# Reports for WinForms

2018.04.10 更新

グレープシティ株式会社

# 目次


はじめに 10-11
コンポーネントをプロジェクトに組み込む方法 11-12
ファイルの配布について 12
製品の概要 13
製品構成 13-14
更新履歴 14
5.0J 14
新機能 14-17
機能の改善 17-20
仕様変更 20
クラスメンバ 20-25
4.0J C1Report 25
新機能 25-29
クラスメンバ 29-30
4.0J（C1PrintDocument） 30
新機能 30-41
クラスメンバ 41-55
3.0J（C1Report） 55
新機能 55-59
クラスメンバ 59-60
移行情報 60
ActiveX製品からの移行 60
ActiveX製品とのメンバ比較表 60-66
旧Windowsフォーム製品からの移行 66-67
主な特長 68-74
クイックスタート 75
レポート生成方法 75
レポート生成方法の違い 75-76
レポート生成方法の選択 76
レポート作成を開始する 76-77
C1Report クイックスタート 77



Copyright © GrapeCity, Inc. All rights reserved.

手順1：レポート定義の作成 77-82
手順2：レポートの修正 82-84
手順3：C1Report コンポーネントへのレポートのロード 84-85
手順4：レポートの表示 85-86
VBScript 式を作成する 86-87
値に応じてフィールドを書式設定する 87-88
データが存在しないセクションを非表示にする 88-89
ページカウンタをリセットする 89-90
印刷プレビューを開始する 90
C1PrintDocument クイックスタート 90
手順1:フォームへの C1Preview コントロールの追加 90-91
手順2:フォームとコントロールの設定 91-92
手順3:プロジェクトへのコードの追加 92-93
手順4：プログラムの実行 93-96
テーブルを作成する 96
単純なテーブルを作成する 96-99
3行3列のテーブルの作成 99-102
テーブルセルへのテキストの追加 102-105
テーブルの特定のセルへの2つの画像の追加 105-108
テーブル内の行と列の周囲の境界線の作成 108-111
テーブル内の特定のセルの背景色の作成 111-114
文字列を追加する 114
ドキュメントに段落を追加する 114-116
ドキュメント内のテーブルの下にテキストを追加する 116-118
C1PrintDocument 内のテキストのフォントとスタイルを変更する 118-120
C1PrintDocument 内のページヘッダーの作成 120
3つの部分から成るページヘッダーを作成する 120-122
ページヘッダーへの背景色の追加 122-124
水平方向の直線の描画 124-125
C1PrintDocument 内にページフッターを作成する 125-129
設計時サポート 130
C1Report タスクメニュー 130-131
C1Report コンテキストメニュー 131


Copyright © GrapeCity, Inc. All rights reserved.

C1RdlReport タスクメニュー 131-132
C1RdlReport コンテキストメニュー 132-133
C1MultiDocument タスクメニュー 133-134
C1MultiDocument コンテキストメニュー 134
C1PrintPreviewControl タスクメニュー 134-135
スマートデザイナ 135-137
[メインメニュー]フローティングツールバー 137-141
[ツールストリップ]フローティングツールバー 141-143
[サムネール]フローティングツールバー 143-144
[アウトライン]フローティングツールバー 144-145
[ルーラー]フローティングツールバー 145-146
[プレビューペインの外観]フローティングツールバー 146-147
[マージン]フローティングツールバー 147
[プレビューペイン]フローティングツールバー 147-149
[テキスト検索パネル]フローティングツールバー 149-150
ローカライズ 150-151
ローカライズツールバー 151-155
C1Report の使い方 156
オブジェクトモデル 156-157
レポートのセクション 157-159
クライアントアプリケーションの開発 159
組み込みレポート(設計時に読み込み) 159-160
組み込みレポート(設計時に作成) 160-161
実行時に読み込まれるレポート 161-164
実行時にカスタマイズされるレポート 164-165
Web アプリケーションの開発 165
静的 Web レポート 165-166
動的 Web レポート 166-172
レポートの作成、読み込み、描画 172-173
レポート定義の作成 173-179
レポートデータの読み込み 179
データベースから読み込む 179-181
ストアドプロシージャからのデータのロード 181-183


Copyright © GrapeCity, Inc. All rights reserved.

DataTable から読み込む 183-184
カスタムデータソースから読み込む 184
グループ化とソート 184
データのグループ化 184-188
データをソートする 188-189
レポートへ積算合計を追加する 189-190
小計などの集計の追加 190
クロスタブ型レポートを作成する 190-191
VBScript 式 191-194
オブジェクトと変数 194-197
互換性のための関数 197-199
集計関数の使い方 199-200
イベントの使用 200-201
値に応じてフィールドを書式設定する 201-202
データが存在しないセクションを非表示にする 202-203
値に応じてフィールドを表示または非表示にする 203-205
ユーザーにパラメータの入力を求める 205-206
ページカウンタをリセットする 206-207
フィールドのサイズを変更して棒グラフを作成する 207-208
便利な機能の活用 208-209
パラメータクエリー 209-210
アンバウンドレポート 210-212
データセキュリティ 212-213
Windows NT 統合セキュリティ 213
パスワードを含む接続文字列 213-214
アプリケーション定義のエイリアス 214-215
カスタムデータセット 215
SSRS レポートの ComponentOne レポート 215-217
C1RdlReport の使い方 218
レポート定義言語(RDL) 218
C1RdlReport のメリット 218
C1RdlReport の制限 218-219



Copyright © GrapeCity, Inc. All rights reserved.

RDL ファイルのロード 219-220
C1Report デザイナの使い方 221
C1Report デザイナについて 221-222
[ファイル]メニュー 222-223
デザインモード 223
「デザイン」タブ 223-227
「挿入」タブ 227-228
「レイアウト」タブ 228-230
「ページ設定」タブ 230
「プレビュー」モード 230-232
「スタイルギャラリー」ダイアログボックス 232-234
Visual Studio からの C1Report デザイナへのアクセス 235-236
C1Report デザイナオプションの設定 236-240
基本的なレポート定義ファイルの作成 240-245
レポートレイアウトの変更 245-247
フィールドによるレポートの強化 247-248
グラフフィールドの追加 248-252
グラデーションフィールドの追加 252-255
フィールドの選択、移動、およびコピー 255-256
プロパティの変更 256
データソースの変更 256-257
サブレポートを使用したマスター／詳細レポートの作成 257-260
レポートのプレビューと印刷 260-262
レポートのエクスポートと配布 262-263
レポート定義ファイルの管理 263
Microsoft Access レポートのインポート 263-267
Crystal Reports レポートのインポート 267-268
Reports のグラフ作成 268-269
グラフ種類 269-272
その他の便利なグラフプロパティ 272-275
複数の系列を含むグラフ 275
グループ化したレポートのグラフ 275-276
グラフのデータプロット 276-277


Copyright © GrapeCity, Inc. All rights reserved.

集計グラフ 277-279
レポート内のマップ 279-280
レイヤ 280
トラッキング 280
スタイル 280-281
空間位置 281
ポイントレイヤ 281
ラインレイヤ 281-282
KMLレイヤ 282
凡例 282-283
マップのウォークスルー 283-285
レポート内バーコードの使用 286
バーコードシンボル 286-291
バーコードのプロパティ 291-293
レポートとドキュメントビューアの概要 294-295
[ファイル]メニュー 295-296
[表示]タブ 296
[ファイル]グループ 296-297
[ページ]グループ 297
[ズーム]グループ 297
[移動]グループ 297-298
[ツール]グループ 298
ReportPreview コントロールの概要 299-300
C1ReportsScheduler の使い方 301
C1ReportsScheduler について 301
インストールとセットアップ 301-302
ユーザーインタフェース 302
キャプションとステータスバー 302-303
タスクリスト 303-305
アクションリスト 305-306
スケジュール 306-307
メニューシステム 307-311
C1PrintDocument の使い方 312




Copyright © GrapeCity, Inc. All rights reserved.

描画オブジェクト 312-313
描画オブジェクトの階層 313-314
描画オブジェクトの包含、配置、スタック規則 314
表示領域 314-315
スタック規則 315
描画オブジェクトのサイズと位置の指定 315-318
表示オブジェクトの相対位置決め方式の例 318-322
描画オブジェクトの影 322-323
オブジェクトの枠線 323
スタイル 323
Style プロパティを持つクラス 323-324
インラインスタイルと非インラインスタイル 324-325
アンビエントスタイルプロパティと非アンビエントスタイルプロパティ 325
スタイルの継承 - Parent と AmbientParent 325-326
スタイルプロパティとそのデフォルト値 326-327
複雑なスタイルプロパティのサブプロパティ 327
計算されたスタイルプロパティ 327-328
段落オブジェクトのスタイル 329
テーブルのスタイル 329
テーブル 329-330
セル、列、および行へのアクセス 330-331
テーブルと列の幅、行の高さ 331-332
行と列のグループ - ヘッダーとフッター 332-335
ユーザーセルグループ 335
テーブル内のスタイル 335-336
アンカーとハイパーリンク 336
同じドキュメント内のアンカーへのハイパーリンクを追加する 336-337
別のC1PrintDocument内のアンカーへのハイパーリンクを追加する 337-338
現在のドキュメント内の場所へのハイパーリンクを追加する 338-339
外部ファイルへのハイパーリンクを追加する 339-340
同じドキュメント内のページへのハイパーリンクを追加する 340-341
ユーザーイベントへのハイパーリンクを追加する 341-342



Copyright © GrapeCity, Inc. All rights reserved.

リンクターゲットクラスの階層 342
式、スクリプト、タグ 342-343
タグ 343-344
タグ／式の構文 344-345
実行時のタグ値の編集 345
すべてのタグの表示 345-346
特定のタグの表示 346-347
[タグ]ダイアログボックスを表示する場合の指定 347-349
デフォルトの[タグ]ダイアログボックスの定義 349
スクリプト／式の言語 349-350
アセンブリと名前空間 350-351
テキスト式でアクセス可能な ID 351-356
フィルタ、グループ化、およびソート内で式にアクセス可能な ID 356
DataSet 内の集計フィールドの指定に使用する式でアクセス可能な ID 356
データ連結 356
レンダリングオブジェクトのデータ連結 356-358
RenderTable によるデータ連結の使用 358-360
データ連結のサンプル 360
グループの使用 360-362
集計の使用 362-364
データ集計 364-367
目次 367
用語索引 367-368
索引生成機能をサポートするクラス 368
コードによる索引の生成 368
索引の外観のカスタマイズ 368-369
索引のスタイルの階層 369-370
生成される索引の構造 370-371
アウトライン 371
埋め込みフォント 371-372
フォントの代替 372
選択フォントの埋め込み 372-373
辞書 373




Copyright © GrapeCity, Inc. All rights reserved.

C1Report 定義 373-374
C1Report インポートの制限 374-375
プリンタドライバでの作業 375-377
レポート定義言語(RDL)ファイル 377-378
RDL インポートの制限 378-379
C1MultiDocument の使い方 380
C1MultiDocument の制限 380-381
C1MultiDocument ファイルの作成とプレビュー 381-382
C1MultiDocument ファイルのエクスポート 382
C1MultiDocument ファイルの印刷 382-383
C1MultiDocument のアウトライン 383


Copyright © GrapeCity, Inc. All rights reserved.

# はじめに

**Reports for WinForms** では、レポート／ドキュメントの作成、書式設定、プレビュー印刷、管理、およびエクスポート機能をすばやくかつ簡単に .NET アプリケーションに追加できます。

**Reports for WinForms** は、2つの旧製品の **True WinReports for .NET** と **Preview for .NET** を組み合わせたものです。旧製品のすべての機能は保持されていますが、アセンブリ名と名前空間が変更されています。

> **メモ**: Reports for WinForms は ComponentOne Enterprise／for WinForms でのみ開発に利用できます。その他の製品では Reports for WinForms を開発に利用することはできませんので、ご注意ください。

## レポート

**Reports for WinForms** では、.NET アプリケーションで Microsoft Access スタイルのレポートをすばやくかつ簡単に生成します。

- データベース化され、結合されたレポートを生成する **C1Report** コンポーネント。レポートをプリンタまたはプレビューコントロールに直接表示するか、さまざまな移植可能な形式（XLS、PDF、HTML、テキスト、および画像を含む）にエクスポートします。コンポーネントは、レポート定義を作成、カスタマイズ、ロード、および保存するための充実したオブジェクトモデルを表示します。詳細については、「C1Report の使い方 」を参照してください。
- **C1RdlReport** コンポーネント。2008 バージョンの RDL 仕様を使用して定義された RDL（レポート定義言語）レポートを表すコンポーネントです。C1RdlReport コンポーネントは、RDL サポートを追加した C1Report コンポーネントに似ています。詳細については、「C1RdlReport の使い方 」を参照してください。
- コードを記述することなく、レポート定義を作成するために使用されるスタンドアロンアプリケーションである **C1ReportDesigner**。デザイナを使用して、レポート定義をすばやく作成および編集したり、既存の Microsoft Access と Crystal レポート定義をインポートしたりできます。デザイナは、Microsoft Access インタフェーイスを模倣しています。したがって、現在 Microsoft Access を使用している場合、**C1ReportDesigner** の使用にすばやく適応します。詳細については、「C1Report デザイナの使い方」を参照してください。
- **C1ReportDesigner** コントロール（スタンドアロンのデザイナと混同しないこと）は、レポートを設計モードで表示して、ユーザーがレポートのフィールドやセクションをドラッグ、コピー、サイズ変更できるようにします。このコントロールには、無制限の元に戻す／やり直しスタックに加えて、Visual Studio に付属する **PropertyGrid** コントロールと組み合わせて使用できるように設計された選択メカニズムも用意されています。C1ReportDesigner コントロールを使用して、アプリケーションにレポート設計機能を組み込んだり、完全な機能を備えた独自のレポートデザイナアプリケーションを記述できます。
- **C1ReportsScheduler** アプリケーションは、レポートの作成がバックグラウンドで実行されるようにスケジュールするスタンドアロンアプリケーションです。**C1ReportsScheduler** アプリケーションを使用して、エクスポートまたは印刷するレポート、レポートのエクスポート先の形式、およびレポート作成のスケジュールや頻度を選択できます。詳細については、「C1ReportsScheduler の操作」を参照してください。

## 印刷とプレビュー

**Reports for WinForms** では、印刷要件がいかに単純または複雑なものであっても、印刷とプレビュー機能をプロジェクトにすばやくかつ簡単に追加できます。

- **C1PrintDocument** コンポーネントには、任意の、複雑なドキュメントをコードで作成できる充実したオブジェクトモデルが含まれています。オブジェクトモデルは、特にページ付けされたドキュメントを対象とし、複雑な、構造化されたドキュメントの自動かつインテリジェントなページネーションを容易にする一連の豊富な機能を提供します。ドキュメントは、完全にコードで作成するか、または強力かつ柔軟なデータバインディングモデルを介してデータベースにバインドすることができます。C1PrintDocument は、レポート定義をインポートしたり、生成したりすることもできます。詳細については、「C1PrintDocument の使い方」を参照してください。
- **C1MultiDocument** コンポーネントは、メモリ制限のため単一の C1PrintDocument オブジェクトで処理できない大容量のドキュメントを作成、保持、エクスポートできるように設計されています。C1MultiDocument は、含まれるドキュメント間のリンク、共通 TOC、共通ページ番号、および総ページ数をサポートします。
- **Reports for WinForms** の視覚的なプレビューコンポーネントは、ドキュメント表示機能をアプリケーションにすば


Copyright © GrapeCity, Inc. All rights reserved.

やく追加できる、パワフルで、柔軟性にすぐれた、使いやすい一連のツールを備えています。統合されたコンポーネント（**C1PrintPreviewControl** コンポーネントと **C1PrintPreviewDialog** ダイアログボックス）が、アプリケーションへのプロフェッショナルなプレビューの追加を簡単に行えるようにする一方で、一連の専用コントロール（**C1PreviewPane**, **C1PreviewThumbnailView**, **C1PreviewOutlineView**, **C1PreviewTextSearchPanel**）は、プレビューの必要に応じた微調整を可能にします。

- WinForms の **ReportPreview** コントロールは、2015 v2 リリースでベータ版としてリリースされた新しいリボンベースのプレビューコントロールです。さまざまなソースから取得した複数のドキュメントまたはレポートを 1 つのビューアに表示する機能を提供します。最新の設計トレンドであるリボンベースの UI を備えており、リボン要素、グループ、サイドバーなどを簡単に使用できるため、これまで複雑だった実装が簡略化されています。詳細については、「ReportPreview コントロールの概要」を参照してください。

# コンポーネントをプロジェクトに組み込む方法

## コンポーネントの組み込み

Visual Studio では、ツールボックスにコンポーネントを追加しただけでは、プロジェクトにコンポーネントを追加したことにはなりません。プロジェクトの参照設定へ追加された時点でコンポーネントが組み込まれます。

以下のいずれかの操作を行うとプロジェクトへコンポーネントが組み込まれます。

1. フォームにコンポーネントを配置する
2. ソリューションエクスプローラ上で参照の追加を行う

プロジェクトに組み込まれているコンポーネントの一覧は、ソリューションエクスプローラで確認できます。また、各コンポーネントが使用している DLL もソリューションエクスプローラに登録される場合があります。詳細については、Visual Studio の製品ヘルプを参照してください。

本製品で使用しているコンポーネントの一覧を以下に示します。

| ファイル | 内容 |
|---|---|
| C1.C1Report.4.dll | 本体アセンブリ |
| C1.C1Report.CustomFields.4.dll | 本体アセンブリ |
| C1.Win.C1Report.4.dll | 本体アセンブリ |
| C1.Win.C1ReportDesigner.4.dll | 本体アセンブリ |
| C1.C1Report.4.Design.dll | C1Report デザイナアセンブリ |
| C1.Win.C1Report.4.Design.dll | C1Report デザイナアセンブリ |
| C1.Win.C1ReportDesigner.4.Design.dll | C1Report デザイナアセンブリ |

※ .NET Framework 4 以上でご利用いただけます。

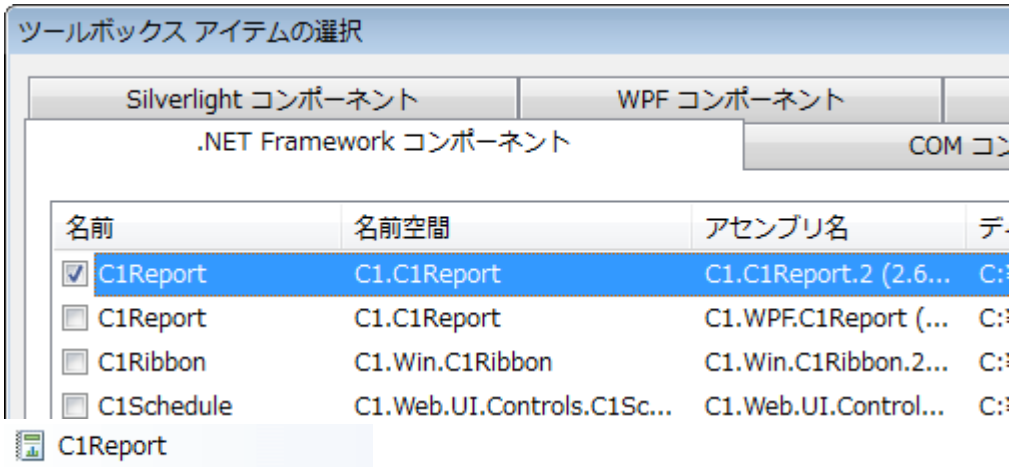
### フォームにコンポーネントを配置する方法

以下に、C1Sizer コントロールをツールボックスに追加し、フォームに配置する方法を示します。これにより、コンポーネントがプロジェクトに組み込まれます。

**配置手順**

1. [ツール]メニューから[ツールボックス アイテムの選択]を選択します。
2. [ツールボックス アイテムの選択]ダイアログの「.NET Framework コンポーネント」タブを選択します。
3. 使用するコンポーネントのチェックボックスを ON にして〈OK〉ボタンをクリックしてください。ツールボックスに指定したコントロールのアイコンが表示されます。


Copyright © GrapeCity, Inc. All rights reserved.




4. ツールボックスから指定したコントロールのアイコンを選択してフォームに配置します。ソリューションエクスプローラの参照設定に指定したコントロールの名前空間が追加されます。

## ファイルの配布について

Ribbon for WinForms のランタイムファイルは、C:\Program Files (x86)\ComponentOne\WinForms\Bin\v4 フォルダにインストールされる次のファイルです。

| ファイル | 内容 |
|---|---|
| C1.C1Report.4.dll | 本体アセンブリ |
| C1.C1Report.CustomFields.4.dll | 本体アセンブリ |
| C1.Win.C1Report.4.dll | 本体アセンブリ |
| C1.Win.C1ReportDesigner.4.dll | 本体アセンブリ |

※ .NET Framework 4 以上でご利用いただけます。

**注意:**以下のファイルは開発環境のみに必要なファイルです。配布しないように注意してください。

- C1.C1Report.4.Design.dll
- C1.Win.C1Report.4.Design.dll
- C1.Win.C1ReportDesigner.4.Design.dll


Copyright © GrapeCity, Inc. All rights reserved.

# 製品の概要

**Reports for WinForms** では、**C1Report**、**C1PrintDocument**、および6種類のプレビューコントロールの数多くの機能をを活用して、カスタマイズされたレポートを作成することができます。

# 製品構成

機能は旧バージョンの製品に似ていますが、**C1PrintDocument** コンポーネントと印刷プレビューコントロールの両方のオブジェクトモデルが大幅に変更され、そのため、新しいコンポーネントは、古いコンポーネントとバイナリまたはコードでの互換性がないことに注意してください。

Reports for WinForms は、以下のアセンブリで構成されています。

## C1.C1Report.2.dll

C1.C1Report.2.dll は、C1PrintDocument および**C1Report** コンポーネントとエクスポートなどの非 Windows Forms 固有のサービスを提供するその他のパブリッククラスを備えています。このアセンブリには、以下のコンポーネントが含まれています。

- **C1Report**

  C1Report コンポーネントは、データに基づくバンドレポートを生成します。レポートをプリンタまたはプレビューコントロールに直接表示するか、さまざまな移植可能な形式（XLS、PDF、HTML、テキスト、および画像を含む）にエクスポートします。コンポーネントは、レポート定義を作成、カスタマイズ、ロード、および保存するための充実したオブジェクトモデルを表示します。

- **C1PrintDocument**

  C1PrintDocument コンポーネントを使用して、印刷、プレビュー、ディスクファイルへの保存を行ったり、PDF（Portable Document Format）および RTF（Rich Text File）を含む多くの外部形式にエクスポートしたりできる複雑なドキュメントを作成できます。

- **C1MultiDocument**

  **C1MultiDocument** コンポーネントは、メモリ制限のため単一のC1PrintDocumentオブジェクトで処理できない大容量ドキュメントを作成、保持、およびエクスポートできるように設計されています。

- **C1RdlReport**

  **C1RdlReport** により、Microsoft Reporting Services として SQL サーバーデータだけでなく、任意のデータソース（.mdb ファイルなど）を使用できる RDL レポートを生成できます。C1RdlReport コンポーネントは RDL 2008 仕様に基づく RDL ファイルをサポートします。

このアセンブリは、他の ComponentOne DLL ファイルを参照しません。このアセンブリのほとんどのクラスは、C1.C1Preview、C1.C1Report、および C1.C1Rdl.Rdl2008 名前空間内にあります。

## C1.Win.C1Report.2.dll

C1.Win.C1Report.2.dll は、C1PrintDocument および以下の他のドキュメントタイプで使用できる Windows Forms コントロールを提供します。

- **C1PreviewPane**

  プレビューペイン。このコントロールでは、プレビューするドキュメントのページを表示し、パン、ズーム、その他のプレビュー操作を行うことができます。フォームデザイナでは、コンテキストメニュー項目を介して、現在のフォーム上で標準


Copyright © GrapeCity, Inc. All rights reserved.

ツールバーとステータスバーを作成できます。

- **C1PrintPreviewControl**

  統合された印刷プレビューコントロール。プレビューペイン、操作に関連する標準プレビューを備えるツールバー、サムネイルとアウトラインページを含むナビゲーションパネル、および拡張／縮小が可能なテキスト検索パネルが含まれています。

- **C1PrintPreviewDialog**

  ネストされた印刷プレビューコントロールを持つダイアログボックスフォーム。

- **C1PreviewThumbnailView**

  ナビゲーションページサムネイルを表示するために、プレビューペインに（PreviewPane プロパティを使用）アタッチできるパネル。

- **C1PreviewOutlineView**

  ドキュメントのアウトラインを表示するために、プレビューペインに（PreviewPane プロパティを使用）アタッチできるパネル。

- **C1PreviewTextSearchPanel**

  ドキュメントのテキスト検索を実行するために、プレビューペインに（PreviewPane プロパティを使用）アタッチできるパネル。

このアセンブリは、C1PrintDocument を参照します。このアセンブリのほとんどのクラスは、C1.Win.C1Preview 名前空間内にあります。

### 含まれるアプリケーション

含まれるレポーティングコンポーネントおよびコントロールに加えて、Reports for WinForms には、C1ReportDesigner のスタンドアロンアプリケーションも用意されています。

- **C1ReportDesigner**

  **C1ReportDesigner** アプリケーションは、C1Report レポート定義ファイルの作成と編集に使用するツールです。デザイナでは、C1Report コンポーネントによって読み取ることができるファイル（XML）を作成、編集、ロード、および保存することができます。

この製品は、.NET バージョン 2.0 以降が必要です。

# 更新履歴

このセクションでは、旧バージョンのReports for Windows Forms および旧製品である True WinReports for .NET、Preview for .NET の更新履歴について説明します。

現行バージョンの機能の改善、不具合の修正、対応環境の追加に関する情報については、リリースノートを参照してください。

## 5.0J

このセクションでは、Reports for WinForms 5.0J の更新履歴について説明します。

## 新機能


Copyright © GrapeCity, Inc. All rights reserved.

Reports for WinForms 5.0J では、次の新しい機能が追加されました。

## レポートと印刷プレビューの統合

**Reports for WinForms 5.0J** では、**True WinReports for .NET 4.0J** と **Preview for .NET 4.0J** のコンポーネントが統合されました。4.0J とは機能的に互換性が保たれていますが、アセンブリ名と名前空間が変更されました。

- **新しいアセンブリ**

  **C1Report** アセンブリ名が変更されました。古い **C1.Win.C1Report.2** アセンブリの既存の参照は、新しい **C1.C1Report.2** アセンブリの参照に置き換える必要があります。

- **新しい名前空間**

  C1Report 名前空間が変更されました。**C1.Win.C1Report** の代わりに、**C1.C1Report** を使用してください。たとえば、**C1Report** コンポーネントの完全名が **C1.Win.C1Report.C1Report** から **C1.C1Report.C1Report** に変更されました。

- **移行ツール**

  **Convert2Report.exe** ツールにより、**Preview for .NET** および **True WinReports for .NET** を使用するプロジェクトを **Reports for WinForms 5.0J** に簡単に変換できます。

## PDF フォーム

プレビューフォームを PDF フォームにエクスポートできるようになりました。

以前のバージョンでは、RenderInput* オブジェクト／コントロールは単に画像として PDF にエクスポートされていましたが、今回のビルドから以下のようなフォームフィールドとしてエクスポートされます。

| オブジェクト／コントロール | フォームフィールド |
|---|---|
| **RenderInputCheckBox** | CHECKBOX |
| **RenderInputComboBox** | COMBO |
| **RenderInputRadioButton** | RADIOBUTTON |
| **RenderInputText** | TEXT |
| **RenderInputButton** | PUSHBUTTON |

表示入力フィールドの現在の値は、対応するフィールドのデフォルト値として PDF に保存されます。利用可能なコンボボックス値のリストは、ラジオボタングループと同様に保存されます。

作成した PDF ファイルでは、PDF フォームに記入した値を保存することはできません。

## 角丸の枠線

**C1PrintDocument** の表示オブジェクトで、丸みのある角の枠線を使用できるようになりました。詳細については、**IsRound**、**RadiusX**、および **RadiusY** の説明を参照してください。

## C1Report デザイナの新機能

- **リボン UI**

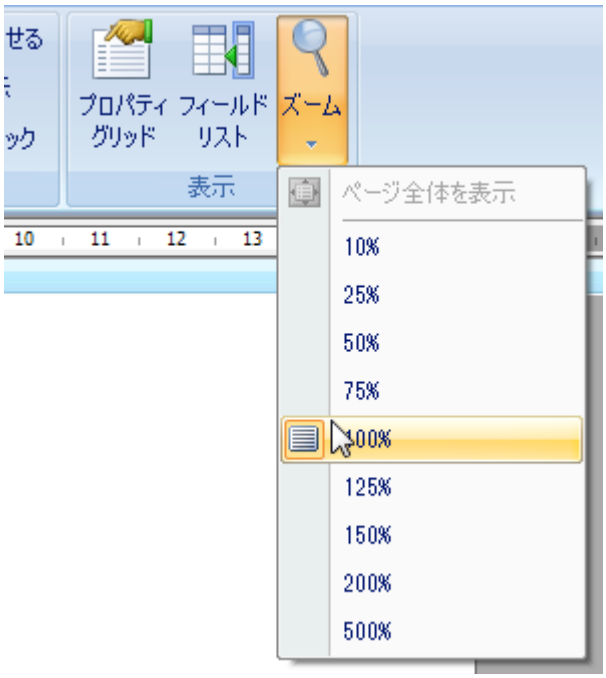
  C1Report デザイナの UI がリボン UI を使用して刷新されました。


Copyright © GrapeCity, Inc. All rights reserved.

- **デザイン画面の拡大／縮小表示**

  ステータスバーの右側にあるズームスライダを使用して、デザイン画面においてレポートを拡大／縮小して表示できるようになりました。

「ツール」グループのリボンにある「デザイン」タブの「ズーム」ドロップダウンメニューを選択して、指定した値で拡大／縮小して表示することもできます。




- **グリッド線とフィールドの境界線の色**

  デザイナオプションメニューを使用して、主なグリッドとフィールドおよび副次的なグリッドとフィールドの端の色を設定できるようになりました。

## 索引

アルファベット順の用語索引をドキュメントに挿入できるようになりました。詳細については、「用語索引」セクションと **RenderIndex**、**IndexEntry**、および **IndexEntryOccurrence** クラスの説明を参照してください。

## タグ入力フォーム

エンドユーザーが**C1PrintDocument**の Tags コレクションに含まれているタグのタグ値を入力するフォームを表示できるようになりました。詳細については、「実行時のタグ値の編集」を参照してください。

## 行間隔

**LineSpacing** プロパティで行間隔をパーセントによって取得または設定できるようになりました。デフォルト値は、標準行間隔に相当する 100 です。**RTF** が **True** の場合、このプロパティは無視されることに注意してください。新しい動作は、**LineSpacing** と完全な互換性があります。


Copyright © GrapeCity, Inc. All rights reserved.

## プレビュー画面のコンテキストメニュー

プレビュー画面でコンテキストメニューを使用できるようになりました。デフォルトの **ContextMenuStrip** が**C1PreviewPane** に追加されました。コンテキストメニューをカスタマイズするには、「コンテキストメニューからの項目の削除」と「コンテキストメニューの無効化」を参照してください。

## その他の機能

- Field.BarCode プロパティへの BarCode.UpcE 設定の追加（UPC-E エンコードはUPC-A の約半分のサイズです）。
- **印刷エラーの発生時のイベント**

  **C1PreviewPane**.**Error** と **C1PrintManager**.**Error** イベントでユーザーハンドラが定義されている場合、印刷時に外部例外が発生すると、その例外は捕捉され、代わりにイベントハンドラが呼び出されます。これは、印刷エラーが発生した際のプログラムクラッシュの回避に役立ちます。

- **新しいスクリプトタグの追加**

  新しいスクリプトタグ **HyperlinkPageNo** が追加されました。生成されるドキュメントでは、ハイパーリンクの対象が存在するページ番号に置き換えられます（オブジェクトは、これを動作させるためにハイパーリンクとする必要があります）。
  たとえば、次のコードを参照してください。

**Visual Basic コードの書き方**

Visual Basic

```
Dim rt as New RenderText()
rt.Text = "(C1LinkTargetDocumentLocation) Goto page: [HyperlinkPageNo]."
rt.Hyperlink = New C1Hyperlink(New C1LinkTargetDocumentLocation(rt2))
rt.Style.Borders.All = LineDef.[Default]
doc.Body.Children.Add(rt)
```

**C# コードの書き方**

C#

```
rt = new RenderText();
rt.Text = "(C1LinkTargetDocumentLocation) Goto page: [HyperlinkPageNo].";
rt.Hyperlink = new C1Hyperlink(new C1LinkTargetDocumentLocation(rt2));
rt.Style.Borders.All = LineDef.Default;
doc.Body.Children.Add(rt);
```

この結果、次のテキストが生成されます。
(C1LinkTargetDocumentLocation) Goto page: 12.
「12」は、ハイパーリンクの対象を含む実際のページ番号に置き換えられます。

# 機能の改善

Reports for WinForms 5.0J では、次の機能が改善されました。

- **エクスポートオプション**
  すべてのエクスポート形式に対してエクスポートオプションダイアログボックスを表示できるようになりました。**書式固有のオプション**をサポートしない書式では、デフォルトのダイアログボックスは[**プレビュー**]オプションを提供します。特に、すべての **C1Report** 固有のエクスポート形式で、デフォルトでオプションダイアログボックスが表示されます（この


Copyright © GrapeCity, Inc. All rights reserved.

ビルドの前は、デフォルトでオプションダイアログボックスは表示されませんでした)。また、プレビューの[**名前を付けて保存**]ダイアログボックスでは、従来別個の行として表示された複数の C1Report エクスポート形式がマージされると共に、エクスポートオプションダイアログボックスを介して利用可能な変形を選択できるようになりました。

- **エクスポート時のプレビュー**
  対応するファイルタイプを開くことができるプログラムが現在のシステムに登録されている場合、すべてのエクスポート形式で[プレビュー]オプションがサポートされるようになりました。さらに、C1D/C1DX 形式では、デフォルトのプログラムが見つからない場合、**C1PrintPreviewDialog** コントロールのロードおよび表示を介してプレビューを表示できるかどうか試行されます。C1D/C1DX 以外のすべてのエクスポート形式では、[**プレビュー**]オプションのデフォルト値は **True** です。
- **ドキュメント生成時刻**
  ドキュメントの生成時に、**RevisionTime** プロパティが **DateTime.Now** に設定されるようになりました。以前に設定されなかった場合、**CreationTime** プロパティも **DateTime.Now** に設定されます。
- **警告の最大数の制限**
  ドキュメントの生成時に **Warnings** コレクションに追加される警告が、最大 100 個までに制限されるようになります。警告が多過ぎるとドキュメントを生成する速度が大幅に低下する可能性があるため、警告は制限されています。
- **折り返しテキストにおける改行方法の改善**
  C1Report と C1PrintDocument の自動折り返しテキストで使用される改行アルゴリズムが、Unicode 標準の更新案に記載されたルールに準拠するように大幅に改善されました(http://unicode.org/reports/tr14/tr14-23.html を参照)。特に、日本語テキストが正しく折り返されるようになりました。
- **プリンタドライバの互換性**
  プリンタドライバに起因する特定の問題に対処するために新しい機能が追加されました。詳細については、「プリンタドライバの操作」を参照してください。
- **新しいローカライズ文字列**
  以下のエンドユーザーが表示できるローカライズ文字列を含む、新しい ToolTip がプレビューペインに追加されました。
  - **DoubleClickToHideMarginsHorz**:「ダブルクリックして、上/下余白とヘッダーを非表示にします。」
  - **DoubleClickToHideMarginsVert**:「ダブルクリックして、左/右余白を非表示にします。」
  - **DoubleClickToHideMarginsAll**:「ダブルクリックして、すべての余白を非表示にします。」
  - **DoubleClickToShowMarginsHorz**:「ダブルクリックして、上/下余白とヘッダーを表示します。」
  - **DoubleClickToShowMarginsVert**:「ダブルクリックして、左/右余白を表示します。」
  - **DoubleClickToShowMarginsAll**:「ダブルクリックして、すべての余白を表示します。」

  アプリケーションのローカライズについての詳細は、「ローカライズ」を参照してください。

- ハンドラが Error イベントに関連付けられている場合、IO 操作(ファイルを保存する、開く、印刷するなど)中にエラーが発生したときにエラーメッセージが表示されず、代わりにイベントが発生します。それ以外の場合、デフォルトのエラーメッセージが表示されます(これは、以前の動作でした)。
- プレビューで C1PrintDocument ページを表示するために、アンチエイリアススムージングモードが使用されるようになりました。これにより、ドキュメントの外観(丸みのある角の枠の使用など)が改善されます。必要な場合、**DefaultSmoothingMode** プロパティを設定して、スムージングモードを抑制できます。
- **C1PreviewPane** では、Microsoft Word 内のページ間の境界をダブルクリックして、ページ間のスペースを表示したり、非表示にしたりできるようになりました。
  従来、この操作はシングルクリックで実行されました。シングルクリックでスペースを表示したり、非表示にしたりする場合は、**HideMarginsOnSingleClick** プロパティを True に設定します。
- **C1PreviewPane** のページの枠に Microsoft Vista スタイルのドロップシャドウが追加されました。
  従来どおりの方法で影を表示する場合は、**LegacyPageBorder** プロパティを True に設定し、Vista スタイルのドロップシャドウを無効にします。
- **C1PreviewPane** ルーラーでのマウストラックの表示(**TrackMousePosition** が **True** の場合、表示されます)が改善され、Microsoft Vista でのちらつきが減少しました。
  従来どおりマウストラックを表示する場合は、**LegacyMouseTrack** プロパティを **True** に設定し、旧来の方法を使用してルーラー上にマウストラックを表示します。
- **フォントの埋め込み**
  フォントの埋め込みが、フォントの代替に対応するように大幅に変更されました。古い **AutoEmbedFonts** プロパティが廃止になり、新しいパブリックプロパティとサポートタイプのセットが C1PrintDocument に追加されました。詳細については、「埋め込みフォント」を参照してください。
- **RenderToc オブジェクト**


Copyright © GrapeCity, Inc. All rights reserved.

任意の表示オブジェクトを**RenderToc** オブジェクトに追加できるようになりました。たとえば、ページごとに反復されるヘッダーを含む **RenderToc** オブジェクトを作成できます。

**Visual Basic コードの書き方**

Visual Basic

```
Dim doc As New C1PrintDocument()
Dim toc As New RenderToc(doc)
toc.Style.Font = New Font("Times New Roman", 15)
doc.Body.Children.Add(toc)
Dim rt As New RenderText()
rt.Text = "Header of RendertToc"
rt.Style.Font = New Font("Verdana", 20)
rt.Style.TextAlignHorz = AlignHorzEnum.Center
rt.Style.BackColor = Color.LightBlue
toc.Children.Add(rt)
' rt をヘッダーとして設定します
toc.HeaderVert = rt
' サンプルコンテンツを追加します
For i As Integer = 0 To 29
    toc.AddItem("Level1", 2, 1)
    toc.AddItem("Level2", 3, 2)
Next
toc.BreakAfter = BreakEnum.Page
```

**C# コードの書き方**

C#

```
C1PrintDocument doc = new C1PrintDocument();
RenderToc toc = new RenderToc(doc);
toc.Style.Font = new Font("Times New Roman", 15);
doc.Body.Children.Add(toc);
RenderText rt = new RenderText();
rt.Text = "Header of RendertToc";
rt.Style.Font = new Font("Verdana", 20);
rt.Style.TextAlignHorz = AlignHorzEnum.Center;
rt.Style.BackColor = Color.LightBlue;
toc.Children.Add(rt);
// rt をヘッダーとして設定します。
toc.HeaderVert = rt;
// サンプルコンテンツを追加します。
for (int i = 0; i < 30; ++i)
{
    toc.AddItem("Level1", 2, 1);
    toc.AddItem("Level2", 3, 2);
}
toc.BreakAfter = BreakEnum.Page;
```

- レイアウトを初期化する際のパフォーマンスの向上。古いコードの **PageSettings.Bounds** が、かなりパフォーマンスを向上できる同等のコードに置き換えられました。
- 数値の右揃えの改善。丸括弧を使用して負の数値を表す場合、使用する形式は「#,##0 ;(#,##0);Zero」などの多数の部分に分割された文字列とする必要があります。
  正とゼロの値を書式設定するために使用する部分は、スペースで終了することに注意してください。この結果、正と負


Copyright © GrapeCity, Inc. All rights reserved.

の数値が右側に正しく揃えられます。
- 長いレポートを生成するマルチスレッドアプリケーションに影響を与えるスレッドセーフティの改善。
- 以前のビルドでは、**RenderInputComboBox** の幅の自動調整は **Text** プロパティの文字列の幅として計算されました。今回のビルドから、可能な値のリスト内のすべての項目の幅も使用されるようになり、幅の自動調整がこれらの値のどのような値にも対応できるようなりました。
- RenderRectangle、RenderLine のデフォルトサイズが変更されました。以前のバージョンでは、表示オブジェクト自体のサイズが空でない場合でも、デフォルトで、RenderShape オブジェクトが目に見える形状を生成しないように、形状の初期サイズはゼロとなっていました。今回のバージョンから、デフォルトで、形状が表示オブジェクトの領域を満たすようになりました（表示オブジェクトの幅と高さを持っている場合など）。RenderLine オブジェクトは、(0,0) から (self.width,self.height) まで、対角線上に線を表示します。
- **C1PrintDocument** .Pages コレクションからページが削除された場合、残りのページの X および Y プロパティはドキュメント内の実際の位置を反映するように更新されます。

# 仕様変更

Reports for WinForms 5.0J では、次の仕様変更が行われました。

- 以下のエクスポートプロバイダクラスについて、"obsolete" 属性が設定され使用を推奨されなくなりました。
  - **ReportHTMLDrillDownExportProvider**（代わりに **ReportHTMLExportProvider** を使用してください）。
  - **ReportHTMLPagedExportProvider**（代わりに **ReportHTMLExportProvider** を使用してください）。
  - **ReportRTFLabelsExportProvider**（代わりに **ReportRtfExportProvider** を使用してください）。
  - **ReportTextSinglePageExportProvider**（代わりに **ReportTextExportProvider** を使用してください）。
- 表示オブジェクトの位置を指定する場合、**x** と **y** が **left** と **top** として解釈されるようになりました。たとえば、以下のコードは、次のコードと等価になります。

使用前のコード

```
RenderText rtxt = new RenderText("My text.");
rtxt.Y = "prev.y";
```

使用後のコード

```
RenderText rtxt = new RenderText("My text.");
rtxt.Y = "prev.top";
```

この場合、旧バージョンでは、x と y は位置とサイズの式では特別な意味はありません（これは、微妙なエラーを引き起こす可能性があります）。

- 印刷、エクスポート、およびページのハイパーリンク（**C1LinkTargetPage**）のすべてが物理ページ番号（基本的には、1ページ単位のインデックス）を使用するようになりました。
  以前のビルドでは、数字によるページの印刷およびエクスポートなどの操作は、論理ページ番号（ページの番号付けの変更がドキュメントのオブジェクトに適用される場合、物理ページ番号と異なる可能性がある）で動作しました。この操作は、特定のページを印刷できないなどの問題を引き起こす可能性がありました（たとえば、論理ページ番号が 1、1、1 のドキュメントを印刷すると仮定した場合、最後のページは印刷できませんでした）。この変更は、**PageNumberingChange** を使用しないドキュメントには影響を与えません。
- **RenderObject.RepeatBordersVert** と **RenderObject.RepeatBordersHorz** プロパティは、廃止として指定されました。これらのプロパティは、**BordersSplitVertMode** と **BordersSplitHorzMode** プロパティに置き換えられました。
- 以下の2つのパブリッククラスの名前が変更されました。
  - **C1.C1Preview.Page** が **C1Page** に変更されました。
  - **C1.C1Preview.PageCollection** が**C1PageCollection** に変更されました。

これは、Reports for WPF での潜在的な名前の競合を回避するために行われました。

# クラスメンバ

| クラス | 説明 |
|---|---|


Copyright © GrapeCity, Inc. All rights reserved.

| | |
|---|---|
| **C1ReportHtmlExporter** | **C1Report**ドキュメントのエクスポートを HTML に指定します。 |
| **C1ReportRtfExporter** | **C1Report** ドキュメントのエクスポートを RTF に指定します。 |
| **C1ReportTextExporter** | **C1Report** ドキュメントのエクスポートをプレーンテキストに指定します。 |
| **AdjustPrintPageEventArgs クラス** | **AdjustPrintPage** イベントの引数を記述します。 |
| **AdjustPrintPageEventHandler デリゲート** | **AdjustPrintPage** イベントを処理できるメソッドを表します。 |
| **BordersSplitMode 列挙体** | オブジェクトがページ間で分割されたときに枠を表示する方法を指定します。<br>以下の設定があります。<br>• **なし**<br>オブジェクトが分割された場所に枠は表示されません。<br>• **正方形**<br>オブジェクトが分割された場所に正方形の枠が表示されます。<br>• **角丸**<br>オブジェクトが分割された場所に枠が表示されます。<br>枠が角丸の場合、オブジェクトが分割された場所に角丸の枠が表示されます。 |
| **CompatibilityOptions クラス** | 前のバージョンの **C1PreviewPane** との互換性を改善するオプションを表します。 |
| **ContextMenuTags クラス** | 組み込みコンテキストメニュー項目の Tag プロパティに割り当てられ、これらの項目を識別するために、ユーザーコードで使用できる静的な読み取り専用オブジェクトを定義します。 |
| **DocumentFont** | **C1PrintDocument** で使用されるフォントを記述します。 |
| **DocumentFontCollection** | **C1PrintDocument** で使用するフォントを記述する **DocumentFont** オブジェクトのコレクションを表します。 |
| **FontHandling** | **C1PrintDocument** によりフォントの埋め込みと代替を処理する方法を指定します。 |
| **PageResolvedInfo** | 実際の絶対ページサイズ、絶対余白値など、解決された **C1Page** に関する情報を取得できるプロパティを提供します。このクラスによって提供されるすべてのサイズと長さの値は、ResolvedUnit で表される絶対値です。 |
| **TagInputParams** | 個別タグの入力をカスタマイズするタグ入力ダイアログボックスによって使用される専用クラスの抽象ベースクラス。 |
| **TagStringInputParams** | 文字列タグ値の入力に対するカスタマイズパラメータを表します。 |
| **TagDateTimeInputParams** | DateTime タグ値の入力に対するカスタマイズパラメータを表します。 |
| **TagListInputParams** | リストから選択できるタグ値の入力に対するカスタマイズパラメータを表します。 |
| **TagBoolInputParams** | ブール型タグ値の入力に対するカスタマイズパラメータを表します。 |
| **TagNumericInputParams** | 数値型タグ値の入力に対するカスタマイズパラメータを表します。 |

## 新しい列挙体

| 列挙体 | 説明 |
|---|---|
| **PageLayoutEnum 列挙体** | PDF ドキュメントを開くときに使用するページレイアウトを指定します。 |
| **PageModeEnum 列挙体** | PDF ドキュメントを開くときに、それを表示する方法を指定します。 |


Copyright © GrapeCity, Inc. All rights reserved.

## 新しいメンバ

| クラス | メンバ | 説明 |
|---|---|---|
| **C1Page** | **AsMetafile** メソッド | C1Page の **AsMetafile**メソッド上の新しいオーバーロード： AsMetafile(Graphics referenceGraphics, EmfType emfType, bool whiten, bool renderInputControls, System.Drawing.Drawing2D.SmoothingMode smoothingMode) 新しい **smoothingMode** 引数により、デフォルトのスムージングモードをアンチエイリアスモードにオーバーライドできます（プレビューで使用）。 |
| **C1PreviewPane** | **AdjustPrintPage** イベント | ドキュメントの印刷に使用する **C1PrintManager** の PrintPage イベントハンドラから発生します。 |
| | **CompatibilityOptions** プロパティ | 前のバージョンの **C1PreviewPane** との互換性を改善するオプションを表します。 |
| **C1PrintManager** | **AdjustPrintPage** イベント | 実際にページを印刷する前に、現在のプリントマネージャの **PrintDocument.PrintPage** イベントハンドラから発生します。 |
| **C1PrintOptions** | **DrawPrintableAreaBounds** プロパティ | ページの印刷可能領域の周囲に線を表示するかどうかを示す値を取得または設定します（プリンタの問題のデバッグに便利です）。 |
| | **PrintableAreaBoundsPen** プロパティ | **DrawPrintableAreaBounds** が True の場合、印刷可能領域の枠線を表示するために使用するペンを取得または設定します。 |
| | **PrintAsBitmap** プロパティ | 印刷前に、ページのメタファイルをビットマップに変換し、プリンタのハードマージンにクリップするかどうかを示す値を取得または設定します。 |
| **Borders** | **RadiusX** プロパティ | 枠の丸みのある角を表示するために使用する楕円の X 軸方向の半径を取得または設定します。 |
| | **RadiusY** プロパティ | 枠の丸みのある角を表示するために使用する楕円の Y 軸方向の半径を取得または設定します。 |
| | **IsRound** プロパティ | 現在の **Borders** 構造が丸みのある角を定義するかどうか（つまり、**RadiusX** と **RadiusY** の両方にゼロ以外の値を設定するかどうか）を示す値を取得します。 |
| **RenderObject** | **BordersSplitVertMode** プロパティ | オブジェクトが高過ぎるためページ間で分割される場合、枠を表示する方法を示す値を取得または設定します。 |
| | **BordersSplitHorzMode** プロパティ | オブジェクトの幅が広過ぎるため水平方向のページ間で分割される場合、枠を表示する方法を示す値を取得または設定します。 |
| **C1Page** | **ResolvedInfo** プロパティ | 現在のページに対応する **PageResolvedInfo** オブジェクトを取得します。現在のページが解決された後で、このプイロパティは、ページ、余白、および列の配置を記述する絶対値を取得できます。現在のページ |


Copyright © GrapeCity, Inc. All rights reserved.

| | | |
|---|---|---|
| | | が解決されていない場合、このプロパティの値は Null になるため、ネストされたプロパティにアクセスする前に、この値を必ずテストする必要があることに注意してください。 |
| **C1PageSettings** | **PrinterResolutionKind** プロパティ | 希望するプリンタ解像度の種類を取得または設定します。 |
| | **PaperSourceKind** プロパティ | 希望する給紙方法の種類を取得または設定します。 |
| **C1PrintDocument** | **DocumentFonts** プロパティ | 現在のドキュメントで使用するフォントを記述する **DocumentFont** オブジェクトのコレクションを取得します。 |
| | **EditTags** メソッド | タグ入力ダイアログボックスを表示し、ユーザーが Tags コレクションに含まれるタグの値を入力／更新できるようにします。ユーザーがダイアログを受け入れた場合 True、そうでない場合 False を返します。 |
| | **FontHandling** プロパティ | 現在の **C1PrintDocument** によりフォントの埋め込みと代替を処理する方法を示す値を取得または設定します。 |
| | **ShowTagsInputDialog** プロパティ | ドキュメントを生成する前に、一部またはすべてのユーザー定義タグ（Tags コレクションの要素）の値を入力できるダイアログをユーザーに表示するかどうかを示す値を取得または設定します。特定のタグをダイアログに含めるか、またはダイアログから除外するには、Tag.Flags を使用します。このプロパティは、デフォルトで False です。 |
| | **TagsInputDialogClass** プロパティ | タグ値を入力するために使用されるフォームのタイプを取得または設定します（**ShowTagsInputDialog** が True の場合、フォームが表示されます）。フォームタイプは **TagsInputFormBase** から派生する必要があります。 |
| | **TagsInputDialogClassName** プロパティ | タグ値を入力するために使用されるフォームのタイプ名を取得または設定します（**ShowTagsInputDialog** が True の場合、フォームが表示されます）。フォームタイプは **TagsInputFormBase** から派生する必要があります。 |
| | **UsageInfo** プロパティ | 生成した日付と時刻、直前にドキュメントを保存またはロードするために使用されたファイル名など、現在のオブジェクトの使用方法に関連する情報を含む **UsageInfo** オブジェクトを取得します。 |
| **RenderFragment** | **HasClippedChildren** メソッド | 現在のフラグメントの任意の子フラグメントがその境界によってクリップされているかどうかを確認します。再帰パラメータによっては、すべての子フラグメントが再帰的にテストされる場合もあります。 |
| | **TableColOnPage** プロパティ | 現在の表示オブジェクト（現在のページの列インデックスなど）を含む、最もネストされた **RenderTableFragment** 内の列のゼロベースのインデックスを取得するか、現在のオブジェクトがテーブルに含まれていない場合 -1 を取得します。 |


Copyright © GrapeCity, Inc. All rights reserved.

| | | |
|---|---|---|
| | **TableFragment** プロパティ | 現在のフラグメントを含む、最もネストされた **RenderTableFragment** を取得するか、現在のオブジェクトがテーブルに含まれていない場合 Null を取得します。 |
| | **TableRowOnPage** プロパティ | 現在の表示オブジェクト（現在のページの行インデックスなど）を含む、最もネストされた **RenderTableFragment** 内の行のゼロベースのインデックスを取得するか、現在のオブジェクトがテーブルに含まれていない場合 -1 を取得します。 |
| **RenderObject** | **Table** プロパティ | 現在の表示オブジェクトを含む、最もネストされた RenderTable を取得するか、現在のオブジェクトがテーブルに含まれていない場合 Null を取得します。 |
| | **TableCol** プロパティ | 現在の表示オブジェクトを含む、最もネストされた RenderTable 内の列のゼロベースのインデックスを取得するか、現在のオブジェクトがテーブルに含まれていない場合 -1 を取得します。 |
| | **TableRow** プロパティ | 現在の表示オブジェクトを含む、最もネストされた **RenderTable** 内の行のゼロベースのインデックスを取得するか、現在のオブジェクトがテーブルに含まれていない場合 -1 を取得します。 |
| **RenderTableFragment** | **BottomFooterRowCount** プロパティ | 現在のフラグメント（現在のページまたはページ列など）の下部にあるテーブルフッター内の行数を取得します。 |
| | **ColCount** プロパティ | 現在のテーブルフラグメント（現在のページまたはページ列など）内の列数を取得します。 |
| | **LeftHeaderColCount** プロパティ | 現在のフラグメント（現在のページまたはページ列など）の左側にある垂直方向のテーブルヘッダー内の列数を取得します。 |
| | **RightFooterColCount** プロパティ | 現在のフラグメント（現在のページまたはページ列など）の右側にある垂直方向のテーブルフッター内の列数を取得します。 |
| | **RowCount** プロパティ | 現在のテーブルフラグメント（現在のページまたはページ列など）内の行数を取得します。 |
| | **TopHeaderRowCount** プロパティ | 現在のフラグメント（現在のページまたはページ列など）の上部にあるテーブルヘッダー内の行数を取得します。 |
| **RenderToc** | **LinksOnPageNumbersOnly** プロパティ | 表示される TOC 内のページ番号がクリック可能なハイパーリンクだけを表すかどうかを示す値を取得または設定します（そうでない場合、エントリのキャプションとページ番号間の領域全体がクリック可能になります）。デフォルト値は **False** です。 |
| **Tag** | **Description** プロパティ | タグの説明を取得または設定します。空でない場合、タグ入力ダイアログのラベルとして使用されます。（説明が空の場合、名前は入力ダイアログボックス内に表示されます）。 |
| | **InputParams** プロパティ | 現在のタグのエディタをカスタマイズするために、タグ |


Copyright © GrapeCity, Inc. All rights reserved.

| | | |
|---|---|---|
| | | 入力ダイアログによって使用される **TagInputParams** オブジェクトを取得または設定します。デフォルトで、このオプションは **Null** です。 |
| | **SerializeValue** プロパティ | 現在のタグの値をシリアル化するかどうかを示す値を取得または設定します。デフォルトは **True** インチです。 |
| | **ShowInDialog** プロパティ | 現在のタグを表示し、その値をタグ入力ダイアログで編集可能にするかどうかを示す値を取得または設定します。**True**（デフォルト） |
| **C1DataBinding** | **OutlineText** プロパティ | アウトラインキャプションの生成に使用する式を定義する Expression オブジェクトを取得または設定します。 |
| | **ParentOutline** プロパティ | 親アウトラインノードを取得または設定します。 |
| **C1PreviewPane** | **Error** イベント | 外部例外がスローされたとき（印刷中など）に発生します。 |
| **C1PrintDocument** | **TagEscapeString** プロパティ | **TagOpenParen** と **TagCloseParen** をエスケープするために使用できる文字列を取得または設定します。デフォルトは、「\」です。 |
| **C1PrintManager** | **Error** イベント | 外部例外がスローされたとき（印刷中など）に発生します。 |
| **HtmlTableFilter** | **CellPadding** プロパティ | セルとそのコンテンツ間の端部間のピクセル数を取得または設定します。このプロパティは、出力 HTML テーブルに適用され、フィールドが透明な場合にセクションの BackColor を受け入れます。 |
| | **CellSpacing** プロパティ | 出力テーブル内の隣接するセル間のピクセル数を取得または設定します。このプロパティは、出力 HTML テーブルに適用され、フィールドが透明な場合にセクションの BackColor を受け入れます。 |
| **RenderObject** | **Clip** プロパティ | オブジェクトをクリップするかどうかを示す値を取得または設定します。形状（**RenderShapeBase** から派生する）以外のすべてのオブジェクトに対してデフォルトは **True** です。 |
| | **FormatDataBindingInstanceScript**プロパティ | 表示オブジェクトの新しいインスタンスがデータバインドの解決によって作成されるたびに実行されるスクリプトを取得または設定します。詳細については、ComponentOne HelpCentral から入手可能な **AlternatingBackground** サンプルも参照してください。 |

# 4.0J C1Report

このセクションでは、True WinReports for .NET 4.0J（C1Report）で追加された新しい機能について説明します。

# 新機能

True WinReports for .NET 4.0J（C1Report）では、次の新しい機能が追加されました。


Copyright © GrapeCity, Inc. All rights reserved.

## 固定行数のレイアウト

1 ページに出力する行数を指定できます。指定の行数を超えた場合は改ページし、指定の行数に満たない場合は空白行として出力します。その結果、各ページのデータ数が異なっていても、すべてのページを同じレイアウトで出力することができます。

## グラフフィールド

グラフ種別が6種類追加されました。また、グラフの3D表示、グラフの色を一括して指定するテーマ色、売上予測で役立つ近似曲線グラフにも対応しました。

## 拡張バーコードフィールド

バーの細い幅と太い幅の設定や、バー幅のドット補正、バーコード画像の解像度指定に対応しました。バーの幅はミリメートル単位だけでなくピクセル単位でも指定できます。

## PDF エクスポート

- PDF エクスポートで権限の許可／禁止やユーザーパスワードの設定に対応しました。
- PDF エクスポートでビューア表示方法の設定に対応しました。
- サブレポートの PDF アウトラインのレンダリング。
- 大きな **True** Color 画像の .pdf エクスポートの強化（新しいフィルタは、これまでより多くの画像を高速に圧縮します）。この違いは、大きな .jpg 画像で特に顕著です。

## Excel エクスポート

- Open XML Excel と EMF のエクスポートに対応しました。
  OpenXml ファイルは、標準 Zip 形式を使用して圧縮されるため、通常の Excel ファイル（XLS）よりコンパクトになります。また、実際のコンテンツは XML で保存されるため、プログラムによる操作が簡単です。
  OpenXml ファイルを保存するには、**FileFormatEnum** を使用します。たとえば、次のように記述します。

  **Visual Basic コードの書き方**

  Visual Basic
  ```
  C1.Win.C1Report c1r = C1.Win.C1Report.C1Report;
  XlsFilter filter = new XlsFilter(fileName);
  filter.OpenXml = true;
  c1r.RenderToFilter(filter);
  ```

- Excel エクスポートで画像の出力に対応しました。画像を含む Excel 出力の強化
  このバージョンから、Excel 出力にはデフォルトで画像が含まれます。この機能を無効にするには、レポートをレンダリングする前に、フィルタの **Images** プロパティを **False** に設定します。たとえば、次のように記述します。

  **Visual Basic コードの書き方**

  Visual Basic
  ```
  Dim filter As XlsFilter = New XlsFilter(outputFileName)
  filter.Images = False
  c1r.RenderToFilter(filter)
  ```

  **C# コードの書き方**


Copyright © GrapeCity, Inc. All rights reserved.

C#

```
XlsFilter filter = new XlsFilter(outputFileName);
filter.Images = false;
c1r.RenderToFilter(filter);
```

- Excel エクスポートの強化。複数のフィールドの並びを1つのシート行に保持します。
- XLS ストリームへのレポートのレンダリング
  レポートを XLS ストリームにレンダリングできます [**RenderToStream** メソッド(Stream, FileFormatEnum.Excel)]。次のコードは、この新しい機能を使用して、一時ファイルを使用せずに Excel ファイルを電子メールの添付ファイルとして送信します。

**Visual Basic コードの書き方**

Visual Basic

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    ' 電子メールメッセージを作成します
    Dim msg As MailMessage = New MailMessage("from@from.com", "to@to.com",
"Catalog Spreadsheet.", "Here's the spreadsheet I promised you.")

    ' スプレッドシートで添付ファイルを作成します
    Dim stream As MemoryStream = New MemoryStream
    C1Report1.RenderToStream(stream, FileFormatEnum.Excel)
    stream.Position = 0
    Dim att As Attachment = New Attachment(stream, "Catalog.xls",
"application/x-excel")
    msg.Attachments.Add(att)
    ' 別の添付ファイルを作成します。そうしないと機能しません
    stream = New MemoryStream(Encoding.ASCII.GetBytes("This is some plain
text"))
    msg.Attachments.Add(New Attachment(stream, "plain.txt"))
    ' メッセージを送信します
    Dim client As SmtpClient = New SmtpClient("smtp.mymail.com")
    client.Credentials = System.Net.CredentialCache.DefaultNetworkCredentials
    client.Send(msg)
End Sub
```

**C# コードの書き方**

C#

```
private void button1_Click(object sender, EventArgs e)
{
    // 電子メールメッセージを作成します
    MailMessage msg = new MailMessage(
        "from@from.com",
        "to@to.com",
        "Catalog Spreadsheet.",
        "Here's the spreadsheet I promised you.");
    // スプレッドシートで添付ファイルを作成します
    MemoryStream stream = new MemoryStream();
    c1Report1.RenderToStream(stream, FileFormatEnum.Excel);
    stream.Position = 0;
```


Copyright © GrapeCity, Inc. All rights reserved.

```
    Attachment att = new Attachment(stream, "Catalog.xls", "application/x-
excel");
    msg.Attachments.Add(att);
   // 別の添付ファイルを作成します。そうしないと機能しません
    stream = new MemoryStream(Encoding.ASCII.GetBytes("This is some plain
text"));
    msg.Attachments.Add(new Attachment(stream, "plain.txt"));
    // メッセージを送信します
    SmtpClient client = new SmtpClient("smtp.mymail.com");
    client.Credentials = System.Net.CredentialCache.DefaultNetworkCredentials;
    client.Send(msg);
}
```

## その他のエクスポート

- EMF のエクスポートに対応しました。
- TIFF エクスポートで画像の解像度設定に対応しました。
- TIFF画像エクスポートがモノクロ画像出力に対応。
- 新しい HTML エクスポートフィルタ：**HtmlTableFilter**
  **HtmlTableFilter** クラスは、従来の HtmlFilter クラスによって生成された DIV ベースの出力ではなく、TABLE タグに基づいて HTML 出力を生成します。**HtmlTableFilter** は、絶対位置を使用しないため、出力は、**HtmlFilter** クラスによって生成された DIV ベースの出力より正確さに欠けます。ただし、このクラスには次の3つの重要な利点があります。
  - テーブルベースのレポートは、標準の HTML エディタで容易に編集できます。たとえば、行を簡単に挿入することができます。これは、絶対位置で配置されたフィールドでは不可能です。
  - テーブルベースのレポートでは、クリップボードを効率よく操作できます。たとえば、ブラウザでレポートセクションをコピーし、Excel に貼り付けることができます。
  - テーブルベースのレポートは、フィールド内の垂直方向の配置（上、中央、下）をサポートします。

  HTML 出力の印刷または表示だけでなく、HTML 出力を操作（編集）する場合は、新しい **HtmlTableFilter** を使用することをお勧めします。

- **MetafileFilter** クラスおよび**FileFormatEnum** .Metafile の追加。

## レポートデザイナ

- レポートデザイナでグリッドサイズを変更可能になりました。
- レポートデザイナで Access 2007 レポートのインポートに対応しました。

## その他

- ReportParameter.Type プロパティの追加
- **ReportParameter** クラスに **Type** プロパティが追加されました。これがパラメータ名、値、および型を公開します。次のコードは、**Type** プロパティを使用して、ダイアログが表示されたときに、すべての DateTime パラメータのデフォルト値を今日の日付に変更します。

  **Visual Basic コードの書き方**

| Visual Basic |
| --- |
| `Private Sub C1Report1_InitializeParametersDialog(ByVal sender As System.Object, ByVal e As C1.Win.C1Report.DialogEventArgs) Handles C1Report1.InitializeParametersDialog` |


Copyright © GrapeCity, Inc. All rights reserved.

```
    ' すべての DateTime パラメータのデフォルト値を変更します
    For Each p As ReportParameter In e.Parameters
     If p.Type = GetType(DateTime) Then
       p.Value = DateTime.Now
     End If
    Next
End Sub
```

**C# コードの書き方**

C#

```
private void c1Report1_InitializeParametersDialog(object sender, DialogEventArgs
e)
{
    // すべての DateTime パラメータのデフォルト値を変更します
    foreach (ReportParameter p in e.Parameters)
    {
        if (p.Type == typeof(DateTime))
            p.Value = DateTime.Now;
    }
}
```

- **AddScriptObject メソッドの強化**

  **AddScriptObject メソッド**のオーバーロードが追加され、スクリプトオブジェクトをすべてのサブレポートにも追加できます。

- RTF（リッチテキスト形式）のサポートの強化。インポートされた RTF で表形式および両端揃えがサポートされます。
- 部分的に信頼できる状況で参照グラフィックオブジェクトを破棄する際のチェックの追加。
- AddScriptObject メソッドの強化
- 新しいプロパティ：IgnoreScriptErrors
- 「VarType」VBScript 機能の追加。連結に + が使用されている場合のキャストの強化。
- **BarCode** プロパティの新しい設定（**BarCodeEnum**.UpcA）の追加。UPC-A は、11 桁の商品コードの一般的なエンコードです。
- **新しいドキュメント保存形式**
  新しいドキュメント保存形式 C1DX が追加されました。この形式は、Microsoft Open Packaging Conventions に準拠し、構造的に Office 2008 で使用される Open XML Format に似ています。C1DX ファイルは、ドキュメントの別の部分を表す要素である zip アーカイブです。**Reports for WinForms** プレビューコントロールでファイルを開くことで、C1DX ファイルを開けるようになったことに注意してください。
  **C1DocumentFormatEnum** パラメータを受け入れないすべてのドキュメントの **Load** および **Save** メソッドは、形式がファイル名拡張子によって指定できないかぎり、古い C1D 形式を使用することに注意してください。

## クラスメンバ

True WinReports for .NET 4.0J（C1Report）で追加された新しいメンバを下記に示します。

- **RecordsPerPage** プロパティ
- OpenXml メンバ（**FileFormatEnum 列挙体**）
- **PdfSecurityOptions** クラス
- **PdfSecurityOptions** プロパティ
- **PdfViewerOptions** クラス
- **PdfViewerOptions** プロパティ
- **OpenXml** プロパティ
- **Images** プロパティ


Copyright © GrapeCity, Inc. All rights reserved.

- **Resolution** プロパティ
- UpcE メンバ（**BarCodeEnum 列挙体**）
- **MetafileType** プロパティ
- **Monochrome** プロパティ
- **Type** プロパティ
- **IgnoreScriptErrors** プロパティ
- **HtmlTableFilter** クラス

# 4.0J（C1PrintDocument）

このセクションでは、Preview for .NET 4.0J（C1PrintDocument）で追加された新しい機能について説明します。

# 新機能

Preview for .NET 4.0J（C1PrintDocument）では、以下の機能が強化されました。

- PrintDocument のプレビュー印刷に対応しました。
- データバインド機能が追加されました。
- 均等割り付け、文字間隔、文字幅の書式設定に対応しました。
- XML MS Word 形式（docx）へのエクスポートのサポート。
- Microsoft Excel Open XML 形式へのエクスポートのサポート。
- C1Report 定義のインポートおよび生成の追加。
- PDF エクスポートでビューア表示方法の設定に対応しました。
- 画像エクスポートで画像解像度の設定に対応しました。
- 9種類のバーコード出力に対応しました。
- テーブルの行の高さ／列の幅を自動的に調節します。
- テーブルセルの途中で改ページされたときに、分割された部分に境界線を表示します。
- 入力フォームの表示スタイル設定が追加されました。
- **CanSplit** プロパティのデフォルト値の変更。
- **PageLayout** オブジェクトへのデータ連結子オブジェクトの追加。
- レンダリングオブジェクトへの Max および Min 関数の追加。
- **FormsStyleEnum** 列挙体への新しい要素の追加。
- C1PrintDocument への新しい複合プロパティ CompatabilityOptions の追加。
- 新しいデフォルトページ設定プロパティの追加。
- **Style** クラスへのアンビエントフォント関連プロパティの追加。
- テーブルおよびセルでグリッド線を繰り返すためのプロパティの追加。
- **RenderC1Printable** クラスへの新しいメソッドの追加。
- レンダリングオブジェクトの分割動作の強化。
- テーブル行の引き伸ばしの強化。
- 開発者によるローカライズが可能な C1.C1Preview および C1.Win.C1Preview。
- 新しい集計関数（Min、Max、CountDistinct、First、Last、Var、VarP、StDev、StDevP）の追加。
- スクリプトオプションのサポート。
- 回転されたテキストのレンダリングの強化。
- ドキュメント生成時のエラー処理のサポート。
- スクリプトのサポート。
- データ連結のサポート。
- ローカライズのサポート。
- **GDI+ テキストの表示**
  UseGdiPlusTextRendering プロパティが追加されました。このプロパティのデフォルト値は、以前の Preview for .NET 製品のバージョンと互換性のある False です。
  GDI+ テキスト表示を使用する利点は、テキストの見栄えがほぼ間違いなく改善され、MS Office 2008 および他の新製品内のテキストに類似して表示されることです。たとえば、GDI+ テキストレイアウトは XPS のデフォルトのテキストレイアウトに一致します。欠点は、GDI+ テキスト表示が低速であり、今回のバージョンでは、これらのテキストでは新しいテキスト関連機能（均等間隔、文字の表示間隔および幅）がサポートされないことです。


Copyright © GrapeCity, Inc. All rights reserved.

- **新しいテキストの両端揃えモードの追加**
  文字の両端揃えモードが追加されました。この水平方向のテキストの両端揃えのモードを指定した場合、テキストは単語間のスペースだけではなく、単語内の個々の文字間のスペースも揃えられます。これをサポートするために、JustifyChars 要素が AlignHorzEnum 列挙体に追加され、テキストが水平方向に両端揃えされ、すべての文字間にスペースが追加されました。ドキュメントの UseGdiPlusTextRendering が False に設定されている場合、現在このプロパティは無視される（AlignHorzEnum.Justify として扱われる）ことに注意してください（この制限は今後のバージョンで削除されます）。
- **PDF ビューア環境設定の指定**
  エクスポートした PDF ドキュメントを Adobe Acrobat Reader で開いた場合のビューア環境設定を指定する機能が追加されました。

## TableVectorGroup.CanSplit プロパティのデフォルト値の変更

CanSplit プロパティのデフォルト値が false から true に変更されました。したがって、デフォルトでは、テーブル行または列のグループがページ間で分割されます。この変更の理由は、すべてがデフォルト設定で1つの行グループがデータ連結されている（データ連結テーブルを作成する適切な方法）場合、生成されるテーブルが現在のページに収まらないと、切り捨てられてしまったためです。現在は、期待されるとおりに動作するようになりました。

## C1Report 定義のインポートおよび生成の追加

C1Preview は、C1Report 定義のインポートおよび生成をサポートするようになりました。この機能は C1Report アセンブリに依存しません。

C1Report をC1PrintDocumentにインポートするには、C1PrintDocument に追加された新しい ImportC1Report メソッドを使用します。たとえば、次のコードを使用して、C1Report に付属する ReportBrowser サンプルの「Alphabetical List of Products」（製品のアルファベット順リスト）をインポートしてプレビューできます。

**Visual Basic コードの書き方**

Visual Basic

```
C1PrintDocument doc = new C1PrintDocument();
doc.ImportC1Report("NWind.xml", "Alphabetical List of Products");
C1PrintPreviewDialog pdlg = new C1PrintPreviewDialog();
pdlg.Document = doc;
pdlg.ShowDialog();
```

C1Preview には、C1Report 定義のインポートをサポートするための新しいクラスが追加されました。RenderField、RenderSection、および RenderReport クラスです。これらのクラスは、C1Report のインポートをサポートするために追加されましたが、必要に応じて「通常の」レンダリングオブジェクトとして使用できます。

C1Report 定義の詳細については、「C1Report 定義」を参照してください。

## PageLayout オブジェクトへのデータ連結子オブジェクトの追加

PageLayout オブジェクトの PageHeader、PageFooter、WaterMark、および Overlay オブジェクトは、データ連結子オブジェクトを保持することができます。ただし、これらのすべてのオブジェクトのデータソースがC1PrintDocument内に含まれている場合に限ります。つまり、データソースが DataSchema の DataSet である必要があります。そうでない場合は、ドキュメントをシリアライズ解除する際に、データソースを手作業で再作成する必要があります。

## レンダリングオブジェクトへの Max および Min 関数の追加

レンダリングオブジェクトのサイズを指定する際に、Max および Min 関数を使用できます。たとえば、次のコードは、テキストオブジェクトの幅が少なくとも 5cm になります。


Copyright © GrapeCity, Inc. All rights reserved.

**Visual Basic コードの書き方**

```
Visual Basic
Dim rt As New RenderText("Hello, World!")
rt.Style.BackColor = Color.Green
rt.Width = "Max(auto, 5cm)"
```

**C# コードの書き方**

```
C#
RenderText rt = new RenderText("Hello, World!");
rt.Style.BackColor = Color.Green;
rt.Width = "Max(auto, 5cm)";
```

次のコードは、「rt2」または「rt3」の下端のいずれか低い方に「rt」の上端を揃えます。

**Visual Basic コードの書き方**

```
Visual Basic
rt.Top = "Max(rt2.Bottom, rt3.Bottom)"
```

**C# コードの書き方**

```
C#
rt.Top = "Max(rt2.Bottom, rt3.Bottom)";
```

**メモ**:「auto」キーワードは、オブジェクトのサイズを指定する式でのみ使用できます。.

つまり、次の式は有効ですが

**Visual Basic コードの書き方**

```
Visual Basic
rt.Width = "auto + 10mm"
```

**C# コードの書き方**

```
C#
rt.Width = "auto + 10mm";
```

（オブジェクトの幅を自動の幅より 10mm 大きくします）、次の式は無効です。

**Visual Basic コードの書き方**

```
Visual Basic
rt.X = "auto + 10mm"
```

**C# コードの書き方**

```
C#
rt.X = "auto + 10mm";
```


Copyright © GrapeCity, Inc. All rights reserved.

## FormsStyleEnum 列挙体への新しい要素の追加

**FileFormatEnum 列挙体**には、次の2つの新しい要素が追加されました。

- WinXP: Windows XP（テーマ）コントロールスタイル。
- WinClassic: Windows クラシックコントロールスタイル。

また、InfoPath スタイルはすべてのコントロールでサポートされるようになりました。以前は、RenderInputText クラスでのみサポートされていました。

## C1PrintDocument への新しい複合プロパティ CompatabilityOptions の追加

C1PrintDocument に、新しい複合プロパティ CompatibilityOptions が追加されました。

- **CompatibilityOptions** プロパティ
  C1PrintDocument の以前のバージョンとの互換性オプションを定義できます。

  現在、このクラス/プロパティには、次の1つのサブプロパティが含まれます。

- **WhiteRtfBackground** プロパティ
  RTF の背景色を白色にするか、指定した色を使用するかを示す値を取得または設定します。デフォルト値は false です。

## 新しいデフォルトページ設定プロパティの追加

**C1PrintDocument** クラスに、次の新しいパブリック静的プロパティが追加されました。

- **UsePrinterForDefaultPageSettings** プロパティ
  デフォルトのページ設定をデフォルトのプリンタから取得するか（プリンタがネットワークプリンタの場合は処理速度が遅くなることがある）、現在のロケールに基づいて計算するかを示す値を取得または設定します。

**C1PrintDocument** のロードに時間がかかる場合は、システムにインストールされている（ネットワーク）デフォルトのプリンタが遅いことが原因である可能性があります。この場合は、**MeasurementDevice** を使用してC1PrintDocumentを作成する前に、このプロパティを false に設定してください。たとえば、次のように記述します。

**Visual Basic コードの書き方**

Visual Basic

```
C1PrintDocument.UsePrinterForDefaultPageSettings = False
C1PrintDocument.MeasurementDevice = MeasurementDeviceEnum.Screen
Dim doc As New C1PrintDocument()
```

**C# コードの書き方**

C#

```
C1PrintDocument.UsePrinterForDefaultPageSettings = false;
C1PrintDocument.MeasurementDevice = MeasurementDeviceEnum.Screen;
C1PrintDocument doc = new C1PrintDocument();
```

または、システムに高速のローカルプリンタと低速のデフォルトネットワークプリンタが存在している場合は、MeasurementPrinterName をローカルプリンタの名前に設定できます。これにより、C1PrintDocument は、測定およびデフォルトのページ設定の取得の両方でそのプリンタを使用します。たとえば、次のように記述します。


Copyright © GrapeCity, Inc. All rights reserved.

**Visual Basic コードの書き方**

```
Visual Basic
C1PrintDocument.UsePrinterForDefaultPageSettings = True
C1PrintDocument.MeasurementDevice = MeasurementDeviceEnum.Printer
C1PrintDocument.MeasurementPrinterName = "My local printer name"
Dim doc As New C1PrintDocument()
```

**C# コードの書き方**

```
C#
C1PrintDocument.UsePrinterForDefaultPageSettings = true;
C1PrintDocument.MeasurementDevice = MeasurementDeviceEnum.Printer;
C1PrintDocument.MeasurementPrinterName = "My local printer name";
C1PrintDocument doc = new C1PrintDocument();
```

## Style クラスへのアンビエントフォント関連プロパティの追加

スタイルでは、子スタイルで個々のフォントプロパティを指定/再定義できます。Style クラスには、次のパブリックアンビエントフォント関連プロパティが追加されました。

スタイルでは、子スタイルで個々のフォントプロパティを指定/再定義できます。Style クラスには、次のパブリックアンビエントフォント関連プロパティが追加されました。

- **FontName** プロパティ
  このフォントの名前を取得または設定します。
- **FontSize** プロパティ
  Unit プロパティで指定された単位で測定されるこのフォントの em サイズを取得または設定します。
- **FontBold** プロパティ
  このフォントが太字かどうかを示す値を取得または設定します。
- **FontItalic** プロパティ
  このフォントが斜体かどうかを示す値を取得または設定します。
- **FontUnderline** プロパティ
  このフォントが下線かどうかを示す値を取得または設定します。
- **FontStrikeout** プロパティ
  このフォントが取り消し線かどうかを示す値を取得または設定します。

次のコードは、新しいプロパティの使用例を示します。

**Visual Basic コードの書き方**

```
Visual Basic
Dim rt As RenderTable = New RenderTable
rt.Style.Font = New Font("Arial", 12)
rt.Cells(3, 4).Style.FontBold = True
```

**C# コードの書き方**

```
C#
RenderTable rt = new RenderTable();
rt.Style.Font = new Font("Arial", 12);
rt.Cells[3, 4].Style.FontBold = true;
```


Copyright © GrapeCity, Inc. All rights reserved.

これで、セル(3, 4)のフォントが太字になり、その他のすべてのフォントプロパティは親オブジェクトから継承されます。

## テーブルおよびセルでグリッド線を繰り返すためのプロパティの追加

テーブルおよびセルに追加された次の新しいパブリックプロパティは、セルがページ間で分割される場合にグリッド線を繰り返し表示するかどうかを制御します。

- **RepeatGridLinesVert** プロパティ
  セルが分割されているときに、水平グリッド線を描画するかどうかを示す値を取得または設定します。
- **RepeatGridLinesHorz** プロパティ
  セルが分割されているときに、垂直グリッド線を描画するかどうかを示す値を取得または設定します。
- **RepeatGridLinesVert** プロパティ
  セルが分割されているときに、垂直グリッド線を描画するかどうかを示す値を取得または設定します。
- **RepeatGridLinesHorz** プロパティ
  セルが分割されているときに、水平グリッド線を描画するかどうかを示す値を取得または設定します。

追加された新しいパブリック列挙体:

- **ThreeStateEnum** 列挙体
  3つのいずれかの状態(未定義、true、false)で値を定義します。
  使用可能な値:
  - Undefined - 値は未定義です。
  - True - 値は true です。
  - False - 値は false です。

## RenderC1Printable クラスへの新しいメソッドの追加

**RenderC1Printable** クラスには、次の新しいメソッドが追加されました。

- **RenderC1Printable.GenerateInnerRenderObjects メソッド**

  印刷されるオブジェクト(Object プロパティに割り当て)を表す内部レンダリングオブジェクトを強制的に生成します。生成されたレンダリングオブジェクトには、Children プロパティを使用してアクセスできます。

- **RenderC1Printable.ResetInnerRenderObjects メソッド**

  印刷されるオブジェクトを表す内部レンダリングオブジェクトを削除します。

RenderC1Printable を介して**C1PrintDocument** に他の C1 コンポーネントを挿入する場合は、C1 コンポーネント(C1TrueDBGrid または C1FlexGrid など)を表す内部レンダリングオブジェクトにアクセスする必要がある場合があります。内部レンダリングオブジェクトは、それを格納するドキュメントが生成されるまで作成されなかったため、アクセスは困難でした。それが、この新しいメソッドによって容易になります。次のコードは、この新しいメソッドを使用して、C1TrueDBGrid を表す RenderTable を生成し、テーブルの幅を 50% に設定し、そのテーブルをドキュメントに直接挿入する方法を示します(RenderC1Printable を「ラッパー」として使用しません)。

**Visual Basic コードの書き方**

Visual Basic

```
Dim grid As C1TrueDBGrid = New C1TrueDBGrid
' ... グリッドプロパティを設定します
Dim c1pr As RenderC1Printable = New RenderC1Printable
c1pr.Object = grid
c1pr.GenerateInnerRenderObjects
'' 次の行は、グリッドが単一レンダリングオブジェクト
' (RenderTable)で表されるとしています。
```


Copyright © GrapeCity, Inc. All rights reserved.

```
c1pr.Children(0).Width = "50%"
Dim doc AsC1PrintDocument= NewC1PrintDocument
doc.Body.Children.Add(c1pr.Children(0))
```

**C# コードの書き方**

C#

```
C1TrueDBGrid grid = new C1TrueDBGrid();
'// ... グリッドプロパティを設定します
RenderC1Printable c1pr = new RenderC1Printable();
c1pr.Object = grid;
c1pr.GenerateInnerRenderObjects();
'// 次の行は、グリッドが単一レンダリングオブジェクト
// (RenderTable)で表されるとしています。
c1pr.Children[0].Width = "50%";
C1PrintDocument doc = new C1PrintDocument();
doc.Body.Children.Add(c1pr.Children[0]);
```

## レンダリングオブジェクトの分割動作の強化

オブジェクトが1ページに収まらない場合にのみオブジェクトをページ間で分割するための新しいオプションが追加されました。この操作を容易にするため、次の変更が行われました。

- **SplitBehaviorEnum** 列挙体が追加され、次の要素が含まれます。
  - **Never**
    オブジェクトはページ間で分割されません。つまり、必ず1ページに一緒に表示されます。
  - **SplitIfNeeded**
    現在のページにオブジェクトが収まらない場合は、オブジェクトが分割されます。
  - **SplitIfLarge**
    オブジェクトが大きすぎて空のページに収まらない場合にのみ、オブジェクトが分割されます。その場合は、オブジェクトが即座に分割されます。そうでない場合は、新しいページが開始され、そのページにオブジェクトが分割されずに配置されます。
- **RenderObject** クラスに、次の新しいプロパティが追加されました。
  - **SplitHorzBehavior** プロパティ
  - **SplitVertBehavior** プロパティ
- 古いブール値プロパティの **CanSplitVert** および **CanSplitHorz** は、古い形式としてマークされています。現在、これらは新しいプロパティのラッパーとして機能します。
  - CanSplit を true に設定すると、対応する Split*Behavior が SplitIfNeeded に設定されます。
  - CanSplit を false に設定すると、対応する Split*Behavior が Never に設定されます。
  - CanSplit をクエリーすると、対応する Split*Behavior が Never である場合に false が返されます。そうでない場合は true が返されます。

## テーブルの自動サイズ変更のサポート

テーブルの自動サイズ変更をサポートするために、次の変更が行われました。

- **TableSizingModeEnum** 列挙体が追加され、次の要素が含まれます。
  - **Default**テーブルレベルでは、行のデフォルトが Auto、列のデフォルトが Fixed です。行／列レベルでは、デフォルトはテーブルを継承します。
  - **Auto**
    サイズ(行の場合は高さ、列の場合は幅)は、コンテンツに基づいて自動的に計算されます。
  - **Fixed**
    サイズ(行の場合は高さ、列の場合は幅)を明示的に指定する必要があります。省略すると、使用可能なス


Copyright © GrapeCity, Inc. All rights reserved.

ペースと行／列数によって決定されます。

- 次の新しいパブリックプロパティが追加されました。
  - **ColumnSizingMode** プロパティ
    テーブル列の幅が Unit.Auto と指定された場合の計算方法を示す値を取得または設定します。
  - **RowSizingMode** プロパティ
    テーブル行の高さが Unit.Auto と指定された場合の計算方法を示す値を取得または設定します。
  - **SizingMode** プロパティ
    現在のベクター（行または列）のサイズの計算方法を指定する値を取得または設定します。このプロパティは、ベクターのサイズ（列の幅または行の高さ）が Unit.Auto と指定されている場合にのみ使用されます。

## テーブル行の引き伸ばしの強化

以前のバージョンでは、ページの最後の列をページの端まで引き伸ばすように指定できました。このバージョンでは、行に対しても指定できます。この操作を容易にするため、次の変更が行われました。

- **StretchTableEnum** 列挙体が変更されました。新しい要素が追加され、古い要素は古い形式としてマークされます。ただし、すべての変更は下位互換性があります。
  - **None**
    行/列は引き伸ばされません。
  - **AllVectors**
    すべての行/列がページ全体に合わせて引き伸ばされます。
  - **AllColumns**
    AllVectors に相当し、古い形式としてマークされています。
  - **LastVectorOnPage**
    ページの最後の行/列だけがページの余白に合わせて引き伸ばされます。
  - **LastColumnOnPage**
    LastVectorOnPage に相当し、古い形式としてマークされています。
- **StretchRowEnum** 列挙体が追加され、次の要素が含まれます（StretchColumnEnum 列挙体の要素に類似）。
  - **Default**
    行引き伸ばしモードは、テーブルの StretchRows プロパティによって決定されます。
  - **Yes**
    行は引き伸ばされます。
  - **No**
    行は引き伸ばされません。
  - **IfLastOnPage**
    ページの最後の行だけが引き伸ばされます。
- 次の新しいプロパティが追加されました。
  - **Stretch** プロパティ
    現在行の引き伸ばしモードを取得または設定します。
  - **StretchColumns** プロパティ
    列の引き伸ばしモードを取得または設定します。
  - **StretchRows** プロパティ
    行の引き伸ばしモードを取得または設定します。
- **Stretch** プロパティは、古い形式としてマークされています。代わりに、**StretchColumns** を使用してください。

## スクリプトオプションのサポート

スクリプトオプションをサポートするための **ScriptingOptions** クラスが追加されました。**ScriptingOptions** クラスは、C1PrintDocument のスクリプトオプションを定義します。このクラスには、次のパブリックプロパティが含まれます。

- **ExternalAssemblies** プロパティ
  現在のドキュメントのスクリプトで使用できる外部アセンブリの一覧を取得します。
- **Namespaces** プロパティ
  現在のドキュメントのスクリプトで使用できる名前空間の一覧を取得します。


Copyright © GrapeCity, Inc. All rights reserved.

**C1PrintDocument** クラスには次の新しいプロパティも追加されました。

- **ScriptingOptions** プロパティ
  ドキュメントのスクリプトオプションを取得します。

## テキストのレンダリング

回転されたテキスト（**TextAngle** が 0 でない）のレンダリングが強化されました。

使用例は、**RotatedText** サンプルを参照してください。

## RenderTable

解決後に行の高さまたは列の幅が 0 になる場合、行または列はまったく表示されなくなります。これは、行または列の **Visible** プロパティが **False** に設定されていることと同じです。以前のバージョンとは異なり、行または列のグリッド線も表示されなくなります。

使用例は、**ZeroWidthRowsCols** サンプルを参照してください。

## ドキュメント生成時のエラー処理

ドキュメント要素のサイズおよび位置に競合する値を指定できるようになりました。次のコードに典型的な例を示します。

**Visual Basic コードの書き方**

Visual Basic

```
Dim rt As RenderTable = New RenderTable
rt.Width = "100mm"
rt.Cols(0).Width = "20mm"
rt.Cols(0).Width = "20mm"
```

**C# コードの書き方**

C#

```
RenderTable rt = new RenderTable();
rt.Width = "100mm";
rt.Cols[0].Width = "20mm";
rt.Cols[0].Width = "20mm";
```

このコードでは、テーブルに2つだけ列があり、明示的に指定されたテーブルの幅よりサマリの幅が小さくなっています。この場合、以前のバージョンではドキュメントが生成されず、**ThrowExceptionOnError** プロパティの値に基づき、例外が生成されるか、**Generate/StartDoc** メソッドが **False** を返しました。デフォルトでは、**ThrowExceptionOnError** に設定されいたため、ほとんどの場合、このようなコードでは例外が生成されるだけで、ユーザーにとってストレスの種でした。また、複雑な状況では、問題を診断して原因を見つけることが困難でした。

2006 v3 バージョンでは、この問題が修正されています。現在は、どのような場合でも、競合するパラメータがコンテキストに基づいて調整され、ドキュメントが生成されます。上のコード例では、テーブルの幅に合わせて各列の幅が同じ割合で拡大されます。また、**ThrowExceptionOnError** プロパティのデフォルト値は False に変更されました。

## スクリプト

2006 v3 リリースより、C1PrintDocument ではドキュメント要素内でスクリプトおよび式を使用できるようになりました。現在のところ、式は VB.NET で記述され、RenderText、ParagraphText（RenderParagraph、RenderTocItem）、および DataBinding レンダリングオブジェクトで使用できます。

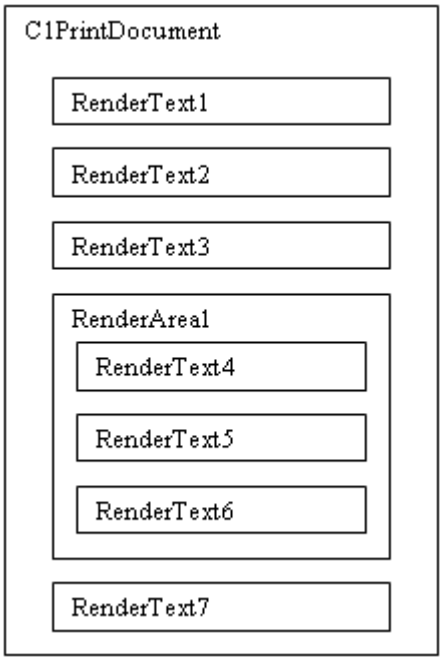

Copyright © GrapeCity, Inc. All rights reserved.

レンダリングオブジェクトでスクリプトを使用する方法については、「式、スクリプト、タグ」を参照してください。

## データ連結

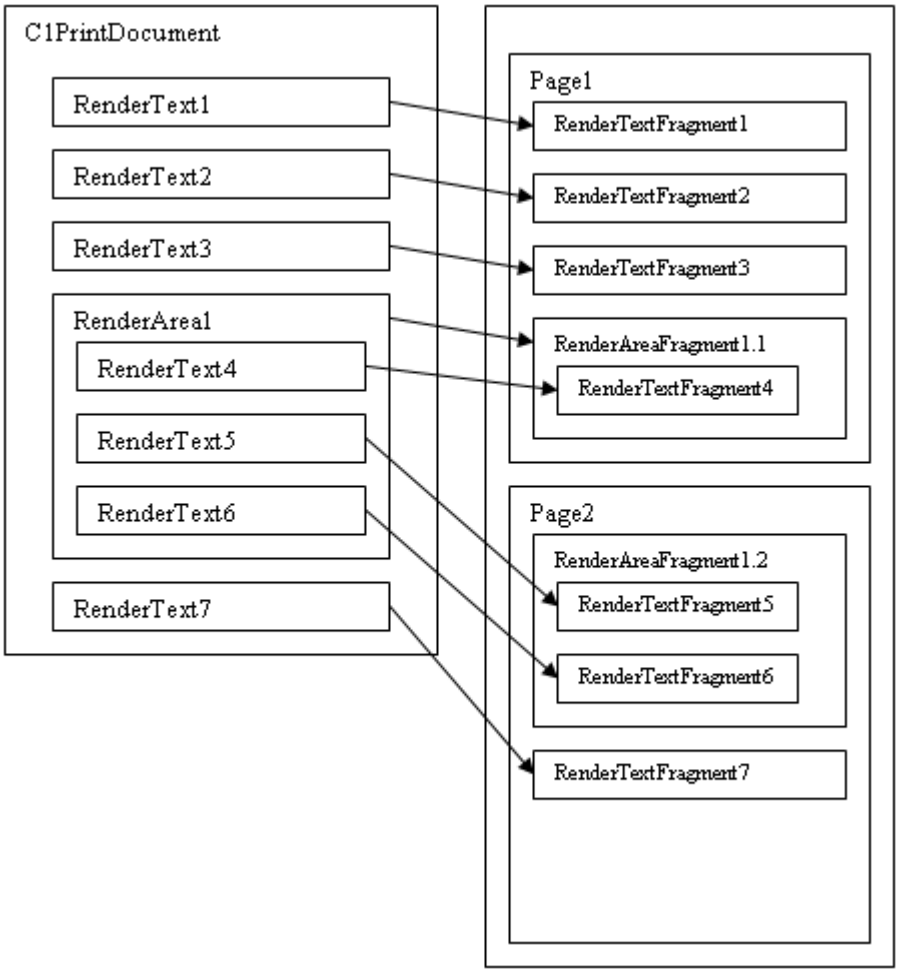
以前のバージョンでは、C1PrintDocument の生成プロセスにより、次のような変換が行われていました。

最初、C1PrintDocument（まだ生成されていない）は、RenderObject ベースのオブジェクトのツリーとして、次の構造で表されます。




ドキュメント生成時には、ドキュメントのページ区切りが実行され、この間に各 RenderObject は1つ以上の RenderFragment を生成します。RenderFragment は、特定のページに配置される元の RenderObjcet の一部を記述します。その結果、次の図のような構造になります。


Copyright © GrapeCity, Inc. All rights reserved.




C1PrintDocument にデータ連結を追加すると、C1DataBinding 型の RenderObject クラスに新しい複合プロパティが追加されます。これにより、レンダリングオブジェクトで表示されるデータのデータソースを指定できます。また、データ連結では、データソース内のすべてのレコードに対してレンダリングオブジェクトを繰り返すように指定できます。この場合、レンダリングオブジェクトは、水平区切りレポートジェネレータの「領域」と同じになります。この方法/構造は、Microsoft の RDL 定義と同じです。

このように、データ連結が追加されたことで（2006 v3 バージョン）、ドキュメントの生成には次の2つの段階が含まれます。

- すべてのデータ連結レンダリングオブジェクトが選択され、これらを（テンプレートとして）使用して、データに基づいて「実際の」レンダリングオブジェクトが作成されます。
- 生成されるドキュメントは、以前のバージョンと同様にページ区切りされます。

ドキュメントには、データベーススキーマ（C1DataSchema クラスによって表され、データベース接続情報、SQL クエリーなどを含む）を含めることができます。ドキュメント内の C1DataBinding オブジェクトは、そのスキーマのプロパティを参照できます。ドキュメント内のすべてのデータ連結オブジェクトがドキュメント自体の C1DataSchema のプロパティだけを参照する場合、そのドキュメントは「データのリフロー可能」になります。つまり、ドキュメントを作成したプログラムとは関係なく、データを完全に更新してドキュメントを再生成することができます。

また、C1DataBinding は既存のデータソース（DataTable など）を参照できます。これは、フォーム上またはその**C1PrintDocument**を作成したプログラムの任意の場所で作成されたものです。この場合は、そのプログラムのコンテキスト内でのみデータを更新してドキュメントを再生成できます。そのドキュメントを保存した後で（.C1D ファイルとして）読み込むと、そのデータとの接続はすべて失われます。

詳細については、「データ連結」を参照してください。

## ローカライズ

C1Preview はローカライズをサポートするようになりました。これにより、エンドユーザーに表示されるすべての文字列を開発


Copyright © GrapeCity, Inc. All rights reserved.

者がローカライズ(翻訳)できます。C1Preview のローカライズは、Localizable プロパティを使用した .NET Windows フォームで採用されている標準のローカライズと同じ方法に基づきます。Windows フォームと同様に、各 C1Preview アセンブリ(C1.C1Preview.2.dll または C1.Win.C1Preview.2.dll)に対して、カルチャごとに個別ファイルを使用してリソースファイル(.resx 拡張子)セットを作成できます。アプリケーションの実行時に、これらのリソース(言語)を切り替えることができます。ローカライズされたリソースは、個々の C1Preview コンポーネントではなく、プレビューアセンブリ全体(C1.C1Preview.2.dll または C1.Win.C1Preview.2.dll)にアタッチされることに注意してください。また、C1Preview .dll のコンポーネントを使用するアプリケーションのすべての部分で、同じローカライズリソースを使用する必要があります。

resx ファイルを作成する際は、次の規則に従うことをお勧めします。

- すべての .resx ファイルをプロジェクトの C1LocalizedResources サブフォルダに格納します。
- ファイル名は次のように指定します。
  XXX.YYY.resx。ここで、XXX は ComponentOne アセンブリの名前、YYY はリソースのカルチャコードです。
  たとえば、次のように記述します。
  C1.C1Preview.2.de.resx – C1.C1Preview.2 アセンブリのドイツ語(de)リソース。
  C1.Win.C1Preview.2.resx – C1.Win.C1Preview.2 アセンブリの不変カルチャリソース。

リソースファイルは、新しい設計時エディタを使用して簡単に作成できます。このエディタには、プレビューコンポーネントの[ローカライズ]コンテキストメニュー/アクション項目からアクセスできます。詳細については、「ローカライズ」を参照してください。

実行時に現在のカルチャを切り替える場合は、次のコードを使用すると、ドイツ語のロケールに切り替えることができます。

**Visual Basic コードの書き方**

```
Visual Basic
Thread.CurrentThread.CurrentUICulture = New CultureInfo("de")
```

**C# コードの書き方**

