**受信者の追加**］ をクリックします。
- リストから受信者を削除するには，リストから受信者を選択して［**受信者の削除**］ をクリックします。
- リスト内の受信者の詳細を変更するには，リストから受信者を選択して［**受信者詳細の変更**］ をクリックします。

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。

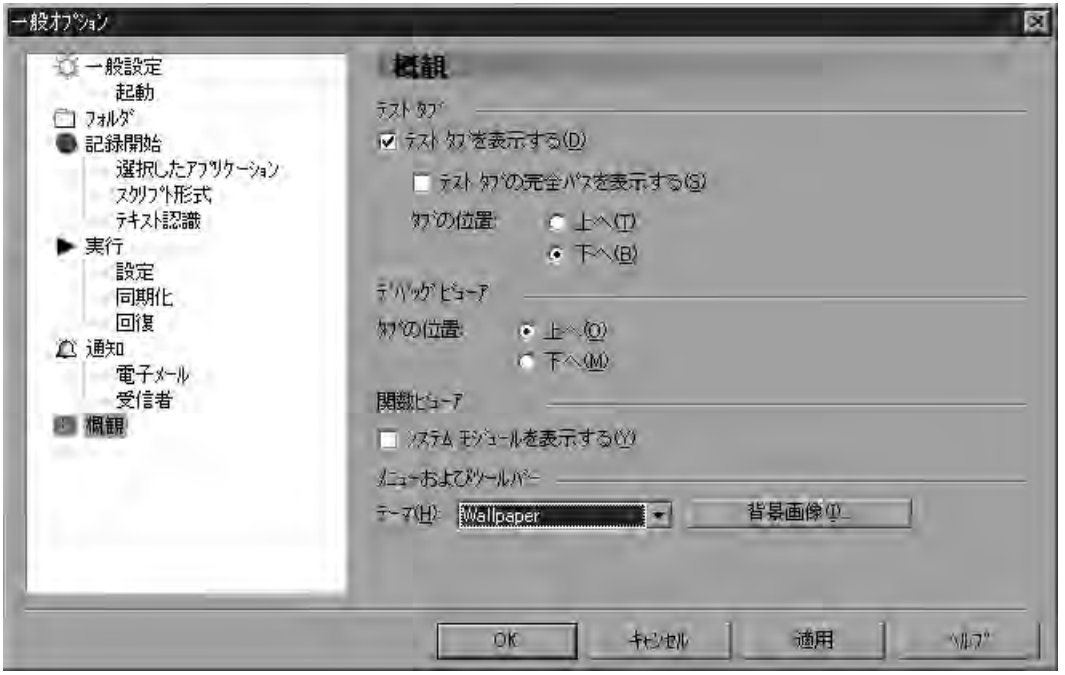
テーマの選択

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，548 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，445 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，546 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

索引

記号

$ 記号，Range プロパティ検査 172
\ 文字，正規表現 172
_web_set_tag_attr 関数 186

A

Acrobat Reader xvii
ActiveX_activate_method 関数 235
ActiveX_get_info 関数 41
ActiveX_get_info 関数 232
ActiveX_set_info 関数 234
ActiveX コントロール
  TSL テーブル関数の使用 241
  アクティブ 235
  概要 223–224
  サブオブジェクトのプロパティの検査 238–240
  サポート 223–241
  プロパティの取得 232, 235
  プロパティの設定 232, 235
  プロパティの表示 229–232
ActiveX コントロール
  概要 225–228
ActiveX コントロールのプロパティ
  取得 232–235
  設定 232–235
  表示 229–232
ActiveX コントロールをアクティブにする 235
ActiveX プロパティ・ビューア 229, 232
ActiveX，ポインタの値 41
add-ins
  QuickTest. *See* the WinRunner Advanced Features User's Guide
Advanced Features User's Guide, WinRunner xvi
application being tested, illustration 30
Attribute/ コメント 185

B

batch tests. *See* the WinRunner Advanced Features User's Guide
breakpoints. *See* the WinRunner Advanced Features User's Guide
button_check_info 関数 132, 362
button_check_state 関数 363
button_wait_info 関数 411

C

calendar クラス 163
calling functions from external libraries. *See* the WinRunner Advanced Features User's Guide
calling tests. *See* the WinRunner Advanced Features User's Guide
CHECK BITMAP OF OBJECT/WINDOW ソフトキー 113
CHECK BITMAP OF SCREEN AREA ソフトキー 113
CHECK DATABASE (CUSTOM) ソフトキー 113
CHECK DATABASE (DEFAULT) ソフトキー 113
CHECK GUI FOR MULTIPLE OBJECTS ソフトキー 113
CHECK GUI FOR OBJECT/WINDOW softkey 113
CHECK GUI FOR SINGLE PROPERTY ソフトキー 113
check_button クラス 164
check_window 関数 333
click_on_text 関数 348, 351
command line
  running tests from the. *See* the WinRunner Advanced Features User's Guide
compare_text 関数 352
Compare プロパティ検査，引数の指定 170
compiled modules. *See* the WinRunner Advanced Features User's Guide
configurations, initializing. *See* the WinRunner Advanced Features User's Guide
CRV アイコン 11

customizing
WinRunner's user interface. *See* the WinRunner Advanced Features User's Guide
customizing test scripts. *See* the WinRunner Advanced Features User's Guide
customizing the Function Generator. *See* the WinRunner Advanced Features User's Guide

D

Data Bound Grid コントロール 241
Data Junction
TransliterationIn プロパティ 325
TransliterationOut プロパティ 325
データベース・チェックポイントのデータベースの指定 324–325
標準のデータベース検査 288–289
DataWindow
オブジェクトのプロパティ検査 249–251
計算カラム 251–252
検査の選択中にプロパティを検査 248
標準の検査でプロパティを検査 247
プロパティの検査実行 247–249
DateFormat プロパティ検査
使用できる日付書式 171
引数の指定 171
DateFormat プロパティ検査でサポートされている日付書式 171
db_check 関数 269, 320
db_connect 関数 327
db_disconnect 関数 328
db_dj_convert 関数 329
db_execute_query 関数 327
db_get_field_value 関数 327
db_get_headers 関数 327
db_get_last_error 関数 328, 329
db_get_row 関数 328
db_record_check 関数 272
db_write_records 関数 328
ddt_close 関数 365, 399
ddt_export 関数 399
ddt_func.ini ファイル 368
ddt_get_current_row 関数 402
ddt_get_parameters 関数 402
ddt_get_row_count 関数 365, 373, 400
ddt_is_parameter 関数 402
ddt_next_row 関数 400
ddt_open 関数 365, 373, 380, 398
ddt_report_row 関数 391, 403
ddt_save 関数 366, 374, 378, 399, 405
ddt_set_row 関数 373, 400
ddt_set_val_by_row 関数 401, 405
ddt_set_val 関数 401, 405
ddt_show 関数 400
ddt_update_from_db 関数 366, 374, 403
ddt_val 367
ddt_val_by_row 関数 403
ddt_val 関数 367, 376, 402
debugging test scripts. *See* the WinRunner Advanced Features User's Guide
dialog boxes for interactive input
creating. *See* the WinRunner Advanced Features User's Guide
documentation, printed
WinRunner Advanced Features User's Guide xvi
DropDown DataWindow, 「DropDown オブジェクト」 *参照*
DropDownListBoxContent プロパティの検査 245
DropDown オブジェクト
検査の指定時のプロパティ検査 245
標準の検査によるプロパティ検査 244
プロパティ検査 244–247
DropDown リスト, 「DropDown オブジェクト」 *参照*
DWComputedContent プロパティ検査 251
DWTableContent プロパティの検査 247

E

edit_check_info 関数 132, 362, 363
edit_check_selection 関数 363
edit_wait_info 関数 411
edit クラス 164
error handling. *See* the WinRunner Advanced Features User's Guide
Excel, 「Microsoft Excel」 *参照*