```
C#
Thread.CurrentThread.CurrentUICulture = new CultureInfo("de");
```

# クラスメンバ

## クラスのメンバ

**Preview for .NET** に追加された新しいメンバを次に示します。

## 新しいメンバ

| クラス | メンバ | 説明 |
|---|---|---|
| **C1PrintDocument** | **Load** メソッド | 現在のドキュメントをストリームまたはファイルからロードします。 |
| | **Save** メソッド | 現在のドキュメントをストリームまたはファイルに保存します。 |
| | **FromFile** メソッド | 指定したファイルから**C1PrintDocument**の新しいインスタンスを作成します。 |
| | **FromStream** メソッド | 指定したデータストリームから |


Copyright © GrapeCity, Inc. All rights reserved.

| | | |
|---|---|---|
| | | **C1PrintDocument** の新しいインスタンスを作成します。 |
| | **UseGdiPlusTextRendering** プロパティ | GDI+ テキスト API を使用して現在のドキュメント内のすべてのテキストを表示するかどうかを示す値を取得または設定します。 |
| **C1Report** | **RecordsPerPage** プロパティ | ページまたはグループごとに表示する詳細記録の数を取得または設定します。たとえば、この値を 5 に設定した場合、レポートはグループまたはページごとに5つの詳細記録を表示します。必要に応じて、ページ／グループを満たすために空（空白）の記録が追加されます。これは、特定の行数を含むフォームの印刷時に便利です。 |
| **PdfExporter** | **ViewerPreferences** プロパティ | 生成された PDF ドキュメントを開くときに使用するビューア環境設定を説明する Pdf**ViewerPreferences** オブジェクトを取得します。< br/> ビューアで開くときにエクスポートされた PDF ファイルを表示する方法を制御する ViewerPreferences オブジェクトのプロパティを設定します。 |
| | **PdfViewerPreferencesFormClass** プロパティ | PdfViewerPreferences クラスのフィールドを編集するために使用するフォームのタイプを取得または設定します。このフォームはオプションフォームから呼び出されます。 |
| **PdfFilter** | **PdfSecurityOptions** プロパティ | ドキュメントの編集および印刷を行うためのパスワード（オーナーおよびユーザー）およびアクセス権を設定するためのプロパティを含むオブジェクトを返します。 |
| | **PdfViewerOptions** プロパティ | ビューアがそのメニューを表示または非表示にするかどうか、初期ページレイアウトをどうするかなど、ビューアオプションを設定するためのプロパティを含むオブジェクトを返します。 |
| **Style** | **JustifyLastLine** プロパティ | 現在のスタイルで **TextAlignHorz** が Justify ま |


Copyright © GrapeCity, Inc. All rights reserved.

| | | |
|---|---|---|
| | | たは JustifyChars に設定されている場合、テキストの最後の行を両端揃えするかどうかを示す値を取得または設定します。< br /> このプロパティは、現在廃止として指定され、使用されない TreatLastLineNotEndOfParagraph プロパティに優先することに注意してください。 |
| | **JustifyEndOfLines** プロパティ | 現在のスタイルで **TextAlignHorz** が Justify または **JustifyChars** に設定されている場合、新しい行の文字で終了するテキストの行を両端揃えするかどうかを示す値を取得または設定します。< br /> このプロパティは、現在廃止として指定され、使用されない EolIsEndOfParagraph プロパティに優先することに注意してください。 |
| | **CharSpacing** プロパティ | テキストの文字間のスペースを指定する値を取得または設定します。デフォルト値は 0 です。<br>ドキュメントの **UseGdiPlusTextRendering** が False に設定されている場合、このプロパティは無視されることに注意してください（この制限は今後のバージョンで削除されます）。 |
| | **CharWidth** プロパティ | テキストの文字の幅を拡大／縮小する量（パーセント）を取得または設定します。デフォルト値は 100 です。<br>ドキュメントの **UseGdiPlusTextRendering** が False に設定されている場合、このプロパティは無視されることに注意してください（この制限は今後のバージョンで削除されます）。 |

| メンバ | 説明 |
|---|---|
| **C1DocumentFormatEnum** 列挙体 | ドキュメント永続性形式を定義します。要素は次のとおりです。<br>• C1d：元の **Preview for .NET** 2.0 形式。 |


Copyright © GrapeCity, Inc. All rights reserved.

| | |
|---|---|
| | • C1dx：新しい Open Packaging Conventions 準拠形式。 |
| C1PrintDocument.CompatibilityOptions プロパティ | **C1PrintDocument**の以前のバージョンとの互換性を判定する CompatibilityOptions オブジェクトを取得します。 |
| C1PrintDocument.DocumentDirectoryName プロパティ | C1PrintDocument.DocumentFileName のディレクトリを取得します。 |
| C1PrintDocument.DocumentFileName プロパティ | 現在のドキュメントに関連付けられているファイル名を取得します。つまり、ドキュメントは、そのファイルから読み込まれているか、そのファイルに保存されています。 |
| C1PrintDocument.ImportC1Report メソッド | 現在のドキュメントに C1Report レポート定義をインポートします。 |
| C1PrintDocument.MaxPages プロパティ | 生成されるドキュメントの最大ページ数を制限する値を取得または設定します。 |
| C1PrintDocument.UsePrinterForDefaultPageSettings プロパティ | デフォルトのページ設定をデフォルトのプリンタから取得するか（プリンタがネットワークプリンタの場合は処理速度が遅くなることがある）、現在のロケールに基づいて計算するかを示す値を取得または設定します。 |
| CompatibilityOptions.HeaderFooterWidthAsTotalColumnWidths プロパティ | ページヘッダーおよびフッターの幅がページのすべての列の幅の合計と等しいかどうかを示す値を取得または設定します。デフォルト値は false です。このプロパティを true に設定すると、C1Report との互換性が向上します。 |
| CompatibilityOptions.WhiteRtfBackground プロパティ | RTF の背景色を白色にするか、指定した色を使用するかを示す値を取得または設定します。 |
| DataSet.MaxRecords プロパティ | データセットから取得されるレコードの最大数を取得または設定します。デフォルトは -1 です。これは、すべてのレコードを取得します。 |
| DataSet.RowFilter プロパティ | ドキュメントに含めるデータ行をフィルタするために使用される式を取得または設定します。このプロパティは、基底の DataTable オブジェクトの RowFilter プロパティに割り当てられます。 |
| ImageExporter.Resolution プロパティ | エクスポートされた画像の解像度を取得または設定します。 |
| RenderArea.HeaderHorz プロパティ | 現在のレンダリング領域が広すぎて1ページに収まらないためにレンダリング領域が水平方向に分割される場合、すべての「水平方向」のページで繰り返し表示されるレンダリングオブジェクトを取得または設定します。 |
| RenderArea.HeaderVert プロパティ | 現在のレンダリング領域が高すぎて1ページに収まらないためにレンダリング領域が垂直方向に分割される場合、すべてのページ（または |


Copyright © GrapeCity, Inc. All rights reserved.

| | |
|---|---|
| | 列）で繰り返し表示されるレンダリングオブジェクトを取得または設定します。 |
| RenderBarCode クラス | バーコードオブジェクトを表します。 |
| RenderBarCode.BarCodeType プロパティ | バーコードの生成時に使用するエンコーディングのタイプを取得または設定します。BarCodeType.None は、このプロパティの有効な値ではありません。 |
| RenderBarCode.BarDirection プロパティ | バーコードの方向を取得または設定します。 |
| RenderBarCode.ShowText プロパティ | Text プロパティを表す文字列をバーコード画像に並べて表示するかどうかを示す値を取得または設定します。 |
| RenderBarCode.Text プロパティ | バーコード画像としてエンコードされる値を取得または設定します。 |
| RenderC1Printable.GenerateInnerRenderObjects メソッド | 印刷されるオブジェクト（Object プロパティに割り当て）を表す内部レンダリングオブジェクトを強制的に生成します。生成されたレンダリングオブジェクトには、Children プロパティを使用してアクセスできます。 |
| RenderC1Printable.ResetInnerRenderObjects メソッド | 印刷されるオブジェクトを表す内部レンダリングオブジェクトを削除します。 |
| RenderField クラス | C1Report の Field オブジェクトに類似するレンダリングオブジェクトを表します。 |
| RenderField.Calculated プロパティ | Text プロパティをリテラル値と計算式のどちらとして解釈するかを指定します。 |
| RenderField.CanGrow プロパティ | 現在のフィールドの高さをフィールドのコンテンツに合わせて自動的に大きくするかどうかを示す値を取得または設定します。 |
| RenderField.CanShrink プロパティ | 現在のフィールドの高さをフィールドのコンテンツに合わせて自動的に小さくするかどうかを示す値を取得または設定します。 |
| RenderField.SubReport プロパティ | 現在のフィールド内にレンダリングされる RenderReport オブジェクトへの参照を取得または設定します。 |
| RenderField.Text プロパティ | 現在のフィールドのテキストを取得または設定します。 |
| RenderObject.KeepTogether プロパティ | 現在のオブジェクトと同じページに印刷されるレンダリングオブジェクトを取得または設定します。指定されたオブジェクトは、現在のオブジェクトと同じ親を持つ必要があります。 |
| RenderObject.Visibility プロパティ | オブジェクトの表示状態を示す値を取得または設定します。 |
| RenderReport クラス | RenderField オブジェクト内にネストされ、RenderRenderField.SubReport プロパティで指定されるサブレポートである C1Report オブ |


Copyright © GrapeCity, Inc. All rights reserved.

| | |
|---|---|
| | ジェクトを表します。 |
| RenderReport.ExposeScriptObjects プロパティ | スクリプトオブジェクトをネストされたオブジェクトに公開するかどうかを示す値を取得または設定します。 |
| RenderReport.GrowShrinkMode プロパティ | RenderField.CanGrow プロパティと RenderField.CanShrink プロパティの処理方法を示す値を取得または設定します。 |
| RenderSection クラス | C1Report の Section オブジェクトに類似するレンダリングオブジェクトを表します。 |
| RenderSection.CanGrow プロパティ | 現在のセクションの高さをフィールドのコンテンツに合わせて自動的に大きくするかどうかを示す値を取得または設定します。 |
| RenderSection.CanShrink プロパティ | 現在のセクションの高さをフィールドのコンテンツに合わせて自動的に小さくするかどうかを示す値を取得または設定します。 |
| RenderSection.OnFormatScript プロパティ | 現在のセクションが書式設定される前に実行されるスクリプトを含む文字列を取得または設定します。このスクリプトはレコードごとに実行されるため、レコードデータに基づいてセクションのコンテンツを変更するために使用できます。 |
| RenderSection.OnPrintScript プロパティ | 現在のセクションが書式設定された後で、印刷される前に実行されるスクリプトを含む文字列を取得または設定します。このスクリプトは、レイアウトに影響を及ぼさないセクション（テキスト、背景色など）のビジュアルプロパティを調整するために使用できます。 |
| RenderTable.RepeatGridLinesHorz プロパティ | セルが分割されているときに、水平グリッド線を描画するかどうかを示す値を取得または設定します。 |
| RenderTable.RepeatGridLinesVert プロパティ | セルが分割されているときに、垂直グリッド線を描画するかどうかを示す値を取得または設定します。 |
| Style.FontBold プロパティ | このフォントが太字かどうかを示す値を取得または設定します。 |
| Style.FontItalic プロパティ | このフォントが斜体かどうかを示す値を取得または設定します。 |
| Style.FontName プロパティ | このフォントの名前を取得または設定します。 |
| Style.FontSize プロパティ | Unit プロパティで指定された単位で測定されるこのフォントの em サイズを取得または設定します。 |
| Style.FontStrikeout プロパティ | このフォントが取り消し線かどうかを示す値を取得または設定します。 |
| Style.FontUnderline プロパティ | このフォントが下線かどうかを示す値を取得または設定します。 |


Copyright © GrapeCity, Inc. All rights reserved.

| | |
|---|---|
| TableCell.RepeatGridLinesHorz プロパティ | セルが分割されているときに、垂直グリッド線を描画するかどうかを示す値を取得または設定します。 |
| TableCell.RepeatGridLinesVert プロパティ | セルが分割されているときに、水平グリッド線を描画するかどうかを示す値を取得または設定します。 |
| ThreeStateEnum 列挙体 | 3つのいずれかの状態（未定義、true、false）で値を定義します。 |
| VisibilityEnum 列挙体 | オブジェクトの表示状態を指定します。 |
| WinClassic Member@FormsStyleEnum 列挙体 | Windows クラシックコントロールスタイル。 |
| WinXP Member@FormsStyleEnum 列挙体 | Windows XP（テーマ）コントロールスタイル。 |
| メンバ | 説明 |
| C1PrintDocument.FormsStyle プロパティ | ドキュメントの RenderInput オブジェクトの外観を取得または設定します。 |
| C1PrintDocument.ScriptingOptions プロパティ | このC1PrintDocumentオブジェクトのスクリプトオプションを定義する ScriptingOptions オブジェクトを取得します。 |
| FormsStyleEnum 列挙体 | RenderInput オブジェクトの外観の種類を定義します。 |
| RenderObject.SplitHorzBehavior プロパティ | オブジェクトが水平方向に分割される方法を決定する値を取得または設定します。 |
| RenderObject.SplitVertBehavior プロパティ | オブジェクトが垂直方向に分割される方法を決定する値を取得または設定します。 |
| RenderTable.ColumnSizingMode プロパティ | テーブル列の幅を計算する方法を決定する値を取得または設定します。 |
| RenderTable.RowSizingMode プロパティ | テーブル行の高さを計算する方法を決定する値を取得または設定します。 |
| RenderTable.StretchColumns プロパティ | 列の引き伸ばしモードを取得または設定します。 |
| RenderTable.StretchRows プロパティ | 行の引き伸ばしモードを取得または設定します。 |
| ScriptingOptions クラス | **C1PrintDocument** のスクリプトオプションを定義します。 |
| ScriptingOptions.ExternalAssemblies プロパティ | 現在のドキュメントのスクリプトで使用できる外部アセンブリの一覧を取得します。 |
| ScriptingOptions.Namespaces プロパティ | 現在のドキュメントのスクリプトで使用できる名前空間の一覧を取得します。 |
| ScriptingOptions.AssignFrom メソッド | 別の Scripting オブジェクトからプロパティをコピーします。 |
| ScriptingOptions.Reset メソッド | この Scripting オブジェクトのプロパティをデフォルト値にリセットします。 |


Copyright © GrapeCity, Inc. All rights reserved.

| | |
|---|---|
| StretchRowEnum 列挙体 | 行の引き伸ばしモードを定義します。 |
| SplitBehaviorEnum 列挙体 | レンダリングオブジェクトの分割動作を定義します。 |
| TableRow.Stretch プロパティ | 現在行の引き伸ばしモードを取得または設定します。 |
| TableSizingModeEnum 列挙体 | テーブルのサイズ変更モードを定義します。 |
| TableVector.SizingMode プロパティ | 現在のベクターのサイズの計算方法を指定する値を取得または設定します。 |
| メンバ | 説明 |
| Aggregate クラス | ドキュメント生成時に計算された集計値を表します。 |
| Aggregate.DataBinding プロパティ | 算術関数が計算されるデータ連結を取得または設定します。 |
| Aggregate.Expression プロパティ | 集計値を計算するために使用される式を取得します。 |
| Aggregate.ExpressionText プロパティ | 集計値を計算するために使用される式のテキストを取得します。 |
| Aggregate.Func プロパティ | 計算する集計関数を取得または設定します。 |
| Aggregate.IgnoreNullValues プロパティ | 集計の計算時に null 値を無視するかどうかを指定する値を取得または設定します。 |
| Aggregate.Name プロパティ | コレクション内の NamedCollectionItem オブジェクトの一意の名前を取得または設定します。 |
| Aggregate.Owner プロパティ | このオブジェクトを含む AggregateCollection オブジェクトを取得します。 |
| Aggregate.Running プロパティ | すべてのレコードに対して集計を実行するか、グループごとに集計を実行するかを示す値を取得または設定します。 |
| Aggregate.Value プロパティ | フィールドの値を取得します。 |
| Aggregate.AssignFrom メソッド | 別の Aggregate オブジェクトからプロパティをコピーします。 |
| Aggregate.Clone メソッド | このオブジェクトのクローンを作成します。作成されたオブジェクトは、このオブジェクトと同じ名前になりますが、このオブジェクトの所有者には属しません。作成されたオブジェクトの所有者は null になります。 |
| AggregateCollection クラス | Field オブジェクトのコレクションです。 |
| AggregateCollection.Item プロパティ(Int32) | インデックスに基づいて集計を列挙体します。 |
| AggregateCollection.Item プロパティ(文字列) | 名前に基づいて集計を列挙体します。 |
| AggregateCollection.Owner プロパティ | この AggregateCollection オブジェクトを含む C1DataSchema オブジェクトを取得します。 |


Copyright © GrapeCity, Inc. All rights reserved.

| | |
|---|---|
| AggregateFuncEnum 列挙体 | サポートされる集計関数のセットを定義します。 |
| C1DataBinding クラス | **C1PrintDocument** 内のオブジェクトのデータ連結プロパティを定義します。現在のところ、RenderObject および TableVectorGroup オブジェクトだけがこの型の DataBinding プロパティを持ちます。 |
| C1DataBinding.DataSource プロパティ | カスタムデータを提供するオブジェクトを取得または設定します。現在のところ、このプロパティには、Ienumerable および C1.C1Preview.DataBinding.DataSet 型のオブジェクトを割り当てることができます。 |
| C1DataBinding.Defined プロパティ | この C1DataBinding オブジェクトが空でなく、何らかのデータにリンクされていることを示す値を取得します。 |
| C1DataBinding.Document プロパティ | この C1DataBinding オブジェクトを含む C1PrintDocumentオブジェクトを取得します。 |
| C1DataBinding.Fields プロパティ | データフィールドのコレクションを取得します。 |
| C1DataBinding.Filters プロパティ | リンクされたデータに適用されるフィルタを含む式のコレクションを取得します。 |
| C1DataBinding.Grouping プロパティ | データのグループ化方法を定義する Grouping オブジェクトを取得します。 |
| C1DataBinding.Parent プロパティ | 親の C1DataBinding オブジェクトを取得します。この値は、所有者オブジェクトの階層に基づいて計算されます。 |
| C1DataBinding.RowNumber プロパティ | 現在の行またはグループインスタンスの行番号を取得します。このプロパティは、ドキュメント生成時にのみ使用できます。ドキュメントが生成状態ではない場合、またはこの C1DataBinding オブジェクトの RenderObject の処理が開始されていない場合は、0 を返します。データが列挙体されている場合は行の番号を返し、データの列挙体が終了した場合は最後の行の番号を返します。 |
| C1DataBinding.Sorting プロパティ | データのソート方法を定義する Sorting オブジェクトを取得します。 |
| C1DataBinding.AssignFrom メソッド | 別のオブジェクトからプロパティをコピーします。 |
| C1DataSchema クラス | ドキュメントに埋め込まれるデータスキーマ（データソース、SQL クエリー、計算フィールド）を定義します。 |
| C1DataSchema.Aggregates プロパティ | ドキュメント生成時に計算される集計値に関する情報を含む AggregateCollection オブジェクトを取得します。 |
| C1DataSchema.DataSets プロパティ | ドキュメントの一部として取得されるデータセッ |


Copyright © GrapeCity, Inc. All rights reserved.

| | |
|---|---|
| | トに関する情報を含む DataSetCollection オブジェクトを取得します。 |
| C1DataSchema.DataSources プロパティ | さまざまな DataSet のデータのソースへの接続方法に関する情報を含む DataSourceCollection オブジェクトを取得します。 |
| C1DataSchema.Owner プロパティ | このデータスキーマを含むC1PrintDocument オブジェクトを取得します。 |
| C1DataSchema.Clear メソッド | C1DataSchema オブジェクトをクリアし、定義されているすべての DataSets、DataSources、および Aggregates を削除します。 |
| C1PrintDocument.DataSchema プロパティ | データスキーマ組み込みドキュメントを含むこのC1PrintDocumentオブジェクトによって所有されている C1DataSchema オブジェクトを取得します。 |
| CommandTypeEnum 列挙体 | 許可されるクエリータイプを記述します。 |
| ConnectionProperties クラス | データソースへの接続方法に関する情報を含みます。 |
| ConnectionProperties.ConnectString プロパティ | データソースの接続文字列を取得または設定します。 |
| ConnectionProperties.DataProvider プロパティ | データソースの型を取得または設定します。これは、ConnectionProperties.ConnectString および Query.CommandText の構文を決定します。 |
| ConnectionProperties.IntegratedSecurity プロパティ | このデータソースへの接続で統合セキュリティを使用するかどうかを示す値を取得または設定します。 |
| ConnectionProperties.AssignFrom メソッド | 別の ConnectionProperties オブジェクトからプロパティをコピーします。 |
| DataProviderEnum 列挙体 | データソースの型です。これは、ConnectionProperties.ConnectString および Query.CommandText の構文を決定します。 |
| DataSet クラス | ドキュメントの一部として取得されたデータセットに関する情報を含みます。 |
| DataSet.Fields プロパティ | データセット内のフィールドを含む DataSetFieldCollection オブジェクトを取得します。フィールドのリストは空になる場合があります。この場合は、クエリーによって取得されるすべてのフィールドにアクセスできます。 |
| DataSet.Filters プロパティ | データセットのレコードに適用されたフィルタを含む ExpressionCollection コレクションを取得します。 |
| DataSet.Owner プロパティ | このオブジェクトを含む DataSetCollection オブジェクトを取得します。 |


Copyright © GrapeCity, Inc. All rights reserved.

| | |
|---|---|
| DataSet.Query プロパティ | データソースからデータを取得するために必要な接続情報、クエリーなどのデータソースに関する情報を定義する Query オブジェクトを取得します。 |
| DataSet.AssignFrom メソッド | 別の DataSet オブジェクトからプロパティをコピーします。 |
| DataSetCollection クラス | DataSet オブジェクトのコレクションです。 |
| DataSetCollection.Add メソッド | コレクションに DataSet オブジェクトを追加します。 |
| DataSetField クラス | データモデル内のフィールドを表します。 |
| DataSetField.DataField プロパティ | クエリーのフィールドの名前を取得または設定します。DataField 名は一意でなくてもかまいません。複数のフィールドから同じデータフィールドを参照できます。DataField が指定されない場合は、フィールドの値を決定するために Expression が使用されます。 |
| DataSetField.Expression プロパティ | このフィールドの値を計算するために使用される式を定義する Expression オブジェクトを取得します。DataSetField では、Expression または DataField のいずれか（両方ではない）を定義する必要があります。両方とも定義した場合は、DataFiled が使用されます。 |
| DataSetField.ExpressionText プロパティ | 式のテキストを取得または設定します。このフィールドは、Expression.Text プロパティのショートカットです。 |
| DataSetField.Name プロパティ | コレクション内の NamedCollectionItem オブジェクトの一意の名前を取得または設定します。 |
| DataSetField.Owner プロパティ | このオブジェクトを含む NamedCollection オブジェクトを取得します。 |
| DataSetField.AssignFrom メソッド | 別の DataSetField オブジェクトからプロパティをコピーします。 |
| DataSetField.Clone メソッド | このオブジェクトのクローンを作成します。作成されたオブジェクトは、このオブジェクトと同じ名前になりますが、このオブジェクトの所有者には属しません。作成されたオブジェクトの所有者は null になります。 |
| DataSetFieldCollection クラス | DataSetField オブジェクトのコレクションです。コレクション内のすべてのフィールドは、一意の空でない名前にする必要があります。 |
| DataSetFieldCollection.Item プロパティ（Int32） | インデックスに基づいてフィールドを列挙体します。 |
| DataSetFieldCollection.Item プロパティ（String） | 名前に基づいてフィールドを列挙体します。 |
| DataSetFieldCollection.Owner プロパティ | この DataSetFieldCollection オブジェクトを含 |


Copyright © GrapeCity, Inc. All rights reserved.

| | |
|---|---|
| | む DataSet オブジェクトを取得します。 |
| DataSetFieldCollection.IndexOfName メソッド | NamedCollection から、指定された名前の NamedCollectionItem オブジェクトの 0 から始まるインデックスを返します。 |
| DataSource クラス | データソースに関する情報を含みます。 |
| DataSource.ConnectionProperties プロパティ | ConnectString、データプロバイダなどのデータソースへの接続方法に関する情報を含みます。 |
| DataSource.DataSchema プロパティ | この DataSource オブジェクトを含む C1DataSchema オブジェクトを取得します。 |
| DataSource.Owner プロパティ | この DataSource オブジェクトを含む DataSourceCollection オブジェクトを取得します。 |
| DataSource.Transaction プロパティ | このデータソースを使用するデータセットを1つのトランザクションで実行する必要があることを示します。 |
| DataSource.AssignFrom メソッド | 別の DataSource オブジェクトからプロパティをコピーします。 |
| DataSourceCollection クラス | DataSource オブジェクトのコレクションです。 |
| DataSourceCollection.Add メソッド | コレクションに DataSource オブジェクトを追加します。 |
| Expression クラス | グループ化、ソートなどで使用できる式を表します。 |
| Expression.Text プロパティ | 式のテキストを取得または設定します。 |
| Expression.AssignFrom メソッド | 別のオブジェクトからプロパティをコピーします。 |
| Grouping クラス | Grouping クラスは、データをグループ化するための式を定義します。 |
| Grouping.Defined プロパティ | グループ化が定義されているかどうかを示す値を取得します。 |
| Grouping.Expressions プロパティ | データをグループ化するための式を含む ExpressionCollection を取得します。 |
| Grouping.Owner プロパティ | このオブジェクトを含む C1DataBinding オブジェクトを取得します。 |
| Grouping.AssignFrom メソッド | 別のオブジェクトからプロパティをコピーします。Name プロパティと Owner プロパティはコピーされません。 |
| Query クラス | ドキュメントのデータを取得するために実行するクエリーの説明を含みます。 |
| Query.CommandText プロパティ | ドキュメントのデータを取得するために実行するクエリーを取得または設定します。 |
| Query.CommandType プロパティ | CommandText に含まれるクエリーの種類を |


Copyright © GrapeCity, Inc. All rights reserved.

| | |
|---|---|
| | 取得または設定します。 |
| Query.DataSource プロパティ | クエリーを実行する対象の DataSource オブジェクトを取得または設定します。 |
| Query.Parameters プロパティ | クエリーの一部としてデータソースに渡されるパラメータを含む QueryParameterCollection オブジェクトを取得します。 |
| Query.Timeout プロパティ | クエリーがタイムアウトするまでの秒数を取得または設定します。負でない値にする必要があります。省略したり 0 を指定した場合、クエリーはタイムアウトしません。 |
| Query.AssignFrom メソッド | 別の Query オブジェクトからプロパティをコピーします。 |
| QueryParameter クラス | クエリーの一部としてデータソースに渡されるパラメータに関する情報を含みます。 |
| QueryParameter.Name プロパティ | コレクション内の NamedCollectionItem オブジェクトの一意の名前を取得または設定します。 |
| QueryParameter.Owner プロパティ | このクエリーパラメータを含む QueryParameterCollection オブジェクトを取得します。 |
| QueryParameter.Value プロパティ | パラメータの値を取得または設定します。 |
| QueryParameter.AssignFrom メソッド | 別の QueryParameter オブジェクトからプロパティをコピーします。 |
| QueryParameter.Clone メソッド | このオブジェクトのクローンを作成します。作成されたオブジェクトは、このオブジェクトと同じ名前になりますが、このオブジェクトの所有者には属しません。作成されたオブジェクトの所有者は null になります。 |
| QueryParameterCollection クラス | QueryParameter オブジェクトのコレクションです。コレクション内のすべてのパラメータは、一意の空でない名前にする必要があります。 |
| QueryParameterCollection.Item プロパティ（Int32） | インデックスに基づいてパラメータを列挙体します。 |
| QueryParameterCollection.Item プロパティ（String） | 名前に基づいてパラメータを列挙体します。 |
| QueryParameterCollection.Owner プロパティ | この QueryParameterCollection オブジェクトを含む Query オブジェクトを取得します。 |
| QueryParameterCollection.IndexOfName メソッド | NamedCollection から、指定された名前の NamedCollectionItem オブジェクトの 0 から始まるインデックスを返します。 |
| RefreshModeEnum 列挙体 | ドキュメント生成時のデータと式のリフレッシュモードを記述します。 |
| RenderObject.Copies プロパティ | データ連結の結果として生成されるレンダリングオブジェクトのコレクションを取得します。オ |


Copyright © GrapeCity, Inc. All rights reserved.

| | |
|---|---|
| | ブジェクトがデータ連結でない場合、このプロパティの値は null です。 |
| RenderObject.DataBinding プロパティ | この RenderObject オブジェクトのデータ連結プロパティを表す C1DataBinding クラスのインスタンスを取得します。 |
| RenderObject.DataBindingDefined プロパティ | このレンダリングオブジェクトが DataBinding プロパティによるデータ連結であることを示す値を取得します。現在のレンダリングオブジェクトがデータ連結であるかどうかをテストするには、このプロパティを使用する必要があります。 |
| RenderObject.Original プロパティ | データ連結の処理中に、このオブジェクトの生成元の RenderObject オブジェクトを取得します。このようなオブジェクトが存在しない場合は、null を返します。 |
| SortDirectionEnum 列挙体 | ソートの種類を定義します。 |
| SortExpression クラス | ソート式を表します。 |
| SortExpression.Direction プロパティ | ソートの方向を取得または設定します。 |
| SortExpression.Expression プロパティ | 式を定義する Expression オブジェクトを取得します。 |
| SortExpression.ExpressionText プロパティ | ソート式のテキストを取得または設定します。 |
| SortExpression.AssignFrom メソッド | 別のオブジェクトからプロパティをコピーします。Owner プロパティはコピーされません。 |
| SortExpressionCollection クラス | SortExpression オブジェクトのコレクションです。 |
| SortExpressionCollection.Item プロパティ | コレクション内の指定されたインデックスにある項目を取得または設定します。 |
| Sorting クラス | Sorting クラスは、データをソートするための式を定義します。 |
| Sorting.Defined プロパティ | ソートが定義されていることを示す値を取得します。 |
| Sorting.Expressions プロパティ | ソート式のコレクションを取得します。 |
| Sorting.AssignFrom メソッド | 別のオブジェクトからプロパティをコピーします。Owner プロパティはコピーされません。 |
| TableVector.Visible プロパティ | テーブルの行または列の表示状態を制御します。 |
| Warning クラス | 競合の解決が必要な状況を記述します。 |
| Warning.Code プロパティ | エラーのカテゴリを取得します。 |
| Warning.Description プロパティ | エラーの詳細な説明を含みます。 |
| Warning.Source プロパティ | エラーを発生したオブジェクトを取得します。 |
| WarningCollection クラス | Warning オブジェクトのコレクションです。 |


Copyright © GrapeCity, Inc. All rights reserved.

次の **Preview for .NET プロパティ**は古い形式です。

| メンバ | 説明 |
|---|---|
| C1PrintDocument.ErrorDesc プロパティ | 古い形式。現在は、Warnings コレクションの最初の要素の説明を取得します。警告がない場合は、空文字列を返します。 |
| C1PrintDocument.ErrorCode プロパティ | 古い形式。現在は、Warnings コレクションの最初の要素のコードを取得します。警告がない場合は、GenerateErrorEnum.None を返します。 |

# 3.0J（C1Report）

このセクションでは、True WinReports for .NET 3.0J（C1Report）で追加された新しい機能について説明します。

# 新機能

True WinReports for .NET 3.0J（C1Report）では、次の新しい機能が追加されました。

- パラメータのカスタマイズを可能にするよう**InitializeParametersDialog** イベントが改良されました。
- **DialogEventArgs** クラスに **Parameters** および **ShowDialog** フィールドが追加されました。
- GroupPage および GroupPages のスクリプト関数が追加されました。
- サブレポートで検出されたエラーによってメインレポートの方で **ReportError** イベントが発生されるように改良されました。
- **C1PrintDocument** ストリームまたはファイルにレポートが出力されるよう新エクスポートフィルタが追加されました。
- 印刷サービスを向上させるよう**UsePrinterResolution** プロパティが追加されました。
- ドキュメントのロードが最適化され、特に .NET 2.0 でレポートのロード時間が短縮されました。
- チェック機能およびカラー機能のためのスクリプト構文が C1Report デザイナアプリケーションに追加されました。
- Count および CountDistinct の集計関数が追加されました。
- クエリーに List 型と Time 型の2つの新しいパラメータタイプのサポートが追加されました。
- 内部レコードセットクラスに IListSource の実装が追加されました。
- データ関連のエラーメッセージが強化されました。
- HTML、RTF、およびテキストエクスポートフィルタが強化されました。

## パラメータのカスタマイゼーション

**InitializeParametersDialog** イベントを使用してパラメータのカスタマイゼーションが可能になりました。

たとえば、以下のコードは **InitializeParametersDialog** を使用してレポートのパラメータの値を変更し、パラメータダイアログの表示を抑えます。

**Visual Basic コードの書き方**

Visual Basic

```
Private Sub c1Report1_InitializeParametersDialog(sender As Object, e As
C1.Win.C1Report.DialogEventArgs)
    ' パラメータ値を変更する
     Dim p As ReportParameterCollection = e.Parameters
     p("TheString").Value = "east"
     p("TheNumber").Value = 12
     p("TheDate").Value = DateTime.Now
```


Copyright © GrapeCity, Inc. All rights reserved.

```
    p("TheBool").Value = True

    ' ダイアログを表示しない
    e.ShowDialog = False
End Sub
```

**C# コードの書き方**

C#

```
private void c1Report1_InitializeParametersDialog(object sender,
C1.Win.C1Report.DialogEventArgs e)
{
    // パラメータ値を変更する
    C1.Win.C1Report.ReportParameterCollection p = e.Parameters;
    p["TheString"].Value = "east";
    p["TheNumber"].Value = 25;
    p["TheDate"].Value   = DateTime.Today;
    p["TheBool"].Value   = true;
    // ダイアログを表示しない
    e.ShowDialog = false;
}
```

## スクリプト関数

GroupPage および GroupPages スクリプト関数によってグループ内の現在のページ番号とグループ内の全ページ数の報告が可能になりました（例えば、"グループ US の売り上げ（ページ2/4）"）。以前のバージョンでは、そうするのにレポートを二回出力する必要がありました。最初はグループごとのページ数を計算するためと二回目はレポートを出力するためでした。

## エクスポートフィルタ

C1dFilter クラスによって連載された**C1PrintDocument** を含む .c1d ファイルの作成またはそんな文書を含むストリームの作成が可能です。**C1PrintDocument** に関するより詳しい情報については C1Preview コンポーネントのヘルプを参照してください。

C1dFilter を実行する際には C1Preview のアセンブリが必要ありません。ただし、後ほどC1PrintDocumentをプレビュー／印刷するためには C1Preview が必要になります。

以下のコードはレポートをC:\myReport.c1d に出力します。この場合は、C1.C1Preview.2 および C1.Win.C1Preview.2 アセンブリは必要ありません。

**Visual Basic コードの書き方**

Visual Basic

```
Imports C1.Win.C1Report
...
' レポートを読み込む
Dim c1r As New C1Report()
c1r.Load(fileName, reportName)
...
' C1Dフィルタを生成してファイルに連結する
Dim filter as C1dFilter
```


Copyright © GrapeCity, Inc. All rights reserved.

```
filter = New C1dFilter("C:\myreport.c1d.")
' .c1dファイルに使用するフォントを埋め込む
filter.EmbedFonts = True
' フィルタにレポートを出力する
c1r.RenderToFilter(filter)
```

**C# コードの書き方**

C#

```
using C1.Win.C1Report;
...
// レポートを読み込む
C1Report c1r = new C1Report();
c1r.Load(fileName, reportName);
...
// C1Dフィルタを生成してファイルに連結する
C1dFilter filter = new C1dFilter(@"C:\myReport.c1d");
// .c1dファイルに使用するフォントを埋め込む
filter.EmbedFonts = true;
// フィルタにレポートを出力する
c1r.RenderToFilter(filter);
```

以下のコードはレポートを c1d ストリームに出力して、そのストリームからC1PrintDocumentを作成して C1PrintPreviewDialogで表示します。この場合は、C1.C1Preview.2 および C1.Win.C1Preview.2 アセンブリが必要になります。

**Visual Basic コードの書き方**

Visual Basic

```
Imports C1.Win.C1Report
Imports C1.C1Preview
Imports C1.Win.C1Preview
...
' C1Dフィルタを生成してメモリストリームに連結する
Dim stream As New MemoryStream()
Dim filter As New C1dFilter(stream)
' フィルタにレポートを出力する
c1r.RenderToFilter(filter)
...
' ストリームからC1PrintDocumentを生成する
Dim docC1 AsC1PrintDocument= C1PrintDocument.FromStream(stream)
' C1PrintPreviewDialogでプレビューする
Dim dlg As New C1PrintPreviewDialog()
dlg.Document = docC1
dlg.ShowDialog()
```

**C# コードの書き方**

C#

```
using C1.Win.C1Report;
using C1.C1Preview;
using C1.Win.C1Preview;
```


Copyright © GrapeCity, Inc. All rights reserved.

```
...
// C1Dフィルタを生成してメモリストリームに連結する
MemoryStream stream = new MemoryStream();
C1dFilter filter = new C1dFilter(stream);
// フィルタにレポートを出力する
c1r.RenderToFilter(filter);
...
// ストリームからC1PrintDocumentを生成する
C1PrintDocument docC1 = C1PrintDocument.FromStream(stream);
// C1PrintPreviewDialogでプレビューする
C1PrintPreviewDialog dlg = new C1PrintPreviewDialog();
dlg.Document = docC1;
dlg.ShowDialog();
```

## 印刷サービスの機能強化

印刷サービスの改良を目的に **UsePrinterResolution** プロパティが追加されました。デフォルトでは C1Report が .NET 印刷サポートサービスを使用してレイアウト計算するために、プリンタの高解像度グラフィックオブジェクトを取得します。この場合、レポートは高画質で出力されますが環境によっては非常に時間がかかってしまうこともあります。このプロパティを False に設定すると、レポートはより早く出力されますが画質が悪化することもあります。

また、アプリケーションによっては.NET 印刷サポートルーチンが使用可能でないことがあります。その時これらのサービスを使用または参照すると問題が発生してしまいます。これは .NET フレームワークの問題です。詳細については、Microsoft 社サポート情報の「文書番号:324565」を参照してください。

## 2つの集計関数を追加

Count() は、null 以外の の数を返します。

CountDistinct() は、重複をカウントしないで null 以外の の数を返します。

## クエリパラメータタイプの追加

クエリーに2つの新しいパラメータタイプのサポートが追加されました。

1. List 型パラメータ

List 型パラメータ

```
PARAMETERS EmployeeList list "1,2,3";
select * from employees where employeeid in EmployeeList;
```

2. Time 型パラメータ

Time 型パラメータ

```
PARAMETERS StartTime time "09:45";
select * from employees where shiftStart > StartTime;
```

## IListSource のサポート

C1Report に DataTable または DataView が割り当てられると、**IC1ReportRecordset** インタフェースを実装するラッパークラスが自動的に作成されます。このラッパークラスは内部クラスであり、**IC1ReportRecordset** インタフェースだけを公開します。このため、基底のデータオブジェクトを取得する方法がありませんでしたが、IListSource がこれを可能にしました。サンプルについては、**Recordset** プロパティを参照してください。


Copyright © GrapeCity, Inc. All rights reserved.

## エラーメッセージの強化

OleDbException が生成された場合、コンポーネントは、実際に使用されている **ConnectionString** 値と **RecordSource** 値を示します。スクリプトコードまたはパラメータの置換によって変更されるため、これらは元のプロパティの値とは異なる場合があります。

## その他の新機能

RTF エクスポートで画像をサポートする機能が強化されました。クリップボードに依存したり、ASP.NET の下で実行されることはなくなりました。また、スレッドの安全性が強化され、**Barcode** のフィールドでは **TextDirection** が使用されます。さらに、カスタムフィールドエディタに **ICustomTypeDescriptor** のサポートが追加されました。

# クラスメンバ

True WinReports for .NET 3.0J（C1Report）で追加された新しいメンバを下記に示します。

- C1dFilter クラス
- C1dFilter コンストラクタ
- C1dFilter.EmbedFonts プロパティ
- C1dFilter.AddOutlineEntry メソッド
- C1dFilter.Dispose メソッド
- C1dFilter.EndReport メソッド
- C1dFilter.EndSection メソッド
- C1dFilter.IsPaged メソッド
- C1dFilter.NewPage メソッド
- C1dFilter.RenderField メソッド
- C1dFilter.StartReport メソッド
- C1dFilter.StartSection メソッド
- C1dFilter.UsesImages メソッド
- C1Report.UsePrinterResolution プロパティ
- **Parameters** フィールド
- **ShowDialog** フィールド
- C1PrintDocument メンバ（**FileFormatEnum 列挙体**）
- **Layout.OverlayReplacements** プロパティ
- **ReportParameter** クラス
- ReportParameter.Name プロパティ
- ReportParameter.Value プロパティ
- **ReportParameterCollection** クラス
- **ReportParameterCollection.Count** プロパティ
- **ReportParameterCollection.Item** プロパティ
- **XlsFilter.FreezePageHeader** プロパティ
- **C1Report.ReportInfo** プロパティ
- **C1Report.ParseExpression** メソッド
- **C1Report.ParseStatement** メソッド
- **ReportInfo** クラス
- **ReportInfo.Author** プロパティ
- **ReportInfo.Creator** プロパティ
- **ReportInfo.Keywords** プロパティ
- **ReportInfo.Subject** プロパティ
- **ReportInfo.Title** プロパティ
- **ReportInfo.Clear** メソッド
- **ReportInfo.Clone** メソッド
- ReportInfo.ToString メソッド


Copyright © GrapeCity, Inc. All rights reserved.

- **RtfFilter.RtfTarget** プロパティ
- **C1Report.GrowShrinkMode** プロパティ
- **Field.PictureScale** プロパティ
- **Layout.PictureScale** プロパティ
- **HtmlFilter.MultiFile** プロパティ
- Ean8 列挙型設定（**BarCodeEnum 列挙体**）
- Ean13 列挙型設定（**BarCodeEnum 列挙体**）
- RtfLabels 列挙型設定（**FileFormatEnum 列挙体**）

# 移行情報

このトピックでは、C1Reportコントロールとレポート定義ファイルの移行方法について説明します。

# ActiveX製品からの移行

このトピックでは、ActiveX製品（VS-VIEW Reporting Edition）からReports for WinFormsへの移行方法について説明します。

## レポートデザイナの場合

レポートデザイナでレポートを作成している場合は、本製品のレポートデザイナやC1Reportコントロールで、ActiveX製品で作成したレポート定義ファイル（*.xml）をそのまま読み込むことができます。

## プログラムコードの場合

C1Reportコントロールは、ActiveX製品（VS-VIEW）に収録されているVSReportコントロールとは互換性がなく、移行ツールも用意されていません。そのため、プロジェクトとプログラムコードを新規に作成する必要があります。ただし、C1Reportコントロールのオブジェクト構造はVSReportコントロールと似ているため、ActiveX製品を理解していれば、本製品での開発を比較的容易に行うことができます。ActiveX製品と本製品との違いについては、ActiveX製品とのメンバ比較表を参照してください。

# ActiveX製品とのメンバ比較表

## コントロール／オブジェクト

| VS-VIEW | Reports for WinForms | 機能の説明／備考 |
|---|---|---|
| VSReport | C1Report | C1Report コンポーネントは、Access スタイルの水平区切りデータベースレポートを生成します。生成されたレポートは、直接プリンタにレンダリングしたり、さまざまな移植可能な形式にエクスポートしたり、プレビューコントロールにレンダリングすることができます。 |
| Section | Section | レポート領域を表します。 |
| Field | Field | レポート内の個別のデータ項目を表示します。 |
| Group | Group | レポートデータをソートおよびグループ化するために使用されるオブジェクト。 |
| SectionCollection | SectionCollection | Section オブジェクトのコレクション。 |
| FieldCollection | FieldCollection | FieldCollection クラスは、Field オブジェクトのコレクションを表します。 |
| GroupCollection | GroupCollection | GroupCollection クラスは、Group オブジェクトのコレクションを表します。 |


Copyright © GrapeCity, Inc. All rights reserved.

| | | |
|---|---|---|
| DataSource | DataSource | レポートのデータを提供します。 |
| Layout | Layout | ページにレポートをレンダリングする方法を制御します。 |

## VSView コントロールのプロパティ

| VS-VIEW | Reports for WinForms | 機能の説明／備考 |
|---|---|---|
| Cancel | Cancel | 現在のレポートの生成をキャンセルするかどうかを設定または取得します。 |
| DataSource | DataSource | 現在のレポートの DataSource オブジェクトを参照します。 |
| DoEvents | DoEvents | レポートのレンダリング中に、コントロールが Windows のメッセージを処理するかどうかを設定または取得します。 |
| Fields | Fields | レポート定義内の Field オブジェクトのコレクションを返します。 |
| Font | Font | 新しいフィールドのデフォルトフォントを設定または取得します。 |
| Groups | Groups | レポート内のすべてのグループを含むコレクションを返します。 |
| IsBusy | IsBusy | レポートが現在生成中であるかどうかを判定する値を返します。 |
| Layout | Layout | レポートの Layout オブジェクトへの参照を返します。 |
| MaxPages | MaxPages | 1つのレポートに許可される最大ページ数を設定または取得します。 |
| OnClose | OnClose | OnClose イベントのための VBScript ハンドラを設定します。 |
| OnError | OnError | OnError イベントのための VBScript ハンドラを設定します。 |
| OnNoData | OnNoData | OnNoData イベントのための VBScript ハンドラを設定します。 |
| OnOpen | OnOpen | OnOpen イベントのための VBScript ハンドラを設定します。 |
| OnPage | OnPage | OnPage イベントのための VBScript ハンドラを設定します。 |
| Page | Page | 現在のページ番号を設定または取得します。 |
| ReportName | ReportName | レポートの名前を設定または取得します。 |
| Sections | Sections | 現在のレポートの Sections コレクションを参照します。 |
| Tag | Tag | 呼び出し側のプログラムで必要な追加のデータを格納します。 |

## VSView コントロールのメソッド

| VS-VIEW | Reports for WinForms | 機能の説明／備考 |
|---|---|---|
| Clear | Clear | 現在のレポートをクリアします。 |
| Copy | CopyFrom | 既存のレポートオブジェクトから、すべての情報をコピーします。 |
| Evaluate | Evaluate | 現在のレポートコンテキストで VBScript 式を評価します。 |
| Execute | Execute | 現在のレポートコンテキストで VBScript 文を実行します。 |
| GetReportInfo | GetReportInfo | レポート定義ファイルから情報を返します。 |
| Load | Load | XML レポート定義ファイルからレポートをロードします。 |
| Render | Render | VSPrinter コントロール内にレポートを出力します。 |


Copyright © GrapeCity, Inc. All rights reserved.

| | | |
|---|---|---|
| | | 注: Reports for WindowsForms でレポートを画像の内部コレクションにレンダリングします。 |
| RenderToFile | RenderToFile | レポートをファイルにレンダリングします。 |

## VSView コントロールのイベント

| VS-VIEW | Reports for WinForms | 機能の説明／備考 |
|---|---|---|
| OnClose | EndReport | コントロールがレポートのレンダリングを終了すると発生します。 |
| OnError | ReportError | レポートのレンダリング中にエラーがあると発生します。 |
| OnFormat | StartSection | 各 Section が書式設定される前に発生します。 |
| OnNoData | NoData | コントロールによるレポートの生成が開始された際、ソースレコードセットが空の時に発生します。 |
| OnOpen | StartReport | コントロールによるレポートの生成が開始され、ソースレコードセットを開く前に発生します。 |
| OnPage | EndPage | コントロールが各ページのレンダリングを終了すると発生します。 |
| OnPrint | PrintSection | 各 Section が印刷される前に発生します（セクションを書式設定した後）。 |

## Section オブジェクトのプロパティ

| VS-VIEW | Reports for WinForms | 機能の説明／備考 |
|---|---|---|
| BackColor | BackColor | Section の背景色を設定または取得します。 |
| CanGrow | CanGrow | セクションの高さが、そのコンテンツに合わせて自動的に大きくなるかどうかを設定または取得します。 |
| CanShrink | CanShrink | 1つ以上のフィールドが縮小された場合に、Section の高さも縮小されるかどうかを設定または取得します。 |
| Fields | Fields | この Section 内のすべてのフィールドのコレクションを返します。 |
| ForcePageBreak | ForcePageBreak | セクションがレンダリングされる前後にページ区切りが挿入されるかどうかを設定または取得します。 |
| Height | Height | セクションの高さを設定します。 |
| KeepTogether | KeepTogether | Section が1ページに収められるかどうかを設定または取得します。 |
| Name | Name | Section 名を設定または取得します。 |
| OnFormat | OnFormat | Section が書式設定される前に実行される VBScript コードを含む文字列を設定または取得します。 |
| OnPrint | OnPrint | Sectionが印刷される前（フォーマット後）に実行される VBScript コードを含む文字列を設定します。 |
| RenderHeight | RenderHeight | CanShrink プロパティがセクションに適用された後のセクションの高さを返します。 |
| Repeat | Repeat | Section を各ページの先頭に繰り返し表示するかどうかを設定または取得します。 |
| Tag | Tag | 呼び出し側のプログラムで使用される追加のデータを設定または取得します。 |


Copyright © GrapeCity, Inc. All rights reserved.

| | | |
|---|---|---|
| Type | Type | Section のタイプ（PageHeader、Detail など）を返します。 |
| Visible | Visible | Section がレポートにレンダリングされるかどうかを設定または取得します。 |

## Field オブジェクトのプロパティ

| VS-VIEW | Reports for WinForms | 機能の説明／備考 |
|---|---|---|
| Align | Align | フィールドの配置を設定します。 |
| Anchor | Anchor | フィールドを含む Section が CanGrow/CanShrink プロパティによりサイズ変更されると、フィールドの位置および寸法を取得または設定します。 |
| BackColor | BackColor | フィールドの背景色を取得または設定します。 |
| BackStyle | 該当なし | フィールドの背景を透明にするか不透明にするかを設定または取得します。<br>注: Backcolor プロパティを Transparent に設定できます。 |
| BorderColor | BorderColor | フィールドの境界線の色を取得または設定します。 |
| BorderStyle | BorderStyle | フィールドの境界線のスタイルを取得または設定します。 |
| Calculated | Calculated | Text プロパティをリテラル値と計算式のどちらとして解釈するかを指定します。 |
| CanGrow | CanGrow | フィールドの高さがフィールドのコンテンツに合わせて自動的に大きくなるかどうかを指定します。 |
| CanShrink | CanShrink | フィールドの高さがフィールドのコンテンツに合わせて自動的に小さくなるかどうかを指定します。 |
| Checkbox | Checkbox | フィールドがチェックボックスとして表示されるかどうかを指定します。 |
| Font | Font | フィールドのフォントを取得または設定します。 |
| ForcePageBreak | ForcePageBreak | フィールドの前後にページ区切りが挿入されるかどうかを指定します。 |
| ForeColor | ForeColor | フィールドの前景色を取得または設定します。 |
| Format | Format | フィールド値の書式設定に使用される文字列を取得または設定します。 |
| Height | Height | フィールドの高さを取得または設定します。 |
| HideDuplicates | HideDuplicates | 連続するレコードで、重複する値を非表示にするかどうかを指定します。 |
| Left | Left | フィールドのサイズ・位置を取得または設定します。 |
| LineSlant | LineSlant | フィールドを横切る線（フィールドを囲むボックスではなく）がレンダリングされるかどうかを指定します。 |
| LineSpacing | LineSpacing | フィールドの配置を設定します。<br>注: 設定の単位が異なります。 |
| LineWidth | LineWidth | フィールドの境界線または線の幅を取得または設定します。 |
| LinkTarget | LinkTarget | フィールドがクリックされたときにアクセスされる URL に評価される式を取得または設定します。 |
| MarginBottom | MarginBottom | フィールドの下端とコンテンツの間隔を取得または設定します。 |
| MarginLeft | MarginLeft | フィールドの左端とコンテンツの間隔を取得または設定します。 |
| MarginRight | MarginRight | フィールドの右端とコンテンツの間隔を取得または設定します。 |
| MarginTop | MarginTop | フィールドの上端とコンテンツの間隔を取得または設定します。 |


Copyright © GrapeCity, Inc. All rights reserved.

| | | |
|---|---|---|
| Name | Name | フィールド名を取得または設定します。 |
| Picture | Picture | フィールドに表示する画像を取得または設定します(ファイル名を設定)。 |
| PictureAlign | PictureAlign | フィールド画像が配置される方法およびスケーリングを取得または設定します (Picture プロパティを参照) 。 |
| RenderHeight | RenderHeight | CanGrow/CanShrink フィールドが収まるようにセクションを調整した後の、フィールドの高さを取得します。 |
| RenderTop | RenderTop | CanGrow/CanShrink フィールドが収まるようにセクションを調整した後の、フィールドの上端の位置を取得します。 |
| RTF | RTF | { } で囲まれた文字列を RTF としてレンダリングするかどうかを指定します。 |
| RunningSum | RunningSum | フィールドの積算合計を表示するかどうかを指定します。 |
| Section | Section | フィールドが属するセクションを取得または設定します。 |
| Subreport | Subreport | フィールド内にレンダリングされる別のレポート(サブレポート)への参照を取得または設定します。 |
| Tag | Tag | 呼び出し側のプログラムで必要な追加のデータを取得または設定します。 |
| Text | Text | フィールドのリテラルテキストおよび計算式を取得または設定します(Calculated プロパティの値に依存します)。 |
| Top | Top | フィールドの上端の位置を参照します。 |
| Value | Value | フィールドの計算値を取得または設定します。 |
| Visible | Visible | Field がレポートにレンダリングされるかどうかを指定します。 |
| Width | Width | フィールドの幅を取得または設定します。 |
| WordWrap | WordWrap | フィールドのコンテンツがフィールド四角形内で折り返されるかどうかを指定します。 |
| ZOrder | ZOrder | フィールドの ZOrder を取得または設定します。 |

## Field オブジェクトのメソッド

| VS-VIEW | Reports for WinForms | 機能の説明／備考 |
|---|---|---|
| SetZOrder | SetZOrder | フィールドを ZOrder の最前面または最背面に移動します。 |

## Group オブジェクトのプロパティ

| VS-VIEW | Reports for WinForms | 機能の説明／備考 |
|---|---|---|
| GroupBy | GroupBy | レコードのグループ化に使用される式を設定または取得します。 |
| KeepTogether | KeepTogether | グループが1ページに収められるかどうかを設定または取得します。 |
| Name | Name | グループ名を設定または取得します。 |
| SectionFooter | SectionFooter | このグループのフッターとして使用される Section を返します。 |
| SectionHeader | SectionHeader | このグループのヘッダーとして使用される Section を返します |
| Sort | Sort | グループのソート方法を設定または取得します。 |
| Tag | Tag | 呼び出し側のプログラムで必要な追加のデータを設定または取得します。 |


Copyright © GrapeCity, Inc. All rights reserved.

## Group オブジェクトのメソッド

| VS-VIEW | Reports for WinForms | 機能の説明／備考 |
|---|---|---|
| Move | Move | グループを Fields も調整できます。 |

## SectionCollection オブジェクトのプロパティ

| VS-VIEW | Reports for WinForms | 機能の説明／備考 |
|---|---|---|
| Count | Count | コレクション内の要素の数を返します。 |
| Item | Item | コレクション内の特定のオブジェクトを参照します。 |

## FieldCollection オブジェクトのプロパティ

| VS-VIEW | Reports for WinForms | 機能の説明／備考 |
|---|---|---|
| Count | Count | Fields コレクション内のフィールドの数を返します。 |
| Item | Item | コレクション内の特定のオブジェクトを参照します。 |

## FieldCollection オブジェクトのメソッド

| VS-VIEW | Reports for WinForms | 機能の説明／備考 |
|---|---|---|
| Add | Add | Fields コレクションにフィールドを追加します。 |
| Remove | Remove | Fields コレクションからフィールドを削除します。 |

## GroupCollection オブジェクトのプロパティ

| VS-VIEW | Reports for WinForms | 機能の説明／備考 |
|---|---|---|
| Count | Count | Groups コレクション内のグループの数を返します。 |
| Item | Item | コレクション内の特定のオブジェクトを参照します。 |

## GroupCollection オブジェクトのメソッド

| VS-VIEW | Reports for WinForms | 機能の説明／備考 |
|---|---|---|
| Add | Add | コレクションに新しい Group を追加します。新しいグループのヘッダーおよびフッターセクションはSections コレクションに追加されます。 |
| Remove | Remove | C1Report.Groups コレクションから Group を削除し、C1Report.Sections コレクションからそのヘッダーセクションとフッターセクションも削除します。 |

## DataSourceオブジェクトのプロパティ


Copyright © GrapeCity, Inc. All rights reserved.

| VS-VIEW | Reports for WinForms | 機能の説明／備考 |
|---|---|---|
| ConnectionString | ConnectionString | レポートのデータソースを指定する文字列を取得または設定します。 |
| FieldInfo | 該当なし | すべてのフィールド名、タイプ、カレントレポートデータソースが含まれるバリアント型の配列を参照します。 |
| Filter | Filter | レポートに含めるデータ行をフィルタするために使用される式を取得または設定します。 |
| MaxRecords | MaxRecords | レポートに入れるデータ行の最大数を取得または設定します。 |
| Recordset | Recordset | レポートにデータを提供するデータソースオブジェクトを設定または取得します。 |
| RecordSource | RecordSource | レポートにソースレコードセットを指定する文字列を設定します。 |

## DataSource オブジェクトのメソッド

| VS-VIEW | Reports for WinForms | 機能の説明／備考 |
|---|---|---|
| GetRecordSource | GetRecordSource | RecordSource 文字列内の PARAMETERS 句を処理し、その結果を返します。 |

## Layout オブジェクトのプロパティ

| VS-VIEW | Reports for WinForms | 機能の説明／備考 |
|---|---|---|
| ColumnLayout | ColumnLayout | レポートにソースレコードセットを指定する文字列を設定します。 |
| Columns | Columns | 詳細行の列の数を設定します。 |
| MarginBottom | MarginBottom | 用紙の下マージンを設定します。 |
| MarginLeft | MarginLeft | 用紙の左マージンを設定します。 |
| MarginRight | MarginRight | 用紙の右マージンを設定します。 |
| MarginTop | MarginTop | 用紙の上マージンを設定します。 |
| Orientation | Orientation | 用紙の方向を設定または取得します。 |
| PageFooter | PageFooter | どのページにページフッターセクションを表示するかを決定する値を設定または取得します。 |
| PageHeader | PageHeader | どのページにページヘッダーセクションを表示するかを決定する値を設定または取得します。 |
| PaperSize | PaperSize | レポートの用紙サイズを設定または取得します。 |
| Picture | Picture | レポート本体の背景画像を設定または取得します。 |
| PictureAlign | PictureAlign | 背景画像の配置方法を設定または取得します。 |
| PictureShow | PictureShow | 背景画像が表示される場所を設定または取得します。 |
| Width | Width | レポートの詳細セクションの幅を設定または取得します（ツイップ単位）。 |


Copyright © GrapeCity, Inc. All rights reserved.

# 旧Windowsフォーム製品からの移行

このトピックでは、旧Windowsフォーム製品（VSVIEW for .NETおよびTrue WinReports for .NET）からReports for WinFormsへの移行方法について説明します。

## レポートデザイナの場合

レポートデザイナでレポートを作成している場合は、本製品のレポートデザイナやC1Reportコントロールで、旧Windowsフォーム製品（VSVIEW for .NETおよびTrue WinReports for .NET）で作成したレポート定義ファイル（*.xml）をそのまま読み込むことができます。

## プログラムコードの場合

旧Windowsフォーム製品（VSVIEW for .NETおよびTrue WinReports for .NET）に収録されているC1Reportコントロールは、本製品のC1Reportコントロールと上位互換性が保たれています。ただし、アセンブリ名と名前空間が変更されており、「C1Report移行ツール」（Convert2Report.2.exe）を使用することでプロジェクトを移行することができます。


Copyright © GrapeCity, Inc. All rights reserved.

# 主な特長

**Reports for WinForms** を使用して、カスタマイズしたレビューやレポートを作成します。**C1Report** および **C1PrintDocument** コンポーネントのさまざまな機能や、6つのビジュアルプレビューコントロールを活用できます。

## C1Report の機能

**C1Report** コントロールは、以下の機能を備えています。

- **レポートデザイナアプリケーション**
  コードを記述することなく、レポート定義ファイルをすばやく作成、編集、プレビュー、ロード、および保存します。
- **レポートウィザード**
  レポートデザイナのウィザードに示された手順に従って、新しいレポートを簡単に作成できます。**C1Report ウィザード**を使用してレポートを作成するのに、専門知識は必要ありません。最初から最後までを5つの簡単な手順で苦もなく新しいレポートを作成できます。**C1Report ウィザード**のガイドに従って各手順を実行して、レポートのデータソース、レポートフィールド、レイアウトを選択できます。
- **バンドレポートモデル**
  レポートはグループ、セクション、およびフィールドに基づくバンドレポートモデルを使用します。バンドレポートモデルにより、高度に構成されたレポートレイアウトが可能です。
- **30 種類以上の組み込みレポートテンプレート**
  強化されたレポートデザイナアプリケーションは現在、34 種類のレポートテンプレートを備えています。[**C1Report ウィザード**]でレポートテーマを選択するだけで、プロフェッショナルなスタイルのレポートが得られます。コーディングは不要です。色彩豊かなレポートが1回のクリックだけで完成です。
- **Microsoft Access と Crystal Reports の互換性**
  Reports は、Microsoft Access と Crystal Reports に見られる機能をサポートします。ボタンのクリック1つで、**C1ReportDesigner** を使用して Access レポートファイル（MDB）と Crystal レポートファイル（RPT）をインポートします。
- **柔軟なデータバインディング**
  レポート定義で接続文字列と SQL ステートメントを指定すると、Reports はデータを自動的にロードします。オプションで、XML ファイル、カスタムコレクション、およびその他のデータソースを使用します。
- **パラメータによるデータの追加／制限**
  Reports は、パラメータ化されたクエリーを含むことができるため、ユーザーは、レンダリング前に、レポートに含める必要があるデータを追加／制限することで、レポートをカスタマイズできます。レポートフィールドの値の指定、データのフィルタリング、ソートとグループ化の制御など、多くの操作が可能です。レポートのパラメータを使用し、必要なデータのみを表示します。
- **複数のレポートを1つに結合**
  Reports は任意のレベルにネストされたレポート（サブレポート）を含むことができます。メインレポートを使用して詳細情報を表示し、サブレポートを使用して各グループの先頭に概要データを表示できます。
- **VBScript 式**
  Reports は埋め込みの VBScript イベントハンドラを組み込んで、それらを内蔵型にできます。レポートのレンダリング時に、値に応じたフィールドの書式設定、ページ数の更新、データがないセクションの非表示化などを行います。
- **書式設定、グループ化、フィルタリング、ソートなどの制御**
  集計式（合計、最大、平均など）用の式を含む VBScript 式を使用して、レポートのコンテンツを取得、計算、表示、グループ化、フィルタリング、ソート、パラメータ化、および書式設定を行います。
- **グラフフィールド**
  グラフをレポートに埋め込んで、数値データをグラフで表示します。Report の[**グラフ**]フィールドは、**C1Chart コントロール**を使用して実装され、複数のデータ系列を表示できます。サポートされたグラフタイプには、横棒、面、散布図、円、折れ線、および縦棒が含まれます。
- **集計グラフ機能**
  希望の aggregate 関数を使用し、同じカテゴリ（ValueX）を持つデータ値（ValueY）を自動的に集計するグラフを作成します。グラフフィールドの Aggregateプロパティは、同じカテゴリを持つ値をグラフ内の単一ポイントに集計する方法をグラフに通知します。これは、任意の共通集計関数（合計、平均、カウント、最大、最小、


Copyright © GrapeCity, Inc. All rights reserved.

標準偏差、および分散）をデータ上で実行するように設定できます。

- **エクスポート形式**
  レポートをプリンタまたはプレビューコントロールに直接出力するか、レポートをさまざまな移植可能な形式（Excel（XLS、XLSX）、PDF、HTML、Rich Text、および圧縮メタファイル）にエクスポートします。
- **自動レポート作成**
  **C1ReportsScheduler** を使用してレポートの作成を自動化できます。**C1ReportsScheduler**は、レポートの作成がバックグラウンドで実行されるようにスケジュールするスタンドアロンアプリケーションです。**C1ReportsScheduler**アプリケーションを使用して、エクスポートまたは印刷するレポート、レポートのエクスポート先の形式、およびレポート作成のスケジュールや頻度を選択できます。

## C1PrintDocument の機能

C1PrintDocument コントロールは、以下の機能を備えています。

- **強力なドキュメント指向オブジェクトモデル**
  **C1PrintDocument** コンポーネントは、強力な自動レイアウト、書式設定、およびページネーションコントロール機能を持つ、柔軟な階層ドキュメントオブジェクトモデルを提供します。このため、手動によるレイアウトの計算、改ページの挿入などを行う必要はありません。

- **豊富な書式設定オプション**
  複数のフォント、テキストの色と背景色、テキストの位置合わせ（下付き、上付き）、インライン画像、各種テキスト配置（テキストの両端揃えを含む）などを含むテキストの段落のサポートによりドキュメントの外観を制御します。

- **強力な Excel スタイルのテーブル**
  テーブル要素をグループ化し、グループにスタイルを適用し、行と列のヘッダーなどを変更します。テーブルは、論理的に無限の列数と行数を持つ Excel スタイルのオブジェクトモデルをサポートします。単にテーブル要素にアクセスし、それを初期化します。このため、現在のテーブルサイズの指定を気にする必要はありません。

- **ドキュメント要素の強力なサイズ変更と配置**
  ドキュメント要素のサイズと位置は、絶対値、他の要素のサイズと位置に対する相対値、または絶対値と相対値を組み合わせた単純な式として指定することができます。たとえば、親要素または現在のページ幅の比率として要素の幅を指定します。

- **TOC と用語索引の自動生成**
  **C1PrintDocument** は、目次（TOC）とアルファベット順の用語索引の自動生成をサポートしています。出力形式に応じて、TOC と索引の両方のリンクがクリック可能で、ユーザーは参照ページに移動できます。

- **ハイパーリンクの追加**
  ハイパーリンク、リンク先などを追加して、ドキュメントをインタラクティブにします。どのドキュメント要素も、ハイパーリンク、またはハイパーリンクのジャンプ先を指定できます。

- **データバインディングサポート**
  ドキュメントは、完全にコードで作成するか、または強力かつ柔軟なデータバインディングモデルを使用してデータベースにバインドできます。

- **レポート定義のインポート**
  **C1PrintDocument** を、レポート定義の作成、カスタマイズ、ロード、および保存のための機能豊富なオブジェクトモデルを公開する強力な **C1Report** コンポーネントと組み合わせます。**C1PrintDocument** コンポーネントでレポート定義を素早くインポートおよび生成できます。

- **SQL Server Reporting Services**
  レポート定義言語（RDL）は、SQL Server Reporting Services で一般に使用されるレポート作成スキームです。**C1PrintDocument** により、SSRS 定義ファイル（.rdl）をインポートできます。結果は、インポートされたレポートのデータバインドしたドキュメント表現となります。


Copyright © GrapeCity, Inc. All rights reserved.

- **エクスポート形式**
  複数のエクスポート形式オプションにより、ドキュメントの保存と共有が簡単になります。ドキュメントを、PDF（Adobe Portable Document Format）、Excel（XLS と XLSX）、Word（RTF と DOCX）、HTML、および複数の画像形式にエクスポートします。

- **Adobe Acroform の作成**
  ドキュメントは、インタラクティブな（エンドユーザーが入力する）フォームを含むことができます。テキストボックス、リストボックス、ドロップダウンリスト、チェック、ラジオ、およびプッシュボタンを **C1PrintDocument** に追加します。これらのコントロールは、**C1PrintPreviewControl**（「印刷プレビュー」を参照）内に表示されるとインタラクティブになります。また、これらのドキュメントを Adobe Acroform にもエクスポートできます

- **エクスポートした PDF 表示の制御**
  エクスポートした PDF ファイルを Adobe Acrobat で表示する方法を制御できます。たとえば、ページの表示方法（一度に1ページを表示、または2ページを列に表示するなど）、およびさまざまな要素の表示形式（サムネイル画像またはドキュメントのアウトラインビューの表示など）を設定します。

- **小さいファイルサイズ用の C1DX ファイル形式**
  **C1PrintDocument** オブジェクトの新しい OPC ベースのファイル形式である C1D OpenXML（C1DX）は、Microsoft Open Packaging Conventions に準拠し、Microsoft Office 2007 OpenXML 形式に似ています。組み込み圧縮によって、生成されるファイルのサイズが小さくなります。現在の Preview for WinForms C1D 形式も、下位互換用に完全にサポートされています。

- **複数ページのレイアウト**
  さまざまな用紙サイズ、ページ設定、列数、ページヘッダーなどに対応する複数のページレイアウトを事前定義し、単一のプロパティを設定して実行時に選択できます。

- **複数の大容量ドキュメントの結合**
  **C1MultiDocument** を使用し、複数の **C1PrintDocument** を結合すると、共有のページ番号、共通の TOC、用語索引、ページカウント、およびドキュメント間のハイパーリンクを持つ1つの完全な連続したドキュメントとして表示されます。これにより、メモリ制限のため単一の **C1PrintDocument** オブジェクトで処理できない大容量ドキュメントを作成およびエクスポートできます。

- **階層スタイル**
  階層スタイルは、アンビエントまたは非スタイル属性のインテリジェントなサポートにより、すべてのドキュメント要素の外観を制御します。個別フォント属性（太字やフォントサイズなど）、テーブルのグリッド線などを指定します。

- **GDI+ テキストの表示**
  1つのプロパティを設定し、GDI+ テキスト API を使用してテキストを表示できます。GDI+ テキスト表示したテキストは、Microsoft Office 2007 のテキストのように表示され、XPS でのデフォルトのテキストレイアウトに一致します。

- **True Type Font の埋め込み**
  ドキュメントのプレビューまたは印刷に使用するシステムに、使用するフォントがインストールされていない場合でも、システム上でテキストが正しく表示されるようにフォントを埋め込みます。

- **ディクショナリのサポート**
  リソース（ドキュメント全体に使用される画像など）をドキュメントのディクショナリに格納し、スペースと時間を節約します。

## 印刷プレビュー機能

**Reports for WinForms** の印刷プレビューコントロールの機能には、以下のものが含まれます。

- **フル機能のプレビューコントロール**
  統合された **C1PrintPreviewControl** と **C1PrintPreviewDialog**コントロールは、細かい設定をせずにそのまま使えるサムネイルとアウトラインビュー、テキスト検索と定義済みツールバーを含む、すぐ使える、フル機


Copyright © GrapeCity, Inc. All rights reserved.

能の UI を提供します。

- **レポートのプレビュー**
  **Reports** を **Reports for WinForms** に簡単に統合して、プレビュー、書式設定、印刷、およびエクスポート機能をレポートに追加できます。**C1PrintPreviewControl** または **C1PreviewPane** の Document を各自のレポーティングコントロールに設定するだけで完了です。
- **PrintDocument 機能**
  **C1PrintDocument コンポーネント**に加えて、プレビューコントロールは、標準 **.NET PrintDocument** コンポーネントをロードおよび表示し、外部形式（PDF など）にエクスポートすることもできます。アプリケーションを簡単に変換します。
- **サムネイル表示**
  **Reports** には、レンダリングされたドキュメント内のすべてのページに組み込みのサムネイル表示が用意されています。サムネイル表示により、任意のページへ素早くナビゲートできます。サムネイルはページの作成時にオンザフライで生成されるため、すべてのページが表示を完了していない場合にもサムネイルを即座に取得できます。サムネイル表示を自動的に表示するには、個別の**C1PreviewThumbnailView** コントロールを使用してそれを **C1PreviewPane** にアタッチするか、すべて内蔵した **C1PrintPreviewControl**を使用するだけです。
- **インタラクティブなドキュメントの再フロー**
  エンドユーザーは、実行時にドキュメントをインタラクティブに変更することができます（ページの余白または向きの変更など）。その結果、ドキュメントは変更に応じて自動的に再フローされます。
- **テキスト検索**
  実行時に追加のコーディングなしにテキスト検索を実行します。**C1PrintPreviewControl** には、組み込みのテキスト検索パネルが用意されています。検索結果は、ページ番号と見つかった位置へのリンクを含みます。
- **複数の組み込みツールバーセット**
  複数のプリセットツールバー画像コレクションから選択するか、独自のものを選択します。**C1PrintPreviewControl** には、Adobe、XP、Classic Windows、および Macオペレーティングシステムに対応するテーマが用意されています。
- **ズームツール**
  Reports は、Microsoft Word に見られる多種多様なズームオプションをサポートします。定義済みのビューには、実サイズ、ページ幅、テキスト幅、およびページ全体が含まれます。また、パーセントを使用して特定のズーム値を定義することもできます。**C1PrintPreviewControl**は、ズームインとズームアウトツールも備えており、ユーザーはページ上のズームイン／アウトする場所を指定できます。
- **C1PrintDocument 機能サポート**
  すべてのプレビューコントロールが、ハイパーリンクやアウトラインなどの **C1PrintDocument** コンポーネントの機能をサポートします。
- **コード不要の開発**
  Reports には、フローティングツールバーによる ComponentOne SmartDesigner® 技術を含む、広範なデザイン時サポートが含まれており、コードを記述することなく、プレビューウィンドウを簡単にカスタマイズできます。
- **柔軟なモジュール形式の設計**
  個別の専用コントロール（プレビューペイン、サムネイルとアウトラインビュー、およびテキスト検索パネル）を使用して、ドキュメントビューをカスタマイズできます。Reports コントロールを **Ribbon for WinForms** などの他の **ComponentOne for WinForms** コントロールと組み合わせて、任意の UI に合ったカスタムプレビューウィンドウを作成します。
- **ローカライズ**
  デザイン時に、異なるカルチャのすべてのエンドユーザーの視覚的文字列のローカライズしたバージョンを作成し、実行時に言語を切り替えることができます。

## C1ReportDesigner の機能

**C1ReportDesigner** アプリケーションは、以下の機能を備えています。

- **簡単にアクセス可能**
  **C1ReportDesigner** はスタンドアロンアプリケーションですが、Visual Studio 内からこれを簡単に起動してナビゲートできます。C1Report コンポーネントのスマートタグで、[**レポートの編集**]を選択する


Copyright © GrapeCity, Inc. All rights reserved.

と、**C1ReportDesigner** アプリケーションが開きます。

- **新しいレポートの作成**
  **C1Report ウィザード**]を使用し、新しいレポートを素早く簡単に作成します。レポートを作成するには、以下を実行するだけです。
  1. 新しいレポートのデータソースを選択します。
  2. レポートに含めるフィールドを選択します。
  3. 新しいレポートのレイアウト、スタイル、およびタイトルを設定します。
- **既存のレポートのインポート**
  既存のレポートを受け取って ComponentOne Reports に変える能力は、**C1ReportDesigner** の最も強力な機能の1つです。既存のレポート定義を Microsoft Access ファイル(.mdb と .adp)または Crystal Reports (.rpt)からインポートします。
- **レポートの設計と変更**
  Access スタイルの WYSIWYG デザイン面により、レポートの設計が簡単で直観的になります。レポートフィールドをツールバーからレポートのデザイン面にドラッグ &ドロップします。バンド領域は、ヘッダー、ボディ、およびフッターなどのレポートの各領域をマークします。すべてのフィールド関連のプロパティをアプリケーション本体で直接設定します。プロパティウィンドウからカスタムの VBScript コードを記述することもできます。
- **レポートのエクスポート**
  レポートをサポートされた任意のファイル形式(HTML、PDF、RTF、XLS、XLSX、TIF、TXT、または ZIP)に直接エクスポートします。
- **レポート定義の保存と配布**
  作成またはインポートしたレポートは、XML ベースのレポート定義ファイルに保存できます。**C1Report** コンポーネントによって定義ファイルを公開されるアプリケーションにパッケージし、レポートをエンドユーザーに配布します。**C1ReportDesigner** アプリケーション本体を配布またはカスタマイズするには、**Reports for .NET Designer Edition** を使用する必要があります。

## C1ReportsScheduler の機能

**C1ReportsScheduler** アプリケーションには、次の機能があります。

- **ComponentOne レポートの生成とエクスポート**
  **C1ReportsScheduler** は、**Reports for WinForms** に付属するスタンドアロンのスケジュールアプリケーションです。設定したスケジュールに基づいてバックグラウンドで ComponentOne レポートを生成およびエクスポートすることができます。
- **バックグラウンドでの実行とスケジュールの制御**
  C1ReportsScheduler アプリケーションは、フロントエンドと Windows サービスという相互にやり取りする2つの部分で構成されます。Windows サービスはバックグラウンドで実行され、指定されたタスクをスケジュールに基づいて実行します。フロントエンドは、タスクリストの表示または編集、スケジュールの開始または停止、およびサービスの制御を行います。フロントエンドは、サービスのインストールと設定には使用されますが、サービスの実行には必要ありません。
- **多様な形式へのエクスポート**
  **ComponentOne レポート**は、多様な形式(PDF、リッチテキスト、Open XML Word、Excel、HTML、メタファイル、画像など)にエクスポートできます。
- **4つの簡単な手順でレポート出力をスケジュール可能**
  レポート出力をスケジュールするには、次の手順を実行するだけです。
  1. マシンから任意の数の **C1Report** 定義ファイル(.xml)を選択します。
  2. **[タスクアクション]**ペインで、レポートごとに、エクスポートや印刷などのアクションをいくつでも選択します。
  3. レポートごとに、1回または定期的なスケジュールを設定します。
  4. **[開始]**をクリックして、バックグラウンドサービスを開始します。レポートは、スケジュールされた時間に指定された出力形式で生成されます。
- **完全なソースの提供**
  ビルド済みのアプリケーションとサービスを「そのまま」エンドユーザーに提供することも、ニーズに応じて UI や機能を変更することもできます。**C1ReportsScheduler** の完全なソースコードがサンプルとして提供されています。
- **C1ReportsScheduler Windows サービス**


Copyright © GrapeCity, Inc. All rights reserved.

フロントエンドアプリケーションは常に実行されているとは限らないため、スケジュールされた時間にレポートが生成されるように、**C1ReportsScheduler** Windows サービスが提供されています。このサービスは、フロントエンドアプリケーションを使用して、簡単にマシンにインストールしたり、マシンからアンインストールできます。**C1ReportsScheduler**を初めて実行する際に、このサービスをインストールするかどうかを尋ねるダイアログが表示されます。フロントエンドアプリケーションが常に実行されている場合、このサービスをインストールする必要はありません。

## C1MultiDocument の機能

**C1MultiDocument** コンポーネントは、以下の機能を備えています。

- **大容量ドキュメントの処理**
  **C1MultiDocument** は、それ以外の場合はメモリ制限のため作成／エクスポート／印刷できない可能性がある大容量ドキュメントを処理できます。
- **複数のドキュメントの結合**
  圧縮と一時ディスクストレージの使用により、**C1MultiDocument** は複数の **C1PrintDocument** オブジェクトを、全ページが単一の **C1PrintDocument** に属している場合はメモリ不足状態が発生する可能性がある1つの大容量のマルチドキュメントに結合できます。

## C1RdlReport の機能

**C1RdlReport** コンポーネントは、以下の機能を備えています。

- **完全な RDL オブジェクトモデルの公開**
  **C1RdlReport** コンポーネントは、最新の RDL 2008 specificationに基づく完全な RDL オブジェクトモデルを公開します。これにより、既存のレポートを変更するか、さらには、新しい RDL レポートをコードで完全に作成できます。これは Microsoft Reporting Services だけではできません。
- **任意のデータソースからの RDL レポートの生成**
  データソースとして SQL サーバーデータを使用するだけに制約されません。C1RdlReport は、Access データベースなどの任意のデータソースを使用して RDL レポートを生成できます。
- **外部依存関係が不要**
  **C1RdlReport** は、Microsoft Reporting Services サーバーの必要性などの外部依存関係を使用せずに、内蔵型の RDL レポートソリューションを提供します。
- **C1Reports とのシームレスな統合**
  **C1RdlReport** は、**ComponentOne Reports** スイート全体とのシームレスな統合を提供します。**C1RdlReport** を **C1PrintPreviewControl** と共に使用し、レポートのプレビュー、書式設定、印刷、エクスポート機能を提供します。
- **RDL オブジェクトおよびプロパティへのサポート**
  **C1RdlReport** は、サブレポート、パラメータ、ハイパーリンク、グラフ、形状、画像、テキストボックスなど、一般的な Microsoft レポート機能の大部分をサポートします。

## C1RdlReport の機能

C1ReportDesigner コントロールには、次の機能があります。

- **アプリケーションへのデザイナの緊密な統合**
  独自にカスタマイズしたレポートデザイナを記述することにより、デザイナアプリケーションを別個に実行するのではなく、アプリケーションにデザイナを緊密に統合することができます。エンドユーザーが使用できるデータソースや、レポートに追加できるフィールドの種類をカスタマイズできます。たとえば、アプリケーションによって定義されたカスタムデータソースオブジェクトを使用できます。
- **エンドユーザーへのカスタム株式レポート定義の提供**
  独自にカスタマイズしたレポートデザイナを使用すると、アプリケーションのスコープ内で意味を持つ株式レポート定義のメニューを提供できます。エンドユーザーは、このメニューを Microsoft Outlook の印刷オプションのように使用して、各株式レポートのさまざまな側面をカスタマイズできます。


Copyright © GrapeCity, Inc. All rights reserved.

- **実行時の無償配布**
  無制限の数のクライアントにアプリケーションを柔軟に展開できるようにするために、**C1ReportDesigner**アプリケーションを実行時に無償で配布できるようになりました。開発者は、エンドユーザー向けレポートデザイナを含む強力な統合レポートソリューションを簡単に作成、カスタマイズ、および展開し、それらをロイヤルティに縛られることなく配布することができます。
- **C1ReportDesigner アプリケーションの完全なソースコード**
  デザイナエディションには、**ComponentOne Reports for .WinForms Designer Edition** に含まれる **C1ReportDesigner** アプリケーションの完全なソースコードが付属しています。完全なソースコードを使用して、デザイナアプリケーションをカスタマイズしたり、それを独自のアプリケーションに統合することもできます。
- **使いやすい C1ReportDesigner コンポーネント**
  **C1ReportDesigner** コンポーネントは、レポートを設計モードで表示して、ユーザーがレポートのフィールドやセクションをドラッグ、コピー、サイズ変更できるようにします。このコンポーネントには、無制限の元に戻す／やり直しスタックに加えて、Microsoft Visual Studio .NET に付属する **PropertyGrid** コントロールと組み合わせて使用できるように設計された選択メカニズムも用意されています。


Copyright © GrapeCity, Inc. All rights reserved.

# クイックスタート

# レポート生成方法

**C1Report** には、2 種類のレポート生成方法が用意されています。このセクションでは、それらの生成方法の違いとどちらを選択すべきかの目安について、説明します。

# レポート生成方法の違い

**C1Report** は、レポートを生成するために以下の2つの方法を提供します。

- **C1PrintDocument.ImportC1Report** メソッドを使用
- **C1Report** コンポーネントを使用

どちらの方法でも既存のレポートを正常に生成できますが、上記の2つの方法の主な相違点は以下の通りです。

## データ連結:

インポートするとデータ連結されたC1PrintDocumentが生成され、データが取得されます。したがって、データが更新されるとドキュメントも更新されます。

一方、**C1Report** コンポーネントを使用する場合は、作成されたドキュメントで生成中に取得されたデータは既に組み込まれています。したがって、ドキュメントはデータと連結されていませんので、データを更新したときは、再びレンダリングする必要があります。

## ドキュメント構造:

レポートをC1PrintDocumentでインポートする場合は、フィールドおよびセクションのすべてはそれぞれ **RenderField**クラスおよび **RenderSection** クラスとして示されています。レポートの定義でグループが含まれている場合は、各グループは **RenderArea** オブジェクトとして表されます。グループ内のヘッダとフッタは RenderSection クラス、ネストされたグループは RenderArea クラスが使用されます。

**C1Report** コンポーネントを使用すると、各セクションは RenderArea オブジェクトとしてレンダリングされます。フィールドのレンダリングは以下の通りです。

- **LineSlant** プロパティが指定されている場合は、 **RenderLine** クラス が生成されます。
- 画像またはバーコードが指定されている場合は **RenderImage** クラスが生成されます。
- **RTF** プロパティは **True** に設定されている場合は、**RenderRichText** が生成されます。
- その他の場合は、**RenderText** クラスが生成されます。

## ページサイズ:

**CustomWidth** および **CustomHeight** がどちらも0に設定されているレポートをプリンタがインストールされていないシステムにインポートすると、C1PrintDocument では、現在のロケールに基づいて用紙サイズが決定されます (たとえば、用紙サイズはアメリカやカナダではレター、日本やヨーロッパでは A4 に設定されます)。

同じレポートを C1Report コンポーネントで読み込んだ場合は、用紙サイズは C1.Win.C1Report.C1Report と同様に現在のロケールに基づいて用紙サイズが決定されます。

## デフォルトプリンタ:

1つ以上のプリンタがインストールされているシステムでは、インポートされたレポートのデフォルトページサイズは通常


Copyright © GrapeCity, Inc. All rights reserved.

の**C1PrintDocument**と同じロジックを使用して決定されます (プリンタを決定するには、 **MeasurementPrinterName**プロパティが使用されます)。この動作は、デフォルトネットワークプリンタがインストールされているシステムで長い保留時間を回避するために提供されています。**C1Report** コンポーネントの場合は、システムのデフォルトプリンタのデフォルトページサイズが使用されます。

## Import メソッドの制限:

C1Report をC1PrintDocumentにインポートする際は、次の制限があります。

- インポートした場合は、**C1Report** のオブジェクトモデルを参照するまたはレポートのイベントを使用することができません。つまり、C# または VB.NET イベントハンドラに依存するレポートをインポートすると、正しくレンダリングできません。
- スクリプトの制限:
  1. **C1PrintDocument オブジェクト**
     - **Font** プロパティは読み取り専用です。
  2. **Field オブジェクト**
     - **Section**, **Font**, および**Subreport** プロパティは読み取り専用です。
     - **LineSpacing**, **SubreportHasData**, および **LinkValue** プロパティはサポートされません。**LinkTarget** プロパティに式が含まれる場合は、評価されずに文字列として使用されます。
  3. **Layout オブジェクト**
     - **ColumnLayout** プロパティはサポートされません。列は常に上から下、左から右に並べられます。
     - **LabelSpacingX**、**LabelSpacingY**、**OverlayReplacements** プロパティはサポートされません。
     - **ForcePageBreak** を **OnPrint** ハンドラに使用することはできません。
- レポートのパラメータを入力するためのダイアログは表示されません。その代わりに、パラメータの種類によってデフォルト値が使用されます (数値には0、文字列には空文字列、日付には現在の日付が使用されます)。
- **PageHeader** および **PageFooter** では、データベースフィールドを使用できません。
- **C1Report** では、段組みレポートのレポートヘッダがすべての列に出力されますが、**C1PrintDocument** では、最初の列だけに出力されます。
- 左から右方向の列はサポートされません。列は常に上から下方向へ設定されます。
- **C1Report** で画像を格納している **Field** クラス の **CanGrow** プロパティ が **True**、**PictureScale** プロパティ が **Clip** に設定されている場合は、フィールドの幅と画像の幅が等しくなります。これをインポートすると、フィールドの幅は画像幅と比例的にスケールされます。

# レポート生成方法の選択

Reports for WPF でレポートを生成する方法は 2 つ(**C1Report** コンポーネントの使用、または、**C1PrintDocument**をインポート)ありますが、最適な方法を選択するには以下の条件を確認します。

- アプリケーションで必須機能は制限事項に記載されている場合は、**C1Report** を使用します (Import の制限事項の詳細については、レポート生成方法の違い を参照してください) 。
- 今まで **C1Report** の旧バージョンを使用しており、**C1PrintDocument** のオブジェクトモデルに慣れていない場合は、**C1Preview** が提供する **C1Report** コンポーネントを使用できます。
- 一方、**C1PrintDocument** を使用した経験があるか、新しいプロジェクトを開始する場合は、インポートを使用することを推奨されています。インポートのメリットは以下の通りです:
  - **C1PrintDocument** にレポート定義ファイルをインポートすると、生成されたドキュメントを普通の**C1PrintDocument**として調整できます。例えば、本文にオブジェクトを追加したり、プロパティを変更したりすることができ、これらの修正はドキュメントを更新してからも表示されます。
  - **C1Report** のいくつかの問題はインポートで修正されました。例えば、**C1Report** で隣接したオブジェクトを複数ページに正しくスプリットできません。また、スプリットされたオブジェクトの境界線が正しく描画されません。**C1PrintDocument**にインポートされたレポートでこの問題が発生しません。
  - メモリおよびスピード上にインポートは最適な方法です。
  - 将来の機能拡張はインポートを注目にして行われます。


Copyright © GrapeCity, Inc. All rights reserved.

# レポート作成を開始する

このセクションでは、基本的な **C1Report** 機能を使用して単純なレポートを作成する方法について学びます。このセクションは、C1Report のすべての機能の包括的なチュートリアルとなるものではありませんが、コンポーネントを使用するためのクイックスタートを提供し、いくつかの一般的なアプローチを強調して説明します。

# C1Report クイックスタート

デスクトップおよび Web 上のさまざまな多くのシナリオで C1Report を使用できますが、手順のメインシーケンスは常に同じです。

1. **レポート定義の作成**

   これは、**C1Report デザイナ**によるか、Microsoft Access のレポートデザイナを使用して直接実行し、C1Report デザイナにインポートできます。オブジェクトモデルを使用してグループとフィールドを追加するか、またはカスタム XML ファイルを記述するかのいずれかによって、コードを使用して実行することもできます。

2. **C1Report コンポーネントへのレポートのロード**

   これは、[**レポートのロード**]コンテキストメニューを使用するか、プログラム操作で **C1Report.Load** メソッドを使用して、デザイン時に実行できます。デザイン時にレポートをロードすると、レポートはコントロールに保持(保存)され、レポート定義ファイルを配布する必要はありません。

3. **レポートの表示(デスクトップアプリケーション)**

   デスクトップアプリケーションを記述する場合、**C1Report.Document** プロパティを使用して、レポートを **C1PrintPreview** コントロール(または、**MicrosoftPrintPreview コントロール**)に表示できます。プレビューコントロールは、画面上にレポートを表示し、ユーザーは、フルズーム、パンなどを使用してそれをプレビューできます。

4. **レポートの表示(Web アプリケーション)**

   Web アプリケーションを記述する場合、C1Report.**RenderToFile** メソッドを使用して、レポートを HTML または PDF ファイルに表示し、ユーザーは任意のブラウザを使用して、これらを表示できます。

以下の手順では、レポート定義を作成する方法、レポートを C1Report コンポーネントにロードする方法、およびレポートを表示する方法について説明します。

# 手順1:レポート定義の作成

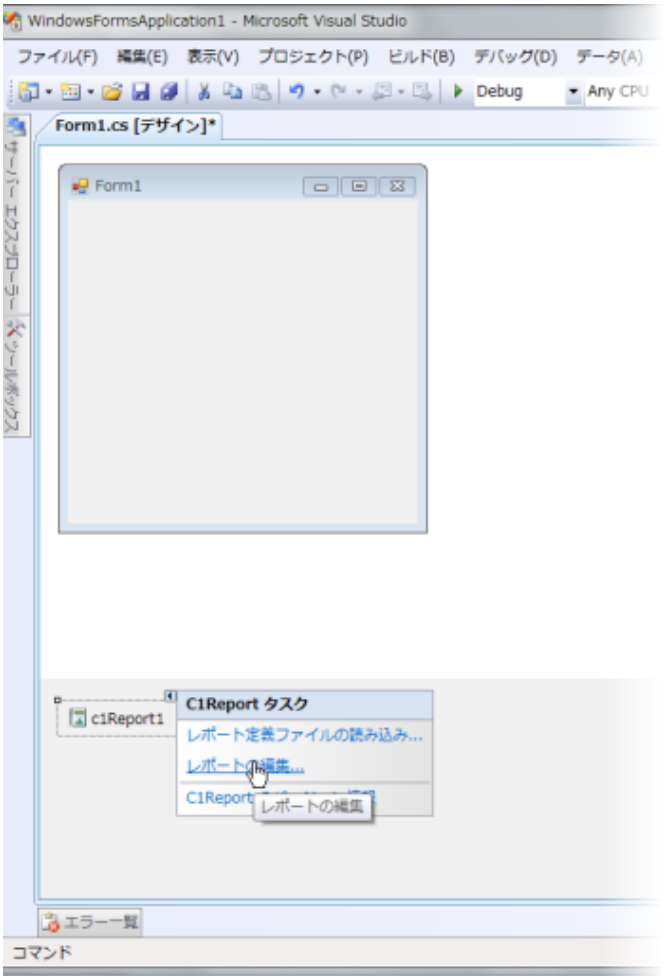
以下のトピックでは、**C1ReportDesigner** を使用してレポート定義を作成する方法について説明します。レポート定義の作成は、レポートの表示と同じではないことに注意してください。レポートを表示するために、単に既存の定義をロードし、**C1Report.Render** メソッドを呼び出すことができます。レポート定義を作成する最も簡単な方法は、Microsoft Access のレポートデザイナに類似したスタンドアプリケーションである **C1ReportDesigner** を使用することです。

> **Note:** Y注意:コードを使用して、ゼロからレポートを作成することもできます。この方法は余分な作業が必要なりますが、完全な柔軟性が提供されます。独自のレポートデザイナまたは臨時のレポートデザイナを記述することもできます。詳細については、「レポート定義の作成」を参照してください。

C1Report ウィザードは、新しいレポートを作成する手順をステップバイステップで最初から最後まで実行します。開始するには、以下の手順を実行します。

1. まず、.NET プロジェクトを作成し、**C1Report** コンポーネントをツールボックスに追加します。新しいプロジェクトの作成についての詳細は、「プロジェクトの作成」を参照してください。
2. ツールボックスで、**C1Report** アイコンをダブルクリックし、コンポーネントをプロジェクトに追加します。コンポーネントは、コンポーネントトレイ内のフォームの下に表示されます。
3. **C1Report** コンポーネントのスマートタグをクリックし、その**タスクメニュー**から[**レポートの編集**]を選択します。


Copyright © GrapeCity, Inc. All rights reserved.




**C1Report ウィザードで**、以下の5つの手順を実行してレポートを作成します。

**C1Report ウィザード**で、以下の5つの手順を実行してレポートを作成します。

1. **新しいレポートのデータソースを選択します。**

   このページを使用して、レポートのデータを取得するために使用する **DataSource.ConnectionString** と **DataSource.RecordSource** を選択します。

   次の3つの方法で、**DataSource.ConnectionString** を指定できます。

   - 文字列を直接エディタに入力します。
   - ドロップダウンリストを使用して、最近使用した接続文字列を選択します(デザイナは、直前に使用した8つの接続文字列の記録を保持します)。
   - 〈**...**〉ボタンをクリックして、標準接続文字列ビルダーを起動します。

   次の2つの方法で、**DataSource.RecordSource** 文字列を指定できます。

   - [**テーブル**]オプションをクリックして、リストからテーブルを選択します。
   - [**SQL**]オプションをクリックして、SQL ステートメントをエディタに入力するか、または貼り付けます。

   **手順1の実行**:

   以下の手順を実行します。

   1. 〈**...**〉ボタンをクリックして、標準接続文字列ビルダーを起動します。
   2. 「**プロバイダ**」タブを選択し、リストからデータプロバイダを選択します。この例では、**Microsoft Jet 4.0 OLE DB Provider** を選択します。
   3. 〈**次へ**〉ボタンをクリックするか、「**接続**」タブを選択します。ここで、データソースを選択する必要があります。
   4. ダイアログボックスを選択するには、〈**...**〉ボタンをクリックします。[**Accessデータベースの選**択]ダイアログボックスが表示されます。この例では、C:\Users\<ユーザー名>\Documents\ComponentOne Samples\Common ディレクトリにある **Nwind.mdb** を選択します。このディレクトリは、デフォルトのインストールパスを反映したものであり、インス

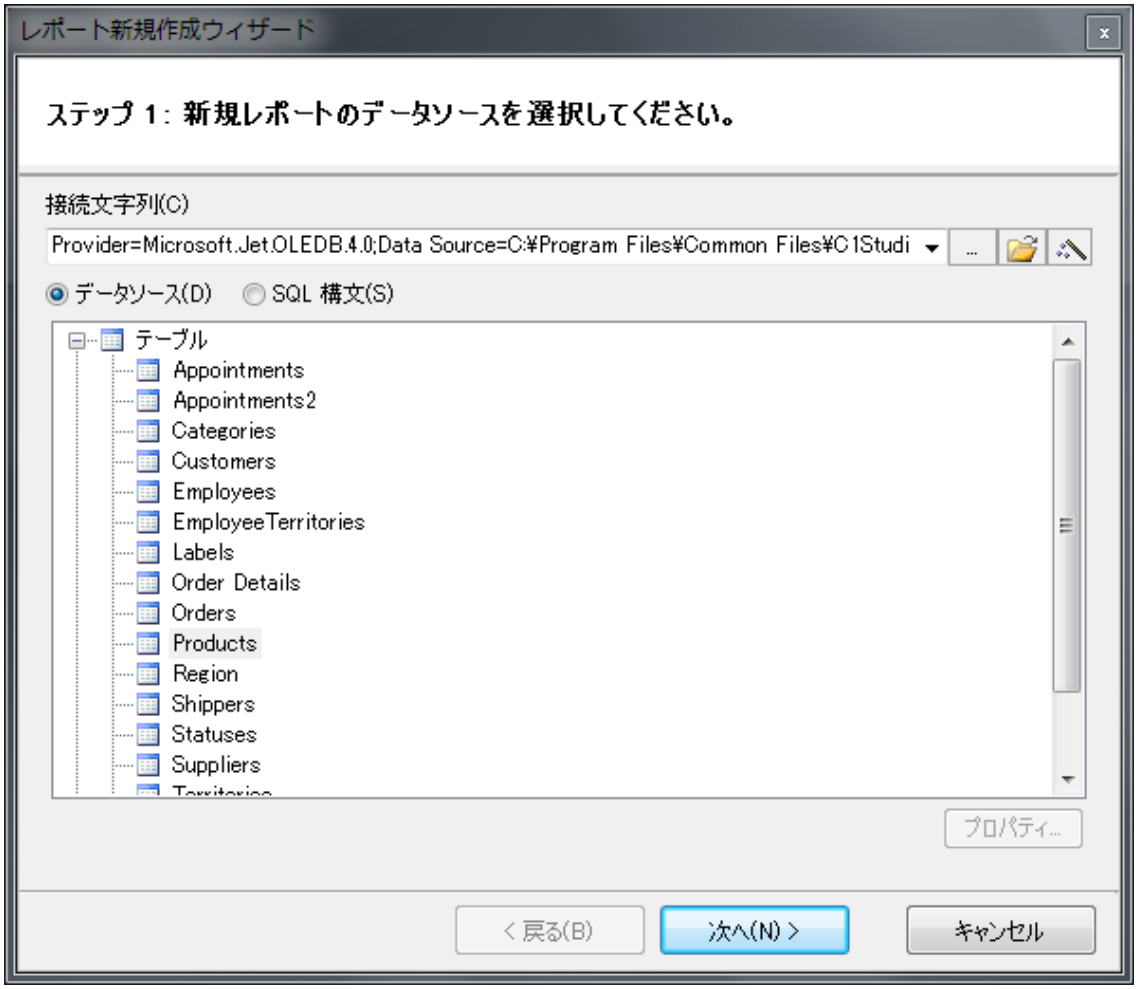

Copyright © GrapeCity, Inc. All rights reserved.

トールパスを変更した場合、パスが異なることがあることに注意してください。

5. 〈**開く**〉をクリックします。接続をテストできます。完了したら〈**OK**〉をクリックします。
6. 〈**OK**〉をクリックして、[**Accessデータベースの選択**]ダイアログボックスを閉じます。
7. データソースの選択を完了したら、実際のデータを提供するテーブル、ビュー、またはストアドプロシージャを選択します。次の2つの方法で、**DataSource.RecordSource** 文字列を指定できます。
   - [**テーブル**]オプションをクリックして、リストから[**商品**]テーブルを選択します。
   - [**SQL**]オプションをクリックして、SQL ステートメントをエディタに入力するか、または貼り付けます。

   たとえば、次のようになります。

   ```
   select * from products
   ```




8. 〈**次へ**〉をクリックします。ウィザードに従って、残りの手順を実行します。

2. **レポートに取り込むフィールドを選択します。**

   このページには、手順1で選択したレコードセットから入手できるフィールドのリストとレポートのグループおよび詳細フィールドを定義する2つのリストが含まれています。グループフィールドでは、データをソートし、要約する方法を定義し、詳細フィールドでは、レポートに表示する情報を定義します。

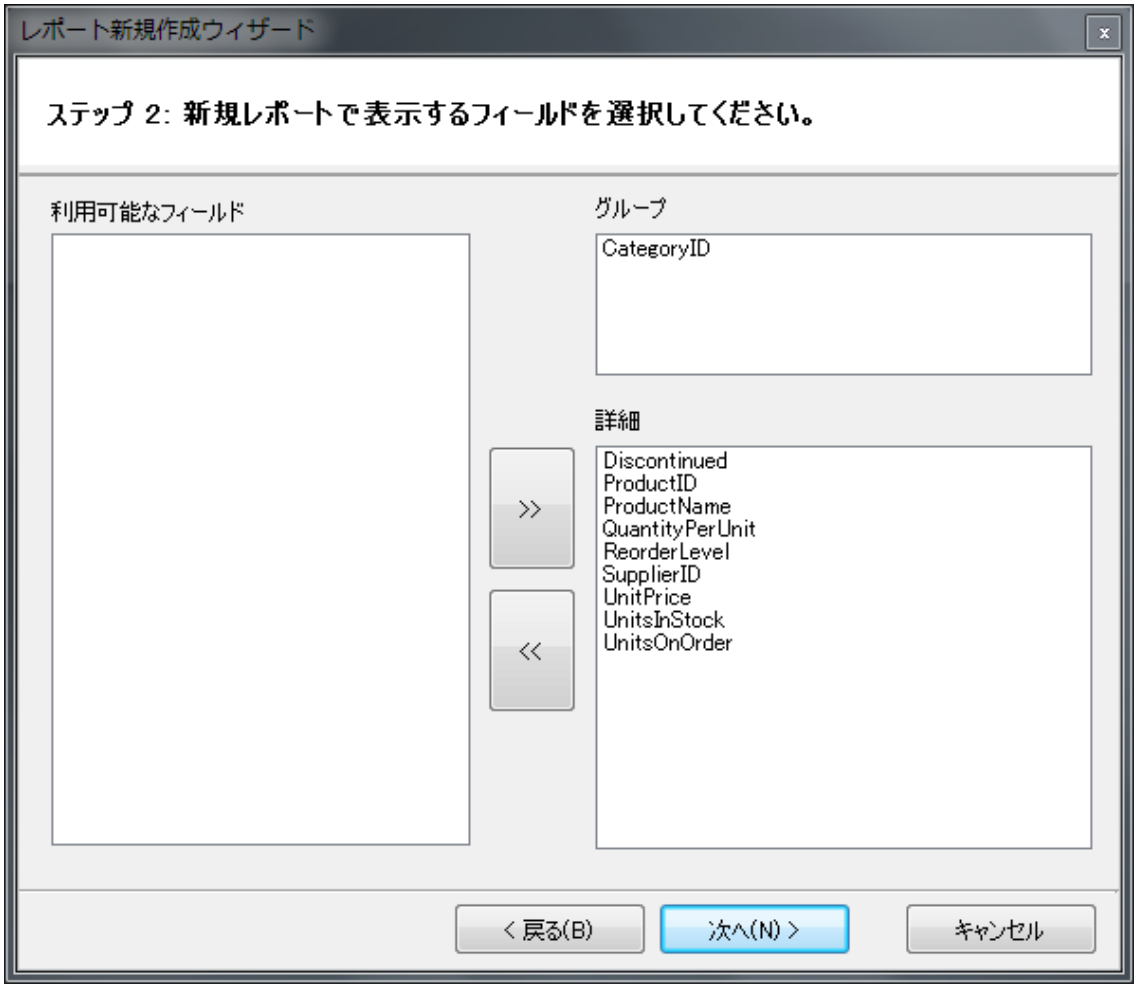
   マウスポインタを使ってドラッグすることで、あるリストから別のリストにフィールドを移動することができます。フィールドを[**詳細**]リストにドラッグして、それらをレポートに取り込むか、リスト内でドラッグして、それらの順序を変更します。ドラッグ操作により、フィールドを[**使用可能**]リストに戻して、それらをレポートから削除します。

   **手順2の実行**:

   以下の手順を実行します。

   1. マウスポインタを使って、**CategoryID** フィールドを選択し、それを[**グループ**]リストにドラッグします。
   2. 〈>>〉ボタンを押して、残りのフィールドを[**詳細**]リストに移動します。


Copyright © GrapeCity, Inc. All rights reserved.




3. 〈**次へ**〉をクリックします。ウィザードに従って、残りの手順を実行します。

3. **新しいレポートのレイアウトを選択します。**

   このページには、ページ上でデータを整理する方法を定義するいくつかのオプションが用意されています。レイアウトを選択すると、レイアウトがページ上でどのように表示されるかを確認できるサムネイルプレビューが左側に表示されます。レポートには、グループを含まないレポートとグループを含むレポートの2つのレイアウトのグループがあります。希望する最終レポートに最も近いと思われるレイアウトを選択します。

   このページでは、ページの向きとフィールドをページ幅に合うように調整するかどうかを選択できます。

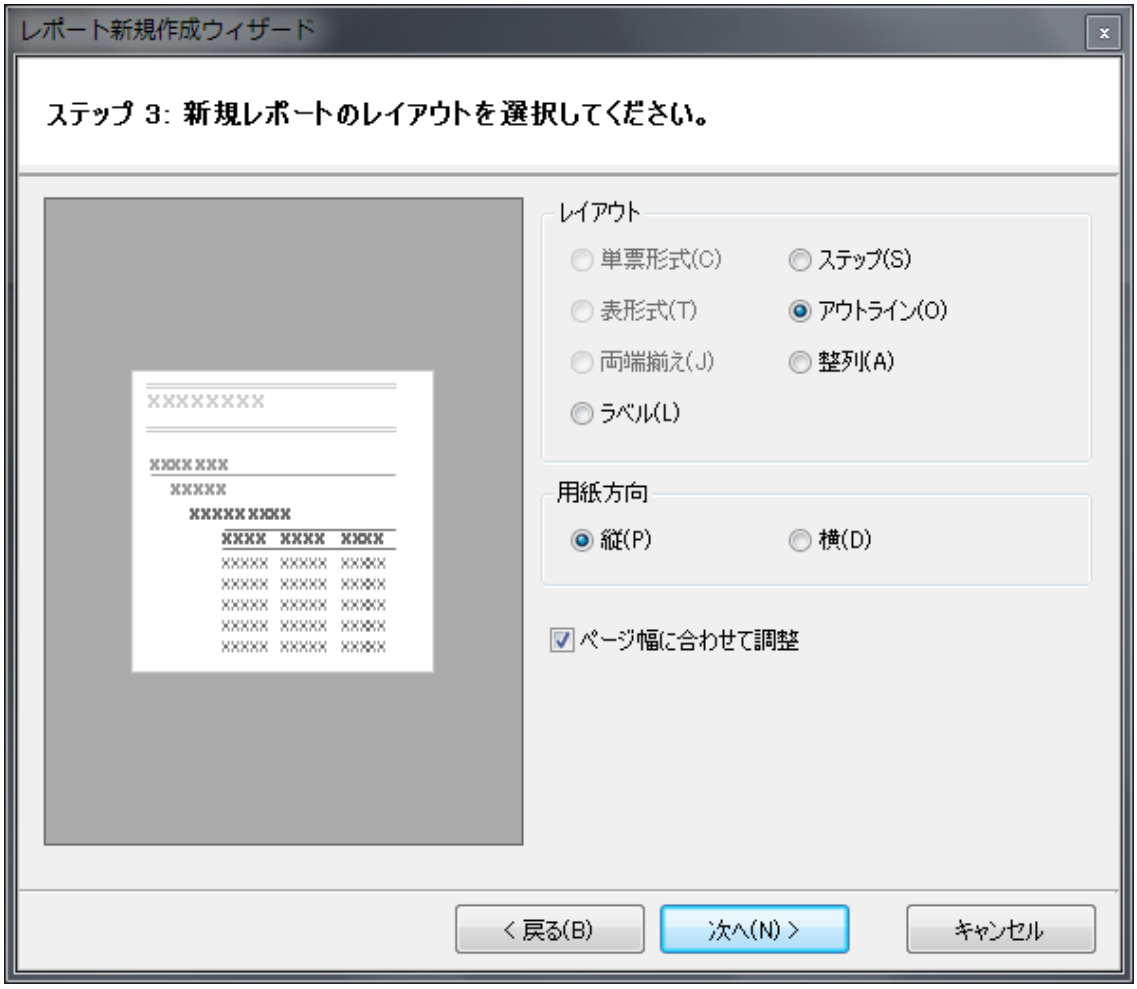
   レイアウトの[**ラベル**]オプションは、Avery スタイルのラベルの印刷に使用されます。このオプションを選択すると、印刷するラベルの種類を指定するように求めるページが表示されます。

   **手順3の実行**:

   以下の手順を実行します。

   1. **アウトライン**レイアウトを維持します。


Copyright © GrapeCity, Inc. All rights reserved.




2. 〈**次へ**〉をクリックします。ウィザードに従って、残りの手順を実行します。

4. **新しいレポートのスタイルを選択します。**

   このページでは、新しいレポートで使用するフォントと色を選択できます。前のページと同様に、表示されるスタイルを確認できるプレビューが表示されます。最も適切と思われるスタイルを選択します（後で、スタイルを改善し、詳細を調整できることを忘れないでください）。

   **手順4の実行**:

   1. **MS PGothic** スタイルを選択します。

   2. 〈**次へ**〉をクリックします。ウィザードに従って、残りの手順を実行します。

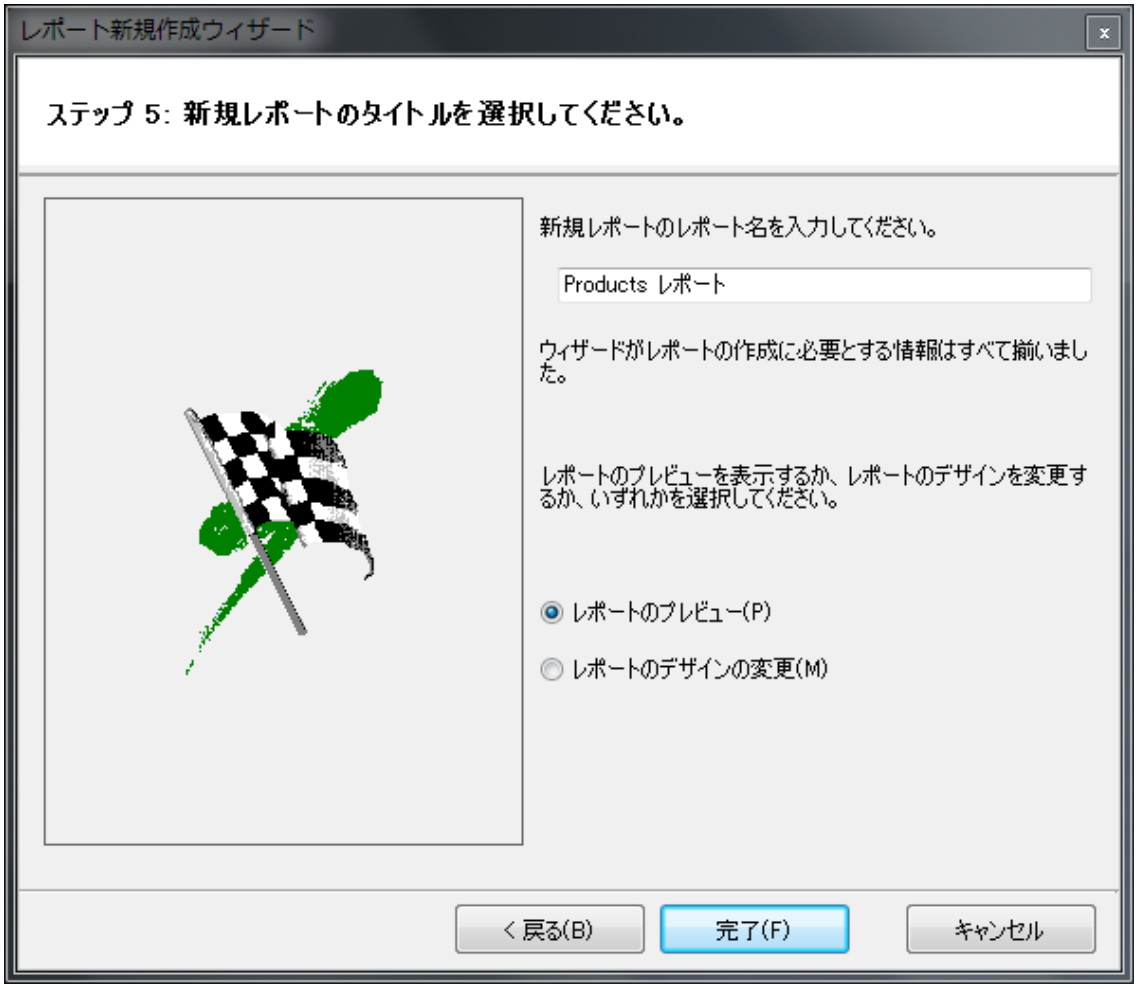
5. **新しいレポートのタイトルを選択します。**

   この最後のページでは、新しいレポートのタイトルを選択し、新しいレポートをすぐにプレビューするか、または編集モードに入り、プレビューする前にデザインの改善を開始するかを指定できます。

   **手順5の実行**:

   1. **商品レポート**など、新しいレポートのタイトルを入力します。


Copyright © GrapeCity, Inc. All rights reserved.




2. [**レポートをプレビューする**]を選択し、〈**完了**〉をクリックします。
   レポートが、**デザイナ**のプレビューペインに即座に表示されます。

レポートは、いくつかの調整が必要であることに気付きます。次の手順では、レポートを修正する方法について学びます。

## 手順2:レポートの修正

プレビューモードの**デザイナ**では、レポートを調整することはできません。

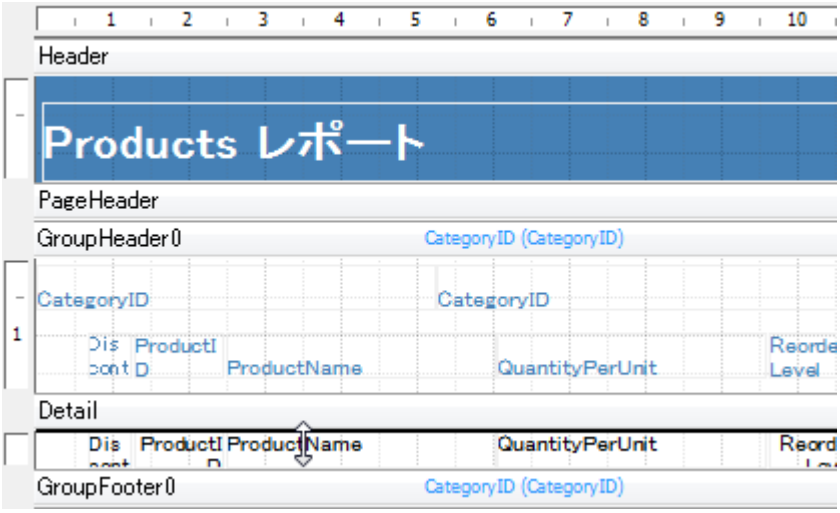
1. 〈**デザイン**〉ボタンをクリックして、デザインモードに切り替え、修正を開始します。
2. 〈**デザイン**〉ボタンを選択すると、メインウィンドウの右ペインがプレビューモードからデザインモードに切り替わり、レポートを作成するコントロールとフィールドが表示されます。

**レポートを修正する:**

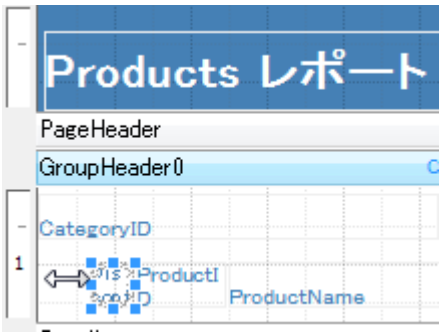
この例では、[グループヘッダー]セクションとフィールドのサイズを変更すると共に、フィールド値を書式設定します。これを行うには、以下の手順を実行します。

1. [グループヘッダー]セクションのサイズを変更するには、その枠を選択し、マウスポインタを使って希望する位置にドラッグします。


Copyright © GrapeCity, Inc. All rights reserved.

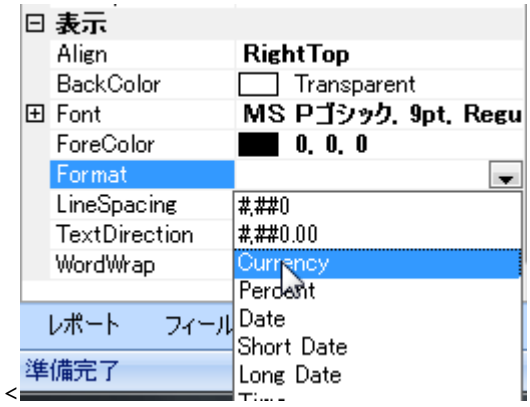



2. マウスポインタを使用し、フィールドの角をドラッグして、フィールドのサイズを変更します。



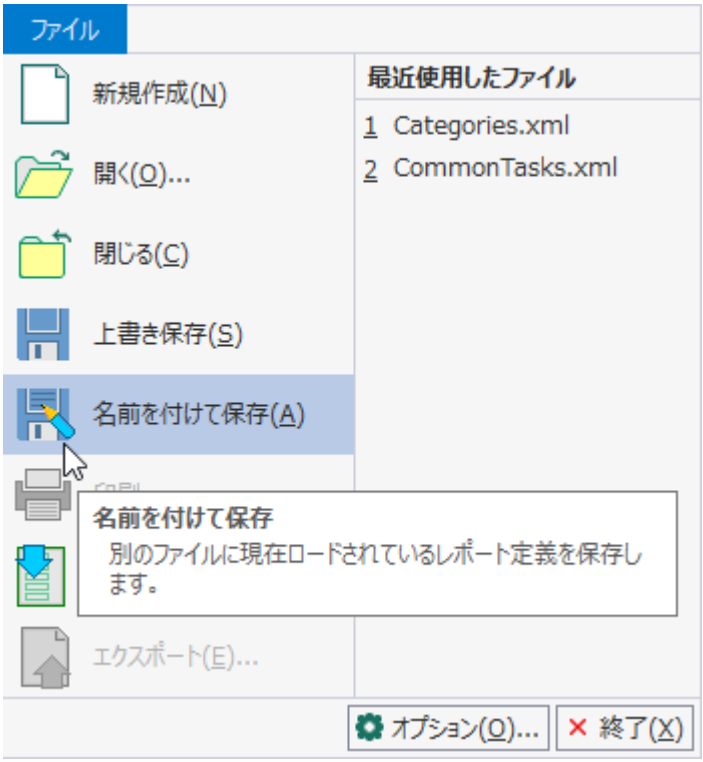

> **ヒント**:テキストがフィールド内に収まらない場合は、プロパティウィンドウで、フィールドの Appearance.WordWrap プロパティを True に設定します。

3. [単価]列の下にある[詳細]セクションで、**UnitPriceCtl** フィールドを選択します。
4. プロパティウィンドウで、フィールドの Apperance.**Format** プロパティを **Currency** に設定します。




5. 〈**プレビュー**〉ボタンをクリックし、プレビューモードに切り替えて、修正結果を確認します。
6. 〈**デザイン**〉ボタンをクリックしプレビューを閉じて、デザインビューに切り替えます。
7. 〈**ファイル**〉ボタンをクリックし、表示されるメニューから〈**名前を付けて保存**〉を選択します。


Copyright © GrapeCity, Inc. All rights reserved.




8. [**レポート定義ファイルの保存**]ダイアログボックスで、[**ファイル名**]ボックスに **ProductsReport.xml** を入力します。後で利用するために、覚えやすい場所にファイルを保存します。
9. **デザイナ**を閉じて、Visual Studio プロジェクトに戻ります。

これで、レポート定義ファイルの作成は完了です。次の手順では、**C1Report** コンポーネントにレポートをロードします。

## 手順3：C1Report コンポーネントへのレポートのロード

デザイン時にファイルからレポート定義をロードするには、次のいずれかのタスクを実行します。

- **C1Report** コンポーネントを右クリックし、[**レポートのロード**]メニューオプションを選択します。

  あるいは

- **C1Report** コンポーネントの上にあるスマートタグ（▶）をクリックし、[**C1Report タスク**]メニューから[**レポートのロード**]を選択します。

[**レポートの選択**]ダイアログボックスを使用し、希望するレポートを選択して、以下の手順を実行します。

1. 〈**...**〉ボタンをクリックします。[**開く**]ダイアログボックスが表示されます。
2. **ProductsReport.xml** ファイルを保存した場所に移動し、ファイルを選択して、〈**開く**〉をクリックします。
3. 利用可能なレポート定義が〈**レポート**〉ドロップダウンボックスにリスト表示されます。ロードする**商品レポート**定義を選択します。
4. 〈**ロード**〉をクリックし、〈**OK**〉をクリックして、ダイアログボックスを閉じます。


Copyright © GrapeCity, Inc. All rights reserved.

次の手順では、レポートをプレビューコントロールに表示します。

## 手順4:レポートの表示

レポート定義を作成し、データソースを定義し、**C1Report** コンポーネントにロードすると、プリンタ、プレビューコントロール、またはレポートファイルにレポートを表示できます。

レポートをプレビューするには、**C1Report. Document** プロパティを使用します。レポートを C1PrintPreviewControl の **Document** プロパティに割り当てるか、.NET **PrintPreview** または **PrintPreviewDialog** コントロールに割り当てると、レポートがプレビューコントロールに表示され、ユーザーはレポートを閲覧、ズーム、または印刷できます。

> **注意:C1Report** は .NET プレビューで動作しますが、組み込まれた **Reports for WinForms** プレビューコントロールで動作するように最適化されています。組み込まれたコントロールで使用する場合、各レポートページが生成されたときにそれを表示できます。標準コントロールでは、最初のページを表示する前に、レポート全体が使用できるようになるまで待つ必要があります。

この手順を実行するには:

1. ツールボックスで、**C1PrintPreviewControl** アイコンをダブルクリックし、コンポーネントをプロジェクトに追加します。
2. プロパティウィンドウで、**C1PrintPreviewControl.Dock** プロパティを **Fill** に設定します。
3. マウスを使って Windows フォームを選択し、ドラッグして、サイズを変更します。この例では、プレビューパネルが適切に表示されるように、フォームを **600x500** にサイズ変更します。
4. フォームをダブルクリックし、次のコードを **Form_Load** イベントハンドラに入力します。

**Visual Basic コードの書き方**

```
Visual Basic
' レポート定義をロードします
C1Report1.Load("C:\ProductsReport.xml", "Products Report")
' ドキュメントをプレビューします
C1PrintPreviewControl1.Document = C1Report1.Document
```

**C# コードの書き方**

```
C#
// レポート定義をロードします
c1Report1.Load(@"C:\ProductsReport.xml", "Products Report");
```


Copyright © GrapeCity, Inc. All rights reserved.

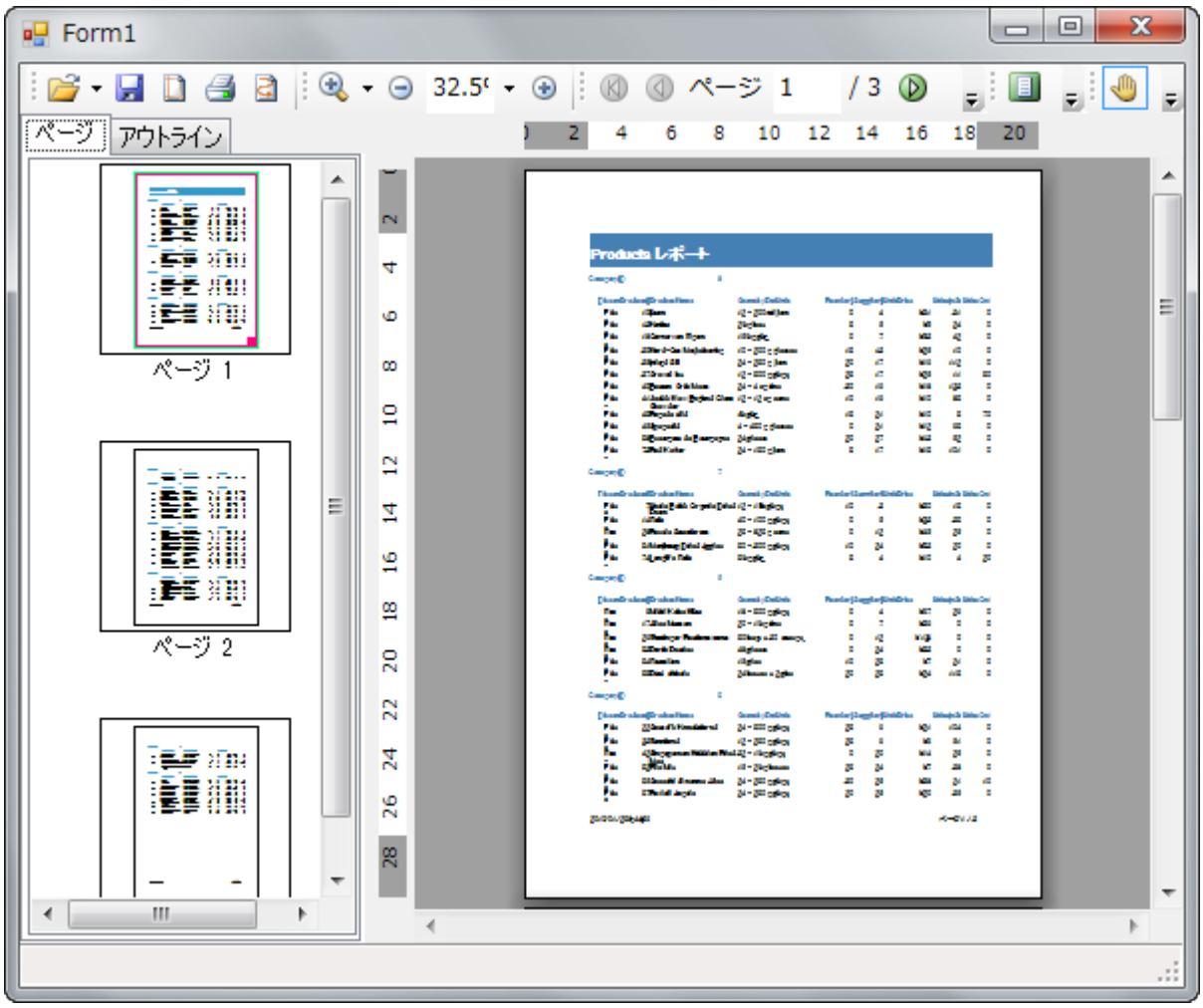
```
// ドキュメントをプレビューします
c1PrintPreviewControl1.Document = c1Report1.Document;
```

上記のディレクトリは、**ProductsReport.xml** ファイルが保存されている場所に変更する必要があります。

**C1Report.Load(String, String)** メソッドには、以下のパラメータが含まれていることに注意してください。

- *fileName*:XML レポート定義ファイルの完全名。
- *reportName*:ファイルから取得するレポートの名前(大文字と小文字を区別しない)。

5. **[デバッグ開始]**ボタンをクリックして、アプリケーションを実行します。レポートがプレビューコントロールに表示されます。




おめでとうございます! これで、単純なレポート定義の作成、レポートの修正、C1Report コンポーネントへのレポートのロード、プレビューコントロールへのレポートの表示が完了しました。さらに VBScript 式を使用してレポート定義をカスタマイズする方法を学ぶには、以下のセクションをお読みください。

# VBScript 式を作成する

VBScript 式は、レポートのコンテンツを取得、計算、表示、グループ化、ソート、フィルタ、パラメータ化、および書式設定するためにレポート定義内で広く使用されます。いくつかの式は自動的に作成されます。たとえば、ツールボックスからフィールドをレポートのセクションにドラッグすると、テキストボックスにそのフィールドの値を取得する式が表示されます。ただし、ほとんどの場合、レポートに機能を追加するには独自に式を作成する必要があります。

**C1Report** は、計算フィールドでの式の評価やレポートイベントの処理を VBScript に依存しています。

VBScript は多機能の言語で、**C1Report** の式を記述する際は、VBScript のほとんどのメソッドと関数にアクセスできます。


Copyright © GrapeCity, Inc. All rights reserved.

VBScript 固有の機能については、Microsoft Developer's Network (MSDN)をはじめ、多くの書籍や Web サイトで説明されているため、本マニュアルでは説明しません。

次のトピックでは、VBScript 式を作成してレポートにより多くの機能を提供する方法を紹介します。

# 値に応じてフィールドを書式設定する

これは、おそらく最も一般的な OnPrint プロパティの使用方法です。Nwind.xml サンプルファイルの "Orders by Value" レポートを例にします。このレポートでは、製品別にグループ化された注文数がリストされます。別のフィールドを使って在庫数を表示する代わりに、製品の名前を赤色の太字で表示することによって、在庫が追加注文レベルより少ない製品を強調表示します。

## コードを使用して、追加注文レベルより少ない製品を強調表示するには

在庫が追加注文レベルより少ない製品を強調表示するためには、次のようなイベントスクリプトを使用します。

**Visual Basic コードの書き方**

Visual Basic

```
Dim script As String = _
  "If UnitsInStock < ReorderLevel Then" & vbCrLf & _
  "ProductNameCtl.ForeColor = RGB(255,0,0)" & vbCrLf & _
  "ProductNameCtl.Font.Bold = True" & vbCrLf & _
  "Else" & vbCrLf & _
  "ProductNameCtl.ForeColor = RGB(0,0,0)" & vbCrLf & _
  "ProductNameCtl.Font.Bold = False" & vbCrLf & _
  "End If"
c1r.Sections.Detail.OnPrint = script
```

**C# コードの書き方**

C#

```
If UnitsInStock < ReorderLevel Then
  ProductNameCtl.ForeColor = RGB(255,0,0)
  ProductNameCtl.Font.Bold = True
Else
  ProductNameCtl.ForeColor = RGB(0,0,0)
  ProductNameCtl.Font.Bold = False
End If
```

このコードは、VBScript のイベントハンドラを1つの文字列として作成し、その文字列をセクションの **OnPrint** プロパティに割り当てます。C1Report デザイナを使用すると、上のコードを記述する代わりに、イベントエディタウィンドウに直接スクリプトコードを入力できます。

## C1Report デザイナを使用して、追加注文レベルより少ない製品を強調表示するには

コードを記述する代わりに、**C1Report デザイナ**を使用して、詳細**セクションの OnPrint** プロパティの VBScript エディタに次のスクリプトコードを直接入力できます。次の手順を実行します。

1. デザイナの[**プロパティ**]ウィンドウのドロップダウンリストから、[**詳細**]を選択します。これで、セクションで使用できるプ


Copyright © GrapeCity, Inc. All rights reserved.

ロパティが表示されます。

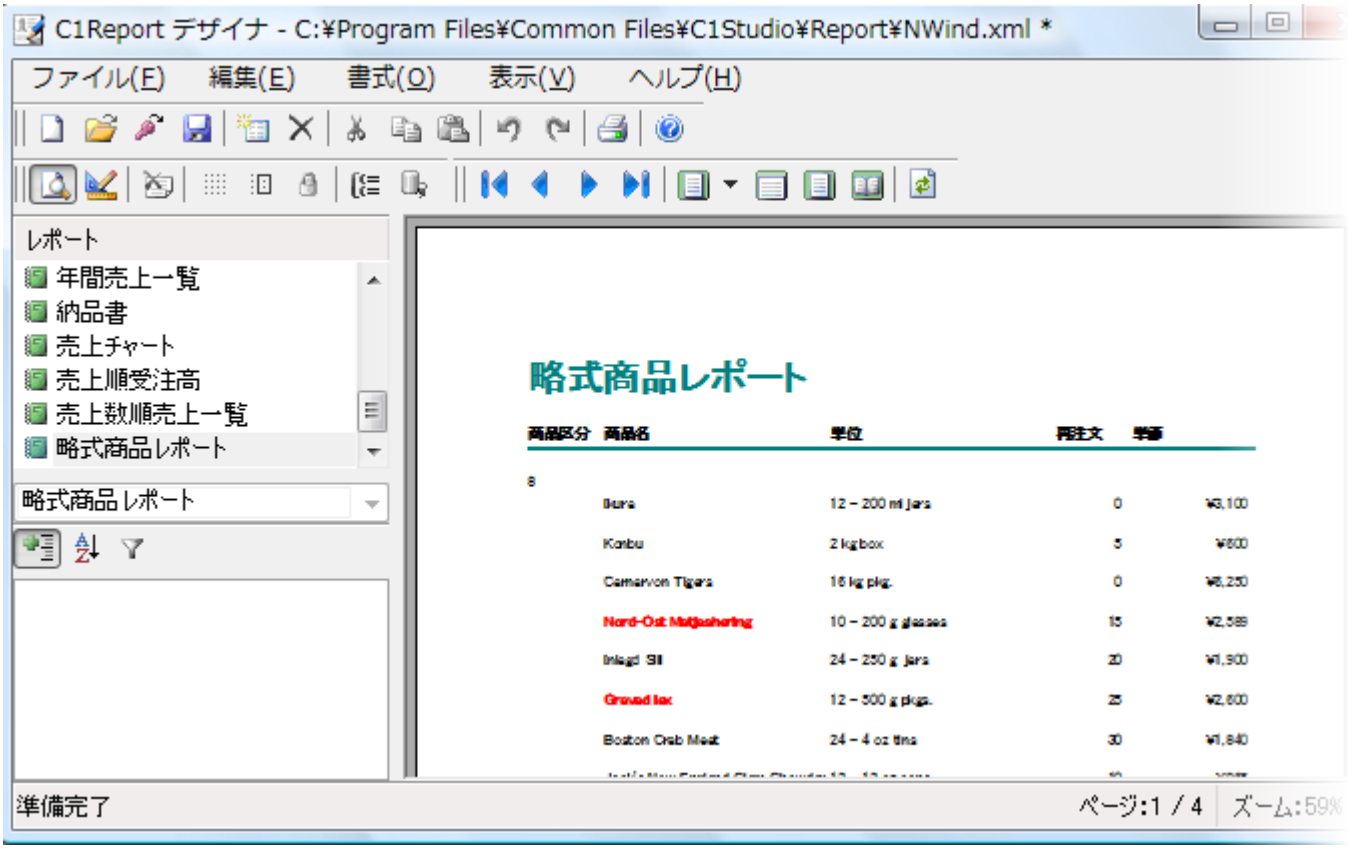
2. **OnPrint** プロパティの横にある空のボックスをクリックし、ドロップダウン矢印をクリックして、リストから[**スクリプトエディタ**]を選択します。.
3. **VBScript エディタ**のウィンドウに、次のスクリプトを入力します。

```
If UnitsInStock < ReorderLevel Then
  ProductNameCtl.ForeColor = RGB(255,0,0)
  ProductNameCtl.Font.Bold = True
Else
  ProductNameCtl.ForeColor = RGB(0,0,0)
  ProductNameCtl.Font.Bold = False
End If
```

4. **OK** をクリックしてエディタを終了します。

コントロールは、このセクションを出力するたびにこの VBScript コードを実行します。スクリプトは "ReorderLevel" データベースフィールドの値を取得し、その値に基づいて、"ProductName" レポートフィールドの **Font.Bold** プロパティと **ForeColor** プロパティを設定します。在庫が追加注文レベルより少ない製品は、製品名が赤色の太字になります。.

次に、レポートの外見を示します。




## データが存在しないセクションを非表示にする

詳細セクションの **OnFormat** プロパティに式を指定することで、データに基づいてレポートフィールドの書式を変更できます。

たとえば、詳細セクションに画像コントロールを含むフィールドがある場合は、レコードに画像のデータが存在しないときに、そのレコードを非表示にできます。データ(この例では、レコードの画像)が存在しない場合に詳細セクションを非表示にするには、詳細セクションの **OnFormat** プロパティに次のスクリプトを追加します。:

```
If isnull(PictureFieldName) Then
  Detail.Visible = false
 Else
```


Copyright © GrapeCity, Inc. All rights reserved.

```
  Detail.Visible = true
End If
```

## コードを使用して、データが存在しないセクションを非表示にするには:

データが存在しないセクション(この例では、レコードの画像)を非表示にするには、次のようなイベントスクリプトを使用します。

**Visual Basic コードの書き方**

Visual Basic

```
C1Report1.Sections.Detail.OnFormat = "Detail.Visible = not
isnull(PictureFieldName)"
```

**C# コードの書き方**

C#

```
c1Report1.Sections.Detail.OnFormat = "Detail.Visible = not
isnull(PictureFieldName)";
```

## デザイナを使用して、データが存在しないセクションを非表示にするには

コードを記述する代わりに、**C1Report デザイナ**を使用して、詳細セクションの **OnFormat** プロパティの VBScript エディタに次のスクリプトコードを直接入力できます。次の手順を実行します。

1. デザイナの[プロパティ]ウィンドウのドロップダウンリストから、[**詳細**]を選択します。これで、セクションで使用できるプロパティが表示されます。
2. **OnFormat** プロパティの横にある空のボックスをクリックし、ドロップダウン矢印をクリックして、リストから[**スクリプトエディタ**]を選択します。
3. **エディタ**で、次の作業を実行します。
   - ウィンドウに次のスクリプトを入力します。

     ```
     If isnull(PictureFieldName) Then
     Detail.Visible = false
     Else
     Detail.Visible = true
     End If
     ```
   - または、簡潔なスクリプトを使用することもできます。`Detail.Visible = not isnull(PictureFieldName)`

# ページカウンタをリセットする

**Page** 変数は、コントロールによって作成されて自動的に更新されます。ページのヘッダーまたはフッタにページ数を追加する場合に便利です。また、新しいグループの先頭でページカウンタをリセットすることもできます。これらの処理はコードを追加したりデザイナを使用したりして実行する事ができます。

## コードを使用する場合:

新しいグループ(新しい国など)の先頭でページカウンタをリセットするには、PageFooter フィールドの **Text** プロパティを設定します。次のコードを入力します。

**Visual Basic コードの書き方**


Copyright © GrapeCity, Inc. All rights reserved.

Visual Basic

```
C1Report1.Fields("PageFooter").Text = "[ShipCountry] & "" "" & [Page]"
```

**C# コードの書き方**

C#

```
c1Report1.Fields("PageFooter").Text = "[ShipCountry] + "" "" + [Page]";
```

## C1Report デザイナを使用する場合:

新しいグループ(新しい国など)の先頭でページカウンタをリセットするには、次の手順を実行して、PageFooter フィールドの **Text** プロパティを設定します。

1. デザイナの[プロパティ]ウィンドウのドロップダウンリストから、PageFooter の PageNumber フィールドを選択します。これで、フィールドで使用できるプロパティが表示されます。
2. **Text** プロパティの横にある空のボックスをクリックし、ドロップダウン矢印をクリックして、リストから[**スクリプトエディタ**]を選択します。
3. VBScript エディタのウィンドウに、次のスクリプトを入力します。`[ShipCountry] & " - Page " & [Page]`
4. **OK** をクリックしてエディタを終了します。

# 印刷プレビューを開始する

このセクションでは、**C1PrintDocument** の基本機能を使用してシンプルなドキュメントを作成する方法を学びます。このセクションは、C1PrintDocument のすべての機能の包括的なチュートリアルとなるものではありませんが、コンポーネントを使用するためのクイックスタートを提供し、いくつかの一般的なアプローチを強調して説明します。

# C1PrintDocument クイックスタート

このセクションでは、**C1PrintDocument** の基本機能を使用して、単純なドキュメントを作成する方法を説明します。このセクションは、C1PrintDocument のすべての機能に関する包括的なチュートリアルではありません。この製品を初めて使用するにあたり、一般的な使用方法を説明するチュートリアルです。

このクイックスタートガイドでは、伝統に従って簡単な「Hello World!」ドキュメントを作成します。次の手順では、プロジェクトにいくつかの **C1Preview コントロール**を追加し、プレビューを設定し、C1Preview の実行時の操作をいくつか使用してみます。

# 手順1:フォームへの C1Preview コントロールの追加

**コントロールを追加し**、フォームを設定して、簡単な「Hello World!」ドキュメントのプレビューを作成します。最も単純なドキュメントとして、「Hello, World!」という文を出力するドキュメントを作成します。 次の手順を完了して、フォームを設定し、ドキュメントを作成してください。

> **メモ**:このトピック内のサンプルコードは、「using C1.C1Preview;」ディレクティブ(C# の場合。他の言語の場合は相当する構文)がファイルに挿入されていることを前提としています。したがって、完全修飾された型名(C1.C1Preview.RenderText など)ではなく、クラス名部分(RenderText)だけを使用します。).

1. 新しい .NET 2.0 Windows アプリケーションを作成し、**HelloWorld** と名前を付けます。
2. ツールボックスに移動し、**C1PrintPreviewControl** をダブルクリックしてフォームに追加します。

   サンプルドキュメントを表示する C1PrintPreviewControl1 という名前の印刷プレビューコントロールがフォームに表示されます。**C1PrintPreviewControl コントロール**は、プレビューペイン、ナビゲーションパネル、テキスト検索パネル、定義済みのボタンを含むツールバーから成る複合コントロールです。


Copyright © GrapeCity, Inc. All rights reserved.

さらに、プロジェクトの[参照]部分に、C1.C1Preview.2 と C1.Win.C1Preview.2 の2つの追加エントリが表示されます。

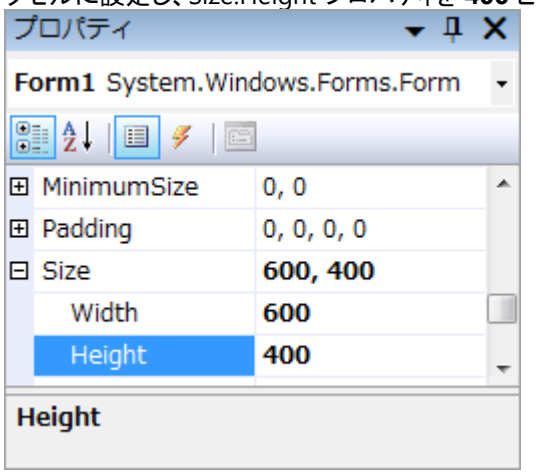
3. ツールボックスで、**C1PrintDocument** アイコンをダブルクリックして、このコントロールをプロジェクトに追加します。新しいコンポーネントは、デフォルトで **C1PrintDocument1** と名前で、フォームの下のコンポーネントトレイに表示されます。

これで、プロジェクトを作成し、そのプロジェクトに **C1Preview コンポーネント**を追加し、クイックスタートガイドの手順1を完了しました。次の手順では、フォームとコントロールを設定します。

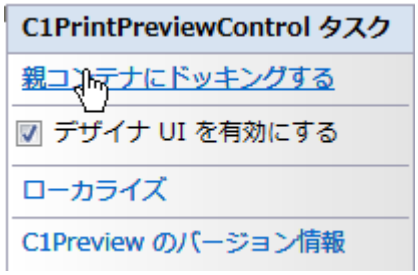
## 手順2:フォームとコントロールの設定

フォームに **C1Preview コントロール**を追加したので、フォームとコントロールを設定します。次の手順を実行します。

1. [**Form1**]を右クリックして選択し、[**プロパティ**]を選択します。プロパティウィンドウで、**Size.Width** プロパティを **600** ピクセルに設定し、Size.Height プロパティを **400** ピクセルに設定して、**C1PrintDocument** のサイズに合わせます。

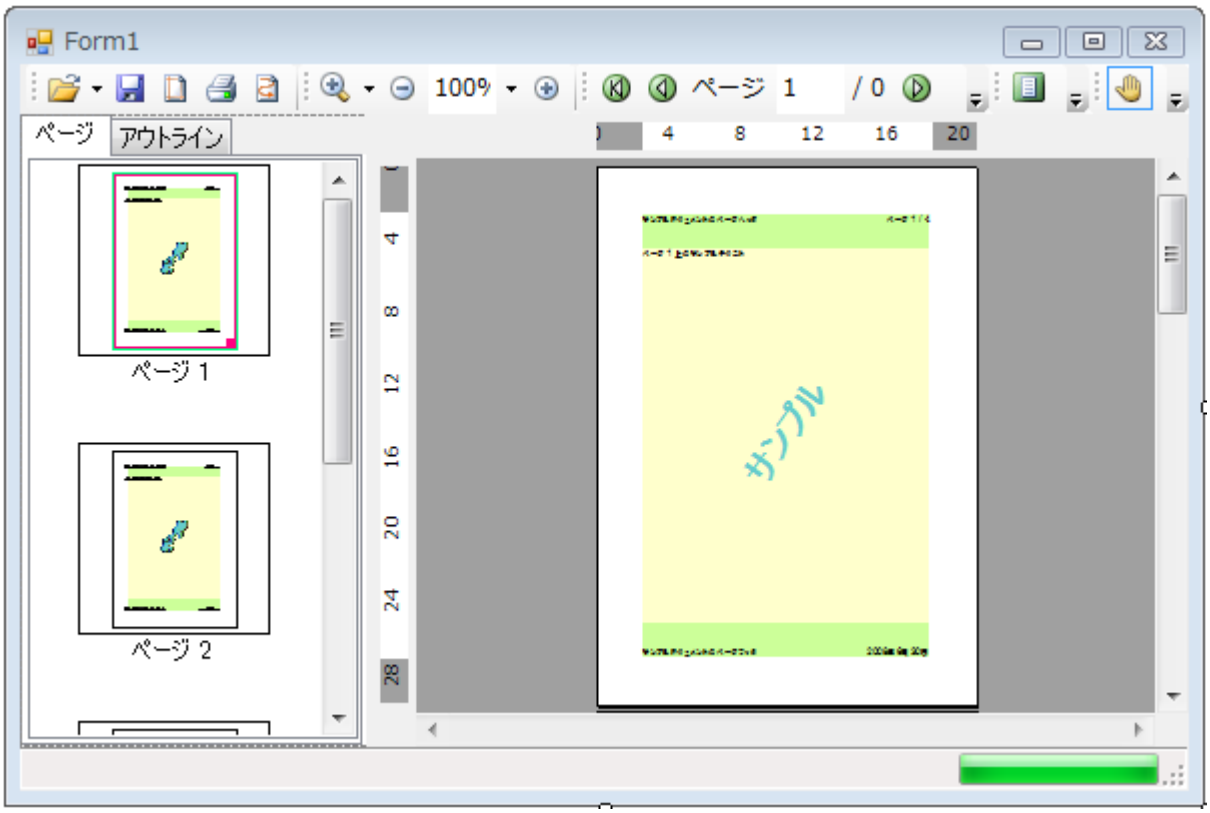



2. **C1PrintPreviewControl1** のスマートタグをクリックして [**C1PrintPreviewControl タスク**]メニューを開き、[**親コンテナにドッキング**]を選択します。

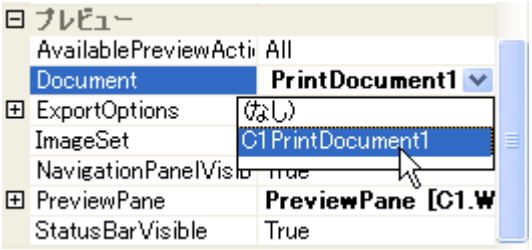



フォームデザイナは次のようになります。


Copyright © GrapeCity, Inc. All rights reserved.




3. **C1PrintPreviewControl1** をクリックして選択し、プロパティウィンドウで Document プロパティを新しく追加した **C1PrintDocument1**（**Document** プロパティドロップダウンリストに表示される）に設定します。




選択したドキュメント（ここでは **C1PrintDocument1**）が実行時にプレビューウィンドウに表示されます。

これで、フォームとコントロールを設定し、C1Preview クイックスタートガイドの手順2を完了しました。次の手順では、プロジェクトにコードを追加します。

## 手順3:プロジェクトへのコードの追加

フォームに **C1Preview コントロール**を追加し、フォームとコントロールをカスタマイズしたので、プロジェクトを実行するために残された手順は1つだけです。この手順では、プロジェクトに表示されるテキストを設定するためのコードをプロジェクトに追加します。

1. フォームのタイトルバーをダブルクリックしてコードビューに切り替え、**Form_Load** イベントのハンドラを作成します。
2. プレビュードキュメントにテキストを追加するための次のコードを **Form_Load** イベントに追加します。:

**Visual Basic コードの書き方**

Visual Basic

```
Me.C1PrintDocument1.Body.Children.Add(New C1.C1Preview.RenderText("Hello,
World!"))
```

**C# コードの書き方**


Copyright © GrapeCity, Inc. All rights reserved.

```
C#
this.c1PrintDocument1.Body.Children.Add(new RenderText("Hello, World!"));
```

3. ドキュメントを生成するための次のコードを **Form_Load** イベントに追加します。

**Visual Basic コードの書き方**

```
Visual Basic
Me.C1PrintDocument1.Generate()
```

**C# コードの書き方**

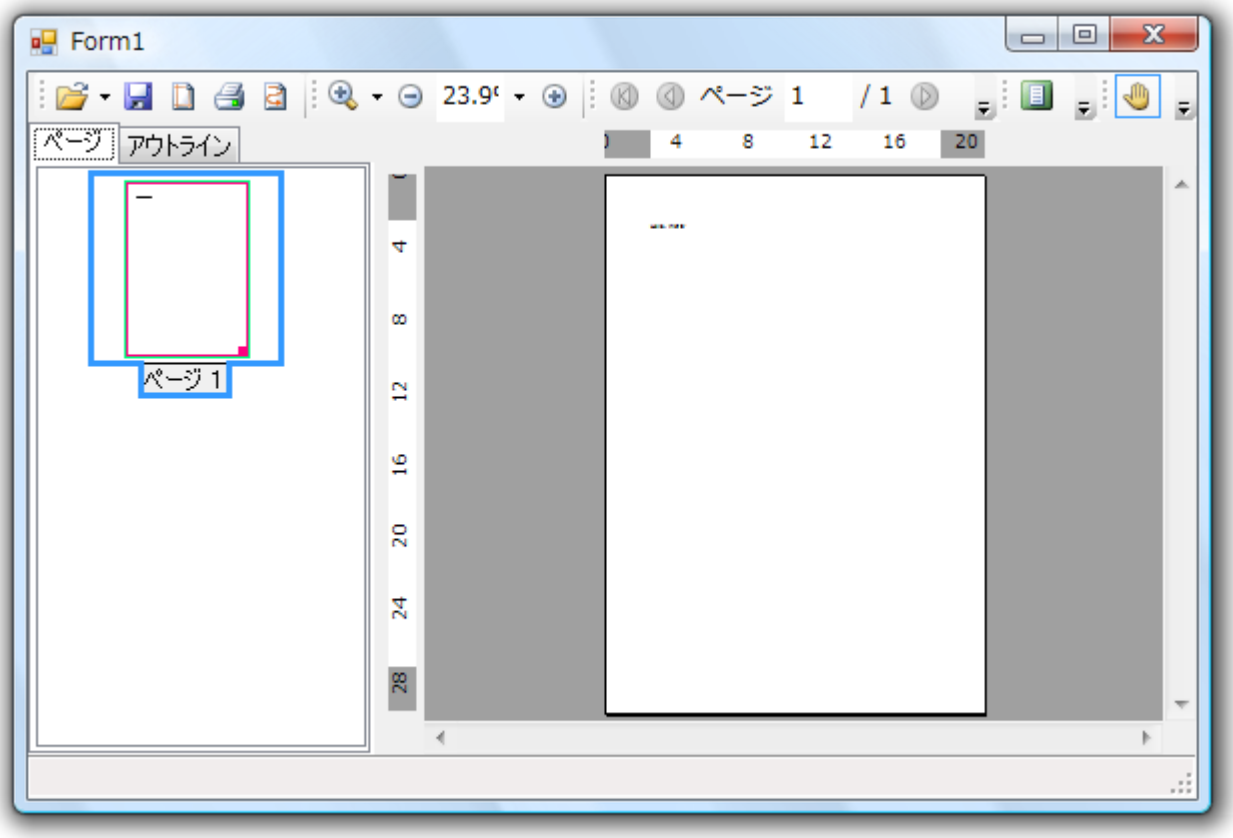
```
C#
this.c1PrintDocument1.Generate();
```

これで、プロジェクトにコードを追加し、C1Preview クイックスタートガイドの手順3を完了しました。最後の手順ではプログラムを実行します。

## 手順4:プログラムの実行

フォームと **C1Preview コンポーネント**を設定し、C1Preview コンポーネントのプロパティを設定し、プロジェクトにコードを追加しました。残された作業は、プログラムを実行し、C1Preview の実行時の操作をいくつか確認することだけです。

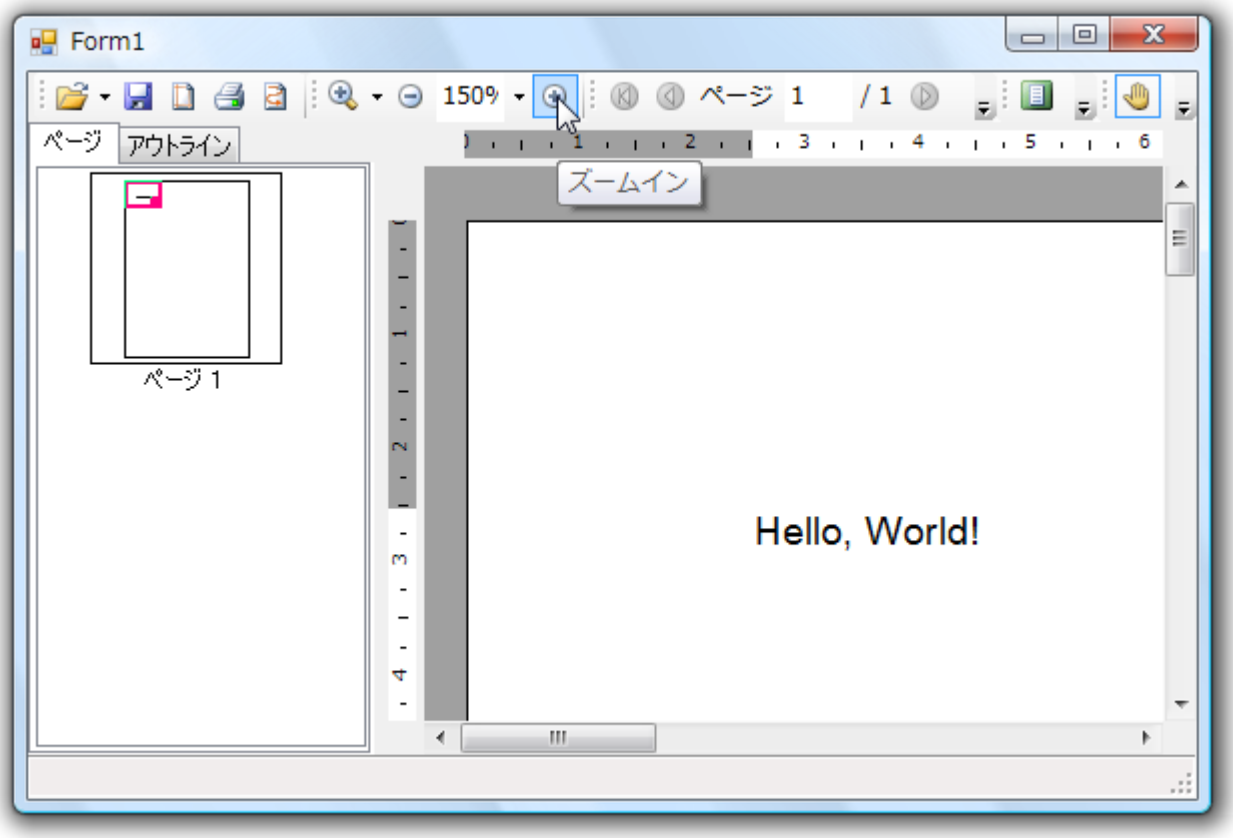
1. プログラムを実行し、次の点を確認します。



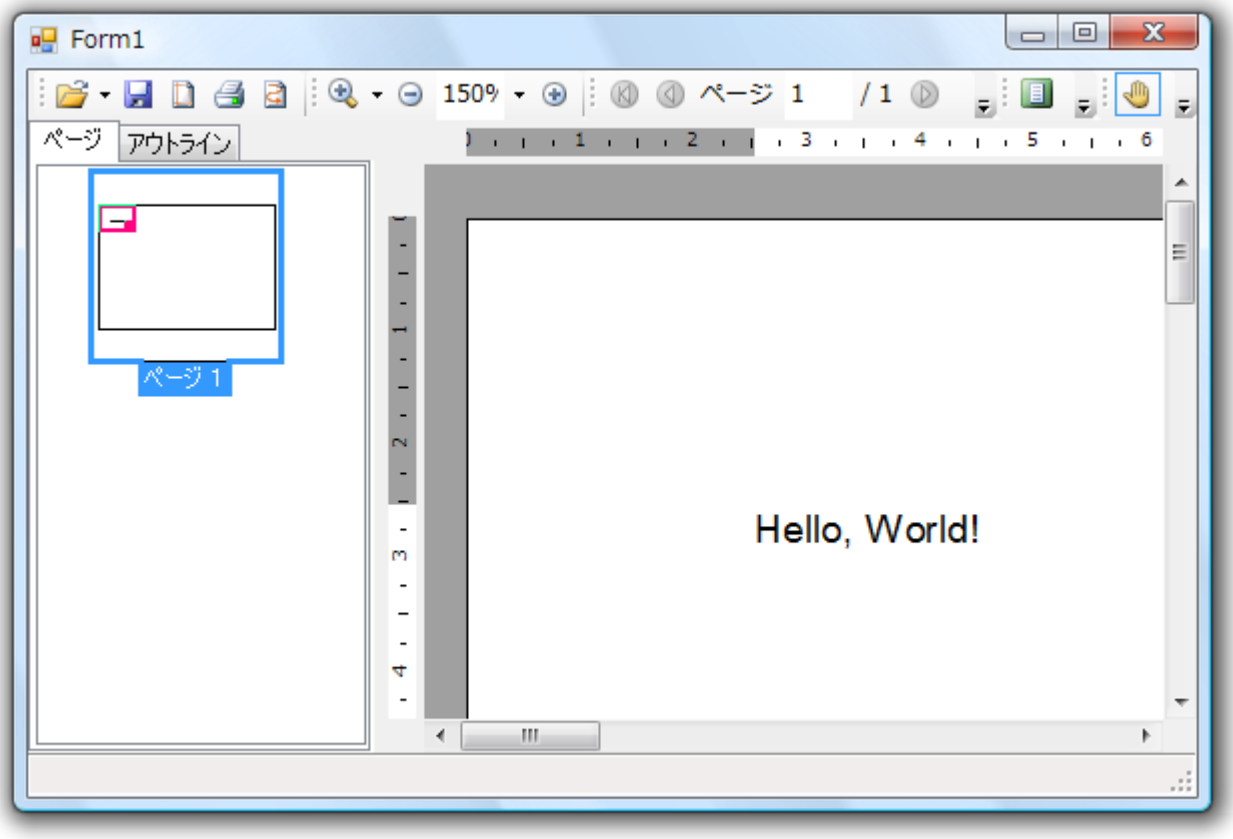

プレビューには、左上隅に「Hello World!」という文字を含む1つのページから成るドキュメントが表示されます。

2. ドキュメントのテキストが判別しやすくなるまで、[**拡大**]ボタンを繰り返しクリックします。


Copyright © GrapeCity, Inc. All rights reserved.

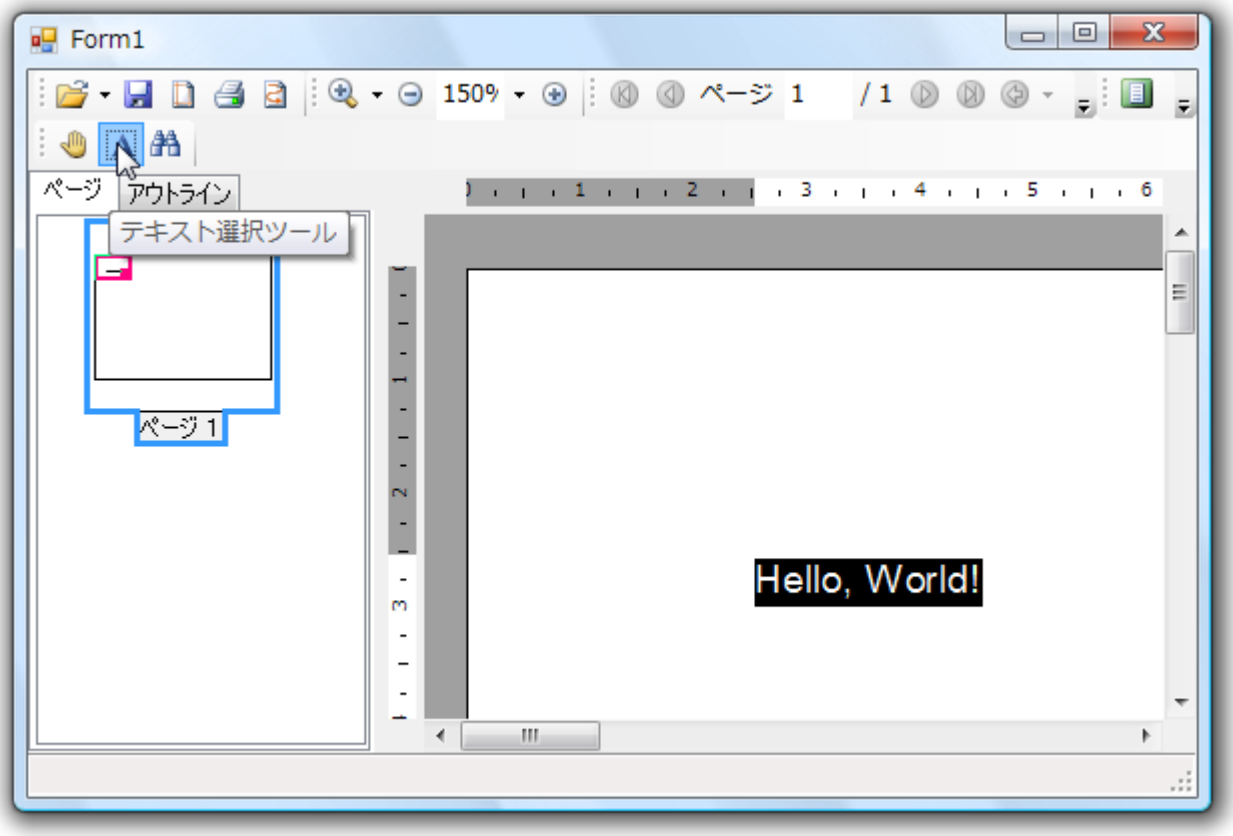



3. [**ページ設定**]ボタンをクリックして、[**ページ設定**]ダイアログボックスを開きます。[**ページ設定**]ダイアログボックスの[**向き**]で[横]を選択し、[**OK**]をクリックしてドキュメントに横方向を適用します。

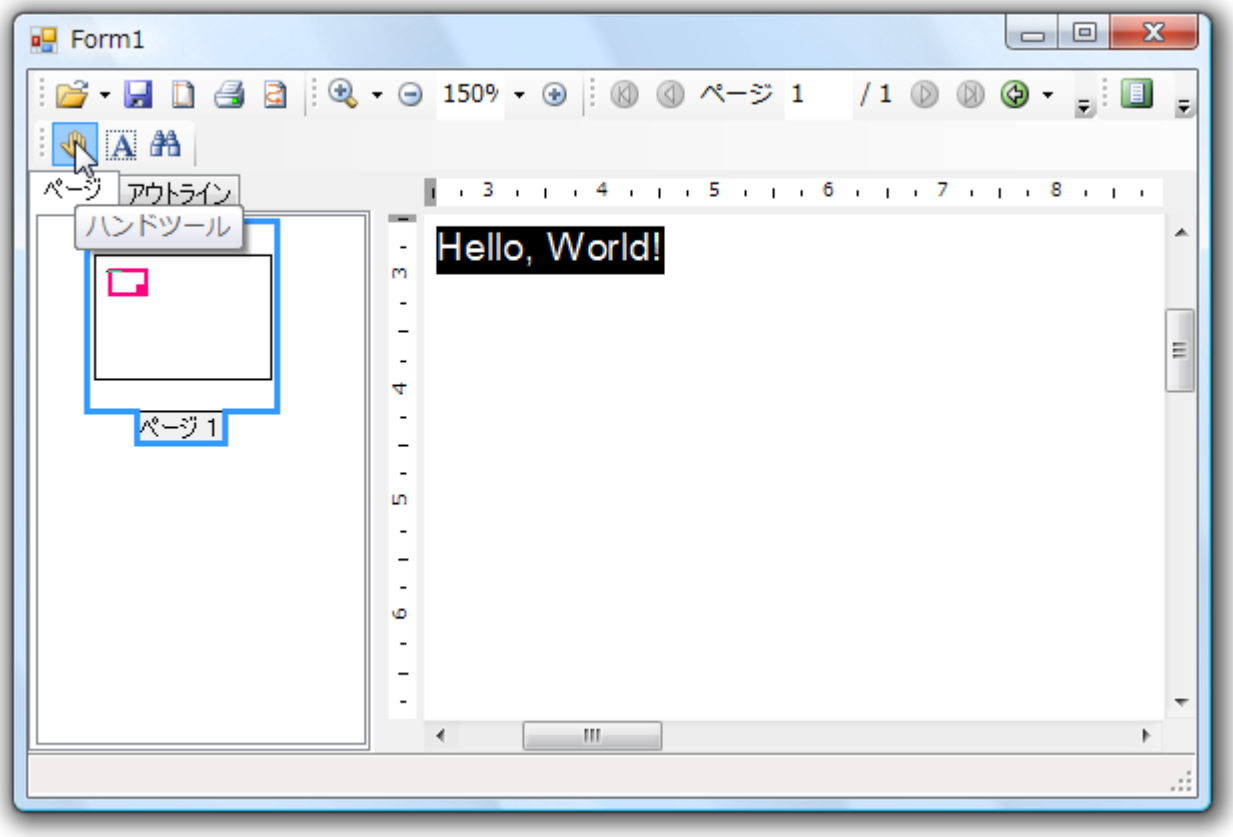



4. [**テキスト選択**]ボタンをクリックして、ドキュメントのテキストを強調表示します。


Copyright © GrapeCity, Inc. All rights reserved.




5. [**ハンドツール**]ボタンをクリックし、ドキュメントの本文でドラッグアンドドロップ操作を実行して、テキストを別の場所に表示します。




6. [**保存**]または[印刷]をクリックすると別のダイアログボックスが開き、さらに他のプレビュー機能を試してみることができます。


Copyright © GrapeCity, Inc. All rights reserved.

これで、「Hello, World!」ドキュメントが作成され、C1Preview クイックスタートガイドが完了しました。プロジェクトを保存します。この後のチュートリアルでは、さらにプロジェクトに追加していきます。

# テーブルを作成する

このトピックでは、簡単なテーブルおよび3行3列のテーブルを作成する方法、テーブルのセルにテキストを追加する方法、テーブルの特定のセルに画像を追加する方法、ドキュメントのテーブルの下にテキストを追加する方法、テーブルの行と列の周囲に境界線を作成する方法、テーブルの特定のセルに背景色を作成する方法について説明します。

**メモ**:次のトピックでは、「 クイックスタート」トピックで作成したサンプルアプリケーションをベースとして使用します。

# 単純なテーブルを作成する

テーブルは、ドキュメント内の最も便利な機能の1つです。テーブルは、データを一覧表で表したり、ドキュメント内の他の要素をレイアウトするために使用できます。**C1PrintDocument** は、完全な機能を備えたテーブルを提供します。このセクションでは、テーブルの使用方法を説明します。ここでは、前のセクションで作成した「Hello, World!」サンプルアプリケーションを使用し、そのアプリケーションにテーブルを追加します。

**メモ**:このトピック内のサンプルコードは、「using C1.C1Preview」ディレクティブ(C# の場合。他の言語の場合は相当する構文)がファイルに挿入されていることを前提としています。したがって、完全修飾された型名(C1.C1Preview.RenderText など)ではなく、クラス名部分(RenderText)だけを使用します。

1. 前のセクションで作成した **HelloWorld** アプリケーションを開きます。または、新しい Windows Forms アプリケーションを作成し、フォームに C1PrintPreviewControl とC1PrintDocumentを追加して、前のセクションで説明したようにドキュメントをプレビューコントロールにアタッチすることもできます。
2. **Form Load** イベントハンドラ(必要に応じて作成)で、c1PrintDocument1.**Generate** () の呼び出しの前に、次のコードを追加します。

**Visual Basic コードの書き方**

Visual Basic

```
Dim rt As New RenderTable()
Me.C1PrintDocument1.Body.Children.Add(rt)

Dim row As Integer = 0
Do While (row < 10)
    Dim col As Integer = 0
    Do While (col < 6)
        rt.Cells.Item(row, col).Text = String.Format("Cell ({0},{1})", row, col)
        col += 1
    Loop
    row += 1
Loop
```

**C# コードの書き方**

C#

```
RenderTable rt = new RenderTable();
this.c1PrintDocument1.Body.Children.Add(rt);

for (int row = 0; row < 10; ++ row)
{
    for (int col = 0; col < 6; ++ col)
```


Copyright © GrapeCity, Inc. All rights reserved.

```
    {
        rt.Cells[row, col].Text = string.Format("Cell ({0},{1})", row, col);
    }
}
```

3. ドキュメントの **Generate** メソッドを忘れずに呼び出します。

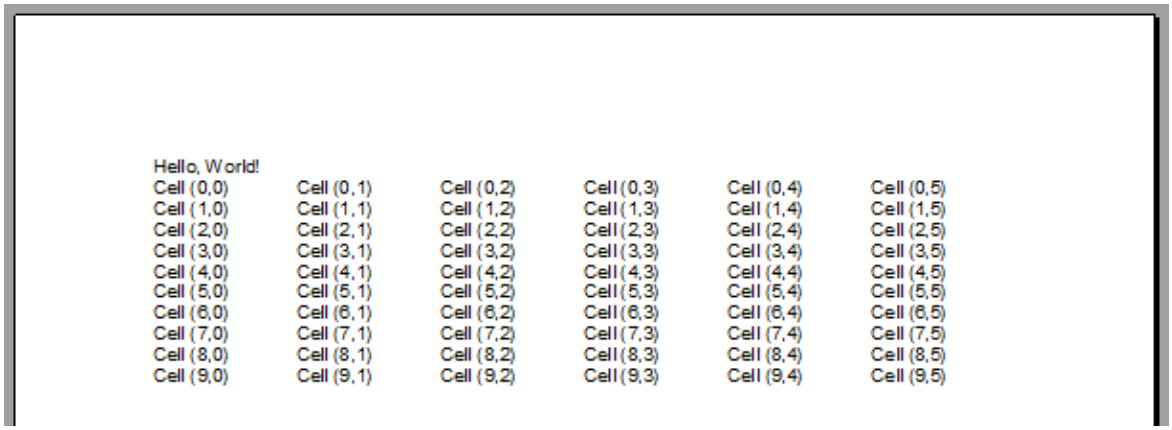
**Visual Basic コードの書き方**

```
Visual Basic
Me.C1PrintDocument1.Generate()
```

**C# コードの書き方**

```
C#
this.c1PrintDocument1.Generate();
```

アプリケーションをビルドし、実行します。プレビューには、次の画像に示すようなドキュメントが表示されます。




この単純なサンプルには、C1PrintDocument でテーブルを使用するための重要な点が含まれています。

- テーブルは、**RenderObject** から継承される **RenderTable** クラスによって表されます。
- テーブルは、Microsoft Excel で使用されるモデルに準拠しています。テーブルのサイズには初期の制限がありません。テーブルがレンダリングされるときの実際のテーブルのサイズは、内容が設定されている最大の行番号と列番号のセルによって決定されます。このサンプルのテーブルは 10 行6列です。設定されているセルの最大の行および列インデックスが(9,5)だからです(インデックスは0から始まる)。たとえば、コードを修正して、(10,7)の位置にあるセルのテキストを設定すると、テーブルは 11 行8列まで拡大されます。

**Visual Basic コードの書き方**

```
Visual Basic
rt.Cells(10, 7).Text = "text at row 10, column 7"
```

**C# コードの書き方**

```
C#
rt.Cells[10, 7].Text = "text at row 10, column 7";
```

- デフォルトでは、テーブルにグリッド線は表示されません。なお、C1Preview で「グリッド線」という用語は、描画オブジェクトの周囲に描画できる境界線ではなく、テーブルを描画するために使用される線です。グリッド線を追加するには(0.5pt の黒のペンで描画)、**Form Load** イベントハンドラに次のコードを追加します。

**Visual Basic コードの書き方**

```
Visual Basic
```


Copyright © GrapeCity, Inc. All rights reserved.

```
rt.Style.GridLines.All = LineDef.Default
```

**C# コードの書き方**

C#

```
rt.Style.GridLines.All = LineDef.Default;
```

デフォルトでは、テーブルは、その親のクライアント領域と同じ幅になり(このサンプルでは、ページ全体)、すべての列の幅が同じになります。ただし、行の高さは、自動的に設定されます。したがって、テーブル内の任意のセルのテキストを長いテキストに設定するコードを追加すると、そのセルを含む行は、すべてのテキストが入るように縦に広げられます。たとえば、次のコードをサンプルに追加します。

**Visual Basic コードの書き方**

Visual Basic

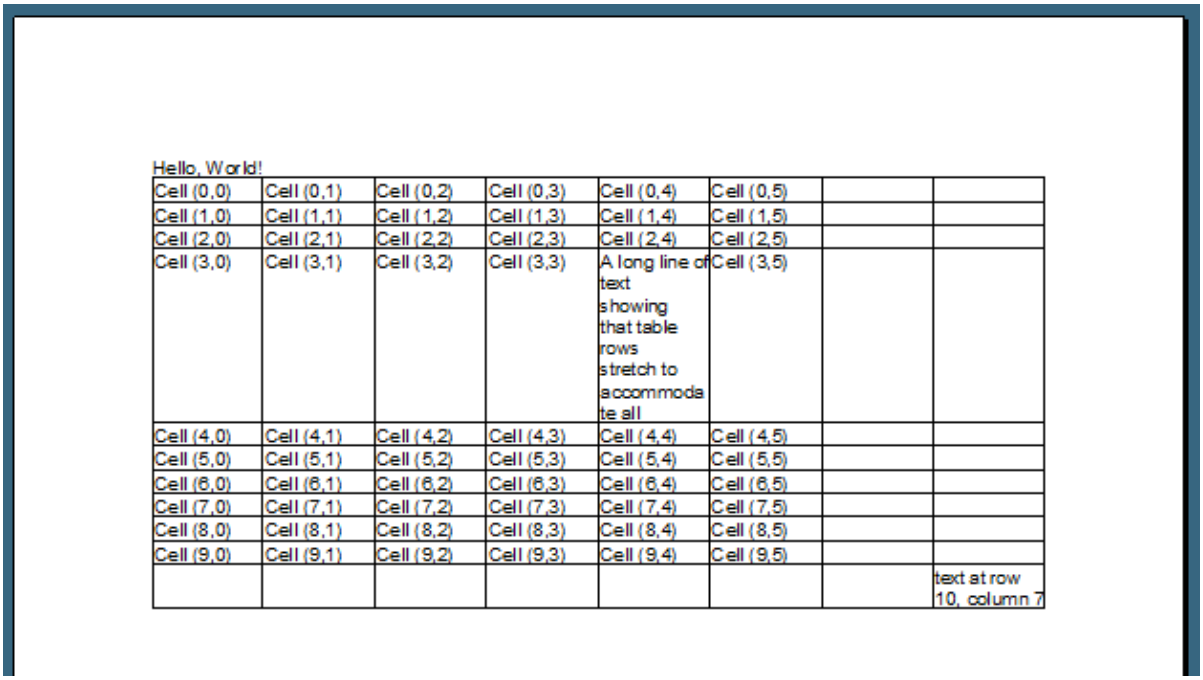
```
rt.Cells(3, 4).Text = "A long line of text showing that table rows stretch " + "to
accommodate all content."
```

**C# コードの書き方**

C#

```
rt.Cells[3, 4].Text = "A long line of text showing that table rows stretch " + "to
accommodate all content.";
```

次のようなテーブルが生成されます。このテーブルには、これまでに説明した2つの変更が含まれています。




参考までに、上のドキュメントを生成したロードイベントハンドラの完全なコードを次に示します。

**Visual Basic コードの書き方**

Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    Me.C1PrintDocument1.Body.Children.Add(New RenderText("Hello, World!"))
    Dim rt As New RenderTable()
```


Copyright © GrapeCity, Inc. All rights reserved.

```
    Me.C1PrintDocument1.Body.Children.Add(rt)

Dim row As Integer = 0
Do While (row < 10)
    Dim col As Integer = 0
    Do While (col < 6)
        rt.Cells.Item(row, col).Text = String.Format("Cell ({0},{1})", row, col)
        col += 1
    Loop
    row += 1
Loop
    rt.Cells(3, 4).Text = "A long line of text showing that table rows " + "stretch to
accommodate all content."
    rt.Cells(10, 7).Text = "text at row 10, column 7"
    rt.Style.GridLines.All = LineDef.Default
    Me.C1PrintDocument1.Generate()
End Sub
```

**C# コードの書き方**

C#

```
private void Form1_Load(object sender, EventArgs e)
{
    this.c1PrintDocument1.Body.Children.Add(new RenderText("Hello, World!"));
    RenderTable rt = new RenderTable();
    this.c1PrintDocument1.Body.Children.Add(rt);
    for (int row = 0; row < 10; ++row)
    {
        for (int col = 0; col < 6; ++col)
        {
            rt.Cells[row, col].Text = string.Format("Cell ({0},{1})", row, col);
        }
    }
    rt.Cells[3, 4].Text = "A long line of text showing that table rows " + "stretch to
accommodate all content.";
    rt.Cells[10, 7].Text = "text at row 10, column 7";
    rt.Style.GridLines.All = LineDef.Default;
    this.c1PrintDocument1.Generate();
}
```

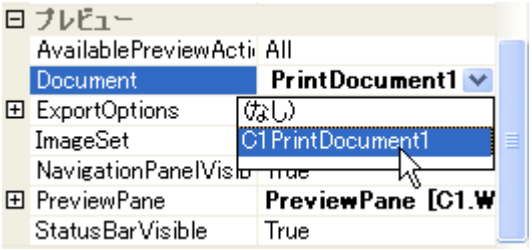
# 3行3列のテーブルの作成

**C1PrintDocument** では、**RenderTable** クラスを使ってテーブルを作成できます。

このトピックは、3行3列のテーブルを設定するための基本を説明します。

1. 最初に、ドキュメントの生成とプレビューを行うサンプルの基本フレームワークを設定します。新しい .NET Windows アプリケーションプロジェクトを作成します。**C1PrintPreviewコントロール** と**C1PrintDocument** コンポーネントをフォームにドロップします。
2. C1PrintPreview の **Dock** プロパティを **Fill** に設定します（プレビューがフォーム内で唯一のコントロールになる）。次の画像に示すように、プレビューの **Document** プロパティをC1PrintDocumentに設定すると、C1PrintPreview に**C1PrintDocument**が表示されます。


Copyright © GrapeCity, Inc. All rights reserved.




3. 空のフォームをダブルクリックします。これにより、ソースコードにフォームロードイベントの新しいハンドラが自動的に作成されます。このハンドラは次のようになります。
4. 次に、新しい C1.C1PrintDocument.RenderTable オブジェクトを作成し、次のコードを使用して、これを変数に代入します。

**Visual Basic コードの書き方**

```
Visual Basic
Dim table As C1.C1Preview.RenderTable = New C1.C1Preview.RenderTable(Me.C1PrintDocument1)
```

**C# コードの書き方**

```
C#
C1.C1Preview.RenderTable table = new C1.C1Preview.RenderTable(this.c1PrintDocument1);
```

**メモ**:上のコードは、Form1_Load イベントの左カッコと右カッコの間に置く必要があります。

5. テーブルに3列を追加し、テーブルの本体に3行を追加します。

**Visual Basic コードの書き方**

```
Visual Basic
' 3行を追加します
Dim r As Integer = 3

' 3列を追加します
Dim c As Integer = 3

Dim row As Integer
Dim col As Integer

For row = 0 To r - 1 Step +1
    For col = 0 To c - 1 Step +1
        Dim celltext As C1.C1Preview.RenderText = New
C1.C1Preview.RenderText(Me.C1PrintDocument1)

        ' 空のセルを追加します
        celltext.Text = String.Format("", row, col)
        table.Cells(row, col).RenderObject = celltext
    Next
Next
```

**C# コードの書き方**

```
C#
// 3行を追加します
```


Copyright © GrapeCity, Inc. All rights reserved.

```
const int r = 3;

// 3列を追加します
const int c = 3;

for (int row = 0; row < r; ++row)
{
    for (int col = 0; col < c; ++col)
    {
        C1.C1Preview.RenderText celltext = new
C1.C1Preview.RenderText(this.c1PrintDocument1);
        celltext.Text = string.Format("", row, col);

        // 空のセルを追加します
        table.Cells[row, col].RenderObject = celltext;
    }
}
```

**メモ**:上のコードは、手順3で入力したコードの後に置く必要があります。

列はテーブルに直接追加したのに対して、行はテーブルの本体に追加しています。これは、**RenderTable** が常に3つの"テーブル領域"(**ヘッダー、本体、フッター**)で構成されるためです。テーブルの3つのテーブル領域は空でもかまいません。単純なテーブルを作成する場合は、このサンプルのように本体に行を追加できます。

6. 列の幅と行の高さを5センチにします。

**Visual Basic コードの書き方**

Visual Basic

```
table.Height = New C1.C1Preview.Unit(15, C1.C1Preview.UnitTypeEnum.Cm)
table.Width = New C1.C1Preview.Unit(15, C1.C1Preview.UnitTypeEnum.Cm)
```

**C# コードの書き方**

C#

```
table.Height = new C1.C1Preview.Unit(15, C1.C1Preview.UnitTypeEnum.Cm);
table.Width = new C1.C1Preview.Unit(15, C1.C1Preview.UnitTypeEnum.Cm);
```

7. デフォルトでは、テーブルには境界線がありません。テーブルに濃い灰色のグリッド線を追加します。

**Visual Basic コードの書き方**

Visual Basic

```
table.Style.GridLines.All = New C1.C1Preview.LineDef(Color.DarkGray)
```

**C# コードの書き方**

C#

```
table.Style.GridLines.All = new C1.C1Preview.LineDef(Color.DarkGray);
```

8. 描画オブジェクトを作成したら、それをドキュメントに追加する必要があります。それには、最初にドキュメントの **Add** メソッドを呼び出して、テーブルをドキュメントの本体に追加します。次に、**Generate** メソッドを使用して、ドキュメントを作成します。このコードを次に示します。


Copyright © GrapeCity, Inc. All rights reserved.

**Visual Basic コードの書き方**

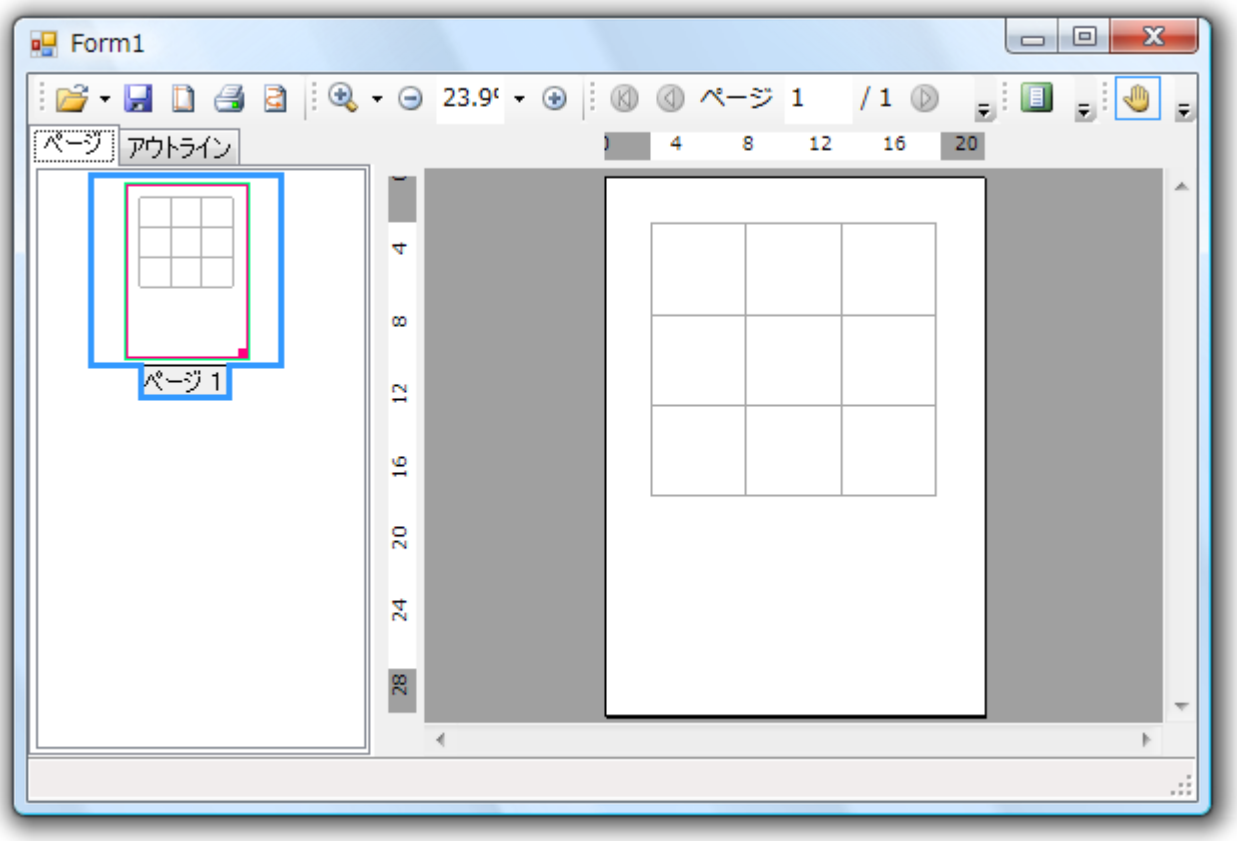
```
Visual Basic
Me.C1PrintDocument1.Body.Children.Add(table)
Me.C1PrintDocument1.Generate()
```

**C# コードの書き方**

```
C#
this.c1PrintDocument1.Body.Children.Add(table);
this.c1PrintDocument1.Generate();
```

## アプリケーションを保存し、実行します。

実行時に、アプリケーションは次の画像のように表示されます。




## テーブルセルへのテキストの追加

このトピックは、**RenderText** クラスを使用してテーブルの特定のセルにテキストを追加する方法を示します。

1. 濃い灰色のグリッド線を使用するテーブルを設定する次のコードがソースファイルに既にある必要があります。

**Visual Basic コードの書き方**

```
Visual Basic
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
```


Copyright © GrapeCity, Inc. All rights reserved.

```
System.EventArgs) Handles MyBase.Load

    ' テーブルを作成します。
    Dim table As C1.C1Preview.RenderTable = New
C1.C1Preview.RenderTable(Me.C1PrintDocument1)
    table.Style.GridLines.All = New C1.C1Preview.LineDef(Color.DarkGray)

    ' ドキュメントを生成します。
    Me.C1PrintDocument1.Body.Children.Add(table)
    Me.C1PrintDocument1.Generate()

End Sub
```

**C# コードの書き方**

C#

```
private void Form1_Load(object sender, System.EventArgs e)
{
    // テーブルを作成します。
    C1.C1Preview.RenderTable table = new
C1.C1Preview.RenderTable(this.c1PrintDocument1);
    table.Style.GridLines.All = new C1.C1Preview.LineDef(Color.DarkGray);

    // ドキュメントを生成します。
    this.c1PrintDocument1.Body.Children.Add(table);
    this.c1PrintDocument1.Generate();

}
```

2. テーブルセルに任意の種類のコンテンツを表示するには、そのコンテンツを表すレンダリングオブジェクトをセルの **RenderObject** プロパティに割り当てます。ただし、テーブルセルにテキストを表示することはよく行うタスクなので、セルには専用のプロパティ RenderText が用意されています。ここでは、これを使用します。テーブル内のすべてのセルのテキストを設定するには、テーブル内の各行をループし、そのループ内で行内の各列をさらにループします。ネストしたループの本体で、次のように、**Text** プロパティを目的のテキストに設定します。このサンプルでは、セル(1,1)と(1,2)は空にしています。

**Visual Basic コードの書き方**

Visual Basic

```
' 3行を追加します。
Dim r As Integer = 3

' 3列を追加します。
Dim c As Integer = 3

Dim row As Integer
Dim col As Integer

For row = 0 To r - 1 Step +1
    For col = 0 To c - 1 Step +1
        If (Not (row = 1 And col = 1)) And (Not (row = 1 And col = 2)) Then
           Dim celltext As C1.C1Preview.RenderText = New
C1.C1Preview.RenderText(Me.C1PrintDocument1)
```


Copyright © GrapeCity, Inc. All rights reserved.

```
            celltext.Text = String.Format("Cell ({0},{1})", row, col)

            ' テキストを含むセルを追加します。
            table.Cells(row, col).RenderObject = celltext
        End If
    Next
Next
```

**C# コードの書き方**

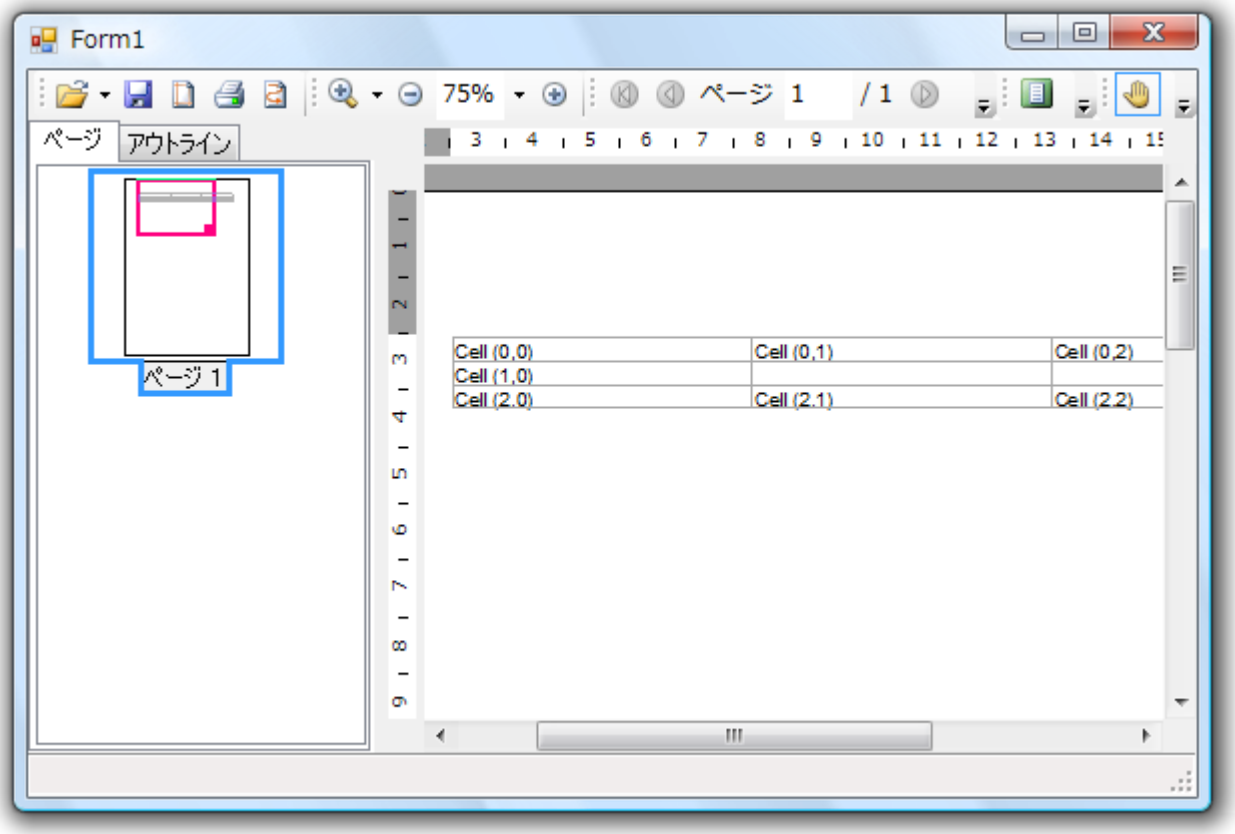
C#

```
// 3行を追加します。
const int r = 3;

// 3列を追加します。
const int c = 3;

for (int row = 0; row < r; ++row)
{
    for (int col = 0; col < c; ++col)
    {
        if (!(row == 1 && col == 1) && !(row == 1 && col == 2))
        {
            C1.C1Preview.RenderText celltext = new
C1.C1Preview.RenderText(this.c1PrintDocument1);
            celltext.Text = string.Format("Cell ({0}, {1})", row, col);

            // テキストを含むセルを追加します。
            table.Cells[row, col].RenderObject = celltext;
        }
    }
}
```

## プログラムを実行し、次の点を確認します。

テーブルは次のように表示されます。


Copyright © GrapeCity, Inc. All rights reserved.




# テーブルの特定のセルへの2つの画像の追加

このトピックは、**RenderImage** クラスを使用して既存のテーブルの特定のセルに2つの異なる画像を追加する方法を示します。また、**ImageAlignHorzEnum**を使用してセル内の画像を配置する方法も示します。下のサンプルは、「3行3列のテーブルの作成」で作成した3行3列の空のテーブルを使用します。また、このトピックの手順を完了するには、2つの GIF または JPEG 画像が必要です。次の手順を実行します。

1. 次のコードが既にソースファイルにある必要があります。

**Visual Basic コードの書き方**

Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load

    ' テーブルを作成します。
    Dim table As C1.C1Preview.RenderTable = New
C1.C1Preview.RenderTable(Me.C1PrintDocument1)
    table.Style.GridLines.All = New C1.C1Preview.LineDef(Color.DarkGray)

    Dim r As Integer = 3
    Dim c As Integer = 3
    Dim row As Integer
    Dim col As Integer
    For row = 0 To r - 1 Step +1
        For col = 0 To c - 1 Step +1
            Dim celltext As C1.C1Preview.RenderText = New
C1.C1Preview.RenderText(Me.C1PrintDocument1)
```


Copyright © GrapeCity, Inc. All rights reserved.

```
            ' 空のセルを追加します。
            celltext.Text = String.Format("", row, col)
            table.Cells(row, col).RenderObject = celltext
        Next
    Next

    ' ドキュメントを生成します。
    Me.C1PrintDocument1.Body.Children.Add(table)
    Me.C1PrintDocument1.Generate()
End Sub
```

**C# コードの書き方**

C#

```
private void Form1_Load(object sender, System.EventArgs e)
{
    // テーブルを作成します。
    C1.C1Preview.RenderTable table = new
C1.C1Preview.RenderTable(this.c1PrintDocument1);
    table.Style.GridLines.All = new C1.C1Preview.LineDef(Color.DarkGray);

    const int r = 3;
    const int c = 3;
    for (int row = 0; row < r; ++row)
    {
        for (int col = 0; col < c; ++col)
        {
            C1.C1Preview.RenderText celltext = new
C1.C1Preview.RenderText(this.c1PrintDocument1);
            celltext.Text = string.Format("", row, col);

            // 空のセルを追加します。
            table.Cells[row, col].RenderObject = celltext;
        }
    }

    // ドキュメントを生成します。
    this.c1PrintDocument1.Body.Children.Add(table);
    this.c1PrintDocument1.Generate();
}
```

2. 行を追加するコード行の後に次のコードを追加します。新しいコードは、テーブル内の中央のセルのサイズを固定します。

**Visual Basic コードの書き方**

Visual Basic

```
' 中央のセルのサイズを固定します。
table.Rows(1).Height = New C1.C1Preview.Unit(5, C1.C1Preview.UnitTypeEnum.Cm)
table.Cols(1).Width = New C1.C1Preview.Unit(8, C1.C1Preview.UnitTypeEnum.Cm)
```

**C# コードの書き方**

C#


Copyright © GrapeCity, Inc. All rights reserved.

```
// 中央のセルのサイズを固定します。
table.Rows[1].Height = new C1.C1Preview.Unit(5, C1.C1Preview.UnitTypeEnum.Cm);
table.Cols[1].Width = new C1.C1Preview.Unit(8, C1.C1Preview.UnitTypeEnum.Cm);
```

3. 2つの JPEG または GIF 画像を作成するか、既にある画像を使用します。
4. フォームに2つの **PictureBox** コントロールを追加します。**Image** プロパティを前のステップで作成した2つの画像に設定します。また、この2つの画像ボックスを非表示にして（**Visible** を **False** に設定）、フォームの外観を整理します。これらのコントロールは画像の保管場所としてのみ使用されます。画像は **C1PrintDocument**にレンダリングされます。
5. **CellStyle** プロパティを使用して、セルのコンテンツの基本スタイルを変更します。このサンプルでは、セルの **ImageAlign** プロパティを変更します。次のコードを入力して、画像の配置を設定します。

**Visual Basic コードの書き方**

Visual Basic

```
' 画像の配置を設定します。
table.CellStyle.ImageAlign.StretchHorz = False
table.CellStyle.ImageAlign.StretchVert = False
table.CellStyle.ImageAlign.AlignHorz = C1.C1Preview.ImageAlignHorzEnum.Center
```

**C# コードの書き方**

C#

```
// 画像の配置を設定します。
table.CellStyle.ImageAlign.StretchHorz = false;
table.CellStyle.ImageAlign.StretchVert = false;
table.CellStyle.ImageAlign.AlignHorz = C1.C1Preview.ImageAlignHorzEnum.Center;
```

6. C1PrintDocument では、画像が RenderImage クラス（**RenderObject** のサブクラス）を使用してレンダリングされます。次のように、2つの画像に対応する2つの新しい **RenderImage** オブジェクトを作成します。

**Visual Basic コードの書き方**

Visual Basic

```
Dim img1 As C1.C1Preview.RenderImage = New
C1.C1Preview.RenderImage(Me.C1PrintDocument1)
Dim img2 As C1.C1Preview.RenderImage = New
C1.C1Preview.RenderImage(Me.C1PrintDocument1)
```

**C# コードの書き方**

C#

```
C1.C1Preview.RenderImage img1 = new
C1.C1Preview.RenderImage(this.c1PrintDocument1);
C1.C1Preview.RenderImage img2 = new
C1.C1Preview.RenderImage(this.c1PrintDocument1);
```

7. これらの RenderImage の **Image** プロパティを画像ボックスに格納した画像に設定します。

**Visual Basic コードの書き方**

Visual Basic

```
img1.Image = Me.PictureBox1.Image
img2.Image = Me.PictureBox2.Image
```

**C# コードの書き方**


Copyright © GrapeCity, Inc. All rights reserved.

```
C#
img1.Image = this.pictureBox1.Image;
img2.Image = this.pictureBox2.Image;
```

8. これらの RenderImage オブジェクトをセルの RenderObject プロパティに割り当てて、それらのセルに画像がレンダリングされるようにします。

**Visual Basic コードの書き方**

```
Visual Basic
table.Cells(1, 1).RenderObject = img1
table.Cells(1, 2).RenderObject = img2
```

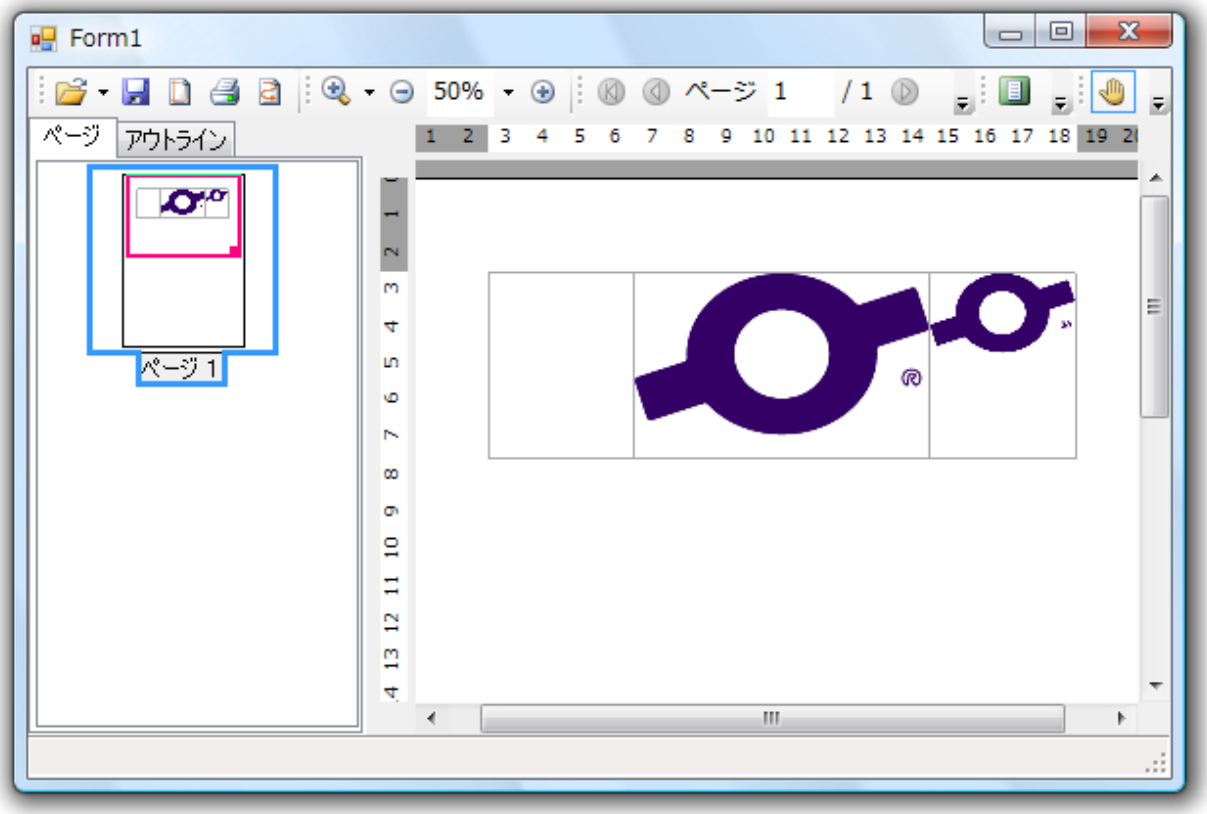
**C# コードの書き方**

```
C#
table.Cells[1, 1].RenderObject = img1;
table.Cells[1, 2].RenderObject = img2;
```

**メモ**:テーブルの左上のセルは行0、列0の位置にあります。

## プログラムを実行し、次の点を確認します。

テーブルは次のように表示されます。




## テーブル内の行と列の周囲の境界線の作成

このトピックは、**LineDef** クラスを使用して行と列の周囲にはっきりとした境界線を作成する方法を示します。このトピックは、3


Copyright © GrapeCity, Inc. All rights reserved.

行3列のテーブルが既にあることを前提としています。

1. 次のコードが既にソースファイルにある必要があります。

**Visual Basic コードの書き方**

Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    ' テーブルを作成します。
    Dim table As C1.C1Preview.RenderTable = New
C1.C1Preview.RenderTable(Me.C1PrintDocument1)
    table.Style.GridLines.All = New C1.C1Preview.LineDef(Color.DarkGray)

    Dim r As Integer = 3
    Dim c As Integer = 3
    Dim row As Integer
    Dim col As Integer
    For row = 0 To r - 1 Step +1
        For col = 0 To c - 1 Step +1
            Dim celltext As C1.C1Preview.RenderText = New
C1.C1Preview.RenderText(Me.C1PrintDocument1)

            ' 空のセルを追加します。
            celltext.Text = String.Format("", row, col)
            table.Cells(row, col).RenderObject = celltext
        Next
    Next

    ' ドキュメントを生成します。
    Me.C1PrintDocument1.Body.Children.Add(table)
    Me.C1PrintDocument1.Generate()
End Sub
```

**C# コードの書き方**

C#

```
private void Form1_Load(object sender, System.EventArgs e)
{
        // テーブルを作成します。
    C1.C1Preview.RenderTable table = new
C1.C1Preview.RenderTable(this.c1PrintDocument1);
    table.Style.GridLines.All = new C1.C1Preview.LineDef(Color.DarkGray);

    const int r = 3;
    const int c = 3;
    for (int row = 0; row < r; ++row)
    {
        for (int col = 0; col < c; ++col)
        {
        C1.C1Preview.RenderText celltext = new
```


Copyright © GrapeCity, Inc. All rights reserved.

```
C1.C1Preview.RenderText(this.c1PrintDocument1);
            celltext.Text = string.Format("", row, col);

            // 空のセルを追加します。
            table.Cells[row, col].RenderObject = celltext;
        }
    }

    // ドキュメントを生成します。
    this.c1PrintDocument1.Body.Children.Add(table);
    this.c1PrintDocument1.Generate();
}
```

2. プロジェクトに次のコードを追加して、テーブルの幅と高さを 15cm にします。

**Visual Basic コードの書き方**

Visual Basic

```
table.Height = New C1.C1Preview.Unit(15, C1.C1Preview.UnitTypeEnum.Cm)
table.Width = New C1.C1Preview.Unit(15, C1.C1Preview.UnitTypeEnum.Cm)
```

**C# コードの書き方**

C#

```
table.Height = new C1.C1Preview.Unit(15, C1.C1Preview.UnitTypeEnum.Cm);
table.Width = new C1.C1Preview.Unit(15, C1.C1Preview.UnitTypeEnum.Cm);
```

3. 次のように、プロジェクトに次のコードを追加して、LineDef クラスの新しいインスタンスを3行目の境界線に割り当てます。使用するコンストラクタでは、新しい境界線を2ポイント幅の赤色に指定します。

**Visual Basic コードの書き方**

Visual Basic

```
table.Rows(2).Style.Borders.All = New C1.C1Preview.LineDef("2pt", Color.Red)
```

**C# コードの書き方**

C#

```
table.Rows[2].Style.Borders.All = new C1.C1Preview.LineDef("2pt", Color.Red);
```

4. 次のように、LineDef クラスの新しいインスタンスを最初の列の境界線に割り当てます。使用するコンストラクタでは、新しい境界線を6ポイント幅の青色に指定します。

**Visual Basic コードの書き方**

Visual Basic

```
table.Cols(0).Style.Borders.All = New C1.C1Preview.LineDef("6pt", Color.Blue)
```

**C# コードの書き方**

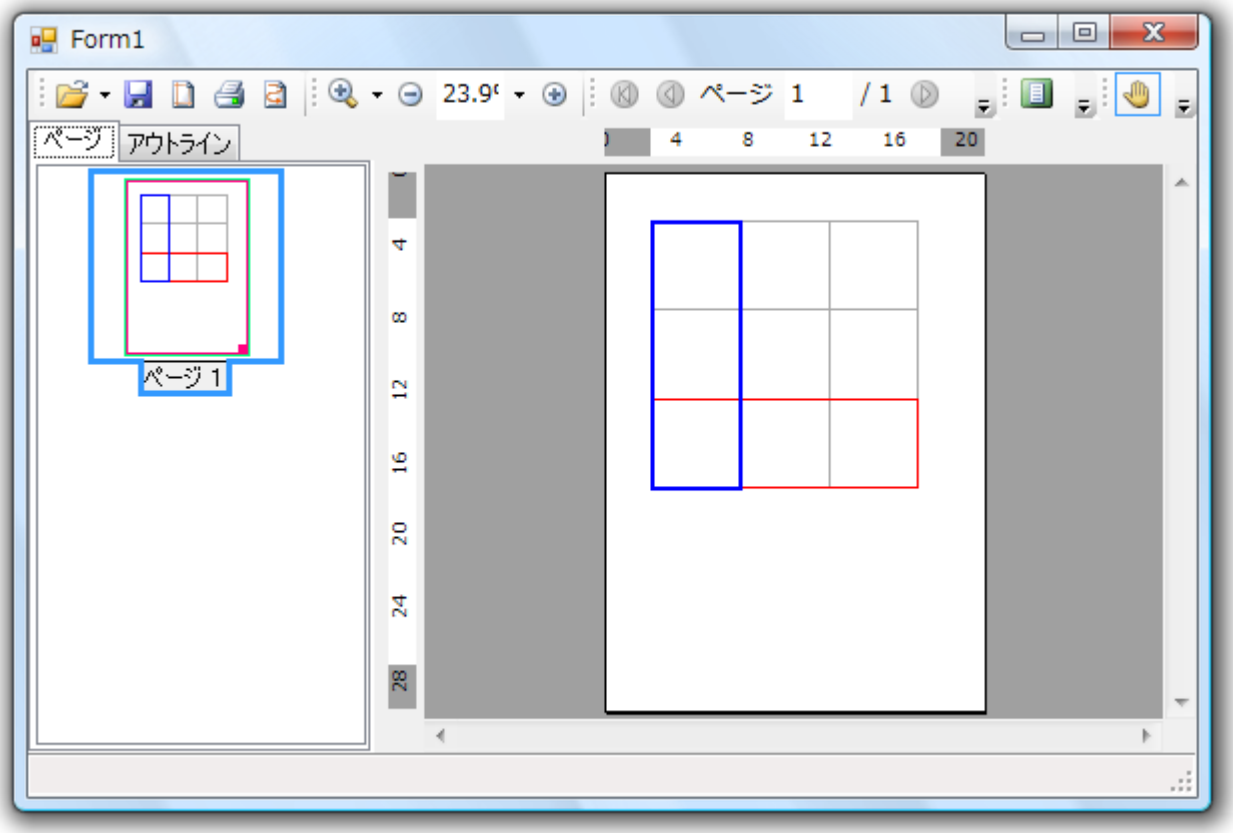
C#

```
table.Cols[0].Style.Borders.All = new C1.C1Preview.LineDef("6pt", Color.Blue);
```


Copyright © GrapeCity, Inc. All rights reserved.

## プログラムを実行し、次の点を確認します。

実行時に、境界線は次のテーブルのように表示されます。




# テーブル内の特定のセルの背景色の作成

このトピックは、テーブル内の特定のセルの背景色を作成する方法を示します。また、**CellStyle** プロパティを使用して、レンダリングされるテーブルで使用するスタイルを設定する方法も示します。このトピックは、3行3列のテーブルがあることを前提としています。

1. 次のコードが既にソースファイルにある必要があります。

**Visual Basic コードの書き方**

Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    ' テーブルを作成します。
    Dim table As C1.C1Preview.RenderTable = New
C1.C1Preview.RenderTable(Me.C1PrintDocument1)
    table.Style.GridLines.All = New C1.C1Preview.LineDef(Color.DarkGray)

    Dim r As Integer = 3
    Dim c As Integer = 3
    Dim row As Integer
    Dim col As Integer
```


Copyright © GrapeCity, Inc. All rights reserved.

```
    For row = 0 To r - 1 Step +1
        For col = 0 To c - 1 Step +1
            Dim celltext As C1.C1Preview.RenderText = New
C1.C1Preview.RenderText(Me.C1PrintDocument1)

            ' 空のセルを追加します。
            celltext.Text = String.Format("", row, col)
            table.Cells(row, col).RenderObject = celltext
        Next
    Next

    ' ドキュメントを生成します。
    Me.C1PrintDocument1.Body.Children.Add(table)
    Me.C1PrintDocument1.Generate()
End Sub
```

**C# コードの書き方**

C#

```
private void Form1_Load(object sender, System.EventArgs e)
{
    // テーブルを作成します。
    C1.C1Preview.RenderTable table = new
C1.C1Preview.RenderTable(this.c1PrintDocument1);
    table.Style.GridLines.All = new C1.C1Preview.LineDef(Color.DarkGray);

    const int r = 3;
    const int c = 3;
    for (int row = 0; row < r; ++row)
    {
        for (int col = 0; col < c; ++col)
        {
            C1.C1Preview.RenderText celltext = new
C1.C1Preview.RenderText(this.c1PrintDocument1);
            celltext.Text = string.Format("", row, col);

            // 空のセルを追加します。
            table.Cells[row, col].RenderObject = celltext;
        }
    }

    // ドキュメントを生成します。
    this.c1PrintDocument1.Body.Children.Add(table);
    this.c1PrintDocument1.Generate();
}
```

2. テーブルの幅と高さを 15cm にします。

**Visual Basic コードの書き方**

Visual Basic

```
table.Height = New C1.C1Preview.Unit(15, C1.C1Preview.UnitTypeEnum.Cm)
table.Width = New C1.C1Preview.Unit(15, C1.C1Preview.UnitTypeEnum.Cm)
```


Copyright © GrapeCity, Inc. All rights reserved.

**C# コードの書き方**

```
C#
table.Width = new C1.C1Preview.Unit(15, C1.C1Preview.UnitTypeEnum.Cm);
```

3. 上にリストしたコードの最後の行の下に次のコードを追加します。このコードは、行1、列2の背景色を深紅色にします。

**Visual Basic コードの書き方**

```
Visual Basic
table.Cells(1, 2).CellStyle.BackColor = Color.Crimson
```

**C# コードの書き方**

```
C#
table.Cells[1, 2].CellStyle.BackColor = Color.Crimson;
```

**メモ**:行と列は0から始まります。上のコードでは、**CellStyle プロパティ**を使用してセルのスタイルを設定します。

4. 行0、列1の背景色を青紫色にします。次のコードを入力します。

**Visual Basic コードの書き方**

```
Visual Basic
table.Cells(0, 1).CellStyle.BackColor = Color.BlueViolet
```

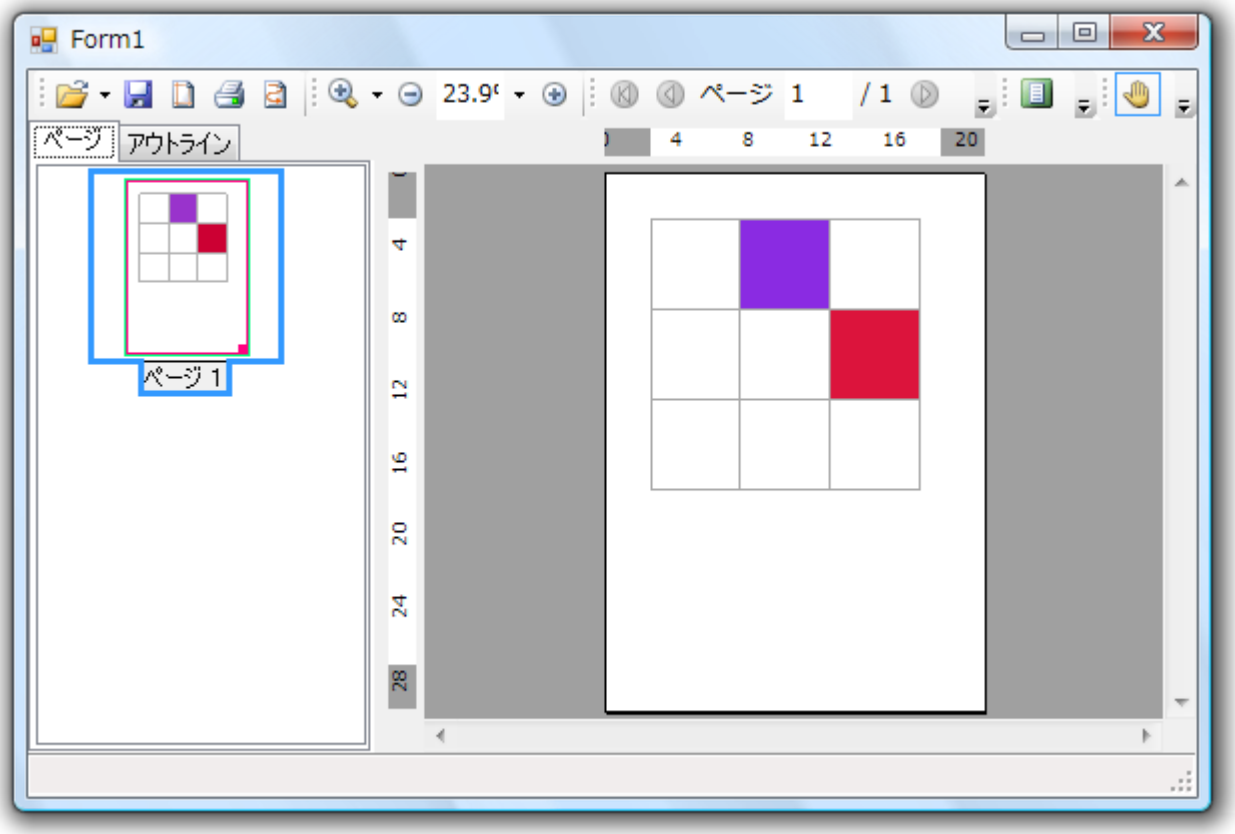
**C# コードの書き方**

```
C#
table.Cells[0, 1].CellStyle.BackColor = Color.BlueViolet;
```

## プログラムを実行し、次の点を確認します。

テーブルは次のように表示されます。


Copyright © GrapeCity, Inc. All rights reserved.




# 文字列を追加する

次のトピックでは、段落の追加方法、テーブルの下へのテキストの追加方法、テキストのフォントとスタイルの変更方法について説明します。

## ドキュメントに段落を追加する

**C1PrintDocument** のすべてのコンテンツは、描画オブジェクトによって表されます。C1Preview アセンブリは、**RenderObject** から派生されたクラスの階層を提供します。これらのクラスは、テキスト、画像などのさまざまなタイプのコンテンツを表すように設計されています。たとえば、先ほどは **RenderText**クラスを使用して、ドキュメントにテキスト行を追加しました。このセクションでは、**RenderParagraph** クラスを使用して、テキストの段落を作成する方法を説明します。段落では、複数のスタイル、インライン画像、およびハイパーリンクを使って描画される複数のテキストフラグメントを組み合わせることができます。

> **メモ**:このトピック内のサンプルコードは、「using C1.C1Preview;」ディレクティブ(C# の場合。他の言語の場合は相当する構文)がファイルに挿入されていることを前提としています。したがって、完全修飾された型名(C1.C1Preview.RenderText など)ではなく、クラス名部分(RenderText)だけを使用します。

RenderParagraph は、次のコードを使って作成できます。

**Visual Basic コードの書き方**

Visual Basic

```
Dim rp As New RenderParagraph()
```

**C# コードの書き方**

C#


Copyright © GrapeCity, Inc. All rights reserved.

```
RenderParagraph rp = new RenderParagraph();
```

次の場合は、RenderText ではなく、段落を使用する必要があります。

- 同じ段落内で複数のスタイルでテキストを表示する必要がある。
- インライン画像（通常は小さなアイコン画像）をテキストフローに挿入する必要がある。
- ハイパーリンクをテキスト全体ではなく、テキストの一部（単語など）に関連付ける必要がある（この後の「アンカーとハイパーリンク アンカーとハイパーリンク」セクションを参照）。

段落のコンテンツは、いくつかの **ParagraphObject** オブジェクトから成ります。ParagraphObject は抽象基本クラスです。継承クラスとして **ParagraphText** と **ParagraphImage** があり、これらはそれぞれテキストフラグメントとインライン画像を表します。段落にコンテンツを挿入するには、この2つの型のオブジェクトを作成し、それらを **RenderParagraph.Content** コレクションに追加します。これらのオブジェクトを作成／設定しやすいように、さまざまなオーバーロードコンストラクタとプロパティが提供されています。段落内ハイパーリンクを作成するには、ハイパーリンクにする段落オブジェクトの **Hyperlink**プロパティを指定します。別の方法では、次の例に示すように、ショートカットメソッド **AddText**、**AddImage** および **AddHyperlink**、のさまざまなオーバーロードを使用します。

**Visual Basic コードの書き方**

Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load

    ' 段落を作成します
    Dim rpar As New RenderParagraph()
    Dim f As New Font(rpar.Style.Font, FontStyle.Bold)

    rpar.Content.AddText("This is a paragraph. This is normal text. ")
    rpar.Content.AddText("This text is bold. ", f)
    rpar.Content.AddText("This text is red. ", Color.Red)
    rpar.Content.AddText("This text is superscript. ", TextPositionEnum.Superscript)
    rpar.Content.AddText("This text is bold and red. ", f, Color.Red)
    rpar.Content.AddText("This text is bold and red and subscript. ", f, Color.Red,
TextPositionEnum.Subscript)
    rpar.Content.AddText("This is normal text again. ")
    rpar.Content.AddHyperlink("This is a link to the start of this paragraph.",
rpar.Content(0))
    rpar.Content.AddText("Finally, here is an inline image: ")
    rpar.Content.AddImage(Me.Icon.ToBitmap())
    rpar.Content.AddText(".")

    ' 段落をドキュメントに追加します
    Me.C1PrintDocument1.Body.Children.Add(rpar)
    Me.C1PrintDocument1.Generate()

End Sub
```

**C# コードの書き方**

C#

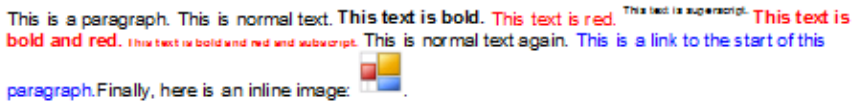
```
private void Form1_Load(object sender, EventArgs e)
{
    // 段落を作成します
    RenderParagraph rpar = new RenderParagraph();
    Font f = new Font(rpar.Style.Font, FontStyle.Bold);
```


Copyright © GrapeCity, Inc. All rights reserved.

```
    rpar.Content.AddText("This is a paragraph. This is normal text. ");
    rpar.Content.AddText("This text is bold. ", f);
    rpar.Content.AddText("This text is red. ", Color.Red);
    rpar.Content.AddText("This text is superscript. ", TextPositionEnum.Superscript);
    rpar.Content.AddText("This text is bold and red. ", f, Color.Red);
    rpar.Content.AddText("This text is bold and red and subscript. ", f, Color.Red,
TextPositionEnum.Subscript);
    rpar.Content.AddText("This is normal text again. ");
    rpar.Content.AddHyperlink("This is a link to the start of this paragraph.",
rpar.Content[0]);
    rpar.Content.AddText("Finally, here is an inline image: ");
    rpar.Content.AddImage(this.Icon.ToBitmap());
    rpar.Content.AddText(".");

    // 段落をドキュメントに追加します
    this.c1PrintDocument1.Body.Children.Add(rpar);
    this.c1PrintDocument1.Generate();

}
```

## プログラムを実行し、次の点を確認します。

テーブルは次のように表示されます。




# ドキュメント内のテーブルの下にテキストを追加する

このトピックでは、C1.C1PrintDocument.RenderTable オブジェクトを使用して、テキストをブロックフローにレンダリングする方法を示します。また、**Padding** プロパティを使用して、ブロックの位置を進める方法を示します。これにより、次の描画オブジェクト（この場合はテキスト）がその位置にレンダリングされます。このトピックでは、Padding プロパティを使用して、ドキュメント内のテーブルの1センチ下の位置にテキストを挿入します。このトピックは、すでにテーブルが作成されていることを前提としています。

1. C1.C1PrintDocument.RenderTable オブジェクトを使用して、表示されるテキストを作成します。

**Visual Basic コードの書き方**

Visual Basic

```
Dim caption As C1.C1Preview.RenderText = New
C1.C1Preview.RenderText(Me.C1PrintDocument1)
caption.Text = "In the table above, there are three rows and three columns."
```

**C# コードの書き方**

C#

```
C1.C1Preview.RenderText caption = new
C1.C1Preview.RenderText(this.c1PrintDocument1);
caption.Text = "In the table above, there are three rows and three columns.";
```


Copyright © GrapeCity, Inc. All rights reserved.