F

FarPoint Spreadsheet コントロール 241

file_compare 関数 490
find_text 関数 348–349
FPSpread，Spread，1 MSW_class，「FarPoint Spreadsheet コントロール」*参照*
Function Generator. *See* the WinRunner Advanced Features User's Guide
functions
  calling from external libraries. *See* the WinRunner Advanced Features User's Guide
  user-defined. *See* the WinRunner Advanced Features User's Guide

G

General タブ，テストのプロパティ・ダイアログ・ボックス 391
generating functions. *See* the WinRunner Advanced Features User's Guide
GET TEXT FROM OBJECT/WINDOW ソフトキー 114
GET TEXT FROM SCREEN AREA ソフトキー 114
get_text 関数 344–347
GUI
  WinRunner による学習 42
  学習 42
GUI map
  configuring. *See* the WinRunner Advanced Features User's Guide
GUI map files
  merging. *See* the WinRunner Advanced Features User's Guide
GUI_close 関数 63
GUI_load 関数 62–63, 440
GUI_open 関数 63
GUI_unload_all 関数 63
GUI_unload 関数 63
gui_ver_add_class 関数 154, 158, 162
gui_ver_set_default_checks 関数 134, 139
GUI オブジェクト
  検査 129–179
  識別 27–30
  プロパティ値の検査 132–134
GUI オブジェクトの識別 27–30
  概要 27–29
GUI オブジェクトの探索 36–41
GUI オブジェクトのプロパティ，表示 36–41
GUI 検証結果ダイアログ・ボックス 474
  オプション 475
  期待値の更新ボタン 489
GUI 検証結果ダイアログ・ボックス
  該当なしメッセージ 151
  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メッセージ 152
  次を取得できませんメッセージ 152
  複雑な値メッセージ 151
GUI スパイ 36–41
  ActiveX タブ 40, 229–232
  記録済みタブ 38
  すべて標準タブ 37, 39
GUI チェック
  引数の指定 169–175
  標準のオブジェクトで 163–168
GUI チェック・ダイアログ・ボックス 153–155
  該当なしメッセージ 151
  このオブジェクトのプロパティを検出できませんメッセージ 152
  次を取得できませんメッセージ 152
  テーブルの検査 256
  引数を指定せずに閉じる 173
  複雑な値メッセージ 151
GUI チェックポイント
  失敗したチェックポイントのオプション 131
  テスト結果 463, 473
GUI チェックポイント 129–179, 181–222
  GUI チェックポイント・ダイアログ・ボックス 150–163
  GUI チェックリストの共有フォルダへの保存 145–146
  GUI チェックリストの変更 145–150
  GUI チェックリストの編集 146–150
  Web オブジェクトの検査 181–222
  Web オブジェクトのテキストの検査 218–222
  ウィンドウ内のすべてのオブジェクトの検査 139–141
  ウィンドウ内の複数のオブジェクトの検査 137–139
  概要 130–131
  既存の GUI チェックリストの使用 143–145
  期待結果の変更 177–179

検査を指定した場合の単数オブジェクトの検査 135–137
検査を指定してウィンドウ内のすべてのオブジェクトを検査する 141
単数のオブジェクトの検査 134–137
データ駆動型テスト 393–398
引数の指定 169–175
標準の検査 163–168
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 140
標準の検査による単数オブジェクトの検査 134–135
プロパティ検査 163–168
プロパティの期待値の編集 175–177
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 135, 140, 141, 244, 245
GUI チェックポイント・コマンド 135, 137, 140, 141
GUI チェックポイント作成ダイアログ・ボックス 156–159
該当なしメッセージ 151
このオブジェクトのプロパティを検出できませんメッセージ 152
次を取得できませんメッセージ 152
引数を指定せずに閉じる 173
複雑な値メッセージ 151
GUI チェックポイント・ダイアログ・ボックス 150–163
GUI チェックポイント・ダイアログ・ボックスの終了 173
GUI チェックポイント－単数のプロパティ・ソフトキー 133
GUI チェックポイントの期待結果 142
変更 177–179
編集 175–177
GUI チェックポイントの単数のプロパティ・コマンド
データ駆動型テスト 361
GUI チェックポイント－複数のオブジェクト・ソフトキー 137
GUI チェックリスト 142
既存の使用 143–145
共有 145–146
変更 145–150
編集 146–150
GUI チェックリスト編集コマンド 146, 147, 160
GUI チェックリスト編集ダイアログ・ボックス 160–163
このオブジェクトのプロパティを検出できませんメッセージ 152
引数を指定せずに閉じる 173
GUI，テスト対象アプリケーション
GUI マップ・エディタを使った学習 57
テスト・スクリプト・ウィザードを使った学習 50
GUI テスト・ビルダ，GUI マップ・エディタ *参照*
GUI ファイル・コマンド（GUI マップ・エディタ）75
GUI ファイルを TestDirector プロジェクトに保存ダイアログ・ボックス 61
GUI ファイルを開くダイアログボックス 64
GUI ファイルを保存ダイアログ・ボックス 59
GUI マップ
オブジェクトまたはウィンドウの検索 43
概要 35–36
作成 50–58
紹介 27–33
表示 32
保存 59–61
理解 35–46
ロード 61–65
GUI マップ・エディタ 75–77
GUI ファイルのロード 63–65
アプリケーションの GUI の学習 57–58
オブジェクトの削除 90
学習ボタン 57
拡大表示 86
記述 76
紹介 32
表示されるオブジェクトのフィルタ処理 91
ファイル間でのオブジェクトのコピーと移動 85
GUI マップ・コマンド（GUI マップ・エディタ）75
GUI マップ・コマンドの検索 43
GUI マップの構成設定
Web オブジェクト 183
GUI マップの作成 50–58

GUI マップ・エディタを使用 57–58
記録による 56–57
テスト・スクリプト・ウィザードを使用 50–56
GUI マップの編集 73–92
GUI マップ・ファイル
テスト特有の GUI マップ・ファイル・モードでの更新 70
GUI マップ・ファイル
GUI_load 関数を使ったロード 62–63
GUI マップ・エディタを使ったロード 63–65
一時的に保存 59
オブジェクトの削除 90
オブジェクトの追加 89
ガイドライン 43
削除 90
単一オブジェクトの検索 87
テスト間での共有 49–50
ファイル間でのオブジェクトの検索 88
ファイル間でのオブジェクトのコピーと移動 85
複数のオブジェクトの検索 88
編集 73–92
保存 59–61
ロード 61–65
変更の保存 92
GUI マップ・ファイルからのオブジェクトの削除 90
GUI マップ・ファイル間で，GUI オブジェクトの記述を移動する 85
GUI マップ・ファイル間で，GUI オブジェクトの記述をコピーする 85
GUI マップ・ファイルでの引用符 82
GUI マップ・ファイルに対する変更の保存 92
GUI マップ・ファイルの削除 90
GUI マップ・ファイルのロード
GUI_load 関数の使用 62–63
GUI マップ・エディタの使用 63–65
GUI マップ・ファイルへのオブジェクトの追加 89
GUI マップ・ファイル・モード
グローバルな GUI マップ・ファイル 525
テスト特有の GUI マップ・ファイル 525
GUI マップ・ファイル・モード
グローバル GUI マップ・ファイル・モード 47–66
テストごとの GUI マップ・ファイル・モード 67–71
比較 44–46

H

h mm AM/PM コマンド，データ・テーブル 383
html_check_button オブジェクト 183
html_combobox オブジェクト 183
html_edit オブジェクト 183
html_frame オブジェクト 183
html_listbox オブジェクト 183
html_push_button オブジェクト 183
html_radio_button オブジェクト 183
html_rect オブジェクト 183
html_text_link オブジェクト 183
HWND ウィンドウ・ハンドル 183

I

initialization tests. *See* the WinRunner Advanced Features User's Guide
INSERT FUNCTION FOR OBJECT/WINDOW ソフトキー 114
INSERT FUNCTION FROM FUNCTION GENERATOR softkey 114
invoke_application function 106
invoke_application 関数 513

L

list_check_info 関数 133, 362
list_check_item 関数 363
list_check_selected 関数 363
list_wait_info 関数 411
list クラス 166
load testing. *See* the WinRunner Advanced Features User's Guide
LoadRunner
説明 せつめい 10
LoadRunner. *See* the WinRunner Advanced Features User's Guide

M

managing the testing process. *See* the WinRunner

Advanced Features User's Guide
menu_item クラス 166
menu_select_item 関数 108
menu_wait_info 関数 411
merging GUI map files. *See* the WinRunner Advanced Features User's Guide
MHGLBX，Mh3dListCtrl，1 MSW_class，「MicroHelp MH3dList コントロール」*参照*
MicroHelp MH3dList コントロール 241
Microsoft Excel，データ・テーブル 380, 398
Microsoft Grid コントロール 241
Microsoft Query
　データベースからのデータのインポート 386–387
　データベース・チェックポイントのデータベースを指定 323–324
　標準のデータベース検査 287–288
modules, compiled. *See* the WinRunner Advanced Features User's Guide
move_locator_text 関数 349–350
MSDBGrid，DBGrid MSW_class，「Data Bound Grid コントロール」*参照*
MSGrid，Grid MSW_class，「Microsoft Grid コントロール」*参照*

N

INSERT FUNCTION FROM FUNCTION GENERATOR ソフトキー 113, 114

O

obj_check_bitmap 関数 337
　データ駆動型テスト 393
obj_check_gui 関数 141–143
　データ駆動型テスト 393
obj_check_info 関数 133, 362
obj_check_text 関数 347
obj_click_on_text 351
obj_exists 関数 410
obj_find_text 関数 348–349
obj_get_text 関数 344–347
obj_mouse 関数 102
obj_move_locator_text 349–350
obj_wait_bitmap 関数 416
　データ駆動型テスト 393
obj_wait_info 関数 411
objects
　virtual. *See* the WinRunner Advanced Features User's Guide
object クラス 166
OCX コントロール，「ActiveX コントロール」*参照*
OCX プロパティ・ビューア，「ActiveX プロパティ・ビューア」*参照*
ODBC
　データベース・チェックポイントのデータベースを指定 323–324
　標準のデータベース検査 287–288
ODBC Query の変更ダイアログ・ボックス 313, 316
OLE コントロール，「ActiveX コントロール」*参照*
online resources xvii

P

pause ソフトキー 432
pausing test execution using breakpoints. *See* the WinRunner Advanced Features User's Guide
PowerBuilder
　DataWindow 247–249, 249–251, 251–252
　DropDown オブジェクト 244–247
PowerBuilder アプリケーション
　概要 243–244
PowerBuilder アプリケーション 243–252
programming in TSL. *See* the WinRunner Advanced Features User's Guide
push_button クラス
　プッシュボタン・オブジェクト 167

Q

qcdb_add_defect 関数 493, 495
Quality Center 509
　TdApiWnd アイコン 11
　working with
　説明 9
　テスト実行中の不具合の報告 495
　統一レポートからの接続 469
　不具合の追加ダイアログ・ボックス 493
Quality Center プロジェクトにテストを保存ダイアログ・ボックス 119

Quality Center への接続ダイアログ・ボックス 469
QuickTest
  loading associated add-ins
  supported versions

R

radio_button クラス 164
Range プロパティ検査
  通貨記号 172
  引数の指定 172
Read Me file xvii
RECORD softkey 113
recovery scenarios. *See* the WinRunner Advanced Features User's Guide
regular expressions. *See* the WinRunner Advanced Features User's Guide
RegularExpression プロパティ検査，引数の指定 172
results of tests. *See* the WinRunner Basic Features User's Guide xvi
RTL スタイルのウィンドウ
  WinRunner のアプリケーションのサポート 107
RTL スタイルのウィンドウを持つアプリケーションの WinRunner サポート 107
run from arrow ソフトキー 431
run from top ソフトキー 431
running tests
  batch run. *See* the WinRunner Advanced Features User's Guide
  for debugging. *See* the WinRunner Advanced Features User's Guide
  from the command line. *See* the WinRunner Advanced Features User's Guide

S

sample tests xvii
scroll_check_info 関数 132, 362
scroll_check_pos 関数 363
scroll_wait_info 関数 411
scroll クラス 167
Section 508 96
set_window 関数 33
setting test properties. *See* the WinRunner Basic Features User's Guide xvi
setting testing options
  within a test script. *See* the WinRunner Advanced Features User's Guide
setvar 関数 439
Sheridan Data Grid コントロール 241
spin_wait_info 関数 411
SQL ステートメント
  既存のクエリ 327
  結果セットの作成，基準 327
  実行時チェックリストの編集 283
  データベースチェックポイントウィザードで指定 297
  データベース・チェックポイントでのパラメータ化 319
SSDataWidgets，SSDBGridCtrl，1，「Sheridan Data Grid コントロール」*参照*
static_check_info 関数 132, 362
static_check_text 関数 363
static_text クラス 164
static_wait_info 関数 411
statusbar_wait_info 関数 411
Step button 430
Step Into button 430
step into ソフトキー 431
STEP OUT softkey 432
step to cursor ソフトキー 432
step ソフトキー 431
Stop button 102, 106
Stop Recording command 102, 106
STOP softkey 114
stop ソフトキー 432
SYNCHRONIZE BITMAP OF OBJECT/WINDOW ソフトキー 113
SYNCHRONIZE BITMAP OF SCREEN AREA ソフトキー 113
SYNCHRONIZE OBJECT PROPERTY (CUSTOM) ソフトキー 113

T

tab_wait_info 関数 411
TableContent プロパティ検査 255–257
tbl_activate_cell 関数 241
tbl_activate_header 関数 241
tbl_get_cell_data 関数 241
tbl_get_cols_count 関数 241
tbl_get_column_name 関数 241

tbl_get_rows_count 関数 241
tbl_get_selected_cell 関数 241
tbl_get_selected_row 関数 241
tbl_select_col_header 関数 241
tbl_set_cell_data 関数 241
tbl_set_selected_cell 関数 241, 244, 245
tbl_set_selected_row 関数 241
TdApiWnd アイコン 11
tddb_ 関数，「qcdb_ 関数」*参照*
technical support online xviii
Test Script Language (TSL). *See* the WinRunner Advanced Features User's Guide
test scripts
  customizing. *See* the WinRunner Advanced Features User's Guide
TestDirector データベースからテストを開くダイアログ・ボックス 123
TestDirector 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 65
testing options
  within a test script. *See* the WinRunner Advanced Features User's Guide
testing process
  managing the
tests
  calling. *See* the WinRunner Advanced Features User's Guide
TestSuite 9
TimeFormat プロパティ検査
  使用できる時間書式 173
  引数の指定 173
TimeFormat プロパティ検査でサポートされている時間書式 173
toolbar_select_item function 107
True DBGrid コントロール 241
TrueDBGrid50，TDBGrid MSW_class，「True DBGrid コントロール」*参照*
TrueDBGrid60，TDBGrid MSW_class，「True DBGrid コントロール」*参照*
TrueOleDBGrid60，TDBGrid MSW_class，「True DBGrid コントロール」*参照*
TSL Online Reference xvii
TSL Reference Guide xvi
TSL 関数
  データ駆動型テストで 398–404
  データベース作業 325–329
TSL を表示ボタン，WinRunner テスト結果ウィンドウ 179, 317
typographical conventions xix

U

user interface, customizing. *See* the WinRunner Advanced Features User's Guide
user-defined functions
  adding to the Function Generator. *See* the WinRunner Advanced Features User's Guide
user-defined functions. *See* the WinRunner Advanced Features User's Guide

V

variables
  monitoring. *See* the WinRunner Advanced Features User's Guide
virtual objects. *See* the WinRunner Advanced Features User's Guide
Visual Basic コントロール
  概要 223–224
  サブオブジェクトのプロパティの検査 238–240
  サポート 223–241
  プロパティの取得 232
  プロパティの設定 232
  プロパティの表示 229–232
Visual Basic コントロール
  概要 225–228
Visual Basic コントロールのプロパティ
  取得 232–235
  設定 232–235
  表示 229, 232