2. Padding プロパティを使用して、テキストをテーブルの1センチ下に配置します。

**Visual Basic コードの書き方**

```
Visual Basic
caption.Style.Padding.Top = New C1.C1Preview.Unit(1,
C1.C1Preview.UnitTypeEnum.Cm)
```

**C# コードの書き方**

```
C#
caption.Style.Padding.Top = new C1.C1Preview.Unit(1,
C1.C1Preview.UnitTypeEnum.Cm);
```

3. Add メソッドを使用して、テーブルの下にテキストを追加します。次のコードに示すように、テーブルの **Add** メソッドの下にテキストの **Add** メソッドを挿入します。

**Visual Basic コードの書き方**

```
Visual Basic
Me.C1PrintDocument1.Body.Children.Add(table)
Me.C1PrintDocument1.Body.Children.Add(caption)
```

**C# コードの書き方**

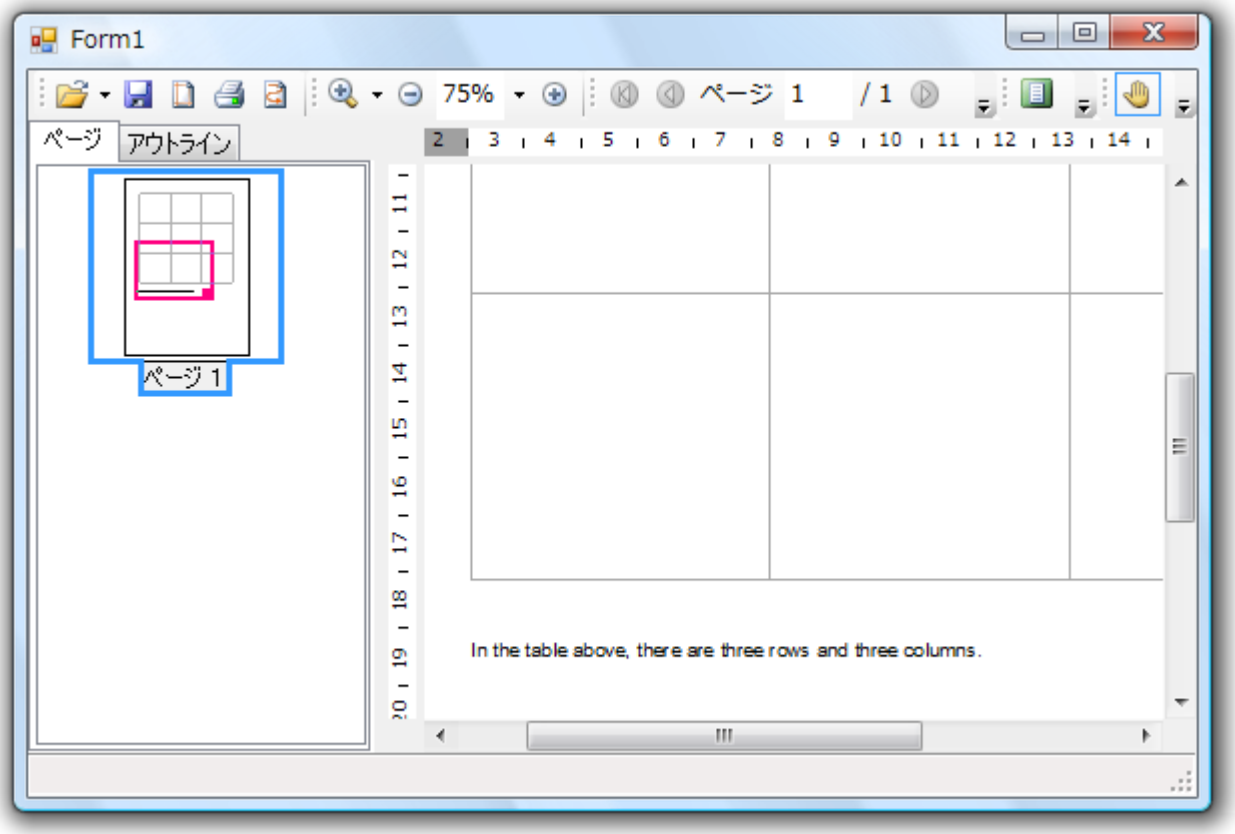
```
C#
this.c1PrintDocument1.Body.Children.Add(table);
this.c1PrintDocument1.Body.Children.Add(caption);
```

**メモ**:テキストの **Add** メソッドをテーブルの **Add** メソッドの下に配置すると、テーブルの下にテキストが挿入されます。テーブルの **Add** メソッドの上に配置すると、テーブルの上にテキストが表示されます。

## プログラムを実行し、次の点を確認します。

テーブルは次のように表示されます。


Copyright © GrapeCity, Inc. All rights reserved.




# C1PrintDocument 内のテキストのフォントとスタイルを変更する

**C1PrintDocument** には、指定された文字列を新しい段落を開始せずにブロックフローにレンダリングする **RenderInLineText** メソッドがあります。**RenderInLineText** メソッドはテキストを自動的にラップします。このトピックでは、**RenderInLineText** メソッドの使用方法を示します。

1. 新しい Windows フォームアプリケーションを作成します。**C1PrintPreview** コントロールをフォームに追加します。C1PrintDocument コンポーネントをフォームに追加します。これは、フォームの下のコンポーネントのトレイに表示されます。プレビューには **c1PrintPreview1**、ドキュメントには c1PrintDocument1 のデフォルト名が与えられます。**c1PrintPreview1** コントロールの **Document** プロパティの値を **c1PrintDocument1** に設定します。これにより、アプリケーションの実行時にプレビューにドキュメントが表示されます。
2. フォームをダブルクリックして、**フォームロード**イベントのハンドラを作成します。ここに、以下のすべてのコードを記述します。
3. **StartDoc** メソッドでドキュメントを開始し、デフォルトのフォントを使ってテキスト行を作成します。次に例を示します。

   **Visual Basic コードの書き方**

   Visual Basic
   ```
   Me.C1PrintDocument1.StartDoc()
   Me.C1PrintDocument1.RenderInlineText("With C1PrintDocument you can print ")
   ```

   **C# コードの書き方**

   C#
   ```
   this.c1PrintDocument1.StartDoc();
   this.c1PrintDocument1.RenderInlineText("With C1PrintDocument you can print ");
   ```
4. 別のフォントと色で続行し、デフォルトのフォントと色に戻ります。


Copyright © GrapeCity, Inc. All rights reserved.

**Visual Basic コードの書き方**

```
Visual Basic
Me.C1PrintDocument1.RenderInlineText("Line by Line", New Font("Times New Roman",
30, FontStyle.Bold), Color.FromArgb(0, 0, 125))
Me.C1PrintDocument1.RenderInlineText(" and modify text attributes as you go.")
```

**C# コードの書き方**

```
C#
this.c1PrintDocument1.RenderInlineText("Line by Line", new Font("Times New
Roman", 30, FontStyle.Bold), Color.FromArgb(0, 0, 125));
this.c1PrintDocument1.RenderInlineText(" and modify text attributes as you
go.");
```

5. 行内の最後の数ワードを緑色にします。

**Visual Basic コードの書き方**

```
Visual Basic
Me.C1PrintDocument1.RenderInlineText(" The text wraps automatically, so your
life becomes easier.", Color.Green)
```

**C# コードの書き方**

```
C#
this.c1PrintDocument1.RenderInlineText(" The text wraps automatically, so your
life becomes easier.", Color.Green);
```

6. EndDoc メソッドでドキュメントを終了します。

**Visual Basic コードの書き方**

```
Visual Basic
Me.C1PrintDocument1.EndDoc()
```

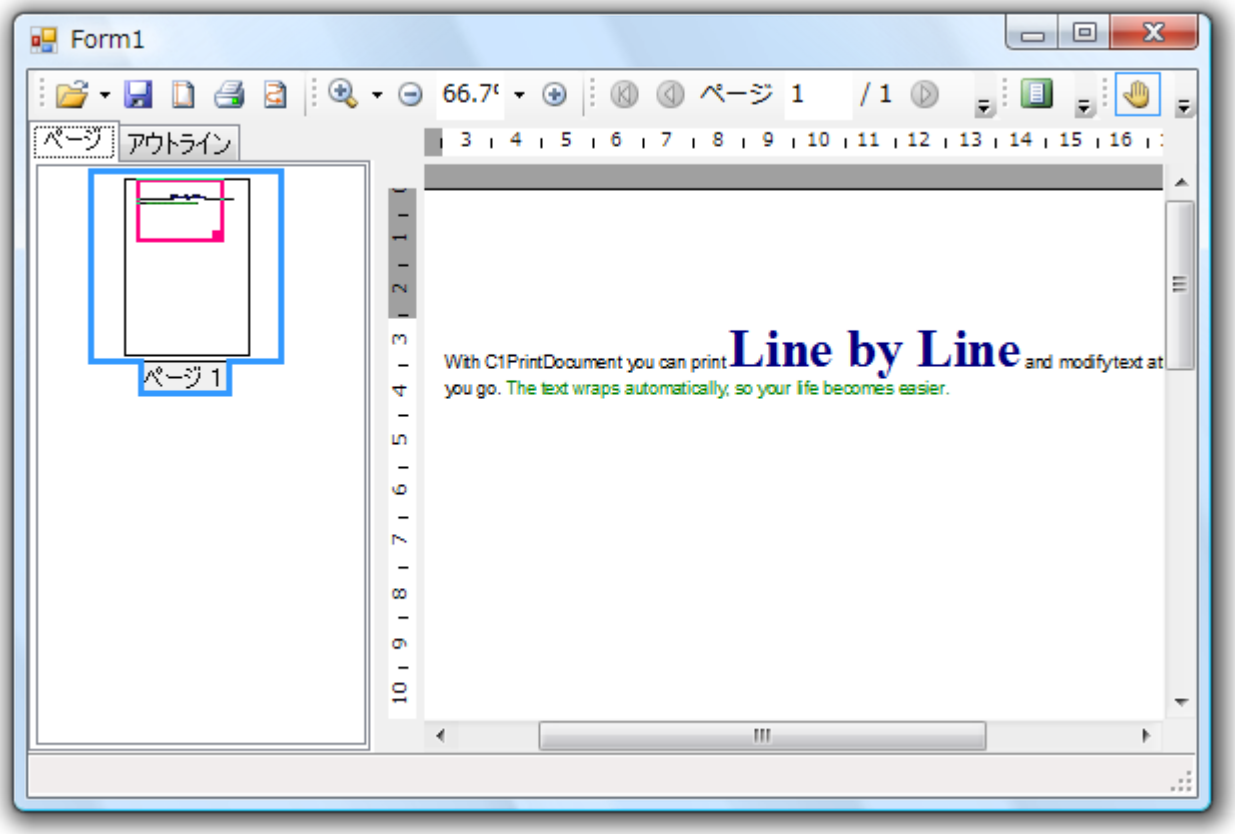
**C# コードの書き方**

```
C#
this.c1PrintDocument1.EndDoc();
```

## プログラムを実行し、次の点を確認します。

テーブルは次のように表示されます。


Copyright © GrapeCity, Inc. All rights reserved.




# C1PrintDocument 内のページヘッダーの作成

次のトピックでは、3つのパーツを含むページヘッダーを作成する方法と、ページヘッダーに背景色を追加する方法について説明します。

# 3つの部分から成るページヘッダーを作成する

このトピックは、3つの列に分かれたヘッダーを作成する方法を示します。このトピックでは、次の主要なポイントを示します。

- **C1PrintDocument** 内に1行3列のテーブルを作成する
- ページヘッダーの各セクションのテキスト配置を設定する
- **Style** クラスの **TextAlignHorz** プロパティを使用して、テキストの水平方向の配置を指定します。AlignHorzEnum のメンバ（left、right、justify、center）を TextAlignHorz プロパティに割り当てます。

次の手順は、3つの部分から成るヘッダーを作成する方法を示します。

1. 新しい Windows フォームアプリケーションを作成します。
2. **C1PrintPreview** コントロールをフォームに追加します。
3. **C1PrintDocument** コンポーネントをフォームに追加します。これは、フォームの下のコンポーネントのトレイに表示されます。プレビューには **c1PrintPreview1**、ドキュメントには **c1PrintDocument1** のデフォルト名が与えられます。
4. **c1PrintPreview1** コントロールの **Document** プロパティの値を **c1PrintDocument1** に設定します。これにより、アプリケーションの実行時にプレビューにドキュメントが表示されます。
5. フォームをダブルクリックして、**フォームロード**イベントのハンドラを作成します。ここに、以下のすべてのコードを記述します。**フォームロード**イベントで、ドキュメントを設定します。ページヘッダーの RenderTable を作成します。

**Visual Basic コードの書き方**

```
Visual Basic
```


Copyright © GrapeCity, Inc. All rights reserved.

```
Me.C1PrintDocument1.StartDoc()
Dim theader As New C1.C1Preview.RenderTable(Me.C1PrintDocument1)
```

**C# コードの書き方**

C#

```
this.c1PrintDocument1.StartDoc();
C1.C1Preview.RenderTable theader = new
C1.C1Preview.RenderTable(this.c1PrintDocument1);
```

6. 本体に1行を追加し、ヘッダーの左、中央、および右部分に対応する3つの列を追加します。TextAlignHorz プロパティを使用して、ページヘッダーの各列のテキストの配置を設定します。また、ページヘッダー内のテキストに新しいフォントスタイルを割り当てます。この例では、フォントは 14 ポイントサイズの Arial です。

**Visual Basic コードの書き方**

Visual Basic

```
' ヘッダーの各部分の配置を設定します
theader.Cells(0, 0).Style.TextAlignHorz = C1.C1Preview.AlignHorzEnum.Left
theader.Cells(0, 1).Style.TextAlignHorz = C1.C1Preview.AlignHorzEnum.Center
theader.Cells(0, 2).Style.TextAlignHorz = C1.C1Preview.AlignHorzEnum.Right
theader.CellStyle.Font = New Font("Arial", 14)
```

**C# コードの書き方**

C#

```
// ヘッダーの各部分の配置を設定します
theader.Cells[0, 0].Style.TextAlignHorz = C1.C1Preview.AlignHorzEnum.Left;
theader.Cells[0, 1].Style.TextAlignHorz = C1.C1Preview.AlignHorzEnum.Center;
theader.Cells[0, 2].Style.TextAlignHorz = C1.C1Preview.AlignHorzEnum.Right;
theader.CellStyle.Font = new Font("Arial", 14);
```

7. ページヘッダーのテーブルの各列にテキストを描画します。ドキュメントの PageHeader の RenderObject プロパティを **theader** に設定します。EndDoc メソッドを呼び出して、ドキュメントの生成を終了します。

**Visual Basic コードの書き方**

Visual Basic

```
theader.Cells(0, 0).Text = "Left part"
theader.Cells(0, 1).Text = "Center part"
theader.Cells(0, 2).Text = "Right part"
Me.C1PrintDocument1.RenderBlock(theader)
Me.C1PrintDocument1.EndDoc()
```

**C# コードの書き方**

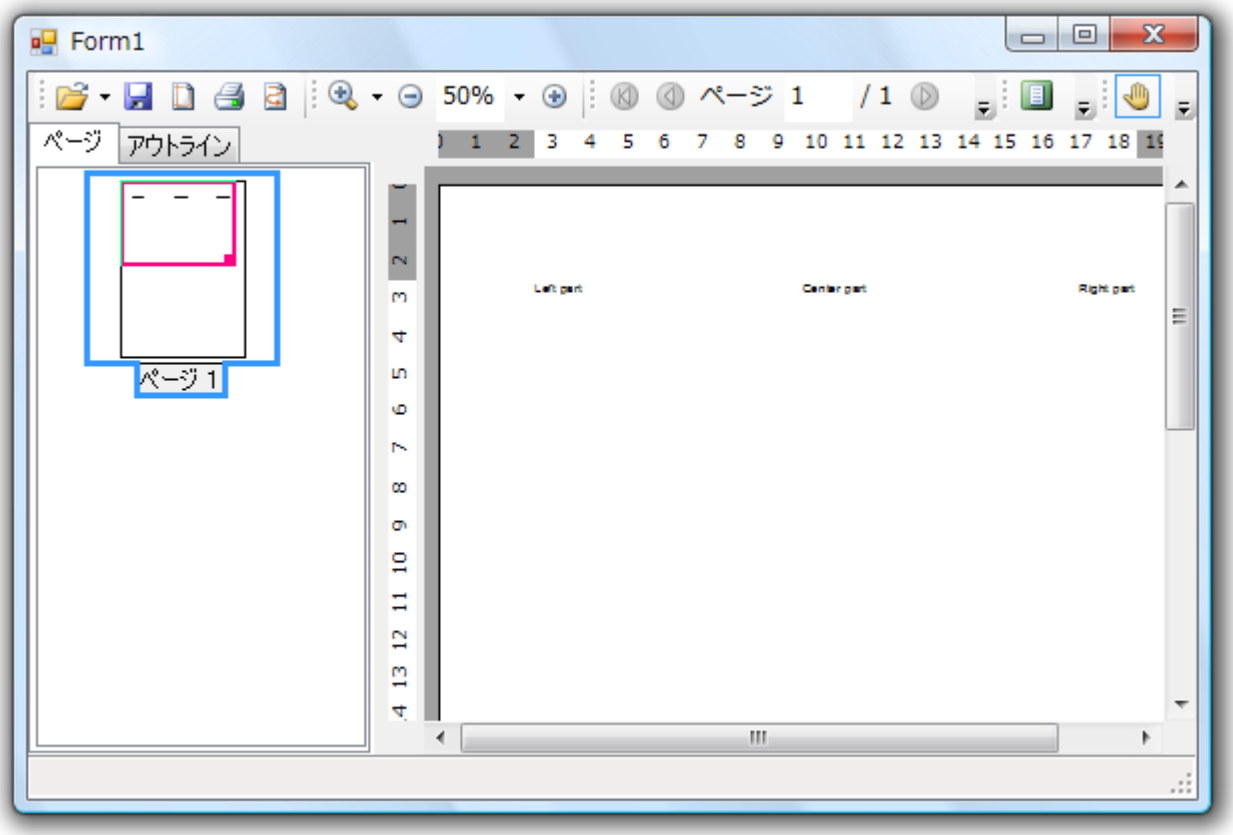
C#

```
theader.Cells[0, 0].Text = "Left part";
theader.Cells[0, 1].Text = "Center part";
theader.Cells[0, 2].Text = "Right part";
this.c1PrintDocument1.RenderBlock(theader);
this.c1PrintDocument1.EndDoc();
```


Copyright © GrapeCity, Inc. All rights reserved.

## プログラムを実行し、次の点を確認します。

実行時に、3つの部分から成る新しいページヘッダーが次のように表示されます。




# ページヘッダーへの背景色の追加

このトピックでは、ページヘッダーに背景色を追加する方法を示します。次のオブジェクトを使用します。

- **C1PrintDocument**
- **C1DocStyle**
- **RenderTable**

次の手順を実行します。

1. 新しい Windows フォームアプリケーションを作成します。
2. **C1PrintPreview** コントロールをフォームに追加します。
3. **C1PrintDocument** コンポーネントをフォームに追加します。これは、フォームの下のコンポーネントのトレイに表示されます。プレビューには C1PrintPreview1、ドキュメントには C1PrintDocument1 のデフォルト名が与えられます。
4. C1PrintPreview1 コントロールの **Document** プロパティの値を **C1PrintDocument1** に設定します。これにより、アプリケーションの実行時にプレビューにドキュメントが表示されます。
5. フォームをダブルクリックして、**Form_Load** イベントのハンドラを作成します。ここに、以下のすべてのコードを記述します。**StartDoc** メソッドを呼び出してドキュメントの生成を開始します。テーブルを **RenderTable** クラスに割り当てます。

**Visual Basic コードの書き方**

| Visual Basic |
| --- |
| `Me.C1PrintDocument1.StartDoc()` |


Copyright © GrapeCity, Inc. All rights reserved.

```
Dim theader As New C1.C1Preview.RenderTable(Me.C1PrintDocument1)
```

**C# コードの書き方**

C#

```
this.c1PrintDocument1.StartDoc();
C1.C1Preview.RenderTable theader = new
C1.C1Preview.RenderTable(this.c1PrintDocument1);
```

6. この例では、テーブル内のページヘッダーに金色を割り当てます。TableRow クラスの **Style** プロパティを使用して、テーブル（theader）に背景色を適用します。

**Visual Basic コードの書き方**

Visual Basic

```
theader.Style.BackColor = Color.Gold
```

**C# コードの書き方**

C#

```
theader.Style.BackColor = Color.Gold;
```

7. 1行1列のテーブルを作成する必要があります。ページヘッダーのテーブルにテキストを描画します。テキストを右に配置します。このテキストのために 14 ポイントサイズの新しい Arial フォントを作成します。

**Visual Basic コードの書き方**

Visual Basic

```
theader.Cells(0, 0).Style.TextAlignHorz = C1.C1Preview.AlignHorzEnum.Right

' いくつかのスタイルを設定します。
theader.CellStyle.Font = New Font("Arial", 14)

' テキストを割り当てます。
theader.Cells(0, 0).Text = "Swim Team Practice Schedule"

' テーブルをページヘッダーとして設定します。
Me.C1PrintDocument1.RenderBlock(theader)
Me.C1PrintDocument1.EndDoc()
```

**C# コードの書き方**

C#

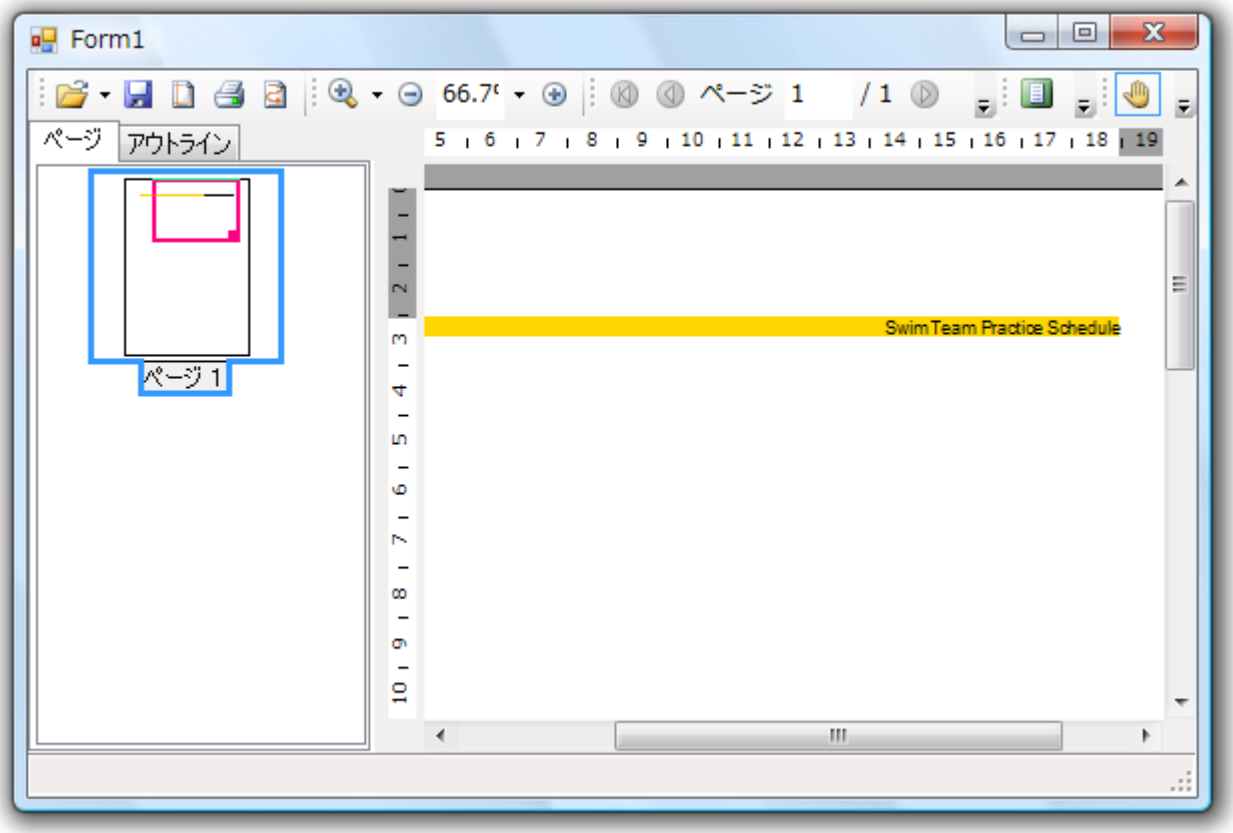
```
theader.Cells[0, 0].Style.TextAlignHorz = C1.C1Preview.AlignHorzEnum.Right;

// いくつかのスタイルを設定します。
theader.CellStyle.Font = new Font("Arial", 14);
```


Copyright © GrapeCity, Inc. All rights reserved.

```
// テキストを割り当てます。
theader.Cells[0, 0].Text = "Swim Team Practice Schedule";

// テーブルをページヘッダーとして設定します。
this.c1PrintDocument1.RenderBlock(theader);
this.c1PrintDocument1.EndDoc();
```

## プログラムを実行し、次の点を確認します。

ヘッダーは、次のように表示されます。




# 水平方向の直線の描画

このトピックは、**C1PrintDocument** 内に水平方向の直線を描画する方法を示します。

1. 新しい Windows フォームアプリケーションを作成します。
2. **C1PrintPreviewControl** コントロールをフォームに追加します。
3. C1PrintDocument コンポーネントをフォームに追加します。これは、フォームの下のコンポーネントのトレイに表示されます。プレビューには C1PrintPreview1、ドキュメントには C1PrintDocument1 のデフォルト名が与えられます。
4. C1PrintPreview1 コントロールの **Document** プロパティの値を **C1PrintDocument1** に設定します。これにより、アプリケーションの実行時にプレビューにドキュメントが表示されます。
5. フォームをダブルクリックして、**フォームロード**イベントのハンドラを作成します。ここに、以下のすべてのコードを記述します。
6. RenderLine レンダリングオブジェクトを使用して、C1PrintDocument 内に直線を描画できます。直線の主なプロパティは、2つの点（直線の開始と終了）です。直線をブロックフローにレンダリングし、それが水平になるように、2つの点の Y 座標をデフォルトの0のままにします。開始 X 座標も0のままにします。終了座標はページ幅に設定します。これで、ページ全体を横切る水平線をブロックフローの現在の Y 座標の位置にレンダリングする RenderLine オブジェクトになります。


Copyright © GrapeCity, Inc. All rights reserved.

**Visual Basic コードの書き方**

Visual Basic

```
Me.C1PrintDocument1.StartDoc()
Dim rl As New C1.C1Preview.RenderLine(Me.C1PrintDocument1)
rl.X = Me.C1PrintDocument1.PageLayout.PageSettings.Width
Dim ld As New C1.C1Preview.LineDef("4mm", Color.SteelBlue)
Me.C1PrintDocument1.RenderBlockHorzLine(rl.X, ld)
Me.C1PrintDocument1.EndDoc()
```

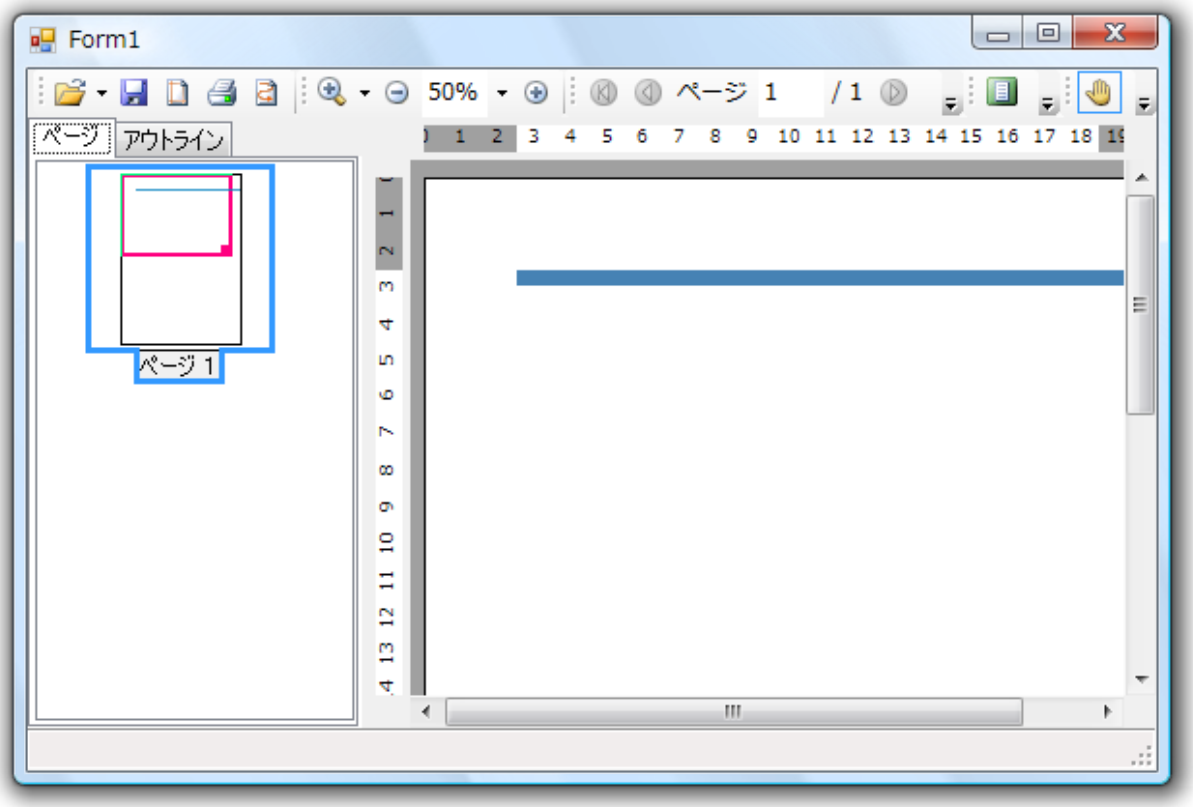
**C# コードの書き方**

C#

```
this.c1PrintDocument1.StartDoc();
C1.C1Preview.RenderLine rl = new C1.C1Preview.RenderLine(this.c1PrintDocument1);
rl.X = this.c1PrintDocument1.PageLayout.PageSettings.Width;
C1.C1Preview.LineDef ld = new C1.C1Preview.LineDef("4mm", Color.SteelBlue);
this.c1PrintDocument1.RenderBlockHorzLine(rl.X, ld);
this.c1PrintDocument1.EndDoc();
```

## プログラムを実行し、次の点を確認します。

ドキュメントは、次の図のように表示されます。




# C1PrintDocument 内にページフッターを作成する

このトピックは、2つの列に分かれたテーブルフッターを作成する方法を示します。このトピックでは、次の主要なポイントを示し


Copyright © GrapeCity, Inc. All rights reserved.

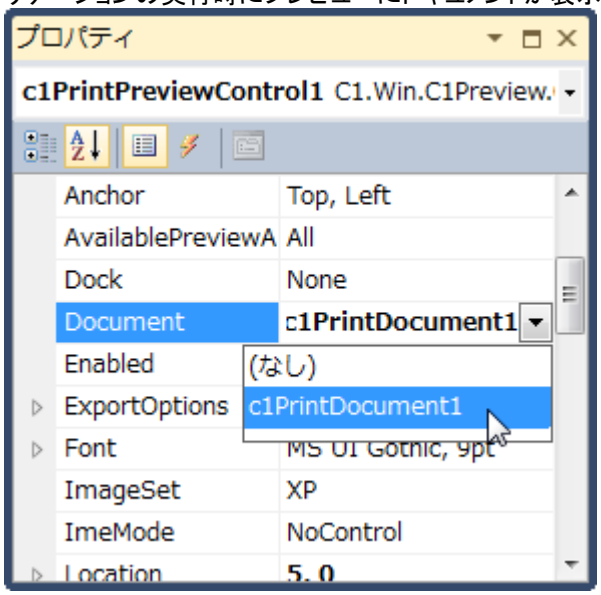
ます。

- **C1PrintDocument** 内の複数の行と列から成るテーブルにフッターを追加する。
- 各ページの最後にテーブルフッターを設定する。TableVectorCollection クラスの **Count** プロパティを使用して、各ページのテーブルの最後にフッターを挿入します。
- ページフッターの各セクションで行および列スパンを設定する。
  **TableCell** クラスの **SpanRows** および **SpanCols** プロパティを使用して、行および列スパンを指定します。
- ページフッターの各セクションのテキスト配置を設定する。**Style** クラスの **TextAlignHorz** および **TextAlignVert**プロパティを使用して、テキストの水平および垂直方向の配置を指定します。**AlignHorzEnum** のメンバ(left、right、justify、center)を TextAlignHorz プロパティに、**AlignVertEnum** のメンバ(bottom、center、justify、top)を TextAlignVert プロパティに割り当てることができます。

**メモ**:このトピック内のサンプルコードは、「using C1.C1Preview;」ディレクティブ(C# の場合。他の言語の場合は相当する構文)がファイルに挿入されていることを前提としています。したがって、完全修飾された型名(C1.C1Preview.RenderText など)ではなく、クラス名部分(RenderText)だけを使用します。

次の手順を完了して、2つのパーツを含むテーブルフッターを作成します。

1. 新しい Windows フォームアプリケーションを作成します。
2. C1PrintPreview コントロールをフォームに追加します。C1PrintDocument コンポーネントをフォームに追加します。これは、フォームの下のコンポーネントのトレイに表示されます。プレビューには **c1PrintPreview1**、ドキュメントには **c1PrintDocument1** のデフォルト名が与えられます。
3. c1PrintPreview1 コントロールの **Document** プロパティの値を **c1PrintDocument1** に設定します。これにより、アプリケーションの実行時にプレビューにドキュメントが表示されます。




4. フォームをダブルクリックして、**フォームロード**イベントのハンドラを作成します。ここに、以下のすべてのコードを記述します。**フォームロード**イベントで、ドキュメントを設定します。ページフッターの RenderTable を作成し、100 行4列のテーブルとサンプルテキストを追加します。
5. ページフッターの RenderTable を作成し、100 行4列のテーブルとサンプルテキストを追加します。

**Visual Basic コードの書き方**

Visual Basic

```
Dim rt1 As New C1.C1Preview.RenderTable(Me.C1PrintDocument1)

' 100 行4列のテーブルを作成し、テキストを挿入します
Dim r As Integer = 100
Dim c As Integer = 4
Dim row As Integer
```


Copyright © GrapeCity, Inc. All rights reserved.

```
Dim col As Integer
For row = 0 To r - 1 Step +1
    For col = 0 To c - 1 Step +1
        Dim celltext As C1.C1Preview.RenderText = New
C1.C1Preview.RenderText(Me.C1PrintDocument1)
        celltext.Text = String.Format("Cell ({0},{1})", row, col)
        rt1.Cells(row, col).RenderObject = celltext
    Next
Next

' テーブルをドキュメントに追加します
Me.C1PrintDocument1.Body.Children.Add(rt1)
```

**C# コードの書き方**

C#

```
C1.C1Preview.RenderTable rt1 = new
C1.C1Preview.RenderTable(this.c1PrintDocument1);

// 100 行4列のテーブルを作成し、テキストを挿入します
const int r = 100;
const int c = 4;
for (int row = 0; row < r; ++row)
{
    for (int col = 0; col < c; ++col)
    {
        C1.C1Preview.RenderText celltext = new
C1.C1Preview.RenderText(this.c1PrintDocument1);
        celltext.Text = string.Format("Cell ({0},{1})", row, col);
        rt1.Cells[row, col].RenderObject = celltext;
    }
}

// テーブルをドキュメントに追加します
this.c1PrintDocument1.Body.Children.Add(rt1);
```

6. 次のコードを追加して、フォントタイプを Arial、10 ポイントに設定し、背景色をレモンクリーム色に設定します。

**Visual Basic コードの書き方**

Visual Basic

```
' テーブルフッターを設定します。
rt1.RowGroups(rt1.Rows.Count - 2, 2).PageFooter = True
rt1.RowGroups(rt1.Rows.Count - 2, 2).Style.BackColor = Color.LemonChiffon
rt1.RowGroups(rt1.Rows.Count - 2, 2).Style.Font = New Font("Arial", 10,
FontStyle.Bold)
```

**C# コードの書き方**

C#

```
// テーブルフッターを設定します。
rt1.RowGroups[rt1.Rows.Count - 2, 2].PageFooter = true;
```


Copyright © GrapeCity, Inc. All rights reserved.

```
rt1.RowGroups[rt1.Rows.Count - 2, 2].Style.BackColor = Color.LemonChiffon;
rt1.RowGroups[rt1.Rows.Count - 2, 2].Style.Font = new Font("Arial", 10,
FontStyle.Bold);
```

ここでは、**Count** プロパティを使用して、ページの最後の2行をフッター用に確保し、**RowGroups** プロパティを使用して、その行をグループ化しました。次に、テキストに新しいフォントスタイルを割り当て、ページフッター内のセルに新しい背景色を割り当てました。

7. TextAlignHorz および TextAlignVert プロパティを使用して、ページフッターの各列のテキストの配置を設定します。また、**SpanRows** および **SpanCols** プロパティを使用して、最後の2行にまたがるフッターを作成し、2つの列を作成します。ページフッターのテーブルの各列にテキストを描画します。最後に、**Generate** メソッドを使用して、ドキュメントを作成します。

**Visual Basic コードの書き方**

Visual Basic

```
' テーブルフッターテキストを追加します
rt1.Cells(rt1.Rows.Count - 2, 0).SpanRows = 2
rt1.Cells(rt1.Rows.Count - 2, 0).SpanCols = rt1.Cols.Count - 1
rt1.Cells(rt1.Rows.Count - 2, 0).Style.TextAlignHorz =
C1.C1Preview.AlignHorzEnum.Left
rt1.Cells(rt1.Rows.Count - 2, 0).Style.TextAlignVert =
C1.C1Preview.AlignVertEnum.Center
Dim tf As C1.C1Preview.RenderText = New
C1.C1Preview.RenderText(Me.C1PrintDocument1)
tf = CType(rt1.Cells(rt1.Rows.Count - 2, 0).RenderObject,
C1.C1Preview.RenderText)
tf.Text = "This is a table footer."

' ページ番号を追加します
rt1.Cells(rt1.Rows.Count - 2, 3).SpanRows = 2
rt1.Cells(rt1.Rows.Count - 2, 3).Style.TextAlignHorz =
C1.C1Preview.AlignHorzEnum.Right
rt1.Cells(rt1.Rows.Count - 2, 3).Style.TextAlignVert =
C1.C1Preview.AlignVertEnum.Center

' タグ（ページ番号／ページ数など）をドキュメントの任意の位置に挿入できます
Dim pn As C1.C1Preview.RenderText = New
C1.C1Preview.RenderText(Me.C1PrintDocument1)
pn = CType(rt1.Cells(rt1.Rows.Count - 2, 3).RenderObject,
C1.C1Preview.RenderText)
pn.Text = "Page [PageNo] of [PageCount]"

Me.C1PrintDocument1.Generate()
```

**C# コードの書き方**

C#

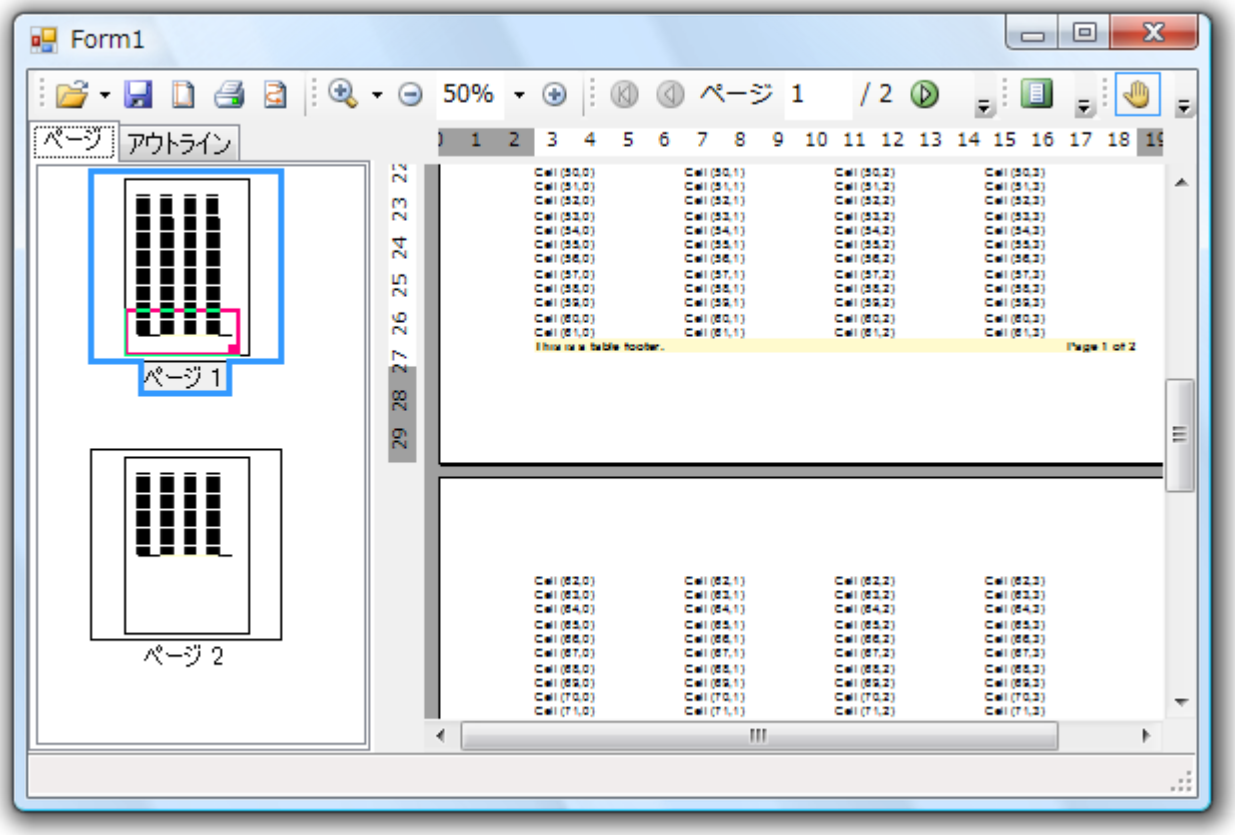
```
// テーブルフッターテキストを追加します
rt1.Cells[rt1.Rows.Count - 2, 0].SpanRows = 2;
rt1.Cells[rt1.Rows.Count - 2, 0].SpanCols = rt1.Cols.Count - 1;
rt1.Cells[rt1.Rows.Count - 2, 0].Style.TextAlignHorz =
C1.C1Preview.AlignHorzEnum.Center;
```


Copyright © GrapeCity, Inc. All rights reserved.

```
rt1.Cells[rt1.Rows.Count - 2, 0].Style.TextAlignVert =
C1.C1Preview.AlignVertEnum.Center;
((C1.C1Preview.RenderText)rt1.Cells[rt1.Rows.Count - 2, 0].RenderObject).Text =
"This is a table footer.";

// ページ番号を追加します
rt1.Cells[rt1.Rows.Count - 2, 3].SpanRows = 2;
rt1.Cells[rt1.Rows.Count - 2, 3].Style.TextAlignHorz =
C1.C1Preview.AlignHorzEnum.Right;
rt1.Cells[rt1.Rows.Count - 2, 3].Style.TextAlignVert =
C1.C1Preview.AlignVertEnum.Center;

// タグ(ページ番号／ページ数など)をドキュメントの任意の位置に挿入できます
((C1.C1Preview.RenderText)rt1.Cells[rt1.Rows.Count - 2, 3].RenderObject).Text =
"Page [PageNo] of [PageCount]";

this.c1PrintDocument1.Generate();
```

## アプリケーションを保存し、実行します。

実行時に、2つの部分から成る新しいページフッターが次のように表示されます。





Copyright © GrapeCity, Inc. All rights reserved.

# 設計時サポート

**Report for WinForms** では、プロパティグリッド、メニュー、そして Visual Studio のデザイナを使用して設計時に簡単に設定する事ができます。次のセクションでは、設計時に **Report for WinForms** のコントロールを設定するための方法を説明します。

# C1Report タスクメニュー

Visual Studio では、**C1Report** コンポーネントにスマートタグが含まれます。スマートタグは、最もよく使用されるプロパティを提供するショートカットタスクメニューです。

**C1Report** コンポーネントから、C1Report ウィザード（まだ作成されていないレポートを定義する）または **C1ReportDesigner**（プロジェクトに既に存在するレポートを定義する）にすばやく簡単にアクセスしたり、スマートタグからレポートをロードすることができます。

[**C1Report タスク**]メニューにアクセスするには、C1Report コンポーネントの右上隅にあるスマートタグ をクリックします。

[**C1Report タスク**]メニューの機能は次のとおりです。

- **レポート定義ファイルの読み込み**

  [レポート定義ファイルの読み込み]をクリックすると、[**レポートの選択**]ダイアログボックスが表示されます。

- **レポートの編集**

  [**レポートの編集**]をクリックすると、レポート定義がまだ作成されていない場合は **C1Report ウィザード**が表示され、既にレポートが作成されている場合は **C1Report デザイナ**が表示されます。

  **C1Report ウィザード**の使用方法については、「基本的なレポート定義ファイルの作成」を参照してください。**C1Report デザイナ**の使用方法については、「C1Report デザイナの使い方」を参照してください。

  > **メモ**:タスクメニューおよび[プロパティ]ウィンドウに[**レポートの編集**]コマンドが表示されていない場合は、コントロールが **C1Report デザイナ**アプリケーションを見つけることができなかった可能性があります。この場合は、**C1Report デザイナ**アプリケーションをスタンドアロンモードで一度実行します。デザイナの場所がレジストリに保存されるため、これで C1Report コンポーネントが デザイナを検出できるようになります。

- **ドキュメントを表示するビューア**

  任意の視覚的なプレビューコントロール（**C1PrintPreviewControl** コントロールなど）がアプリケーションに含まれている場合、[**C1Report タスク**]メニューはそれらをここに表示します。プレビューコントロールの名前の横にあるチェックボックスがオンの場合、**C1Report** コントロールはプレビューコントロールに関連付けられ、レポートがプレビューコントロールに表示されます。

- **ローカライズ**

  [**ローカライズ**]をクリックすると、[**ローカライズ**]ダイアログボックスが開きます。[**ローカライズ**]ダイアログボックスで、ローカライズ設定をカスタマイズできます。[**ローカライズ**]ダイアログボックスの詳細については、「ローカライズ」を参照してください。


Copyright © GrapeCity, Inc. All rights reserved.

- **C1Report のバージョン情報**

  [**C1Report のバージョン情報**]をクリックすると、[**バージョン情報**]ダイアログボックスが表示され、[C1Report のバージョン情報]のバージョン番号とオンラインリソースを確認できます。

# C1Report コンテキストメニュー

**C1Report ウィザード**または **C1Report デザイナ**にアクセスしたり、レポートをロードする場合は、C1Report コンポーネントのコンテキストメニューを使用できます。

**C1Report** コントロールのコンテキストメニューにアクセスするには、C1Report コンポーネントを右クリックします。

C1Report コンテキストメニューの機能は次のとおりです。

- **レポート定義ファイルの読み込み**

  [**レポート定義ファイルの読み込み**]をクリックすると、[**レポートの選択**]ダイアログボックスが表示されます。

- **レポートの編集**

  [**レポートの編集**]をクリックすると、レポート定義がまだ作成されていない場合は **C1Report ウィザード**が表示され、既にレポートが作成されている場合は **C1Report デザイナ**が表示されます。

  **C1Report ウィザード**の使用方法については、「基本的なレポート定義ファイルの作成」を参照してください。C1Report デザイナの使用方法については、「 C1Report デザイナの使い方」を参照してください。

  > **メモ**:コンテキストメニューおよび[プロパティ]ウィンドウに[**レポートの編集**]コマンドが表示されていない場合は、コントロールが **C1Report デザイナ**アプリケーションを見つけることができなかった可能性があります。この場合は、**C1Report デザイナ**アプリケーションをスタンドアロンモードで一度実行します。デザイナの場所がレジストリに保存されるため、これで C1Report コンポーネントがデザイナを検出できるようになります。

- **ローカライズ**

  [**ローカライズ**]をクリックすると、[**ローカライズ**]ダイアログボックスが開きます。[**ローカライズ**]ダイアログボックスで、ローカライズ設定をカスタマイズできます。[**ローカライズ**]ダイアログボックスの詳細については、「ローカライズ」を参照してください。

- **C1Report のバージョン情報**

  [**C1Report のバージョン情報**]をクリックすると、[**C1Report のバージョン情報**]ダイアログボックスが表示されます。ここに、C1Report のバージョン番号やオンラインリソースが表示されます。

# C1RdlReport タスクメニュー


Copyright © GrapeCity, Inc. All rights reserved.

Visual Studio では、**C1RdlReport** コントロールにスマートタグが用意されています。スマートタグとは、最もよく使用されるプロパティを提供するショートカットタスクメニューです。

[**C1RdlReport タスク**]メニューにアクセスするには、C1RdlReport コンポーネントの右上端にあるスマートタグ（）をクリックします。

[**C1RdlReport タスク**]メニューは次のように動作します。

- **レポートの読み込み中**

  [**レポート定義ファイルの読み込み**]をクリックすると、[**開く**]ダイアログボックスが開き、RDL ファイルを選択して C1RdlReport コンポーネントにロードできます。

- **クリア**

  [**クリア**]をクリックすると、現在ロードされている任意のレポート定義をクリアします。このオプションをクリックすると、ロードされたレポート定義をクリアするかどうかの確認を求めるダイアログボックスが表示されます。ダイアログボックスで、操作を続行する場合は〈**Yes**〉をクリックし、レポート定義をクリアしない場合は〈**No**〉をクリックします。

- **ドキュメントを表示するビューア**

  任意の視覚的なプレビューコントロール（**C1PrintPreviewControl**コントロールなど）がアプリケーションに含まれている場合、[**C1RdlReport タスク**]メニューはそれらをここに表示します。プレビューコントロールの名前の横にあるチェックボックスがオンの場合、C1RdlReport コントロールはプレビューコントロールに関連付けられ、レポートがプレビューコントロールに表示されます。

- **ローカライズ**

  [**ローカライズ**]をクリックすると、[**ローカライズ**]ダイアログボックスが開きます。[**ローカライズ**]ダイアログボックスで、ローカライズ設定をカスタマイズできます。[ローカライズ]ダイアログボックスの詳細については、「ローカライズ」を参照してください。

- **C1RdlReport のバージョン情報**

  [**C1RdlReport のバージョン情報**]をクリックすると、[**C1RdlReport のバージョン情報**]ダイアログボックスが表示され、C1RdlReport のバージョン情報 のバージョン番号とオンラインリソースを確認できます。

# C1RdlReport コンテキストメニュー

**C1RdlReport** コントロールのコンテキストメニューにアクセスするには、C1RdlReport コンポーネントを右クリックします。コンテキストメニューが表示されます。


Copyright © GrapeCity, Inc. All rights reserved.

**C1RdlReport** コンテキストメニューは次のように動作します。

- **レポート定義ファイルの読み込み**

  [**レポート定義ファイルの読み込み**]をクリックすると、[**開く**]ダイアログボックスが開き、RDL ファイルを選択して C1RdlReport コンポーネントにロードできます。

- **クリア**

  [**クリア**]をクリックすると、現在ロードされている任意のレポート定義をクリアします。このオプションをクリックすると、ロードされたレポート定義をクリアするかどうかの確認を求めるダイアログボックスが表示されます。ダイアログボックスで、操作を続行する場合は〈**Yes**〉をクリックし、レポート定義のクリアをキャンセルする場合は〈**No**〉をクリックします。

- **ローカライズ**

  [**ローカライズ**]をクリックすると、[**ローカライズ**]ダイアログボックスが開きます。[**ローカライズ**]ダイアログボックスで、ローカライズ設定をカスタマイズできます。[**ローカライズ**]ダイアログボックスの詳細については、「ローカライズ」を参照してください。

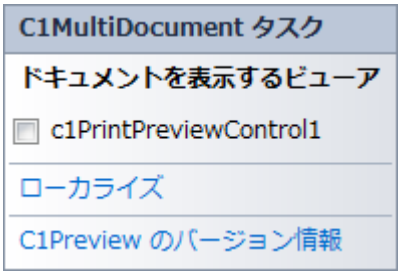
- **C1RdlReport のバージョン情報**

  [**C1RdlReport のバージョン情報**]をクリックすると、[**C1RdlReport のバージョン情報**]ダイアログボックスが表示され、C1RdlReport のバージョン情報 のバージョン番号とオンラインリソースを確認できます。

# C1MultiDocument タスクメニュー

Visual Studio では、**C1MultiDocument** コントロールにスマートタグが用意されています。スマートタグとは、最もよく使用されるプロパティを提供するショートカットタスクメニューです。

[**C1MultiDocument タスク**]メニューにアクセスするには、C1MultiDocument コンポーネントの右上端にあるスマートタをクリックします。




[**C1MultiDocument タスク**]メニューは次のように動作します。

- **このドキュメントのビューア**


Copyright © GrapeCity, Inc. All rights reserved.

任意の視覚的なプレビューコントロール（**C1PrintPreviewControl** コントロールなど）がアプリケーションに含まれている場合、[**C1MultiDocument タスク**]メニューはそれらをここに表示します。プレビューコントロールの名前の横にあるチェックボックスがオンの場合、C1MultiDocument コンポーネントはプレビューコントロールに関連付けられ、レポートがプレビューコントロールに表示されます。

- **ローカライズ**

  [ローカライズ]をクリックすると、[**ローカライズ**]ダイアログボックスが開きます。[**ローカライズ**]ダイアログボックスで、ローカライズ設定をカスタマイズできます。[**ローカライズ**]ダイアログボックスの詳細については、「ローカライズ」を参照してください。

- **バージョン情報**

  [**バージョン情報**]をクリックすると、[**バージョン情報**]ダイアログボックスが表示され、**Reports for WinForms** のバージョン番号とオンラインリソースを確認できます。

# C1MultiDocument コンテキストメニュー

**C1MultiDocument** コンポーネントのコンテキストメニューにアクセスするには、C1MultiDocument コンポーネントを右クリックします。コンテキストメニューが表示されます。

C1MultiDocument コンテキストメニューは次のように動作します。

- **ローカライズ**

  [**ローカライズ**]をクリックすると、[**ローカライズ**]ダイアログボックスが開きます。[**ローカライズ**]ダイアログボックスで、ローカライズ設定をカスタマイズできます。[**ローカライズ**]ダイアログボックスの詳細については、「ローカライズ」を参照してください。

- **バージョン情報**

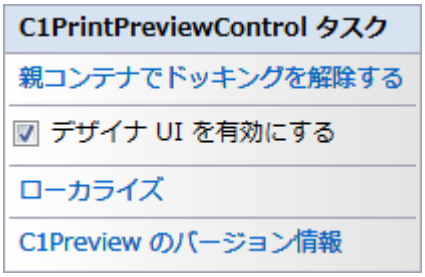
  [**バージョン情報**]をクリックすると、[**バージョン情報**]ダイアログボックスが表示され、Reports for WinForms のバージョン番号とオンラインリソースを確認できます。

# C1PrintPreviewControl タスクメニュー

**C1PrintPreviewControl タスク** メニューでは、**C1PrintPreviewControl**を親コンテナにすばやく簡単にドッキングしたり、**スマートデザイナ**を有効にしたり、[**ローカライズ**]ダイアログボックスにアクセスすることができます。

**C1PrintPreviewControl タスク** メニューにアクセスするには、コントロールの右上隅にあるスマートタグ（）をクリックします。**C1PrintPreviewControl タスク** メニューが表示されます。


Copyright © GrapeCity, Inc. All rights reserved.




**C1PrintPreviewControl タスク** メニューは次のように動作します。

- **親コンテナでドッキングを解除する**

  [**親コンテナでドッキングを解除する**]をクリックすると、C1PrintPreviewControl の **Dock** プロパティが **Fill** に設定されます。

  **C1PrintPreviewControl** が親コンテナにドッキングされている場合は、親コンテナから C1PrintPreviewControl をドッキング解除するオプションを使用できるようになります。[**親コンテナからドッキング解除**]をクリックすると、**C1PrintPreviewControl** の **Dock** プロパティが **None** に設定されます。

- **スマートデザイナの有効化**

  [**スマートデザイナの有効化**]チェックボックスをオンにすると、**C1PrintPreviewControl** を設計時に詳細に設定できるスマートデザイナが起動します。[スマートデザイナの有効化]チェックボックスはデフォルトでオンで、**スマートデザイナ**は有効になっています。スマートデザイナの詳細については、「**スマートデザイナ**」を参照してください。

- **ローカライズ**

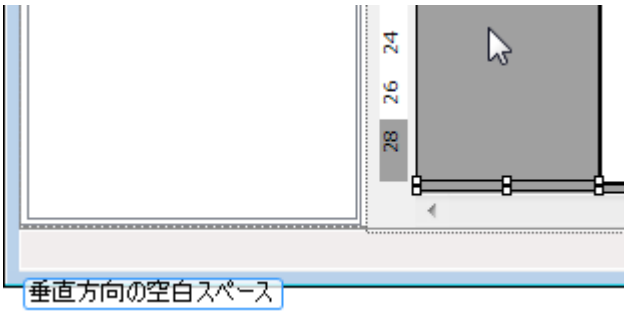
  [**ローカライズ**]ボタンをクリックすると、[**ローカライズ**]ダイアログボックスが表示されます。[**ローカライズ**]ダイアログボックスで、**ローカライズ**設定をカスタマイズできます。[**ローカライズ**]ダイアログボックスの詳細については、「ローカライズ」を参照してください。

- **バージョン情報**

  [**バージョン情報**]をクリックすると、コントロールの[**バージョン情報**]ダイアログボックスにコントロールのビルド番号が表示されます。.

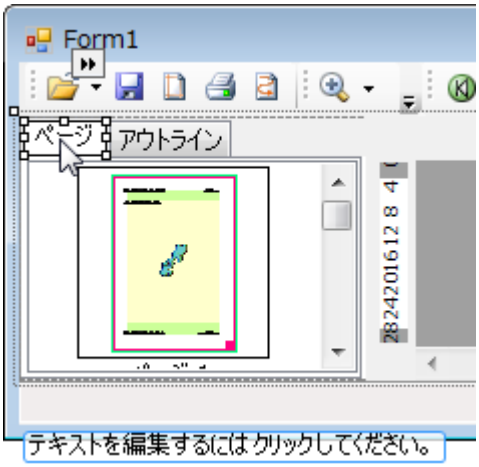
# スマートデザイナ

C1Preview には、設計時に詳細な設定を行うことができるスマートデザイナが用意されています。スマートデザイナを使用して、フォーム内で直接プロパティを設定できます。フォームの項目上にポインタが置かれると、フローティングツールバーと共にフォームの左下にタブが表示され、ポインタが置かれている項目がタブに示されます。




フローティングツールバーが関連付けられていない項目の場合は、その項目をフォーム内で直接カスタマイズするための方法が示されます。


Copyright © GrapeCity, Inc. All rights reserved.




スマートデザイナは、次のフローティングツールバーから成ります。

| ツールバー | 説明 |
| --- | --- |
| | **メインメニュー:** [メインメニュー]フローティングツールバーを使用して、プリセットツールバー画像の選択、プレビューコントロールのパネルの表示／非表示、ツールバーの表示／非表示、印刷オプションの設定、プレビューペインのツールチップの設定、エクスポートオプションの設定、およびプレビューのローカライズを行うことができます。 |
| | **ツールストリップ:** [ツールストリップ]フローティングツールバーを使用して、テキスト関連のプロパティ、画像、背景画像、項目の色、項目のレイアウトなどのプロパティを編集できます。 |
| | **サムネール :** [サムネール]フローティングツールバーを使用して、ナビゲーションパネルのサムネールビューの外観と動作を設定できます。 |
| | **アウトライン :** [アウトライン]フローティングツールバーを使用して、ナビゲーションパネルのアウトラインビューの外観と動作を設定できます。 |
| | **ルーラー :** [ルーラー]フローティングツールバーを使用して、水平／垂直ルーラーをそれぞれ表示／非表示にできます。 |
| | **プレビューペインの外観 :** [プレビューペインの外観]フローティングツールバーを使用して、プレビューペインのパディング、色、および境界線スタイルを設定できます。 |
| | **マージン :** [マージン]フローティングツールバーを使用して、ドキュメント |

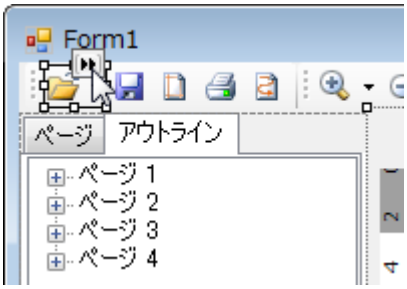


Copyright © GrapeCity, Inc. All rights reserved.

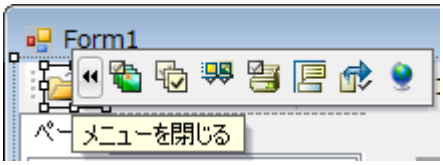
| ツールバー | 説明 |
|---|---|
| | ページのマージンを表示／非表示にできます。 |
| | **プレビューペイン :** [プレビューペイン]フローティングツールバーを使用して、プレビューペインのレイアウト、ズーム、動作の設定を制御できます。 |
| | **テキスト検索パネル :** [テキスト検索パネル]フローティングツールバーを使用して、テキスト検索パネルの外観と動作を制御できます。 |

## [メインメニュー]フローティングツールバー

ポインタが置かれている項目に関係なくフォーム内に表示されるフローティングツールバーは、[メインメニュー]だけです。フォームの左上隅に表示されるボタン をクリックすると、メインメニューを表示できます。




メインメニューを閉じるには、ボタンをクリックします。




メインメニューには、次のツールバーボタンがあります。

| ボタン | 説明 |
|---|---|
| | **ツールバー画像 :** プリセットツールバー画像を選択します。 |
| | **レイアウト :** プレビューコントロールのパネルを表示／非表示にします。 |
| | **ツールバーボタン :** ツールバーを表示／非表示にします。 |
| | **印刷 :** 印刷オプションを設定します。 |
| | **ツールチップ :** プレビューペインのツールチップを設定します |
| | **エクスポート :** エクスポートオプションを設定します。 |
| | **ローカライズ :** プレビューをローカライズします。 |

各ボタンをクリックすると、フォーム内の設定をカスタマイズするためのダイアログボックスが表示されます。


Copyright © GrapeCity, Inc. All rights reserved.

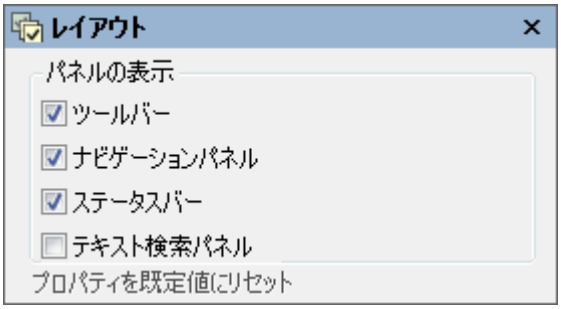
**注意**:ダイアログボックスで設定を変更すると、[**プロパティをデフォルトにリセット**]リンクが表示されます。このリンクをクリックすると、すべてのプロパティがデフォルト設定にリセットされます。

## ツールバー画像

[**ツールバー画像**]ボタンをクリックすると、[**ツールバー画像**]ダイアログボックスが表示され、ここでプリセットツールバー画像を選択できます。デフォルトのプリセットは[**XP**]です。

## レイアウト

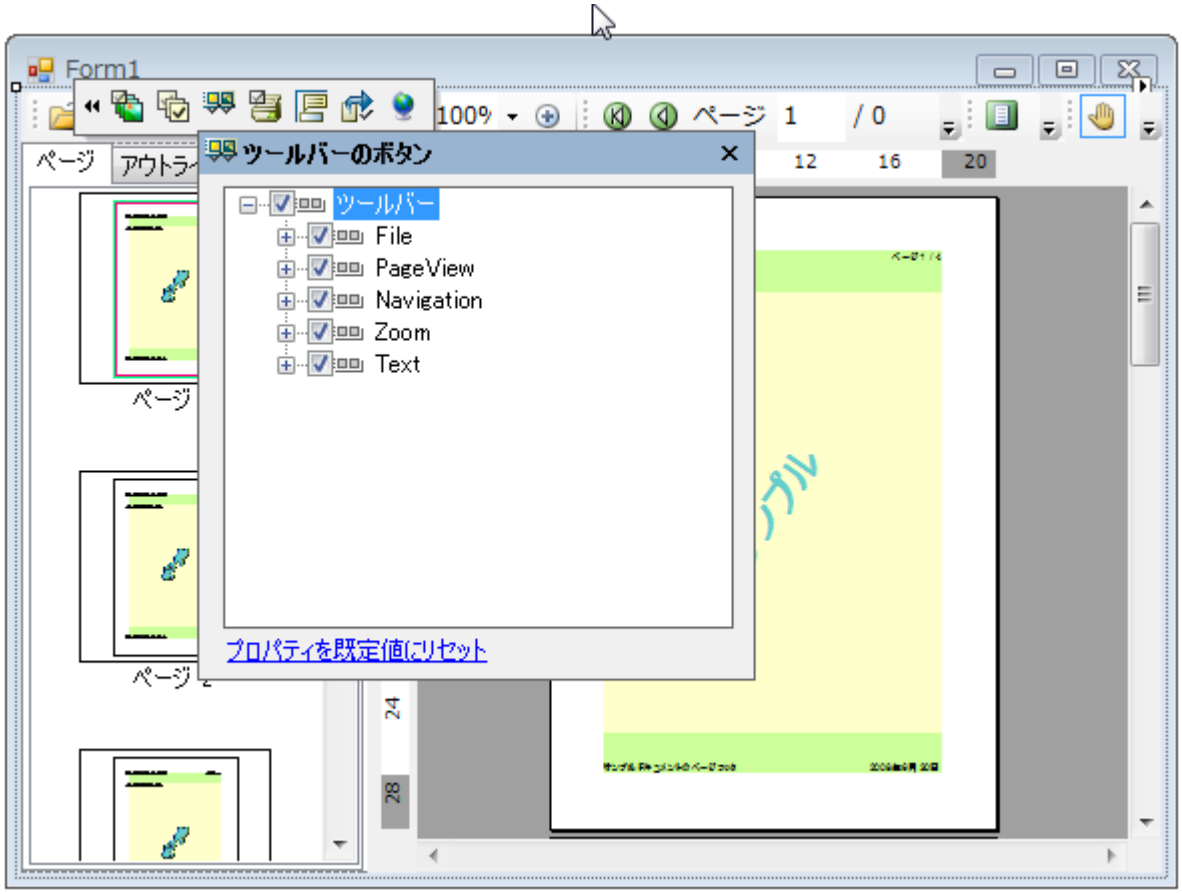
[**レイアウト**]ボタンをクリックすると、[**レイアウト**]ダイアログボックスが表示され、ここでプレビューコントロールのパネルを表示／非表示にできます。デフォルトでは、[ツールバー]、[ナビゲーションパネル]、および[ステータスバー]がオンになっています。




## ツールバーボタン

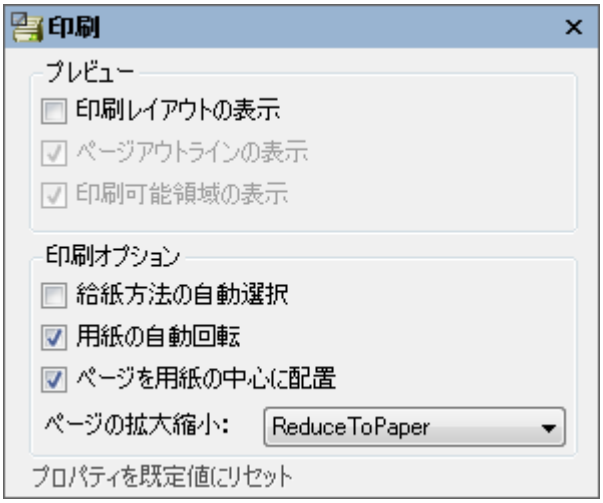
[**ツールバーボタン**]ボタンをクリックすると、[**ツールバーボタン**]ダイアログボックスが表示され、ここでツールバーやボタンを表示／非表示にできます。デフォルトでは、すべてのツールバーとボタンがオンになっています。


Copyright © GrapeCity, Inc. All rights reserved.




## 印刷

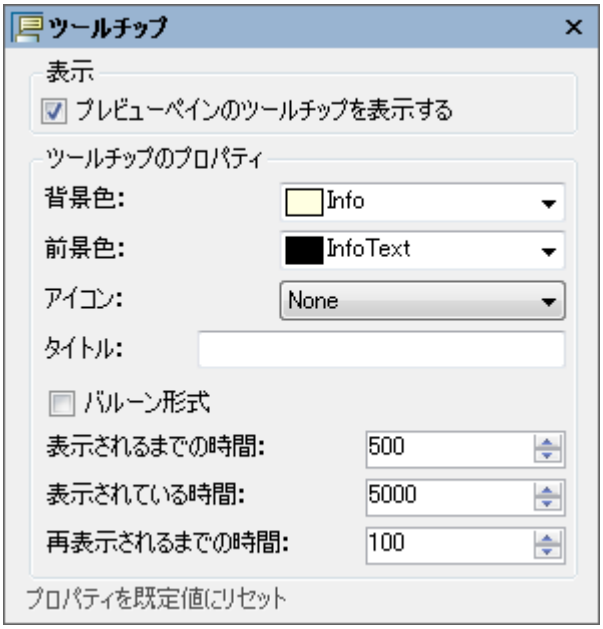
[**印刷**]ボタンをクリックすると、[**印刷**]ダイアログボックスが表示され、ここで印刷オプションを設定できます。デフォルトの設定は次のとおりです。




## ツールチップ

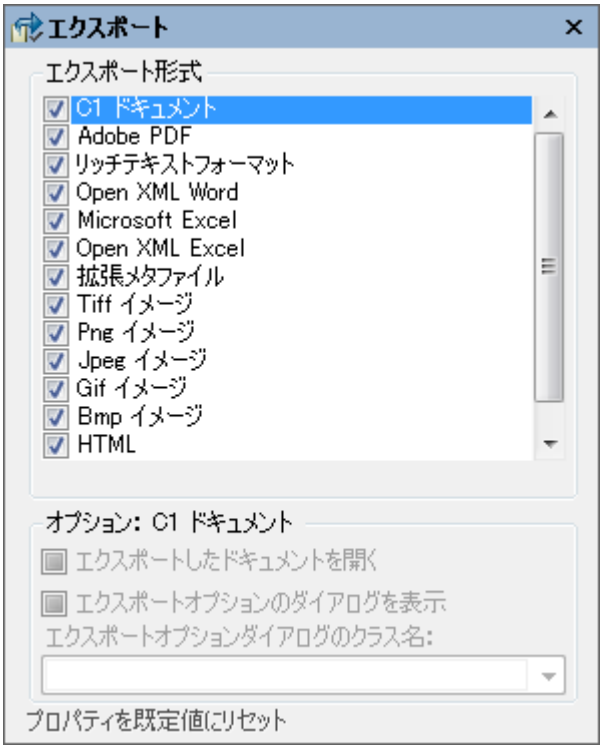
[**ツールチップ**]ボタンをクリックすると、[**ツールチップ**]ダイアログボックスが表示され、ここでプレビューペインのツールチップの設定をカスタマイズできます。デフォルトの設定は次のとおりです。


Copyright © GrapeCity, Inc. All rights reserved.




## エクスポート

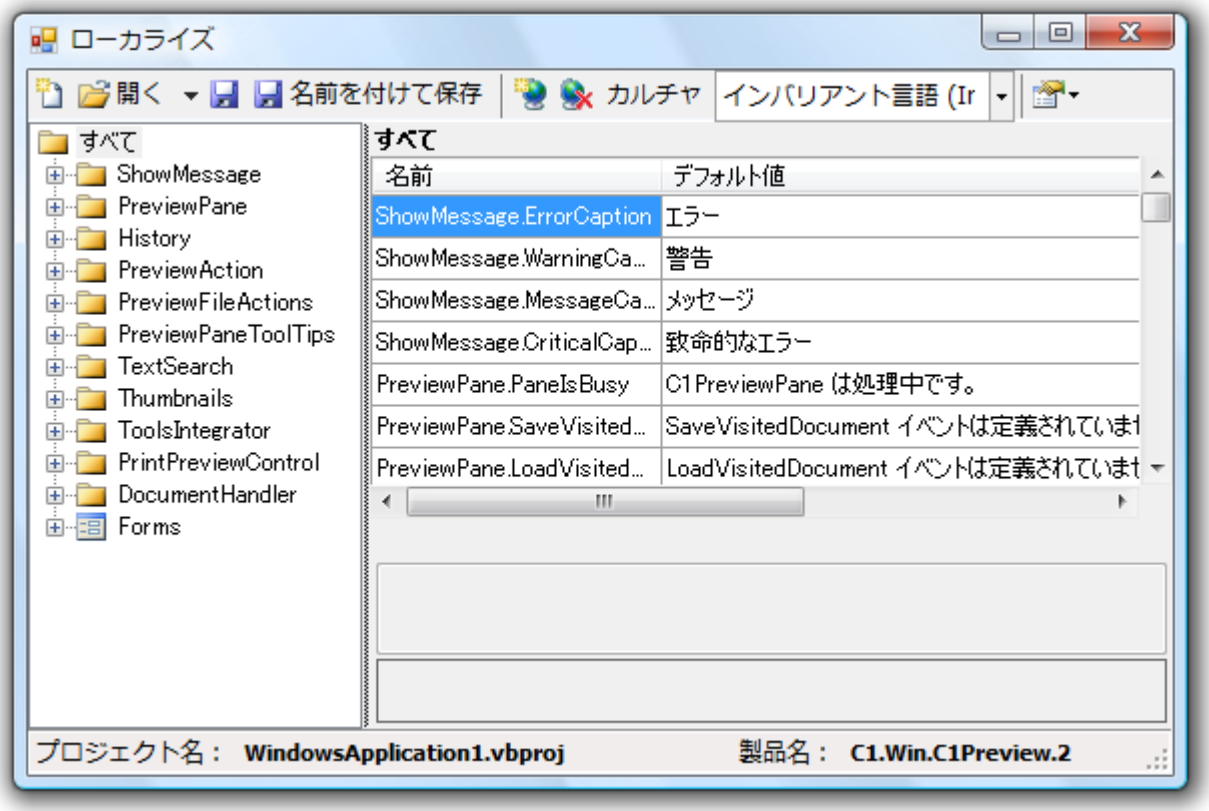
[**エクスポート**]ボタンをクリックすると、[**エクスポート**]ダイアログボックスが表示され、ここでエクスポートオプションを設定できます。デフォルトでは、すべての形式がオンになっています。




## ローカライズ

[**ローカライズ**]ボタンをクリックすると、[**ローカライズ**]ダイアログボックスが表示されます。[**ローカライズ**]ダイアログボックスで、ローカライズ設定をカスタマイズできます。


Copyright © GrapeCity, Inc. All rights reserved.




[**ローカライズ**]ダイアログボックスの詳細については、「ローカライズ」を参照してください。

## [ツールストリップ]フローティングツールバー

[ツールストリップ]フローティングツールバーは、ツールストリップ内の各項目に表示され、次のボタンがあります。

| ボタン | 説明 |
|---|---|
| | **テキスト：**テキスト関連のプロパティを編集します。 |
| | **画像：**画像および背景画像を編集します。 |
| | **色：**項目の色を編集します。 |
| | **レイアウト：**項目のレイアウトを編集します。 |
| | **プロパティ：**その他のプロパティを編集します。 |

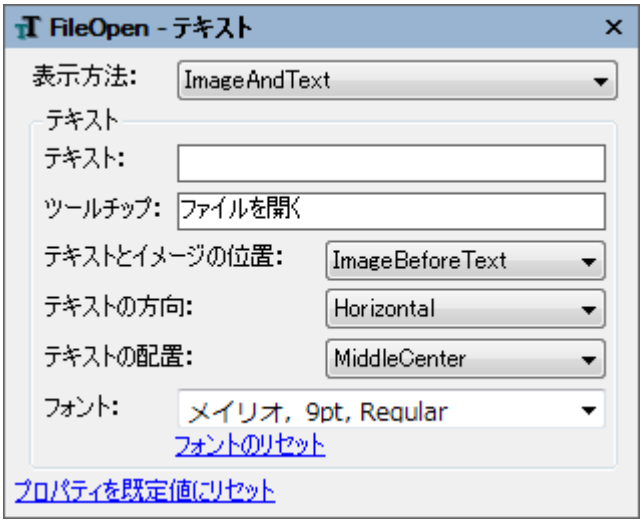
各ボタンをクリックすると、フォーム内の設定をカスタマイズするためのダイアログボックスが表示されます。

**注意**:ダイアログボックスで設定を変更すると、[**プロパティをデフォルトにリセット**]リンクが表示されます。このリンクをクリックすると、すべてのプロパティがデフォルト設定にリセットされます。

### Text

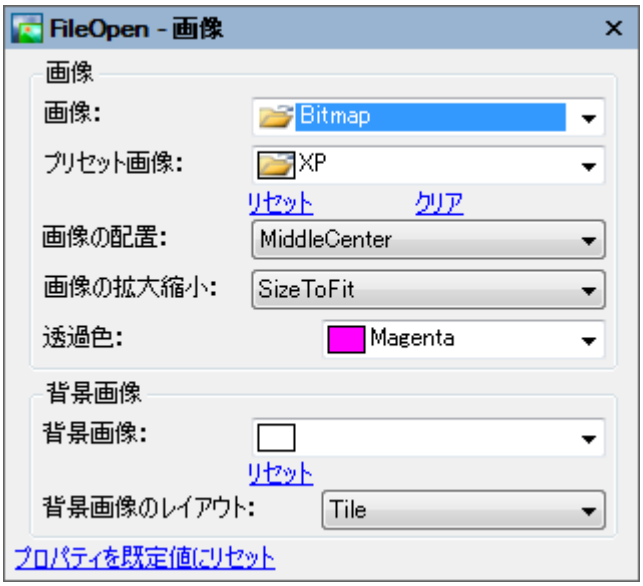
[**テキスト**]ボタンをクリックすると、[**テキスト**]ダイアログボックスが表示され、ここでこのツールバー項目のテキスト関連のプロパティを編集できます。


Copyright © GrapeCity, Inc. All rights reserved.



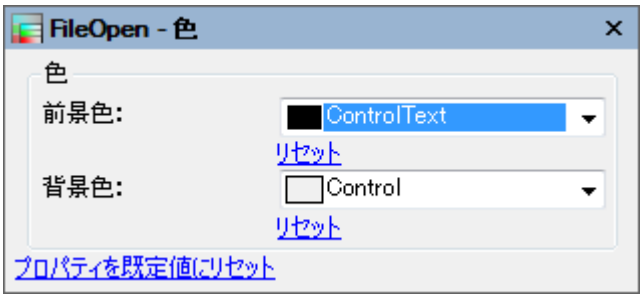

## 画像

[**画像**]ボタンをクリックすると、[**画像**]ダイアログボックスが表示され、ここでこのツールバー項目の画像および背景画像を編集できます。




## 色

[**色**]ボタンをクリックすると、[**色**]ダイアログボックスが表示され、ここでこのツールバー項目の色を編集できます。



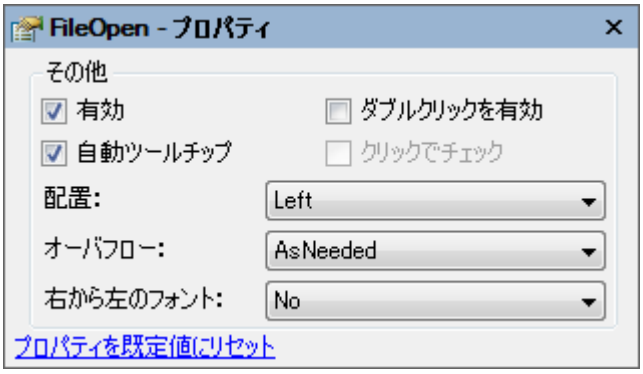

## レイアウト


Copyright © GrapeCity, Inc. All rights reserved.

[**レイアウト**]ボタンをクリックすると、[**レイアウト**]ダイアログボックスが表示され、ここでこのツールバー項目のレイアウトを編集できます。

## プロパティ

[**プロパティ**]ボタンをクリックすると、[**プロパティ**]ダイアログボックスが表示され、ここでこのツールバー項.目のその他のプロパティを編集できます。




# [サムネール]フローティングツールバー

[サムネール]フローティングツールバーは、ナビゲーションパネルのサムネールビュー内にポインタが置かれたときに表示されます。

| ボタン | 説明 |
|---|---|
| | **外観 :** サムネールビューの外観を編集します。 |
| | **動作 :** サムネールビューの動作を編集します。 |

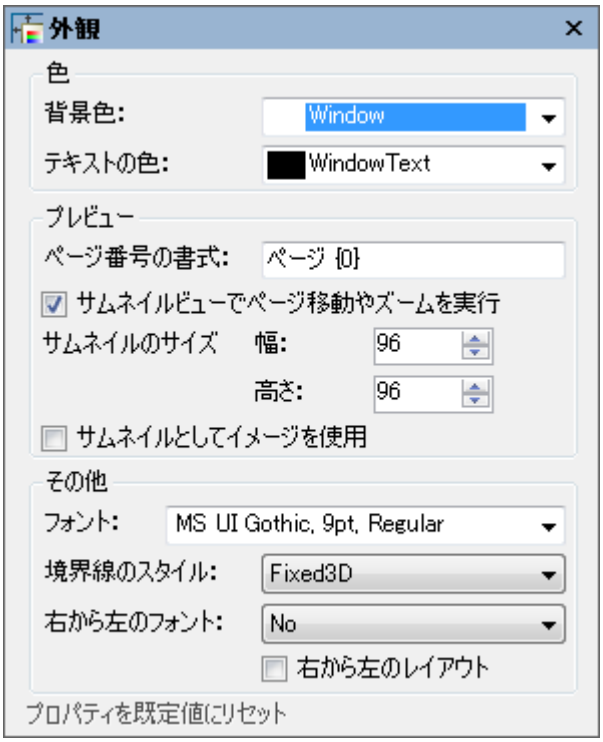
各ボタンをクリックすると、フォーム内の設定をカスタマイズするためのダイアログボックスが表示されます。

**注意**:ダイアログボックスで設定を変更すると、[**プロパティをデフォルトにリセット**]リンクが表示されます。このリンクをクリックすると、すべてのプロパティがデフォルト設定にリセットされます。


Copyright © GrapeCity, Inc. All rights reserved.

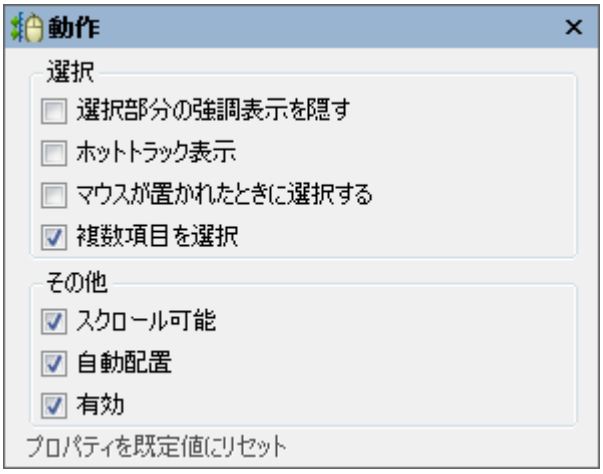
## 外観

[**外観**]ボタンをクリックすると、[**外観**]ダイアログボックスが表示され、ここでサムネールビューの色、サムネール設定、およびフォントを設定できます。




## 動作

[**動作**]ボタンをクリックすると、[**動作**]ダイアログボックスが表示され、ここでサムネールビューの動作設定を編集できます。




# [アウトライン]フローティングツールバー

[アウトライン]フローティングツールバーは、ナビゲーションパネルのアウトラインビュー内にポインタが置かれたときに表示されます。

| ボタン | 説明 |
|---|---|
| | **外観**：アウトラインビューの外観を編集します。 |


Copyright © GrapeCity, Inc. All rights reserved.

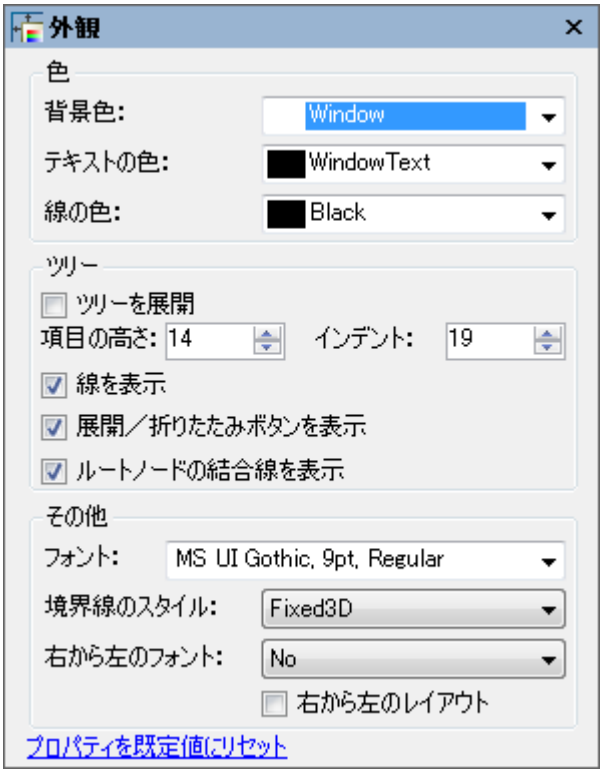
| ボタン | 説明 |
| --- | --- |
| | **動作**：アウトラインビューの動作を編集します。 |

各ボタンをクリックすると、フォーム内の設定をカスタマイズするためのダイアログボックスが表示されます。

**注意**:ダイアログボックスで設定を変更すると、[**プロパティをデフォルトにリセット**]リンクが表示されます。このリンクをクリックすると、すべてのプロパティがデフォルト設定にリセットされます。

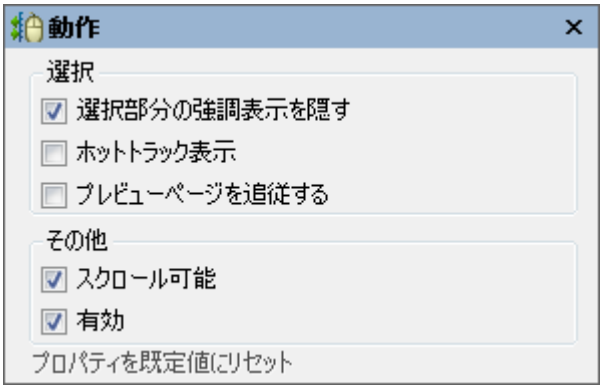
## 外観

[**外観**]ボタンをクリックすると、[**外観**]ダイアログボックスが表示され、ここでアウトラインビューの色、ツリー設定、およびフォントを設定できます。




## 動作

[**動作**]ボタンをクリックすると、[**動作**]ダイアログボックスが表示され、ここでアウトラインビューの動作設定を編集できます。





Copyright © GrapeCity, Inc. All rights reserved.

# [ルーラー]フローティングツールバー

[ルーラー]フローティングツールバーは、ポインタが水平または垂直ルーラー内にあるときに表示されます。

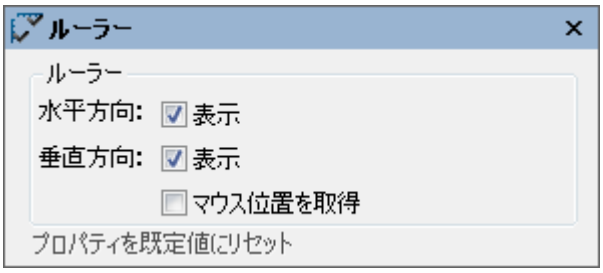
| ボタン | 説明 |
|---|---|
| | **ルーラー :** ルーラーを表示／非表示にします。 |

このボタンをクリックすると、フォーム内の設定をカスタマイズするためのダイアログボックスが表示されます。

> **注意**:ダイアログボックスで設定を変更すると、[**プロパティをデフォルトにリセット**]リンクが表示されます。このリンクをクリックすると、すべてのプロパティがデフォルト設定にリセットされます。

## ルーラー

[**ルーラー**]ボタンをクリックすると、[**ルーラー**]ダイアログボックスが表示され、ここでルーラーを有効／無効にしたり、マウストラッキングを設定することができます。




> **注意**:少なくとも1つのルーラーが表示されていないと、[**ルーラー**]ダイアログボックスは使用できません。ルーラーが表示されていない場合は、プレビューペインメニューの[**動作**]ダイアログボックスで[**ルーラー**]の[**水平**]および[**垂直**]チェックボックスをオンにしてください。

# [プレビューペインの外観]フローティングツールバー

[プレビューペインの外観]フローティングツールバーは、ポインタが水平または垂直ページパディング内にあるときに表示されます。

| ボタン | 説明 |
|---|---|
| | **外観 :** プレビューペインの外観を編集します。. |

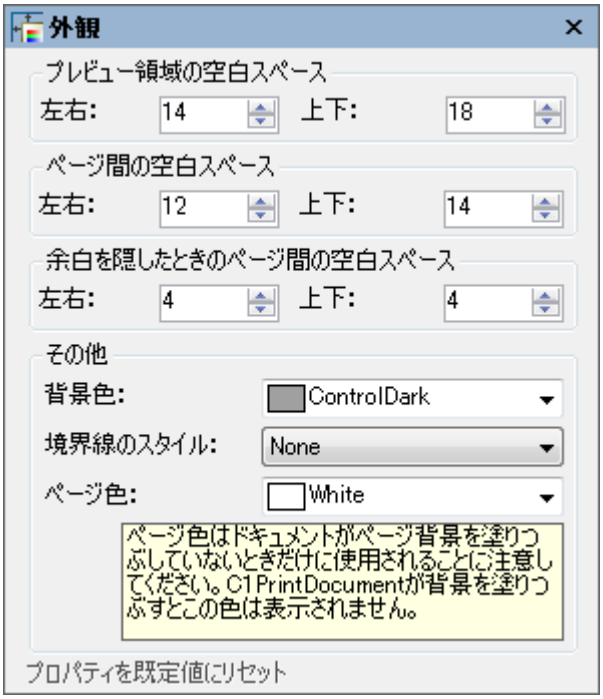
このボタンをクリックすると、フォーム内の設定をカスタマイズするためのダイアログボックスが表示されます。

> **注意**:ダイアログボックスで設定を変更すると、[**プロパティをデフォルトにリセット**]リンクが表示されます。このリンクをクリックすると、すべてのプロパティがデフォルト設定にリセットされます。

## 外観

**外観**ボタンをクリックすると、[**外観**]ダイアログボックスが表示され、ここでプレビューペインのパディング、色、および境界線のスタイルを設定できます。


Copyright © GrapeCity, Inc. All rights reserved.




## [マージン]フローティングツールバー

[マージン]フローティングツールバーは、ポインタが水平または垂直マージン内にあるときに表示されます。

| ボタン | 説明 |
|---|---|
| | **マージン**:ドキュメントページのマージンを表示／非表示にします。 |

このボタンをクリックすると、フォーム内の設定をカスタマイズするためのダイアログボックスが表示されます。

**注意**:ダイアログボックスで設定を変更すると、[**プロパティをデフォルトにリセット**]リンクが表示されます。このリンクをクリックすると、すべてのプロパティがデフォルト設定にリセットされます。

### マージン

[**マージン**]ボタンをクリックすると、[**マージン**]ダイアログボックスが表示され、ここでドキュメントページのマージンの表示／非表示を設定できます。

## [プレビューペイン]フローティングツールバー

[プレビューペイン]フローティングツールバーは、ポインタがページ本体内にあるときに表示されます。


Copyright © GrapeCity, Inc. All rights reserved.

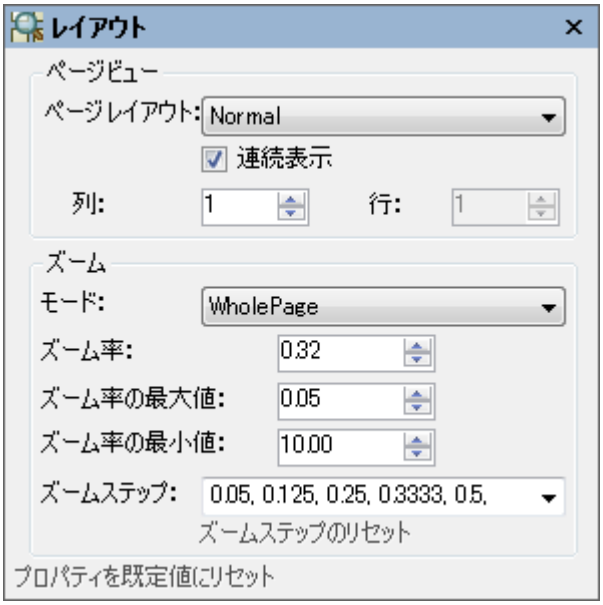
| ボタン | 説明 |
| --- | --- |
| | **レイアウト :** プレビューペインのレイアウトとズーム設定を編集します。 |
| | **動作 :** プレビューペインの動作を編集します。 |

各ボタンをクリックすると、フォーム内の設定をカスタマイズするためのダイアログボックスが表示されます。

> **注意**:ダイアログボックスで設定を変更すると、[プロパティをデフォルトにリセット]リンクが表示されます。このリンクをクリックすると、すべてのプロパティがデフォルト設定にリセットされます。

## レイアウト

[**レイアウト**]ボタンをクリックすると、[**レイアウト**]ダイアログボックスが表示され、ここでプレビューペインのレイアウトとズーム設定を編集できます。




## 動作

[**動作**]ボタンをクリックすると、[**動作**]ダイアログボックスが表示され、ここでプレビューペインの動作設定を編集できます。


Copyright © GrapeCity, Inc. All rights reserved.

# [テキスト検索パネル]フローティングツールバー

[テキスト検索パネル]フローティングツールバーは、ポインタがテキスト検索パネル上にあるときに表示されます。

| ボタン | 説明 |
|---|---|
| | **外観**：テキスト検索パネルの外観を編集します。 |
| | **動作**：テキスト検索パネルの動作を編集します。 |

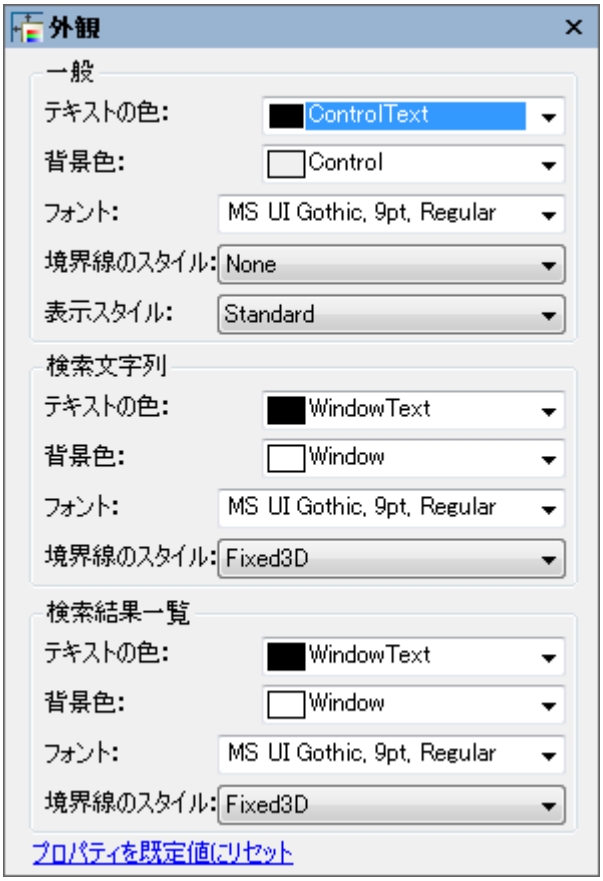
各ボタンをクリックすると、フォーム内の設定をカスタマイズするためのダイアログボックスが表示されます。

**注意**:ダイアログボックスで設定を変更すると、[**プロパティをデフォルトにリセット**]リンクが表示されます。このリンクをクリックすると、すべてのプロパティがデフォルト設定にリセットされます。

## 外観

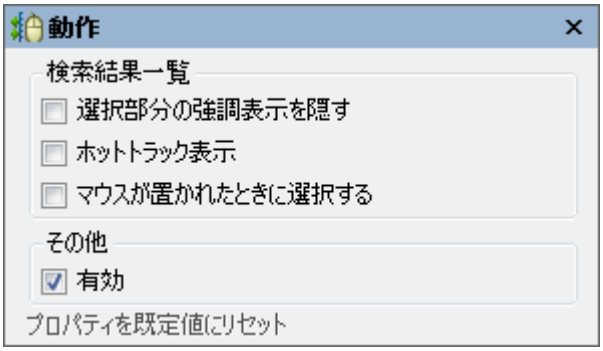
[**外観**]ボタンをクリックすると、[**外観**]ダイアログボックスが表示され、ここでテキスト検索パネルの色、ツリー設定、およびフォントを設定できます。


Copyright © GrapeCity, Inc. All rights reserved.




## 動作

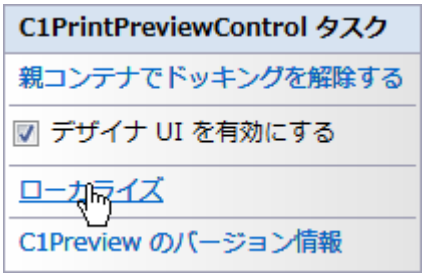
[**動作**]ボタンをクリックすると、[**動作**]ダイアログボックスが表示され、ここでテキスト検索パネルの動作設定を編集できます。



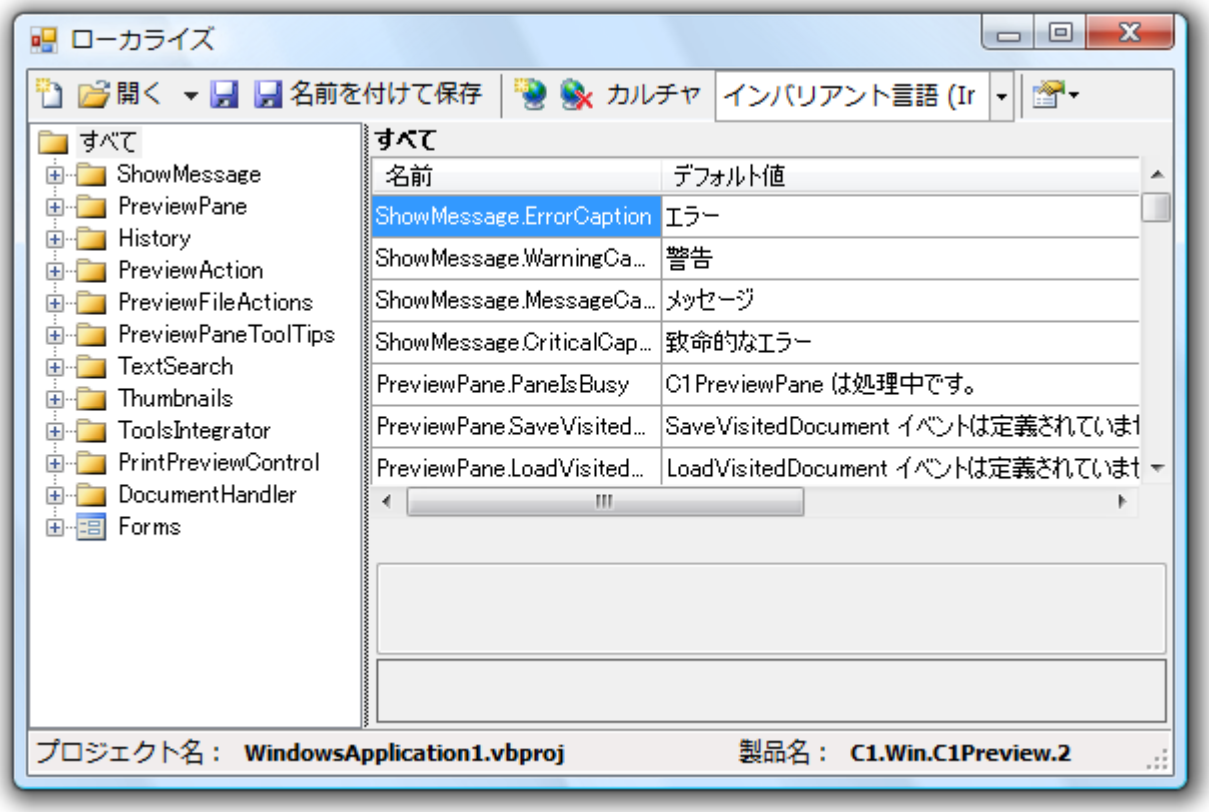

# ローカライズ

アプリケーション内の C1Preview コンポーネントをローカライズするには、C1PrintPreviewControl タスクメニュー、[メインメニュー]フローティングツールバー、または C1Preview コンポーネント(**C1PrintDocument**、**C1PrintPreview**、**C1PreviewPane**)のコンテキストメニューの[**ローカライズ**]をクリックします。


Copyright © GrapeCity, Inc. All rights reserved.




[**ローカライズ**]ボタンをクリックすると、[**ローカライズ**]ダイアログボックスが表示されます。




[**ローカライズ**]ダイアログボックスを使用して、いずれかの C1Preview アセンブリ(C1.C1Preview.2.dll または C1.Win.C1Preview.2.dll)をローカライズし、ローカライズしたリソースをソリューションの任意のプロジェクトに保存できます。

[**ローカライズ**]ダイアログボックスの左側のツリーには、ローカライズ可能な文字列の ID がリストされ、右側には、それらの文字列が表示されます。このツリー構造は、Strings クラスのサブクラスの階層を反映しています。右パネルには、すべての文字列を表示することも、選択したツリーノードに属する文字列だけを表示することもできます。

この文字列リストには次の列があります。

| 列 | 説明 |
|---|---|
| **名前** | 文字列の名前(ID)。この列には、選択されているツリーノードが繰り返し表示されます。非表示にすることもできます。 |
| **デフォルト値** | 文字列のデフォルト(英語)値。 |
| **値** | 現在選択されているカルチャ(列ヘッダーに表示)に対応する文字列値。 |

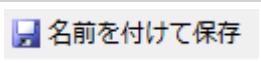
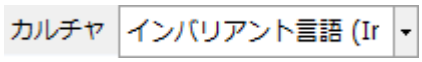
リストの下には、現在選択されている文字列の値と、オプションで説明が表示されます。

ステータスバーには、ローカライズされたリソースが含まれるプロジェクトと、現在ローカライズされている C1 アセンブリの名前が表示されます。


Copyright © GrapeCity, Inc. All rights reserved.

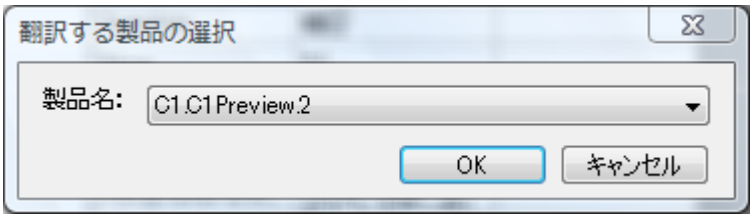
# ローカライズツールバー

[**ローカライズ**]ダイアログボックスには、次のツールバーメニューボタンがあります。

| ボタン | 説明 |
|---|---|
| | [**新しい翻訳の作成**]は、ComponentOne アセンブリの新しいローカライズを開始します。 |
| | [**開く**]は、指定されたアセンブリの既存の翻訳を開きます。 |
| | [**保存**]は、現在の翻訳を保存します。 |
|  | [**名前を付けて保存**]は、現在の翻訳を保存します。また、保存先のプロジェクトを選択できます。 |
| | [**カルチャの追加**]は、新しいカルチャを追加します。 |
| | [**カルチャの削除**]は、翻訳からカルチャを削除します。 |
|  | [**カルチャの選択**]は、表示して編集するカルチャを選択します。 |
| | [**オプション**]は、ローカライズウィンドウの外観と動作をカスタマイズします。 |

## 新しい翻訳の作成

[**新しい翻訳の作成**]ボタンをクリックすると、ComponentOne アセンブリの新しいローカライズが開始されます。ローカライズする ComponentOne アセンブリを選択するためのダイアログボックスが表示されます。



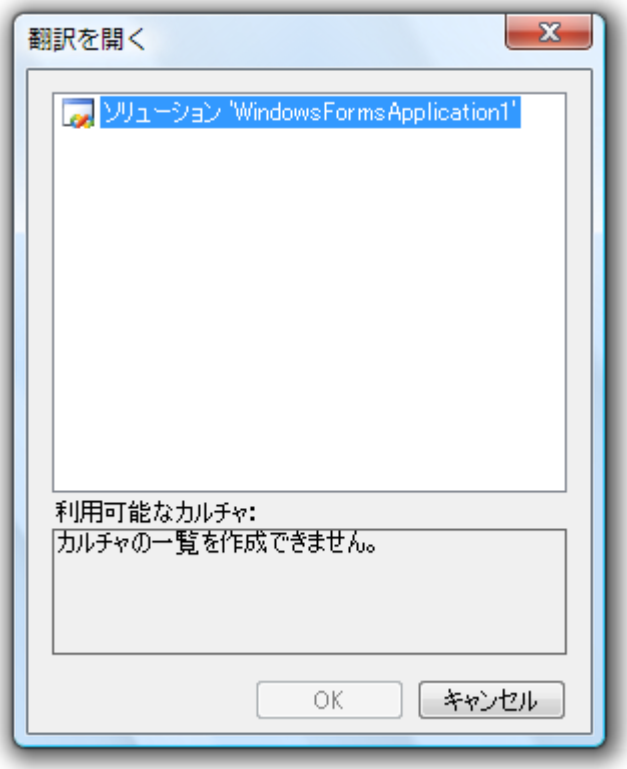

**注意**:このアセンブリは、現在開いているソリューション内で参照されている必要があります。

## 開く

[**開く**]ボタンをクリックすると、指定したアセンブリの既存の翻訳を開くことができます。作成した翻訳はすべて .resx ファイルとして保存され、翻訳の保存時に、選択したプロジェクトに自動的に追加されます。この項目をクリックすると、以前に保存した翻訳を選択するためのダイアログボックスが表示されます。

新しいソリューションを作成した場合は、翻訳が含まれないため、最初は次のようなウィンドウが表示されます。


Copyright © GrapeCity, Inc. All rights reserved.