W

wait_window 関数 419
Watch List. *See* the WinRunner Advanced Features User's Guide
WDiff ユーティリティ 490
Web exception handling. *See* the WinRunner Advanced Features User's Guide
Web objects
  properties for all objects 184
web_frame_get_text 218, 219
web_frame_text_exists 218, 221

web_obj_get_text 218, 219
web_obj_text_exists 218, 221
WebTest アドイン
GUI マップの構成設定での 183
WebTest アドイン 104
Web オブジェクト
Web 画像のプロパティ 187
Web テーブル・セルのプロパティ 189
Web テーブルのプロパティ 188
Web ボタン・オブジェクトのプロパティ 192
記録済みプロパティの表示 182
作業 181–222
チェック・ボックス・オブジェクトのプロパティ 190
テキスト・リンクのプロパティ 188
テストでのプロパティの使用 183–193
フレーム・オブジェクトのプロパティ 187
編集ボックス・オブジェクトのプロパティ 191
ラジオ・ボタンのプロパティ 190
リストおよびコンボ・ボックス・オブジェクトのプロパティ 192
Web オブジェクト 181–222
壊れたリンクの検査 205–207
テーブルの内容の検査 209–210
テキストの検査 218–222
テキスト・リンクのフォントまたは色の検査 203–204
標準的なフレーム・プロパティの検査 194–195
フレーム，セル，リンク，画像の内容の検査 198–199
フレーム，テーブル，セルの構造の検査 196–198
フレーム内のオブジェクト数の検査 195–196
リンクの URL の検査 201–202
列数と行数の検査 199–200
Web 画像プロパティ 187
Web テーブル・セル・プロパティ 189
Web テーブル・プロパティ 188
What's New in WinRunner help xvii
win_activate function 106
win_check_bitmap 関数 337, 339
データ駆動型テスト 393
win_check_gui 関数 141–143
データ駆動型テスト 393
win_check_info 関数 133, 362
win_check_text 関数 347
win_click_on_text 351
win_exists 関数 410
win_find_text 関数 348–349
win_get_text 関数 344–347
win_move_locator_text 349–350
win_wait_bitmap 関数 416
データ駆動型テスト 393
win_wait_info 関数 411
window クラス 168
WinRunner 16
online resources xvii
概要 3–10, 11–23
起動 11–13
ステータス・バー 14
タイトル・バー 14
テスト・ウィンドウ 16
メイン・ウィンドウ 14
メニュー・バー 14
WinRunner Context-Sensitive Help xvii
WinRunner Customization Guide xvii
WinRunner Installation Guide xvi
WinRunner Quick Preview xvii
WinRunner Tutorial xvi
WinRunner テストの圧縮 125
WinRunner テストの解凍 125
WinRunner アドインのロード 21–??
WinRunner 起動時のアドインのロード 21
WinRunner 記録 / 実行エンジン・アイコン 11
WinRunner テスト結果ウィンドウ 457–464, 465
WinRunner テスト結果ウィンドウ
GUI チェックポイントの期待結果の 178
WinRunner テストの解凍 125
WinRunner によるアプリケーションの GUI の学習 50–58
概要 42
記録による 56–57
テスト・スクリプト・ウィザードを使用 50–56
GUI マップ・エディタを使用 57–58
WinRunner へようこそウィンドウ 13

WinRunner レポート 457
テスト・サマリ 461
テスト・ログ 463
メニュー・バーとツールバー 459
WinRunner レポート・ビュー
定義 444
WinRunner を最小化，テストの記録時 106

X
XR_GLOB_FONT_LIB 353

Y
yyyy/MM/dd コマンド，データ・テーブル 383

あ
値の指定 438
値を貼り付けコマンド，データ・テーブル 381
圧縮ファイルからのテストのエクスポート 125
圧縮ファイルへのテストのインポート 125
アドイン 508
WinRunner 起動時のロード 21
アドイン・タブ，テストのプロパティ・ダイアログ・ボックス 508
アドインのロード
WinRunner 起動時の 21
アドイン・マネージャ・ダイアログ・ボックス 21
アナログ・モード 5, 104
アプリケーション，起動 512, 512–518
アプリケーションの GUI の学習 42, 50–58
GUI マップ・エディタを使用 57–58
記録による 56–57
テスト・スクリプト・ウィザードを使用 50–56

い
一時 GUI マップ・ファイル
保存 59
一時 GUI マップ・ファイル，保存 59
一時停止コマンド 431
一時停止ボタン 431
一般オプション
一般 523
ãNiÆ 526
概観 570
記録 532
通知 564
フォルダ 528
一般オプション・ダイアログ・ボックス 511, 519
一般タブ
テストのプロパティ・ダイアログ・ボックス 502
一般的な object クラス 166
印刷コマンド 126
データ・テーブル 381
印刷設定コマンド，データ・テーブル 381
印刷，テスト結果 453
インデントを減少コマンド 116
インデントを増加コマンド 116
インポート・コマンド，データ・テーブル 380

う
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 141
ウィンドウの同期化ポイント 409–410
ウィンドウビットマップのチェック・ソフトキー 336
ウィンドウビットマップの同期化ソフトキー 416
ウィンドウ・ラベルの変更 82
上書き保存コマンド 117
データ・テーブル 380

え
エクスポート・コマンド，データ・テーブル 380

お
大文字と小文字を区別しスペースを無視する検証
テーブル 216
大文字と小文字を区別しない検証
テーブル 216
大文字と小文字を区別する検証
テーブル 216
大文字と小文字を区別せずスペースを無視する検証
テーブル 216
置換コマンド
データ・テーブル 382

オブジェクト
GUI マップの検索 43
オブジェクト / ウィンドウからテキスト取得ボタン 20
オブジェクト / ウィンドウからのテキスト取得ボタン 345
オブジェクト / ウィンドウの GUI チェックポイント・ボタン 141
オブジェクト / ウィンドウのの GUI チェックポイント・ボタン 135
「オブジェクト / ウィンドウ用の GUI チェックポイント・コマンド」*参照*
オブジェクト / ウィンドウの関数を挿入するボタン 20
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 336
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 20
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・コマンド 416
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・ボタン 416
オブジェクト / ウィンドウのビットマップの同期化ポイント・ボタン 20
ÉIÉuÉWÉFÉNÉg/ÉEÉBÉiÉhÉEÉrÉbÉgÉ}ÉbÉv ÇÃiØäŽâª ソフトキー 419
オブジェクト / ウィンドウプロパティ同期化コマンド 412
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 412
オブジェクト / ウィンドウプロパティの同期化ポイント・ボタン 20
オブジェクト / ウィンドウ用の GUI チェックポイント・コマンド 135, 140, 141, 153
オブジェクト / ウィンドウの GUI チェックポイント・ボタン 20
オブジェクトのクラス・ダイアログ・ボックス 154, 157, 161
オブジェクトの同期化ポイント 409–410
オブジェクトの物理的記述の修正 79–81
オブジェクトの論理名の修正 79–81
オブジェクト，またはウィンドウの GUI チェックポイント・ボタン 135, 140, 153
オブジェクトまたはウィンドウのビットマップ・チェックポイントのショートカット・キー 333
オブジェクト，またはウィンドウのビットマップチェックポイント・ボタン 336
オプション，グローバル・テスト，「グローバル・テスト・オプションの設定」*参照* 519

か

カーソル行まで実行コマンド 430
概観オプション 570
開始トランザクション 109
該当なしメッセージ
GUI チェックポイントのダイアログ・ボックス内 151
ガイドライン
GUI マップ・ファイルの使用 43
グローバル GUI マップ・ファイル・モードで作業する場合の 65–66
テストごとの GUI マップ・ファイル・モードでの作業 71
学習ボタン，GUI マップ・エディタ 57
壁紙 571
独自の背景の設定 572
画面領域からテキスト取得コマンド 346
画面領域からテキスト取得ボタン 20, 346
画面領域のビットマップ・チェックポイントのショートカット・キー 333
カラム，計算 251–252
関数
起動 512–518
定義，起動 516
関連付け，テストへのアドインの 508

き

キーボード・ショートカット 112, 431
キー割り当て
標準 112, 431
記述，「物理的記述」*参照*
期待結果 427, 435
更新実行 427
指定実行 437
ビットマップ，GUI，データベース・チェックポイントの更新 489
複数セットの作成 435
期待結果値の編集ボタン 175–177
期待結果データ・ビューア 482, 488
期待結果フォルダ，場所 511

期待結果フォルダ・ボックス 511
期待値と実際の値を比較ボタン
GUI 検証結果ダイアログ・ボックス 475
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 485
期待値の更新ボタン
GUI 検証結果ダイアログ・ボックス 475, 489
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 485
期待値を編集ボタン 177
起動アプリケーションおよび起動関数 512, 512–518
起動オプション 526
起動関数 516
コンパイル済みモジュール 517
共有フォルダ
GUI チェックリストの 145–146
データベース・チェックリストの 307–309
切り取りコマンド 115
データテーブル 381
記録
オプション 532
問題 102–104
記録 - アナログ・コマンド 102
記録コマンド 102
記録 - コンテキスト・センシティブ・コマンド 102
記録 / 実行エンジン・アイコン 11
記録ボタン 102

く

クリアー書式コマンド，データ・テーブル 381
クリアー全てコマンド，データ・テーブル 381
クリアー内容コマンド，データ・テーブル 382
クリックしてコンテキストセンシティブモードで記録ボタン 20
グローバル GUI マップ・ファイル・モード 47–66
ガイドライン 65–66
概要 47–49
テスト・オプション
グローバル，「グローバル・テスト・オプションの設定」*参照*
グローバル・テスト・オプション，「グローバル・テスト・オプションの設定」*参照*
グローバル・テスト・オプションの設定 519–577
現在のテストの設定 511
テキスト認識 544–546
テストの実行 548–563
グローバルな GUI マップ・ファイル・モード 525

け

計算カラム 251–252
結果セット 268
結果の絞り込み，統一レポート 452
結果のスキーマ 456
結果の表示，カスタマイズ 456
結果フォルダ
期待結果 427, 435
検証 425, 432
デバッグ 434
検査
ウィンドウ内のすべての GUI オブジェクト 139–141
ウィンドウ内の複数の GUI オブジェクト 137–139
検査を指定した場合の単数 GUI オブジェクト 135–137
検査を指定するウィンドウ内のすべての GUI オブジェクト 141
単数の GUI オブジェクト 134–137
標準の検査によるウィンドウ内のすべての GUI オブジェクト 140
標準の検査による単数の GUI オブジェクト 134–135
現在の行ボックス 510
現在のテスト・タブ，テストのプロパティ・ダイアログ・ボックス 510
現在のテストの設定 510–511
現在のフォルダ・ボックス 511
検索
GUI マップ・ファイルの単一オブジェクト 87
GUI マップ・ファイルの複数のオブジェクト 88
検索コマンド 116
データ・テーブル 382

検証結果 425, 432
検証結果フォルダ，場所 511
検証結果フォルダ・ボックス 511
検証タイプ
　テーブル 216
検証のタイプ
　データベースの 304
　テーブルの 264
検証，ビットマップ，「ビットマップ・チェックポイント」*参照*
検証方式
　データベース 303
　テーブルの 261
検証メソッド
　テーブル 213
検証モード 423, 425, 432
検証ルール・コマンド，データ・テーブル 384

こ
更新モード 423, 427
固定コマンド，データ・テーブル 383
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メッセージ
　GUI チェックポイントのダイアログ・ボックス 152
このオブジェクトのプロパティを検出できませんメッセージ ,GUI チェックポイントのダイアログ・ボックス内 152
コピー・コマンド 115
　データ・テーブル 381
コマンド・ライン
　アプリケーションの実行 513
コメント解除コマンド 116
コメント・コマンド 116
コンテキスト・センシティブ
　記録，一般的な問題 102–104
　テスト，紹介 27–33
　テストの実行，一般的な問題 440–442
　モード 4, 99–101
コンテキスト・センシティブ・モード 27
コンパイル済みモジュール
　起動関数 517

さ
再計算コマンド，データ・テーブル 382
最小化して実行コマンド 429
最小化して実行＞先頭からコマンド 430
最小化して実行＞矢印からコマンド 430
最小化の状態で先頭から実行するコマンド 512
削除コマンド 115
　データ・テーブル 382
削除ボタン
　GUI チェックポイント作成ダイアログ・ボックス内 157
　GUI チェックリスト編集ダイアログ・ボックス内 161

し
指数コマンド，データ・テーブル 383
システム変数，「テスト・オプションの設定」*参照*
実行ウィザード 78–79
実行コマンド 429
実行時レコード・チェック 272–280
実行時レコード・チェックポイント
　成功条件の変更 286
実行時レコード・チェックポイント・ウィザード 281–286
　New アイコン 285
実行時レコード・チェックポイント・ウィザードでの New アイコン 285
実行時レコード・チェックリスト・コマンドの編集 281
実行タブ，テストのプロパティ・ダイアログ・ボックス 512
実行モード
　現在のテストに対する表示 511
　検証 423, 425
　更新 423, 427
　デバッグ 423, 425, 434
　ボックス 511
実行矢印 16, 97
失敗のみ表示ボタン
　GUI 検証結果ダイアログ・ボックス 475
　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 485
指定チェックを削除ボタン，チェックの編集ダイアログ・ボックスの 260
指定，引数 169–175
　DateFormat プロパティ検査 171
　Range プロパティ検査 172

RegularExpression プロパティ検査 172
TimeFormat プロパティ検査 173
引数を指定ダイアログ・ボックス 173
自動挿入リスト・コマンド，データ・テーブル 383
ジャンプ・コマンド，データ・デーブル 382
修正
オブジェクトの論理名 46, 66
修正ダイアログ・ボックス（GUI マップ・エディタ） 81
修正ボタン，データベース・チェックリスト編集ダイアログ・ボックス 313, 316
終了トランザクション
トランザクションの宣言 109
出力パラメータ 505
書式メニュー・コマンド，データ・テーブル 383
新規作成コマンド 117
データ・テーブル 380
新規作成ボタン 117

す

数値の検証
データベース 305
テーブル 216, 264
数値の範囲の検証
データベース 305
テーブル 216, 264
スキーマ，結果の 456
スクリーン領域のチェック・ソフトキー 338
スクリーン領域の同期化ソフトキー 418, 419
スクリプト・ウィザード，「テスト・スクリプト・ウィザード」*参照*
ステータス・バー，WinRunner 14
ステップアウト・コマンド 430
ステップイントゥ・コマンド 430
ステップ・コマンド 430
全てクリア・ボタン
GUI チェック・ダイアログ・ボックス内 154
GUI チェックポイント作成ダイアログ・ボックス内 157
GUI チェックリスト編集ダイアログ・ボックス内 161
全て選択コマンド 115
全て選択ボタン
GUI チェック・ダイアログ・ボックス内 154
GUI チェックポイント作成ダイアログ・ボックス内 157
GUI チェックリスト編集ダイアログ・ボックス内 161
全て追加ボタン
GUI チェック・ダイアログ・ボックス内 154
GUI チェックポイント作成ダイアログ・ボックス内 157
GUI チェックリスト編集ダイアログ・ボックス内 161
全て閉じるコマンド 127
すべてのプロパティを表示ボタン
GUI 検証結果ダイアログ・ボックス 475
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 486
全てのプロパティを表示ボタン
GUI チェック・ダイアログ・ボックス 155
GUI チェックポイントを作成ダイアログ・ボックス 158
GUI チェックリストを編集ダイアログ・ボックス 162
すべて保存 117
全て保存コマンド 117

せ

正規表現
変更，物理的記述 84
設定，テストのプロパティ 499–518
テストの一般情報の文書化 502
テストの説明情報の文書化 504
テストのプロパティ・ダイアログ・ボックス 500
説明タブ，テストのプロパティ・ダイアログ・ボックス 504
選択されたプロパティのみ表示ボタン
GUI チェック・ダイアログ・ボックス 154
GUI チェックポイントを作成ダイアログ・ボックス 158
GUI チェックリストを編集ダイアログ・ボックス 162