翻訳を作成して保存した後は、[使**用可能なカルチャ**]パネルに、選択したアセンブリに対して翻訳が作成されているカルチャのリストが表示されます。

## 保存

[**保存**]ボタンをクリックすると、現在の翻訳を保存できます。

翻訳は、ステータスバーに表示されているプロジェクトに保存されます。翻訳を保存すると、選択されているプロジェクトに **C1LocalizedResources** という名前のフォルダが作成されます(まだ存在しない場合)。翻訳の .resx ファイルはこのフォルダに保存され、プロジェクトに追加されます。カルチャごとに異なる .resx ファイルが作成されます。これらのファイルはソリューションエクスプローラウィンドウに表示されます。

**注意**:翻訳が不変カルチャだけに使用される場合、.resx ファイルの名前の末尾にカルチャ名は付きません。

## 名前を付けて保存

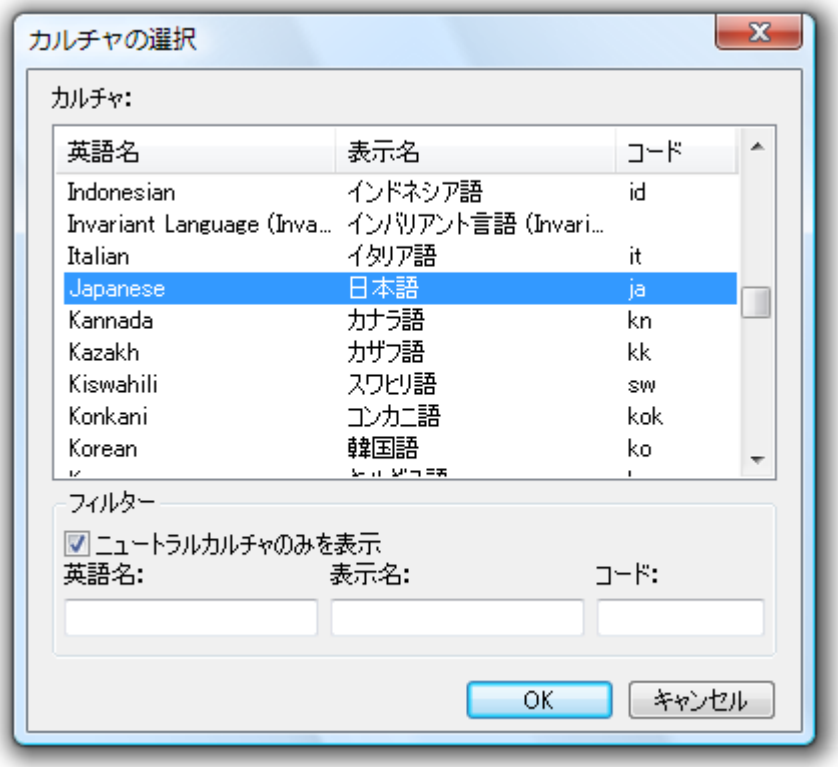
[**名前を付けて保存**]ボタンをクリックすると、現在の翻訳が保存されます。また、保存先のプロジェクトを選択できます。

## カルチャの追加

[**カルチャの追加**]ボタンをクリックすると、新しいカルチャを追加できます。

複数のカルチャに対して翻訳を作成し、それらを実行時に動的に切り替えることができます。カルチャごとに異なる .resx ファイルが **C1LocalizedResources** フォルダに作成されます。[**カルチャの追加**]ボタンをクリックすると、[**カルチャの選択**]ダイアログボックスに使用可能なカルチャのリストが表示されます。

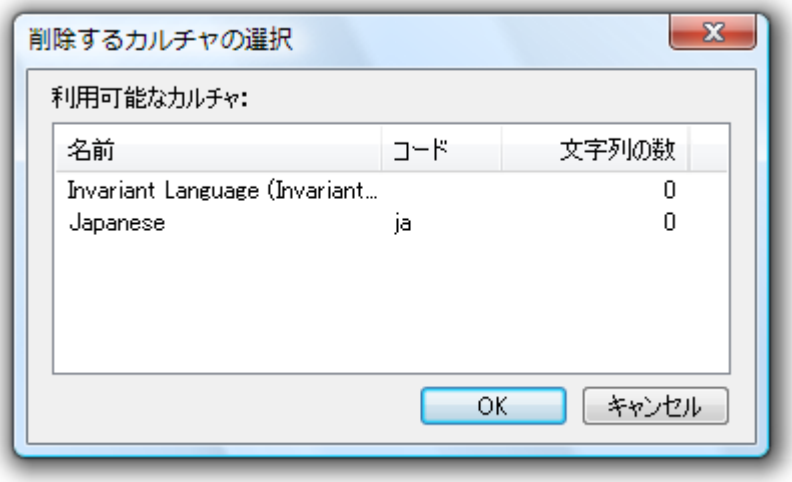

Copyright © GrapeCity, Inc. All rights reserved.




最初、このリストには中立的カルチャだけが表示されます。すべてのカルチャを表示するには、[**中立的カルチャだけを表示**]チェックボックスをオフにします。[**英語名**]、[**表示名**]、および[**コード**]ボックスを使用して、表示されるカルチャをフィルタできます。カルチャを選択したら、[**OK**]ボタンをクリックして翻訳に追加します。新しく追加したカルチャは、ツールバーの[**カルチャ**]ドロップダウンに表示され、ウィンドウの現在のカルチャになります。

## カルチャの削除

[**カルチャの削除**]ボタンをクリックすると、翻訳からカルチャを削除できます。[**削除するカルチャの選択**]ダイアログボックスに、翻訳に含まれる既存のカルチャのリストが表示されます。




カルチャを選択し、[**OK**]をクリックすると、翻訳から削除されます。

## カルチャの選択

[**カルチャ**]ドロップダウンを使用すると、表示して編集するカルチャを選択できます。


Copyright © GrapeCity, Inc. All rights reserved.

## オプション

[**オプション**]ボタンをクリックすると、ローカライズウィンドウの外観と動作をカスタマイズできます。次のローカライズオプションがあります。

| オプション | 説明 |
|---|---|
| **ツリーとグリッドの同期** | このオプションをオンにした場合は、右パネルのリストで文字列を選択すると、左側のツリーでも文字列が選択されます。デフォルトでは、このオプションはオフです。 |
| **グリッドに名前を表示** | このオプションをオンにすると、右パネルに[*名前*]列が表示されます。オフにすると非表示になります。デフォルトでは、このオプションはオンです。 |
| **選択したグループだけを表示** | このオプションをオンにした場合、右パネルの文字列のリストには、左側のツリーで現在選択されているグループ内の文字列だけが表示されます。デフォルトでは、このオプションはオフです。 |


Copyright © GrapeCity, Inc. All rights reserved.

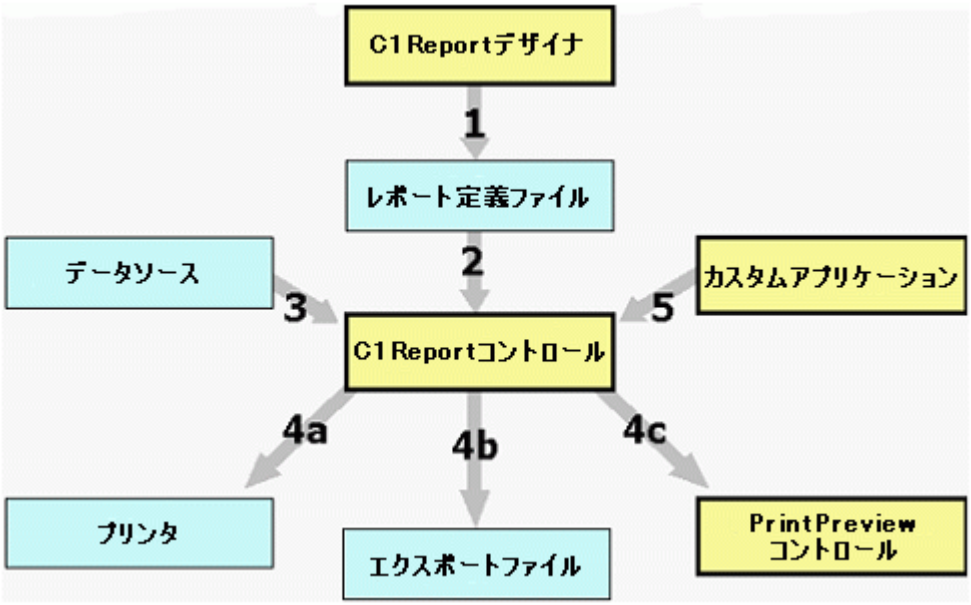
# C1Report の使い方

**C1Report** は、デスクトップおよび Web のさまざまな状況で利用できます。どの場合でも、主な手順は同じです。

1. レポートを作成するには、はじめに **C1Report デザイナ**を使用してレポート定義を作成します。レポート定義は新規に作成するか、Microsoft Access レポートをインポートすることができ、XML ファイルとして保存されます。
2. 次に、C1Report コンポーネントを使用してレポート定義を読み込み、任意の標準的な .NET データソースからデータを取得してレポートを出力します。
3. レポート定義は、設計時に読み込んでアプリケーションに埋め込んだり、実行時に読み込んだり変更したりすることができます(C1Report オブジェクトモデルを使用して、新規にレポート定義を作成することも可能です)。
4. 作成したレポートは、直接プリンタに出力したり、**C1PrintPreview** コントロールに描画したり、Web 配信に適している HTML 形式や PDF 形式でエクスポートすることができます。

次の図は、**C1Report** におけるさまざまなコンポーネントの関係を示します。

**メモ**:太線で囲まれた四角形は、コードコンポーネント(コントロール、アプリケーション)を表します。細線で囲まれた四角形は、情報を含むファイル(レポート定義/データ/作成されたレポート)を表します。番号付きの矢印は、コンポーネント間の関係(以下で説明)を示します。




次の各番号は、コンポーネント間の関係を示す図中の番号付き矢印を指しています。

1. **C1Report デザイナ**を使用して、XML 形式のレポート定義ファイルを作成、編集、および保存します。
2. **C1Report** コンポーネントに、デザイナで作成されたレポート定義ファイルを読み込みます。これは設計時にプロパティページを使用して行うか、実行時に**Load** メソッドを使用して行うことができます。プロパティページからレポートを読み込んだ場合、レポート定義ファイルはコントロールに保存されるので、実行時にはレポート定義ファイルを必要としません。
3. C1Report コンポーネントは、レポート定義ファイルで指定されたデータソースから自動的にデータを読み込みます。これを実行する代わりに、独自のカスタムデータソースを提供することもできます。
4. C1Report コンポーネントは、レポート定義に従ってデータをフォーマットし、レポートをプリンタ(a)、いずれかのファイル形式(b)、または印刷プレビューコントロール(c)に出力します。
5. カスタムアプリケーションにより、豊富なオブジェクトモデルを使用して C1Report コンポーネントと交信し、レポートを簡単にカスタマイズしたり、レポートを一から作成したりすることができます。

# オブジェクトモデル

次の一覧は、**C1Report** コンポーネントのオブジェクトモデルをまとめたものです。このオブジェクトモデルは、主に Microsoft


Copyright © GrapeCity, Inc. All rights reserved.

Access モデルに基づいていますが、Microsoft Access では異なるタイプのコントロール（ラベルコントロール、テキストボックスコントロール、ラインコントロールなど）が存在するのに対し、C1Report に存在するのは単一の**Field**オブジェクトです。C1Report では、オブジェクトに含まれるプロパティの設定によって、ラベル、テキストボックス、線、画像、サブレポートなどの外観に見せています。

以下に、すべてのオブジェクトのリストと、それらに含まれるプロパティやメソッドを示します。

> **メモ:**C1Report では、すべての測定の単位に Twips を使用します。1Twips は 1/20 ポイントです。したがって、72 ポイント = 1440 Twips = 1インチになります。

| |
|---|
| **C1Report オブジェクト:** メインのコンポーネントです。 |
| **ReportName**, **GetReportInfo**, **Load**, **Save**, **Clear**, **Render**, **RenderToFile**, **RenderToStream**, **PageImages**, **Document**, **DoEvents**, **IsBusy**, **Cancel**, **Page**, **MaxPages**, **Font**, **OnOpen**, **OnClose**, **OnNoData**, **OnPage**, **OnError**, **Evaluate**, **Execute** |
| **Layout Object:** ページ上でのレポートの出力方法を指定するオブジェクトです。 |
| **Width**, **MarginLeft**, **MarginTop**, **MarginRight**, **MarginBottom**, **PaperSize**, **Orientation**, **Columns**, **ColumnLayout**, **PageHeader**, **PageFooter**, **Picture**, **PictureAlign**, **PictureShow** |
| **DataSource オブジェクト:** データソースを管理するオブジェクトです。 |
| **ConnectionString**, **RecordSource**, **Filter**, **MaxRecords**, **Recordset** |
| **Groups コレクション:** レポートには複数のグループを含めることができます。 |
| **Group オブジェクト:** データのソートとグループ化を制御するオブジェクトです。 |
| **Name GroupBy**, **Sort**, **KeepTogether**, **SectionHeader**, **SectionFooter**, **Move** |
| **Sections コレクション:** すべてのレポートには、少なくとも5つのセクションが含まれます。 |
| **Section オブジェクト:** Field オブジェクト（"レポート領域"とも呼ばれる）のコンテナです。 |
| **Name**, **Type**, **Visible**, **BackColor**, **OnFormat**, **OnPrint**, **Height**, **CanGrow**, **CanShrink**, **Repeat**, **KeepTogether**, **ForcePageBreak** |
| **Fields コレクション:** レポートには通常複数のフィールドが含まれます。 |
| **Field オブジェクト:** セクション内の情報が表示される四角形の領域です。 |
| **Name**, **Section**, **Text**, **TextDirection**, **Calculated**, **Value**, **Format**, **Align**, **WordWrap**, **Visible**, **Left**, **Top**, **Width**, **Height**, **CanGrow**, **CanShrink**, **Font**, **BackColor**, **ForeColor**, **BorderColor**, **BorderStyle**, **LineSlant**, **LineWidth**, **MarginLeft**, **MarginRight**, **MarginTop**, **MarginBottom**, **LineSpacing**, **ForcePageBreak**, **HideDuplicates**, **RunningSum**, **Picture**, **PictureAlign**, **Subreport**, **CheckBox**, **RTF** |

## レポートのセクション

すべてのレポートには、少なくとも5つのセクションが含まれます。これを次に示します。

| セクション | 説明 |
|---|---|
| Detail | 詳細セクション。ソースレコードセット内の各レコードに1度ずつ出力されるフィールドが含まれます。 |
| Header | レポートヘッダセクション。レポートの最初に出力されます。 |
| Footer | レポートフッタセクション。レポートの最後に出力されます。 |
| Page Header | ページヘッダセクション。各ページの上部（オプションによってはレポートヘッダを含むページを除く）に出力されます。 |

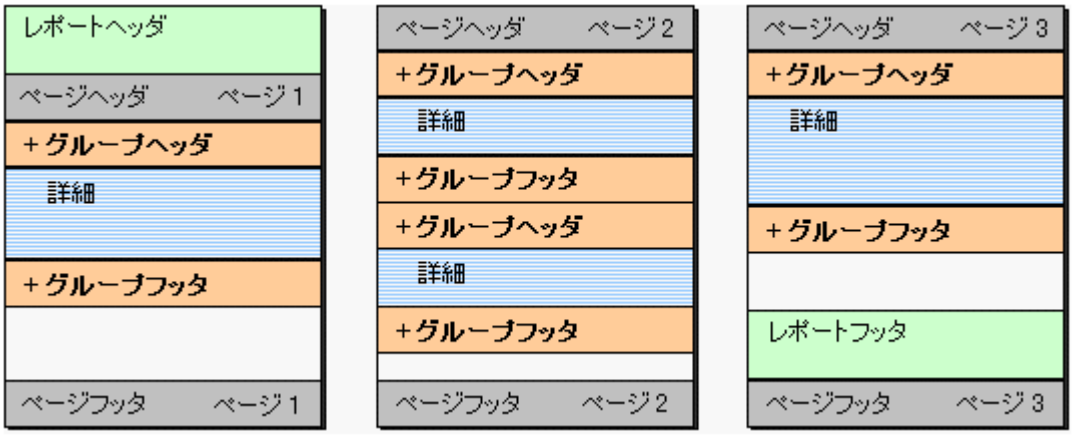

Copyright © GrapeCity, Inc. All rights reserved.

| セクション | 説明 |
| --- | --- |
| Page Footer | ページフッタセクション。各ページの下部（オプションによってはレポートフッタを含むページを除く）に出力されます。 |

これら5つのセクションに加えて、各グループに2つのセクション（グループヘッダとグループフッタ）が追加されます。たとえば、3つのグループレベルを持つレポートには、11 のセクションがあります。

これらのセクションを非表示にすることはできますが、グループを追加または削除しないかぎり、セクションを追加または削除することはできません。

下の図は、一般的なレポートにおける各セクションの出力方法を示しています。




## レポートヘッダ

最初に出力されるセクションはレポートヘッダです。このセクションには通常、レポートを識別する情報が含まれます。

## ページヘッダ

次に出力されるのは、ページヘッダです。レポート内にグループが定義されていない場合、このセクションには通常、詳細セクション内のフィールドについて説明するラベルが含まれます。

## グループヘッダ・フッタ

次に、グループヘッダ、詳細、グループフッタのセクションが続きます。これらが実際のレポートデータを含むセクションです。グループヘッダおよびフッタには通常、グループの合計、割合、最大値、最小値などの集計関数が含まれることがあります。グループヘッダおよびグループフッタは、**GroupBy** プロパティによって指定される式の値が、あるレコードから次のレコードへ変化するたびに挿入されます。

## 詳細

詳細セクションには、各レコードのデータが含まれます。Visible プロパティを **False** に設定に設定してこのセクションを非表示にし、グループヘッダおよびグループフッタのみを表示することもできます。これはサマリーレポートの作成に便利な機能です。

## ページフッタ

各ページの下部には、ページフッタセクションがあります。このセクションには通常、ページ番号、レポートの総ページ数、レポートの印刷日などの情報が含まれます。

## レポートフッタ


Copyright © GrapeCity, Inc. All rights reserved.

最後に、レポートフッタセクションが、最後のページフッタの前に出力されます。このセクションは、レポート全体に関するサマリー情報を表示するために使用されます。

### カスタムセクション

各セクションの表示／非表示は、**Visible** プロパティの設定値（**True**／**False**）を切り替えることで指定できます。グループヘッダは、**Repeat** プロパティを **True** に設定することによって、各ページ（グループの最初のページかどうかにかかわらず）の上部に対して繰り返し表示できます。ページヘッダおよびページフッタは、**Layout** オブジェクトの**PageHeader** プロパティと**PageFooter** プロパティを設定することによって、レポートヘッダセクションとレポートフッタセクションを含むページから削除できます。

## クライアントアプリケーションの開発

このセクションでは、一般的なデスクトップ環境で、C1Report をどのように使用できるかについて説明します。
一般的なデスクトップ環境では、レポートを生成および表示するのと同じコンピュータで C1Report を実行します。その場合でも、レポートのデータ自体は、リモートサーバーから取得されることがあります。

このような例では、**C1Report**が Visual Basic .NET や C# .NET アプリケーションでホストされることを前提とします。

## 組み込みレポート（設計時に読み込み）

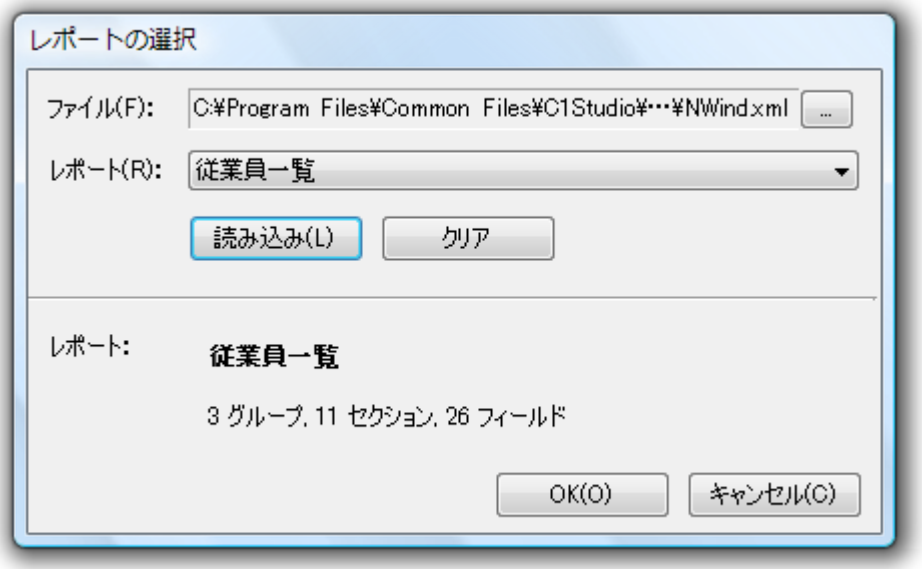
この例では、アプリケーションに組み込まれている一定のレポート定義セットを使ってレポートを生成します。このようなアプリケーションは、外部のレポート定義ファイルに依存しません。また、エンドユーザーがレポートを変更する方法はありません。

このようなアプリケーションの主な長所は、レポート定義ファイルを配布する必要がないことと、レポートの形式を変更されないようにできることです。短所は、アプリケーションを再コンパイルしないとレポートを変更できないことです。

組み込みレポートを備えたアプリケーションを作成するには、次の手順に従います。

1. **C1Report** デザイナを使用して、必要なレポートをすべて作成します（作成方法の詳細については、「C1Report デザイナの使い方」を参照）。
2. 配布するレポート定義ごとに C1Report コンポーネントを1つ追加します。各コントロールに、出力するレポートに応じた名前を付けると、コードの保守が簡単になります。
3. **C1Report** コンポーネントを右クリックし、「**レポート定義ファイルの読み込み**」メニューオプションを選択して、レポート定義をそれぞれのコントロールに読み込みます。C1Report コンポーネントの「レポート定義ファイルの読み込み」リンクをクリックすることもできます。
「レポートの選択」ダイアログボックスが表示されるので、レポート定義ファイルを選択し、さらにそのファイル内のレポートを選択します。レポートを読み込むには、[...]ボタンをクリックし、手順1で作成したレポート定義ファイルを選択します。次に、ドロップダウンリストからレポートを選択し、読み込みボタンをクリックします。プロパティページに、選択したレポート名、グループ数、セクション数、およびフィールド数が表示されます。ダイアログボックスは、次のように表示されます。


Copyright © GrapeCity, Inc. All rights reserved.




4. ユーザーが選択したレポートを出力するためのコードを追加します。たとえば、ボタングループを追加した場合は、次のようなコードを追加します。

**Visual Basic コードの書き方**

Visual Basic

```
Private Sub btnProductsReport_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnProductsReport.Click
  ppv.Document = rptProducts.Document
End Sub
```

**C# コードの書き方**

C#

```
private void btnProductsReport_Click(object sender, System.EventArgs e)
{
  ppv.Document = rptProducts.Document;
}
```

ここで、**rptProducts** は、ユーザーが選択したレポートを含む C1Report コンポーネントの名前、**ppv** は、**C1PrintPreview** コントロールの名前です。

# 組み込みレポート(設計時に作成)

「組み込みレポート(設計時に読み込み)」で説明した「**レポート定義ファイルの読み込み**」コマンドは、既に存在するレポート定義をアプリケーション内に読み込む場合に便利です。しかし、特定のケースでは、レポートをカスタマイズしたかったり、レポート定義の接続文字列やデータソースではなく Visual Studio アプリケーション内で定義したデータソースオブジェクトを使用したい場合があります。このような場合には、代わりに「レポート定義ファイルの編集」コマンドを使用します。

設計時にレポートを作成もしくは編集するには、**C1Report**コンポーネントを右クリックし、「**レポートの編集**」メニューオプションを選択して、**C1Report デザイナ**アプリケーションを起動します。C1Report コンポーネントから「**レポートの編集**」リンクをクリックすることもできます。

**メモ**:コントロールがレポートデザイナアプリケーションを見つけられなかった場合は、「**レポートの編集**」コマンドがコンテキストメニューやプロパティウィンドウ上に表示されません。これを修復するには、単独で **C1Report デザイナ**を起動してください。デザイナはレジストリにその場所を記録するので、C1Report コントロールはデザイナを見つけることができ


Copyright © GrapeCity, Inc. All rights reserved.

るようになります。

**C1Report デザイナ**は、現在 C1Report コンポーネントに読み込まれているレポートを表示します。C1Report コンポーネントが空の場合、デザイナは新規にレポートを作成するためのレポートウィザードを表示します。

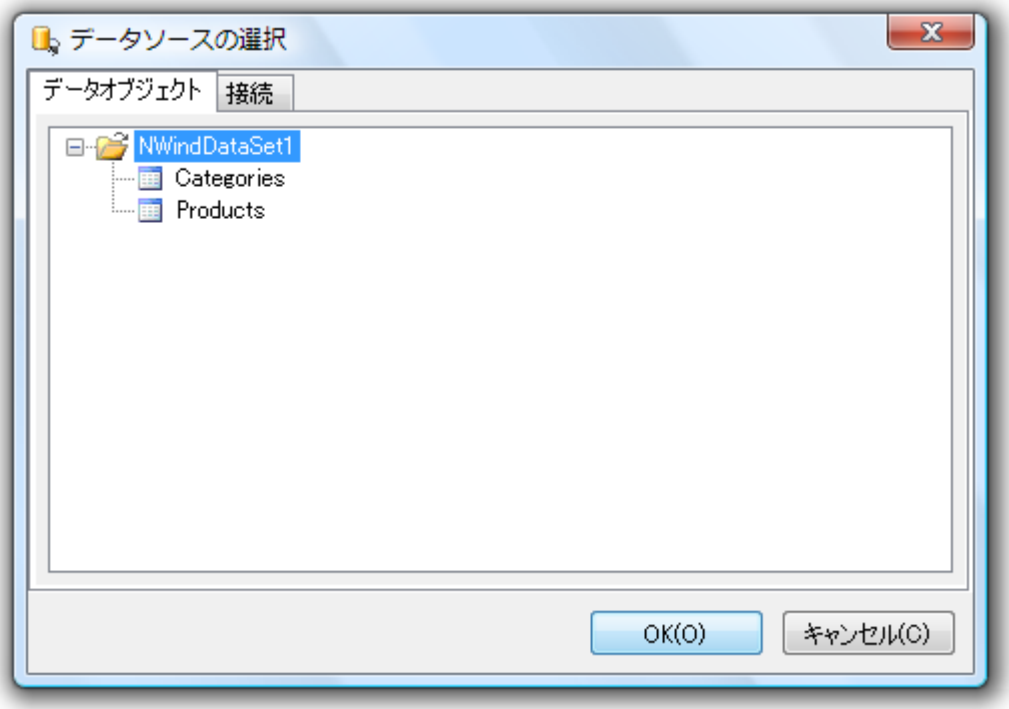
これは、**C1Report デザイナ**を単独で使用した場合と同じです。このモードで起動した場合の唯一の違いは以下の通りです。

- 新規レポートのデータソースとしてアプリケーション内で定義したデータソースオブジェクトを使用することができます。
- デザイナを終了するとき、すべての変更内容をフォーム上の C1Report コンポーネント内に書き戻すことができます。デザイナ内で、[**ファイル**｜**変更内容を破棄して終了**]を選択することで、変更した内容を 保存せずに終了します。

アプリケーション内で定義したデータソースオブジェクトを使用する場合は、デザイナ上の「**データソース**」ボタン をクリックし、データソースダイアログ内の[**データオブジェクト**]ページを選択します。

[**データオブジェクト**]ページには、フォーム上に現在定義されているデータオブジェクトのリストが表示されます（フォーム上に有効なデータソースが1つも無い場合、ページは表示されません）。代わりに、[**接続**]ページを使用して、接続文字列やレコードソースを選択したり構築することができます。

例えば、メインフォームがそれにアタッチされているいくつかの DataTable による DataSet オブジェクトを含んでいる場合、データソースダイアログは以下のように表示されます。




レポートの作成もしくは編集が完了したら、[**ファイル**｜**変更内容を保存して終了**]を選択することでデザイナを終了することができます。これにより、[**レポート定義ファイルの読み込み**]を選択してファイルから読み込んだ場合と同じように、コンポーネント内に直接レポート定義ファイルが保存されます。

変更内容の保存をキャンセルする場合は、[**ファイル**｜**変更内容を破棄して終了**]を選択してください。

## 実行時に読み込まれるレポート

この例では、実行時にファイルからレポート定義を読み込みます。このようなアプリケーションは、レポート定義ファイルを必要とし、ビューアのように機能します。このようなアプリケーションの主な長所は、レポート形式を変更しても、アプリケーションを更新する必要がないことです。新しいレポート定義ファイルをユーザーに送るだけで済みます。

実行時に読み込まれるレポートを備えたアプリケーションを作成するには、次の手順に従います。

1. **C1Report デザイナ**を使用して、必要なレポートをすべて作成します（作成方法の詳細については、「C1Report デザイナの使い方」を参照）。


Copyright © GrapeCity, Inc. All rights reserved.

2. 次のコントロールをアプリケーションに追加します。
   - **c1r** という名前の **C1Report** mponent named
   - **ppv** という名前の **C1PrintPreviewControl**
   - **cmbReport** という名前の **ComboList**
   - **status** という名前の **StatusBar**
3. 次の Imports 文(Visual Basic の場合)をファイルの先頭に追加します。

**Visual Basic コードの書き方**

Visual Basic
```
Imports C1.C1Report
Imports System.IO
```

**C# コードの書き方**

C#
```
using C1.C1Report;
using System.IO;
```

これにより、完全な名前空間を指定しなくても、**C1Report** と System.IO のクラスとオブジェクトを参照できます。

4. レポート定義ファイルを読み取り、ファイル内のすべてのレポートのリストを構築するためのコードを追加します。そのためのコードは次のとおりです。

**Visual Basic コードの書き方**

Visual Basic
```
' アプリケーションパスを取得します。
Dim appPath As String
appPath = Path.GetDirectoryName(Application.ExecutablePath).ToLower()
Dim i As Integer = appPath.IndexOf("/bin")
If (i < 0) Then i = appPath.IndexOf("\bin")
If (i > 0) Then appPath = appPath.Remove(i, appPath.Length - i)

' レポート定義ファイル内の全レポートのリストを取得します。
m_ReportDefinitionFile = appPath & "\Data\Nwind.xml"
Dim reports As String() = c1r.GetReportInfo(m_ReportDefinitionFile)

' コンボボックスに代入します。
cmbReport.Items.Clear()
Dim report As String
For Each report In reports
  cmbReport.Items.Add(report)
Next
```

**C# コードの書き方**

C#
```
// アプリケーションパスを取得します。
 string appPath;
appPath = Path.GetDirectoryName(Application.ExecutablePath).ToLower();
 int i = appPath.IndexOf("/bin");
if ((i < 0) ) { i = appPath.IndexOf("\bin"); }
if ((i > 0) ) { appPath = appPath.Remove(i, appPath.Length - i); }

// レポート定義ファイル内の全レポートのリストを取得します。
```


Copyright © GrapeCity, Inc. All rights reserved.

```
m_ReportDefinitionFile = appPath + "\Data\Nwind.xml";
 string ( reports) = c1r.GetReportInfo(m_ReportDefinitionFile);

// コンボボックスに代入します。
cmbReport.Items.Clear();
 string report;
foreach report In reports
  cmbReport.Items.Add(report);
}
```

上のコードは、最初にレポート定義を含むファイルの場所を取得しています。そのために、システム定義の **Path** クラスおよび **Application** クラスの静的メソッドを使用します。このコードは、実際のレポート定義ファイルの場所と名前に合わせて変更する必要があります。

次に、**GetReportInfo** メソッドを使用してレポート定義ファイル（手順1で作成）内のすべてのレポート名を含む配列を取得し、ユーザーがレポートを選択するためのコンボボックスにレポート名を挿入します。

5. ユーザーが選択したレポートを出力するためのコードを追加します。次に例を示します。

**Visual Basic コードの書き方**

Visual Basic

```
Private Sub cmbReport_SelectedIndexChanged(ByVal sender As Object, ByVal e As
EventArgs) Handles cmbReport.SelectedIndexChanged
  Try
    Cursor = Cursors.WaitCursor

    ' レポートを読み込みます。
    status.Text = "Loading " & cmbReport.Text
    c1r.Load(m_ReportDefinitionFile, cmbReport.Text)

     ' PrintPreview コントロール内に出力します。
    status.Text = "Rendering " & cmbReport.Text
    ppv.Document = c1r.Document

    ' PrintPreview コントロールにフォーカスを移します。
    ppv.StartPage = 0
    ppv.Focus()

    Finally
      Cursor = Cursors.Default
    End Try
End Sub
```

**C# コードの書き方**

C#

```
private void cmbReport_SelectedIndexChanged(object sender, System.EventArgs e)
{
  try {
    Cursor = Cursors.WaitCursor;

    //  レポートを読み込みます。
    status.Text = "Loading " + cmbReport.Text;
    c1r.Load(m_ReportDefinitionFile, cmbReport.Text);

    //  PrintPreview コントロール内に出力します。
```


Copyright © GrapeCity, Inc. All rights reserved.

```
    status.Text = "Rendering " + cmbReport.Text;
    ppv.Document = c1r.Document;

    // PrintPreview コントロールにフォーカスを移します。
    ppv.StartPage = 0;
    ppv.Focus();

    } finally {
      Cursor = Cursors.Default;
    }
}
```

# 実行時にカスタマイズされるレポート

この例は、「実行時に読み込まれるレポート」の応用です。ファイルから基本のレポート定義をロードする部分と、ユーザーの選択に応じてレポートをカスタマイズする部分で構成されています。

たとえば、次のコードは、詳細セクションで使用されるフォントを変更しています。

**Visual Basic コードの書き方**

Visual Basic

```
Imports C1.C1Report

Dim s As Section = c1r.Sections(SectionTypeEnum.Detail)
Dim f As Field
For Each f In s.Fields
  f.Font.Name = "Arial Narrow"
Next
```

**C# コードの書き方**

C#

```
using C1.C1Report;

Section s = c1r.Sections[SectionTypeEnum.Detail];
foreach (Field f in s.Fields)
  f.Font.Name = "Arial Narrow";
```

次のコードは、グループの **Sort** プロパティをオンまたはオフに変更し、グループのヘッダセクションおよびフッタセクションの**Visible** プロパティを設定することによって、グループの表示を切り替えています。

**Visual Basic コードの書き方**

Visual Basic

```
Dim bShowGroup As Boolean
bShowGroup = True
With c1r.Groups(0)
  If bShowGroup Then
    .SectionHeader.Visible = True
    .SectionFooter.Visible = True
    .Sort = SortEnum.Ascending
  Else
```


Copyright © GrapeCity, Inc. All rights reserved.

```
    .SectionHeader.Visible = False
    .SectionFooter.Visible = False
    .Sort = SortEnum.NoSort
    End If
End With
```

**C# コードの書き方**

C#

```
bool bShowGroup;
bShowGroup = true;
  if (bShowGroup)
  {
    c1r.Groups[0].SectionHeader.Visible = true;
    c1r.Groups[0].SectionFooter.Visible = true;
    c1r.Groups[0].Sort = SortEnum.Ascending;
  }
  else
  {
    c1r.Groups[0].SectionHeader.Visible = false;
    c1r.Groups[0].SectionFooter.Visible = false;
    c1r.Groups[0].Sort = SortEnum.NoSort;
  }
```

これらの例は、レポートをカスタマイズする方法のほんの一例にすぎません。オブジェクトモデルには、レポートのすべての側面へのアクセス方法が用意されているため、さまざまなカスタマイズを行うことができます。実際、完全にコードだけを使ってレポート全体を作成することもできます。

# Web アプリケーションの開発

Web（ASP.NET）のレポートを作成する場合、**ComponentOne Enterprise** パッケージに同梱された **C1WebReport** コントロールを使用できます。このコントロールは、**C1Report** コンポーネントをカプセル化し、きわめて簡単にレポートを Web ページに追加できるようにするメソッドとプロパティを提供します。**C1WebReport** コントロールは C1Report と完全に互換性があり、Web 専用に設計された高度なキャッシュおよび表示オプションと共に、ASP.NET サーバーコントロールによって提供される通常のデザイン時の編集オプションを提供します。

希望する場合は、引き続き Web アプリケーションで C1Report コンポーネントを使用できますが、レポートの HTML または PDF バージョンを作成するためにいくつかのコードを作成する必要があります。以下の各セクションでは、これを実行する方法について説明します。

通常の Web シナリオでは、C1Report は、サーバーマシン上で実行され、バッチモードまたはオンデマンドのいずれかでレポートを作成します。ユーザーは、クライアントマシン上で Web ブラウザを使用して、レポートを選択し、それらをプレビューまたは印刷できます。

# 静的 Web レポート

静的 Web レポートは、定期的に実行され、一連の定義済みレポートを作成し、HTML または PDF ファイルに保存するサーバーアプリケーションに基づいています。これらのファイルは、お使いのサイトの Web サイトで参照され、他の任意の Web ページと同様にクライアントマシンにダウンロードされます。

このタイプのアプリケーションを実装するには、以下の手順を実行します。

1. **C1ReportDesigner** アプリケーションを使用して、必要なすべてのレポートを作成します（これを実行する方法についての詳細は、「C1Report デザイナの使い方」を参照してください）。


Copyright © GrapeCity, Inc. All rights reserved.

2. **C1Report** コンポーネントを含むサーバー上でアプリケーションを作成します。フォームまたはウィンドウを使用しない場合は、**CreateObject** 関数を使用してコントロールを作成します。
3. 定期的に実行するルーチンを追加し、ユーザーが利用できるようにするすべてのレポートを更新します。ループは次のようになります。

**Visual Basic コードの書き方**

```
Visual Basic
' この処理を6時間ごとに実行します

' 定義ファイル内のすべてのレポートのリストを取得します
sFile = "c:\inetpub\wwwroot\Reports\MyReports.xml"
sList = c1r.GetReportInfo(sFile)

' サーバー上のレポートを更新します
For i = 0 To sList.Length - 1
  c1r.Load(sFile, sList(i))
  sFile = "Reports\Auto\" & sList(i) & ".htm"
  c1r.RenderToFile(sFile, FileFormatEnum.HTMLPaged)
Next
```

**C# コードの書き方**

```
C#
// この処理を6時間ごとに実行します

//定義ファイル内のすべてのレポートのリストを取得します
sFile = "c:\inetpub\wwwroot\Reports\MyReports.xml";
sList = c1r.GetReportInfo(sFile);

// サーバー上のレポートを更新します
for ( i = 0 ; GAIS <= sList.Length - 1
  c1r.Load(sFile, sList(i));
  sFile = "Reports\Auto\" + sList(i) + ".htm";
  c1r.RenderToFile(sFile, FileFormatEnum.HTMLPaged);
}
```

コードは、**GetReportInfo** メソッドを使用して、MyReports.xml レポート定義ファイル（手順1で作成）に含まれるすべてのレポートのリストを取得し、各レポートをページ付けされた HTML ファイルに表示します（ページ付けされた HTML ファイルには、元のレポートのページごとに、閲覧が可能なナビゲーションバーを備えた HTML ページが1ページずつ含まれています）。

4. 保存されたレポートへのリンクを追加して、ホーム HTML ページを編集します。

HTML に限定されません。**C1Report** は、自由に利用できるプラグインを持つブラウザ上で表示できる PDF ファイルにエクスポートすることもできます。実際、PDF 形式は、特に Web レポートにハードコピーを作成する場合など、多くの点で HTML より優れています。

## 動的 Web レポート

動的 Web レポートは、オンデマンドで、おそらくユーザーが提供するデータに基づいて作成されます。通常、このタイプのソリューションは、ユーザーにフォームを提示する ASP.NET ページを含み、レポートの作成に必要な情報を収集し、**C1Report** コンポーネントを作成して、レポートを一時ファイルに表示し、そのファイルの参照を返します。

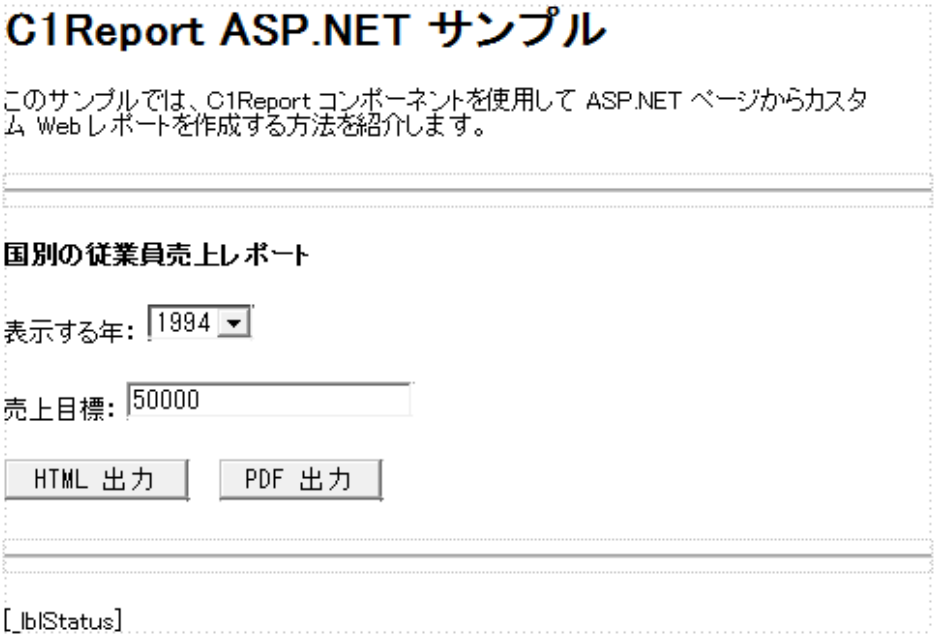

Copyright © GrapeCity, Inc. All rights reserved.

以下の例は、ユーザーが何らかの情報を入力して、希望するレポートのタイプを選択できる単純な ASP.NET ページです。これに基づいて、ASP コードが NorthWind による「Employee Sales by Country（社員の国別売上高）」レポートのカスタムバージョンを作成し、それを選択した形式でユーザーに提示します.

サンプルは、レポートを格納するサーバー上の一時ファイルを使用しています。実際のアプリケーションでは、ファイルを確認する機会をユーザーに提供する前に、レポートが上書きされないように、独自のファイル名を作成し、一定期間経過後それらを削除する必要がある場合があります。ただし、このサンプルでは、C1Report を含む Web 上でのレポートの配信に関連する主な手法について説明します。

このタイプのアプリケーションを実装するには、以下の手順を実行します。

1. 以下のような Web ページを含む新しい Web アプリケーションを作成から始めます。




   ページには、以下の5種類のサーバーコントロールがあります。

   - **_1stYear:** データ（1994、1995、および 1996）が存在する有効な年度のリストを含みます。スマートタグ（）をクリックし、メニューから[**項目の編集**]を選択して、項目を追加できることに注意してください。[**ListItem コレクションエディタ**]から、3つの新しい項目を追加します。
   - **_txtGoal:** 各社員の年次販売目標を含みます。
   - **_btnHTML**, **_btnPDF:** レポートを HTML または PDF に表示するために使用するボタンを含み、結果を表示します。
   - **_lblStatus:** 失敗が発生した場合、エラー情報を表示します。

   > **メモ**:このアプリケーションを **C1Report** コンポーネントのデモ版またはベータ版で実行する場合、コントロールがサーバー上で[**バージョン情報**]ダイアログボックスを表示しようとしたときにエラーが発生します。この場合、単にページをリロードすると問題は解決します。

2. ページを設定した後で、C1Report への参照をプロジェクトに 追加する必要があります。ソリューションエクスプローラウィンドウでプロジェクトを右クリックし、[**参照の追加**]を選択して、C1Report コンポーネントを選択します。
3. Data フォルダ内のプロジェクトに **Nwind.xml** を追加します。**ソリューションエクスプローラ**ウィンドウでプロジェクトを右クリックし、[**新規フォルダ**]を選択して、**Data** フォルダの名前を変更します。次に、フォルダを右クリックし、[**既存項目の追加**]を選択して、デフォルトで C:\Users\<ユーザー名>\Documents\ComponentOne Samples\WinForms\C1Report\C1Report\XML ディレクトリにインストールされる **Nwind.xml** 定義ファイルを参照します。
4. プロジェクトに **Temp** フォルダを追加します。**ソリューションエクスプローラ**ウィンドウでプロジェクトを右クリックし、[**新規フォルダ**]を選択して、**Temp** フォルダの名前を変更します。
5. 従来の ASP を使用している場合、ここから興味深いことが始まります。コントロールをダブルクリックすると、Windows フォームプロジェクトの記述に使用する同じエディタと環境を使用して、イベントを処理する本格的なコードを記述できるコードウィンドウが表示されます。

   次のコードを追加します。


Copyright © GrapeCity, Inc. All rights reserved.

**Visual Basic コードの書き方**

Visual Basic

```
Imports C1.C1Report

' ユーザークリックを処理します
Private Sub _btnHTML_Click(ByVal sender As Object, ByVal e As System.EventArgs)
Handles _btnHTML.Click
     RenderReport(FileFormatEnum.HTMLDrillDown)
End Sub

Private Sub _btnPDF_Click(ByVal sender As Object, ByVal e As System.EventArgs)
Handles _btnPDF.Click
     RenderReport(FileFormatEnum.PDF)
End Sub
```

**C# コードの書き方**

C#

```
using C1.C1Report;

// ユーザークリックを処理します
private void _btnHTML_Click(object sender, System.EventArgs e)
{
   RenderReport(FileFormatEnum.HTMLDrillDown);
}

private void _btnPDF_Click(object sender, System.EventArgs e)
{
   RenderReport(FileFormatEnum.PDF);
}
```

これは、ユーザーがいずれかのボタンをクリックしたときにサーバー上で実行されるコードです。

6. 次のコードは、作業をメインルーチンの **RenderReport** に委任します。

**Visual Basic コードの書き方**

Visual Basic

```
Private Sub RenderReport(ByVal fmt As FileFormatEnum)

    ' ファイル名をビルドします
    Dim rpt As String = "地域別従業員売上高"
    Dim fileIn As String = GetDataPath() & "NWind.xml"
    Dim ext As String = Iif(fmt = FileFormatEnum.PDF, ".pdf", ".htm")
    Dim fileOut As String = GetOutputPath() & rpt & ext

    Try
        ' C1Report コンポーネントを作成します
        Dim c1r As New C1Report()

        ' レポートをロードします
        c1r.Load(fileIn, rpt)
```



Copyright © GrapeCity, Inc. All rights reserved.

```
        '  ユーザーパラメータを取得します
        Dim year As String = _lstYear.SelectedItem.Text
        Dim goal As String = _txtGoal.Text

        ' レポートのデータソースをカスタマイズします
        Dim sSQL As String = "SELECT DISTINCTROW " & _
            "Employees.Country, Employees.LastName, " & _
            "Employees.FirstName, Orders.ShippedDate, Orders.OrderID, " & _
            "  [Order Subtotals].Subtotal AS SaleAmount " & _
            "FROM Employees INNER JOIN (Orders INNER JOIN " & _
            "  [Order Subtotals] ON Orders.OrderID = " & _
            "  [Order Subtotals].OrderID) " & _
            "  ON Employees.EmployeeID = Orders.EmployeeID " & _
            "WHERE Year(Orders.ShippedDate) = " & year & ";"
        c1r.DataSource.RecordSource = sSQL

        ' レポートのイベントハンドラをカスタマイズします
        Dim sScript As String = _
            "If SalespersonTotal > " & goal & " Then" & vbCrLf & _
            "  ExceededGoalLabel.Visible = True" & vbCrLf & _
            "  SalespersonLine.Visible = True" & vbCrLf & _
            "Else" & vbCrLf & _
            "  ExceededGoalLabel.Visible = False" & vbCrLf & _
            "  SalespersonLine.Visible = False" & vbCrLf & _
            "End If"
        c1r.Sections(SectionTypeEnum.GroupHeader2).OnPrint = sScript

        ' レポートを一時ファイルに表示します
        c1r.RenderToFile(fileOut, fmt)

        ' ユーザーをレポートファイルにリダイレクトします
        Response.Redirect("Temp/" + rpt + ext)

    Catch x As Exception

        _lblStatus.Text = "*** " & x.Message

    End Try
End Sub
```

**C# コードの書き方**

C#

```
// ファイル名をビルドします
private void RenderReport(FileFormatEnum fmt)
{


   // build file names
   string rpt  = "地域別従業員売上高";
   string fileIn = GetDataPath() + "NWind.xml";
   string ext = (fmt == FileFormatEnum.PDF)? ".pdf": ".htm";
   string fileOut = GetOutputPath() + rpt + ext;
```


Copyright © GrapeCity, Inc. All rights reserved.

```
    try
    {
    // C1Report コンポーネントを作成します
    C1Report c1r = new C1Report();

    //  レポートをロードします
    c1r.Load(fileIn, rpt);

    // ユーザーパラメータを取得します
    string year = _lstYear.SelectedItem.Text;
    string goal = _txtGoal.Text;

    //  レポートのデータソースをカスタマイズします
    string sSQL = "SELECT DISTINCTROW " +
    "Employees.Country, Employees.LastName, " +
    "Employees.FirstName, Orders.ShippedDate, Orders.OrderID, " +
    "  [Order Subtotals].Subtotal AS SaleAmount " +
    "FROM Employees INNER JOIN (Orders INNER JOIN " +
    "  [Order Subtotals] ON Orders.OrderID = " +
    "  [Order Subtotals].OrderID) " +
    "  ON Employees.EmployeeID = Orders.EmployeeID " +
    "WHERE Year(Orders.ShippedDate) = " + year + ";";
    c1r.DataSource.RecordSource = sSQL;

    // レポートのイベントハンドラをカスタマイズします
    string sScript =
    "If SalespersonTotal > " + goal + " Then \n" +
    "  ExceededGoalLabel.Visible = True\n" +
    "  SalespersonLine.Visible = True\n" +
    "Else\n" +
    "  ExceededGoalLabel.Visible = False\n" +
    "  SalespersonLine.Visible = False\n" +
    "End If";
    c1r.Sections[SectionTypeEnum.GroupHeader2].OnPrint = sScript;

    //  レポートを一時ファイルに表示します
    c1r.RenderToFile(fileOut, fmt);
    // ユーザーをレポートファイルにリダイレクトします
    Response.Redirect("Temp/" + rpt + ext);
    }
    catch (Exception x)
    {
    _lblStatus.Text = "*** " + x.Message;
    }
}
```

**RenderReport** ルーチンは長いですが、非常に単純です。入力および出力ファイルの名前の解決から始めます。すべてのファイル名は、現在のアプリケーションディレクトリと関連付けてビルドされます。

次に、ルーチンは **C1Report** コンポーネントを作成し、「Employee Sales by Country（社員の国別売上高）」レポートをロードします。これは未処理レポートであり、次の手順でカスタマイズします。

ユーザーが入力したパラメータは、**_lstYear** および **_txtGoal** サーバー側コントロールで利用できます。コードはこれらの値を読み取り、これらを使用してレポートの **RecordSource** プロパティをカスタマイズし、**OnPrint** OnPrint の VBScript ハンドラをビルドします。これらの手順は、前のセクションで取り扱いました。

レポート定義の準備が完了すると、コードが **RenderToFile** メソッドを呼び出し、この結果 C1Report コンポーネントによっ


Copyright © GrapeCity, Inc. All rights reserved.

て HTML または PDF ファイルが出力ディレクトリに書き込まれます。メソッドが返されると、レポートはユーザーに表示する準備が完了します。

最後の手順では、**Response.Redirect** を呼び出し、作成したレポートをユーザーのブラウザ上で表示します。

すべてのコードが **try/catch** ブロックで囲まれていることに注意してください。レポートの生成時に失敗が発生した場合、問題を説明するメッセージがユーザーに対して表示されます。

7. 最後に、いくつかの単純なヘルパールーチンを追加する必要があります。

**Visual Basic コードの書き方**

Visual Basic

```
' ファイルのロードおよび保存に使用するディレクトリを取得します
Private Function GetDataPath() As String
   Return Request.PhysicalApplicationPath + "Data\"
End Function

Private Function GetOutputPath() As String
   Return Request.PhysicalApplicationPath + "Temp\"
End Function
```

**C# コードの書き方**

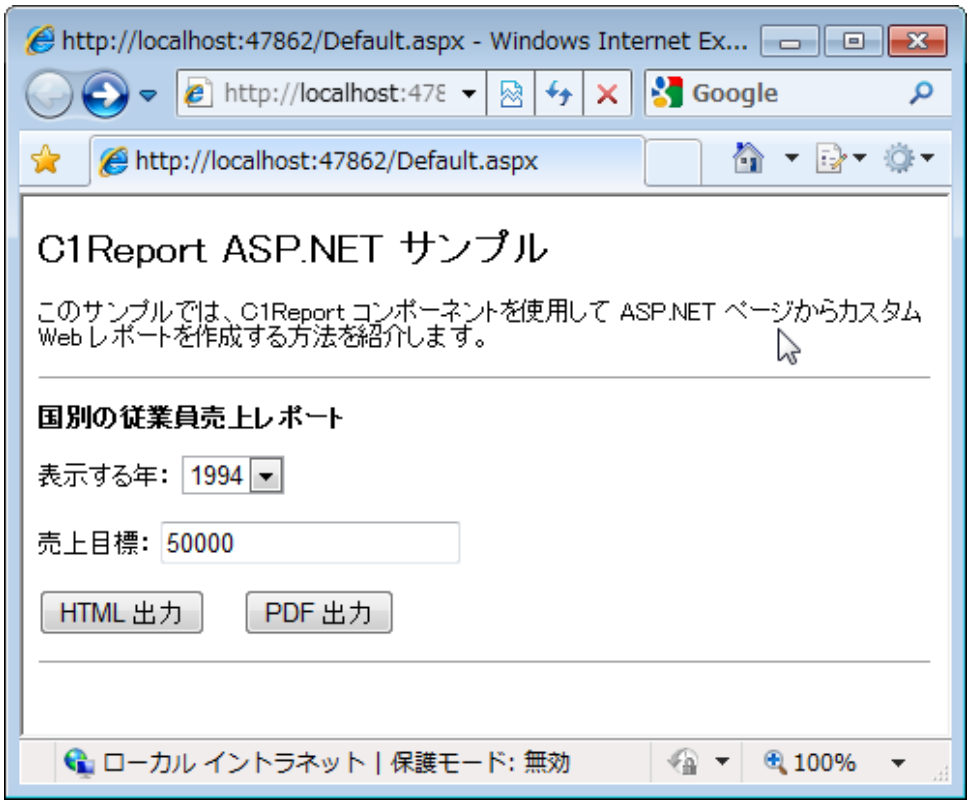
C#

```
// ファイルのロードおよび保存に使用するディレクトリを取得します
private string GetDataPath()
{
   return Request.PhysicalApplicationPath + @"Data\";
}
private string GetOutputPath()
{
   return Request.PhysicalApplicationPath + @"Temp\";
}
```

8. このコードを入力したら、アプリケーションを使用する準備が完了します。[F5]キーを押して、Visual Studio 内でその実行を追跡します。

以下のスクリーンショットは、ブラウザに表示される結果を示しています。


Copyright © GrapeCity, Inc. All rights reserved.




# レポートの作成、読み込み、描画

デスクトップおよび Web 上のさまざまな多くのシナリオで **C1Report** を使用できますが、手順のメインシーケンスは常に同じです。

1. **レポート定義の作成**

   これは、**C1Report デザイナ**によるか、Microsoft Access のレポートデザイナを使用して直接実行し、C1Report デザイナにインポートできます。オブジェクトモデルを使用してグループとフィールドを追加するか、またはカスタム XML ファイルを記述するかのいずれかによって、コードを使用して実行することもできます。

2. **C1Report コンポーネントへのレポートのロード**

   これは、[**レポートのロード**]コンテキストメニューを使用するか、プログラム操作で **Load** メソッドを使用して、デザイン時に実行できます。デザイン時にレポートをロードすると、レポートはコントロールに保持（保存）され、レポート定義ファイルを配布する必要はありません。

3. **レポートの表示（ディスクトップアプリケーション）**

   デスクトップアプリケーションを記述する場合、レポートを安易に **C1PrintPreview** コントロールに表示できます。プレビューコントロールは、画面上にレポートを表示し、ユーザーは、フルズーム、パンなどを使用してそれをプレビューできます。

   **To write code in Visual Basic**

   ```
   Visual Basic
   C1PrintPreview1.Document = c1r
   ```

   **To write code in C#**

   ```
   C#
   c1printPreview1.Document = c1r;
   ```


Copyright © GrapeCity, Inc. All rights reserved.

4. **レポートの表示（Web アプリケーション）**

   Web アプリケーションを記述する場合、**RenderToFile** メソッドを使用して、レポートを HTML または PDF ファイルに表示し、ユーザーは任意のブラウザを使用して、これらを表示できます。

# レポート定義の作成

以下のトピックでは、**C1ReportDesigner** またはコードを使用したレポート定義の作成方法を説明します。レポート定義の作成はレポートのレンダリングと同じではないことに注意してください。レポートをレンダリングするには、単に既存の定義をロードし、**Render** メソッドを呼び出すことができます。

C1ReportDesigner を使用したレポート定義の作成

レポート定義を作成する最も簡単な方法は、Microsoft Access のレポートデザイナに似たスタンドアロン型アプリケーションである **C1ReportDesigner** を使用することです。**C1Report ウィザード**は、新しいレポートを作成する手順を段階的に最初から最後までガイドしてくれます。

新しいレポートの作成手順の詳細については、「基本的なレポート定義ファイルの作成」を参照してください。

# コードを使用したレポート定義の作成

コードを使用して最初からレポートを作成することもできます。この方法では、いくつかの特別な作業が必要になりますが、たいへん柔軟にレポートを作成できます。独自のレポートデザイナを記述したり、アドホックなレポートジェネレータを作成することもできます。

次の例は、コードを使用して、NorthWind データベースに基づく単純な表形式のレポート定義を作成します。コードにはコメントが付けられ、**C1Report** オブジェクトモデルの最も重要な要素が例示されています。次の手順を実行します。

1. 最初に、フォームに C1Report、**C1PrintPreview**Control コントロールを追加し、名前を次のように設定します。

   Button.Name = **btnEmployees**

   C1Report.Name = **c1r**

   C1PrintPreviewControl.Name = **ppv**

2. C1Report コントロールを初期化します。**Clear** メソッドを使ってコンテンツを削除し、コントロールのフォントを設定します。このフォントは、新しいフィールドに割り当てられます。（以降のコードは、すべてフォームロードイベント内に記述します。）

   **Visual Basic コードの書き方**

   Visual Basic

```
 ' コントロールを初期化します。
With c1r
    ' 既存のフィールドを削除します。
  .Clear()
    ' 全てのコントロールのデフォルトフォントを設定します。
  .Font.Name = "Tahoma"
  .Font.Size = 8
End With
```

   **C# コードの書き方**

   C#

```
// コントロールを初期化します。
```


Copyright © GrapeCity, Inc. All rights reserved.

```
// 既存のフィールドを削除します。
c1r.Clear();
// 全てのコントロールのデフォルトフォントを設定します。
c1r.Font.Name = "Tahoma";
c1r.Font.Size = 8;
```

3. 次に、**DataSource** オブジェクトを設定し、NorthWind データベースから必要なデータを取得します。これには、**ConnectionString** プロパティと**RecordSource**プロパティを使用します（Microsoft ADO DataControl と同様）。

**Visual Basic コードの書き方**

Visual Basic

```
' データソースを初期化します。
With c1r.DataSource
  .ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
                      "Data Source=C:\...\ComponentOne
Samples\Common\Nwind.mdb;" & _
                      "Persist Security Info=False"
  .RecordSource = "Employees"
End With
```

**C# コードの書き方**

C#

```
//データソースを初期化します。
DataSource ds = c1r.DataSource;
ds.ConnectionString = @"Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\...\ComponentOne Samples\Common\Nwind.mdb;";
ds.RecordSource = "Employees";
```

4. 次に、ページ上のレポートのレイアウト方法を定義する **Layout** オブジェクトを初期化します。ここでは、レポートの出力方法を縦置きモードに、**Width** を 6.5 インチ（ページ幅の 8.5 インチから、両側のマージン各1インチを引いた値）に設定します。

**Visual Basic コードの書き方**

Visual Basic

```
' レイアウトを初期化します。
With c1r.Layout
  .Orientation = OrientationEnum.Portrait
  .Width = 6.5 * 1440 ' 8.5 - margins, in twips
End With
```

**C# コードの書き方**

C#

```
// レイアウトを初期化します。
Layout l = c1r.Layout;
l.Orientation = OrientationEnum.Portrait;
l.Width = 6.5 * 1440; // 8.5 - margins, in twips
```

5. 次が最も注目すべき部分です。すべてのレポートには、5つの基本セクション（詳細・レポートヘッダ・レポートフッタ・ページヘッダ・ページフッタ）があります。まず、いくつかのプロパティを設定し、タイトルフィールドを追加して、レポート


Copyright © GrapeCity, Inc. All rights reserved.

ヘッダを設定します。

**Visual Basic コードの書き方**

```
Visual Basic
' レポートフィールドの作成とフォーマットに使用する変数。
Dim f As Field

' レポートヘッダを作成します。r
 With c1r.Sections(SectionTypeEnum.Header)
  .Height = 1440
  .Visible = True
  .BackColor = Color.FromArgb(200, 200, 200)
  f = .Fields.Add("FldTitle", "Employees Report", 0, 0, 8000, 1440)
  f.Font.Size = 24
  f.Font.Bold = True
  f.ForeColor = Color.FromArgb(0, 0, 100)
End With
```

**C# コードの書き方**

```
C#
// レポートフィールドの作成とフォーマットに使用する変数。
 Field f;

// レポートヘッダを作成します。
Section s = c1r.Sections[SectionTypeEnum.Header];
  s.Height = 1440;
  s.Visible = true;
  s.BackColor = Color.FromArgb(200, 200, 200);
  f = s.Fields.Add("FldTitle", "Employees Report", 0, 0, 8000, 1440);
  f.Font.Size = 24;
  f.Font.Bold = true;
  f.ForeColor = Color.FromArgb(0, 0, 100);
```

上のセクションオブジェクトには、**Fields** コレクションがあります。このコレクションの**Add** メソッドは、新しいフィールドを作成し、それをセクションに割り当てます。パラメータには、新しいフィールドの **Name**、**Text**、**Left**、**Top**、**Width**、および**Height** の各プロパティを指定しています。デフォルトでは、フィールドのフォントはコントロールと同じです。このフィールドはタイトルなので、フォントを変更してサイズを大きくしています。フォントサイズに合わせてフィールドの高さを十分大きくするように注意してください。フォントサイズに対してフィールドの高さが十分でない場合、何も表示されません。

6. 次に、ページフッタセクションを設定します。このセクションには計算フィールドが含まれており、さらに興味深いコードになっています。計算フィールドは、Text プロパティに VBScript 式が含まれており、これらの式はレポートの出力時に評価されます。計算フィールドを作成するには、**Calculated**プロパティを **True** に設定します。:

**Visual Basic コードの書き方**

```
Visual Basic
' ページを作成します。
With c1r.Sections(SectionTypeEnum.PageFooter)
  .Height = 500
  .Visible = True
```


Copyright © GrapeCity, Inc. All rights reserved.

```
  f = .Fields.Add("FldFtrLeft", """Employees: Printed on "" & Now",_
                  0, 0, 4000, 300)
  f.Calculated = True
  f = .Fields.Add("FldFtrRight", """Page "" & Page & "" of "" & Pages",_
                  4000, 0, 4000, 300)
  f.Calculated = True
  f.Align = FieldAlignEnum.RightTop
  f.Width = c1r.Layout.Width - f.Left
  f = .Fields.Add("FldLine", "", 0, 0, c1r.Layout.Width, 20)
  f.LineSlant = LineSlantEnum.NoSlant
  f.BorderStyle = BorderStyleEnum.Solid
  f.BorderColor = Color.FromArgb(0, 0, 100)
End With
```

**C# コードの書き方**

C#

```
// ページを作成します。
s = c1r.Sections[SectionTypeEnum.PageFooter];
  s.Height = 500;
  s.Visible = true;
  f = s.Fields.Add("FldFtrLeft", @"""Employees: Printed on "" & Now",_
                  0, 0, 4000, 300);
  f.Calculated = true;
  f = .Fields.Add("FldFtrRight", @"""Page "" + Page + "" of "" & Pages",_
                  4000, 0, 4000, 300);
  f.Calculated = true;
  f.Align = FieldAlignEnum.RightTop;
  f.Width = c1r.Layout.Width - f.Left;
  f = s.Fields.Add("FldLine", "", 0, 0, c1r.Layout.Width, 20);
  f.LineSlant = LineSlantEnum.NoSlant;
  f.BorderStyle = BorderStyleEnum.Solid;
  f.BorderColor = Color.FromArgb(0, 0, 100);
```

このページフッタセクションの式では、VBScript 固有の変数ではなく、**C1Report**で定義されている変数が使用されています。**Page** および **Pages** は、現在のページ番号と総ページ数を含む変数です。また、このセクションでは、1本の線として表示されるように設定されたフィールドを使用しています。そのために、**BorderStyle** プロパティと**LineSlant** プロパティが使用されています。

7. 次に、ページヘッダセクションを設定します。このセクションは、各ページの上部に出力され、フィールドラベルが表示されます。ページヘッダセクションをフィールドラベルの表示に使用するのは、表形式のレポートにおける一般的なテクニックです。すべてのフィールドのサイズを指定しているため、多少煩雑に見えるかもしれませんが、コード自体はシンプルです。実際のアプリケーションでは、これらの値はプログラムに組み込まれることはありません。

   **Visual Basic コードの書き方**

Visual Basic

```
' フィールドラベルを含むページヘッダを作成します。
With c1r.Sections(SectionTypeEnum.PageHeader)
  .Height = 500
  .Visible = True
  c1r.Font.Bold = True
  f = .Fields.Add("LblID", "ID", 0, 50, 400, 300)
```


Copyright © GrapeCity, Inc. All rights reserved.

```
  f.Align = FieldAlignEnum.RightTop
  f = .Fields.Add("LblFirstName", "First", 500, 50, 900, 300)
  f = .Fields.Add("LblLastName", "Last", 1500, 50, 900, 300)
  f = .Fields.Add("LblTitle", "Title", 2500, 50, 2400, 300)
  f = .Fields.Add("LblTitle", "Notes", 5000, 50, 8000, 300)
  c1r.Font.Bold = False
  f = .Fields.Add("FldLine", "", 0, 400, c1r.Layout.Width, 20)
  f.LineSlant = LineSlantEnum.NoSlant
  f.LineWidth = 50

  f.BorderColor = Color.FromArgb(100, 100, 100)
End With
```

**C# コードの書き方**

C#

```
//フィールドラベルを含むページヘッダを作成します。    v
s = c1r.Sections[SectionTypeEnum.PageHeader];
  s.Height = 500;
  s.Visible = true;
  c1r.Font.Bold = true;
  f = s.Fields.Add("LblID", "ID", 0, 50, 400, 300);
  f.Align = FieldAlignEnum.RightTop;
  f = s.Fields.Add("LblFirstName", "First", 500, 50, 900, 300);
  f = s.Fields.Add("LblLastName", "Last", 1500, 50, 900, 300);
  f = s.Fields.Add("LblTitle", "Title", 2500, 50, 2400, 300);
  f = s.Fields.Add("LblTitle", "Notes", 5000, 50, 8000, 300);
  c1r.Font.Bold = false;
  f = s.Fields.Add("FldLine", "", 0, 400, c1r.Layout.Width, 20);
  f.LineSlant = LineSlantEnum.NoSlant;
  f.LineWidth = 50;
  f.BorderColor = Color.FromArgb(100, 100, 100);
```

上のコードは、フォントを処理する効果的なテクニックの例を示しています。すべてのフィールドは、作成時にコントロールのフォントを継承するため、フィールドを作成する前にコントロールの **Font.Bold** プロパティを **True** に設定し、後でこの設定を **False** に戻します。これにより、ページヘッダセクション内のすべてのコントロールのフォントが太字になります。

8. 次に詳細セクションを追加して、レポートを完成させます。詳細セクションは、実際のデータを表示するセクションです。ページヘッダセクションの各ラベルの下に、計算フィールドを設定します。

**Visual Basic コードの書き方**

Visual Basic

```
' 詳細セクションを作成します。
With c1r.Sections(SectionTypeEnum.Detail)
  .Height = 330
  .Visible = True
  f = .Fields.Add("FldID", "EmployeeID", 0, 0, 400, 300)
  f.Calculated = True
  f = .Fields.Add("FldFirstName", "FirstName", 500, 0, 900, 300)
  f.Calculated = True
  f = .Fields.Add("FldLastName", "LastName", 1500, 0, 900, 300)
```


Copyright © GrapeCity, Inc. All rights reserved.

```
  f.Calculated = True
  f = .Fields.Add("FldTitle", "Title", 2500, 0, 2400, 300)
  f.Calculated = True
  f = .Fields.Add("FldNotes", "Notes", 5000, 0, 8000, 300)
  f.Width = c1r.Layout.Width - f.Left
  f.Calculated = True
  f.CanGrow = True
  f.Font.Size = 6
  f.Align = FieldAlignEnum.JustTop
  f = .Fields.Add("FldLine", "", 0, 310, c1r.Layout.Width, 20)
  f.LineSlant = LineSlantEnum.NoSlant
  f.BorderStyle = BorderStyleEnum.Solid
  f.BorderColor = Color.FromArgb(100, 100, 100)
End With
```

**C# コードの書き方**

C#

```
//   詳細セクションを作成します。
s = c1r.Sections[SectionTypeEnum.Detail];
  s.Height = 330;
  s.Visible = true;
  f = s.Fields.Add("FldID", "EmployeeID", 0, 0, 400, 300);
  f.Calculated = true;
  f = s.Fields.Add("FldFirstName", "FirstName", 500, 0, 900, 300);
  f.Calculated = true;
  f = s.Fields.Add("FldLastName", "LastName", 1500, 0, 900, 300);
  f.Calculated = true;
  f = s.Fields.Add("FldTitle", "Title", 2500, 0, 2400, 300);
  f.Calculated = true;
  f = s.Fields.Add("FldNotes", "Notes", 5000, 0, 8000, 300);
  f.Width = c1r.Layout.Width - f.Left;
  f.Calculated = true;
  f.CanGrow = true;
  f.Font.Size = 6;
  f.Align = FieldAlignEnum.JustTop;
  f = s.Fields.Add("FldLine", "", 0, 310, c1r.Layout.Width, 20);
  f.LineSlant = LineSlantEnum.NoSlant;
  f.BorderStyle = BorderStyleEnum.Solid;
  f.BorderColor = Color.FromArgb(100, 100, 100);
```

すべてのフィールドが計算フィールドであること、また各フィールドのText プロパティがソースレコードセット内のフィールド名に対応していることに注意してください。Calculated プロパティを **True** に設定すると、Text プロパティはテキストどおりに出力されるのではなく、データベースのフィールド名として認識されます。レポートのフィールド名がレコードセット内のフィールド名とは異なる一意の名前になるように、名前付け規則を決める必要があります。たとえば、"LastName" という名前のフィールドが2つあると、「Left(LastName,1)」という式はあいまいになります。この例では、レポートのフィールド名はすべて "Fld" で始まるという名前付け規則を採用しました。

また、"FldNotes" フィールドの**CanGrow** プロパティが **True** に設定され、他より小さいフォントが設定されていることにも注目してください。これは、データベースの "Notes" フィールドには多くのテキストが含まれており、レポートにそれらをすべて表示するためです。フィールドの高さを大きく設定して無駄なスペースをとるのではなく、CanGrow プロパティを **True** に設定して、コントロールがコンテンツに合わせてフィールドを拡張するようにしています。また、このフィールドを含むセクションのCanGrow プロパティも **True** に設定されるため、フィールドがセクションからはみ出ることはありま


Copyright © GrapeCity, Inc. All rights reserved.

せん。

9. これで、レポート定義は完了です。レポートを **C1PrintPreview** コントロールにレンダリングするには、次のコードを入力します。

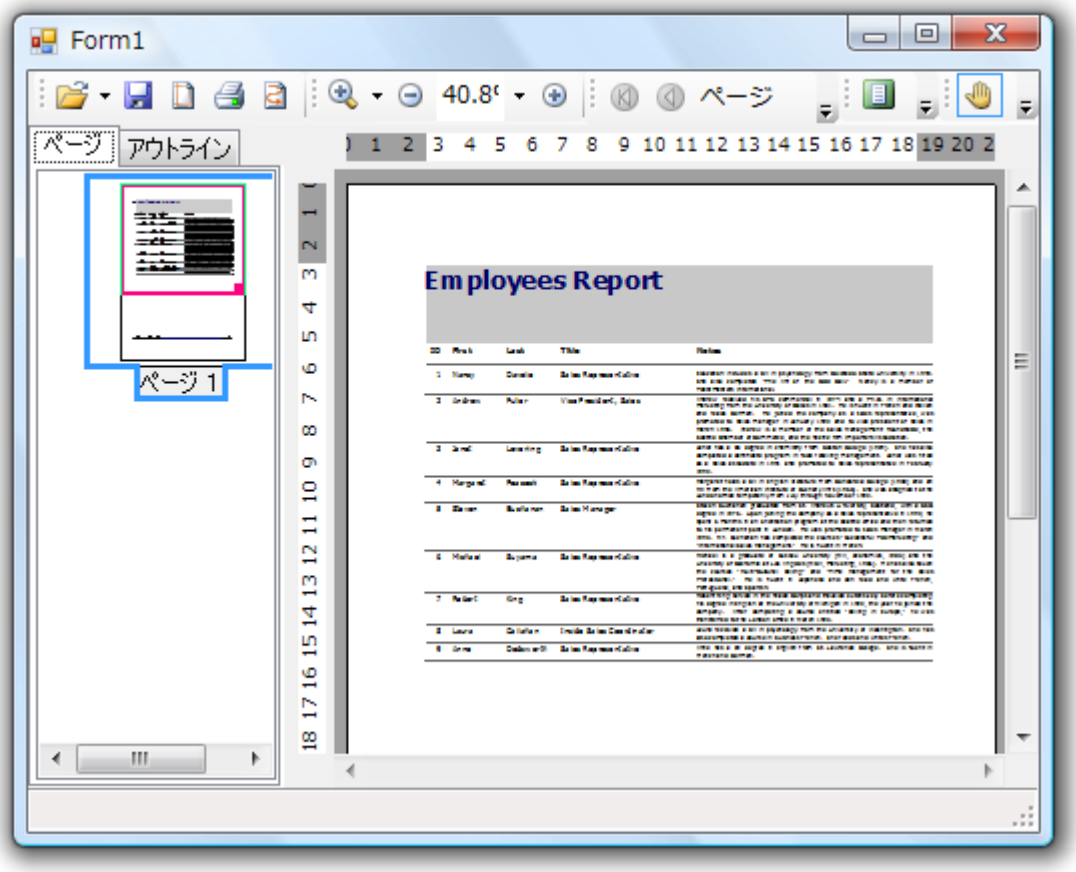
**Visual Basic コードの書き方**

```
Visual Basic
RenderEmployees()
' レポートを C1PrintPreview コントロールにレンダリングします
ppv.Document = c1r.Document
```

**C# コードの書き方**

```
C#
RenderEmployees();
// レポートを C1PrintPreview コントロールにレンダリングします
ppv.Document = c1r.Document;
```

基本レポートの外観は次のようになります。




## レポートデータの読み込み

レポートを作成するには、レポート定義の他に実際のデータが必要です。通常は、データベースからデータを取得しますが、他の場所から取得することもできます。

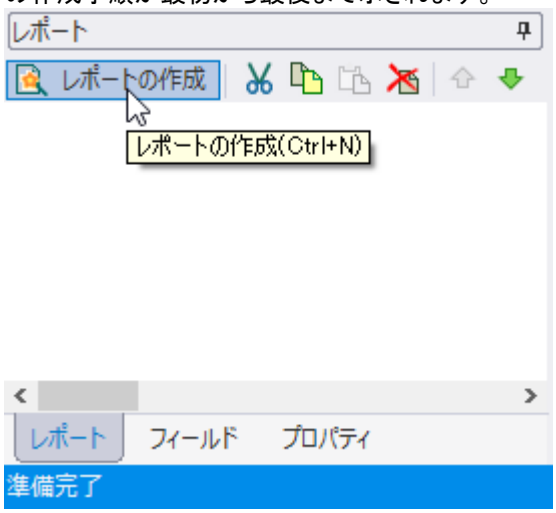

Copyright © GrapeCity, Inc. All rights reserved.

# データベースから読み込む

レポートデータをロードする最も簡単な方法は、**C1Report** コントロールの **DataSource**の **ConnectionString** プロパティと **RecordSource** プロパティを設定することです。これらのプロパティを設定すると、それを使用して、**C1Report** によってデータベースから自動的にデータがロードされます。それ以上の作業は必要ありません。

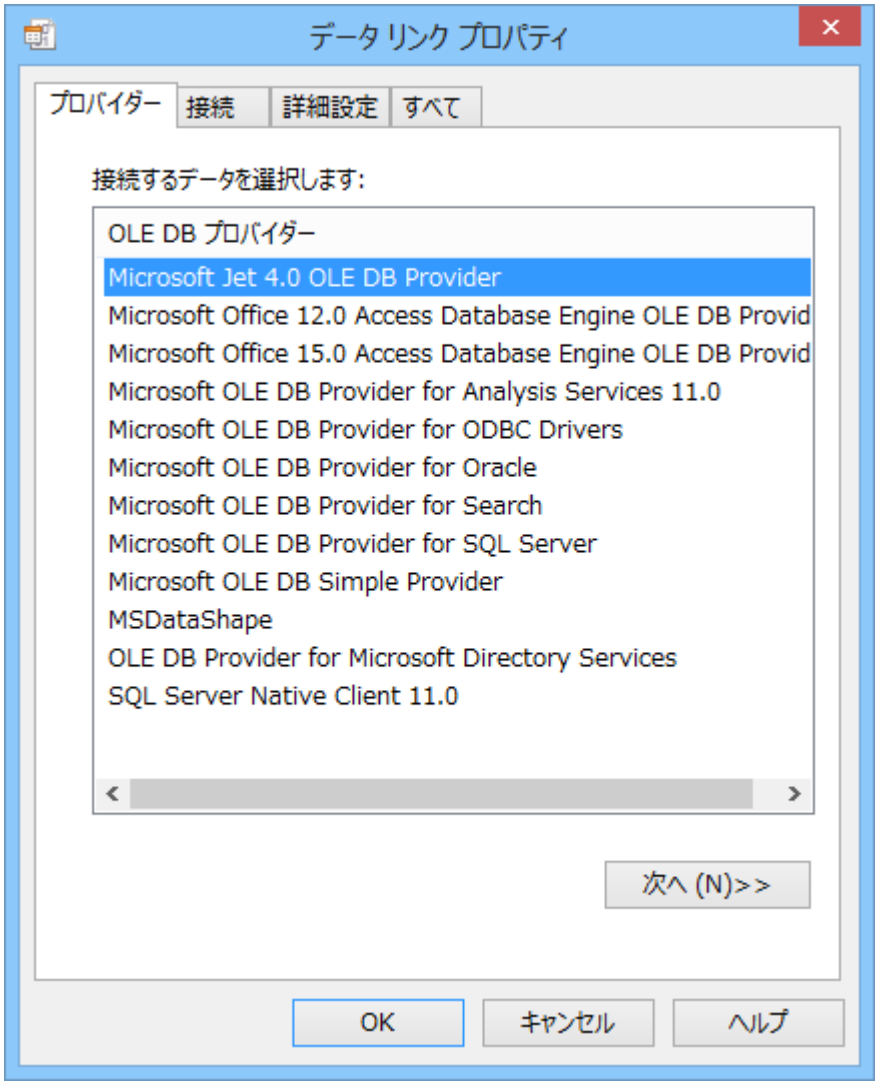
## データベースから読み込む

1. **C1Report デザイナ**で、[**新しいレポートの追加**]ボタンを選択します。C1Report ウィザードが表示され、新しいレポートの作成手順が最初から最後まで示されます。




2. ウィザードの最初の手順で、データソースを選択する必要があります。[**接続文字列の作成**]ボタン をクリックします。[**データリンクプロパティ**]ダイアログボックスが表示されます。
3. 最初に、データプロバイダを選択する必要があります。[**プロバイダ**]タブを選択し、リストからデータプロバイダを選択します。この例では、**Microsoft Jet 4.0 OLE DB Provider** を選択します。


Copyright © GrapeCity, Inc. All rights reserved.




4. [**次へ**]ボタンをクリックするか、[**接続**]タブを選択します。ここで、データソースを選択する必要があります。
5. データベースを選択するには、省略符ボタンをクリックします。[**Access データベースの選択**]ダイアログボックスが表示されます。この例では、**Nwind.mdb** を選択します。このディレクトリはデフォルトのインストールパスに基づいていますが、インストールパスを変更した場合は、パスが異なる場合があります。
6. [**開く**]をクリックします。接続をテストし、[**OK**]をクリックします。
7. [**OK**]をクリックして、[**Access データベースの選択**]ダイアログボックスを閉じます。
8. データソースを選択したら、テーブル、ビュー、またはストアドプロシージャを選択して実際のデータを提供できます。**RecordSource** の文字列は、次の2つの方法で指定できます。
   - [**テーブル**]オプションをクリックし、リストからテーブルを選択します。
   - [**SQL**]オプションをクリックし、エディタに SQL 文を入力(または貼り付け)します。

   たとえば、次のように記述します。
   ```
   select * from products inner join categories on categories.categoryid =
   products.categoryid
   ```
9. [**次へ**]をクリックします。ウィザードに従って残りの手順を実行します。

## ストアドプロシージャからのデータのロード

ストアドプロシージャ(sproc)は、アプリケーション全体の論理の一貫した実装を達成するのを支援し、パフォーマンスを向上させ、データベースのテーブルの詳細を知る必要性からユーザーを解放できます。ストアドプロシージャの主な利点の1つは、パラメータを渡してデータベースにレコードセットをフィルタさせることができる点です。これにより、より小さいデータセットが返され、レポートの操作がスピードアップして簡単になります。

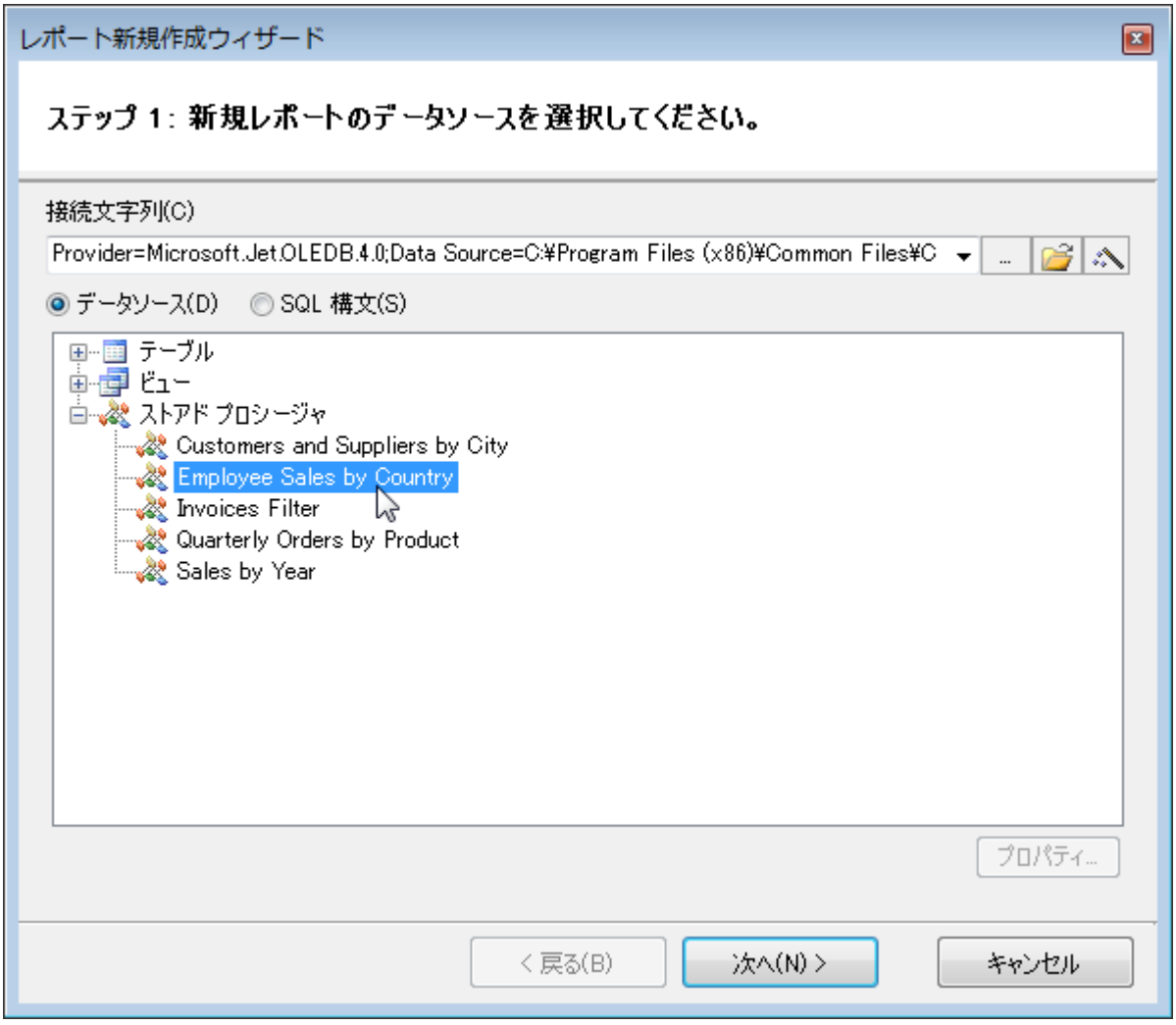

Copyright © GrapeCity, Inc. All rights reserved.

[**C1Report ウィザード**]のストアドプロシージャからレポートにデータを移植できます。[**C1Report ウィザード**]を開くには、次の1つを実行します。

- **C1ReportDesigner** アプリケーションから、「**レポート**」タブの〈**新しいレポート**〉ボタンをクリックします。
- Visual Studio で、**C1Report** コンテキストメニューから[**レポートの編集**]を選択します。
- Visual Studio で、[**C1Report タスク**]メニューから[**レポートの編集**]を選択します。

[**レポートの編集**]リンクへのアクセス方法の詳細については、「C1Report タスクメニュー」または「C1Report コンテキストメニュー」を参照してください。

ストアドプロシージャからレポートにデータを移植することは、SQL ステートメントや単純なテーブルを使用するのと何も違いはありません。**C1ReportWizard** の最初の画面で、〈...〉ボタンをクリックし、データソースを選択します。次に、使用可能な[**データソース**]のリストから[**ストアドプロシージャ**]を選択します。




[**次へ**]を選択して、ウィザードを続行します。

他のデータフォームのロードと同様に、次の2つのオプションがあります。

- DataSource の **ConnectionString** と **RecordSource** プロパティを使用し、データソースを選択できます。

  デザイナで、[DataSource]ダイアログボックスを使用し、接続文字列を選択(省略符号〈...〉ボタンをクリック)した後、リストから使用するテーブルまたは sproc を選択します。たとえば、次のようになります。

  ```
  connectionstring = "Provider=SQLOLEDB.1;Integrated Security=SSPI;" + "Persist
  Security Info=False;Initial Catalog=Northwind;Data Source=YOURSQLSERVER;"
  recordsource = "[Employee Sales by Country]('1/1/1990', '1/1/2010')"
  ```

  (この場合ストアドプロシージャの名前はスペースを含むため、角カッコ([])で囲っています。)


Copyright © GrapeCity, Inc. All rights reserved.

- 希望する任意のメソッドを使用してデータソースを作成した後、それを **DataSource** の **Recordset** プロパティに割り当てることができます。

  このメソッドは、コードの記述を必要とし、どこかにキャッシュしたデータを使用して複数のレポートを生成する場合に便利です。これは、前のメソッドをオーバーライドします（**ConnectionString**、**RecordSource**、および **Recordset** を指定した場合 **C1Report** は **Recordset** を使用します）。

  構文は使用する接続／アダプタのタイプ（OleDb、SQL、Oracle など）によって異なります。構文をすぐに取得する最も簡単な方法は、テーブルまたは sproc を Visual Studio の サーバーエクスプローラからフォームにドラッグする方法です。これにより、必要なすべての暗号要素が追加され、その後、コードを調べて、必要な部分を選択できます。

  ストアドプロシージャを名前によってデータソースとして指定できます。sproc がパラメータを持つ場合、それらをパラメータとして渡します。たとえば、MSSQL および ADVENTURE_WORKS.mdf データベースに対して構築されるレポート定義では、[**C1Report デザイナ**]で指定された SQL 要求は次のようになります（必要に応じて ADVENTUREWORKS_DATA.MDF のパスを調整）。

  ```
  PARAMETERS Empoyee Int 290;
  DECLARE @RC int
  DECLARE @EmployeeID int
  set @EmployeeID = [Empoyee]
  EXECUTE @RC = [C:\ADVENTUREWORKS_DATA.MDF].[dbo].[uspGetEmployeeManagers]
  @EmployeeID
  ```

# DataTable から読み込む

多くのアプリケーションでは、**C1Report**の外部にあるデータを操作したり、それを DataTable オブジェクトにロードする必要があります。このような場合は、これらのオブジェクトをレポートデータソースとして使用して、レポートのレンダリング時にデータを再ロードする無駄を省くことができます。

この方法は、次のようなアプリケーションでも役立ちます。

- セキュリティ制限により、接続文字列を非公開にする必要があり、データだけ（ソースではなく）を公開できる場合。
- データベースが OLE DB（C1Report で内部的に使用されるプロバイダ）によってサポートされていない場合。
- データをデータベース以外から取得する場合。代わりに、DataTable を作成し、カスタムコードを使ってデータを挿入します。

DataTable オブジェクトを C1Report のデータソースとして使用するには、レポート定義をロードし、DataTable を C1Report の **Recordset** プロパティに割り当てるだけです。次に例を示します。

**Visual Basic コードの書き方**

Visual Basic

```
' キャッシュまたはセキュリティで保護されたカスタムプロバイダから DataTable をロードします
Dim dt As DataTable = GetMyDataTable()

' レポート定義をロードします（データソースを設定する前）
c1r.Load(reportFile, reportName)

' DataTable を c1r1 コンポーネントのデータソースとして使用します
c1r.DataSource.Recordset = dt
```

**C# コードの書き方**

C#

```
// ャッシュまたはセキュリティで保護されたカスタムプロバイダから DataTable をロードします
DataTable dt = GetMyDataTable();
```


Copyright © GrapeCity, Inc. All rights reserved.

```
// レポート定義をロードします（データソースを設定する前）
c1r.Load(reportFile, reportName);

// DataTable を c1r1 コンポーネントのデータソースとして使用します
c1r.DataSource.Recordset = dt;
```

# カスタムデータソースから読み込む

カスタムオブジェクトをデータソースとして使用できます。必要な条件は、カスタムオブジェクトが **IC1ReportRecordset** インタフェースを実装することだけです。

i独自のデータソースオブジェクトを実装するには、**IC1ReportRecordset** インタフェースを実装するオブジェクトを作成する必要があります。カスタムデータソースオブジェクトを作成したら、そのインスタンスを作成し、それをRecordset プロパティに割り当てます。完全なカスタムデータソースオブジェクトを作成する例については、CustomData サンプルを参照してください。

カスタムデータソースオブジェクトを使用するには、レポート定義をロードし、そのオブジェクトを C1Report の **Recordset** プロパティに割り当てます。次に例を示します。

**Visual Basic コードの書き方**

Visual Basic

```
' カスタムデータソースオブジェクトを取得します
Dim rs As IC1ReportRecordset = CType(GetMyCustomDataSource(), IC1ReportRecordset)

' レポート定義をロードします（データソースを設定する前）
c1r.Load(reportFile, reportName)

' c1r1 コンポーネントでカスタムデータソースオブジェクトを使用します
c1r.DataSource.Recordset = rs
```

**C# コードの書き方**

C#

```
// カスタムデータソースオブジェクトを取得します
IC1ReportRecordset rs = (IC1ReportRecordset)GetMyCustomDataSource();

// レポート定義をロードします（データソースを設定する前）
c1r.Load(reportFile, reportName);

// c1r1 コンポーネントでカスタムデータソースオブジェクトを使用します
c1r.DataSource.Recordset = rs;
```

# グループ化とソート

このセクションでは、積算合計、集計式、グループ化、およびソートを使用してレポート内のデータを構成する方法について説明します。

# データのグループ化

基本レイアウトの設計後、特定のフィールドまたはその他の基準によりレコードのグループ化を指定して、レポートを読みやすくできます。グループ化によって、レコードのグループを視覚的に分類し、各グループの紹介および概要データを表示できます。区切

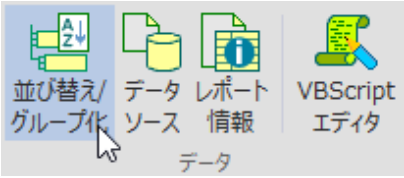

Copyright © GrapeCity, Inc. All rights reserved.

りグループは、グループ化式に基づきます。通常、この式は、1つ以上のレコードセットフィールドに基づきますが、必要に応じて複雑にすることができます。

**C1ReportDesigner** アプリケーションを使用するか、またはコードを使用して、レポート内のデータをグループ化できます。

**C1ReportDesigner を使用したグループ化とソートの追加**

グループは、[グループヘッダー]と[グループフッター]セクションの表示を予定していない場合であっても、データのソートにも使用されます。**C1ReportDesigner** アプリケーションを使用して、グループをレポートに追加できます。

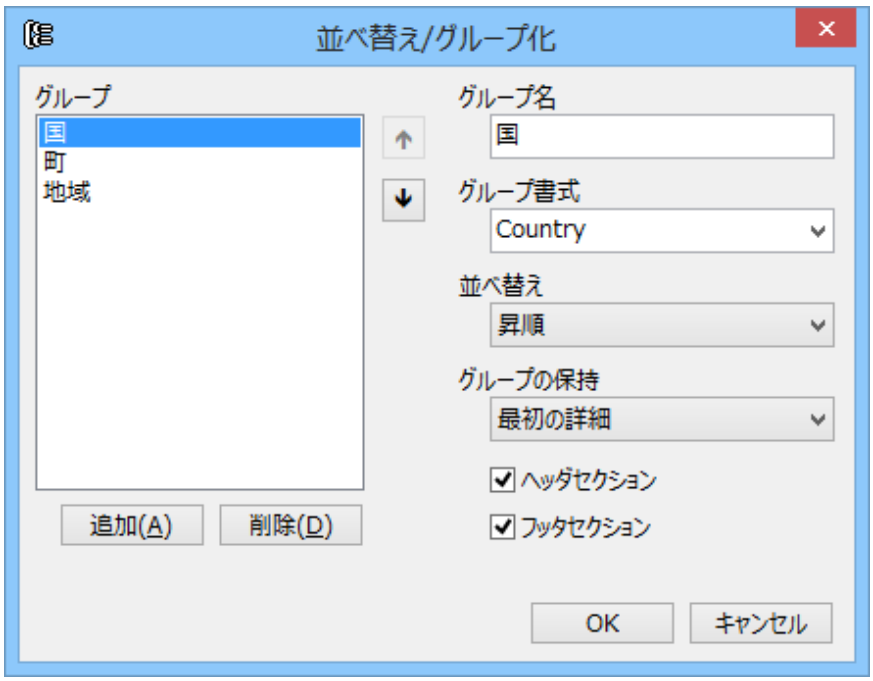



レポートにグループを追加したり、レポート内のグループを編集したりするには、以下の手順を実行します。

1. **C1ReportDesigner** アプリケーションを開きます。詳細については、「Visual Studio からの C1Report デザイナへのアクセス」を参照してください。
2. 「デザイン」タブにある〈**ソートとグループ化**〉ボタンをクリックします。[**ソートとグループ化**]ダイアログボックスが表示されます。このダイアログボックスを使用して、グループの作成、編集、順序変更、および削除を行えます。
3. グループを作成するには、〈**追加**〉ボタンをクリックし、新しいグループのプロパティを設定します。[グループ化]フィールドは、レポート内でレコードをグループ化する方法を定義します。単純なグループ化では、ドロップダウンリストから直接フィールドを選択できます。複雑なグループ化では、グループ化式を入力できます。たとえば、[**国**]を使用して国別にグループ化したり、[**Left(Country, 1)**]を使用して国の頭文字別にグループ化したりできます。
4. このレポートに沿って、[**グループ化**]式で[**国**]を選択します。
5. 次に、希望するソートのタイプを選択します(この例では、[**昇順**])。新しいグループに視覚的な[グループヘッダー]と[グループフッター]セクションを設定するかどうか、およびグループを1ページ上に一緒に表示するかどうかも指定できます。

**メモ**:メモまたはバイナリ(オブジェクト)フィールドを、グループ化およびソートに使用することはできません。これは OLEDB によって与えられた制限です。

この時点で、[**ソートとグループ化**]ダイアログボックスは以下のように表示されます。

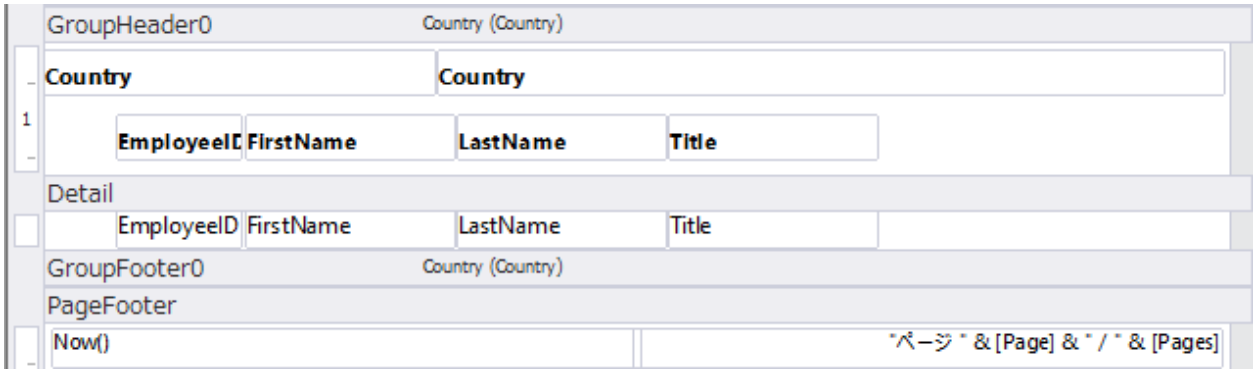



さらに追加する場合は、[**グループ**]リストの右側にある矢印ボタンを使用して、これらの順序を変更できます。この結果、レポート内の[**グループヘッダー**]と[グループフッター]セクションの位置が自動的に調整されます。フィールドを削除するには、〈**削除**〉ボタンを使用します。

フィールドの配置が完了したら、〈**OK**〉をクリックしてダイアログボックスを閉じて、デザイナで変更内容を確認します。新しいグ


Copyright © GrapeCity, Inc. All rights reserved.

ループに、新しい[グループヘッダー]と[グループフッター]セクションが表示されます。この時点では、両方とも高さがゼロですが、端部をマウスでドラッグして、これらを拡大できます。[グループヘッダー]が表示され、[**グループフッター**]が表示されないことに注意してください。これは、ダイアログのボックスで〈グループヘッダー〉ボタンが ON になり、〈**グループフッター**〉ボタンが OFF のままになっているためです。表示されないセクションは、ハッチングパターンで表示され、不可視であることを示します。




新しいセクションの上部にマークされた黄褐色のバーには、セクション名とグループの **GroupBy** プロパティの値を含むラベルがあります。

グループの動作を確認するには、〈**データフィールドの追加**〉ボタン

をクリックし、メニューから[**国**]を選択して、新規に作成した[グループヘッダー]セクション内の領域をマークします。新しいフィールドをクリックし、それを選択して、新しいフィールドが少し目立つように、その**Font**プロパティを変更します。

**コードを使用したグループ化とソートの追加**

効果的なレポートは単にデータを表示するのではなく、整理された状態で表示します。**C1Report** は、*グループを使用して*データをグループ化し、ソートします。この動作を説明するために、「**レポート定義の作成**」トピックのコードに戻り、社員を国別にグループ化します。

次のコードは、レコードを国別にソートし、グループ化するグループオブジェクトを作成します。

**Visual Basic コードの書き方**

Visual Basic

```
If chkGroup.Checked Then

  ' 社員を昇順で国別にグループ化します
  Dim grp As Group
  grp = c1r.Groups.Add("GrpCountry", "Country", SortEnum.Ascending)

  ' 新しいグループの[グループヘッダー]セクションを書式設定します
  With grp.SectionHeader
    .Height = 500
    .Visible = True
    f = .Fields.Add("CtlCountry", "Country", 0, 0, c1r.Layout.Width, 500)
    f.Calculated = True
    f.Align = FieldAlignEnum.LeftMiddle
    f.Font.Bold = True
    f.Font.Size = 12
    f.BorderStyle = BorderStyleEnum.Solid
    f.BorderColor = Color.FromArgb(0, 0, 150)
    f.BackStyle = BackStyleEnum.Opaque
    f.BackColor = Color.FromArgb(150, 150, 220)
```


Copyright © GrapeCity, Inc. All rights reserved.

```
    f.MarginLeft = 100
  End With

  ' 各国内のファーストネームによって社員をソートします
  c1r.Groups.Add("GrpName", "FirstName", SortEnum.Ascending)
End Ifデータのグループ化
```

**C# コードの書き方**

C#

```
if (chkGroup.Checked)
{
  // 社員を昇順で国別にグループ化します
   Group grp = c1r.Groups.Add("GrpCountry", "Country", SortEnum.Ascending);

  //  新しいグループの[グループヘッダー]セクションを書式設定します
  s = grp.SectionHeader;
    s.Height = 500;
    s.Visible = true;
    f = s.Fields.Add("CtlCountry", "Country", 0, 0, c1r.Layout.Width, 500);
    f.Calculated = true;
    f.Align = FieldAlignEnum.LeftMiddle;
    f.Font.Bold = true;
    f.Font.Size = 12;
    f.BorderStyle = BorderStyleEnum.Solid;
    f.BorderColor = Color.FromArgb(0, 0, 150);
    //f.BackStyle = BackStyleEnum.Opaque;
    f.BackColor = Color.Transparent;
    f.BackColor = Color.FromArgb(150, 150, 220);
    f.MarginLeft = 100;

  // 各国内のファーストネームによって社員をソートします
  c1r.Groups.Add("GrpName", "FirstName", SortEnum.Ascending);
}
```

各グループには、[グループヘッダー]と[グループフッター]セクションがあります。デフォルトでは非表示ですが、上記のコードは[グループヘッダー]セクションを可視化し、グループを定義する国を表示します。次に、国を示すフィールドを追加します。新しいフィールドは、無地の背景色で表示されます。

最後に、コードは、第2グループを追加し、各国内でファーストネームによって社員をソートします。このグループは、ソートにのみ使用されます。このため、[グループヘッダー]と[グループフッター]セクションは非表示のままになります。

これで変更が完了しました。新しいレポートを表示するには、Render メソッドを含むルーチンを終了する必要があります。次のコードを **btnEmployees_Click** イベントハンドラに追加します。

**Visual Basic コードの書き方**

Visual Basic

```
' レポートを PrintPreviewControl に表示します
ppv.Document = c1r.Document
```

**C# コードの書き方**

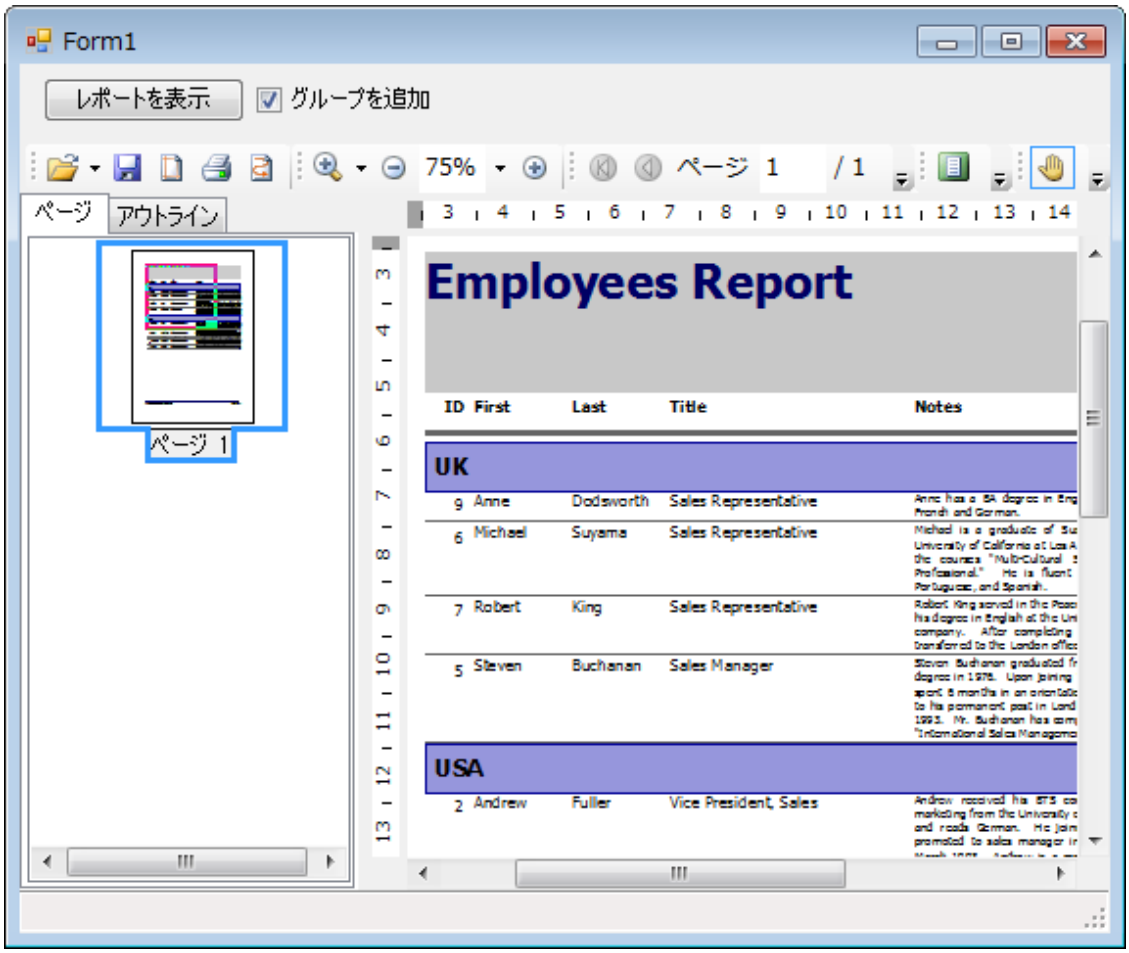
C#

```
// レポートを PrintPreviewControl に表示します
```


Copyright © GrapeCity, Inc. All rights reserved.

```
ppv.Document = c1r.Document;
```

グループオブジェクトを使用したレポートの例を以下に示します。




# データをソートする

レポート内でデータをソートする方法は2つあります。

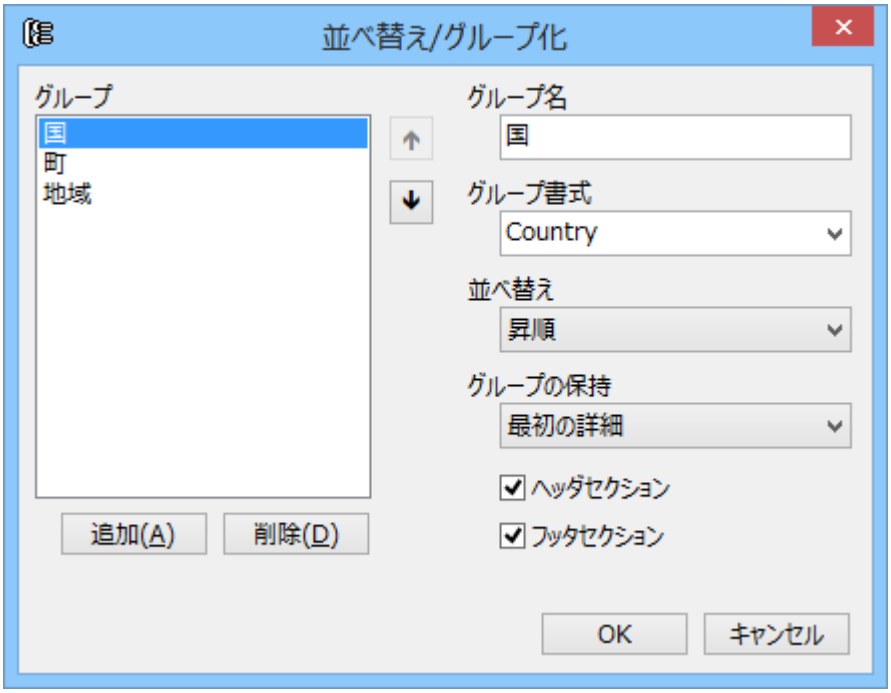
- データソースオブジェクト自体をソートするか（SQL 文の ORDER BY 句を使用）、レポートにグループを追加します。
- そのグループの **GroupBy** プロパティと **Sort** プロパティを使って各グループのソート方法を指定できます。

グループのソートは、**DataView.Sort** プロパティを使って実行されます。このプロパティは、列名に対する式ではなく、列名のリストだけを受け取ります。そのため、グループ式が「DatePart("yyyy", dateColumn)」である場合、通常は、日付の年に基づいてソートされることが期待されますが、実際は *dateColumn* フィールド内の日付に基づいてソートされます。

年に基づいてソートするには、（SQL 文を変更して）データテーブルに計算列を追加し、この計算列に基づいてグループ化またはソートします。サンプルを含むこの XML の説明については、Sort プロパティを参照してください。

次の図は、C1Report デザイナの並べ替え／グループ化エディタです。グループソートを指定できるフィールドに注目してください。


Copyright © GrapeCity, Inc. All rights reserved.