選択範囲のビットマップチェックポイント 338
選択範囲のビットマップ・チェックポイント・ボタン 20
選択範囲のビットマップ同期化ポイント・コマンド 418
選択範囲のビットマップ同期化ポイント・ボタン 418
選択範囲のビットマップチェックポイント・ボタン 338
選択範囲のビットマップの同期化ポイント・ボタン 20
先頭から実行コマンド 429
先頭から実行するコマンド 512
先頭から実行するコマンド 512
先頭から実行ボタン 429

そ

挿入コマンド，データ・テーブル 382
挿入ジェネレータボタンから関数を挿入 20
ソート・コマンド，データ・テーブル 383
ソフトキー
標準の設定 112, 431

た

大 / 小文字を区別しスペースを無視検証
データベース 305
テーブル 264
大小文字を区別しない
データベース 305
大小文字を区別しない検証
テーブル 264
大小文字を区別する検証
データベース 304
テーブル 264
大 / 小文字を区別せずスペースを無視検証
データベース 305
大 / 小文字を区別せずスペースを無視検証タイプ 264
タイトル・バー，WinRunner 14
単数オブジェクトで検査するプロパティの指定 134–137
単数オブジェクトの検査の指定 135–137
単数のプロパティ・コマンドの GUI チェックポイント
データ駆動型テスト 393

ち

チェックの編集ダイアログ・ボックス 211
1 つの列で構成されるテーブル 215
期待データの編集 217, 265, 305
検査するセルの指定 212, 260, 302
検証タイプ 216
検証のタイプ 264, 304
検証方式 261, 303
検証メソッド 213
単一カラムテーブル 263
単数のカラムのデータベース 303
データベースの検査 300–306
テーブルの検査 259–266
複数カラムのテーブル 259
複数のカラムのデータベース 300
チェックポイント
GUI 108, 129–179
概要 108
期待結果の更新 489
失敗したチェックポイントのオプション
ÉrÉbÉgÉ}ÉbÉv 334
GUI 131
ÉfÅ[É^ÉxÅ[ÉX 271
データベース 267–293
テキスト 108, 341–358
ビットマップ 108, 331–339
チェックリスト
「GUI チェックリスト」 *または*「データベース・チェックリスト」*参照*
チェックリストの保存ダイアログ・ボックス
GUI チェックリスト 146
データベース・チェックリスト 308
チェックリストを開くダイアログ・ボックス
GUI チェックリストの 144, 147
データベース・チェックリスト 308, 309, 312, 315
置換コマンド 116

つ

追加ダイアログ・ボックス（GUI マップ・エディタ）89
追加ボタン
GUI チェックポイント作成ダイアログ・ボックス内 157
GUI チェックリスト編集ダイアログ・

ボックス内 161
通貨（0）コマンド，データ・テーブル 383
通貨（2）コマンド，データ・テーブル 383
通貨記号，Range プロパティ検査 172
通知オプション 564
ツールバー
テスト 14, 18
デバッグ 18
ファイル 17
フローティングの作成 17
ユーザ 14, 19
次を検索コマンド 116
次を取得できませんメッセージ
GUI チェックポイントのダイアログ・ボックス内 152
データベースチェックポイント・ダイアログ・ボックス 293
ツリーを圧縮コマンド（GUI マップ・エディタ）76
ツリーを拡大コマンド（GUI マップ・エディタ）76

て

定義，パラメータ 505
停止コマンド 430
停止ボタン 20, 430
データ駆動型テスト 359–406
ddt_func.ini ファイル 368
GUI チェックポイント 393–398
TSL 関数を使った 398–404
ガイドライン 404–405
概要 360
工程 360–391
作成，手作業 372–376
実行 390
データ駆動テスト・ウィザード 364–372
データ駆動テスト・ウィザードを使ったテストの作成 364–370
データ・テーブルの技術仕様 384
データ・テーブルの編集 378–384
データベースからのデータのインポート 378
手作業によるデータ・テーブルの作成 374–376
手作業によるテスト・スクリプトのパラメータ化 374–376
テスト結果の分析 390
テスト・スクリプトの手作業でのパラメータ化 374
テストの変換 363–376
ビットマップ・チェックポイント 393–398
ビットマップ同期化ポイント 393–398
ユーザ定義関数 368
データ駆動テスト・ウィザード 364–372
データ・テーブル
Microsoft Excel での作業 380
Microsoft Excel の使用 398
新しい名前で保存 374
新しい場所に保存 374
カラムの定義 377
技術仕様 384
行の定義 377
最小の正数 384
最大カラム数 384
最大行数 384
最大のカラム幅 384
最大の行の高さ 384
最大の式の長さ 384
最大の正数 384
書式メニュー・コマンド 383
数値の精度 384
データ形式の変更を防ぐ 378
データ・メニュー・コマンド 382
テーブル・フォーマット 384
手作業で作成したデータ駆動型テストの宣言 372
テスト・スクリプト内での複数のデータ・テーブルの使用 374
標準 391
ファイル・メニュー・コマンド 380
編集 378–384
編集メニュー・コマンド 381
メイン 391
有効なカラム名 384
データテーブル・コマンド 379
データ・テーブルの有効なカラム名 384
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 371, 377, 379
データのパラメータ化コマンド 374
データのパラメータ化ダイアログ・ボックス

374
データ比較ビューア 477
データベース
Data Junction のエクスポート・ファイルの実行 329
Data Junction の最後のオペレーションの最後のエラー・メッセージを返す 329
Data Junction でのクエリーの作成 324–325
Data Junction での標準の検査 288
Data Junction での標準のデータベース検査 289
ODBC/Microsoft Query でのクエリーの作成 323–324
ODBC/Microsoft Query を使った標準の検査 287–288
概要 268–270
カラム・ヘッダの数と内容を返す 327
既存クエリーの変更 311–316
期待データの編集 305
行の内容を返す 328
結果セット 268
検査 267–293
検査するセルの指定 302
検証のタイプ 304
実行時の検査 272–280
実行時レコード・チェックリスト，編集 281–286
指定 323–325
使用する TSL 関数 325–329
情報の取得 327
数値の検証 305
数値の範囲の検証 305
接続 327
接続の切断 328
大 / 小文字を区別しスペースを無視検証 305
大小文字を区別しない検証 305
大小文字を区別する検証 304
大 / 小文字を区別せずスペースを無視検証 305
単一フィールドの値を返す 327
単数のカラムを持つデータベースの内容の検証方式 304
チェックポイントの変更 306–316
データ駆動型テストでのデータのインポート 378
データベースチェックポイントウィザード 293–300
テキスト・ファイルへの記録セットの出力 328
標準の検査 286–288
複数のカラムを持つデータベースの内容の検証方式 303
ユーザ定義の検査 289–292
データベース ODBC の最後のオペレーションの最後のエラー・メッセージを返す 328
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 378
Microsoft Query オプション 386
Microsoft Query ファイル，既存 388
Microsoft Query ファイル，新規 387
Microsoft Query の使用 386
SQL ステートメントの指定 389
データベース実行時レコード・チェック 272–280
データベースチェック・ダイアログ・ボックス 291
次を取得できませんメッセージ 293
複雑な値メッセージ 293
データベース・チェック（標準）ソフトキー 287, 288
データベース・チェックポイント
SQL ステートメントのパラメータ化 319
クエリーのパラメータ化 319
失敗したチェックポイントのオプション 271
データベースチェックポイントウィザード 293–300
データベース・チェックリストを共有フォルダに保存 307–309
データベース・チェックリストの編集 309–311
テスト結果 484
内容の検査の期待結果の表示 486
パラメータ化 319–323
パラメータ化，ガイドライン 322
変更 306–316
データベースチェックポイントウィザード