両方の方法を使用する場合は、レポートグループに設定されたソートが優先されます。このソートは、データソースからデータが取得された後に適用されます。

**注意: CommonTasks.xml** にあるレポート「19: ソート」を参照してください。

# レポートへ積算合計を追加する

**C1Report** フィールドオブジェクトには、**RunningSum** プロパティがあるため、グループまたはレポート全体の積算合計を簡単に維持できます。

## グループ全体の積算合計の追加

グループ全体の積算合計を維持するには、以下のタスクを実行します。

1. **C1ReportDesigner** アプリケーションを開きます。**C1ReportDesigner** へのアクセス方法の詳細については、「Visual Studio からの C1Report デザイナへのアクセス」を参照してください。
2. 新しいレポートを作成するか、既存のレポートを開きます。**C1ReportDesigner** アプリケーションでレポートを準備したら、レポートのプロパティを変更できます。
3. 〈**デザイン**〉ボタンをクリックし、レポートの編集を開始します。
4. デザインモードで、プロパティウィンドウのドロップダウンリストからレポートを選択します。レポートの使用可能なプロパティが表示されます.
5. 集計フィールドをレポートに追加します。
    1. デザイナツールバーから〈**集計フィールドの追加**〉ボタンをクリックします。
    2. **VBScript エディタ**で、次のスクリプトを入力します。Sum(ProductSalesCtl)
    3. マウスをレポートの GroupHeader セクション上にドラッグします。カーソルが十字に変わります。クリックとドラッグによって新しいフィールドが占める矩形を定義した後、ボタンを解放して新しいフィールドを作成します。
6. RunningSum プロパティを **SumOverGroup** に設定します（このプロパティを表示するには、プロパティフィルタをオフにする必要があります。これはプロパティウィンドウのじょうごアイコンです）。

## レポート全体の積算合計の追加

TC1Report フィールドオブジェクトには、グループまたはレポート全体の積算合計を維持するために便利な RunningSum プロ


Copyright © GrapeCity, Inc. All rights reserved.

パティがあります。複数のページにまたがる積算合計を維持するには、スクリプトを使用する必要があります。たとえば、レポートに **pageSum** フィールドを追加し、次に示すスクリプトを使用してフィールドを更新します。それには、次のタスクを実行します。

1. **C1Report デザイナ**を開き、新しいレポートを作成するか、既存のレポートを開きます。**C1Report デザイナ**にレポートを開いたら、レポートのプロパティを変更することができます。
2. レポートの編集を開始するには、[**設計**]ボタンをクリックします。
3. 設計モードで、プロパティウィンドウの上にあるドロップダウンリストからレポートを選択します。レポートのプロパティが表示されます。
4. **OnPage** プロパティを見つけ、その横の空のフィールドをクリックしてから、省略符ボタンをクリックします。
5. VBScript エディタが表示されます。コードエディタで次の式を入力します。`' VBScript: Report.OnPage pageSum = 0`
6. プロパティウィンドウの上のドロップダウンリストから[**詳細**]を選択します。詳細セクションのプロパティが表示されます。
7. OnPrint プロパティを見つけ、その横の空のフィールドをクリックしてから、省略符ボタンをクリックします。
8. **VBScript エディタ**が表示されます。コードエディタで次の式を入力します。`' VBScript: Detail.OnPrint pageSum = pageSum + UnitsInStock`

> **注意:CommonTasks.xml** にあるレポート「17: 積算合計」を参照してください

# 小計などの集計の追加

**C1Report**は、すべての計算フィールドで集計式をサポートします。集計式には、通常使用される **Sum、Avg、Min、Max、Count、Range、StDev** などが含まれます。

すべての集計関数は引数として式を受け取り、レポート内の位置によって特定される範囲内でその式を評価します。たとえば、グループヘッダまたはグループフッタ内の集計は、そのグループを対象範囲にします。レポートヘッダーまたはレポートフッター内の集計は、レポート全体を対象範囲にします。

たとえば、次の集計式は、集計の範囲(グループまたはレポート)内で Sales フィールドのすべての値の合計を返します。:`Sum(Sales)`

次の集計式は、レポート内のすべての値に対して消費税の合計を返します(消費税を 8.5% と仮定)。:`Sum(Sales * 0.085)`

*domain* という名前の2番目の引数で、集計の範囲を狭めることができます。*domain* 引数は、現在の範囲内の各値を集計計算に入れる必要があるかどうかを判断するための式です。

たとえば、次の集計式は、カテゴリ1の製品に対して *Sales* フィールド内のすべての値の合計を返します。:`Sum(Sales, Category = 1)`

次の集計式は、$10,000 を超える売上の数を返します。:`Count(*, Sales > 10000)`

> **注意: CommonTasks.xml** にあるレポート「13: 小計などの集計」を参照してください。

# クロスタブ型レポートを作成する

クロスタブ型レポートは、データを2次元(下方向と横方向)にグループ化します。クロスタブ型レポートは、情報を相互参照する形式で大量のデータをまとめる場合に便利です。

クロスタブ型レポートを作成するには、通常、最初に GROUP BY クエリーでデータを行に集約し、次に転置(ピボット)サービスを使ってグループ化された列を作成します。転置サービスは、データベースサーバー自体から提供されるか、カスタムプログラムを使用するか、**C1Report** の組み込みドメイン集計を使用できます。

どのような場合でも、クロスタブ型レポートの最も重要な要素は、元のデータの集約ビューです。たとえば、典型的な集約ビューは次のようになります。


Copyright © GrapeCity, Inc. All rights reserved.

| 年 | 四半期 | 金額 |
|---|---|---|
| 1990 | 1 | 1.1 |
| 1990 | 2 | 1.2 |
| 1990 | 3 | 1.3 |
| 1990 | 4 | 1.4 |
| 1991 | 1 | 2.1 |
| 1991 | 2 | 2.2 |
| 1991 | 3 | 2.3 |
| 1991 | 4 | 2.4 |

各四半期に対応する列を追加し、値を新しい列に集約することで、このデータを転置します。

| 年 | 合計 | 四半期1 | 四半期2 | 四半期3 | 四半期4 |
|---|---|---|---|---|---|
| 1990 | 5 | 1.1 | 1.2 | 1.3 | 1.4 |
| 1991 | 9 | 2.1 | 2.2 | 2.3 | 2.4 |

それには、**C1Report** の集計関数を使用します。レポートを年ごとにグループ化します。詳細セクションを非表示にし、グループヘッダに次の集計を入れます。

| 年 | 合計 | 四半期1 | 四半期2 | 四半期3 | 四半期4 |
|---|---|---|---|---|---|
| [Year] | Sum(Amount) | Sum(Amount, Quarter=1) | Sum(Amount, Quarter=2) | Sum(Amount, Quarter=3) | Sum(Amount, Quarter=4) |

最初の集計は、当年の売上合計を計算します。四半期ごとの集計では、集計を特定の四半期に限定するためにドメインを指定します。

**CommonTasks.xml** にあるレポート「20: クロスタブ型レポート」を参照してください。

# VBScript 式

VBScript 式は、レポートのコンテンツを取得、計算、表示、グループ化、ソート、フィルタ、パラメータ化、および書式設定するためにレポート定義内で広く使用されます。いくつかの式は自動的に作成されます。たとえば、ツールボックスからフィールドをレポートのセクションにドラッグすると、テキストボックスにそのフィールドの値を取得する式が表示されます。ただし、ほとんどの場合、レポートに機能を追加するには独自に式を作成する必要があります。

VBScript の式と文には次の違いがあります:

- 式は値を返します。これを **Field.Text** プロパティなどに割り当てることができます。
  ```
  Field1.Calculated = true
  Field1.Text = "iif( 1=1, 1+2, 1+3 )"
  ```
- 文は値を返しません。**OnFormat** などのイベントプロパティに割り当てることができます。たとえば、次のように記述します。
  ```
  c1r.OnOpen = "if 1=1 then msgbox("OK!!!") else msgbox("ooops")"
  ```

**C1Report** は、計算フィールドでの式の評価やレポートイベントの処理を VBScript に依存しています。レポートイベントとコントロールイベントの違いについては後で説明します。

VBScript は多機能の言語で、C1Report の式を記述する際は、VBScript のほとんどのメソッドと関数にアクセスできます。VBScript 固有の機能については、Microsoft Developer's Network (MSDN)をはじめ、多くの書籍や Web サイトで説明され


Copyright © GrapeCity, Inc. All rights reserved.

ているため、本マニュアルでは説明しません。)

新しい VBScript エディタでグローバルスクリプトを記述できます。このエディタを使用して、レポート全体からアクセス可能な VBScript の関数やサブルーチンを定義できます。エディタ内で、スクリプトを切り替えたり、フォントや色などのオプションを変更することができます。また、構文チェック、定義済み VBScript 関数、再編成されたスクリプト関数などの高度な機能を使用して、直感的なスクリプティングを簡単に行うことができます。

**[VBScript エディタ]**オプションを使用してグローバルスクリプトを記述するには

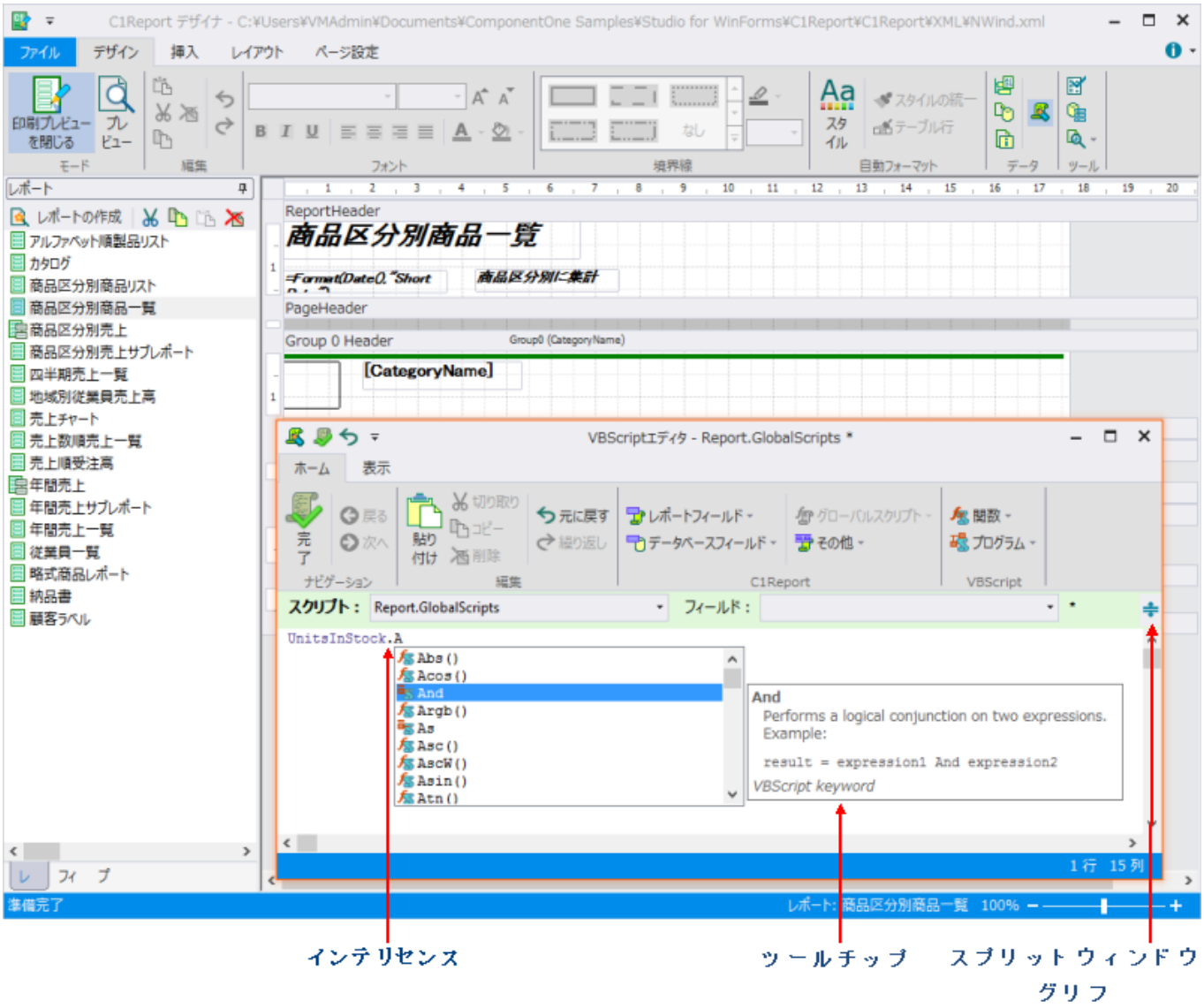
1. C1ReportDesigner の[ホーム]タブに移動します。
2. **[VBScript エディタ]**をクリックし、次の例のようなグローバルスクリプトを記述します。

```
function Backcolor()
Detail.Backcolor=rgb(255,0,0)
end function
```

次のように、**C1ReportDesigner** の **GlobalScripts** プロパティを使用してグローバルスクリプトを記述することもできます。

1. グローバルスクリプトを記述するレポートを選択します。
2. レポートの **GlobalScripts** プロパティに移動し、省略符をクリックします。これで、**[VBScript エディタ]**ダイアログボックスが開きます。
3. 上のように、**VBScript エディタ**でグローバルスクリプトを記述します。


Copyright © GrapeCity, Inc. All rights reserved.




これで、レポート全体から使用できるグローバル関数 'Backcolor()' が定義されます。

The VBScript Editor has the following additional features:

- VBScript エディタには、次の追加機能があります。
  - **IntelliSense**：VBScript エディタがサポートするスクリプトの自動コード補完プロンプトを提供します。VBScript エディタの IntelliSense には、次の機能があります。
    - IntelliSense ウィンドウには、使用可能な語の状況依存リストと共に、VBScript 関数およびキーワードに関する詳細なヘルプが小さなツールチップまたはヘルプウィンドウに表示されます。詳細ヘルプ内の斜体フォントは、基本的に、現在の項目が属するカテゴリを示します（VBScript 関数、C1Report 集計スクリプト関数、.NET オブジェクトプロパティなど）。
    - DataSource.Filter の編集時、エディタは「**式エディタ - DataSource.Filter**」として開かれ、IntelliSense には、エディタで使用できるキーワードまたは関数と、対応するヘルプが表示されます。
    - IntelliSense エントリに関連付けられているアイコンは、エントリのタイプを示します。アイコンのカラーパレットは、VBScript、レポート組み込み要素、DataSource.Filter によって異なります。
    - ユーザーがキー入力し、Intellisense ウィンドウが開くと、入力された文字に基づいてリストがフィルタ処理されます。たとえば、「t」と入力すると、文字「t」を含む語だけが表示され、「te」と入力すると、「te」を含む語にさらに絞り込まれます。
    - IntelliSense ウィンドウでバックスペースキーを押すと、最後のフィルタを元に戻します。
    - 角かっこ（[）キーを押すと、使用可能なデータベースフィールドのリストが表示されます。


Copyright © GrapeCity, Inc. All rights reserved.

- レポート、フィールド、セクションなどのオブジェクトの名前の後にピリオド(.)キーを押すと、そのオブジェクトで使用できる .NET プロパティが表示されます。
- [Ctrl]+[J]キー、[Ctrl]+[Space]キー、または文字以外のキーの後に文字キーを押すと、使用可能な VBScript 関数、キーワードなどのリストが状況に応じて表示されます。

- **分割ウィンドウ**: 同じ VBScriptEditor 内で 2 つの同じまたは異なるスクリプトを表示または記述できます。デフォルトでは、VBScript エディタは単一ウィンドウで開きます。
  **分割ウィンドウに切り替えるには**
  分割ウィンドウモードに切り替えるには、分割ウィンドウグリフをクリックして下へドラッグすると、上部にもう 1 つエディタが開きます。ウィンドウのサイズは、ウィンドウ間の分割線をドラッグして変更できます。
  **単一ウィンドウに戻るには**
  ウィンドウの右上隅にある[x]グリフをクリックすると、上側のウィンドウが閉じられ、分割モードがオフになり、下側のウィンドウがズームアウトします。リボンボタンの有効/無効の状態は、現在のウィンドウによって決まります。現在のウィンドウは、薄緑のキャプションバーで示されます。分割ウィンドウモードには、次の追加機能があります。
  - **[F6]**キーを押すと、2 つのウィンドウを切り替えることができます。
  - 分割ウィンドウグリフまたは分割線を上側のウィンドウの上部までドラッグすると、分割ウィンドウモードの上側のウィンドウを非表示にできます。

**C1Report** は、追加のオブジェクト、変数、および関数を公開することで、VBScript を拡張します。以下のセクションでは、これらの拡張機能について説明します。

> **メモ:VBScript エディタの[グローバルスクリプト]**ドロップダウンは、以前にレポートでグローバルスクリプトを定義した場合にのみ有効になります。

# オブジェクトと変数

The following tables detail VBScript elements, objects, and variables.

### 演算子

次の表に、VBScript の演算子を一覧します。

| 演算子 | 説明 |
|---|---|
| And | 2つの式の論理積を実行します。 |
| Or | 2つの式の論理和を実行します。 |
| Not | 式の論理否定を返します。 |
| Mod | 2つの数値で割り算を実行し、余りだけを返します。 |

### 予約シンボル

次の表に、VBScript の予約シンボルとそれらの使用方法を一覧します。

| キーワード | 説明 |
|---|---|
| True | **True** キーワードは、-1 と等しい値を持ちます。 |
| False | **False** キーワードは、0 と等しい値を持ちます。 |
| Nothing | 実オブジェクトからオブジェクト変数の関連付けを解除します。オブジェクト変数に **Nothing** を割り当てるには、**Set** 文を使用します。たとえば、次のように記述します。<br>Set MyObject = Nothing<br>複数のオブジェクト変数が同じ実オブジェクトを参照している場合があります。1つの |


Copyright © GrapeCity, Inc. All rights reserved.

| キーワード | 説明 |
|---|---|
| | オブジェクト変数に **Nothing** を割り当てた場合、その変数は実オブジェクトを参照しなくなります。複数のオブジェクト変数が同じオブジェクトを参照している場合、それらの変数が参照するオブジェクトに関連付けられているメモリおよびシステムリソースは、**Set** を使用することで明示的に、または最後のオブジェクト変数が **Nothing** に設定されることで暗黙的に、すべての変数が **Nothing** に設定された後でのみ解放されます。 |
| Null | **Null** キーワードは、変数に有効なデータが含まれていないことを示すために使用されます。 |
| vbCr | 印刷および表示機能を呼び出す際に、コード内で実際の値の代わりに次の定数を使用できます。 |
| vbCrLf | 印刷および表示機能を呼び出す際に、コード内で実際の値の代わりに次の定数を使用できます。 |
| vbLf | 印刷および表示機能を呼び出す際に、コード内で実際の値の代わりに次の定数を使用できます。 |
| vbFormFeed | 印刷および表示機能を呼び出す際に、コード内で実際の値の代わりに次の定数を使用できます。 |
| vbNewLine | 印刷および表示機能を呼び出す際に、コード内で実際の値の代わりに次の定数を使用できます。 |
| vbNullChar | 印刷および表示機能を呼び出す際に、コード内で実際の値の代わりに次の定数を使用できます。 |
| vbTab | 印刷および表示機能を呼び出す際に、コード内で実際の値の代わりに次の定数を使用できます。 |
| vbVerticalTab | 印刷および表示機能を呼び出す際に、コード内で実際の値の代わりに次の定数を使用できます。 |
| vbBlack | 黒色。値 = 0x0 |
| vbRed | 赤色。値 = 0xFF |
| vbGreen | 緑色。値 = 0xFF00 |
| vbYellow | 黄色。値 = 0xFFFF |
| vbBlue | 青色。値 = 0xFF0000 |
| vbMagenta | マゼンタ。値 = 0xFF00FF |
| vbCyan | シアン。値 = 0xFFFF00 |
| vbWhite | 白色。値 = 0xFFFFFF |

### 組み込み関数

VBScript の組み込み関数を次の表に一覧します。

| | | | | |
|---|---|---|---|---|
| Abs | Date | Iif | Minute | Sign |
| Acos | DateAdd | InputBox | Month | Space |
| Asc | DateDiff | InStr | MonthName | Sqr |
| Asin | DatePart | InStrRev | MsgBox | StrComp |


Copyright © GrapeCity, Inc. All rights reserved.

| | | | | |
|---|---|---|---|---|
| Atn | DateSerial | Int | Now | String |
| CBool | DateValue | IsDate | Oct | Tan |
| CByte | Day | IsEmpty | Pi | Time |
| CCur | Exp | IsNull | Replace | Timer |
| CDate | Fix | IsNumeric | RGB | TimeSerial |
| CDbl | Format | IsObject | Right | TimeValue |
| Chr | FormatCurrency | LCase | Rnd | Trim |
| CInt | FormatDateTime | Left | Round | TypeName |
| CLng | FormatNumber | Len | RTrim | UCase |
| Cos | FormatPercent | Log | Second | WeekDay |
| CSng | Hex | LTrim | Sgn | WeekDayName |
| CStr | Hour | Mid | Sin | Year |

VBScript 関数の詳細については、「MSDN のドキュメント」を参照してください。

**C1Report**では、次の VBScript 関数が含まれています。

- 集計関数（Sum、Average、StDev、Var、Count など）
- レポートフィールド名とデータベースフィールド名
- Page/Pages 変数
- レポートオブジェクト
- 文字列関数（Chr、Format など）
- データ変換（CBool、CByte など）
- 数値演算関数（Cos、Sin など）
- Function および Sub プロシージャ
- 組み込み関数
- 条件付きステートメント

**C1Report**では、次の VBScript 関数がサポートされていません。

- Arrays
- Select/Case 文

### 文のキーワード

VBScript 文のキーワードを次の表に一覧します。

| | | | | |
|---|---|---|---|---|
| If | ElseIf | To | While | Dim |
| Then | EndIf | Next | Wend | Redim |
| Else | For | Step | Const | |

### レポートフィールド名

**Field** オブジェクトの名前は、評価されるとオブジェクトへの参照を返します。これにより、フィールドのプロパティにアクセスできます。Field オブジェクトのデフォルトプロパティはValue なので、フィールド名自体がフィールドの現在の値を返します。次に例を示します。

```
MyField.BackColor = RGB(200,250,100)
MyField.Font.Size = 14
MyField * 2 ' (same as MyField.Value * 2)
```


Copyright © GrapeCity, Inc. All rights reserved.

**注記**:レポートフィールドにデータフィールドと同じ名前を付けると、レポートフィールドにアクセスできなくなります。

### レポートのセクション名

**Section** オブジェクトの名前は、評価されるとオブジェクトへの参照を返します。これにより、セクションのプロパティにアクセスできます。Section オブジェクトのデフォルトプロパティは**Name** です。次に例を示します。
If Page = 1 Then [Page Footer].Visible = False

### データフィールド名

レポートのソースデータセット内のフィールド名は、評価されると現在のフィールドの値を返します。フィールド名に全角文字、スペース、またはピリオドが含まれる場合は、フィールド名を[ ]で囲む必要があります。次に例を示します。

```
OrderID
UnitsInStock
[Customer.FirstName]
[Name With Spaces]
```

### Report 変数

#### Page 変数

**Page** プロパティの値を取得または設定します。このプロパティは、レポートの出力の開始時にコントロールによって初期化され、ページ区切りごとに1ずつ増加します。コードを使って値をリセットできます。次に例を示します。

```
If Country <> LastCountry Then Page = 1
LastCountry = Country
```

#### Pages 変数

レポートの出力の終了時に総ページ数に置き換えられるトークンを返します。読み取り専用のプロパティで、通常は、ページのヘッダフィールドまたはフッタフィールドで使用されます。次に例を示します。

```
"Page " & Page & " of " & Pages
```

### Report オブジェクト

コントロールオブジェクトへの参照を返します。これにより、スクリプトや式から **C1Report** オブジェクトモデル全体にアクセスできます。次に例を示します。

```
"Fields: " & Report.Fields.Count
```

### Cancel 変数

**True** に設定すると、レポートの出力処理をキャンセルできます。次に例を示します。

```
If Page > 100 Then Cancel = True
```

## 互換性のための関数

Visual Basic や Microsoft Access(VBA)で記述されたコードとの互換性を高めるために、**C1Report** は、**IIf** および **Format** という VBScript では使用できない2つの関数を公開しています。

**IIf** は、論理式を評価し、その結果に基づいて2つの値のうちの1つを返します。次に例を示します。

**VBScript**

```
Iif(SalesAmount > 1000, "Yes", "No")
```

**Format** は、書式設定式に含まれる指定に従って、値を書式設定された文字列に変換します。値には、数値、日付、時刻、


Copyright © GrapeCity, Inc. All rights reserved.

ブール値、文字列などを使用できます。書式は、Visual Basic または VBA で使用される書式文字列に似た構文で、文字列として構築されます。

次の表で、書式文字列に使用される構文について説明します。

| 値の型 | 書式文字列 | 説明 |
|---|---|---|
| **Number** | Percent, % | 数値をパーセント値として書式設定します（小数点以下は0または2桁）。: `Format(0.33, "Percent") = "33%"`<br>`Format(0.3333333, "Percent") = "33.33%"` |
| | #,###.##0 | マスクを使って数値を書式設定します。次の記号が認識されます。<br># 桁のプレースホルダ。<br><br>0常に表示される桁のプレースホルダ。<br>, 1000 単位の区切り。<br>( 負の値を( )で囲む。<br>% パーセント値として書式設定する。<br>`Format(1234.1234, "#,###.##") = "1,234.12"`<br>`Format(-1234, "#.00") = "(1234.12)"`<br>`Format(.1234, "#.##") = ".12"`<br>`Format(.1234, "0.##") = "0.12"`<br>`Format(.3, "#.##%") = "30.00%"` |
| **Currency** | Currency, $ | 数値を通貨値として書式設定します。`Format(1234, "$") = "$1,234.00"` |
| **Boolean** | Yes/No | “Yes” または “No” を取得します。 |
| **Date** | Long Date | `Format(#12/5/1#, "long date") = "December 5, 2001"` |
| | Short Date | `Format(#12/5/1#, "short date") = "12/5/2001"` |
| | Medium Date | `Format(#12/5/1#, "medium date") = "05-Dec-01"` |
| | q,m,d,w,yyyy | 日付の一部（四半期、月、月通算日、年通算週、年など）を取得します。`Format(#12/5/1#, "q") = "4"` |
| **String** | @@-@@/@@ | マスクを使って文字列を書式設定します。アットマーク（@）は、1つの文字（アットマーク（@）が1つだけの場合は、値の文字列全体）を表すプレースホルダです。他の文字は、リテラルとして認識されます。<br>`Format("AC55512", "@@-@@/@@") = "AC-555/12"`<br>`Format("AC55512", "@") = "AC55512"` |
| | @;Missing | 値が null または空文字列ではない場合は、セミコロン(;)の左側の書式文字列を使用します。null または空文字列の場合は、セミコロン(;)の右側の部分を取得します。<br>`Format("@;Missing", "UK") = "UK"` |


Copyright © GrapeCity, Inc. All rights reserved.

| 値の型 | 書式文字列 | 説明 |
|---|---|---|
| | | `Format("@;Missing", "") = "Missing"` |

**メモ**:VBScript には、独自の組み込み書式設定関数(**FormatNumber**、**FormatCurrency**、**FormatPercent**、**FormatDateTime** など)があります。ここで説明した VBA スタイルの **Format** 関数の代わりに、これらの関数を使用することもできます。

# 集計関数の使い方

集計関数は、出力されるグループのデータをまとめるために使用されます。レポートヘッダフィールドで使用された式は、データセット全体の集計値を返します。グループのヘッダまたはフッタで使用された式は、そのグループの集計値を返します。

**C1Report** の集計関数はすべて、次の2つの引数を受け取ります。

- グループ全体で集計される VBScript 式を含む文字列。
- フィルタ(ドメイン集計)として使用される VBScript 式を含むオプションの文字列。フィルタ式は、各値が集計される前に評価される。フィルタが **False** を返すと、その値はスキップされて集計結果に含まれない。

C1Report では、次の集計関数が定義されています。

| 値の型 | 説明 |
|---|---|
| Avg | 現在のグループ内の式の平均値。たとえば、次の式はグループ全体の平均売上高と特定の製品の平均売上高を計算します。`Avg(SalesAmount)`<br>`Avg(SalesAmount, ProductType = 3)` |
| Sum | グループ内のすべての値の合計。 |
| Count | グループ内の、値が null でないレコードの数。式にアスタリスク(*)を使用すると、すべてのレコードを含めることができる。たとえば、次の式は、住所の値が有効な(null ではない)従業員の数と従業員の総数をカウントします。`Count(Employees.Address)`<br>`Count(*)` |
| CountDistinct | グループ内の、値が null でないレコードの数。 |
| Min, Max | 式の最小値および最大値。たとえば、次の式は売上高の最大値を取得します。`"Min Sale = " & Max(SaleAmount)` |
| Range | 式の最小値と最大値の範囲。 |
| StDev, Var | 現在のグループ内の、式の標準偏差および分散。これらの値は、SQL や Microsoft Excel と同様に、サンプル(n-1)の式を使って計算される。 |
| StDevP, VarP | 現在のグループ内の、式の標準偏差および分散。これらの値は、SQL や Microsoft Excel と同様に、母集団(n)の式を使って計算される。 |

集計関数を使用するには、ヘッダセクションまたはフッタセクションに計算フィールドを追加し、フィールドのText プロパティに式を割り当てます。

たとえば、**NWind.xml** サンプルファイルの "Employee Sales by Country" レポートには、複数の集計フィールドが含まれています。このレポートは、レコードを国(Country)別や従業員(Employee)別にグループ化しています。

Employee グループのフッタセクションにある **SalespersonTotal** フィールドには、次の式が含まれています。

```
=Sum([SaleAmount])
```

このフィールドは Employee グループのフッタにあるため、この式は従業員別の総売上高を返します。

**CountryTotal** フィールドと **GrandTotal** フィールドにも、まったく同じ式が含まれています。しかし、これらのフィールドはそれ


Copyright © GrapeCity, Inc. All rights reserved.

ぞれ Country グループのフッタおよびレポートフッタにあるため、これらの式は国別の総売上高およびレコードセット全体の売上高を返します。

グループ内から上位レベルの集計値を参照する場合があります。たとえば、"Employee Sales by Country" レポートには、国別の売上高を総売上高に対する割合(%)で表示するフィールドがあります。Country グループ内で計算されるすべての集計値は、現在の国を参照するため、この割合値を直接計算することはできません。その代わり、**PercentOfGrandTotal** フィールドでは次の式を使用します。

```
=[CountryTotal]/[GrandTotal]
```

**CountryTotal** と **GrandTotal** は、それぞれ、Country フッタセクションとレポートフッタセクションにあるフィールドです。したがって、**CountryTotal** は国別の合計値を、**GrandTotal** はレコードセット全体の合計値を保持します。

コントロールがレコードセット全体に渡ってデータを調べる必要があるため、集計関数の評価には時間がかかります。したがって、集計関数を使用する計算フィールドの数は少なくしてください。集計関数を使用しないフィールドでは、集計式を再度評価するのではなく、集計関数を使用するフィールドから集計値を直接読み取ります。

たとえば、NorthWind データベース内の "Employee Sales by Country" レポートには、各売上高を国別の総売上高に対する割合(%)で表示する詳細フィールドがあります。このフィールドには、次の式が含まれています。

```
=[SaleAmount]/[CountryTotal]
```

**SaleAmount** は、レコードセットのフィールドへの参照で、各詳細レコードごとに値が変化します。**CountryTotal** は、集計関数を含むレポートフィールドへの参照です。コントロールは、この式を評価する際にレポートフィールドから直接集計値を取得します。集計の再計算は行いません。

For the complete report, see report "Employee Sales by Country" in the **Nwind.xml** report definition file, which is available in the **ComponentOne Samples** folder.

## イベントの使用

VBScript は、計算フィールドで式を評価する場合にだけ使用できるわけではありません。レポートの出力時に実行されるスクリプトを指定することもできます。このようなスクリプトを使用して、レポートの書式設定を変更できます。

これらのスクリプトはイベントプロパティに格納されます。イベントプロパティは Visual Basic のイベントハンドラに似ていますが、スクリプトはレポートを表示するアプリケーションの適用範囲ではなく、レポートの適用範囲内で実行される点が異なります。たとえば、イベントプロパティを使用すると、フィールドの**Font** プロパティや**ForeColor** プロパティをフィールドの値に基づいて設定できます。この動作は、レポート自身の一部になり、出力に使用されるアプリケーションに関係なく保持されます。

もちろん、従来のイベントも使用できます。従来のイベントは、レポートではなくアプリケーションに影響を及ぼす動作を実装するために使用します。たとえば、**StartPage** イベントのハンドラを記述して、出力されているレポートに関係なく、アプリケーションでページ数を更新できます。

次の表に、使用できるイベントプロパティと、その一般的な使用方法を示します。

次に、これらのプロパティの一般的な使用例を示します。

| オブジェクト | イベントプロパティ | 説明 |
|---|---|---|
| Report | OnOpen | レポートの出力が開始されると発生します。ConnectionString プロパティや RecordSource プロパティを変更したり、VBScript の変数を初期化するために使用できます。 |
| | OnClose | レポートの出力が終了すると発生します。クリーンアップ作業を実行するために使用できます。 |
| | OnNoData | レポートの出力を開始する際、ソースレコードセットが空の場合に発生します。**Cancel** プロパティを **True** に設定して、レポートの生成を中止できます。また、ダイアログボックスを表示して、レポートが表示されない理由をユーザーに知らせることもできます。 |
| | OnPage | 新しいページが開始されると発生します。さまざまなフィールドのセクションの**Visible** |


Copyright © GrapeCity, Inc. All rights reserved.

| オブジェクト | イベントプロパティ | 説明 |
|---|---|---|
| | | プロパティを状況に応じて設定するために使用できます。コントロールは、新しいページが開始されるたびに自動的に1ずつ増える Page 変数を保持しています。 |
| | OnError | エラーが発生すると発生します。 |
| Section | OnFormat | セクション内のフィールドが評価される前に発生します。この時点では、ソースレコードセット内のフィールドは、これから出力される値を反映しているが、レポートフィールドは値を反映していない。 |
| | OnPrint | セクション内のフィールドが印刷される前に発生します。この時点では、フィールドの評価は終了しており、条件付き書式設定を実行できます。 |

# 値に応じてフィールドを書式設定する

これは、おそらく最も一般的な **OnPrint** プロパティの使用方法です。Nwind.xml サンプルファイルの "Orders by Value" レポートを例にします。このレポートでは、製品別にグループ化された注文数がリストされます。別のフィールドを使って在庫数を表示する代わりに、製品の名前を赤色の太字で表示することによって、在庫が追加注文レベルより少ない製品を強調表示します。

## コードを使用して、追加注文レベルより少ない製品を強調表示するには

在庫が追加注文レベルより少ない製品を強調表示するためには、次のようなイベントスクリプトを使用します。

**Visual Basic コードの書き方**

Visual Basic

```
Dim script As String = _
  "If UnitsInStock < ReorderLevel Then" & vbCrLf & _
  "ProductNameCtl.ForeColor = RGB(255,0,0)" & vbCrLf & _
  "ProductNameCtl.Font.Bold = True" & vbCrLf & _
  "Else" & vbCrLf & _
  "ProductNameCtl.ForeColor = RGB(0,0,0)" & vbCrLf & _
  "ProductNameCtl.Font.Bold = False" & vbCrLf & _
  "End If"
c1r.Sections.Detail.OnPrint = script
```

**C# コードの書き方**

C#

```
string  script =
  "if (UnitsInStock < ReorderLevel) then\r\n" +
  "ProductNameCtl.ForeColor = rgb(255,0,0)\r\n" +
  "ProductNameCtl.Font.Bold = true\r\n" +
  "else\r\n" +
  "ProductNameCtl.ForeColor = rgb(0,0,0)\r\n" +
  "ProductNameCtl.Font.Bold = false\r\n" +
  "end if\r\n";
c1r.Sections.Detail.OnPrint = script;
```

このコードは、VBScript のイベントハンドラを1つの文字列として作成し、その文字列をセクションの **OnPrint** プロパティに割り当てます。C1Report デザイナを使用すると、上のコードを記述する代わりに、イベントエディタウィンドウに直接スクリプトコードを入力できます。


Copyright © GrapeCity, Inc. All rights reserved.

C1Report デザイナを使用して、追加注文レベルより少ない製品を強調表示するには

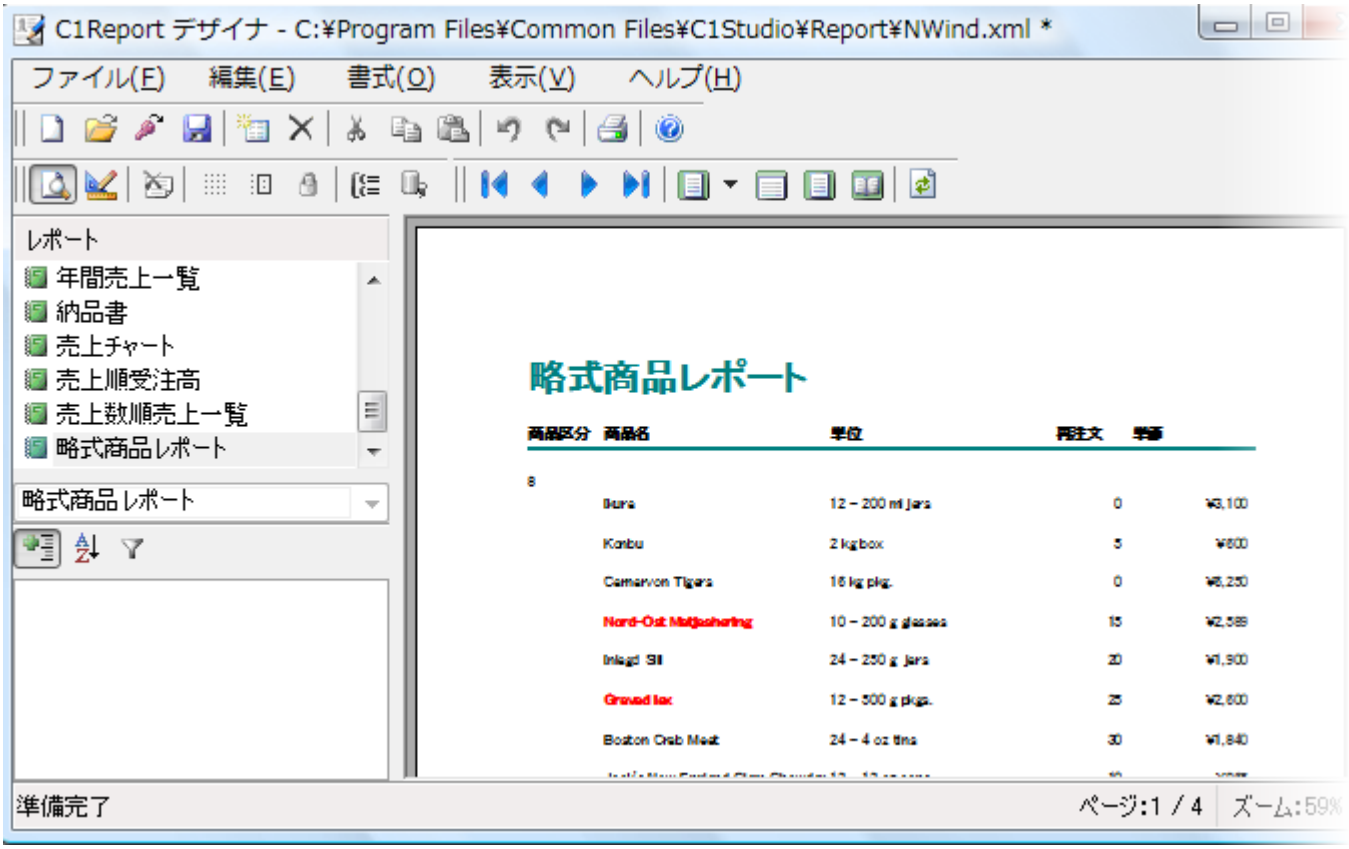
コードを記述する代わりに、**C1Report デザイナ**を使用して、詳細セクションの OnPrint プロパティの VBScript エディタに次のスクリプトコードを直接入力できます。次の手順を実行します。

1. デザイナの[プロパティ]ウィンドウのドロップダウンリストから、[**詳細**]を選択します。これで、セクションで使用できるプロパティが表示されます。
2. OnPrint プロパティの横にある空のボックスをクリックし、ドロップダウン矢印をクリックして、リストから[**スクリプトエディタ**]を選択します。
3. **VBScript エディタ**のウィンドウに、次のスクリプトを入力します。

```
If UnitsInStock < ReorderLevel Then ProductNameCtl.ForeColor = RGB(255,0,0)
ProductNameCtl.Font.Bold = True Else ProductNameCtl.ForeColor = RGB(0,0,0)
ProductNameCtl.Font.Bold = False End If
```

コントロールは、このセクションを出力するたびにこの VBScript コードを実行します。スクリプトは "ReorderLevel" データベースフィールドの値を取得し、その値に基づいて、"ProductName" レポートフィールドの **Font.Bold** プロパティと **ForeColor** プロパティを設定します。在庫が追加注文レベルより少ない製品は、製品名が赤色の太字になります。

次に、レポートの外見を示します。




## データが存在しないセクションを非表示にする

詳細セクションの **OnFormat** プロパティに式を指定することで、データに基づいてレポートフィールドの書式を変更できます。

たとえば、詳細セクションに画像コントロールを含むフィールドがある場合は、レコードに画像のデータが存在しないときに、そのレコードを非表示にできます。データ(この例では、レコードの画像)が存在しない場合に詳細セクションを非表示にするには、詳細セクションの **OnFormat** プロパティに次のスクリプトを追加します。:

```
If isnull(PictureFieldName) Then
  Detail.Visible = false
 Else
  Detail.Visible = true
```


Copyright © GrapeCity, Inc. All rights reserved.

```
End If
```

## コードを使用して、データが存在しないセクションを非表示にするには:

データが存在しないセクション（この例では、レコードの画像）を非表示にするには、次のようなイベントスクリプトを使用します。

**Visual Basic コードの書き方**

Visual Basic

```
C1Report1.Sections.Detail.OnFormat = "Detail.Visible = not
isnull(PictureFieldName)"
```

**C# コードの書き方**

C#

```
c1Report1.Sections.Detail.OnFormat = "Detail.Visible = not
isnull(PictureFieldName)";
```

## デザイナを使用して、データが存在しないセクションを非表示にするには

コードを記述する代わりに、**C1Report デザイナ**を使用して、詳細セクションの **OnFormat** プロパティの VBScript エディタに次のスクリプトコードを直接入力できます。次の手順を実行します。

1. デザイナの[プロパティ]ウィンドウのドロップダウンリストから、[**詳細**]を選択します。これで、セクションで使用できるプロパティが表示されます。
2. **OnFormat** プロパティの横にある空のボックスをクリックし、ドロップダウン矢印をクリックして、リストから[**スクリプトエディタ**]を選択します。
3. **エディタ**で、次の作業を実行します。
   - ウィンドウに次のスクリプトを入力します。
     ```
     If isnull(PictureFieldName) Then
     Detail.Visible = false
     Else
     Detail.Visible = true
     End If
     ```
   - または、簡潔なスクリプトを使用することもできます。`Detail.Visible = not isnull(PictureFieldName)`

# 値に応じてフィールドを表示または非表示にする

フィールドの書式を変更してフィールドのコンテンツを強調表示する代わりに、別のフィールドの **Visible** プロパティを **True** または **False** に設定することで、特殊効果を生成することができます。たとえば、「BoxCtl」という新しいフィールドを作成し、製品名が太線の四角形で囲まれるように書式設定するには、スクリプトを次のように変更します。

```
If UnitsInStock < ReorderLevel Then
BoxCtl.Visible = True
Else
BoxCtl.Visible = False
End If
```


Copyright © GrapeCity, Inc. All rights reserved.

## コードを使用して、追加注文レベルより少ない製品を強調表示するには

追加注文レベルより少ない製品にボックスを表示して強調表示するには、次のようなイベントスクリプトを使用します。

**Visual Basic コードの書き方**

Visual Basic

```
Dim script As String = _
  "If UnitsInStock < ReorderLevel Then" & vbCrLf & _

  "  BoxCtl.Visible = True" & vbCrLf & _
  "Else" & vbCrLf & _
  "  BoxCtl.Visible = False" & vbCrLf & _
  "End If"
c1r.Sections.Detail.OnPrint = script
```

**C# コードの書き方**

C#

```
string script =
  "if (UnitsInStock < ReorderLevel) then\r\n" +
  "BoxCtl.Visible = true\r\n" +
  "else\r\n" +
  "BoxCtl.Visible = false\r\n" +
  "end if\r\n";
c1r.Sections.Detail.OnPrint = script;
```

このコードは、VBScript イベントハンドラを含む文字列を作成し、それをセクションの OnPrint プロパティに割り当てます。

## C1Report デザイナを使用して、追加注文レベルより少ない製品を強調表示するには

コードを記述する代わりに、**C1Report デザイナ**を使用して、詳細セクションの OnPrint プロパティの VBScript エディタに次のスクリプトコードを直接入力できます。次の手順を実行します。

1. デザイナの[プロパティ]ウィンドウのドロップダウンリストから、[**詳細**]を選択します。これで、セクションで使用できるプロパティが表示されます。
2. OnPrint プロパティの横にある空のボックスをクリックし、ドロップダウン矢印をクリックして、リストから[**スクリプトエディタ**]を選択します。
3. **VBScript エディタ**のウィンドウに、次のスクリプトを入力します。

```
If UnitsInStock < ReorderLevel Then
BoxCtl.Visible = True
Else
BoxCtl.Visible = False
End If
```

コントロールは、セクションが印刷されようとするたびに VBScript コードを実行します。このスクリプトは「ReorderLevel」データベースフィールドの値を取得し、その値に基づいて「ProductName」レポートフィールドの Font.Bold と ForeColor プロパティを設定します。商品が追加注文レベル未満の場合、その名前は太い赤字になります。

次のスクリーンショットは、特殊効果が適用されたレポートのセクションです。


Copyright © GrapeCity, Inc. All rights reserved.

| ProductID | ProductName | QuantityPerUnit | ReorderLevel | UnitsInStock |
|---|---|---|---|---|
| 10 | Ikura | 12 - 200 ml jars | 0 | 31 |
| 13 | Konbu | 2 kg box | 5 | 24 |
| 18 | Carnarvon Tigers | 16 kg pkg. | 0 | 42 |
| 30 | Nord-Ost Matjeshering | 10 - 200 g glasses | 15 | 10 |
| 36 | Inlagd Sill | 24 - 250 g jars | 20 | 112 |
| 37 | Gravad lax | 12 - 500 g pkgs. | 25 | 11 |
| 40 | Boston Crab Meat | 24 - 4 oz tins | 30 | 123 |
| 41 | Jack's New England Clam Chowder | 12 - 12 oz cans | 10 | 85 |
| 45 | Røgede sild | 1k pkg. | 15 | 5 |
| 46 | Spegesild | 4 - 450 g glasses | 0 | 95 |

# ユーザーにパラメータの入力を求める

データベースに格納されている追加注文レベルより少ない製品を強調表示する代わりに、ユーザーに、使用する追加注文レベルの入力を求めることができます。

ユーザーから値を取得するには、レポートの**RecordSource** プロパティを変更して、パラメータクエリーを使用します。パラメータクエリーの構築方法については、次の項を参照してください。

**Visual Basic コードの書き方**

Visual Basic

```
c1r.DataSource.RecordSource = _
  "PARAMETERS [Critical Stock Level] Short 10;" & _
  c1r.DataSource.RecordSource
```

**C# コードの書き方**

C#

```
c1r.DataSource.RecordSource =
"PARAMETERS [Critical Stock Level] short 10;" +
c1r.DataSource.RecordSource;
```

このように設定すると、コントロールは、"Critical Stock Level" 値の入力をユーザーに求めます。この値は、VBScript のグローバル変数に格納されるため、イベントがその値を使用できるようになります。変数のデフォルト値は 10 に指定されています。

ユーザーによって指定された値を使用するには、スクリプトを次のように変更します。

**Visual Basic コードの書き方**

Visual Basic

```
Dim script As String = _
  "level = [Critical Stock Level]" & vbCrLf & _
```


Copyright © GrapeCity, Inc. All rights reserved.

```
  "If UnitsInStock < level Then" & vbCrLf & _
  "  ProductNameCtl.ForeColor = RGB(255,0,0)" & vbCrLf & _
  "  ProductNameCtl.Font.Bold = True" & vbCrLf & _
  "Else" & vbCrLf & _
  "  ProductNameCtl.ForeColor = RGB(0,0,0)" & vbCrLf & _
  "  ProductNameCtl.Font.Bold = False" & vbCrLf & _
  "End If"
c1r.Sections("Detail").OnPrint = script
```

**C# コードの書き方**

C#

```
string  script =
  "level = [Critical Stock Level]\r\n" +
  "if (UnitsInStock < level) then\r\n" +
  "ProductNameCtl.ForeColor = rgb(255,0,0)\r\n" +
  "ProductNameCtl.Font.Bold = true\r\n" +
  "else\r\n" +
  "ProductNameCtl.ForeColor = rgb(0,0,0)\r\n" +
  "ProductNameCtl.Font.Bold = false\r\n" +
  "end if\r\n";
c1r.Sections.Detail.OnPrint = script;
```

変更したのは、スクリプトの最初の2行です。上のスクリプトは、"UnitsInStock" フィールドの現在値を、データベースに格納されている追加注文レベルと比較するのではなく、ユーザーによって入力されて VBScript 変数の [Critical Stock Level] に格納された値と比較します。

## ページカウンタをリセットする

**Page** 変数は、コントロールによって作成されて自動的に更新されます。ページのヘッダーまたはフッタにページ数を追加する場合に便利です。また、新しいグループの先頭でページカウンタをリセットすることもできます。これらの処理はコードを追加したりデザイナを使用したりして実行する事ができます。

## コードを使用する場合：

新しいグループ（新しい国など）の先頭でページカウンタをリセットするには、PageFooter フィールドの **Text** プロパティを設定します。次のコードを入力します。

**Visual Basic コードの書き方**

Visual Basic

```
C1Report1.Fields("PageFooter").Text = "[ShipCountry] & "" "" & [Page]"
```

**C# コードの書き方**

C#

```
c1Report1.Fields("PageFooter").Text = "[ShipCountry] + "" "" + [Page]";
```

## C1Report デザイナを使用する場合：

新しいグループ（新しい国など）の先頭でページカウンタをリセットするには、次の手順を実行して、PageFooter フィールドの


Copyright © GrapeCity, Inc. All rights reserved.

**Text** プロパティを設定します。

1. デザイナの[プロパティ]ウィンドウのドロップダウンリストから、PageFooter の PageNumber フィールドを選択します。これで、フィールドで使用できるプロパティが表示されます。
2. **Text** プロパティの横にある空のボックスをクリックし、ドロップダウン矢印をクリックして、リストから[**スクリプトエディタ**]を選択します。
3. VBScript エディタのウィンドウに、次のスクリプトを入力します。`[ShipCountry] & " - Page " & [Page]`
4. **OK** をクリックしてエディタを終了します。

# フィールドのサイズを変更して棒グラフを作成する

これは最も高度な例です。フィールドの値をテキストとして表示するのではなく、フィールドのサイズを変更してグラフを作成します。

## グラフの作成

グラフを作成するには、最初にスケール(最大値および最小値として使用するサイズ)を調べる必要があります。「売上チャート」レポートには、この目的で設計されたフィールドがあります。これは、**SaleAmountMaxFld** という名前のレポートフッターフィールドで、グラフに表示する棒グラフの中で最も長い棒のサイズを保持し、次の式が含まれます。
`=Max([SaleAmount])`

**コードを使用する場合**

グラフの最大値を設定するには、**SaleAmountMaxFld.Text** プロパティを設定します。次のコードを入力します。

**Visual Basic コードの書き方**

Visual Basic
```
SaleAmountMaxFld.Text = "Max([SaleAmount])"
```

**C# コードの書き方**

C#
```
SaleAmountMaxFld.Text = "Max([SaleAmount])";
```

**C1Report デザイナを使用する場合**

グラフの最大値を設定するには、次の手順を実行して、**SaleAmountMaxFld.Text** プロパティを設定します。

1. デザイナの[プロパティ]ウィンドウのドロップダウンリストから、**SaleAmountMaxFld** を選択します。これで、フィールドで使用できるプロパティが表示されます。
2. **Text** プロパティの横にある空のボックスをクリックし、ドロップダウン矢印をクリックして、リストから[**スクリプトエディタ**]を選択します。
3. **VBScript** エディタのウィンドウに、次のスクリプトを入力します。`=Max([SaleAmount])`

## グラフの棒の描画

実際の棒を描画するために、レポートには **BarFld** という名前の詳細フィールドがあります。これは、塗りつぶした四角形になるように書式設定されています。詳細セクションでは、次のスクリプトが **OnPrint** プロパティに割り当てられています。

この式は、参照フィールドの幅と値、および現在のレコードの **SaleAmountFld** の値に基づいて、棒グラフの幅を計算します。

**コードを使用する場合**

グラフの実際の棒を描画するには、OnPrint プロパティを設定します。次のコードを入力します。

**Visual Basic コードの書き方**


Copyright © GrapeCity, Inc. All rights reserved.

Visual Basic

```
c1r.Sections.Detail.OnPrint = & _
  "BarFld.Width = SaleAmountMaxFld.Width * " & _
  "(SaleAmountFld / SaleAmountMaxFld)"
```

**C# コードの書き方**

C#

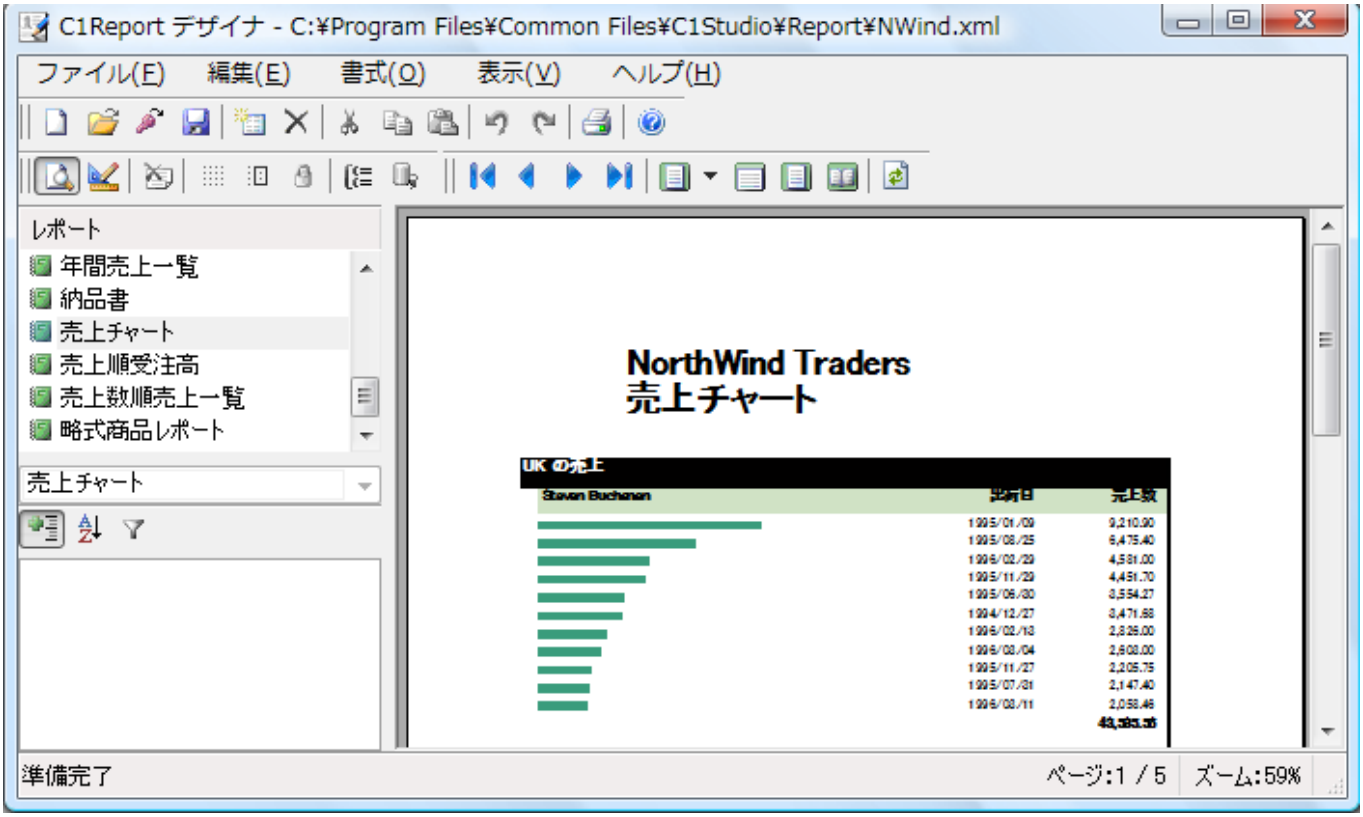
```
c1r.Sections.Detail.OnPrint = +
  "BarFld.Width = SaleAmountMaxFld.Width * " +
  "(SaleAmountFld / SaleAmountMaxFld)";
```

**C1Report デザイナを使用する場合**

グラフの実際の棒を描画するには、次の手順を実行して、OnPrint プロパティを設定します。

1. デザイナの[プロパティ]ウィンドウのドロップダウンリストから、[**詳細**]を選択します。これで、詳細セクションで使用できるプロパティが表示されます。
2. OnPrint プロパティの横にある空のボックスをクリックし、ドロップダウン矢印をクリックして、リストから[**スクリプトエディタ**]を選択します。
3. **VBScript エディタ**のウィンドウに、次のスクリプトを入力します。`:BarFld.Width = SaleAmountMaxFld.Width * (SaleAmountFld / SaleAmountMaxFld)`

次のスクリーンショットは、特殊効果が適用された「売上チャート」レポートのセクションです。




**メモ**:レポートの完全なサンプルについては、**Nwind.xml** レポート定義ファイルの「売上チャート」レポートを参照してください。

# 便利な機能の活用



Copyright © GrapeCity, Inc. All rights reserved.

このセクションでは、パラメータクエリーの追加、非バインドレポートの作成、カスタムデータソースの使用、およびレポートへのデータセキュリティの追加を行う方法を説明します。

# パラメータクエリー

パラメータクエリーは、独自のダイアログボックスを表示して、レコードを取得するための条件やレポートフィールドの値といった情報の入力をユーザーに求めることができるクエリーです。ユーザーに複数の情報の入力を求めるクエリーも設計できます。たとえば、2つの日付を取得するクエリーを設計して、その間の日付に該当するすべてのレコードを取得することができます。

たとえば、パラメータクエリーに基づいて、月次収益のレポートを作成するとします。レポートを出力すると、C1Report はレポートに含める月の入力を求めるダイアログボックスを表示します。ユーザーが月を入力すると、C1Report は該当するレポートを出力します。

パラメータクエリーを作成するには、**RecordSource** プロパティの SQL 文を編集して、PARAMETERS 句を追加します。パラメータクエリーの作成に使用する構文は、Microsoft Access で使用される構文と同じです。

1. パラメータクエリーを作成する最も簡単な方法は、WHERE 句に1つ以上のアイテムがある簡単な SQL 文を元に、WHERE 句にある固定値を手作業でパラメータに置き換えることです。たとえば、次のような一般的な SQL 文を元にします。

```
strSQL = "SELECT DISTINCTROW * FROM Employees " & _
"INNER JOIN (Orders INNER JOIN [Order Subtotals] " & _
"ON Orders.OrderID = [Order Subtotals].OrderID) " & _
"ON Employees.EmployeeID = Orders.EmployeeID " & _
"WHERE (((Orders.ShippedDate) " & _
"Between #1/1/1994# And #1/1/2001#));"
```

2. 次に、この SQL 文のうち、パラメータに変える部分を確認します。この例では、WHERE 句にある日付（太字の部分）をパラメータにします。これらのパラメータの名前を Beginning Date（開始日）および Ending Date（終了日）にします。これらの名前にはスペースが含まれているため、[ ]で囲む必要があります。

```
strSQL = "SELECT DISTINCTROW * FROM Employees " & _
"INNER JOIN (Orders INNER JOIN [Order Subtotals] " & _
"ON Orders.OrderID = [Order Subtotals].OrderID) " & _
"ON Employees.EmployeeID = Orders.EmployeeID " & _
"WHERE (((Orders.ShippedDate) " & _
"Between [Beginning Date] And [Ending Date]));"
```

3. 最後に、パラメータ名、型、およびデフォルト値を含む PARAMETERS 句を使用して、SQL 文の先頭でパラメータを指定します。

```
strSQL = "PARAMETERS [Beginning Date] DateTime 1/1/1994, " & _
"[Ending Date] DateTime 1/1/2001;" & _
"SELECT DISTINCTROW * FROM Employees " & _
"INNER JOIN (Orders INNER JOIN [Order Subtotals] " & _
"ON Orders.OrderID = [Order Subtotals].OrderID) " & _
"ON Employees.EmployeeID = Orders.EmployeeID " & _
"WHERE (((Orders.ShippedDate) " & _
"Between [Beginning Date] And [Ending Date]));"
```

この文が実行されると、コントロールはユーザーにダイアログボックスを表示して、*Beginning Date* と *Ending Date* の値の入力を求めます。ユーザーが入力した値は SQL 文に組み込まれ、ユーザーが選択した期間を対象にしてレポートが生成されま


Copyright © GrapeCity, Inc. All rights reserved.

す。

ダイアログボックスは、**C1Report** によって臨機応変に作成されます。ダイアログボックスにはクエリー内のすべてのパラメータが含まれ、パラメータの型に適したコントロールが使用されます。たとえば、Boolean 型のパラメータにはチェックボックスが使用され、Date 型のパラメータには日時の選択コントロールが使用されます。上の SQL 文を実行して表示されるダイアログボックスを次に示します。

PARAMETERS 句は、各アイテムをカンマで区切ったリストで構成され、最後にセミコロン(;)を付けます。各アイテムは1つのパラメータを表し、次の情報を指定します。

- **パラメータ名:**名前にスペースが含まれる場合は、名前を[ ]で囲む必要がある(たとえば、[Beginning Date])。パラメータ名は、ユーザーから情報を収集するためのダイアログボックスに表示される。また、SQL 文の WHERE 句でも使用され、ユーザーが入力した値と置き換えられる。
- **パラメータの型:** 次の型がコントロールに認識される。

| 値の型 | 説明 |
|---|---|
| Date | adDate |
| DateTime | adDate |
| Bit, Byte, Short, Long | adInteger |
| Currency | adCurrency |
| Single | adSingle |
| Double | adDouble |
| Text, String | adBSTR |
| Boolean, Bool, YesNo | adBoolean |

- **デフォルト値:** ダイアログボックスに最初に表示される値。

パラメータクエリーは、次の手順で簡単に作成できます。最初に、正しく動作する簡単なクエリーを作成します。次に、PARAMETERS 句を追加します。PARAMETERS 句の最後にセミコロン(;)を付けることを忘れないでください。最後に、WHERE 句を編集して、適切な位置にパラメータ名を追加します。

**GetRecordSource** メソッドを使用すると、パラメータクエリーから(PARAMETERS 句がない)適切な SQL 文を取得できます。これは、レポートに含まれるデータを使って独自のレコードセットを作成する場合に便利です。

> **メモ:**パラメータクエリーを使用する代わりに、ダイアログボックスを作成してユーザーから情報を取得し、必要に応じて SQL 文を修正したり、**DataSource** オブジェクトの**Filter** プロパティを設定するコードを、Visual Basic .NET または C# で記述することもできます。パラメータクエリーを使用する利点は、パラメータのロジックをレポート自身に置くことができること、またパラメータクエリーがビューアアプリケーションから独立していることです。コードを記述する手間も省くことができます。

# アンバウンドレポート

アンバウンドレポートとは、基礎になるソースレコードセットがないレポートです。このようなレポートは、次の2つの状況で便利です。:


Copyright © GrapeCity, Inc. All rights reserved.

- 書式が固定されており、出力するたびにいくつかのフィールドのコンテンツだけが変化するドキュメントを作成する場合。帳票がこの典型的な例で、書式が決まっており、フィールドの値だけが変化します。
- 複数のサマリーレポートを1つにまとめる場合。この場合は、メインのアンバウンドレポートを作成し、それにバウンドサブレポートを追加します.

簡単なアンバウンドレポートの例として、ソースレコードセットがない簡単なニュースレターを作成します。これには、**C1Report デザイナ**を使用しますが、**ConnectionString** プロパティと**RecordSource** プロパティは空のままにし、レポートにいくつかのプレースホルダフィールドを追加します。これらのプレースホルダフィールドは単純なラベルで、フィールドのコンテンツはアプリケーションによって設定されます。

"FldHeadLineXXX" および "FldBodyXXX"（XXXは1から3までの値）という名前の6つのプレースホルダフィールドがあるレポートを作成する場合は、次の Visual Basic コードを使ってレポートを出力します。

**Visual Basic コードの書き方**

Visual Basic

```
Private Sub MakeReport()

    ' レポート定義ファイルを検索します。
  Dim path As String = Application.StartupPath
  Dim i As Integer = path.IndexOf("\bin")
  If i > -1 Then path = path.Substring(0, i)
  path = path & "\"

   ' アンバウンドレポートを読み込みます。
  c1r.Load(path & "Newsletter.xml", "NewsLetter")

  ' フィールド値を設定します。
  c1r.Fields("FldHeadline1").Text = "C1Report Launched"
  c1r.Fields("FldBody1").Text = "ComponentOne unveils..."
  c1r.Fields("FldHeadline2").Text = "Competitive Upgrades"
  c1r.Fields("FldBody2").Text = "Get ahead ..."
  c1r.Fields("FldHeadline3").Text = "C1Report Designer"
  c1r.Fields("FldBody3").Text = "The C1Report Designer..."

   ' 完了したら、レポートを表示します。
  c1ppv.Document = c1r.Document
  ' 読者が閲覧できるよう HTML ファイルにエクスポートします。
  ' can get to it over the Web
  c1r.RenderToFile(path & "Newsletter.htm", FileFormatEnum.HTML)
End Sub
```

**C# コードの書き方**

C#

```
private void MakeReport()
{

  //  レポート定義ファイルを検索します。
   string path = Application.StartupPath;
   int i = path.IndexOf("\bin");
  if ( i > -1 ) { path = path.Substring(0, i)
  path = path + "\";
```


Copyright © GrapeCity, Inc. All rights reserved.

```
  //  アンバウンドレポートを読み込みます。
  c1r.Load(path + "Newsletter.xml", "NewsLetter");

  // フィールド値を設定します。
  c1r.Fields["FldHeadline1"].Text = "C1Report Launched";
  c1r.Fields["FldBody1"].Text = "ComponentOne unveils...";
  c1r.Fields["FldHeadline2"].Text = "Competitive Upgrades";
  c1r.Fields["FldBody2"].Text = "get { ahead ...";
  c1r.Fields["FldHeadline3"].Text = "C1Report Designer";
  c1r.Fields["FldBody3"].Text = "The C1Report Designer...";

  // 完了したら、レポートを表示します。
  c1ppv.Document = c1r.Document;

  //  ' 読者が閲覧できるよう HTML ファイルにエクスポートします。
  // can get to it over the Web

  c1r.RenderToFile(path + "Newsletter.htm", FileFormatEnum.HTML);
}
```

このニュースレターの例を次に示します。上の簡単なプログラムは、書式設定を何も行っていないことに注意してください。これは、レポートのコンテンツを提供しているだけです。ロゴ付きの見出し、ページフッタ、フォント、テキストの配置などのすべての書式設定は、**C1Report デザイナ**で作成されたレポート定義が処理します。

書式設定がコンテンツから分離されていることも、アンバウンドレポートの主な利点の1つです。

## データセキュリティ


Copyright © GrapeCity, Inc. All rights reserved.

データのセキュリティは、多くの企業にとって重要な問題です。たとえば、社内用の電話番号レポートを作成して配布する場合は、従業員名と内線番号を表示する必要があります。しかし、何者かがレポート定義を変更したり、社員の給与情報含むレポートを作成しては困ります。また、何者かがレポート定義ファイルを見て、接続文字列をコピーし、データベースの閲覧を始める(つまり、ハッキング)可能性もあります。

これらは、**C1Report** をはじめとしたすべてのデータベースアプリケーションに対する当然の懸念です。この項では、データを保護するためにとることができるいくつかの手段について説明します。

# Windows NT 統合セキュリティ

Windows NT の長所の1つは、セキュリティの処理方法にあります。ユーザーがログオンすると、システムは、システム管理者によって認められたアクセス許可を自動的にそのユーザーに与えます。その後、それぞれのアプリケーションやサービスが Windows NT に問い合わせて、そのユーザーがアクセスできるリソースを確認します。広く利用されているデータベースプロバイダの多くは、このようなセキュリティをオプションとして提供しています。

このようなセキュリティでは、データの共有を許可する人に、そのデータを読み取るための適切なアクセス許可を与えるだけで済みます。この場合、レポート定義ファイル内の**ConnectionString** にパスワードを含める必要はありません。許可されたユーザーはデータを見ることができ、許可されていないユーザーはアクセスできません。

# パスワードを含む接続文字列

これは、たいへん簡単な代替手段です。レポートを出力する前(または、コントロールが接続の失敗エラーを報告したとき)に、ユーザーにパスワードの入力を求め、そのパスワードを接続文字列に組み込むことができます。次に例を示します。

**Visual Basic コードの書き方**

Visual Basic

```
' パスワード用のプレースホルダを含む接続文字列を作成します。
Dim strConn
strConn = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
          "Data Source=C:\SecureData\People.mdb;" & _
          "Password={{THEPASSWORD}};"


' ユーザーからパスワードを取得します。
Dim strPwd$
strPwd = InputBox("Please enter your password:")
If Len(strPwd) = 0 Then Exit Sub

 ' 新しい接続文字列を作成し、コントロールに割り当てます。
strConn = Replace(strConn, "{{THEPASSWORD}}", strPwd)
vsr.DataSource.ConnectionString = strConn
```

**C# コードの書き方**

C#

```
// パスワード用のプレースホルダを含む接続文字列を作成します。
string strConn = "Provider=Microsoft.Jet.OLEDB.4.0;" +
          "Data Source=C:\SecureData\People.mdb;" +
          "Password={{THEPASSWORD}};";

// ユーザーからパスワードを取得します。
```


Copyright © GrapeCity, Inc. All rights reserved.

```
string strPwd = InputBox("Please enter your password:");
if (strPwd.Length == 0) return;

//   新しい接続文字列を作成し、コントロールに割り当てます。
strConn = Replace(strConn, "{{THEPASSWORD}}", strPwd);
c1r.DataSource.ConnectionString = strConn;
```

## アプリケーション定義のエイリアス

他に考えられる例として、一定のユーザーにレポートの表示は許可するものの、データが格納されている場所についての情報や特別な権限は与えないようにする場合があります。

**C1Report** では、2つの簡単な方法でこのようなセキュリティを実現できます。1つは、組み込みレポートを使用する方法です。[**レポートの選択**]プロパティページを使って、レポート定義を設計時にアプリケーションに読み込むと、レポートはアプリケーションに組み込まれます。こうすると、レポート定義ファイルを配布する必要がなく、誰もデータソース情報にアクセスできなくなります。

2番目の方法は、アプリケーションで一連の接続文字列エイリアスを定義する方法です。レポート定義ファイルにはエイリアスが含まれ、アプリケーションはレポートを出力する前にエイリアスを実際の接続文字列に置き換えます。これらのエイリアスは、**C1Report デザイナ**などの他のアプリケーションでは使用できません。さらにセキュリティを重視する場合は、**RecordSource** プロパティをチェックして、データベース内のテーブルやフィールドに不正なアクセスを試みる者がいないことを確認します。

次のコードは、簡単なエイリアススキームの実装例です。

**Visual Basic コードの書き方**

Visual Basic

```
Private Sub RenderReport(strReportName As String)

  ' ユーザーが要求するレポートを読み込みます。
  c1r.Load("c:\Reports\MyReports.xml", strReportName)

  ' 接続文字列エイリアスを置き換えま
  Dim strConn$
  Select Case c1r.DataSource.ConnectionString
    Case "CUSTOMERS"
    Case "EMPLOYEES"
       strConn = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
                 "Data Source=C:\SecureData\People.mdb;" & _
                 "Password=slekrsldksd;"
    Case "$$SALES"
      strConn = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
                 "Data Source=C:\SecureData\Numbers.mdb;" & _
                 "Password=slkkdmssids;"
  End Select

  ' 接続文字列を設定し、レポートを出力します。
  c1r.DataSource.ConnectionString = strConn
  ppv1.Document = c1r.Document

End Sub
```

**C# コードの書き方**


Copyright © GrapeCity, Inc. All rights reserved.

C#

```
private void RenderReport(string  strReportName) {

  // ユーザーが要求するレポートを読み込みます。
  c1r.Load("c:\Reports\MyReports.xml", strReportName);

  //  接続文字列エイリアスを置き換えま
  string strConn$;
  switch (i) { c1r.DataSource.ConnectionString;

    case "$$CUSTOMERS";
    case "EMPLOYEES";
      strConn = "Provider=Microsoft.Jet.OLEDB.4.0;" + _
                "Data Source=CSALES";
      strConn = "Provider=Microsoft.Jet.OLEDB.4.0;" +
               "Data Source=C:\SecureData\Numbers.mdb;" +
                "Password=slkkdmssids;";
  }

  // 接続文字列を設定し、レポートを出力します。
  c1r.DataSource.ConnectionString = strConn;
  ppv1.Document = c1r.Document;
}
```

## カスタムデータセット

アプリケーションによって作成された任意のデータセットを**Recordset** オブジェクトに割り当てることもできます。こうすると、適切なレベルのセキュリティ手段を任意に適用できるため、**ConnectionString**プロパティや**RecordSource** プロパティについて考慮する必要がなくなります。

## SSRS レポートの ComponentOne レポート

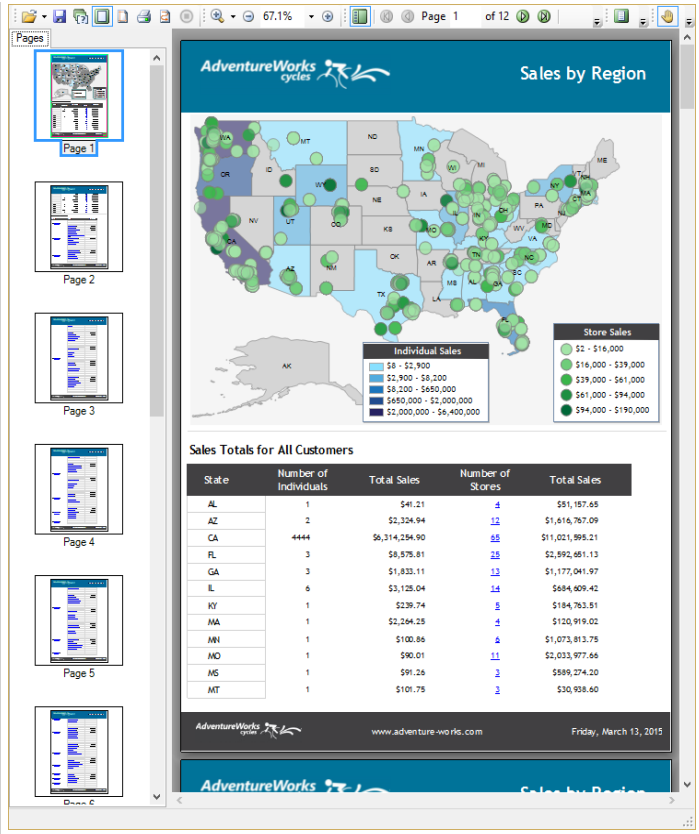
ComponentOne Reporting 製品は、SQL Server Reporting Services（SSRS）を直接サポートしていませんが、SSRS は RDL と RDL で定義されたレポートをサポートしています。ComponentOne Reporting 製品を MS SQL レポートサーバーに直接接続し、それによって生成されたレポートを表示することはできませんが、RDL 形式の SSRS レポートにアクセスすることは可能です。ファイルのロード方法については、「RDL ファイルのロード」を参照してください。

**C1PrintPreviewControl** を使用すると、**C1SSRSDocumentSource** の機能を利用して SSRS レポートをロードすることができます。C1PrintPreviewControl で SSRS レポートをロードする方法については、「**SSRS レポートのプレビュー**」を参照してください。

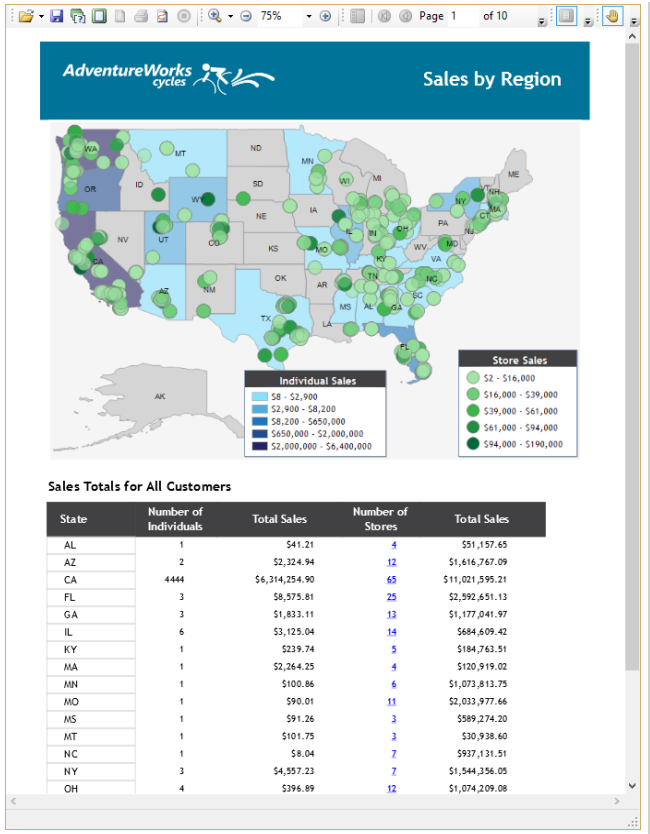
レポートは、ページ区切り付きモードとページ区切りなしモードの両方でプレビューできます。**C1PrintPreviewControl** コントロールの **SsrsPaginated** プロパティで、SSRS レポートのモードを変更できます。デフォルトでは false に設定されており、ページ区切りなしがデフォルトモードになります。

| ページ区切り付きモード | ページ区切りなしモード |
|---|---|


Copyright © GrapeCity, Inc. All rights reserved.

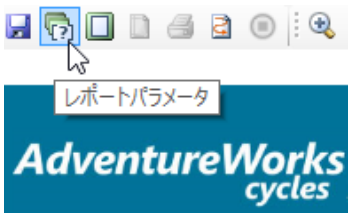



このモードは、ページブレークを追加することで、レポートを固定サイズの複数ページに編成します。




このモードは、すべてのページブレークを削除し、コンテンツ全体を1つのフローで表示します。

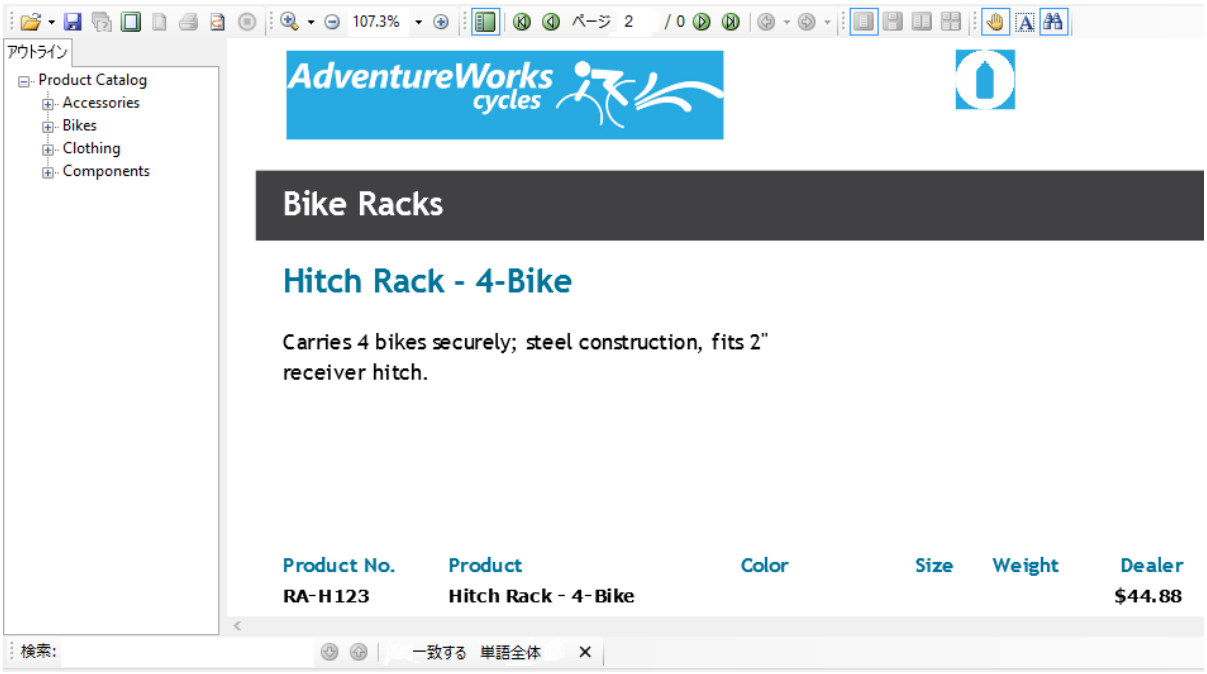
どちらのモードでも、パラメータ付きレポート内のパラメータを変更し、レポートを再ロードできます。**[レポートパラメータ]**ボタンをクリックして[レポートパラメータ]ダイアログを開き、パラメータを再選択します。




**ページ区切りなしモードの機能**

- データは、ページ区切り付きモードより迅速かつ効率的にロードされます。レポート全体をロードする必要はなく、現在のページ区切りなしページだけがロードされます。
- ナビゲーションペインのアウトラインパネルに、レポートのアウトライン（目次に相当）が表示されます。これにより、レポート内の特定の場所に迅速に移動できます。
- C1PrintPreviewControl の下部の検索ツールバーを有効にします。**TextSearchPanelVisible** プロパティを **True** に設定し、**TextSearchUIStyle** プロパティを **Toolbar** に設定します。


Copyright © GrapeCity, Inc. All rights reserved.




- レポートコンテンツをスクロールまたはズームします。
- ページ区切りなしモードで、レポートの各部分を展開または折りたたんだり、項目をソートします。これらのアクションを実行した後にページ区切り付きモードに切り替えると、展開およびソートされたセクションを含む更新後のレポートを印刷できます。
- レポート内のポイントをドリルダウンしたり、ポイントに移動します。または、別のレポートにドリルスルーします。**[前のビュー]**ボタンと**[次のビュー]**ボタン を使用して、これらの場所の間を移動します。
- レポート要素のツールチップを表示します。
- テキストを選択してクリップボードにコピーします。

**ページ区切りなしモードの制限**

ページ区切りなしモードでは、レポートを印刷できません。レポートを印刷するには、ページ区切り付きモードに切り替える必要があります。

**ページ区切り付きモードの機能**

C1PrintPreviewControl は、レポート内の特定のページに迅速に移動するため、ナビゲーションペインにページサムネイルを表示します。

**ページ区切り付きモードの制限**

- ページ区切り付きモードでは、テキストをクリップボードにコピーできません。
- ページ区切り付きモードでは、テキスト検索ツールバーが機能しません。


Copyright © GrapeCity, Inc. All rights reserved.

# C1RdlReport の使い方

**C1RdlReport**レポートは、2008 年版の RDL（Report Definition Language：レポート定義言語）仕様を用いて定義された RDL レポートを表すコンポーネントです。C1RdlReport コンポーネントは、RDL サポートを追加した C1Report コンポーネントに似ています。C1RdlReport コンポーネントは、C1.C1Report.2.dll アセンブリに含まれています。

**C1PrintDocument**（**ImportRdl** メソッドと **FromRdl** メソッドによって提供）の RDL インポートは廃止されました。代わりに C1RdlReport を使用してください。

## .NET 2.0/3.0 バージョンユーザー向けの重要な注意事項：

C1.C1Report.2 アセンブリでは、.NET Framework 3.5 以降にのみ含まれる、**System.Windows.Forms.DataVisualization.dll** アセンブリを使用します。このアセンブリは、**Reports for WinForms** をインストールするときにシステムにインストールされます。**Reports for WinForms** を使用するアプリケーションを他のシステムに展開する場合は、他の ComponentOne Reports アセンブリと共に配布物に含める必要があります。

また、**Reports for WinForms** の DLL を手動で更新する場合、これらのアセンブリのある場所に **DataVisualization** を配置し、プロジェクトがそれを参照するようにする必要があります。.NET 4.0 の場合は、**DataVisualization** がすでに Framework に含まれているため、これは適用されません。

# レポート定義言語（RDL）

レポート定義言語（RDL）は、レポートを表す Microsoft 標準の XML スキーマです。RDL の目標は、共通のスキーマを定義することでレポート定義の交換を可能にし、レポート作成製品の相互運用性の推進することです。RDL は、中立的な出力フォーマットとなるように設計されています。つまり、RDL を使用して定義されたレポートは、Web 、印刷対応フォーマット、XML のようなデータ用フォーマットを含むさまざまな形式に出力できるようにする必要があるということです。

2010 v3 ルーラーで **C1RdlReport** コンポーネントが **Reports for WinForms** に追加されたのも、RDL ファイルの柔軟性を活用するためです。C1RdlReport を使用することで、簡単に RDL ファイルをレポート作成アプリケーションにインポートできます。使い慣れた形式を使用することで、柔軟性と機能性が加わります。

# C1RdlReport のメリット

**C1RdlReport** コントロールには、標準の Microsoft Reporting Services を使用する場合に比べて、いくつかのメリットがあります。C1RdlReport の主なメリットは次のとおりです。

- 現在の RDL 2008 仕様のサポート。
- RDL オブジェクトモデルへのプログラムによるアクセス（2008 RDL 仕様に準拠）- 完全にコードだけで既存の RDL レポートを変更したり、新規 RDL レポートを作成したりできます。
- 任意のデータソース（.mdb ファイルなど）を消費する RDL レポートの生成。
- Microsoft Reporting Services へのアクセスの必要性などの外部依存のない自己完結 RDL レポート作成ソリューション。
- **C1PrintPreviewControl**その他の **Reports for WinForms** レポート生成エンジン（**C1Report** や **C1PrintDocument**）とのシームレスな統合。

# C1RdlReport の制限

**C1RdlReport** コントロールは、RDL のほとんどの機能をサポートしています。ただし、C1RdlReport の初回リリースには一部制限があります。

以下のオブジェクトはサポートされていません。

- Gauge オブジェクトはサポートされていません。


Copyright © GrapeCity, Inc. All rights reserved.

以下の RDL プロパティは、現在サポートされていません。

- Document.AutoRefresh
- Document.Width
- Document.Language
- Document.DataTransform
- Document.DataSchema
- Document.DataElementName
- Document.DataElementStyle
- DataSet.CaseSensitivity
- DataSet.Collation
- DataSet.AccentSensitivity
- DataSet.KanatypeSensitivity
- DataSet.WidthSensitivity
- DataSet.InterpretSubtotalsAsDetails
- TextBox.UserSort
- TextBox.ListStyle
- TextBox.ListLevel
- TextRun.ToolTip
- TextRun.MarkupType

**C1RdlReport** は、Report Builder 3.0（MS SQL 2008 R2）で作成された、2010 RDL 仕様を使用しているレポートをロードできるようになりました。ただし、レポート定義に含まれる **ReportSection** が1つであることが前提です。レポートに複数の **ReportSection** が含まれていると、例外がスローされます。

## .NET 2.0/3.0 バージョンユーザー向けの重要な注意事項:

C1.C1Report.2 アセンブリは、.NET Framework 3.5 以降にしか含まれていない **System.Windows.Forms.DataVisualization.dll** アセンブリを使用します。このアセンブリは、**Reports for WinForms** をインストールするときにシステムにインストールされます。**Reports for WinForms** を使用するアプリケーションを他のシステムに展開する場合は、他の ComponentOne Reports アセンブリと共に配布物に含める必要があります。

また、**Reports for WinForms** の DLL を手動で更新する場合、これらのアセンブリのある場所に **DataVisualization** を配置し、プロジェクトがそれを参照するようにする必要があります。.NET 4.0 の場合は、**DataVisualization** がすでに Framework に含まれているため、これは適用されません。

# RDL ファイルのロード

RDL ファイルを **C1RdlReport**コンポーネントにロードするには、**Load** メソッドを使用します。RDL ファイルを削除するには、**Clear** メソッドを使用します。このメソッドは、それまで C1RdlReport コントロールにロードされていた RDL ファイルをすべてクリアします。C1RdlReport コンポーネントには、RDL ファイルのロードとクリアのためのデザイン時オプションが含まれています。

## タスクメニューで RDL ファイルをロードするには、次の手順に従います。

次の手順を実行してください:

1. デザインビューで、C1RdlReport コンポーネントのスマートタグをクリックし、[**C1RdlReport タスク**]メニューを開きます。
2. [**C1RdlReport タスク**]メニューで、[**レポートのロード**]をクリックします。[**開く**]ダイアログが表示されます。
3. [**開く**]ダイアログボックスで、RDL ファイルの場所に移動してファイルを選択し、〈**開く**〉をクリックします。

レポートがロードされ、C1RdlReport コントロールが **C1PrintPreviewControl**, のようなプレビューコントロールに接続されていれば、デザイン時にプレビューコントロールにレポートがプレビュー表示されます。


Copyright © GrapeCity, Inc. All rights reserved.

# コードによる RDL ファイルのロード

RDL ファイルを C1RdlReport コンポーネントにロードするには、Load メソッドを使用します。以下の手順を実行します。

1. デザインビューでフォームをダブルクリックして、コードエディタを開きます。
2. 次のコードを **Load** イベントに追加します。C:/Report.rdl は、ロードする RDL ファイルの場所と名前で置き換えます。

**Visual Basic コードの書き方**

Visual Basic

```
C1RdlReport1.Load("C:/Report.rdl")
C1RdlReport1.Render()
```

**C# コードの書き方**

C#

```
c1RdlReport1.Load(@"C:/Report.rdl");
c1RdlReport1.Render();
```

レポートがロードされ、C1RdlReport コントロールが **C1PrintPreviewControl** のようなプレビューコントロールに接続されていれば、アプリケーションを実行したときに、レポートがプレビューコントロールに表示されます。


Copyright © GrapeCity, Inc. All rights reserved.

# C1Report デザイナの使い方

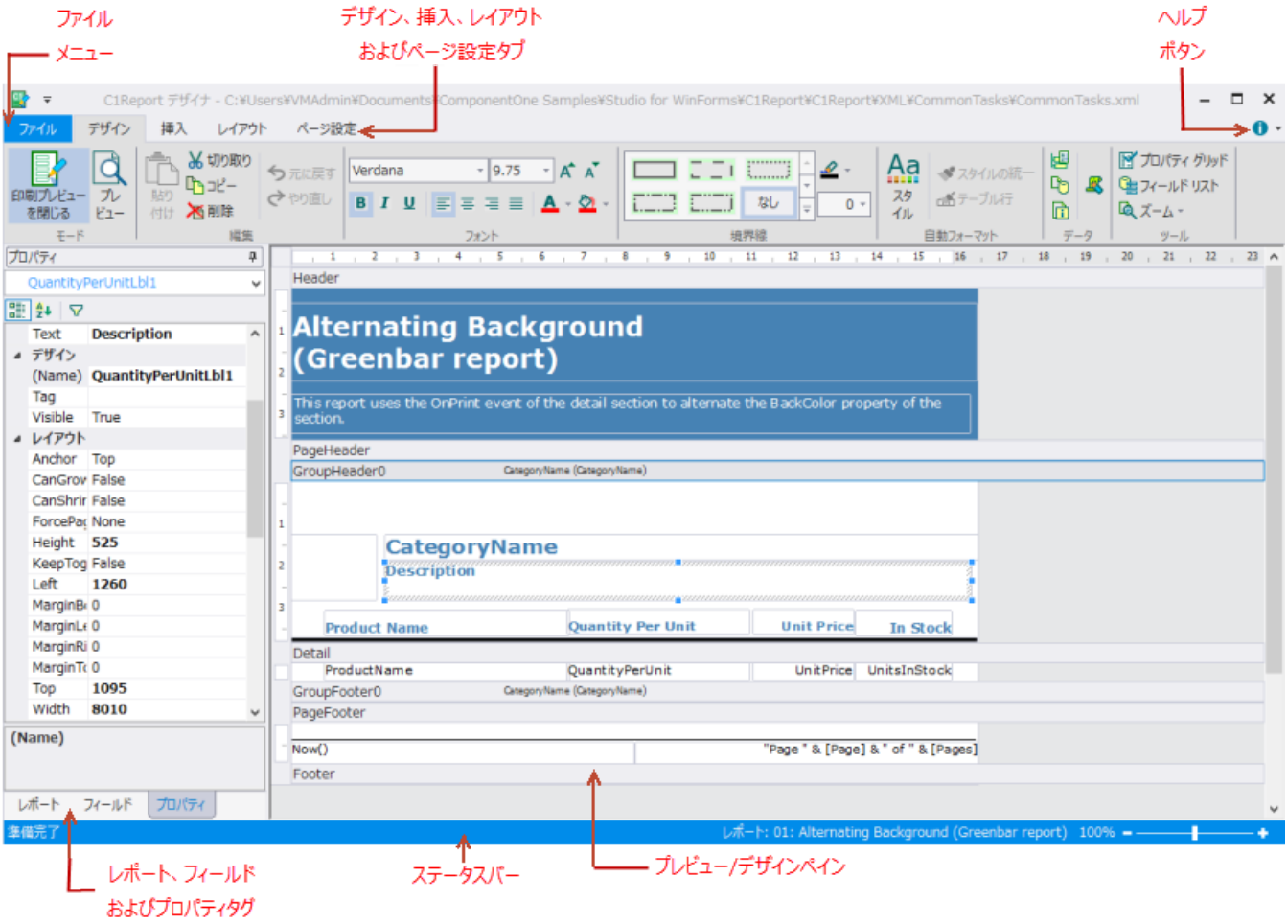
**C1ReportDesigner** は、Microsoft Access のレポートデザイナに似たスタンドアロンアプリケーションです。次のトピックには、**C1ReportDesigner** に関する重要な情報が含まれます。また、基本レポート定義ファイルの作成、レポート定義の変更、印刷、およびエクスポート方法についても説明します。このセクションでは、Microsoft Access で作成したレポートのインポート方法についても説明します。

# C1Report デザイナについて

**C1Report デザイナ**アプリケーションは、**C1Report** レポート定義ファイルの作成と編集に使用するツールです。デザイナでは、C1Report コンポーネントによって読み取ることができるファイル（XML）を作成し、編集し、ロードし、保存することができます。Microsoft Access ファイル（MDB）と VSReport 1.0（VSR）からレポート定義をインポートすることもできます。

デザイナを実行するには、スタートメニューから「**C1Report デザイナ**」をクリックします。このディレクトリは、デフォルトのインストールパスを反映したものであり、インストールパスを変更した場合、パスが異なることがあることに注意してください。Visual Studio からデザイナを実行する手順については、「Visual Studio からの C1Report デザイナへのアクセス」を参照してください。

CommonTasks.xml ファイルを開いたときに表示されるデザイナを以下に示します。




メインデザイナウィンドウには、以下のコンポーネントがあります。

- **〈ファイル〉メニュー**: **ファイル**メニューを開き、レポート定義ファイルをロードおよび保存し、レポート定義をインポートおよびエクスポートします。詳細については、「〈ファイル〉メニュー」を参照してください。
- **「デザイン」モード**: [編集]、[フォント]、[データ]、[フィールド]、および[ツール]メニュー機能へのショートカットを提供します。詳細については、「デザイン」モードを参照してください。


Copyright © GrapeCity, Inc. All rights reserved.

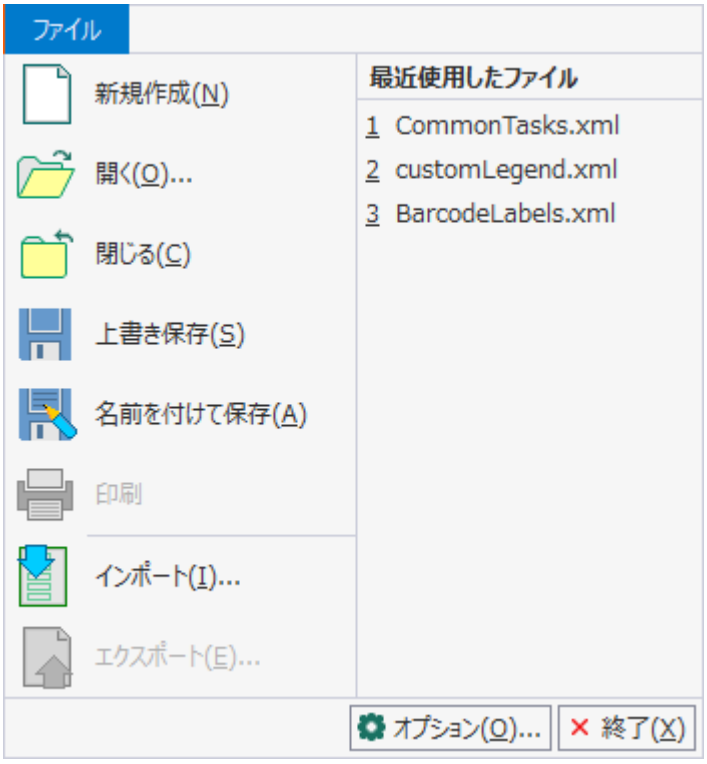
- **「プレビュー」モード**: レポートのプレビューを表示します。詳細については、「プレビュー」モードを参照してください。
- **〈ヘルプ〉ボタン**: C1Report デザイナヘルプと[**バージョン情報**]画面を表示するオプションを提供します。この画面は、アプリケーションに関する情報を表示します。
- **「レポート」タブ**: 現在のレポート定義ファイルに含まれるすべてのレポートをリスト表示します。レポート名をダブルクリックして、レポートをプレビューまたは編集できます。リストを使用して、レポートを名前変更し、コピーし、削除することもできます。
- **「フィールド」タブ**: 現在のレポートに含まれるすべてのフィールドをリスト表示します。
- **「プロパティ」タブ**: デザイナで選択したオブジェクトのプロパティを編集できます。
- **ステータスバー**: ステータスバーは、デザイナが現在実行している作業(ロード、保存、印刷、表示、インポートなど)に関する情報を表示します。ステータスバーの右側にあるズームスライダをドラッグして、選択したレポートをズームできます。

以下のトピックでは、**C1Report デザイナ**アプリケーションを使用して、レポート定義ファイルを作成、編集、使用、および保存する方法について説明します。

## [ファイル]メニュー

**[ファイル]メニュー**では、レポート定義ファイルをロードおよび保存し、レポート定義をインポートおよびエクスポートするためのショートカットメニューが提供されます。[ファイル]メニューを介して **C1Report デザイナ**アプリケーションのオプションにアクセスすることもできます。

**[ファイル]メニュー**をクリックして、次のようなメニューを開きます。




メニューには、以下のオプションが含まれています。

- [**新規作成**]:新しいレポート定義ファイルを作成します。
- [**開く**]:[**レポート定義ファイルを開く**]ダイアログボックスを開きます。これにより、既存のファイルを選択し、開くことができます。
- [**閉じる**]:現在のレポート定義ファイルを閉じます。
- [**上書き保存**]:以前に保存した場所に、レポート定義ファイルを保存します。
- [**名前を付けて保存**]:[レポート定義の保存]ダイアログボックスを開きます。これにより、レポート定義を XML ファイルとして保存できます。
- [**印刷**]:現在のレポートを印刷します。**印刷**ボタンは **C1ReportDesigner** が プレビューモードの場合のみ有効になります。

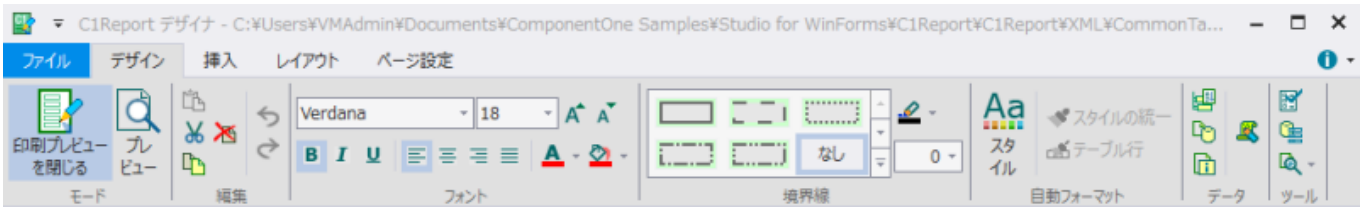

Copyright © GrapeCity, Inc. All rights reserved.

- [**インポート**]:[**レポート定義のインポート**]ダイアログボックスを開きます。これにより、Microsoft Access(.mdb および .adp)ファイルと Crystal Reports(.rpt)ファイルをインポートできます。詳細については、「Microsoft Access レポートのインポート」と「 Crystal Reports レポートのインポート」を参照してください。
- [**エクスポート**]:現在のレポートファイルを HTML、PDF、RTF、XLS、XLSX、TIF、TXT、または ZIP ファイルとしてエクスポートします。**エクスポート**ボタンは **C1ReportDesigner** が プレビューモードの場合のみ有効になります。
- [**最近使用したファイル**]:最近開いたレポート定義ファイルをリスト表示します。ファイルを再び開くには、それをリストから選択します。
- [**オプション**]:[**C1Report デザイナオプション**]ダイアログボックスを開きます。これにより、**C1Report デザイナ**アプリケーションのデフォルトの外観と動作をカスタマイズできます。詳細については、「C1Report デザイナオプションの設定「データ」グループ 」を参照してください。
- [**終了**]:**C1Report** デザイナアプリケーションを閉じます。

# デザインモード

**デザインモード**は現在選択されたレポートのセクションおよびフィールドを表示します。これは、レポートを作成や修正するレポートデザイナの基本操作領域となります。

デザインモードのリボンには以下のタブが含まれています。




- **「デザイン」タブ**:[矢印]、[フォント]、[データ]、[フィールド]、および[ツール]メニュー機能へのショートカットを提供します。詳細については、「「デザイン」タブ」を参照してください。
- **「挿入」タブ**:[選択]、[計算フィールド]、[画像]および[グラフ]メニュー機能へのショートカットを提供します。詳細については、「「挿入」タブ」を参照してください。
- **「レイアウト」タブ**:[グリッド]、[配置]、[位置]および[サイズ]メニュー機能へのショートカットを提供します。詳細については、「「レイアウト」タブ」を参照してください。
- **「ページ設定」タブ**:[編集]および[ページレイアウト]メニュー機能へのショートカットを提供します。詳細については、「「ページ設定」タブ」を参照してください。

# 「デザイン」タブ

**「デザイン」**タブは、以下のグループ内に位置された様々な機能で構成されます。

## 「編集」グループ

「デザイン」タブの「**編集**」グループは、次のように表示されます。

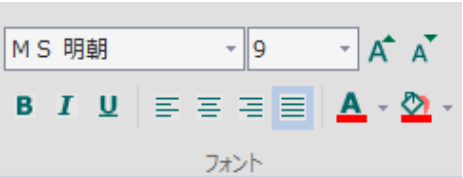
このグループは、以下のオプションで構成されます。

- **〈貼り付け〉**:直前にコピーした項目を貼り付けます。
- **〈切り取り〉**:選択した項目を切り取ります。レポートから選択した項目を削除し、他の場所に貼り付けることができます。
- **〈コピー〉**:選択した項目をコピーして、他の場所に貼り付けます。
- **〈削除〉**:選択した項目を削除します。
- **〈元に戻す〉**:レポート定義に加えた直前の変更を元に戻します。
- **〈やり直し〉**:レポート定義に加えた直前の変更をやり直します。


Copyright © GrapeCity, Inc. All rights reserved.