293–300
Data Junction オプションの指定 298
Data Junction 画面 298–300
Data Junction の変換ファイルの選択 299
ODBC/Microsoft Query 画面 294–298
ODBC（Microsoft Query）オプションの指定 294
SQL ステートメントの指定 297
ソース・クエリー・ファイルの選択 296
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
オプション 485
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
次を取得できませんメッセージ 293
データベース・チェックポイントのカスタムチェック・コマンド
ODBC または Microsoft Query を使用する 290
データベース・チェックポイントのクエリーを指定，ODBC/Microsoft Query での作業 323–324
データベース・チェックポイントのパラメータ化 319–323
SQL ステートメント 319
ガイドライン 322
データベース・チェックポイントの標準チェック・コマンド
Data Junction を使用する 288
ODBC または Microsoft Query を使用する 287
データベース・チェックポイントの変換ファイル，Data Junction での作業 324–325
データベース・チェック（ユーザ定義）ソフトキー 290
データベース・チェックリスト
既存クエリーの変更 311–316
共有 307–309
編集 309–311
データベース・チェックリストの編集コマンド 307, 309, 314
データベース・チェックリストの編集ダイアログ・ボックス 310, 312, 315
データベース・チェックリスト編集ダイアログ・ボックス
修正ボタン 313, 316
データベースの検査 267–293
概要 268–270
「データベース」および「データベース・チェックポイント」*参照*
データベースの内容のプロパティ検査 289–292
データベース・フィールドの対応づけ
実行時チェックリストの編集 284
データベースを対象とする標準の検査 286–289
データベースを対象とするユーザ定義の検査 289–292
データ・メニュー・コマンド，データ・テーブル 382
テーブル
1 つの列で構成されるデータベースの内容に対する検証メソッド 215
大文字と小文字を区別しスペースを無視する検証 216
大文字と小文字を区別しない検証 216
大文字と小文字を区別する検証 216
大文字と小文字を区別せずスペースを無視する検証 216
概要 253
期待データの編集 217, 265
検査 253–266
検査するセルの指定 212, 260
検査を指定しての内容の検査 255–257
検証タイプ 216
検証のタイプ 264
数値の検証 216, 264
数値の範囲の検証 216, 264
大 / 小文字を区別しスペースを無視検証 264
大小文字を区別しない検証 264
大小文字の区別する検証 264
大 / 小文字を区別せずスペースを無視検証 264
単一カラムのデータベースの内容の検証方法 263
内容の検査の期待結果の表示 479
内容の検査の結果の分析 476
標準の検査による内容の検査 255
複数カラム・テーブルの検証方式 261
複数の列で構成されるテーブルの検証メソッド 213
テーブルの検査 253–266

概要 253
「テーブル」*参照*
テーマ 571
テキスト
位置の取得 348–349
検査 341–358
検索 348–351
比較 352
読み取り 344–347
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 345
テキスト取得ースクリーン領域からソフトキー 346
テキスト取得のオブジェクト / ウィンドウからコマンド 345
テキスト・チェックポイント 341–358
WinRunner によるフォントの学習 352–358
概要 341–342
テキストの検索 348–351
テキストの比較 352
テキストの読み取り 344–347
フォント・グループの作成 355–356
テキスト認識
オプション 544–546
リスクと代替手段 546
テキストの読み取り 344–347
ウィンドウまたはオブジェクト 344
オブジェクトの領域またはウィンドウの領域 346
テキスト文字列
指定された文字列のクリック 351
ポインタの移動 349–350
テキスト・リンク・プロパティ 188
テキストを指定ダイアログ・ボックス 220, 221
デザイン
テスト 95–127
テスト
圧縮 125
解凍 125
結果の印刷 453
結果のプレビュー 454
チェックポイント 108
テストの一般情報の文書化 502
テストの説明情報の文書化 504
プログラミング 107
テスト・ウィザード「テスト・スクリプト・ウィザード」*参照*
テスト・ウィザード,「テスト・スクリプト・ウィザード」*参照*
テスト・ウィンドウ 16
WinRunner 97
テスト・オプション 439
テスト・オプションの設定
グローバル 519–577
テスト間での GUI マップ・ファイルの共有 49–50
テスト結果 443–496
GUI チェックポイント 463, 473
Quality Center プロジェクト・データベースからの表示 465–467
WinRunner レポート・ビュー 457
期待結果の更新 489
チェックポイントの結果 471
データベース・チェックポイント 484, 486
テーブル 476
統一レポート・ビュー 445
ビットマップ・チェックポイント 463, 483
表示，概要 464–467
ファイルの比較 463
不具合，報告 493
プロパティ検査 473
テスト結果ウィンドウ 457–464, 465
テスト・サマリ 461
テスト・ツリー 460
テスト・ログ 463
表示ボタン 489
テスト結果ウィンドウからの不具合の報告 493
テスト結果の表示
テスト結果の印刷 453
テスト結果のプレビュー 454
テスト結果の表示，カスタマイズ 456
テスト結果のプレビュー 454
テスト結果を開く，統一レポート 467
テスト工程
概要 5
結果の分析 443–496
テストの実行 423–442
テストごとの GUI マップ・ファイル・モード 67–71

GUI マップ・ファイルの更新 70
概要 67–68
テスト，作成
既存のファイルを開く 116
記録 99–105
新規 116
同期化ポイント 109
編集 115
テスト・サマリ 461
テスト実行
実行，テストも*参照*
結果の表示 464–467
テスト実行時の GUI における変更の検出，「実行ウィザード」*参照*
テスト実行ダイアログ・ボックス 425, 433, 437
テスト実行中に見つかった GUI オブジェクトにおける変更，「実行ウィザード」*参照*
テスト情報 502
テスト・スクリプト 16, 97
テスト・スクリプト・ウィザード
アプリケーションの GUI の学習 50–56
テスト・ウィザード，「テスト・スクリプト・ウィザード」*参照*
テスト・ツールバー 14, 18
テスト・ツリー 460
テスト，デザイン 95–127
テスト特有の GUI マップ・ファイル・モード 525
テスト特有の GUI マップ・ファイル・モード
オプションの設定 68–70
ガイドライン 71
テスト特有の GUI マップ・ファイル・モードのオプション 68–70
テストの記録
WinRunner を最小化 106
アナログ・モード 104
ガイドライン 112
コンテキスト・センシティブ・モード 99–101
テストの結果，「テスト結果」*参照*
テストの実行 423–442
アプリケーションの検査 432
概要 423–424
期待結果の更新 435
グローバル・テスト・オプションの設定 548–563
構成設定パラメータでの制御 439
実行モード 423
テスト・オプションでの制御 439
テスト・スクリプトのデバッグ実行 434
問題 440–442
テストの設定
現在 511
現在，テストのプロパティ・ダイアログ・ボックス
åªç›ÇÃÉeÉXÉgÅEÉ^Éu 510
テストの説明情報 504
テストの同期化 407–419
ヒント 419
テストのプロパティ 499–518
テストのプロパティ・ダイアログ・ボックス
General タブ 391
アドイン 508
一般タブ 502
現在のテスト・タブ 510
実行タブ 512
説明タブ 504
パラメータ・タブ 505
テストのプロパティの設定 500
アドイン 508
パラメータ 505
テストの編集 115
テストの保存
TestDirector プロジェクトに 119
ファイル・システム 117
テスト・パラメータ 505
テスト・ログ 463
テストを開く 116
TestDirector プロジェクト 122
ファイル・システム 121
テストを開くダイアログ・ボックス 121
テストを保存ダイアログ・ボックス 118
デバッグ結果 425, 434
デバッグ・ツールバー 18
デバッグ・ビューア表示枠 14
デバッグ・モード 423, 425, 434

と

統一レポート 445
Quality Center への接続 469
結果の検索 451

結果の絞り込み 452
テスト結果を開く 467
メニュー・バーとツールバー 447
統一レポート・ビュー，定義 444
同期化
ウインドウの待機 409–410
オブジェクトの待機 409–410
オブジェクトやウィンドウのビットマップの待機 415–417
スクリーン領域のビットマップの待機 417–419
プロパティ値の待機 410–414
同期化ポイント 109
データ駆動型テスト 393–398
閉じるコマンド 127
データ • テーブル 381
データ・テーブル 381
トランザクション 109
ドロップダウン・ツールバー，記録 107