### 「フォント」グループ

**「デザイン」**タブの「**フォント**」グループは、次のように表示されます。




このグループは、以下のオプションで構成されます。

- **〈フォント名〉**：選択したテキストの現在のフォントを表示します。選択した項目に対して別のフォントを選択できます（これを行うには、フォント名の横にあるドロップダウン矢印をクリックします）。
- **〈フォントサイズ〉**：選択したテキストの現在のフォントサイズを表示すると共に、別のフォントサイズを選択できます。フォントサイズボックスに数値を入力するか、ドロップダウン矢印をクリックしてフォントサイズを選択できます。
- **〈フォントサイズの拡大〉**：フォントサイズを1ポイントずつ拡大します。
- **〈フォントサイズの縮小〉**：フォントサイズを1ポイントずつ縮小します。
- **〈太字〉**：選択したテキストを太字にします（[Ctrl]＋[B]キーを押すこともできます）。
- **〈斜体〉**：選択したテキストを斜体にします（[Ctrl]＋[I]キーを押すこともできます）。
- **〈下線〉**：選択したテキストに下線を引きます（[Ctrl]＋[U]キーを押すこともできます）。
- **〈左揃え〉**：テキストを左に揃えます。
- **〈中央揃え〉**：テキストを中央に揃えます。
- **〈右揃え〉**：テキストを右に揃えます。
- **〈両端揃え〉**：選択したテキストの両端を揃えます。
- **〈テキストの色〉**：選択したテキストの色を選択できます。
- **〈塗りつぶしの色〉**：選択したテキストの背景色を選択できます。

### 「境界線」グループ

**「デザイン」**タブの「**境界線**」グループは、次のように表示されます。

このグループは、以下のオプションで構成されます。

- **〈境界線のスタイル〉**：現在に選択したフィールドの境界線のスタイルを定義します。使用可能なスタイルでは実線、破線、点線、1点鎖線、2点鎖線および境界線なしが含まれています。
- **〈境界線の色〉**：現在に選択したフィールドの境界線の色を定義します。
- **〈境界線の幅〉**：現在に選択したフィールドの境界線の幅を定義します。

### 〈自動フォーマット〉グループ

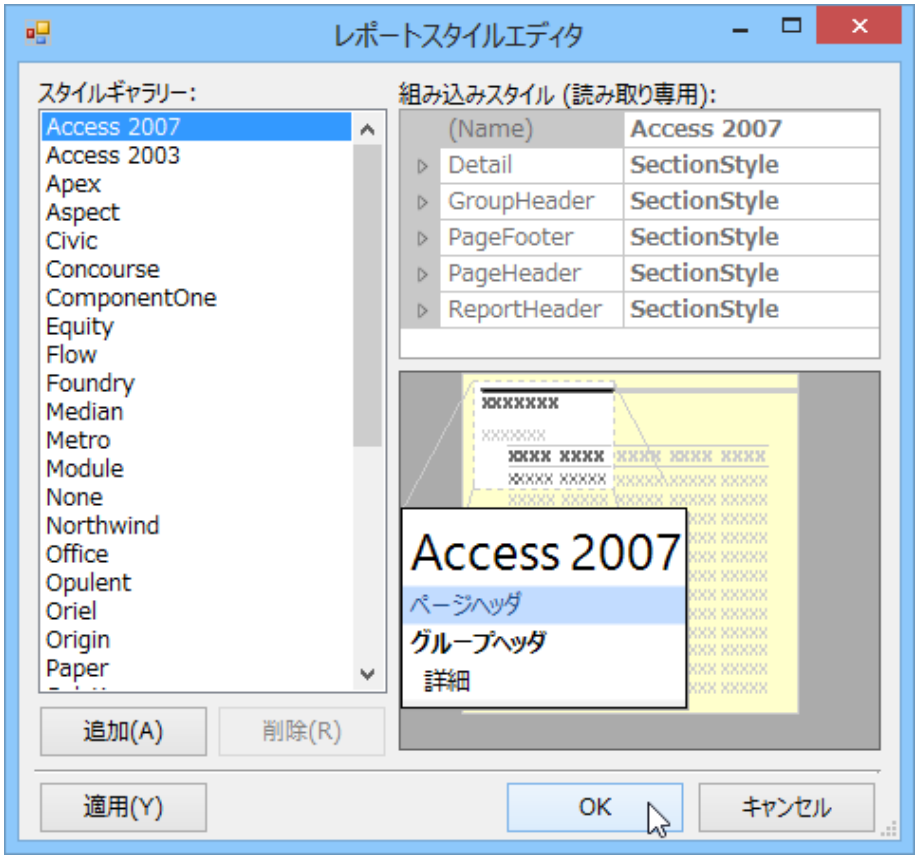
**「デザイン」**タブの「**自動フォーマット**」グループは、次のように表示されます。

このグループは、以下のオプションで構成されます。

- **〈スタイル〉**：**〈レポートスタイルエディタ〉**ダイアログボックスを開きます。このダイアログでは、組み込みスタイルを選択したり、独自のスタイルを編集できます
- **〈スタイルの統一〉**：現在選択されている複数の項目のスタイルを統一します。
- **〈テーブル行〉**：現在選択されている項目をテーブル行に合わせて書式設定します。


Copyright © GrapeCity, Inc. All rights reserved.

「**自動フォーマット**」グループの〈スタイル〉ボタンをクリックして、[レポートスタイルエディタ]ダイアログボックスにアクセスできます。[レポートスタイルエディタ]ダイアログボックスは、次のように表示されます。

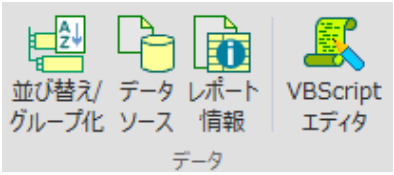



このダイアログボックスには、以下の要素が含まれています。

- **[スタイルギャラリー]**リスト:**[ スタイルギャラリー ]**リストには、現在利用可能なすべての組み込みおよびカスタムスタイルが表示されます。
- **〈追加〉ボタン**:〈追加〉ボタンを使用して、カスタムスタイルを[スタイルギャラリー]リストに追加します。追加されるスタイルは、〈追加〉ボタンをクリックしたときに[スタイルギャラリー]リストで選択したスタイルに基づいています。
- **〈削除〉ボタン**:〈削除〉ボタンを使用して、選択したカスタムスタイルを削除します。このボタンは、[スタイルギャラリー]リストでカスタム スタイルを選択したときにのみ使用可能になります。
- **プロパティグリッド**:プロパティグリッドでは、プロパティを変更し、カスタムスタイルを編集できます。プロパティグリッドは、[スタイルギャラリー]リストでカスタム スタイルを選択したときにのみ使用および編集可能になります。
- **プレビューウィンドウ**:プレビューウィンドウには、[スタイルギャラリー]リストで選択したスタイルのプレビューが表示されます。
- **〈適用〉ボタン**:〈適用〉ボタンをクリックして、ダイアログボックスを閉じることなく、選択した内容をスタイルに適用します。
- **〈OK〉ボタン**:〈OK〉ボタンをクリックして、ダイアログボックスを閉じ、変更内容を適用し、選択したスタイルを現在選択されているスタイルとして設定します。
- **〈キャンセル〉ボタン**:〈キャンセル〉ボタンをクリックして、スタイルに加えた変更内容をキャンセルできます。

### 〈データ〉グループ

「**データ**」グループは、次のように表示されます。

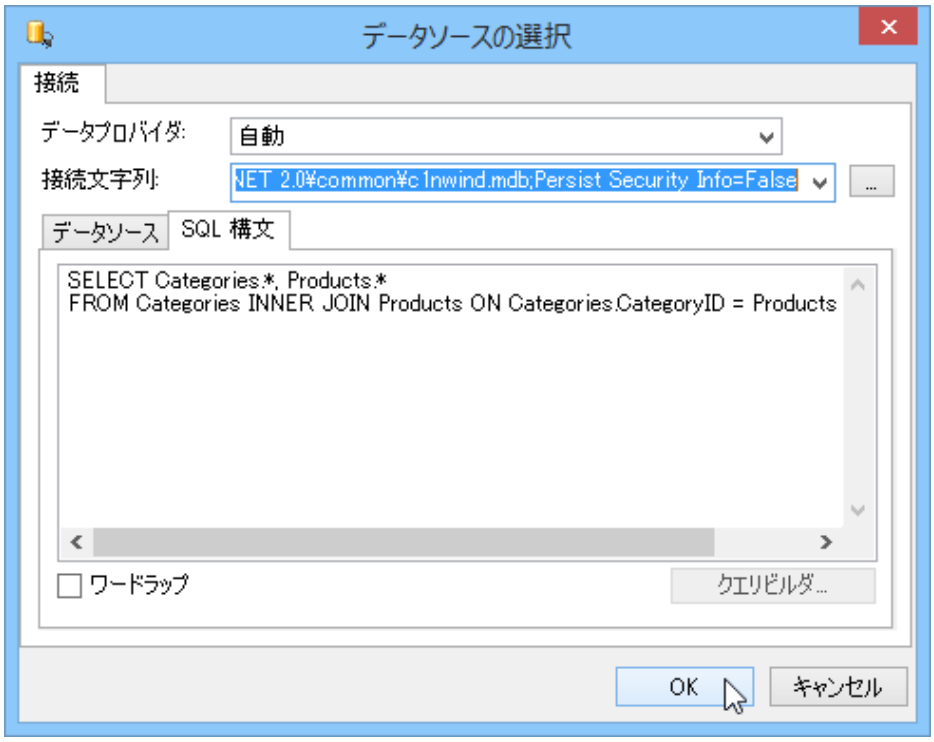



このグループは、以下の項目で構成されます。

- **並び替え/グループ化〉**:このボタンをクリックして、[ソートとグループ化]ダイアログボックスを開きます。これにより、ソートとグループ化の基準を追加／削除できます。詳細については、「データのグループ化」と「データをソートする」を参照してください。
- **〈データソース〉**:このボタンをクリックして、[データソースの選択]ダイアログボックスを開きます。[データソースの選択]ダイ

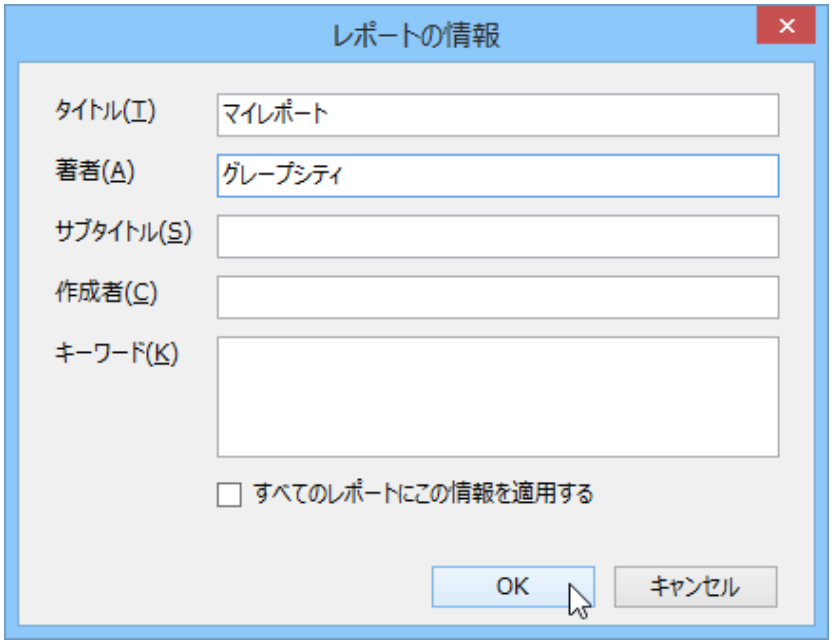

Copyright © GrapeCity, Inc. All rights reserved.

アログボックスでは、新しいデータソースを選択し、接続文字列を変更し、SQL ステートメントを編集できます。〈データソース〉ラジオボタンをクリックして、現在のデータソース内のテーブルとビューを表示します。〈SQL ステートメント〉ラジオボタンをクリックすると、現在の SQL ステートメントを表示できます。



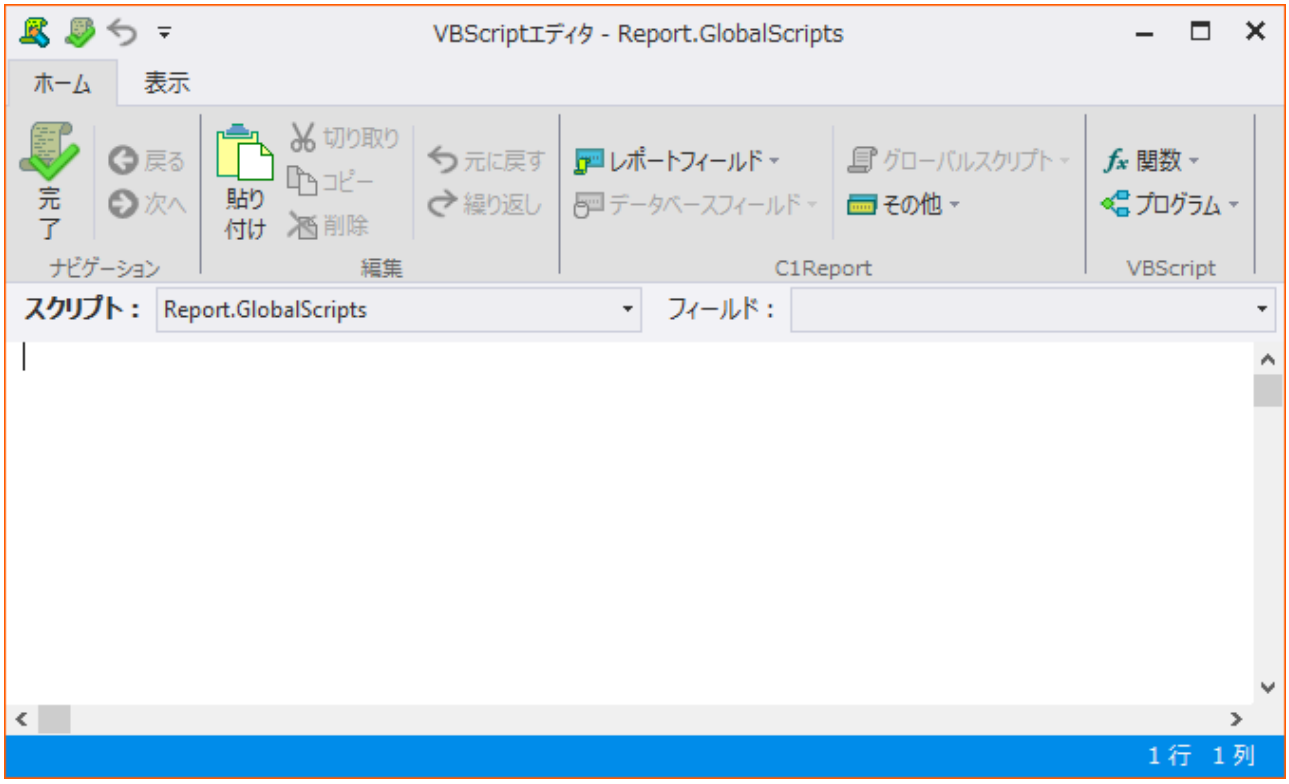

変更する接続文字列を選択して、〈...〉ボタンをクリックします。これによって、[データリンクプロパティ]ダイアログボックスが開きます。XML スキーマ定義ファイル(XDS)を開くには、〈開く〉ボタンをクリックします(詳細については、「XSD ファイルからのデータのロード」を参照してください)。SQL ステートメントを編集または変更するには、〈SQL ステートメントのビルド〉ボタンをクリックし、[SQL ビルダ]ダイアログボックスを開きます。

- **《レポート情報》**:[レポートの情報]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、レポートの[タイトル]、[作成者]、[件名]、[クリエータ]、および[キーワード]を設定できます。レポート情報をすべてのレポートに適用することを選択することもできます。



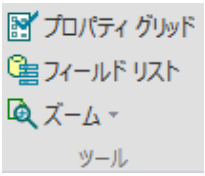

- VBScript エディタ:VBScript エディタはVBScriptエディタ-Report.GlobalScriptsダイアログボックスを開きます。VBScriptエディタで複数なスクリプトが編集できますので簡単にステートメントと式の切り替えます。


Copyright © GrapeCity, Inc. All rights reserved.




### ツールグループ

**「デザイン」**タブの「**ツール**」グループは、次のように表示されます。

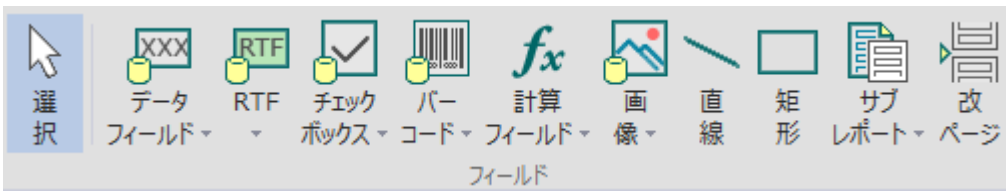



このグループは、以下のオプションで構成されます。

- **《プロパティグリッド》**：この項目を選択すると、左ペインに「プロパティ」タブが表示されます。[F4]キーを使用して「プロパティ」タブを表示することもできます。
- **《フィールドリスト》**：この項目を選択すると、左ペインに「フィールド」タブが表示されます。
- **《ズーム》**：レポートのズームレベルを設定する値を選択できます。

## 「挿入」タブ

**C1Report** デザイナアプリケーションの「挿入」タブの「フィールド」グループは、レポートフィールドを作成するツールを提供します。このツールバーは、デザインモードでのみ有効になります。「挿入」タブの「フィールド」グループは、次のように表示されます。




各フィールドボタンは、フィールドを作成し、そのプロパティを初期化します。「フィールド」グループは、以下のオプションで構成されます。

- **《選択》**：マウスカーソルを矢印カーソルに戻します。
- **《データフィールド》**：ソースレコードセットにバインドされたフィールドを作成します。このボタンをクリックすると、メニュー


Copyright © GrapeCity, Inc. All rights reserved.

が表示され、ソースレコードセットフィールドを選択できます。バインドされたフィールドは、データベースからの未処理データの表示に限定されません。Text プロパティを編集し、VBScript 式を使用できます。

- **〈RTF フィールドの追加〉**: RTF フィールドを作成します。このボタンをクリックすると、メニューが表示され、RTF 形式で表示される同じレポート定義ファイルに含まれる他のフィールドを選択できます。
- **〈チェックボックスの追加〉**: ブール値をチェックボックスとして表示するバインドされたフィールドを作成します。デフォルトで、チェックボックスは通常のチェックマークを表示します。フィールドを作成後、フィールドの Checkbox プロパティの値を変更して、ラジオボタンまたは×印に変更できます。
- **〈バーコードの追加〉**: バーコードを表示するフィールドを作成します。このボタンをクリックすると、メニューが表示され、バーコードとして表示される同じレポート定義ファイルに含まれる他のフィールドを選択できます。
- **〈集計フィールドの追加〉**: 集計フィールドを作成します。このボタンをクリックすると、コードエディタダイアログボックスが表示され、表示する値を含む VBScript 式を入力できます。
- **〈バインドされた画像の追加〉**: レコードセットに格納された画像（またはオブジェクト）を表示するフィールドを作成します。このボタンをクリックすると、メニューが表示され、ソースレコードセットで画像フィールドを選択できます（フィールドが1つの場合、すべてのレコードセットにこの型のフィールドが含まれているわけではありません）。
- **〈バインドされていない画像の追加〉**: ロゴなどの固定画像を表示するフィールドを作成します。このボタンをクリックすると、ダイアログボックスが表示され、画像ファイルをレポートに挿入するように求められます。選択した画像のコピーが作成され、同じディレクトリにレポートファイルとして配置されます。アプリケーションにレポートファイルを埋め込んでいない場合、このファイルをアプリケーションによって配布する必要があります。アプリケーションにレポートファイルを埋め込んでいない場合、バインドされていない画像ファイルも埋め込まれます。
- **〈線の追加〉**: 線を作成します。通常、線はセパレータとして使用されます。
- **〈矩形の追加〉**: 矩形を作成します。通常、矩形はフィールドのグループのハイライト表示、またはテーブルとグリッドの作成に使用されます。
- **〈サブレポートの追加〉**: 他のレポートを表示するフィールドを作成します。このボタンをクリックすると、メニューが表示され、同じレポート定義ファイルに含まれる他のフィールドを選択できます。詳細については、「サブレポートを使用したマスター／詳細レポートの作成 」を参照してください。
- **〈改ページの追加〉**: 改ページを挿入するフィールドを作成します。

C1Report デザイナアプリケーションの「挿入」タブの「カスタムフィールド」グループは、次のように表示されます。

「カスタムフィールド」グループは、以下のオプションで構成されます。

- **〈グラフの追加〉**: グラフを表示するフィールドを作成します。ほとんどのバインドされていないフィールドと異なり、グラフフィールドには複数の値が表示されます。表示するデータを選択するには、グラフフィールドの **Chart.DataX** と **Chart.DataY** プロパティを設定します。X 軸と Y 軸に沿った値を書式設定するには、**Chart.FormatX** と **Chart.FormatY** プロパティを設定します。
- **〈グラデーションの追加〉**: グラデーションフィールドを作成します。通常、グラデーションは、他のフィールドを目立つようにする背景機能として使用されます。詳細については、「グラデーションフィールドの追加」を参照してください。
- **〈スーパーラベルの追加〉**: スーパーラベルを作成します。通常、グラデーションは、他のフィールドを目立つようにする背景機能として使用されます。
- **〈QR コードの追加〉**: QR コードを作成します。
- **〈マップフィールドの追加〉**: マップフィールドを作成します。詳細については、「レポート内のマップ」を参照してください。

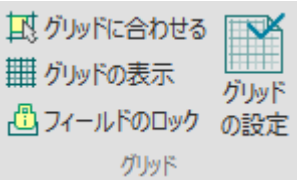
詳細については、「フィールドによるレポートの強化」を参照してください。レポートへのフィールドの追加についての詳細は、「 **レポートフィールドの作成**」を参照してください。

## 「レイアウト」タブ

**「レイアウト」**タブは **C1Report** デザイナの［リボン］メニューにあり、**［グリッド］**、**［配置］**、**［位置］**および**［サイズ］**メニュー機能へのショートカットを提供します。

リボンの**「レイアウト」**タブにある「グリッド」グループは、次のように表示されます。

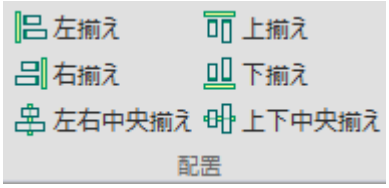

Copyright © GrapeCity, Inc. All rights reserved.




**「グリッド」**グループは、以下のオプションで構成されます。

- **〈グリッドに合わせる〉**:要素をグリッドに配置します。この項目を選択した場合、グリッドの線の間には要素を配置できません。
- **〈グリッドの表示〉**:プレビュー内のレポートの背景にグリッドを表示します。グリッドは、要素の配置と位置揃えに便利です。デフォルトで、このオプションは ON です。
- **〈フィールドのロック〉**:フィールドを希望する位置に配置した後に、不注意によってマウスまたはキーボードで位置が移動されないように、これらをロックします。
- **〈グリッドの設定〉**:**[C1Report デザイナのオプション]**ダイアログボックスを開きます。詳細については、「 C1Report デザイナオプションの設定」を参照してください。

リボンの**「レイアウト」**タブにある**「配置」**グループは、次のように表示されます。




**「配置」**グループは、以下のオプションで構成されます。

- **〈左揃え〉**:選択した項目を水平方向の左側に揃えます。
- **〈右揃え〉**:選択した項目を水平方向の右側に揃えます。
- **〈中央揃え〉**:選択した項目を水平方向の中央に揃えます。
- **〈上揃え〉**:選択した項目を垂直方向の上部に揃えます。
- **〈下揃え〉**:選択した項目を垂直方向の下部に揃えます。
- **〈上下中央揃え〉**:選択した項目を垂直方向の中央に揃えます。

レポートの項目は水平および垂直方向に配置できます。たとえば、ある項目を左側と上部に揃えます。

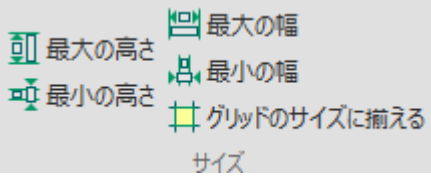
リボンの「レイアウト」タブにある「位置」グループは、次のように表示されます。

**「位置」**グループは、以下のオプションで構成されます。

- **〈最前面へ移動〉**:選択した項目を、層化されたすべての項目の前面に移動します。
- **〈最背面へ移動〉**:選択した項目を、層化されたすべてのオブジェクトの背面に移動します。
- **〈左右を均等〉**:要素の左右の間隔が等しくなるように配置します。
- **〈左右を広く〉**:要素の左右の間隔を拡大します。
- **〈左右を狭く〉**:要素の左右の間隔を縮小します。
- **〈上下を均等〉**:要素の上下の間隔が等しくなるように配置します。
- **〈上下を広く〉**:要素の上下の間隔を拡大します。
- **〈上下を狭く〉**:要素の上下の間隔を縮小します。

リボンの**「レイアウト」**タブにある**「位置」**グループは、次のように表示されます。


Copyright © GrapeCity, Inc. All rights reserved.



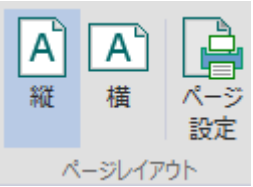

**「サイズ」**グループは、以下のオプションで構成されます。

- **[最大の高さに揃える]**:選択した項目のすべての高さを最も高い項目に設定します。
- **[最小の高さに揃える]**:選択した項目のすべての高さを最も低い項目に設定します。
- **[最大の幅に揃える]**:選択した項目のすべての幅を最も広い項目に設定します。
- **[最小の幅に揃える]**:選択した項目のすべての幅を最も狭い項目に設定します。
- **[グリッドのサイズに揃える]**:選択した項目をグリッドのサイズに設定します。

## 「ページ設定」タブ

「**ページ設定**」タブは C1Report デザイナの[リボン]メニューにあり、[ページレイアウト]メニュー機能へのショートカットを提供します。

リボンの「**ページ設定**」タブにある「**ページレイアウト**」グループは、次のように表示されます。



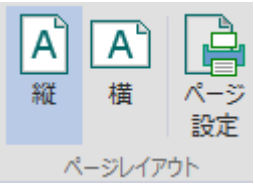

「ページレイアウト」グループは、以下のオプションで構成されます。

- **〈縦〉**:レポートのレイアウトを縦長表示(高さが幅より長い)に変更します。
- **〈横〉**:レポートのレイアウトを横長表示(高さが幅より短い)に変更します。
- **〈ページ設定〉**:プリンタの**[ページ設定]**ダイアログボックスを開きます。

## 「プレビュー」モード

**C1Report** デザイナの「**プレビュー**」モードは現在のレポートを表示します。「**プレビュー**」モードのリボンは以下の項目で構成されます。

リボンの「プレビュー」モードにある「**ページレイアウト**」グループは、次のように表示されます。

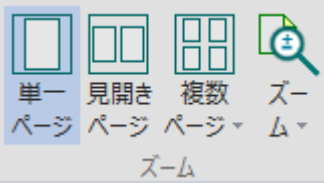



「**ページレイアウト**」グループは、以下のオプションで構成されます。

- **〈縦〉**:レポートのレイアウトを縦長表示(高さが幅より長い)に変更します。
- **〈横〉**:レポートのレイアウトを横長表示(高さが幅より短い)に変更します。
- **〈ページ設定〉**:プリンタの**[ページ設定]**ダイアログボックスを開きます。

リボンの「プレビュー」モードにある「**ズーム**」グループは、次のように表示されます。

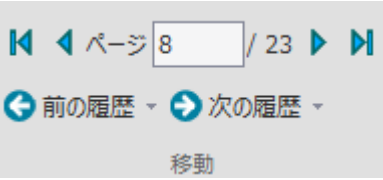

Copyright © GrapeCity, Inc. All rights reserved.




「**ズーム**」グループは、以下のオプションで構成されます。

- **〈単一ページ〉**:一度に1ページをプレビューできます。
- **〈見開きページ〉**:一度に2ページをプレビューできます。
- **〈複数ページ〉**:ドロップダウン矢印をクリックして、一度に3ページ以上をプレビューできます。[4ページ]、[8ページ]、[12 ページ]という3つのオプションがあります。
- **〈ズーム〉**:特定の倍率またはウィンドウに収まるようにページをズームします。

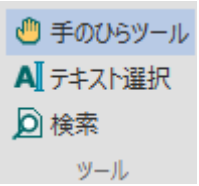
リボンの「プレビュー」タブにある「**移動**」グループは、次のように表示されます。




「**移動**」グループは、以下のオプションで構成されます。

- **〈最初のページ〉**:プレビューの最初のページに移動します。
- **〈前のページ〉**:プレビューの前のページに移動します。
- **[ページ]**:このテキストボックスに数値を入力し、そのページにプレビューを移動します。
- **〈次のページ〉**:プレビューの次のページに移動します。
- **〈最後のページ〉**:プレビューの最後のページに移動します。
- **〈前の履歴〉**:直前に表示したページに戻ります。
- **〈次の履歴〉**:次に表示したページに移動します。これは、〈前の履歴〉ボタンをクリックした後でのみ表示されます。

リボンの「プレビュー」モードにある「**ツール**」グループは、次のように表示されます。




「**ツール**」グループは、以下のボタンで構成されます。

- **〈手のひらツール〉**:手のひらツールでは、ドラッグ＆ドロップ操作を使ってプレビューを移動できます。
- **〈テキスト選択〉**:テキスト選択では、ドラッグ＆ドロップ操作を使ってテキストを選択できます。その後、このテキストをコピーし、別のアプリケーションに貼り付けることができます。
- **〈検索〉**:〈検索〉オプションをクリックすると、ドキュメント内のテキストを検索できる[検索]ペインが開きます。テキストを検索するには、検索対象テキストを入力し、検索オプション(ある場合)を選択して、[検索]をクリックします。

リボンの「プレビュー」モードにある「**エクスポート**」グループは、次のように表示されます。

「エクスポート」グループ内の各項目を使用して、エクスポートするファイルの場所を選択できる**[レポートをファイルにエクスポート]**ダイアログボックスを開きます。「**エクスポート**」グループは、以下のオプションで構成されます。

- **〈PDF〉**:ドキュメントを PDF ファイルにエクスポートします。ドロップダウン矢印には、システムフォントを使用するか、選


Copyright © GrapeCity, Inc. All rights reserved.

択したフォントを PDF ファイルに埋め込むかを選択する[PDF(システムフォント)]と[PDF(埋め込みフォント)]オプションが含まれています。

- **〈HTML〉**:ドキュメントを HTML ファイルにエクスポートします。その後、このテキストをコピーし、別のアプリケーションに貼り付けることができます。ドロップダウン矢印には、プレーン HTML ファイル、組み込まれた矢印リンクを使用してページ付けできる複数の HTML ファイル、またはドリルダウンできるコンテンツを表示する HTML ファイルにエクスポートするかを選択できる[プレーン HTML]、[ページ付けされる HTML]、および[ドリルダウン HTML]オプションが含まれています。
- **〈Excel〉**:ドキュメントを Microsoft Excel ファイルにエクスポートします。ドロップダウン矢印には、ドキュメントを XLS ファイルとして保存するか、XLSX ファイルとして保存するかを選択できる[Microsoft Excel 97]と[Microsoft Excel 2007 - OpenXML]オプションが含まれています。
- **〈RTF〉**:ドキュメントをリッチテキストファイル(RTF)にエクスポートします。
- **〈テキスト〉**:ドキュメントをテキストファイル(TXT)にエクスポートします。
- **〈その他〉**:[その他]ドロップダウン矢印をクリックして、レポートをエクスポートする際に以下の追加オプションを使用できます。[TIFF 画像](TIFF としてエクスポート)、[RTF(位置固定)]、[単一ページ テキスト]、[圧縮されたメタファイル](ZIP としてエクスポート)。

エクスポートについての詳細は、「レポートのエクスポートと配布」を参照してください。

リボンの「プレビュー」モードにある「**生成**」グループは、次のように表示されます。

リボンの「プレビュー」モードにある「生成」グループは「**リフレッシュ**」ボタンで構成されます。このボタンをクリックすると、現在のレポートが再生成されます。ドキュメントが生成されている時このボタンは「停止」ボタンに切り替えますので、「停止」ボタンをクリックすることでレポートの生成操作を停止することもできます。

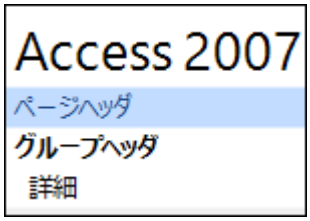
その他「プレビュー」モードのリボンは、以下のオプションで構成されます。

- **「ページ」タブ**:「ページ」タブはC1Report デザイナが「プレビュー」モードにある場合のみ表示されます。「ページ」タブでは、ドキュメントにある全てのページのサムネイルが表示されます。
- **「アウトライン」タブ**:「アウトライン」タブはC1Report デザイナが「プレビュー」モードにある場合のみ表示されます。「アウトライン」タブでは、ドキュメントで表示するデータの概要が表示されます。
- **「検索」タブ**:このタブはC1Report デザイナが「プレビュー」モードにある場合のみ表示されます。「検索」タブに「検索文字列」ペインがあり、そのペインを使用してドキュメントにて文字列を検索できます。

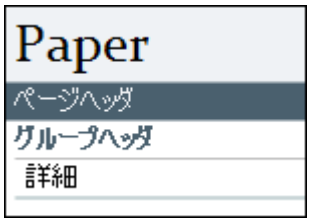
## 「スタイルギャラリー」ダイアログボックス

**「スタイルギャラリー」**ダイアログボックスは、**C1Report デザイナ**組み込みの34種のスタイルを提供し、現在のレポートにいずれかのスタイルを適用する機能を提供します。**「デザイン」**タブの**〈スタイル〉**ボタンをクリックして**「スタイルギャラリー」**ダイアログボックスにアクセスして、カスタムスタイルを作成することもできます。

**「スタイルギャラリー」**ダイアログボックスに含んでいる組み込みのスタイルは以下のように表示されます。

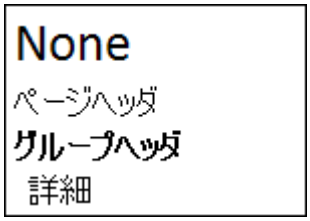
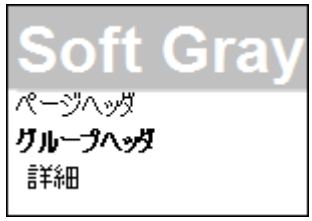
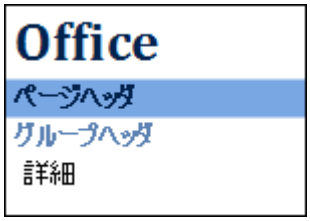
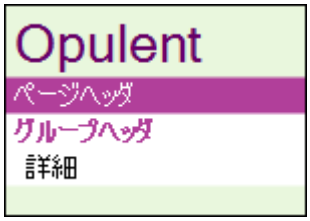
| スタイル名 | 外観 | スタイル名 | 外観 |
|---|---|---|---|
| Access 2007 |  | Oriel | |


Copyright © GrapeCity, Inc. All rights reserved.

| スタイル名 | 外観 | スタイル名 | 外観 |
|---|---|---|---|
| Access 2003 | | Origin | |
| Apex | | Paper |  |
| Aspect | | Solstice | |
| Civic | | Technic | |
| Concourse | | Trek | |
| ComponentOne | | Urban | |
| Equity | | Verve | |
| Flow | | Windows Vista | |



Copyright © GrapeCity, Inc. All rights reserved.

| スタイル名 | 外観 | スタイル名 | 外観 |
|---|---|---|---|
| Foundry | | Bold | |
| Median | | Casual | |
| Metro | | Compact | |
| Module | | Corporate | |
| None |  | Formal | |
| Northwind | | Soft Gray |  |
| Office |  | Verdana | |
| Opulent |  | WebReport | |



Copyright © GrapeCity, Inc. All rights reserved.

# Visual Studio からの C1Report デザイナへのアクセス

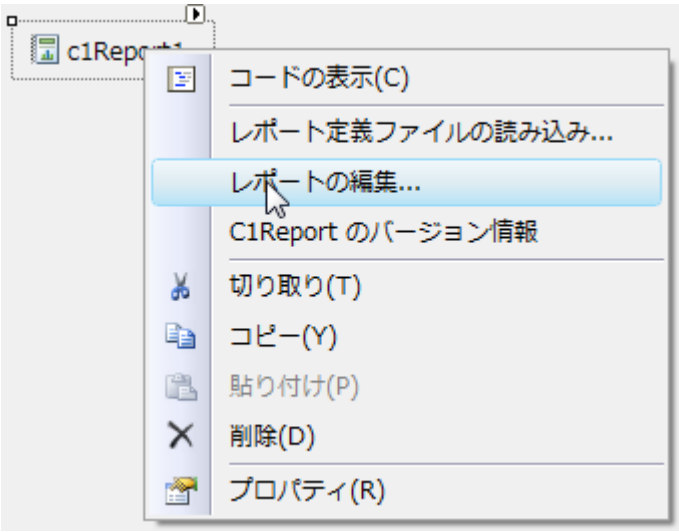
Visual Studio から **C1Report デザイナ**アプリケーションにアクセスするには、以下の方法のいずれかを使用します。

- **[C1Report タスク]メニュー**

  **C1Report**コンポーネントの右上端にあるスマートタグ（）をクリックし、[**C1Report タスク**]メニューを開き、[**レポートの編集**]を選択します。

- **コンテキストメニュー**

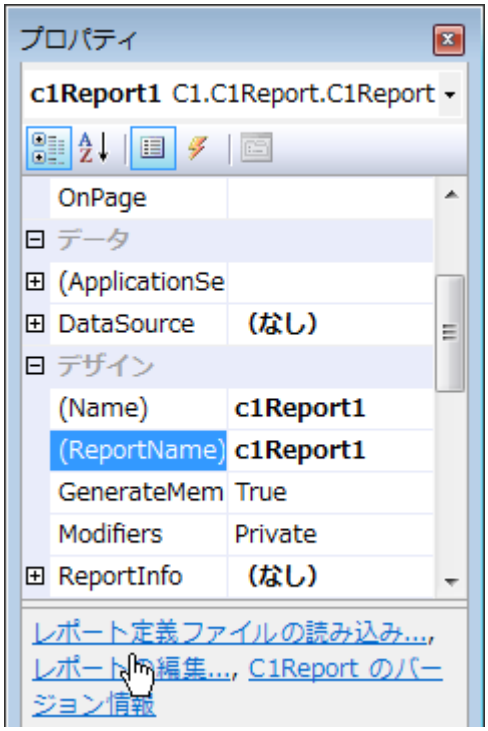
  C1Report コンポーネントを右クリックし、コンテキストメニューから[**レポートの編集**]を選択します。




- **プロパティウィンドウ**

  プロパティウィンドウの下部にある[**レポートの編集**]リンクをクリックします。


Copyright © GrapeCity, Inc. All rights reserved.

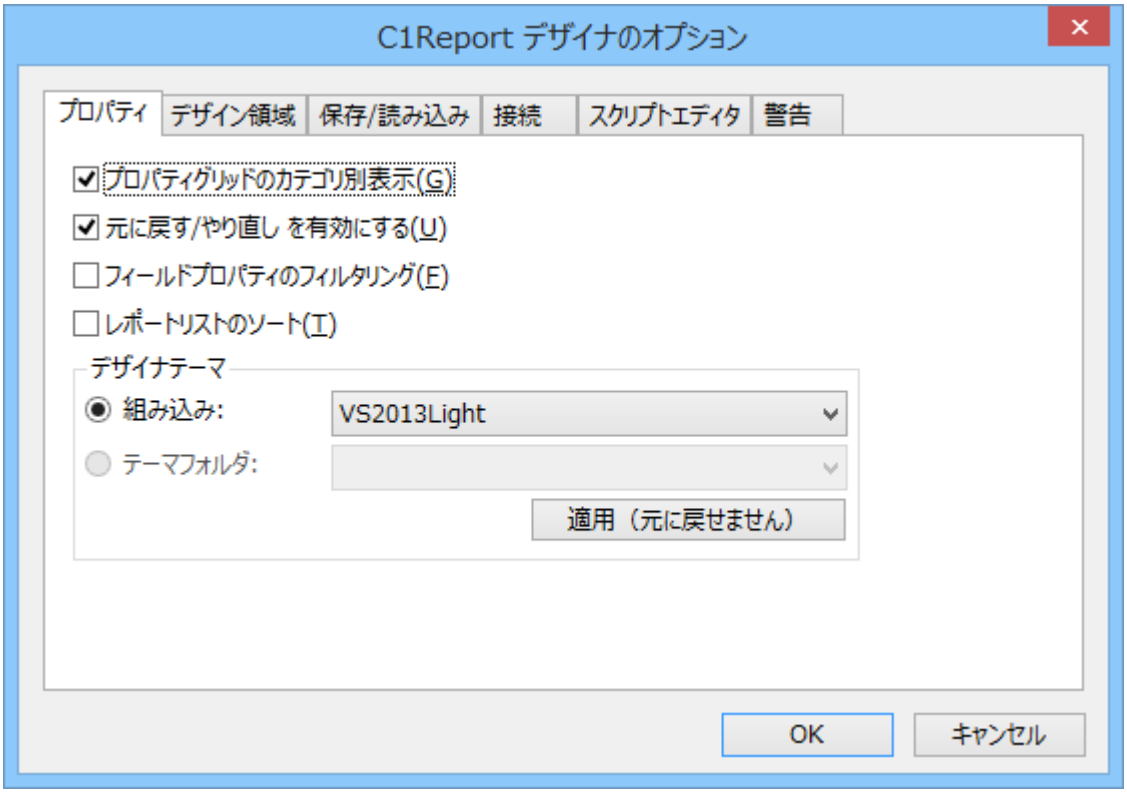



# C1Report デザイナオプションの設定

[**C1Report デザイナのオプション**]ダイアログボックスにアクセスするには、〈**ファイル**〉メニューボタンをクリックし、[**オプション**]をクリックします。〈**ファイル**〉メニューについての詳細は、「〈ファイル〉メニュー」を参照してください。

[**C1Report デザイナのオプション**]ダイアログボックスには、アプリケーションの外観と動作を制御するオプションが含まれています。以下のオプションがあります。

### 「プロパティ」タブ

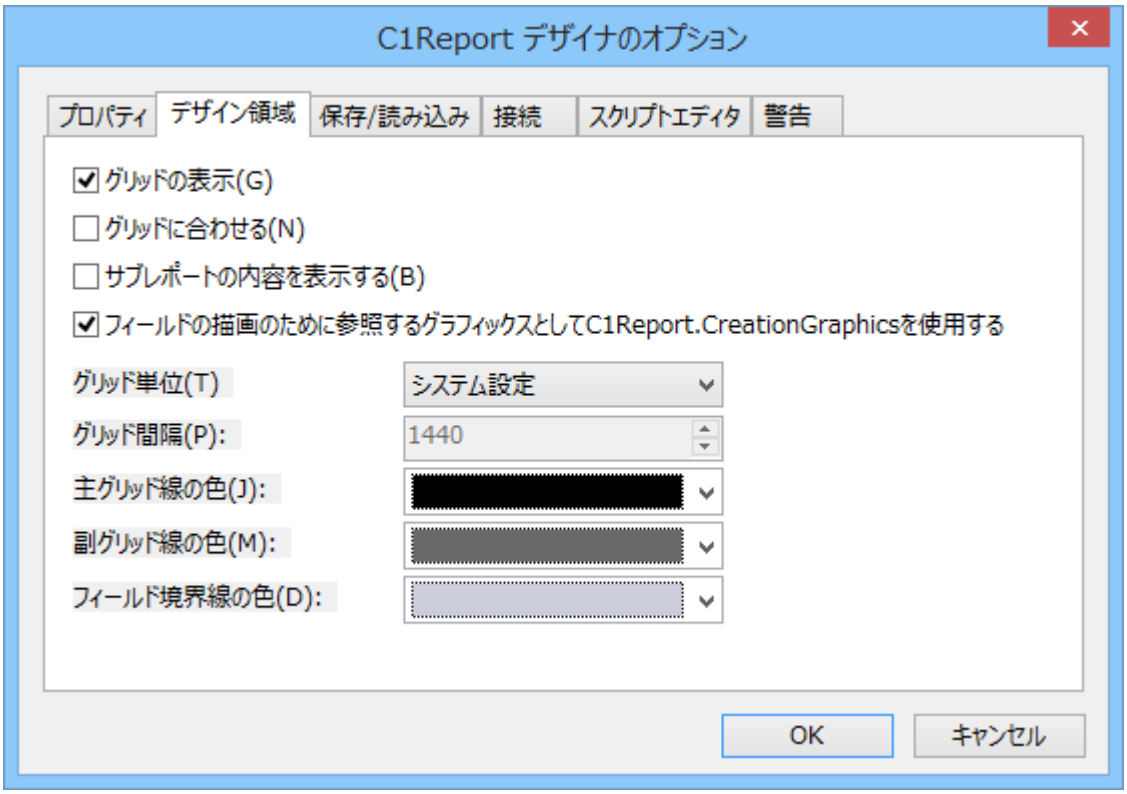




Copyright © GrapeCity, Inc. All rights reserved.

- [**プロパティグリッドのカテゴリ別表示**]：プロパティタイプによってプロパティグリッドを分類します。プロパティグリッドにアクセスするには、デザインビューの左ペインの下部にある「**プロパティ**」タブをクリックします。
- [**元に戻す/やり直しを有効にする**]：アプリケーションの[元に戻す]と[やり直し]を有効にします。
- [**フィールドプロパティのフィルタリング**]：設定したプロパティによって、プロパティグリッドをフィルタリングします。プロパティグリッドにアクセスするには、デザインビューの左ペインの下部にある「プロパティ」タブをクリックします。
- [**レポートリストのソート**]：「**レポート**」タブにリストされたレポートのリストをソートします。レポートにアクセスするには、デザインビューの左ペインの下部にある「**レポート**」タブをクリックします。

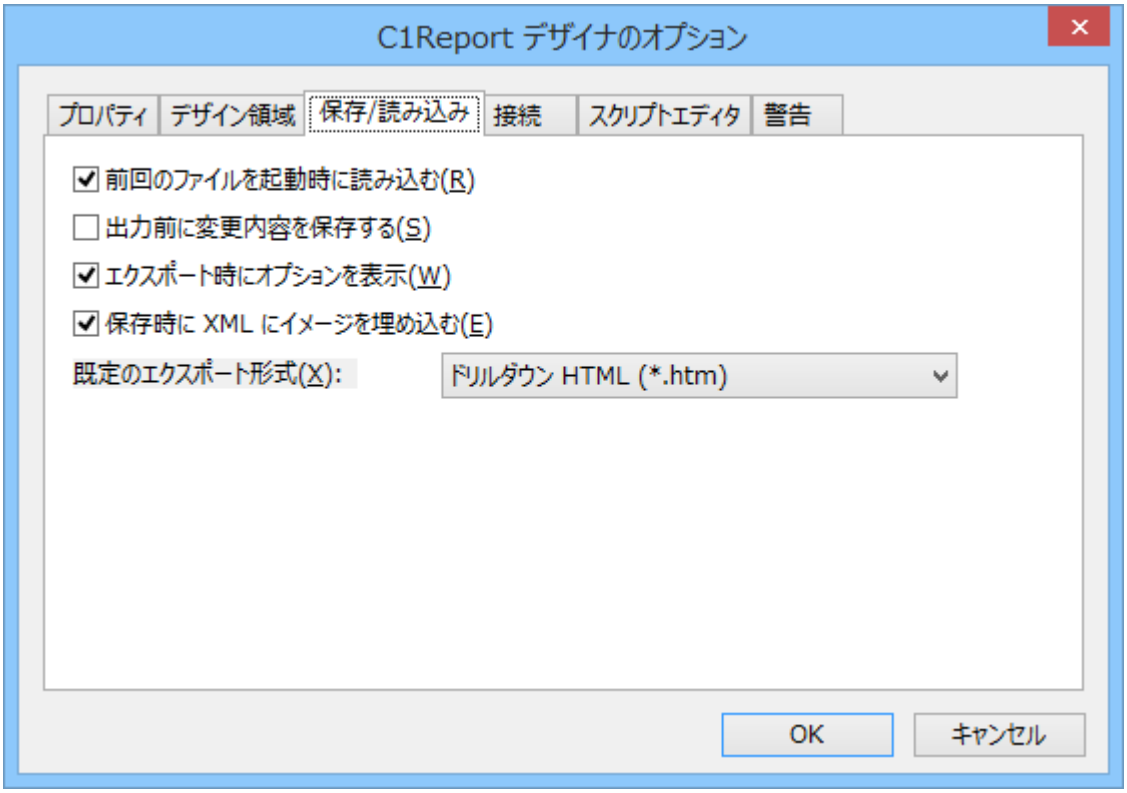
### 「デザイン領域」タブ




- [**グリッドの表示**]：レポートプレビューウィンドウにグリッドを表示します。
- **[グリッドに合わせる]**：すべてのオブジェクトをレポートのグリッドに配置します。このオプションを選択した場合、グリッド線の間にオブジェクトを配置できません。
- [**サブレポートの内容を表示する**]：レポート内のサブレポートの内容を表示します。
- [**グリッド単位**]：グリッドのスペースを設定する方法を示します。オプションには、[**自動**]、[ヤード・ポンド法（インチ）]、[メートル法（センチメートル）]、および[**カスタム**]があります。
- [**グリッド間隔**]：グリッド線の間隔を設定します。このオプションは、[**グリッド単位**]を[**カスタム**]に設定した場合にのみ利用できます。
- [**主グリッド線の色**]：主グリッド線の色を設定します。
- [**副グリッド線の色**]：副グリッド線の色を設定します。
- [**フィールド境界線の色**]：レポート内のフィールドの端の色を設定します。

### 「保存/読み込み」タブ

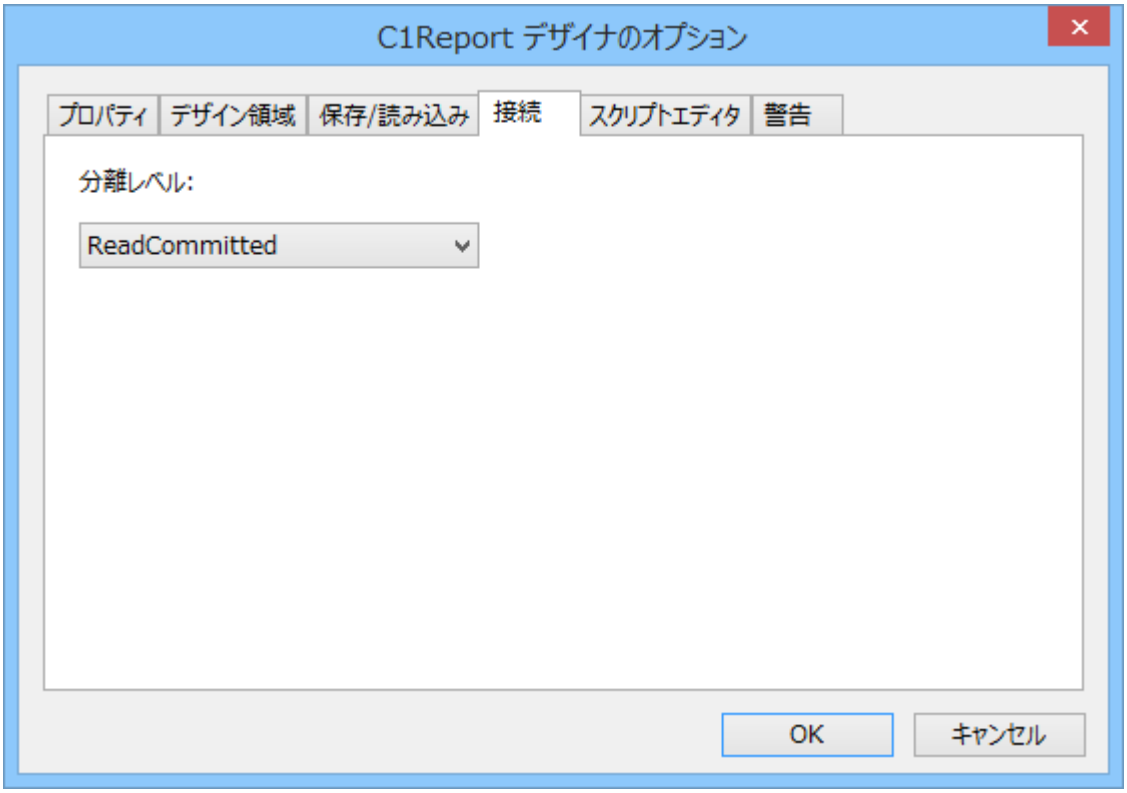

Copyright © GrapeCity, Inc. All rights reserved.




- [**前のファイルを起動時に読み込む**]：このオプションを ON にすると、C1Report デザイナアプリケーションを起動したときに、直前に開いたファイルが常に表示されます。
- [**起動時に更新を確認**]：このオプションを ON にすると、C1Report デザイナアプリケーションを起動したときに、更新が確認されます。
- [**出力前に変更内容を保存する**]：このオプションを ON にすると、表示する前に変更が保存されます。
- [**エクスポート時にオプションを表示**]：このオプションを ON にすると、エクスポート時にオプションが表示されます。
- [**保存時に Xml にイメージを埋め込む**]：このオプションを ON にすると、レポートの保存時に画像が XML に埋め込まれます。
- [**既定のエクスポート形式**]：デフォルトのエクスポート形式を設定します。エクスポートについての詳細は、「プレビュー」モードを参照してください。

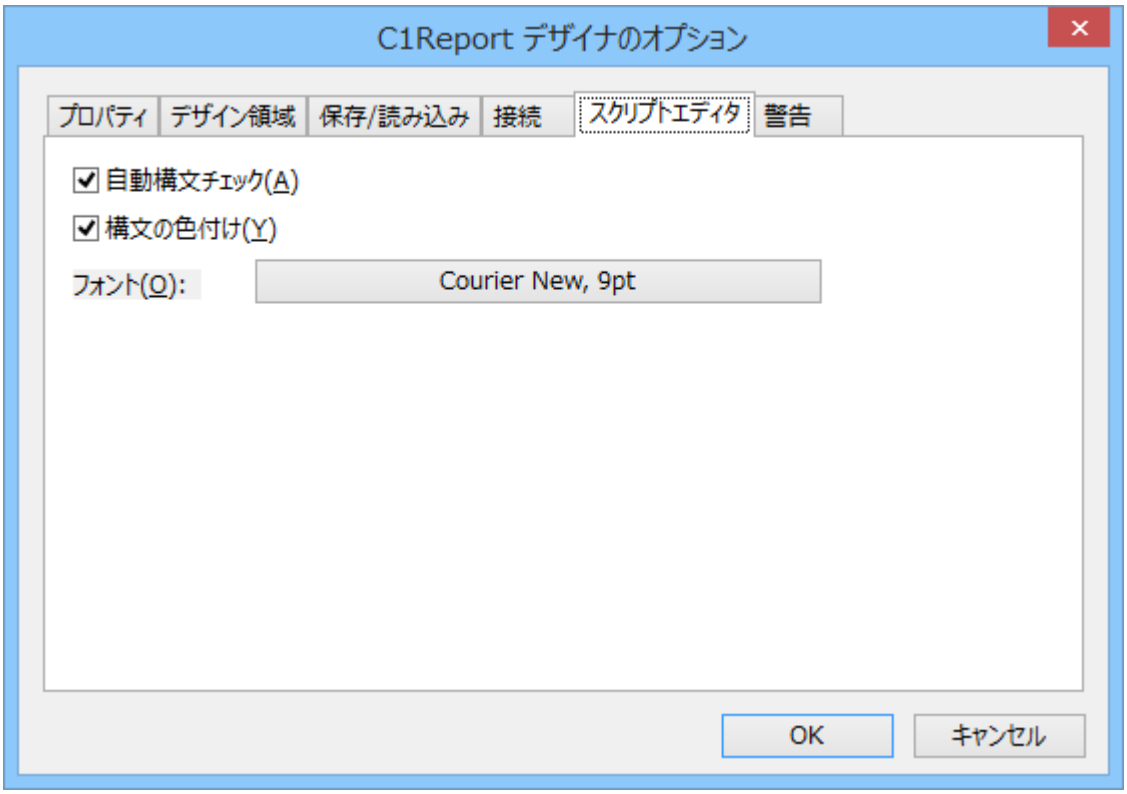
### 「接続」タブ


Copyright © GrapeCity, Inc. All rights reserved.




- [**分離レベル**]:OleDB トランザクションの分離レベルを指定します。デフォルト値は、「ReadCommitted」です。

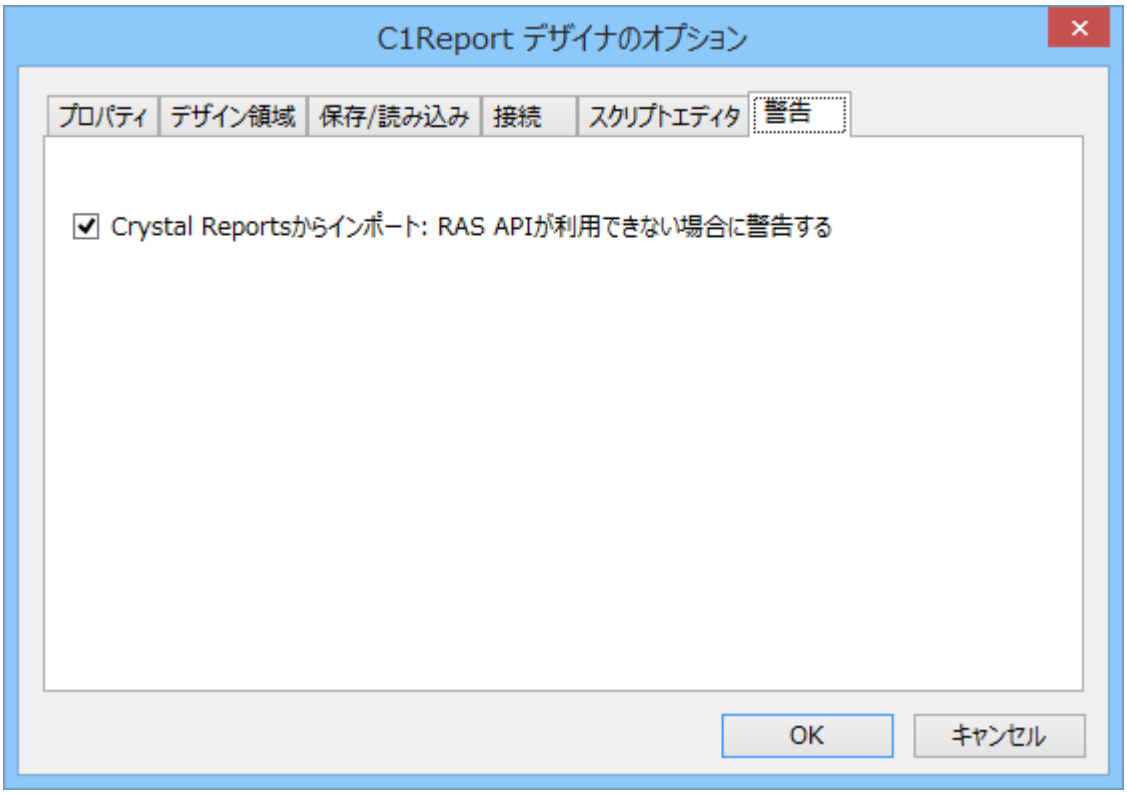
### 「スクリプトエディタ」タブ




- [**フォント**]:[**VBScript エディタ**]ダイアログボックスで使用するテキストの外観を定義します。
- [**構文の色付け**]:[**VBScript エディタ**]ダイアログボックスで、構文を自動的に色付けするかどうかを指定します。
- [**自動構文チェック**]:[**VBScript エディタ**]ダイアログボックスで、構文を自動的にチェックするかどうかを指定します。


Copyright © GrapeCity, Inc. All rights reserved.

### 「警告」タブ

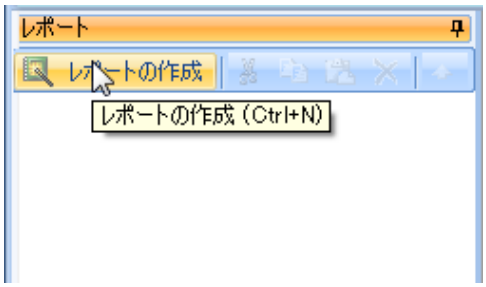



- **[Crystal Reportsからインポート]**:RAS APIが利用できない場合に警告する:このオプションをオンすると、C1Report デザイナにCrystal Reports をインポートしたときRAS APIが存在しない場合、警告がスローされます。

## 基本的なレポート定義ファイルの作成

レポートを設計することで、印刷されるページ上にさまざまな方法でデータを表示できます。**C1Report デザイナ**を使用して、総合リスト、サマリーレポート、またはデータの特別なサブセット(請求書など)を設計できます。

新しいレポートの作成を開始するには、「**新規レポートウィザード**」を使用するのが最も簡単です。メインツールバーの「**レポートの新規作成**」ボタンをクリックするか、[**ファイル | レポートの新規作成**]メニューオプションを選択します。




Visual Studio で、**C1Report** のコンテキストメニューまたは**[C1Report のタスク]**メニューから **C1Report ウィザード**にアクセスして、**[レポートの編集]**をクリックすることもできます。**[レポートの編集]**リンクへのアクセスの詳細については、「C1Report のタスクメニュー」または「C1Report のコンテキストメニュー」を参照してください。

新規レポートウィザードが表示されるので、案内に従って5つの簡単な手順を実行します。

1. **新規レポートのデータソースを選択する**

   このページでは、レポートデータの取得に使用される **ConnectionString** と **RecordSource**を選択します。

   ConnectionString は、次の3つの方法で指定できます。:

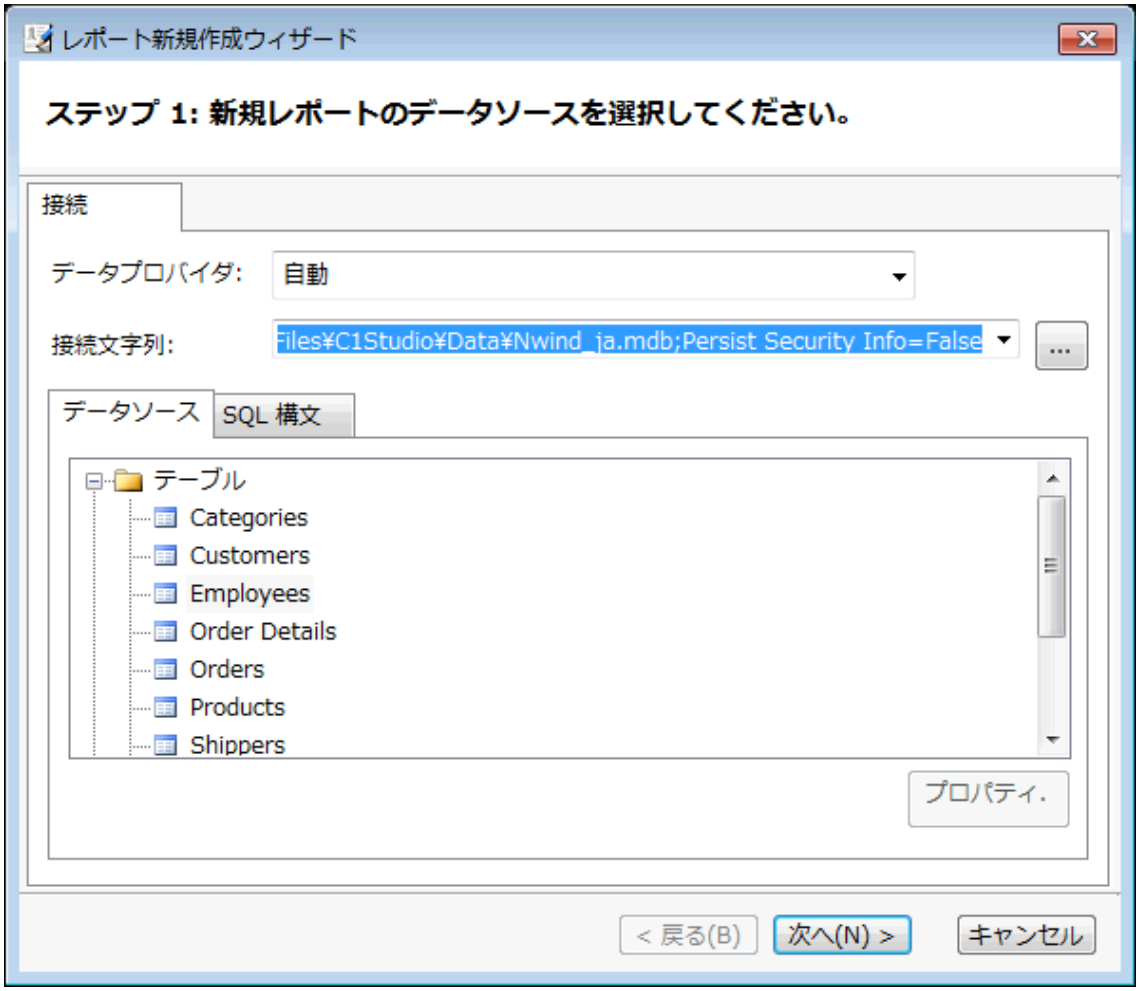
   - 文字列をエディタに直接入力します。
   - ドロップダウンリストを使用して、最近使った接続文字列を選択します。デザイナは、最近使用された接続文字列を8つ


Copyright © GrapeCity, Inc. All rights reserved.

まで記録します。
- 省略符(...)の付いたボタンをクリックして、標準の接続文字列ビルダを起動します。

RecordSource の文字列は、次の2つの方法で指定できます。

- [**テーブル**]ボタンをクリックし、リストから**テーブル**を選択します。
- [**SQL**]ボタンをクリックし、エディタに **SQL** 文を入力(または貼り付け)します。




2. **レポートに含めるフィールドを選択する**

このページには、手順1で選択したレコードセットで利用できるフィールドのリスト、およびレポートのグループフィールドと詳細フィールドを定義する2つのリストがあります。グループフィールドは、データのソート方法と集計方法を定義します。また、詳細フィールドは、レポートに表示する情報を定義します。

フィールドは、ドラッグによってリスト間を移動できます。[**詳細**]リストにフィールドをドラッグすると、そのフィールドをレポートに含めることができます。また、リスト内でフィールドをドラッグすると、フィールドの順番を変更できます。フィールドをドラッグして[**利用可能なフィールド**]リストに戻すと、レポートからそのフィールドが削除されます。


Copyright © GrapeCity, Inc. All rights reserved.

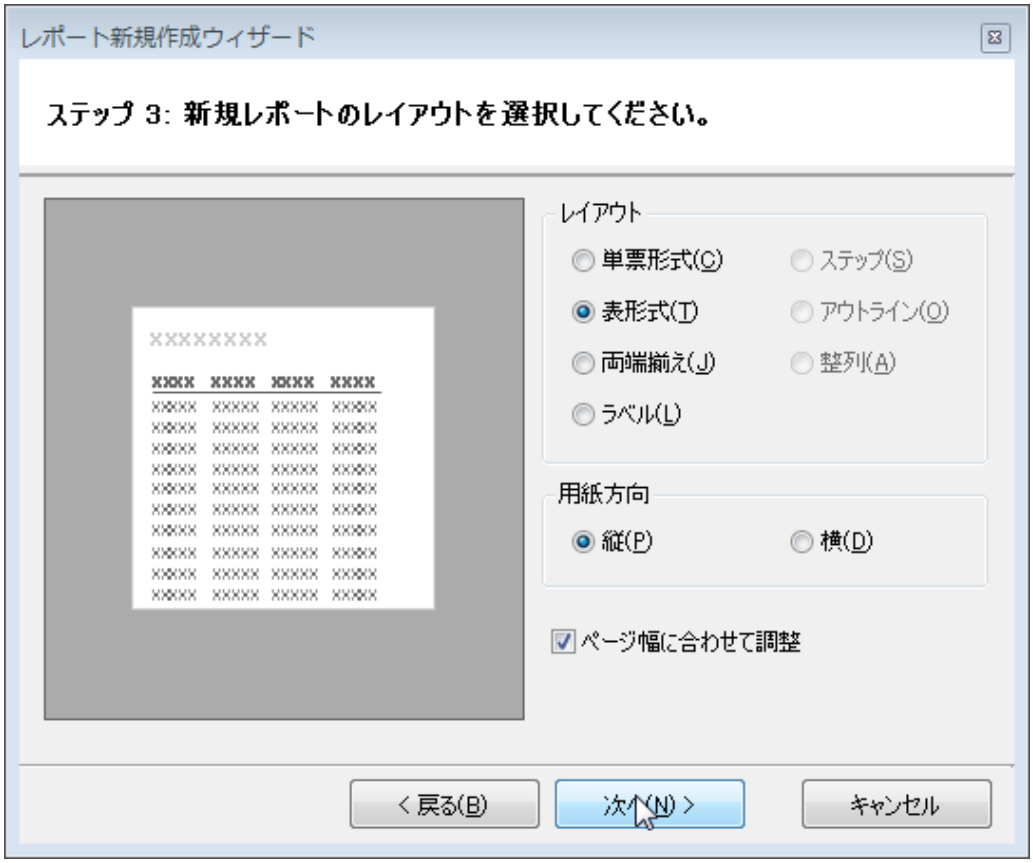
3. **新規レポートのレイアウトを選択する**

   このページには、ページ上でのデータの構成方法を定義するためのさまざまなオプションが用意されています。レイアウトを選択すると左側にサムネールが表示され、ページ上でのレイアウトの外見を確認できます。レイアウトは2種類に分かれており、一方はグループなしのレポート、もう一方はグループ付きのレポートです。作成するレポートの外見に最も近いレイアウトを選択します。

   また、このページでは用紙の方向とフィールドをページ幅に合わせて調整するかどうかを選択できます。

   [**ラベル**]レイアウトオプションは、Avery スタイルのラベルを印刷するために使用されます。このオプションを選択すると、印刷するラベルの種類を指定するためのページが表示されます。

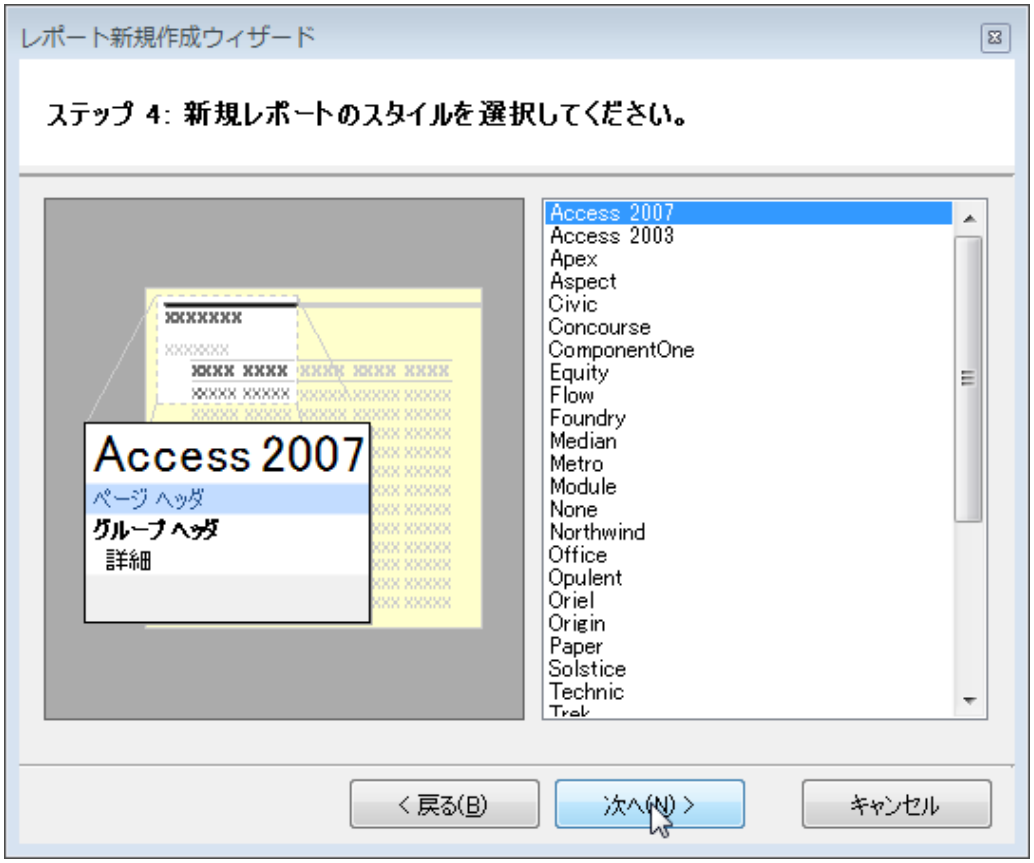

Copyright © GrapeCity, Inc. All rights reserved.



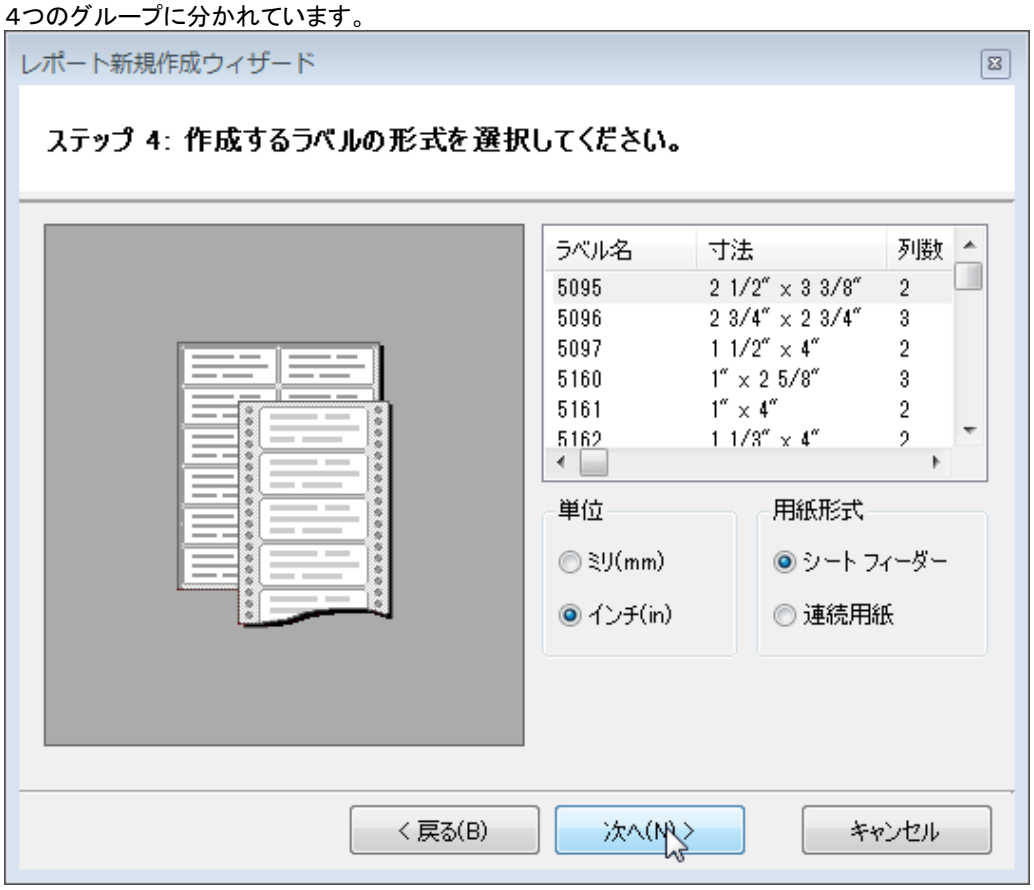

4. **新規レポートのスタイルを選択する(ラベル以外のレイアウトを選択した場合)**

   - **Style Layouts**
     このページでは、新しいレポートで使用するフォントと色を選択できます。手順3のページと同様に、選択したスタイルの外見を確認できるプレビューが表示されるので、好みのスタイルを選択します。スタイルは、後で詳細に設定したり、設定を変更することもできます。W


Copyright © GrapeCity, Inc. All rights reserved.




- **印刷するラベルの種類を選択する(ラベルレイアウトを選択した場合)**
  このページでは、作成するラベルの種類を選択できます。デザイナでは、170種類の定義済みラベルを選択できます。ラベルは、長さの単位(センチメートルまたはインチ)および使用する用紙の種類(単票または連続用紙)に基づいて、4つのグループに分かれています。

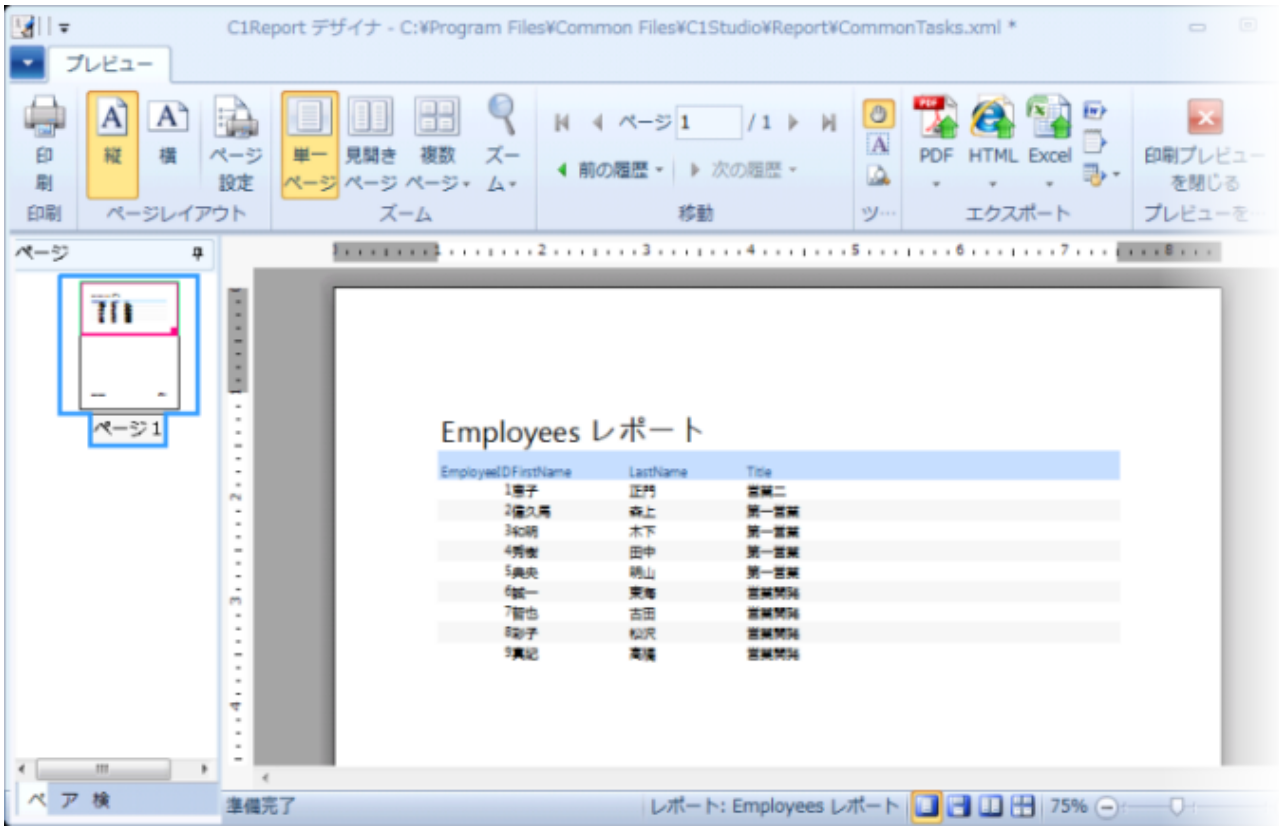



5. **新規レポートのタイトルを選択する**


Copyright © GrapeCity, Inc. All rights reserved.

この最後の手順のページでは、新しいページのタイトルを選択します。また、新しいレポートをすぐにプレビューするか、プレビューする前に編集モードに移動してデザインに手を加えるかを指定します。

レポートのプレビューを選択し、[完了]をクリックすると、すぐに**デザイナ**のプレビューペインにレポートが表示されます。次に表示例を示します。



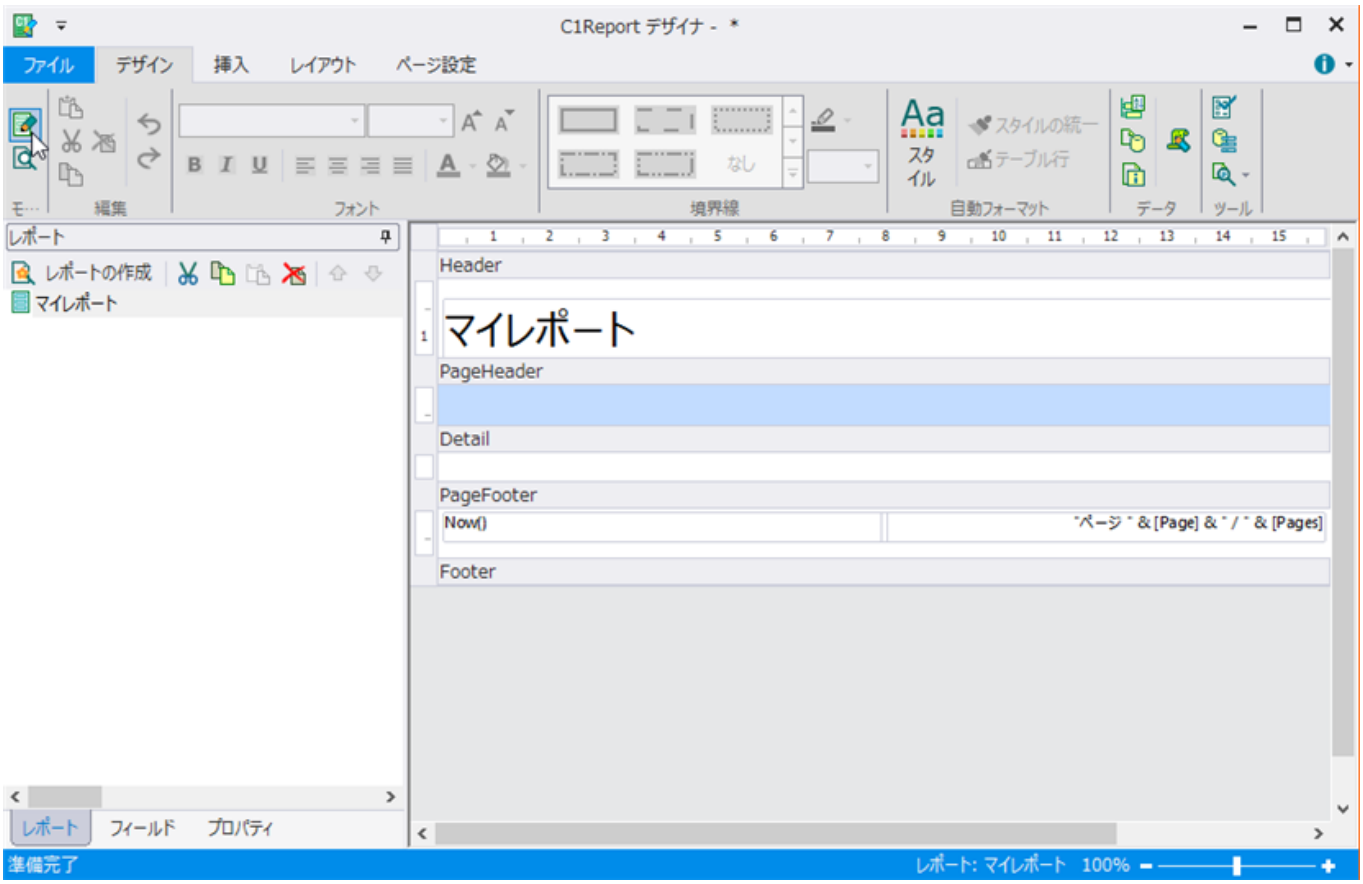


Copyright © GrapeCity, Inc. All rights reserved.

# レポートレイアウトの変更

ウイザードで作成したレポートを使用して、レイアウトを変更する操作を開始できます。また、今まで作成できたレポートを調整するには、**C1ReportDesigner** アプリケーションを使用してみてください。

デザイン機能を使用するには、「**デザイン**」モードをクリックします。メインウィンドウの右側のペインがプレビューモードからデザインモードに切り替わり、レポートを構成するコントロールとフィールドが表示されます。




上の図は、レポートがどのようなセクション（ヘッダ、ページヘッダなど）に分割されているかを示しています。各セクションには、印刷されるレポートで必要なラベル、変数、式などを含むフィールドがあります。この例では、ヘッダセクションにレポートのタイトルを含むラベルがあります。また、ページヘッダセクションには、詳細セクションに表示されるフィールドを識別するラベルがあり、ページフッタセクションには、現在の時刻、ページ番号、およびレポートの総ページ数を表示するフィールドがあります。

レポートのセクションは、各ページ、グループ、およびレポートの最初と最後の外観を決定します。次の表に、各セクションがレポート内のどこに表示されるか、および通常はどのような目的で使用されるかについて説明します。

| セクション | 表示位置 | 通常の内容 |
|---|---|---|
| レポートヘッダ | レポートごとに一度 | レポート全体のタイトルやサマリー情報。 |
| ページヘッダ | ページごとに一度 | 詳細フィールドについて説明するラベルやページ番号。 |
| グループヘッダ | グループごとに一度 | 現在のグループを識別するフィールドやグループの集計値（合計、総計に対する割合など）など。 |
| 詳細 | レコードごとに一度 | ソースレコードセットからのデータを含むフィールド。 |
| グループフッタ | グループごとに一度 | グループの集計値。 |
| ページフッタ | ページごとに一度 | ページ番号、ページ数、印刷日、レポート名など。 |


Copyright © GrapeCity, Inc. All rights reserved.

| セクション | 表示位置 | 通常の内容 |
|---|---|---|
| レポートフッタ | レポートごとに一度 | レポート全体のサマリー情報。 |

セクションは、直接追加または削除できないことに注意してください。レポート内のセクションの数は、グループの数によって決まります。すべてのレポートには、5つの固定セクション(レポートヘッダ／フッタ、ページヘッダ／フッタ、詳細)と、グループごとに2つのセクション(グループヘッダとグループフッタ)があります。セクションの **Visible**プロパティを **False** に設定することによって、任意のセクションを非表示にできます。

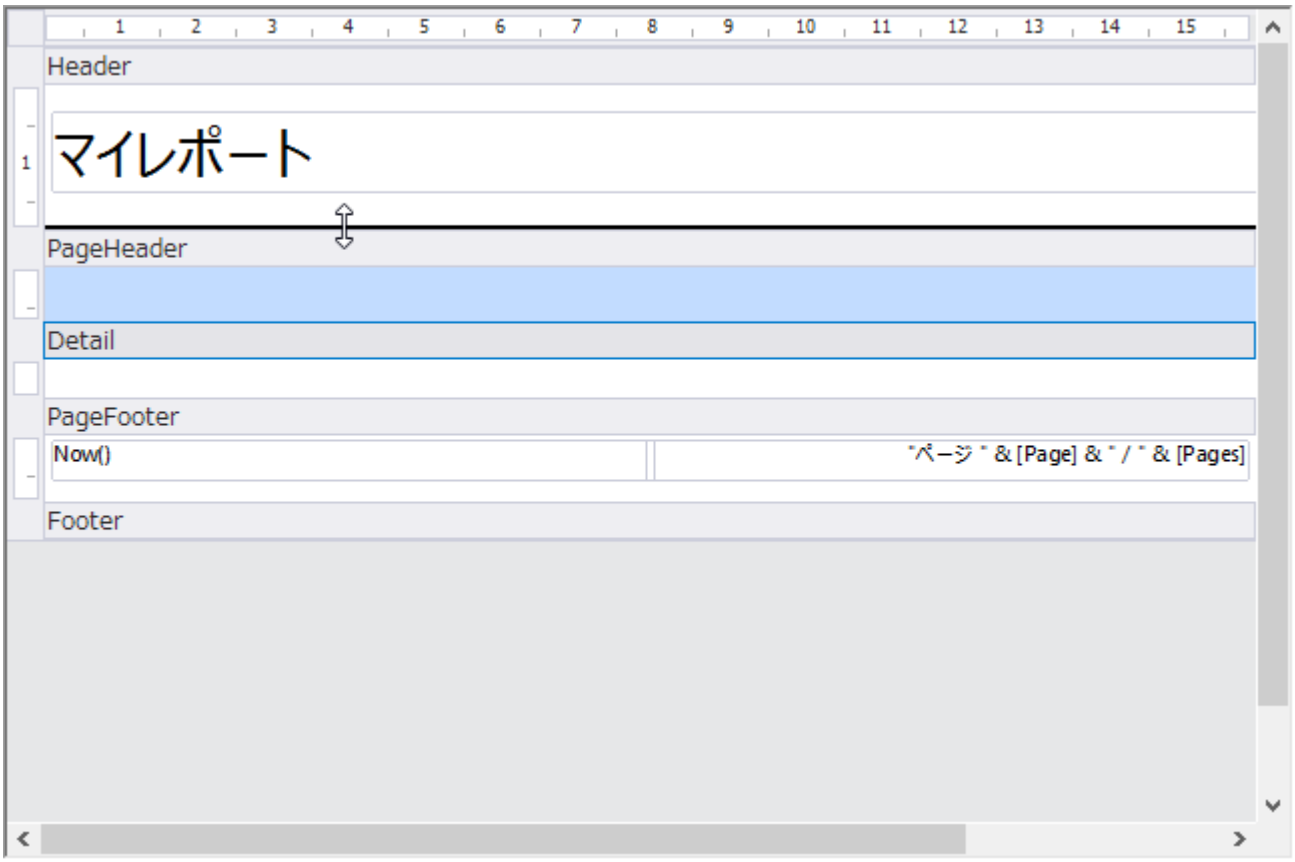
プロパティウィンドウでセクションのプロパティを変更できるし、マウスで移動やサイズ変更することも可能です。

## セクションのサイズ変更

レポートの見栄えをよくするために、任意のセクションに直線、四角形、画像などを追加できます。また、プロパティウィンドウでプロパティを変更することによって、既存のフィールドやセクションに変更を加えることもできます。さらに、マウスを使ってセクションを移動したり、サイズを変更したりすることもできます。

セクションのサイズを変更するには、セクションの境界線を目的の位置までドラッグします。設計ウィンドウの左側と上部にあるルーラーに、各セクションのサイズが(ページマージンを除いて)表示されます。セクションは、その内部にあるフィールドの表示に必要な高さと幅より小さくすることはできません。セクションのサイズをそれより小さくするには、内部にあるフィールドを移動またはサイズ変更した後で、セクションをサイズ変更します。

この操作を確認するために、ページヘッダセクションの下部と詳細セクションの上部にある灰色のバーとの間の領域にマウスを移動してみます。ポインタが変化して、ポインタがサイズ変更領域にあることがわかります。マウスをクリックし、セクションの高さが元の高さの2倍程度になるまで線をドラッグします。




ボタンを離すと、セクションのサイズが変更されます。

# フィールドによるレポートの強化

レポートを強化するには、任意のセクションにフィールド(線、矩形、ラベル、画像、グラフなど)を追加します。プロパティウィン


Copyright © GrapeCity, Inc. All rights reserved.

ドウでプロパティを変更して既存のフィールドを変更したり、マウスを使ってフィールドのサイズを変更したりすることもできます。

## レポートフィールド

**C1Report デザイナ**アプリケーションの「**デザイン**」タブの「**フィールド**」グループは、レポートフィールドを作成するツールを提供します。このツールバーは、デザインモードでのみ有効になります。各ボタンは、フィールドを作成し、そのプロパティを初期化します。「**フィールド**」グループについての詳細は、「**フィールド」グループ**」を参照してください。レポートへのフィールドの追加についての詳細は、「**レポートフィールドの作成**」を参照してください。

## グラフフィールドの追加

グラフフィールドは、**C1Chart** コントロールを使用して実装します。

**メモ**:グラフを使用する場合は、アプリケーションと共に **C1Chart** アセンブリを配置する必要があります。

### レポートにグラフフィールドを追加するには、以下の手順を実行します。

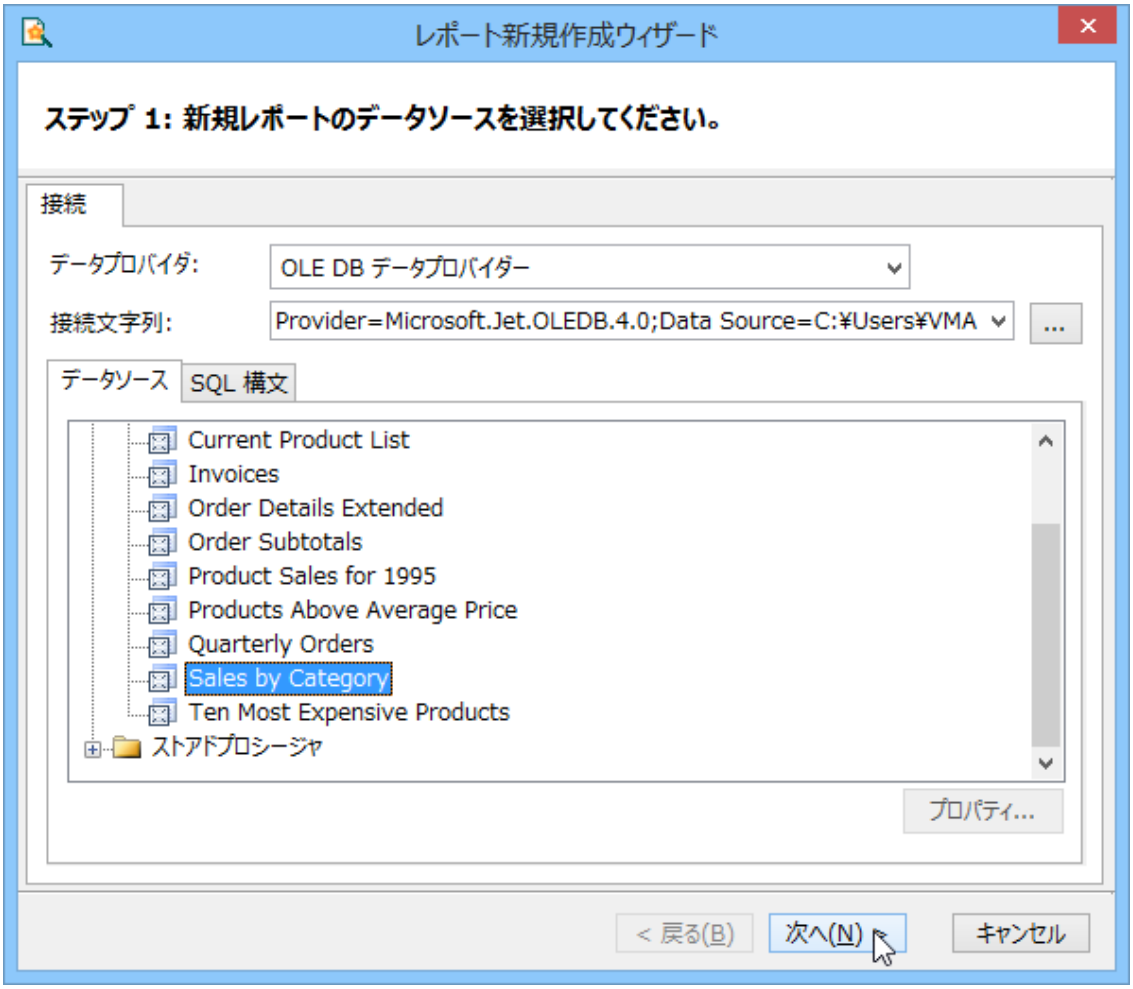
1. **C1Report デザイナ**アプリケーションでレポートを開きます。
2. **[挿入]**タブの**[カスタムフィールド]**グループで**[グラフ]**ボタンをクリックし、グラフを表示するレポート内の領域をマークします。
3. その後、通常どおり、フィールドプロパティを設定します。

グラフフィールドの通常と異なる唯一の側面は、大部分のバインドされたフィールドと異なり、複数の値を表示することです。表示するデータを選択するには、グラフフィールドの **Chart.DataX** と **Chart.DataY** プロパティを設定します。X 軸と Y 軸に沿った値を書式設定するには、**Chart.FormatX** と **Chart.FormatY** プロパティを設定します。**Chart.ChartType**、**Chart.Palette** などの他のプロパティを設定して、グラフの外観をカスタマイズすることもできます。

埋め込みグラフを含む新しいレポートを作成するには、**C1Report ウィザード**を使用します。以下の手順を実行します。

1. **新しいレポートのデータソースを選択します。**
    1. **接続文字列**の省略記号ボタン(**〈...〉**)をクリックします。[**データ リンク プロパティ**]ダイアログボックスが表示されます。
    2. 「**プロバイダ**」タブを選択し、リストから「**Microsoft Jet 4.0 OLE DB Provider**」を選択します。
    3. 〈**次へ**〉をクリックします。
    4. 接続ページで、**〈...〉**ボタンをクリックして、**Nwind.mdb** データベースを参照します。これは、標準 Visual Studio Northwind データベースです。デフォルトで、データベースは C:\Users\<ユーザー名>\Documents\ComponentOne Samples\Common ディレクトリにインストールされます。
    5. 〈データソース〉タブを選択し、区分別売上高ビューを選択します。この手順は、次の画像に示されています。

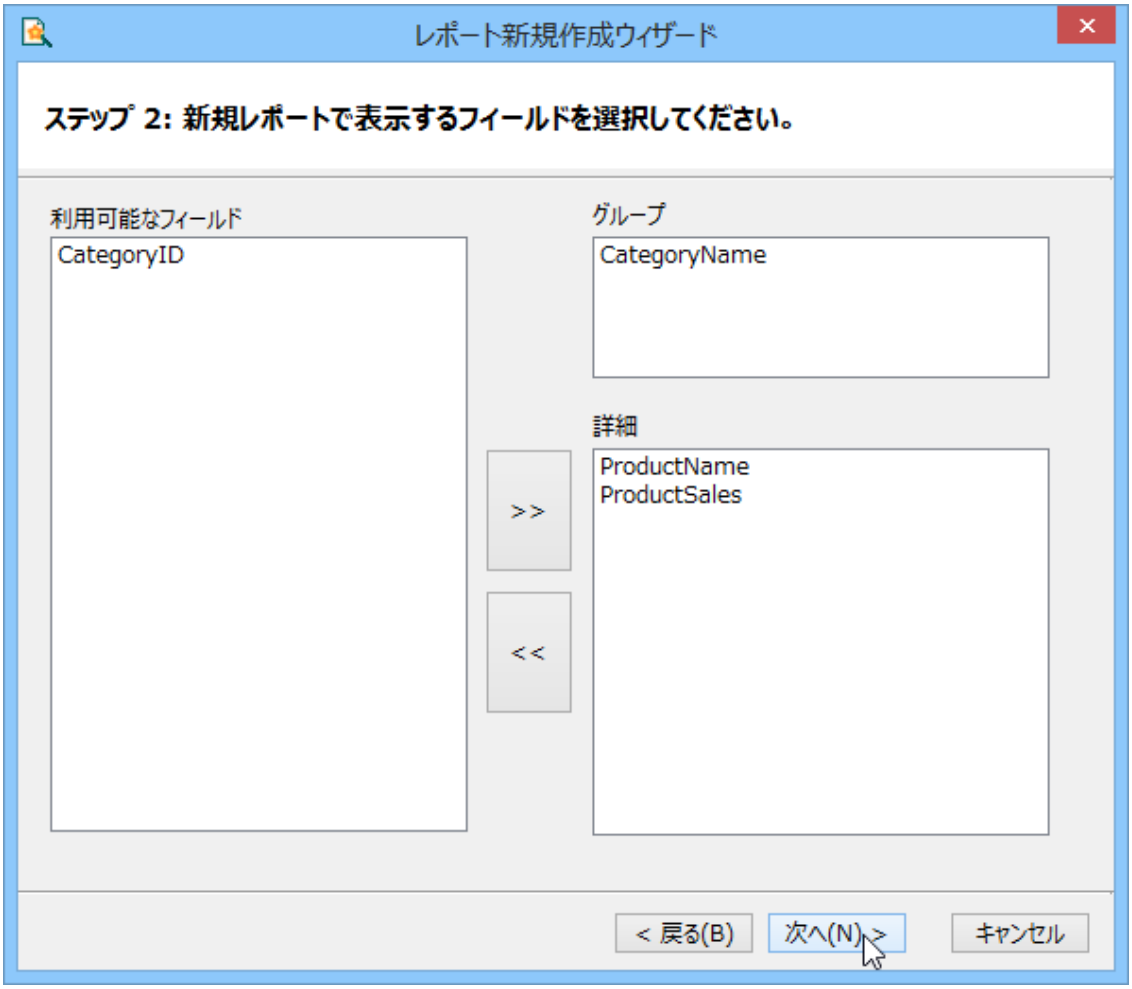

Copyright © GrapeCity, Inc. All rights reserved.




2. **表示するフィールドを選択します。**

   この例では、区分別にデータをグループ化し、レポートの［詳細］セクションに ProductName と ProductSales を表示します。**グループ**と**詳細**フィールドを追加するには、マウスポインタを使って、これらを左側の「**使用可能**」なリストから右側の「グループ」または「詳細」リストにドラッグします。


Copyright © GrapeCity, Inc. All rights reserved.




ウィザードが完了するまで、〈**次へ**〉をクリックして操作を続行します。ウィザードにより、レポートの初期バージョンが作成されます。

**レポートの[グループヘッダー]セクションにグラフを追加します。**

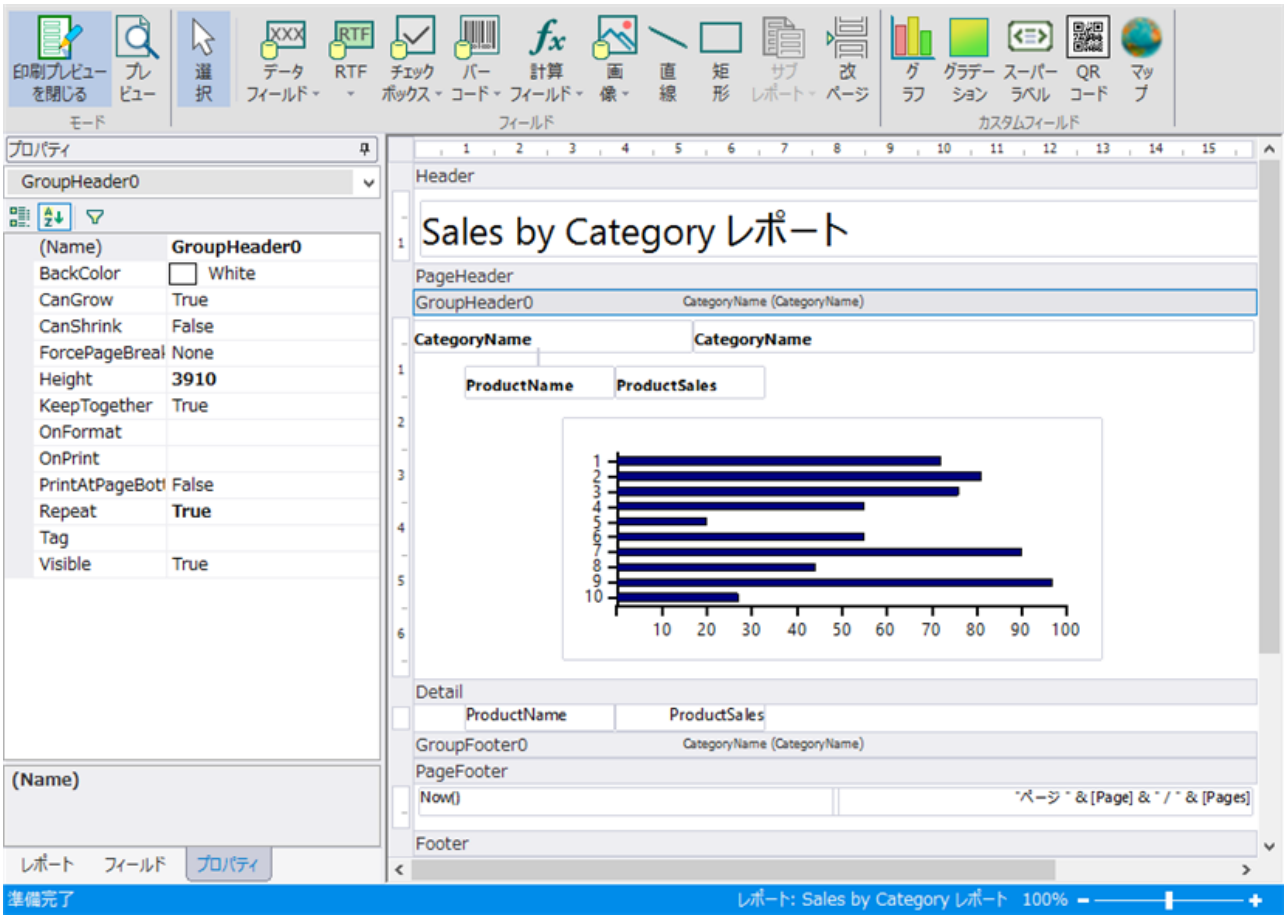
グラフは、グループの詳細を要約して示すために、レポートの[グループヘッダー]セクションに表示します。[グループヘッダー]セクションにグラフを追加するには、以下の手順を実行します。

1. 〈**デザイン**〉ボタンをクリックし、デザインモードに切り替えて、レポートの編集を開始します。
2. セクションの境界でドラッグ &ドロップ操作を実行して、[グループヘッダー]セクションを拡大します。
3. 次に、「**挿入**」タブの「**カスタムフィールド**」グループにある〈**グラフの追加**〉ボタン をクリックして、[グループヘッダー]セクションのレポートにフィールドを配置します。
4. グラフフィールドをクリックし、ドラッグして、グラフのサイズを変更します。.
5. プロパティウィンドウから、グラフ化する値(この場合、**ProductSales**)を含むフィールドの名前に **Chart.DataY** プロパティを設定します。
6. Chart.DataY プロパティは、複数の一連のグラフに指定できることに注意してください。フィールドまたは集計式を必要な数だけ追加し、セミコロンで区切ります。
7. また、各データポイントのラベル(この場合、**ProductName**)を含むフィールドの名前に **Chart.DataX** プロパティを設定します。
8. プロパティウィンドウから、**Chart.FormatY** プロパティを「#,###」に設定し、軸に沿った値を 1000 単位で区切られた値に設定します。

ここで、グラフコントロールにいくつかのサンプルデータが表示され、現在の設定によるプロパティの効果を確認できます(デザイン時に、実際のデータは使用できません)。**Chart.ChartType**、**Chart.DataColor**、**Chart.GridLines** など、いくつかのプロパティの値を変更し、試してみることができます。**Font**、**ForeColor** など、通常のフィールドプロパティを使用することもできます。

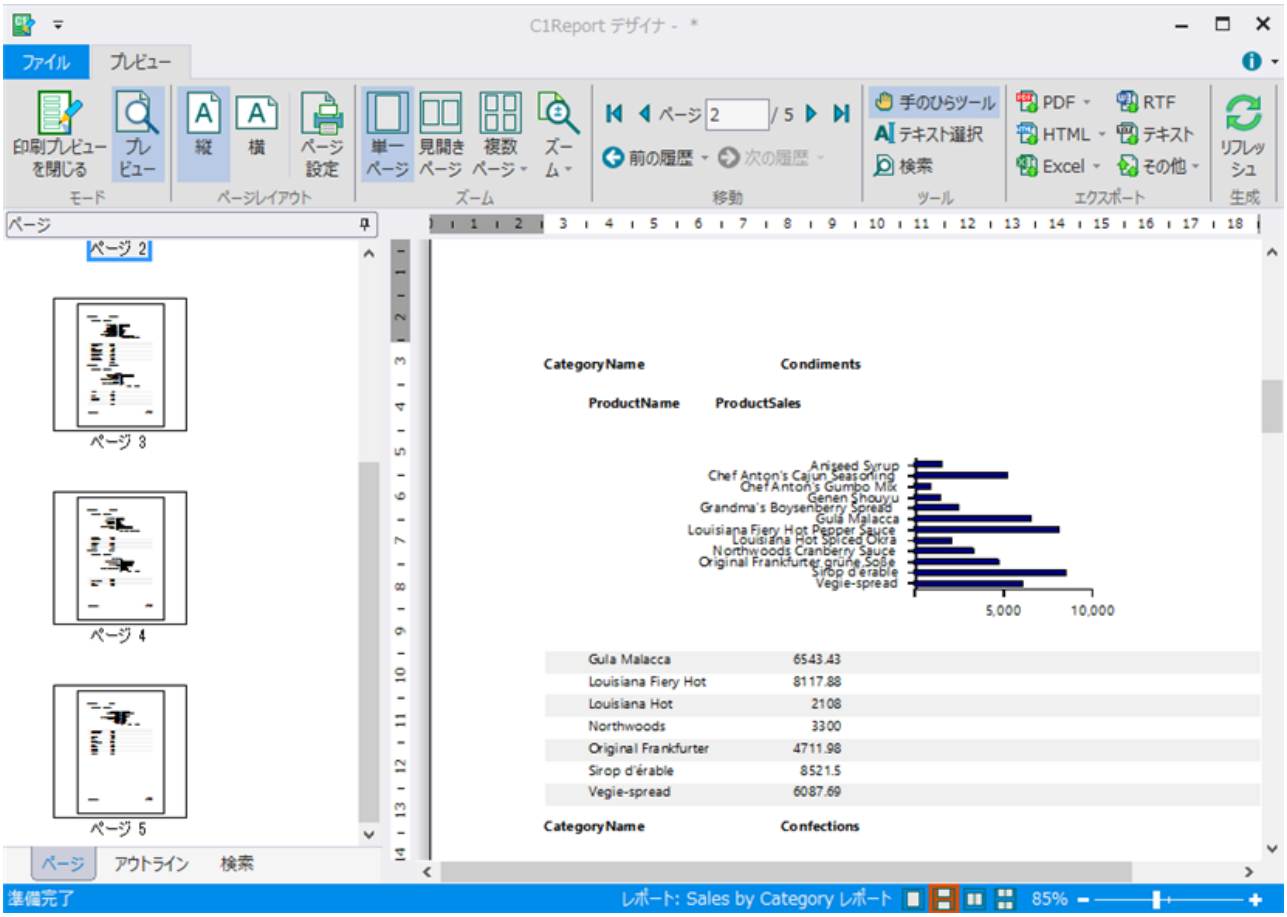
レポートは次のように表示されます。


Copyright © GrapeCity, Inc. All rights reserved.