な

名前，「論理名」*参照*
名前を付けて保存コマンド 117
データ・テーブル 380
名前を付けて保存ボタン
GUI チェックポイント作成ダイアログ・ボックス内 157
GUI チェックリスト編集ダイアログ・ボックス内 161

に

二重引用符，GUI マップ・ファイル 77
入力パラメータ 438, 505

は

パーセント・コマンド，データ・テーブル 383
背景の画像ダイアログ・ボックス 573
バグ，「不具合」*参照*
場所
期待結果フォルダ 511
現在の作業フォルダ 511
検証結果フォルダ 511
パラメータ
出力 505
テストに対する定義 505, 506
テストのための管理 505
入力 505
パラメータ・タブ，テストのプロパティ・ダイアログ・ボックス 505
貼り付けコマンド 115
データ・テーブル 381

ひ

比較引数の指定ダイアログ・ボックス 170
引数，指定
Compare プロパティ検査での 170
引数の指定 169–175
DateFormat プロパティ検査 171
Range プロパティ検査 172
RegularExpression プロパティ検査 172
TimeFormat プロパティ検査 173
引数を指定ダイアログ・ボックスから 173
引数のチェック・ダイアログ・ボックス
DateFormat プロパティ検査 171
Range プロパティ検査 172
Regular Expression プロパティ検査 172
TimeFormat プロパティ検査 173
引数を指定ダイアログ・ボックス 173
引数を指定ボタン 169–175
左方向へコピー・コマンド，データ・テーブル 382
ビットマップ
テスト実行中にキャプチャ 334
ビットマップ・チェックポイント
失敗したチェックポイントのオプション 334
331–339
ウィンドウとオブジェクト 336–338
概要 331–333
画面領域 338–339
結果の表示 483
コンテキスト・センシティブ 336–338
データ駆動型テスト 393–398
データ駆動テストで 333
テスト結果 463
ビットマップ・チェックポイント・コマンド 336–339
ビットマップ同期化ポイント
オブジェクトやウィンドウ 415–417
スクリーン領域 417–419
ビットマップの検証，「ビットマップ・チェッ

クポイント」*参照*
ビットマップの同期化ポイント
データ駆動型テスト 419
非標準プロパティ 158, 162
表示
GUI オブジェクトのプロパティ 36–41
表示ボタン，WinRunner テスト結果ウィンドウ 318
表示ボタン，テスト結果ウィンドウ 489
標準以外のプロパティのみ表示ボタン
GUI 検証結果ダイアログ・ボックス 475
GUI チェック・ダイアログ・ボックス 155
GUI チェックポイントを作成ダイアログ・ボックス 158
GUI チェックリストを編集ダイアログ・ボックス 162
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 486
標準コマンド，データ・テーブル 383
標準設定データベース・チェックポイント・ボタン 287, 288
標準ツールバー 17
標準データベースチェックポイント・ボタン 20
標準のオブジェクト
標準の検査 163–168
プロパティ検査 163–168
標準の検査
ウィンドウ内のすべてのオブジェクト 140
単数の GUI オブジェクトの検査 134–135
標準のオブジェクトで 163–168
標準のデータベース検査
Data Junction 288–289
ODBC/Microsoft Query を使った 287–288
標準プロパティ 158, 162
標準プロパティのみ表示ボタン
GUI チェックリストを編集ダイアログ・ボックス 162
GUI 検証結果ダイアログ・ボックス 475
GUI チェック・ダイアログ・ボックス 154
GUI チェックポイントを作成ダイアログ・ボックス 158
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 485
開くコマンド
データ・テーブル 380
開くボタン
GUI チェックポイント作成ダイアログ・ボックス内 157
GUI チェックリスト編集ダイアログ・ボックス内 161

ふ

ファイル・ツールバー 17
ファイルの管理 116
ファイルの比較
結果の表示 490
テスト結果 463
ファイル・メニュー・コマンド，データ・テーブル 380
フィルタ，GUI マップ・エディタ 91
フィルタ・ダイアログ・ボックス（GUI マップ・エディタ）91
フォルダ・オプション 528
フォント
WinRunner による学習 352–358
学習 353–354
フォント・エキスパート 353
フォント・グループ
作成 355–356
定義 353
アクティブにする 357
フォント・グループ・ダイアログ・ボックス 355
フォントの学習ダイアログ・ボックス 354
フォント・ライブラリ 353
不具合
テスト結果ウィンドウからの報告 493
テスト実行中の報告 495
不具合の追加ダイアログ・ボックス 493
設定 493
不具合，報告 494
複雑な値メッセージ
GUI チェックポイント・ダイアログ・ボックス内 151

データベースチェックポイント・ダイアログ・ボックス 293
複数オブジェクトの GUI チェックポイント・コマンド 137, 144, 156
複数オブジェクトの GUI チェックポイント・ボタン 137, 144, 156
複数のオブジェクトの GUI チェックポイント・ボタン 20
「複数のオブジェクト用の GUI チェックポイント・コマンド」*参照*
物理的記述
修正 79–81
正規表現の変更 84
定義 29–30
フレーム・オブジェクト・プロパティ 187
フローティング・ツールバー 17
プロパティ
テスト 499–518
テストの設定 500
プロパティ検査
テスト結果 473
引数の指定 169–175
標準のオブジェクトで 163–168
プロパティ値の検査 132–134
プロパティ値
同期化ポイント 410–414
編集 175–177
プロパティのチェック・ダイアログ・ボックス 133
プロパティ・ビューア（ActiveX コントロール）229–232
プロパティ・リスト・ボタン 154, 158, 162
分数コマンド，データ・テーブル 383

へ

変更
GUI チェックポイントの期待結果 177–179
GUI チェックリスト 145–150
変更，ウィンドウ・ラベル 82
編集
GUI チェックリスト 146–150
データベース・チェックリスト 309–311
プロパティの期待値 175–177
編集メニュー・コマンド，データ・テーブル 381

ほ

ポインタの値，ActiveX 41
保存
GUI マップ・ファイル 59–61
一時 GUI マップ・ファイル 59
保存コマンド 117
保存ボタン 117

ま

前を検索コマンド 116
マップされていないクラス，「object クラス」*参照*

み

右方向へコピー・コマンド，データ・テーブル 382

め

メイン・データ・テーブル 391
メッセージ
GUI チェックポイントのダイアログ・ボックス 151
データベースチェックポイント・ダイアログ・ボックス 293
メニュー・バー，WinRunner 14
メニュー形式のツールバー，記録 107

も

元に戻すコマンド 115
問題
コンテキスト・センシティブ・テストの記録 102–104
コンテキスト・センシティブ・テストの実行 440–442

や

矢印から実行コマンド 429
矢印から実行ボタン 429
やり直しコマンド 115

ゆ

ユーザ設定数値コマンド，データ・テーブル 384

ユーザ定義オブジェクト 103
ユーザ定義関数
　データ駆動型テストでのパラメータ化 368
ユーザ定義記録関数 103
ユーザ定義クラス 103
ユーザ定義実行関数 103
ユーザ定義ツールバー 14, 19
ユーザ定義のプロパティ 155, 158, 162, 475
ユーザ・プロパティ 475
ユーザプロパティ 155, 158, 162
ユーザ・プロパティのみ表示ボタン
　GUI 検証結果ダイアログ・ボックス 475
ユーザプロパティのみ表示ボタン
　GUI チェック・ダイアログ・ボックス 155
　GUI チェックポイントを作成ダイアログ・ボックス 158
　GUI チェックリストを編集ダイアログ・ボックス 162

よ

呼び出し先のテスト
　実行タブの設定 512
読み込み，アドイン 508

ら

ラジオ・ボタン・プロパティ 190
ラベル，変更 82

ろ

ロード関数 430
論理名
　Web オブジェクト，プロパティの設定 186
　修正 66, 79–81
　変更 46
　定義 31