〈**プレビュー**〉ボタンをクリックしてレポートを確認し、〈**次のページ**〉ボタンをクリックしてレポートをスクロールし、各グループのグラフフィールドを表示します。サンプルレポートは、次のような表示になります。


Copyright © GrapeCity, Inc. All rights reserved.




レポートフィールドは、そのレポート内の位置によって表示が異なることに注意してください。レポートフィールドは[グループヘッダー]セクション内にあるため、そのグループ内のデータのみが表示されます。グラフフィールドを[詳細]セクションに配置すると、レポート全体のすべてのデータが表示されます。各[詳細]セクションには1つのグラフが存在し、それらは同じように見えるので、この処理は有用ではありません。グラフ内に表示するデータを詳細に制御する必要がある場合は、グラフフィールドそのものに **DataSource** プロパティを使用できます。

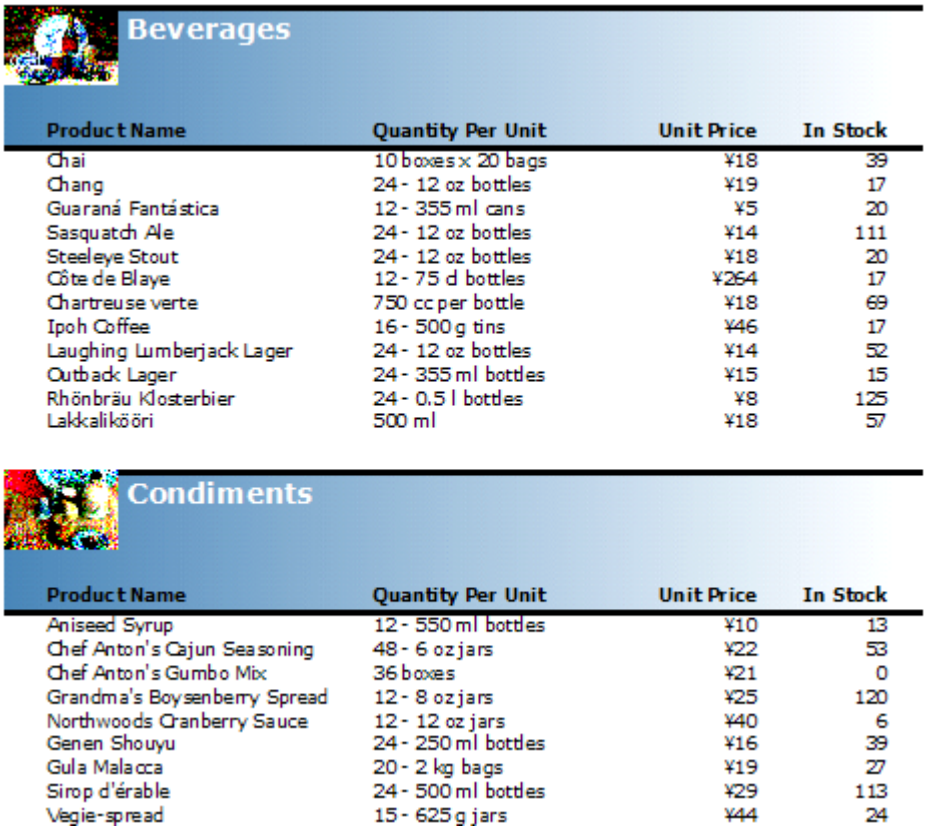
これで、レポートを保存し、それを Windows Forms と ASP.NET アプリケーションで使用できるようになりました。

## グラデーションフィールドの追加

**グラデーション**フフィールドは、グラフより簡単です。グラデーションは、主に他のフィールドを目立つようにする背景機能として使用されます。

以下の画面は、[グループヘッダー]セクションのラベル上にグラデーションフフィールドを使用するレポートを示しています。


Copyright © GrapeCity, Inc. All rights reserved.



**Beverages**

| Product Name | Quantity Per Unit | Unit Price | In Stock |
|---|---|---|---|
| Chai | 10 boxes x 20 bags | ¥18 | 39 |
| Chang | 24 - 12 oz bottles | ¥19 | 17 |
| Guaraná Fantástica | 12 - 355 ml cans | ¥5 | 20 |
| Sasquatch Ale | 24 - 12 oz bottles | ¥14 | 111 |
| Steeleye Stout | 24 - 12 oz bottles | ¥18 | 20 |
| Côte de Blaye | 12 - 75 d bottles | ¥264 | 17 |
| Chartreuse verte | 750 cc per bottle | ¥18 | 69 |
| Ipoh Coffee | 16 - 500 g tins | ¥46 | 17 |
| Laughing Lumberjack Lager | 24 - 12 oz bottles | ¥14 | 52 |
| Outback Lager | 24 - 355 ml bottles | ¥15 | 15 |
| Rhönbräu Klosterbier | 24 - 0.5 l bottles | ¥8 | 125 |
| Lakkalikööri | 500 ml | ¥18 | 57 |

**Condiments**

| Product Name | Quantity Per Unit | Unit Price | In Stock |
|---|---|---|---|
| Aniseed Syrup | 12 - 550 ml bottles | ¥10 | 13 |
| Chef Anton's Cajun Seasoning | 48 - 6 oz jars | ¥22 | 53 |
| Chef Anton's Gumbo Mix | 36 boxes | ¥21 | 0 |
| Grandma's Boysenberry Spread | 12 - 8 oz jars | ¥25 | 120 |
| Northwoods Cranberry Sauce | 12 - 12 oz jars | ¥40 | 6 |
| Genen Shouyu | 24 - 250 ml bottles | ¥16 | 39 |
| Gula Malacca | 20 - 2 kg bags | ¥19 | 27 |
| Sirop d'érable | 24 - 500 ml bottles | ¥29 | 113 |
| Vegie-spread | 15 - 625 g jars | ¥44 | 24 |

同じようなグラデーションフフィールドを作成するには、以下の手順を実行します。

1. デザイナのデザインモードで、「**デザイン**」タブの「**フィールド**」グループから、〈**グラデーションフィールドの追加**〉ボタン を選択します。
2. [グループヘッダー]セクションのラベル上にマウスカーソルを移動し(十字に変わります)、フィールドを希望するサイズにドラッグします。
3. フィールドがラベルの背後にあることを確認するには、グラデーションフィールドを右クリックして、[**最背面へ移動**]を選択します。
4. 次に、**Gradient.ColorFrom** プロパティと **Gradient.ColorTo** プロパティを、それぞれ **SteelBlue** と **White** に設定します。

**Gradient.Angle** プロパティを別の値(デフォルト値はゼロ)に設定して、グラデーションフィールドの角度を変更できることに注意してください。

グラデーションフィールドは、**CornerRadius** プロパティを使用して角に丸みを付けることで、さらにカスタマイズできるようにな



Copyright © GrapeCity, Inc. All rights reserved.

りました。

このようなグラデーションを作成するには、角の半径の X 値と Y 値の両方を設定する必要があります。**CornerRadius** プロパティには、角の X 値と Y 値を設定する次のオプションが提供されています。

- All：すべての角の X 値と Y 値を設定します。
- AllX：すべての角の X 値を設定します。
- AllY：すべての角の Y 値を設定します。
- BottomLeftX：左下の角の X 値を設定します。
- BottomLeftY：左下の角の Y 値を設定します。
- BottomRightX：右下の角の X 値を設定します。
- BottomRightY：右下の角の Y 値を設定します。
- TopLeftX：左上の角の X 値を設定します。
- TopLeftY：左上の角の Y 値を設定します。
- TopRightX：右上の角の X 値を設定します。
- TopRightY：右上の角の Y 値を設定します。

プロパティペインの **ShapeType** プロパティを設定することでグラデーションをさまざまな図形にすることができます。設定できるオプションは次のとおりです。

- Line
- IsoscelesTriangle
- RightTriangle
- Rectangle
- Ellipse
- Arc
- Pie

グラデーションフィールドの 4 つの角の丸みの半径には、いくつかの制限があります。

1. 半径 X の範囲は、0 からフィールドの幅までです。
2. 半径 Y の範囲は、0 からフィールドの高さまでです。


Copyright © GrapeCity, Inc. All rights reserved.

3. 半径 X については、TopLeft + TopRight/BottomLeft + BottomRight の値がフィールドの幅より大きくなってはなりません。
4. 半径 Y については、TopLeft + BottomLeft/TopRight + BottomRight の値がフィールドの高さより大きくなってはなりません。
5. 規則 3 および 4 に違反した設定を行った場合は、TopRight または BottomLeft の半径が小さくなり、TopLeft と BottomRight の設定は維持されます。たとえば、フィールドのサイズが (200, 100) の場合に、TopLeft を (150, 80)、BottomRight を (150, 80)、TopRight を (100, 50)、BottomLeft を (100, 50) にすると、TopRight が (50, 20) に、BottomLeft が (50, 20) に変更されます。

# フィールドの選択、移動、およびコピー

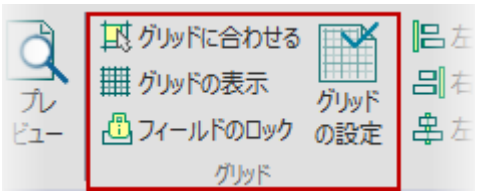
マウスを使って、通常どおり、**C1Report デザイナ**アプリケーション内のフィールドを選択できます。

- フィールドをクリックして、それを選択します。
- [Shift]キーを押しながらフィールドをクリックして、選択状態を切り替えます。
- [Ctrl]キーを押しながらドラッグして、選択したフィールドのコピーを作成します。
- 空領域をクリックし、マウスポインタをドラッグして、複数のフィールドを選択します。
- マウスポインタを使用し、フィールドの角をドラッグして、フィールドのサイズを変更します。
- フィールドの右または下端をダブルクリックして、フィールドのサイズを自動調整します。

レポートの垂直または水平領域を交差するフィールドを選択するには、デザイナの端に沿ったルーラー上でマウスをクリックして、ドラッグします。フィールドが小さいか、またはお互いに接近している場合、名前で選択した方が容易な場合があります。プロパティウィンドウ上のドロップダウンリストからフィールドとセクションを選択し、それらを選択できます。

## グリッドの表示

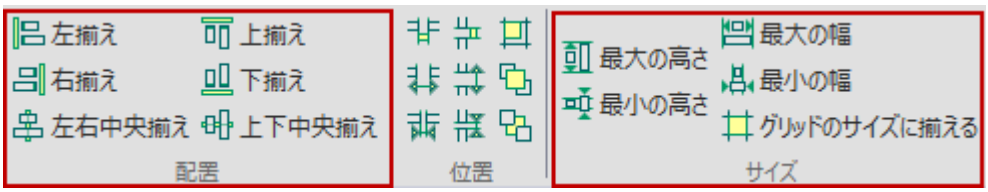
「レイアウト」タブの「**グリッド**」グループにある〈**グリッドに合わせる**〉と〈**グリッドの表示**〉ボタンは、離散した位置にある位置の制御に役立つグリッドを提供します。グリッドが ON の場合、フィールドを作成または移動するときに、フィールドの左上端がグリッドに配置されます。〈**ファイル**〉メニューから[**オプション**]を選択して、グリッドの単位(ヤード・ポンド法またはメートル法)を変更できます。




## フィールドのロック

フィールドを希望する位置に配置した後に、不注意によってマウスまたはキーボードで位置が移動されないように、これらをロックできます。〈**フィールドのロック**〉ボタンを使用して、フィールドをロックおよびロック解除できます。

## 書式フィールド




複数のフィールドを選択した場合、「**レイアウト**」タブの「**配置**」、「**位置**」、および「**サイズ**」グループ上のボタンを使用できます。これらのボタンのいずれかをクリックした場合、選択した最後のフィールドが参照として使用され、設定が選択した残りのフィー


Copyright © GrapeCity, Inc. All rights reserved.

ルドに適用されます。

## スタイルの適用

〈**スタイルの統一**〉ボタンは、参照フィールドのスタイルを選択した項目全体に適用します。フィールドのスタイルには、すべてのフォント、色、線、配置、および余白プロパティが含まれています。プロパティウィンドウを使用して、個別のプロパティの値を選択した項目全体に設定できます。

## 重複するフィールドの順序の指定

いくつかのフィールドが重複する場合、「位置」グループの〈**最前面へ移動**〉／〈**最背面へ移動**〉ボタンを使用して、それらの z-order を制御できます。これにより、他のフィールドの前（または背後）に表示するフィールドを指定します。

キーボードの使用によるフィールドの移動

**C1Report デザイナ**アプリケーションでは、キーボードを使用してフィールドを選択および移動することもできます。

- [Tab]キーを使用して、次のフィールドを選択します。
- [Shift]+[Tab]キーを使用して、前のフィールドを選択します。
- 矢印キーを使用して、選択した項目を一度に1ピクセルずつ（または[Shift]キー + 矢印を5ピクセルずつ）移動します。
- [Delete]キーを使用して、選択したフィールドを削除します。
- 1つのフィールドを選択した場合、そのフィールドに入力して **Text** プロパティを設定できます。

# プロパティの変更

オブジェクトを選択すると、プロパティウィンドウを使用してそのプロパティを編集できます。

複数のフィールドが選択されている場合、プロパティウィンドウには、すべてのフィールドに共通のプロパティの値が表示され、その他のプロパティは空欄になります。フィールドを選択せずにセクション（または、セクションの上部のバー）をクリックすると、**セクション**のプロパティが表示されます。背景の灰色の領域をクリックすると、**レポート**のプロパティが表示されます。

この操作を確認するために、ヘッダセクションのラベルをダブルクリックして、**Font** プロパティと **ForeColor** プロパティを変更してみます。**Left**、**Top**、**Width**、**Height** の各プロパティに新しい値を入力することによって、フィールドの位置やサイズを変更することもできます。

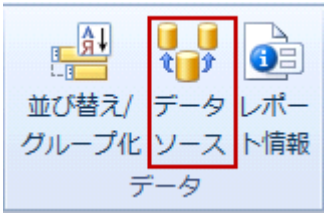
プロパティウィンドウに表示される測定値はすべて Twips 単位（**C1Report** が使用するネイティブの単位）ですが、他の単位で値を入力することもできます。入力した値は、自動的に Twips 単位に変換されます。たとえば、フィールドのHeight プロパティを **0.5in** に設定した場合、プロパティウィンドウではそれが *720Twips* に変換されます。

# データソースの変更

データソースは、**ConnectionString**、**RecordSource**、**Filter** の各プロパティを使用して定義します。これらは、通常の**レポート**プロパティであり、次のいずれかの方法を使用して設定できます。


Copyright © GrapeCity, Inc. All rights reserved.

- プロパティウィンドウで、**DataSource** プロパティの横にある省略符ボタンを選択します。背景の灰色の領域をクリックすると、**レポートの**プロパティが表示されます。
- [**データソース**]ボタンをクリックすると、[**データソースの選択**]ダイアログボックスが表示されます。ここで、ConnectionString プロパティと RecordSource プロパティを直接設定できます。




## サブレポートを使用したマスター/詳細レポートの作成

サブレポートは、他のレポート（メインレポート）内のフィールドに含まれる通常のレポートです。通常、サブレポートは、マスター／詳細レポートシナリオでのメインレポートの現在の値に基づく詳細情報を表示するように設計されています。

以下の例では、メインレポートに商品区分が含まれ、[詳細]セクション内のサブレポートに現在の商品区分の商品明細が含まれています。

***Beverages***
Soft drinks, coffees,

| Product Name | Quantity per unit | Unit price | Units in stock | Units on order |
|---|---|---|---|---|
| Chai | 10 boxes x 20 bags | 18 | 39 | 0 |
| Chang | 24 - 12 oz bottles | 19 | 17 | 40 |
| Aniseed Syrup | 12 - 550 ml bottles | 10 | 13 | 70 |
| Chef Anton's Cajun | 48 - 6 oz jars | 22 | 53 | 0 |
| Chef Anton's | 36 boxes | 21.35 | 0 | 0 |
| Grandma's | 12 - 8 oz jars | 25 | 120 | 0 |
| Uncle Bob's | 12 - 1 lb pkgs. | 30 | 15 | 0 |
| Northwoods | 12 - 12 oz jars | 40 | 6 | 0 |
| Mishi Kobe Niku | 18 - 500 g pkgs. | 97 | 29 | 0 |
| Ikura | 12 - 200 ml jars | 31 | 31 | 0 |
| Queso Cabrales | 1 kg pkg. | 21 | 22 | 30 |
| Queso Manchego | 10 - 500 g pkgs. | 38 | 86 | 0 |
| Konbu | 2 kg box | 6 | 24 | 0 |
| Tofu | 40 - 100 g pkgs. | 23.25 | 35 | 0 |

***Condiments***
Sweet and savory

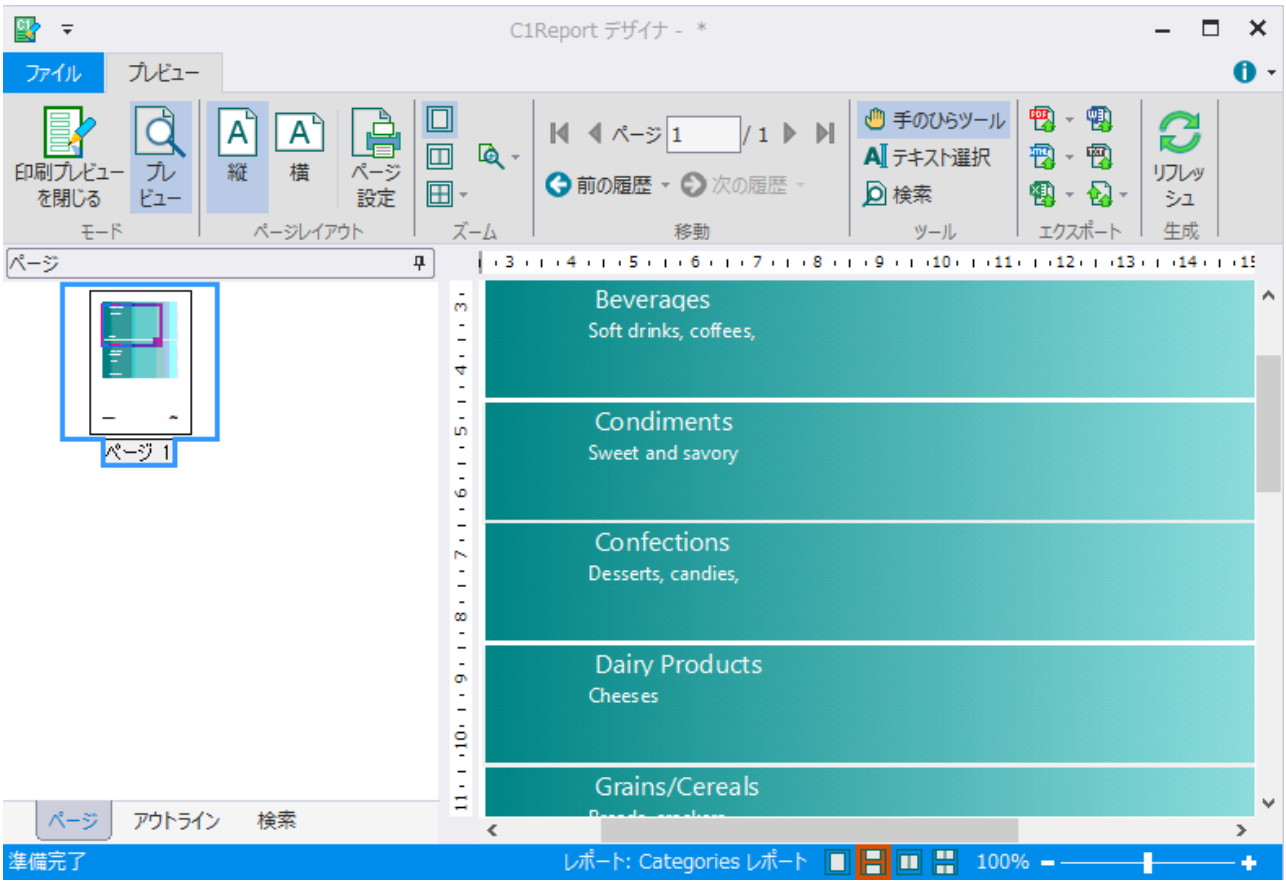
| Product Name | Quantity per unit | Unit price | Units in stock | Units on order |
|---|---|---|---|---|
| Chai | 10 boxes x 20 bags | 18 | 39 | 0 |
| Chang | 24 - 12 oz bottles | 19 | 17 | 40 |
| Aniseed Syrup | 12 - 550 ml bottles | 10 | 13 | 70 |
| Chef Anton's Cajun | 48 - 6 oz jars | 22 | 53 | 0 |
| Chef Anton's | 36 boxes | 21.35 | 0 | 0 |
| Grandma's | 12 - 8 oz jars | 25 | 120 | 0 |
| Uncle Bob's | 12 - 1 lb pkgs. | 30 | 15 | 0 |
| Northwoods | 12 - 12 oz jars | 40 | 6 | 0 |

**Category**および**Products** テーブルに基づいてマスター／詳細レポートを作成するには、Categories レポート（マスタービュー）と Products レポート（詳細ビュー）を作成する必要があります。


Copyright © GrapeCity, Inc. All rights reserved.

### 手順1：マスターレポートの作成

1. **C1Report ウィザード**を使用して、基本的なレポート定義ファイルを作成します。
    1. Northwind データベースから**Category**テーブルを選択します（**Nwind.mdb** は、ComponentOne Samples\Common フォルダーにあります）。
    2. **CategoryName** と **Description** フィールドをレポートに取り込みます。
2. **C1Report デザイナ**アプリケーションで、〈**デザイン**〉ボタンをクリックしてレポートの編集を開始します。
3. [ページヘッダー]セクションと[ヘッダー]セクションの **Visible** プロパティを **False** に設定します。
4. [詳細]セクションで、[**DescriptionCtl**]を選択し、それを直接[**CategoryNameCtl**]の下に移動します。
5. プロパティウィンドウを使用して、外観の設定（Font および ForeColor）を変更します。この例では、[**グラデーション**]フィールドが[詳細]セクションに追加されています。[**グラデーション**]フィールドについての詳細は、「グラデーションフィールドの追加」を参照してください。
6. 〈**プレビュー**〉ボタンを選択すると、Categories レポートは次の画面のように表示されます。




### 手順2：詳細レポートの作成

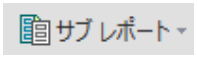
1. **C1Report デザイナ**アプリケーションで、〈**新しいレポート**〉ボタンをクリックし、**C1Report ウィザード**を使用して基本的なレポート定義ファイルの作成を作成します。
    1. Northwind データベースから、**Products**テーブルを選択します。
    2. レポートに以下のフィールドを取り込みます。**ProductName**、**QuantityPerUnit**、**UnitPrice**、UnitsInStock、および **UnitsOnOrder**。
2. レポートデザイナで、〈**デザイン**〉ボタンをクリックしてレポートの編集を開始します。
    1. [ページヘッダー]セクションと[ヘッダー]セクションの **Visible** プロパティを **False** に設定します。
    2. [詳細]セクションで、見出しラベルの揃うようにコントロールを配置します。プロパティウィンドウを使用して、外観の設定を変更します。

### 手順3：サブレポートフィールドの作成

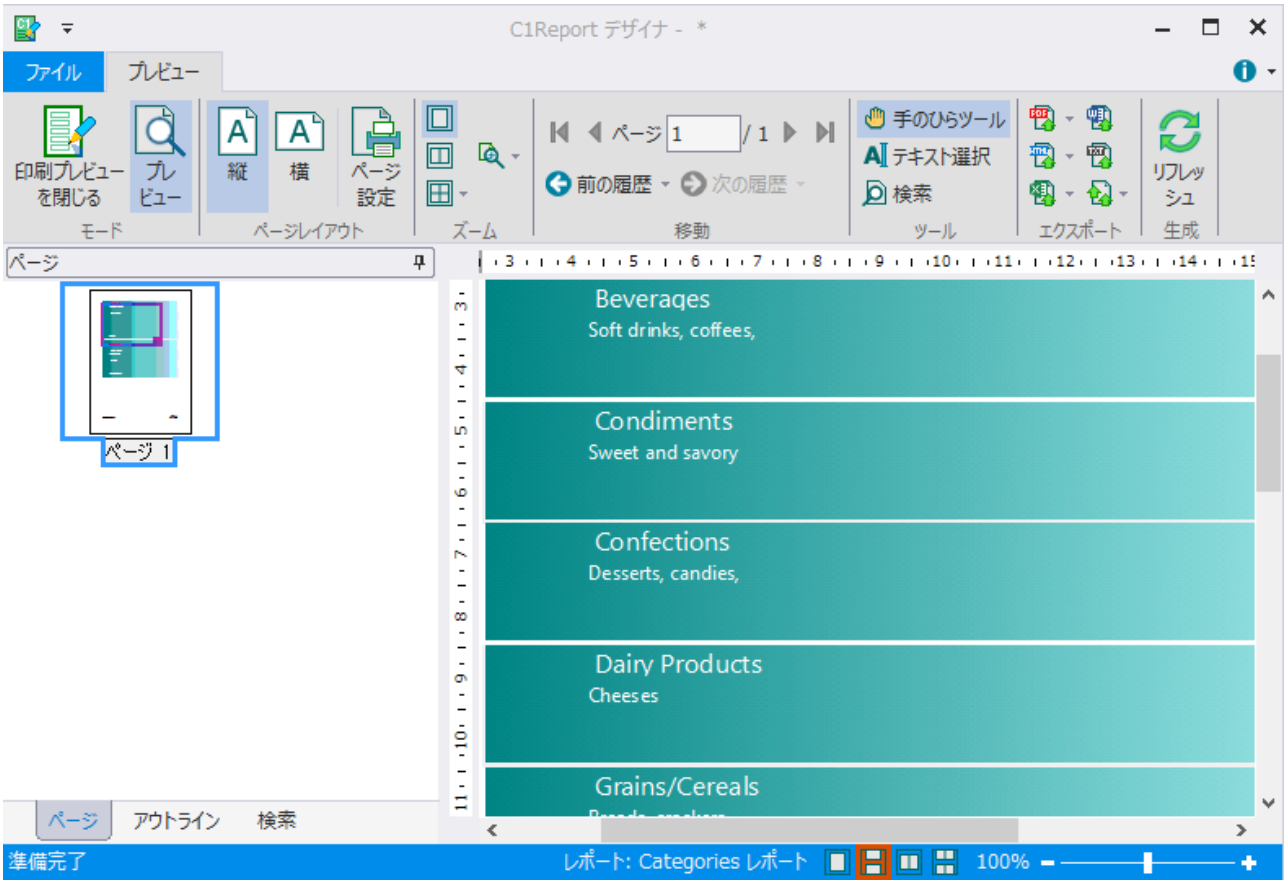
Tここでは、**C1Report デザイナ**アプリケーションには、**Categories レポート**と**Products レポート**の2つの別個のレポートが存在します。次の手順では、ブレポートを作成します。


Copyright © GrapeCity, Inc. All rights reserved.

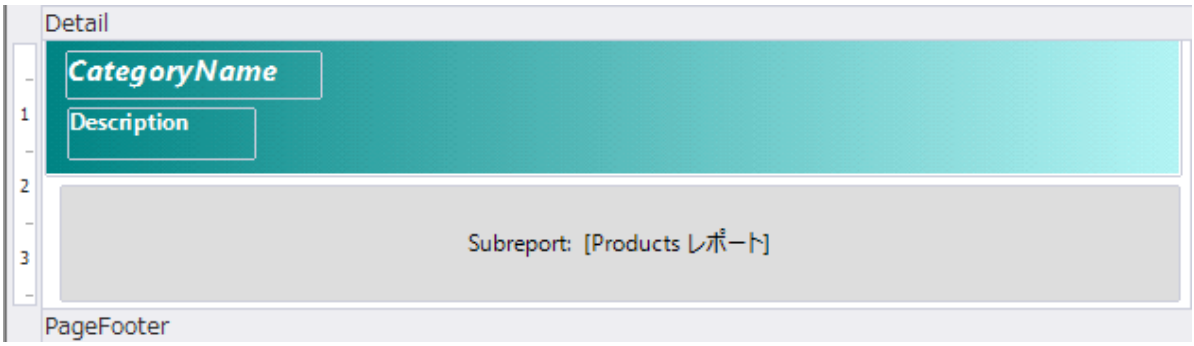
1. デザイナの[レポート]リストから、**Categories レポート**(マスターレポート)を選択します。
2. デザインモードで、「**挿入**」タブの「**フィールド**」グループにある〈**サブレポート**〉




ボタンをクリックし、ドロップダウンメニューから**Products レポート**を選択します。




3. レポートの[詳細]セクションで、マウスポインタをクリックおよびドラッグして、サブレポートのフィールドを作成します。



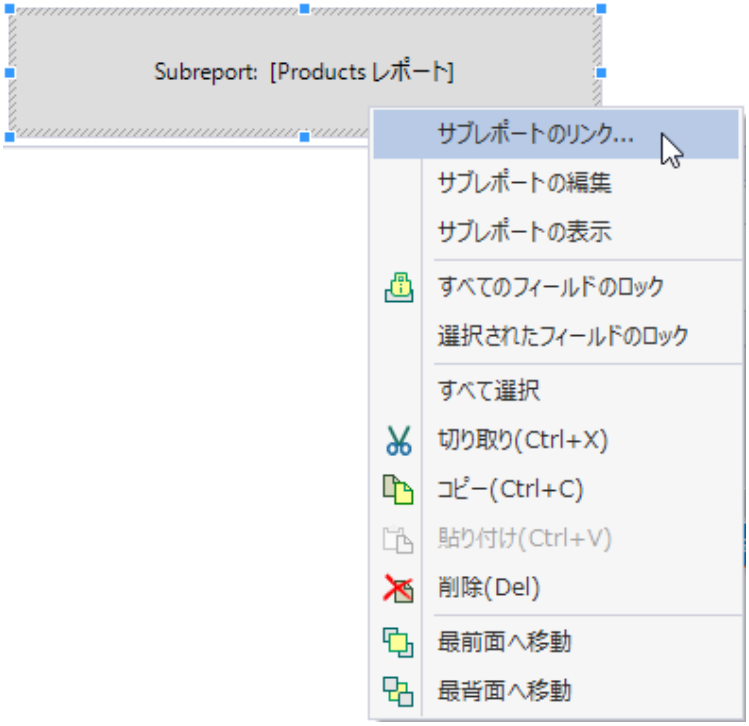

### 手順4:マスターレポートへのサブレポートのリンク

マスター／詳細レポートの関係は、サブレポートフィールドの Text プロパティによって制御されます。このプロパティには、サブレポートのデータソースに適用できるフィルタ条件を評価する式が含まれています。

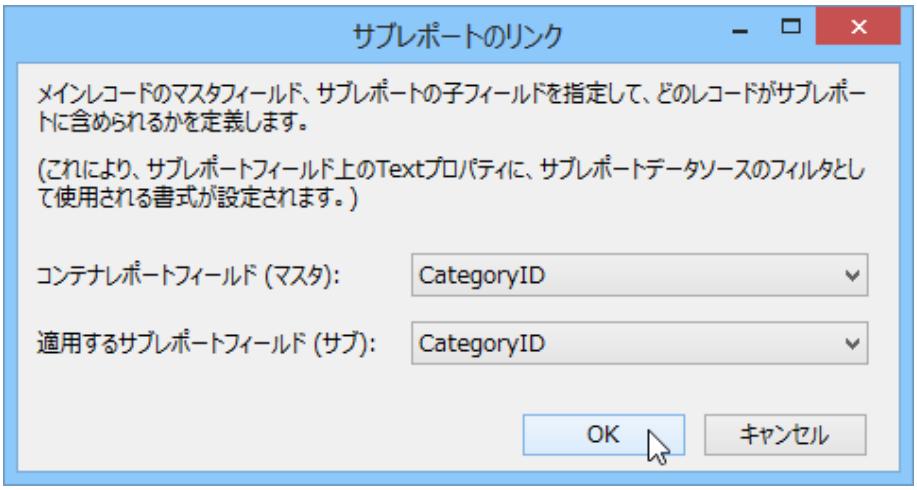
レポートデザイナでは、この数式を自動的にビルドできます。以下の手順を実行します。

1. サブレポートフィールドを右クリックし、メニューから[**サブレポートのリンク**]を選択します。


Copyright © GrapeCity, Inc. All rights reserved.




2. リンクするフィールドを選択できるダイアログボックスが表示されます。




3. 選択を行い、〈**OK**〉をクリックすると、レポートデザイナは、リンク式をビルドし、それをサブレポートフィールドの **Text** プロパティに割り当てます。この場合、式は次のとおりになります。

```
"[CategoryID] = '" & [CategoryID] & "'"
```

または、以下の手順を実行して、サブレポートをマスター レポートにリンクさせることもできます。

1. プロパティウィンドウで、サブレポートフィールドの Text プロパティをクリックし、ドロップダウンリストから[**スクリプトエディタ**]を選択します。
2. VBScript エディタで、次の式を入力します。

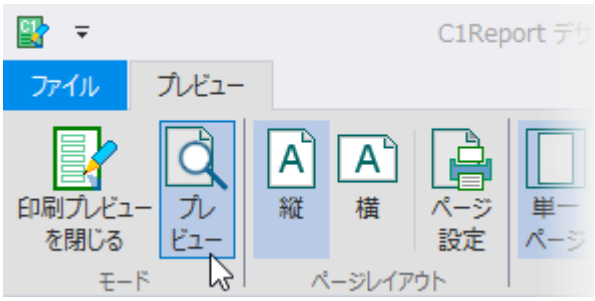
```
"[CategoryID] = '" & [CategoryID] & "'"
```

3. 〈**OK**〉をクリックし、VBScript エディタを閉じて、式をビルドします。

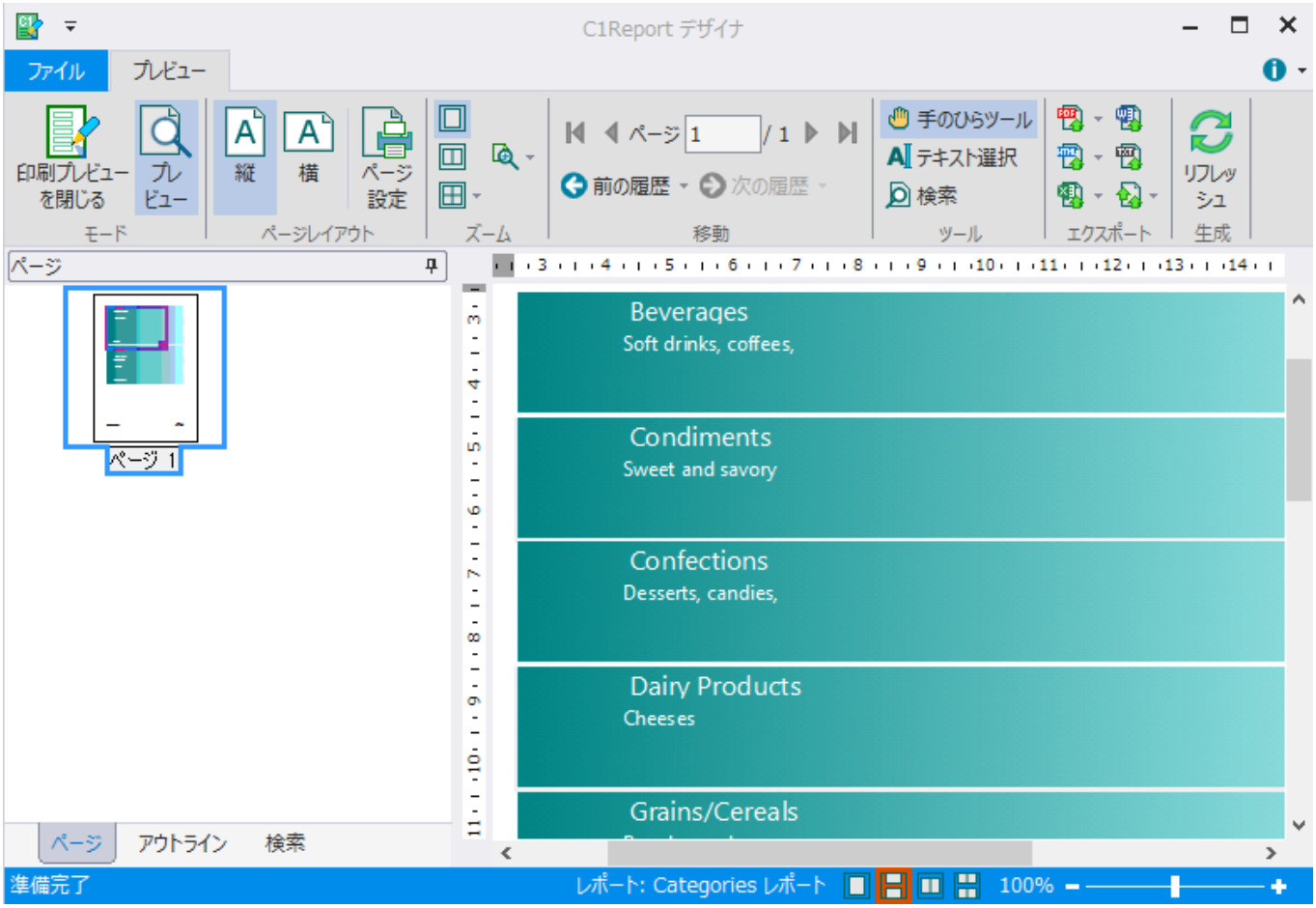
# レポートのプレビューと印刷

レポートをプレビューするには、デザイナウィンドウの左ペイン上の[レポート]リストから表示するレポートを選択し、各「リボン」タブで表示される〈**プレビュー**〉ボタンをクリックします。


Copyright © GrapeCity, Inc. All rights reserved.



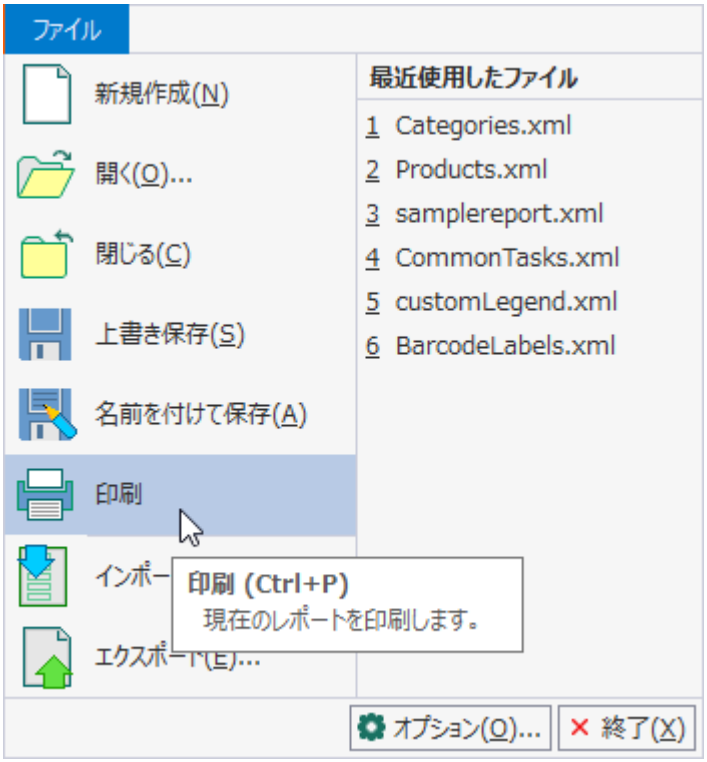

以下のスクリーンショットに示すように、レポートが右ペインに表示されます。




メインウィンドウには、ドキュメントのページを移動できるボタンを含むプレビューナビゲーションツールバーが表示されます。

この時点で、〈**印刷**〉ボタンをクリックして、レポートを印刷できます。


Copyright © GrapeCity, Inc. All rights reserved.




## レポートのエクスポートと配布

レポートを印刷するのではなく、ファイルにエクスポートして、クライアントや同僚に電子メールで配布する場合もあります。デザイナは、次のエクスポート形式をサポートしています。

| フォーマット | 説明 |
|---|---|
| ページ化 HMTL (*.htm) | レポートの1ページごとに HTML ファイルを1つ作成します。HTML ページには、レポート内を移動するためのリンクが含ます。 |
| ドリルダウン HTML (*.htm) | クリックしてセクションを折りたたみ／展開することが可能な単一の HTML ファイルを作成します。 |
| プレーン HMTL(*.htm) | 単一のプレーンな HTML ファイルを作成します。 |
| PDF（システムフォント）(*.pdf) | フォントを埋め込まない PDF ファイルを作成します。 |
| PDF（埋め込みフォント）(*.pdf) | フォントを埋め込んだ PDF ファイルを作成します。 |
| RTF (*.rtf) | 多くのワープロソフトで表示可能な RTF ファイルを作成します。 |
| RTF with fixed positioning (*.rtf) | 多くのワープロソフト(たとえば—＞Microsoft Word, WordPadなど)で対応されている定位置のRTF ファイルを作成します。 |
| Microsoft Excel 97 (*.xls) | Microsoft Excel で開くことができる XLS ファイルを作成します。 |
| Microsoft Excel 2007/2010 Open XML (*.xlsx) | Microsoft Excel 2007 以降で開くことができる XLS ファイルを作成します。 |
| TIFF (*.tif) | マルチページ TIFF ファイルを作成します。 |
| テキスト(*.txt) | プレーンなテキストファイルを作成します。 |
| 単一ページテキスト（*.txt） | 単一のプレーンテキストファイルを作成します。 |
| 圧縮メタファイル(*.txt) | 圧縮したメタファイルテキストファイルを作成します。 |


Copyright © GrapeCity, Inc. All rights reserved.

エクスポートファイルを作成するには、[**ファイル**｜**エクスポート**]メニューを選択し、**ファイルの保存**ダイアログボックスを使用して、作成するファイルの種類、名前、および場所を選択します。

> **注意**:[改ページを保持]オプションをオンにしてドキュメントを RTF または DOCX 形式にエクスポートすると、テキストはテキストボックスに配置され、結果のドキュメントのテキストを再フローする機能を制限できます。

# レポート定義ファイルの管理

レポート定義ファイルには、複数のレポートを入れることができますが、レポートをファイルからファイルに移動することもできます。

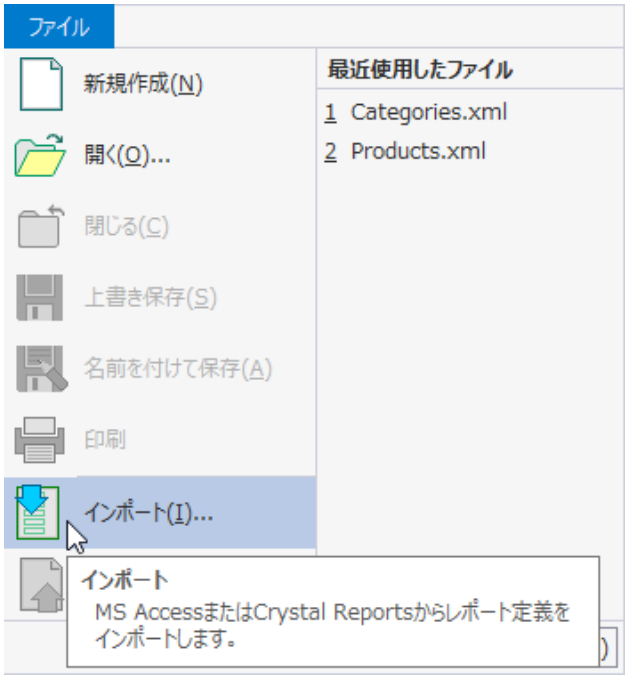
レポートをあるファイルから別のファイルに移動するには、デザイナを2つ開き、レポートを一方のデザイナからもう一方のデザイナにドラッグします。[Ctrl]キーを押したままドラッグすると、レポートをコピーできます。[Ctrl]キーを押さずにドラッグすると、レポートを移動できます。

同じファイルにレポートをコピーすることもできます。これにより、同じレポートのコピーが作成されます。既存のレポートに似たレポートを新しく設計する場合は、このコピーから開始すると便利です。

レポート定義ファイルは XML 形式で保存されるため、テキストエディタを使って編集および管理することもできます。

## Microsoft Access レポートのインポート

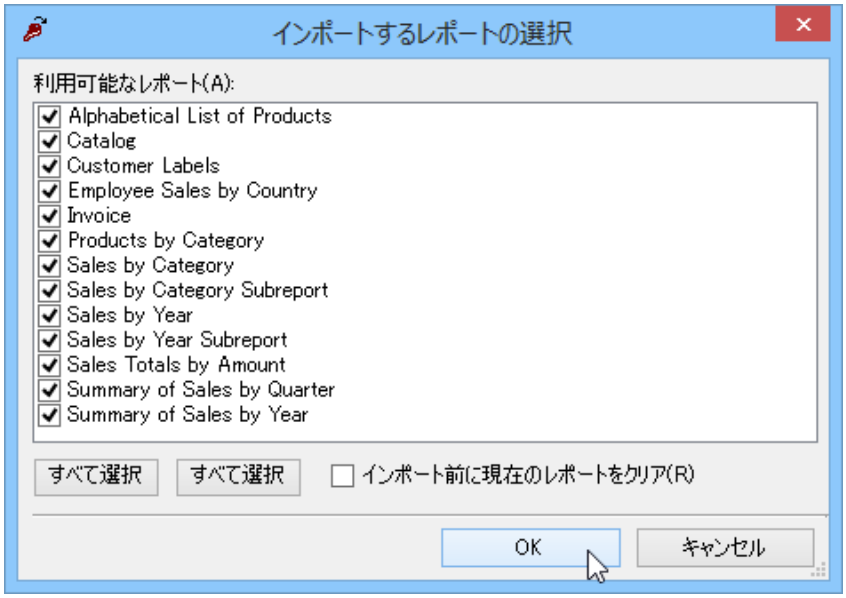
**C1Report デザイナ**の最も強力な機能の1つは、Microsoft Access を使って作成されたレポートをインポートできることです。*.mdb ファイルのほかにも、Access 2007 の *.accdb ファイルにも対応しています。**この機能を使用するには、Microsoft Access がコンピュータにインストールされている必要があります**。ただし、レポートをデザイナにインポートしてしまえば、Microsoft Access は必要なくなります。




Access ファイルからレポートをインポートするには、[**インポート**]ボタンをクリックするか、[**ファイル**｜**インポート**]メニューを選択します。ダイアログボックスが表示され、インポートするファイルの名前の入力を求められます。

デザイナは自動的にファイルをスキャンし、インポートするファイルを選択できるダイアログを表示します。


Copyright © GrapeCity, Inc. All rights reserved.




ダイアログでは、インポート処理を行う前に既存のすべてのレポート定義をクリアするかどうか指定できます。

インポート処理が正しく動作すると、ソースレポートのほとんどの要素が処理されます。ただし、次に挙げるいくつかの例外があります。

- **イベントハンドラコード**

  Access は、VBA、マクロ、およびフォームを使ってレポートを動的に書式設定できます。**C1Report**でも同様の処理が可能ですが、C1Report では VBScript だけが使用されるため、レポートのコードはすべて手作業で変換する必要があります。

- **フォーム指向のフィールドタイプ**

  Access レポートには、デザイナのインポートプロシージャによって処理されないフィールドが含まれていることがあります。サポートしないフィールドタイプは、Chart、CommandButton、ToggleButton、OptionButton、OptionGroup、ComboBox、ListBox、TabCtl、および CustomControl です。

- **VBScript の予約語を使用するレポート**

  Access は VBScript を使用しないため、レポートオブジェクトやデータセットフィールド名の識別子として VBScript の予約語を使用するレポートが設計されている可能性があります。その場合は、VBScript エンジンが式を評価するときに問題が発生し、レポートが正しく出力されない原因になります。

  識別子として使用できない予約語には、**Date**、**Day**、**Hour**、**Length**、**Minute**、**Month**、**Second**、**Time**、**TimeValue**、**Value**、**Weekday**、**Year** などがあります。完全なリストについては、VBScript のリファレンスを参照してください。

- **日付を四半期順(または、曜日、月など)でソートするレポート**

  **C1Report** は、ADO.NET データセットのSort プロパティを使ってグループをソートします。このプロパティは、フィールドの値に基づいてデータセットをソートし、式は使用しません(任意の式に基づいてグループ化はできますが、ソートはできません)。グループを四半期順にソートする Access レポートをインポートすると、ソートは日付順に行われるようになります。これを解決する方法は2つあります。1つは、ソートの基準にする式の値を含むフィールドを作成する方法、もう1つは、データセットを作成する SQL 文を変更し、四半期順にソートを実行する方法です。

上のような制限を受けるレポートは比較的少ないですが、レポートをインポートしたら、すべてのレポートをプレビューし、正しく動作するかどうかを確認してください。

### Nwind.mdb ファイルをインポートする

デザイナが実際に行う処理の一例として、製品に付属の NWind.mdb ファイルをインポートしてみます。このファイルには、次に示す13のレポートが含まれています。そこで、これらのレポートに対する処理について説明します。C1Report に付属する NWind.XML ファイルには、以下で説明する変更がすべて行われています。

1. **Alphabetical List of Products(製品のアルファベット順リスト)**

   処理は不要です。

2. **Catalog(カタログ)**

   処理は不要です。


Copyright © GrapeCity, Inc. All rights reserved.

3. **Customer Labels（顧客ラベル）**

   処理は不要です。

4. **Employee Sales by Country（地域別従業員売上高）**

   このレポートには、手作業で変換する必要があるコードが含まれます。次のコードを Group 1 のヘッダセクションの**OnPrint** プロパティに割り当てます。

   **Visual Basic コードの書き方**

   Visual Basic

```
If SalespersonTotal > 5000 Then
  ExceededGoalLabel.Visible = True
  SalespersonLine.Visible = True
Else
  ExceededGoalLabel.Visible = False
  SalespersonLine.Visible = False
End If
```

   **C# コードの書き方**

   C#

```
if (SalespersonTotal > 5000)
{
  ExceededGoalLabel.Visible = true;
  SalespersonLine.Visible = true;
} else {
  ExceededGoalLabel.Visible = false;
  SalespersonLine.Visible = false;
}
```

5. **Invoice（請求書）**

   処理は不要です。

6. **Products by Category（カテゴリ別製品）**

   処理は不要です。

7. **Sales by Category（カテゴリ別売上高）**

   このレポートには、インポートされないグラフコントロールが含まれます。グラフをレポートに追加するには、アンバウンドピクチャフィールドを使用します。イベントハンドラを記述してグラフを作成し、それを画像としてアンバウンドピクチャフィールドに割り当てます。

8. **Sales by Category Subreport（カテゴリ別売上高サブレポート）**

   処理は不要です。

9. **Sales by Year（年度別売上高）**

   このレポートには、手作業で変換する必要があるコードおよび Form オブジェクトへの参照が含まれます。Form オブジェクトを置き換えるには、**RecordSource** プロパティを編集して [Show Details] パラメータを追加します。

   **Visual Basic コードの書き方**

   Visual Basic

```
PARAMETERS (Beginning Date) DateTime 1/1/1994,
  (Ending Date) DateTime 1/1/2001,
  (Show Details) Boolean False; ...
```

   **C# コードの書き方**

   C#

```
PARAMETERS [Beginning Date] DateTime 1/1/1994,
```


Copyright © GrapeCity, Inc. All rights reserved.

```
  [Ending Date] DateTime 1/1/2001,
  [Show Details] Boolean False; ...
```

レポートの**OnOpen** プロパティで新しいパラメータを使用します。

**Visual Basic コードの書き方**

Visual Basic

```
Dim script As String = _
  "bDetails = [Show Details]" & vbCrLf & _
  "Detail.Visible = bDetails" & vbCrLf & _
  "[Group 0 Footer].Visible = bDetails" & vbCrLf & _
  "DetailsLabel.Visible = bDetails" & vbCrLf & _
  "LineNumberLabel2.Visible = bDetails" & vbCrLf & _
  "Line15.Visible = bDetails" & vbCrLf & _
  "SalesLabel2.Visible = bDetails" & vbCrLf & _
  "OrdersShippedLabel2.Visible = bDetails" & vbCrLf & _
  "ShippedDateLabel2.Visible = bDetails" & vbCrLf & _
  "Line10.Visible = bDetails"
c1r.Sections.Detail.OnPrint = script
```

**C# コードの書き方**

C#

```
string script = "bDetails = [Show Details]" +
  "Detail.Visible = bDetails\r\n" +
  "[Group 0 Footer].Visible = bDetails\r\n" +
  "DetailsLabel.Visible = bDetails\r\n" +
  "LineNumberLabel2.Visible = bDetails\r\n" +
  "Line15.Visible = bDetails\r\n" +
  "SalesLabel2.Visible = bDetails\r\n" +
  "OrdersShippedLabel2.Visible = bDetails\r\n" +
  "ShippedDateLabel2.Visible = bDetails\r\n" +
  "Line10.Visible = bDetails";
c1r.Sections.Detail.OnPrint = script;
```

最後に、2行のコードを書き換える必要があります。

**Visual Basic コードの書き方**

Visual Basic

```
Sections ("Detail").OnPrint = _
  "PageHeader.Visible = True"
Sections("Group 0 Footer).OnPrint = _
  "PageHeader.Visible = False"
```

**C# コードの書き方**

C#

```
Sections ("Detail").OnPrint =
  "PageHeader.Visible = true";
Sections("Group 0 Footer).OnPrint =
  "PageHeader.Visible = false";
```

10. **Sales by Year Subreport（年度別売上高サブレポート）**

    処理は不要です。

11. **Sales Totals by Amount（金額別総売上高）**

    このレポートには、手作業で変換する必要があるコードが含まれます。次のコードをページヘッダセクションのOnPrint プロパティに割り


Copyright © GrapeCity, Inc. All rights reserved.

当てます。

**Visual Basic コードの書き方**

```
Visual Basic
PageTotal = 0
```

**C# コードの書き方**

```
C#
PageTotal = 0;
```

次のコードを詳細セクションのOnPrint プロパティに割り当てる必要があります。

**Visual Basic コードの書き方**

```
Visual Basic
PageTotal = PageTotal + SaleAmount
HiddenPageBreak.Visible = (Counter = 10)
```

**C# コードの書き方**

```
C#
PageTotal = PageTotal + SaleAmount;
HiddenPageBreak.Visible = (Counter = 10);
```

12. **Summary of Sales by Quarter（四半期別売上高のサマリー）**

    このレポートには、四半期順にソートされるグループがあります（前述の4番目の例外を参照）。これを解決するには、RecordSource プロパティを次のように変更することによって、**ShippedDate** 四半期の値を含むフィールドをソースデータセットに追加します。

```
SELECT DISTINCTROW Orders.ShippedDate,
Orders.OrderID,
[Order Subtotals].Subtotal,
DatePart("q",Orders.ShippedDate) As ShippedQuarter
FROM Orders INNER JOIN [Order Subtotals]
ON Orders.OrderID = [Order Subtotals].OrderID;
WHERE (((Orders.ShippedDate) Is Not Null));
```

    新しいフィールド **ShippedQuarter** を使用するように、グループの**GroupBy** プロパティを変更します。

13. **Summary of Sales by Year（年度別売上高のサマリー）**

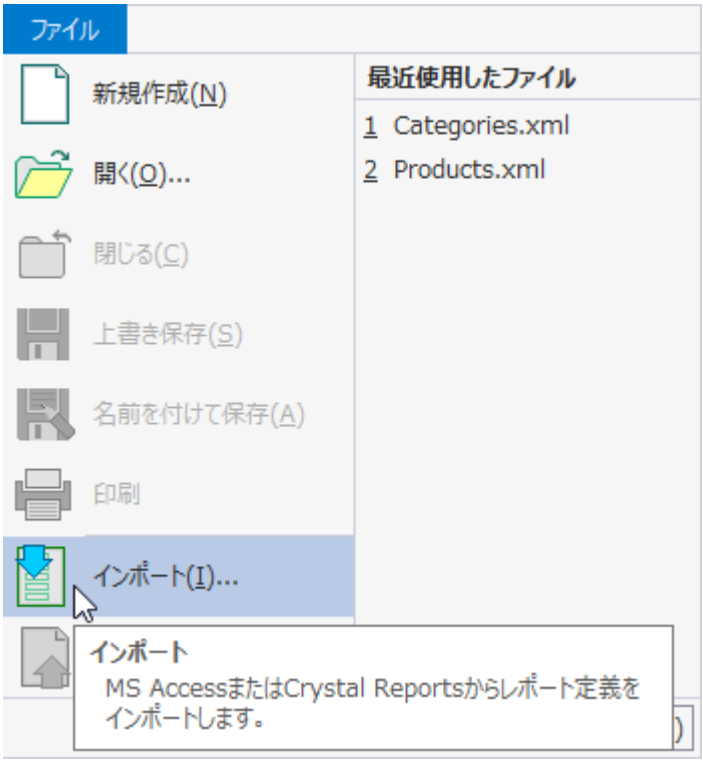
    処理は不要です。

以上をまとめると、NorthWind データベースからインポートされる 13 のレポートのうち、編集の必要がないレポートが8つ、コードを変換する必要があるレポートが3つ、SQL 文を変更する必要があるレポートが1つ、置き換えられないグラフコントロールがあるレポートが1つあるということになりますが、それでも編集に必要な手間はわずかです。インポート機能を使用すれば既存レポートの移行がいかに簡単かお分かりいただけたことと思います。

# Crystal Reports レポートのインポート

**C1Report デザイナ**は、Crystal Reports 定義ファイル（rpt レポート）をインポートすることができます。


Copyright © GrapeCity, Inc. All rights reserved.




Crystal Reports 定義ファイルからレポートをインポートするには

1. **[ファイル]**メニューをクリックして、**[インポート]**を選択します。**[レポート定義のインポート]**ダイアログボックスが表示され、インポートするファイル名の入力を求められます。
2. Crystal Reports 定義ファイル(.rpt)を選択します。**C1ReportDesigner** アプリケーションが、レポートを **C1Report** 形式に変換します。

**C1ReportDesigner** アプリケーションは、Crystal Reports をインポートする際に、次の変換のいくつかをサポートします。

- 内部または外部データソースに連結されたレポートは、元のデータソースパスに変更を行う必要なく、インポートして実行できます。
- 日付、時刻、および数値の形式は、変換中に維持されます。
- パーセンテージの集計を含む式は、正しくインポートされます。
- いくつかの式の TextObject 変換では、**RAS API** がサポートされます。そのような例として、テキストと式の組み合わせ(**Text + [Expression]**)を含む単一の Textbox を変換する場合があります。

**メモ**:レポートをインポートする前に、システムにインストールされている Crystal Reports と Visual Studio のバージョンに互換性があることを確認してください。また、システムに Crystal Reports 2013 がインストールされている場合に、Crystal Reports を C1Report に変換し、そのレポートを実行するには、xtreme.mdb へのデータベースパスを手動で変更する必要があります。

Crystal オブジェクトモデルで公開されていない要素や C1Report でサポートされていない要素などの少数の例外を除くと、ソースレポートの大部分の要素がこのインポートプロセスによって処理されます。例外には、画像フィールド、グラフ、およびクロス集計フィールドがあります。

## Reports のグラフ作成

集計グラフ機能は、パワフルでしかもシンプルな使いやすい機能です。**Reports for WinForms**は、拡張可能なカスタムフィールドアーキテクチャを使用するグラフフィールドをサポートしています。[グラフ]フィールドは、レポートデザイナアプリケーションと共にインストールされる C1.Win.C1Report.CustomFields.2.dll アセンブリのカスタムフィールドとして実装され、完全なソースコードが付属したサンプル(**CustomFields**)としても含まれています。次の各トピックでは、**C1ReportDesigner**アプリケーションを使用してレポートのグラフフィールドをカスタマイズする方法を説明します。**C1ReportDesigner** アプリケー

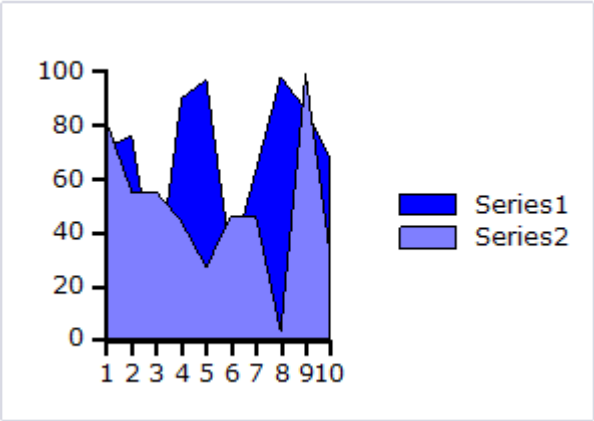

Copyright © GrapeCity, Inc. All rights reserved.

ションは、**Reports for WinForms** と **Reports for WPF** の両方と共にインストールされています。

## グラフ種類

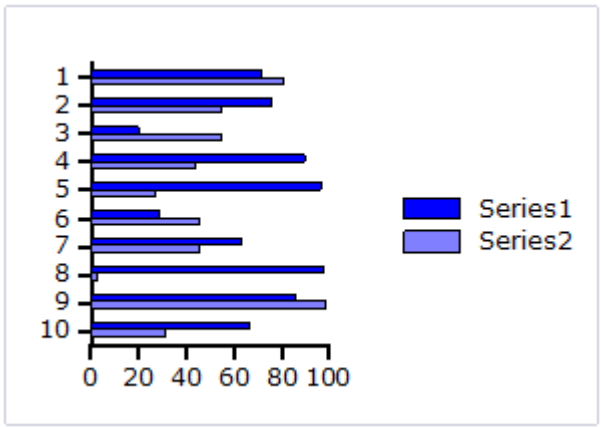
**Report for WinForms** の**チャートフィールド**では、**ChartTypeEnum** を使用してさまざまな種類のチャートを挿入できます。**C1Report** でサポートされているチャートタイプには、**面**、**横棒**(水平の棒グラフ)、**縦棒**(垂直の棒グラフ)、**散布図**(X-Y 値)、**折れ線**、**円**、**階段**、**ヒストグラム**、**レーダー**、**ポーラ**の 10 種類があります。チャートタイプは、**C1ReportDesigner** のプロパティペインで **ChartType** プロパティを使用して簡単に選択できます。

**面グラフ**:面グラフでは、同じ系列のデータポイントどうしを線で結び、この線の下を塗りつぶすことで 1 つの系列が描画されます。系列は、それぞれ前の系列の上に表示されます。



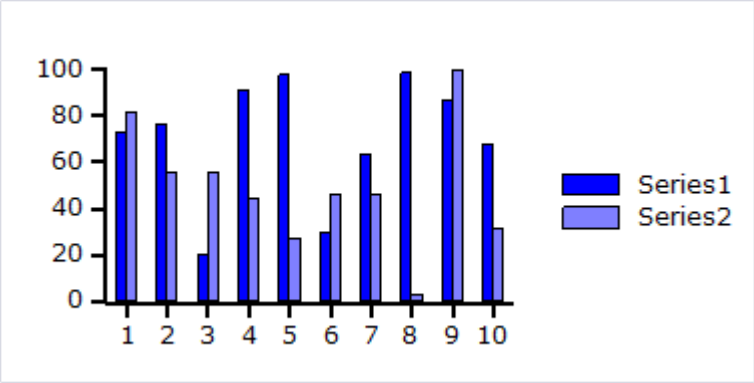

**横棒および縦棒グラフ**:横棒グラフまたは縦棒グラフは、各系列を同じ色と幅の棒の集合として表し、各棒の長さは値によって決定されます。新しい系列は、それぞれ前の系列の棒の集合の隣に棒の集合としてプロットされます。横棒または縦棒グラフは、1 つの棒集合に各系列を 1 つの棒として描画します。棒集合の数は、データ内のポイントの数です。各集合は、各系列の n 番目のデータポイントを表示します。棒が横向きに置かれる場合は横棒グラフと呼ばれ、棒が縦向きに置かれる場合は縦棒グラフと呼ばれます。

次の図は、**横棒**グラフを表します。

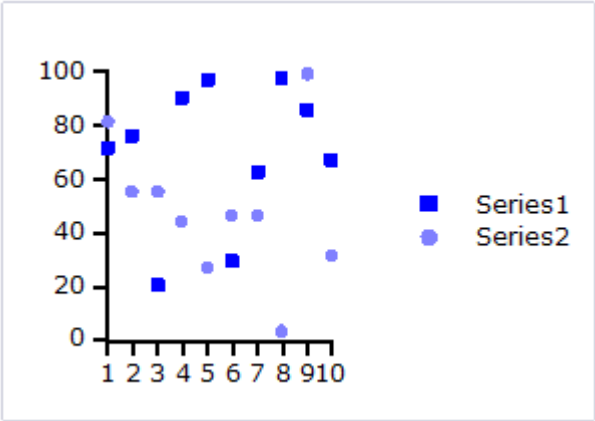



次の図は、**縦棒**グラフを表します。

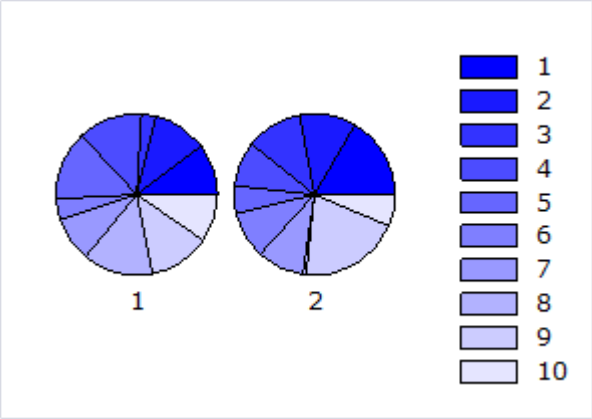

Copyright © GrapeCity, Inc. All rights reserved.




**散布図**:散布図は、2 つの値を使用して各データポイントを表します。このタイプのチャートは、統計的手法によって変数間の関係を定量化(通常、線形回帰分析)するためによく使用されます。

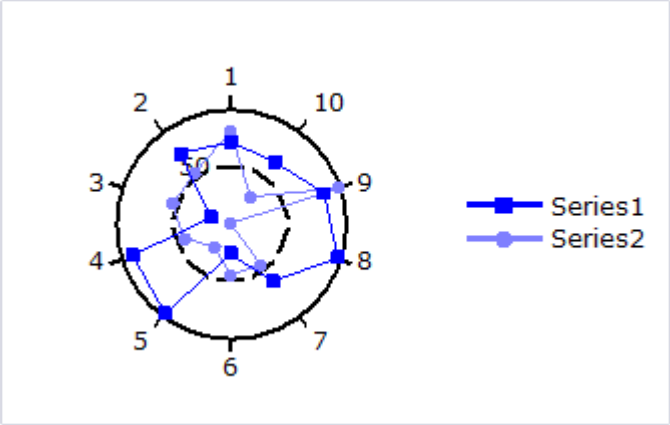



**円グラフ**:円グラフは、各系列を円のスライスとして描画します。円の数は、データ内のポイントの数を表します。各円は、各系列の n 番目のデータポイントを表示します。円グラフをカスタマイズして、凡例やラベルを表示することもできます。詳細は、「チャートプロパティ」を参照してください。

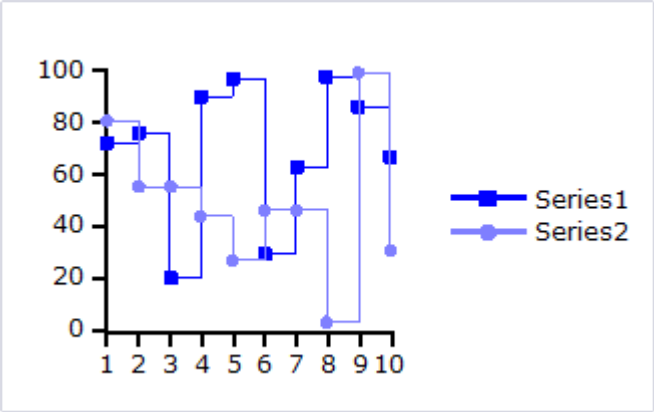



**レーダーチャート**:レーダーチャートは、データセットごとにレーダー線に沿って y 値を描画します(x 値は、ラベル以外は無視されます)。データに一意のポイントが n 個ある場合、グラフ面は、n 個の等しい角度のセグメントに分割され、360/n 度の間隔でレーダー線(各ポイントを表す)が描画されます。デフォルトでは、最初のポイントを表すレーダー線が垂直に描画されます(90 度)。レーダーチャートはさらにカスタマイズが可能です。詳細は、「チャートプロパティ」を参照してください。

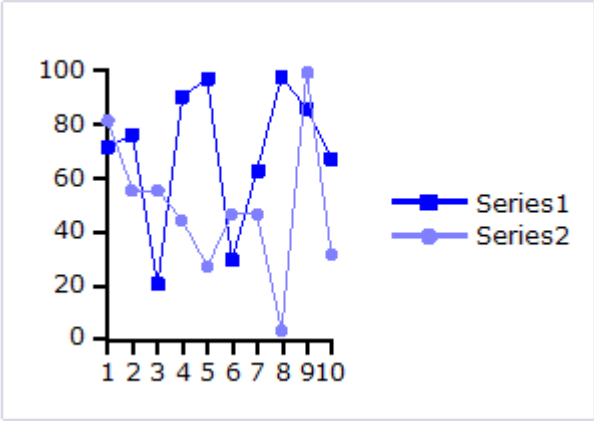

Copyright © GrapeCity, Inc. All rights reserved.




**階段グラフ**:階段グラフは、データポイントどうしを線で結んで系列を描画する XY プロットグラフの形式の 1 つです。これらのチャートは、X の値が特定の値になると Y の値が突然変化するという、値が離散的に変化する場合によく使用されます。最もわかりやすい例としては、時間の経過に伴う小切手口座の残高の変化があります。時間の経過(X 値)と共に、預金が発生したり、小切手が切られたりします。そのたびに、小切手口座の残高(Y 値)は、徐々にではなく突発的に変更されます。時間が経過しても預金が発生せず小切手も切られない場合、その期間の残高(Y 値)は変わりません。




**折れ線グラフ**:折れ線グラフは、データポイントをつないで各系列を描画します。これは、さまざまなデータグループ間の値の変化を最も効率よく示す方法です。これらのグラフは、長期にわたるトレンドやパフォーマンスを示す場合によく使用されます。

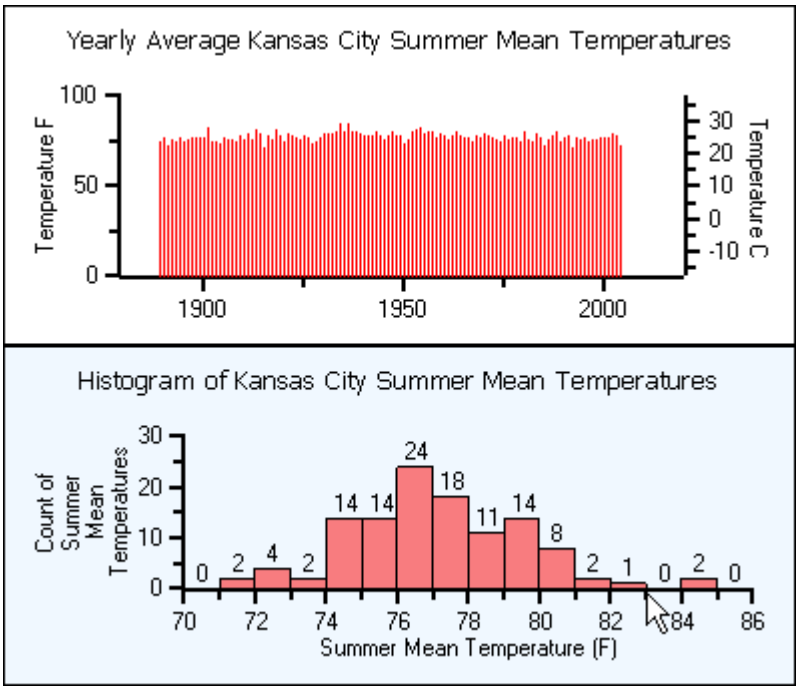



**ヒストグラム**:ヒストグラムは、生データ値のコレクションから度数分布をプロットします。これは、グループを形成するデータによく使用され、生データのコレクションを計測して、定義された区間内に分布するデータ値の数をプロットすることで生成されます。元の値は、ヒストグラムのデータではなく度数を生成するために使用されます。棒グラフに似ていますが、棒グラフが一般に質的変数に使用されるのに対して、ヒストグラムは量的変数に使用されることに注意する必要があります。

ヒストグラムと棒グラフは、外観は類似していますが、機能は異なります。棒グラフがデータポイントから作成されるのに対して、ヒストグラムはデータの度数分布から作成されます。以下のチャートは、棒グラフとヒストグラムの違いを示しています。どちらのチャートも、使用している Y データはまったく同じです。棒グラフ(上)は、各年の平均気温を発生年ごとに示しています。


Copyright © GrapeCity, Inc. All rights reserved.

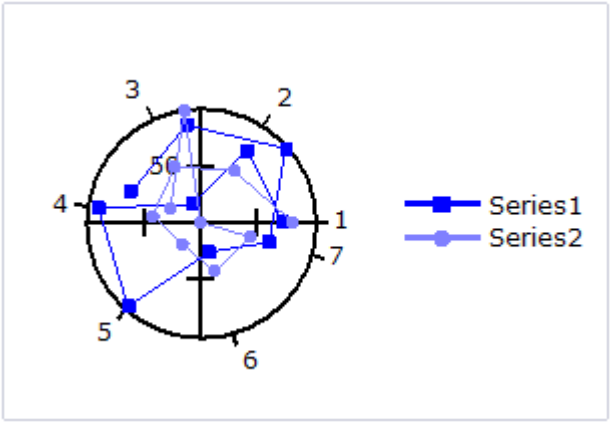
ヒストグラム（下）は、入力として同じ気温データを使用していますが、各区間内に分布する気温の数を自動的に集計し、結果のヒストグラムを描画しています。わかりやすいように、各区間の個数を示したグラフラベルが区間の上部に追加されています。




ヒストグラムは、量的変数のデータ分布にある際だった特徴を特定するために役立ちます。量的変数の重要な特性は以下のとおりです。

- 典型的な値を明らかにします。
- データは一般的な形状になります。データ値は、中央から対称に分布する場合も非対称に分布する場合もあります。
- データグループから離れた値がある場合、それらは異常値として示されます。
- データ値は標準値に近い場合も遠い場合もあります。
- 分布は、単一のピークを形成する場合と、複数のピークや谷を形成する場合があります。

**ポーラチャート**：ポーラチャートは、系列ごとに x 座標と y 座標を（theta,r）として描画します。theta は原点からの回転角度であり、r は原点からの距離です。theta は、度（デフォルト）またはラジアンを単位として指定できます。X 軸は円なので、X 軸の最大および最小値は固定されています。系列は、個別に描画することも、積層にして描画することもできます。ポーラチャートはさらにカスタマイズが可能です。詳細は、「チャートプロパティ」を参照してください。




## その他の便利なグラフプロパティ

上述の **DataX**と **DataY**プロパティに加え、Chart オブジェクトは、次の一般に使用される他のプロパティをいくつか提供しま


Copyright © GrapeCity, Inc. All rights reserved.

す。

- **ChartType**: このプロパティでは、表示するグラフのタイプを選択できます。**Bar**（横棒）、**Column**（縦棒）、**Scatter**（X-Y値）、**Line**、**Area**、および **Pie**の6つのオプションがあります。
- **DataColor**: このプロパティは、横棒、縦棒、面、散布図の記号、および円スライスの描画に使用する色を選択します。グラフが複数の系列を含む場合、[**グラフ**]フィールドは系列ごとに選択した色のさまざまな陰影を自動的に生成します。系列ごとに特定の色を選択する場合、**Palette**プロパティを代わりに使用し、その値を、使用する色を含むセミコロン区切りのリスト（「Red;Green;Blue」など）に設定します。
- **FormatY, FormatX**: これらのプロパティは、各軸の値を表示する際に使用する書式を決定します。たとえば、**FormatY**を「c」に設定すると、[**グラフ**]フィールドは Y 軸の値が通貨値として書式設定されます。これは、通常のレポートフィールドの **Format**プロパティと似ています。
- **XMin, XMax, YMin, YMax**: これらのプロパティでは、各軸の範囲を指定できます。これらのどれかを -1に設定すると、[**グラフ**]は範囲を自動的に計算します。たとえば、**YMax**プロパティを 100 に設定すると、100 を超える任意の値は切り捨てられ、グラフに表示されません。

これらのプロパティはすべてのグラフタイプに適用されます。**円**グラフのみに適用するいくつかの追加のプロパティは次のとおりです。

- **ShowPercentages**: 各円スライスは、スライスの X 値を表示する凡例を持ちます。**ShowPercentages**プロパティが true に設定された場合、凡例も円に対するスライスのサイズを示すパーセント値を含みます。パーセントは **FormatY**プロパティによって指定された値を使用して書式設定されます。たとえば、**FormatY**を「p2」に設定すると、凡例は X 値と小数点以下2桁のパーセントを含みます（北部（15.23%））。
- **RadialLabels**: このプロパティは、凡例をグラフの右側に表示せずに、引き出し線付きのラベルを各スライスにアタッチするように指定します。これは、スライスの数が少ない（最大約10個）円の場合に有効です。

次のプロパティは、**レーダー**チャートに適用されます。

- **Start**：360 度の円上のチャートの開始角度を指定します。角度は、反時計回りに計測されます。**Degrees** プロパティが true に設定されている場合、開始角度は度単位で計測されます。そうでない場合は、ラジアン単位です。
- **Degrees**：チャートの開始角度の計測単位を指定します。このプロパティが true に設定されている場合、計測は度単位で行われます。そうでない場合は、ラジアン単位になります。
- **Filled**：このプロパティを true に設定することで、レーダーチャート内のデータポイントで囲まれた領域の塗りつぶしを設定できます。
- **FlatGridLines**：レーダーチャートの Y 座標グリッド線は、デフォルトでは円形です。FlatGridLines プロパティを使用すると、グリッド線をフラットな Y 座標グリッド線に設定できます。

次のプロパティは、**ポーラ**チャートに適用されます。

- **Start**：360 度の円上のチャートの開始角度を指定します。角度は、反時計回りに計測されます。**Degrees** プロパティが true に設定されている場合、開始角度は度単位で計測されます。そうでない場合は、ラジアン単位です。
- **Degrees**：チャートの開始角度の計測単位を指定します。このプロパティが true に設定されている場合、計測は度単位で行われます。そうでない場合は、ラジアン単位になります。
- **PiRatioAnnotations**：C1Chart では、**Degrees** が False に設定されて、チャートがラジアン値を表現されている場合に、チャートにラジアンではなくπ比を表示するオプションが提供されています。このプロパティを true に設定すると、ポーラチャート上の値にπ比が表示されます。

次のプロパティは、**HistogramOptions** によって公開される**ヒストグラム**にのみ適用されます。

- **DisplayType**：特定の系列の度数データを表示する方法を指定します。これは、1 つのチャートグループでさまざまな度数データを一意に表示する場合に便利です。
- **IntervalCreationMethod**：このプロパティを使用して、ヒストグラムにさまざまな区間境界を指定できます。IntervalCreationMethod プロパティから、以下の 3 つの方法のいずれかを選択できます。
  - **Automatic**：Automatic 法を使用する場合は、データの最大値と最小値を使用し、区間をデータ平均の 3 標準偏差内に収まるように制限して、区間の上限値と下限値が計算されます。区間数はオプションです。区間境界は均等になるように計算されます。
  - **SemiAutomatic**：SemiAutomatic 法を使用する場合は、区間数と共に区間の上限値と下限値を指定します。区間境界は均等になるように計算されます。SemiAutomatic 法を選択した場合は、IntervalStart、IntervalWidth、および IntervalNumber プロパティを使用できます。IntervalStart プロパティは、最初の区間の


Copyright © GrapeCity, Inc. All rights reserved.

開始数値を取得または設定します。
    - **XDataBoundaries**：XDataBoundaries 法を使用する場合は、データ系列の X 値を使用して各区間境界が明示的に設定されます。X 値はソートされ、重複値は除外されます。結果の各昇順値を使用して、次の区間境界が決定されます。したがって、結果の 1 番目と 2 番目の X 値によって先頭の区間が定義され、後続の X 値によってそれぞれ次の区間の終わりが指定されます。N 個の区間を指定するには、N+1 個の一意の X 値が必要です。
- **IntervalNumber**：**Automatic** 法と **SemiAutomatic** 法によって作成されるヒストグラムの区間の数を指定します。
- **IntervalStart**：**SemiAutomatic** 法によって作成されるヒストグラムの最初の区間の開始数値を指定します。
- **IntervalWidth**：**SemiAutomatic** 法によって作成されるヒストグラムの区間の幅の数値を指定します。
- **NormalDisplay**：ヒストグラムと比較する正規分布（ガウス）曲線を表示するために使用されるプロパティを指定します。
- **NormalizationInterval**：不均等な区間を持つヒストグラムの正規区間幅を指定します。ヒストグラムの形状を維持したまま、各区間の高さが単位幅あたり同じ度数を表すように幅を正規化します。
- **Normalized**：ヒストグラムの各系列区間を正規化するかどうかを指定します。

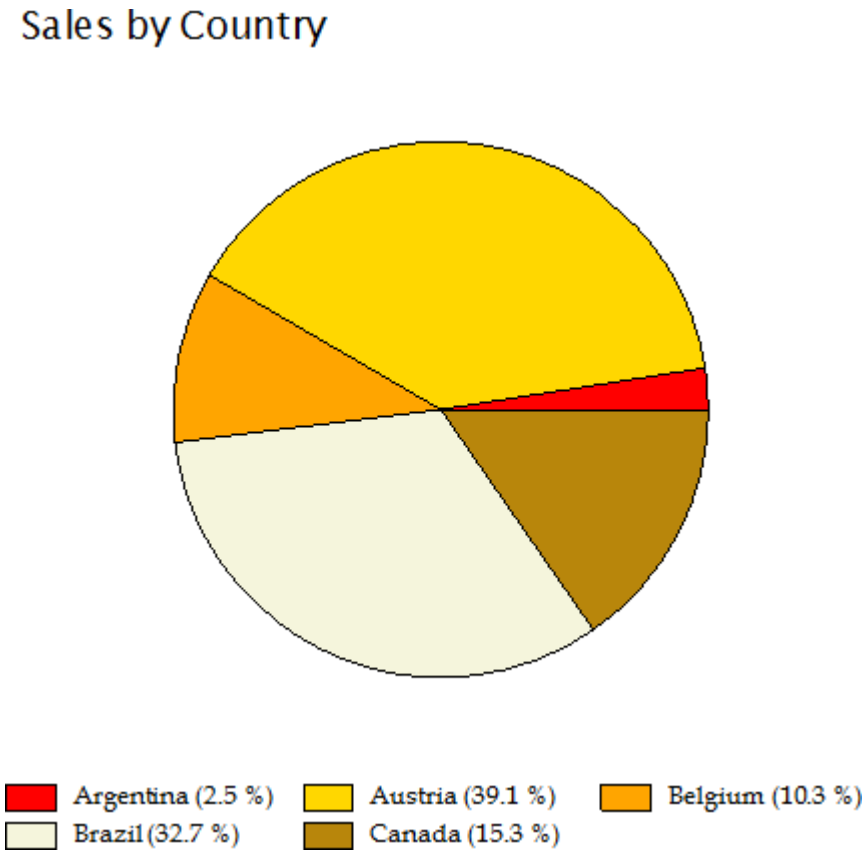
[**グラフ**]フィールドは実質的に、すべてのグラフ作成サービスを提供して独自の非常に機能豊富なオブジェクトを備えた **C1Chart**コントロールのラッパーです。[**グラフ**]フィールドをさらにカスタマイズする場合は、**ChartControl**プロパティを使用し、スクリプトによって内部の **C1Chart**オブジェクトにアクセスできます。

たとえば、[**グラフ**]フィールドは凡例の位置を制御するプロパティを持っていません。しかし、**C1Chart**コントロールが持っており、このプロパティに **ChartControl**プロパティからアクセスできます。たとえば、下のスクリプトにより、グラフの凡例はグラフの右側ではなく、下側に配置されます。

VisualBasic

```
'凡例をグラフの下側に配置します。
chartField.ChartControl.Legend.Compass = "South"
```

このスクリプトをレポートの **OnLoad**プロパティに割り当てると、グラフは下の画像のようになります。




これらのグラフの作成に使用するその他のプロパティは次のとおりです。

**ChartType** = Pie


Copyright © GrapeCity, Inc. All rights reserved.

**FormatY** = "p1"
**ShowPercentage** = true
**Palette** = "Red;Gold;Orange;Beige;DarkGoldenrod;Goldenrod;"

## 複数の系列を含むグラフ

複数の系列を含むグラフを作成するには、**DataY**プロパティを、グラフにする各データフィールドの名前をセミコロン区切りで含む文字列に設定するだけです。

たとえば、製品価格と割引率を表示するグラフを作成するには、**DataY**プロパティを以下のように設定します。:

**DataY** = "UnitPrice;Discount"

各系列の表示に使用する色を指定する場合、**Palette**プロパティをセミコロン区切りの色のリストに設定します。たとえば、下に表示した値の場合、グラフは「単価」系列を赤で、「割引率」系列を青で表示します。

**Palette** = "Red;Blue"

## グループ化したレポートのグラフ

**Reports for WinForms** では、複数のグループを含むレポートを作成できます。たとえば、単一のフラットレポートにすべての製品をリストする代わりに、カテゴリ別に製品をグループ化できます。各グループは、ヘッダーとフッターセクションを持っており、タイトルや小合計などを含むグループに関する情報を表示できます。

グラフをグループヘッダーに追加すると、グラフは現在のグループのデータのみを表示します。反対に、グラフをレポートヘッダーまたはフッターに追加すると、レポート内のすべてのデータが包含されます。

これを例示するために、以下の図で、レポートデザイナに表示されるレポート定義を示し、[**グラフ**]フィールドをレポートヘッダーとグループヘッダーに追加した場合の効果を示します。

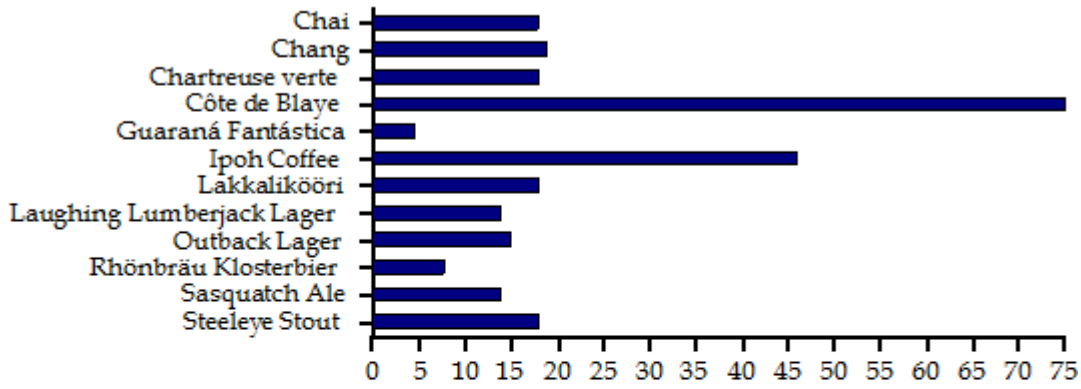
| |
|---|
| **[レポートヘッダー]セクション**<br>ここでは、グラフフィールドはレポート全体に1つのグラフのみ生成します。<br><br>グラフはレポートのデータソース内のすべてのデータを表示します。 |
| **[ページヘッダー]セクション** |
| **[グループヘッダー]セクション(CategoryName)**<br>ここでは、グラフフィールドは CategoryName 値ごとに1つのグラフを生成します。<br><br>各グラフは現在の CategoryName のすべてのデータを表示します。 |
| **[詳細]セクション** |
| **[グループフッター]セクション(CategoryName)** |
| **[ページフッター]セクション** |
| **[レポートフッター]セクション** |

上述の例を続けて、グラフをグループヘッダーに追加した場合に **DataX**プロパティを「ProductName」に、**DataY**プロパティを「UnitPrice」に設定すると、最終的なレポートはカテゴリごとに1つのグラフを含み、各グラフはこのカテゴリの製品の単価を表示します。

下の画像に、グループヘッダー、含まれるグラフ、および数例の詳細レコードを含む上述のレポートの画面例を示します。


Copyright © GrapeCity, Inc. All rights reserved.

## Beverages

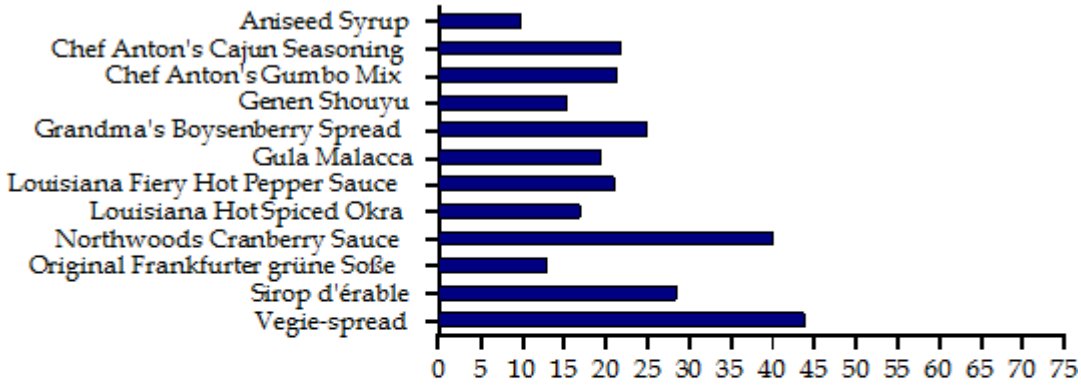



"Unit prices per product: "& CategoryName & " (chart truncated to $75)

| ProductName | QuantityPerUnit | UnitPrice |
|---|---|---|
| Chai | 10 boxes x 20 bags | 18.00 |
| Chang | 24 - 12 oz bottles | 19.00 |
| Chartreuse verte | 750 cc per bottle | 18.00 |
| Côte de Blaye | 12 - 75 cl bottles | 263.50 |
| Guaraná Fantástica | 12 - 355 ml cans | 4.50 |
| Ipoh Coffee | 16 - 500 g tins | 46.00 |

上のグラフは、「飲料」カテゴリの製品の単価を示します。下のグラフは、「調味料」カテゴリの製品の単価を示します。

## Condiments




"Unit prices per product: "& CategoryName & " (chart truncated to $75)

| ProductName | QuantityPerUnit | UnitPrice |
|---|---|---|
| Aniseed Syrup | 12 - 550 ml bottles | 10.00 |
| Chef Anton's Cajun Seasoning | 48 - 6 oz jars | 22.00 |
| Chef Anton's Gumbo Mix | 36 boxes | 21.35 |
| Genen Shouyu | 24 - 250 ml bottles | 15.50 |
| Grandma's Boysenberry Spread | 12 - 8 oz jars | 25.00 |
| Gula Malacca | 20 - 2 kg bags | 19.45 |

**DataX** = "Product Name"
**DataY** = "Unit Price"

グラフが格納先のセクションの範囲に基づいてデータを自動的に選択するため、グループ化したレポートでのグラフの作成は非常に簡単です。.


Copyright © GrapeCity, Inc. All rights reserved.

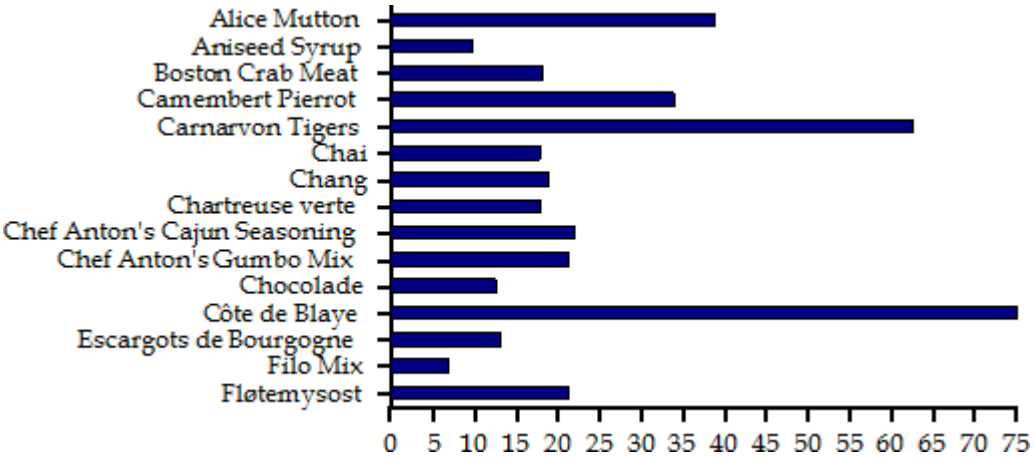
# グラフのデータプロット

単純なグラフの作成は非常に簡単です。単純なグラフを作成するには、次の手順に従います。

1. **C1ReportDesigner**アプリケーションを開き、レポート定義ファイルを作成するか、開きます。
2. [**グラフ**]フィールドをレポートに追加した後、これを選択して、デザイナのプロパティウィンドウでプロパティを表示します。
3. グラフの **DataX**プロパティを、X 軸(グラフカテゴリ)に表示する値のフィールド名に設定します。
4. グラフの **DataY**プロパティを、Y 軸(グラフ値)に表示する値のフィールド名に設定します。
5. オプションで、**ChartType**と **DataColor**などの追加のプロパティを設定します。

たとえば、下のグラフは NorthWind Products テーブルに基づいて作成されました。この場合、次のプロパティが設定されました。

**DataX** = "ProductName"
**DataY** = "UnitPrice"




このグラフタイプ(横棒)の場合、値軸(DataYフィールドを表示)が水平軸で、カテゴリ軸が垂直軸であることに注意してください。

この場合、表示する値の数を制限するために、フィルタがデータに適用されました。フィルタがない場合、グラフは余計な値も含んで、垂直軸の読み取りが難しくなります。

**DataY**プロパティはフィールドの名前に限定されません。系列を指定する文字列は実際に、完全な式として処理され、レポートの任意の通常フィールドのように計算されます。

たとえば、各フィールドの実際の価格を表示するグラフを作成するには、**DataY**プロパティを以下のような値に設定できます。

**DataY** = "UnitPrice * (1 - Discount)"

# 集計グラフ

**Reports for WinForms**の 2009 v3 リリースで組み込まれた[**グラフ**]フィールドは、「集計グラフ機能」と呼ばれるパワフルな新機能を備えています。この機能により、希望の集計関数(合計、平均、標準偏差など)を使用し、同じカテゴリ(**DataX**)を持つデータ値(**DataY**)を自動的に集計するグラフを作成できます。

- Sum
- Count
- Average
- Minimum
- Maximum
- Variance
- VariancePop
- StandardDeviation
- StandardDeviationPop


Copyright © GrapeCity, Inc. All rights reserved.

この機能を例示するために、国、顧客、および注文 ID 別にグループ化する「請求書」レポートを考察します。レポートの概要は次のようになります。

| |
|---|
| **[レポートヘッダー]セクション** |
| **[ページヘッダー]セクション** |
| **[グループヘッダー]セクション(国)** |
| **[グループヘッダー]セクション(顧客)** |
| **[グループヘッダー]セクション(注文 ID)** |
| **[詳細]セクション** |
| **[グループフッター]セクション(注文 ID)** |
| **[グループフッター]セクション(顧客)** |
| **[グループフッター]セクション(国)** |
| **[ページフッター]セクション** |
| **[レポートフッター]セクション** |

ここで、グラフを、現在の国の各顧客が発注したすべての注文の合計値を表示するそれぞれの[**国**]ヘッダーに追加する場合を想定します。

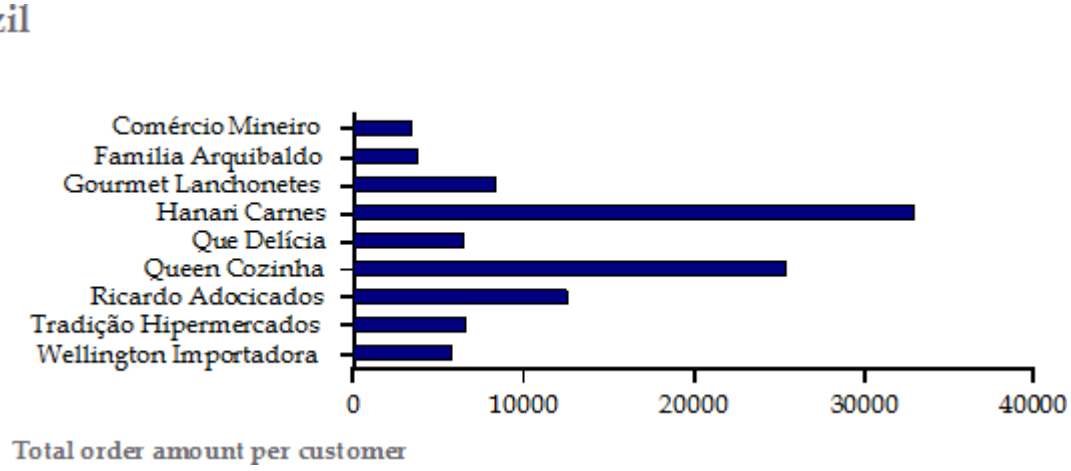
まず、[**グラフ**]フィールドを「国」ヘッダーに追加して、**DataX** と **DataY** プロパティを次のように設定します。

**DataX** = "CustomerName"
**DataY** = "ExtendedPrice"

これは機能しません。国ごとのデータは通常、顧客ごとに複数のレコードを含み、グラフはレコードごとに1つのデータポイントを作成します。グラフには、実際に各顧客の複数の値を単一データポイントに追加する必要があるかどうかを推測できません。

このシナリオを解決するために、**Aggregate**プロパティが[**グラフ**]フィールドに追加されました。このプロパティは、グラフに同じカテゴリを持つ値をグラフの単一ポイントに集計する方法を通知します。**Aggregate**プロパティは、任意の共通集計関数(合計、平均、カウント、最大、最小、標準偏差、および分散)をデータ上で実行するように設定できます。

例を続けると、これで、グラフの **Aggregate**プロパティを「Sum」に設定できるようになりました。これにより、グラフは、同じ顧客に属するレコードのすべての「ExtendedPrice」値を単一データポイントに追加します。結果は次のようになります。




各顧客が一度だけ表示されることに注意してください。グラフに表示された値は、同じ「顧客」のすべてのフィールドの「ExtendedPrice」値の合計に対応します。

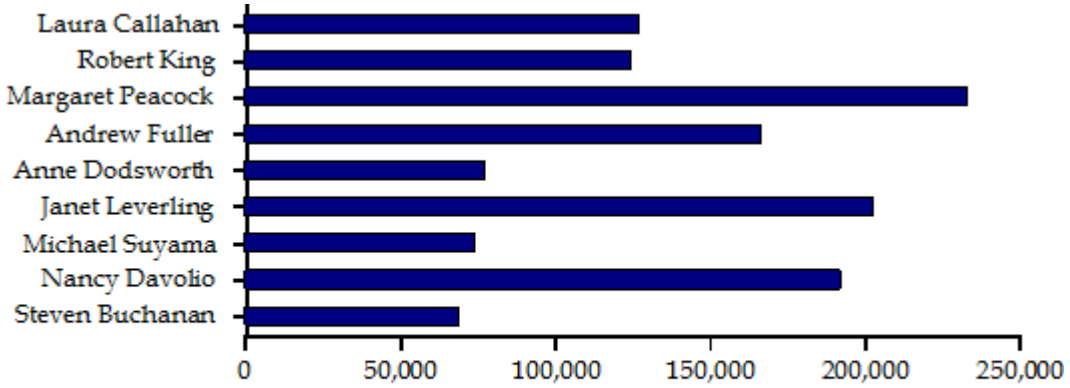
「国」ヘッダーフィールドのグラフが表示されているため、グラフは国ごとに繰り返され、この国のすべての顧客を表示します。


Copyright © GrapeCity, Inc. All rights reserved.

グラフをレポートヘッダーセクションに配置した場合、レポート全体の範囲でデータを集計します。たとえば、各販売員が発注した合計金額を表示するグラフを含む「請求書」レポートを開始すると仮定します。これを実行するには、[**グラフ**]フィールドをレポートヘッダーセクションに追加し、次のプロパティを設定します。

**DataX** = "Salesperson"
**DataY** = "ExtendedPrice"
**Aggregate** = "Sum"

下の画像に、結果のグラフを示します。




Total order amount per Salesperson

グラフがレポートヘッダーセクションに配置されたため、表示された値はすべての国とすべての顧客を包含します。グラフフィールドをレポートヘッダーから「国」グループヘッダーに移動すると、国ごとに類似のグラフが得られ、この国の各販売員が販売した合計金額を表示します。

# レポート内のマップ

**Reports for WinForms** は、拡張可能なカスタムフィールドアーキテクチャを使用するマップフィールドをサポートしています。[**マップ**]カスタムフィールドは、レポートデザイナアプリケーションと共にインストールされる **ComponentOne for WPF** の C1.WPF および C1.WPF.Maps の2つのアセンブリを使用し、2箇所にカスタムフィールドとして実装されます。次のトピックでは、**C1ReportDesigner**アプリケーションを使用し、レポート内のマップフィールドをカスタマイズする方法を説明します。

[マップ]カスタムフィールドが **ComponentOne for WPF** の C1.WPF と C1.WPF.Maps の2つのアセンブリを使用することに注意してください。始める前に、これらのアセンブリが使用可能で、プロジェクトで参照できることを確認してください。

**C1ReportDesigner**アプリケーションの[マップ]カスタムフィールドの使用を開始するには、次の手順を実行します。

1. **C1ReportDesigner**アプリケーションを実行します。詳細については、「Visual Studio からの C1Report デザイナへのアクセス」を参照してください。
2. マップアイコンが **C1ReportDesigner**ツールバーに存在することを確認します。含まれていない場合は、次の行を **C1ReportDesigner.EXE.settings**ファイルの <customfields>セクションに追加する必要がある可能性があります。
   ```
   <item value="C1.C1Report.CustomFields.4;C1.C1Report.CustomFields.Map" />
   ```
3. 新しいレポートを作成するか、既存のレポートを開きます。例については、「基本的なレポート定義ファイルの作成」を参照してください。
4. マップアイコンをクリックしてレポート上にドラッグし、[マップ]フィールドを追加します。

これで完了です。[マップ]フィールドの主な側面は次のとおりです。

- タイルおよびデータレイヤ
- 凡例
- スタイル
- 式
- 自動的なズーム／中央揃え、およびデータトラッキング

詳細については、[マップ]カスタムフィールドの最も重要なプロパティのいくつかを説明している次の各トピックを参照してくだ


Copyright © GrapeCity, Inc. All rights reserved.

さい。

# レイヤ

マップのメイン部分は、地球表面またはその一部分を表すラスターグラフィックスを提供するタイルレイヤ、および空間データを表す0個以上のレイヤです。

タイルレイヤは、**TileSource**プロパティによって指定されます。これは、IVirtualEarth タイルソース（道路、航空写真、またはハイブリッド）に設定できます。タイルソースは「none」に設定でき、その場合、タイルはマップ上に描画されません。これは、特に KML などの他のレイヤがマップの視覚化に十分なデータを提供する場合に役立ちます。

タイルソースが「none」でない限り、タイルはレポートの実行時にネットワーク上の場所からロードされます。このとき、処理速度がかなり低下する場合があります。

タイルレイヤを除き、他のすべてのレイヤは **Layers**コレクションに格納されます。現在、次の3つのレイヤタイプがサポートされます。

- ポイント。「ポイント」レイヤにより、空間データをマップ上のポイントとして表示できます。「ポイント」レイヤでは、マーカーが各データごとに描画されます。
- ライン。「ライン」レイヤは、各データ行の2点間に直線を描画します。
- KMLレイヤ。KML（Keyhole マークアップ言語）は、さまざまな地理情報を記述できる XMLベースの言語です。KML の詳細については、「」を参照してください。KML レイヤでは、ローカルまたは Web ベースの KML ファイルをマップに読み込んで表示できます。

## レイヤデータソースの指定

**Layers**コレクションのレイヤごとに、レイヤ独自の **RecordSource**（SQL ステートメント）を指定できます。省略された場合、データ（KML レイヤを除く）は親レポート（現在のグループ化範囲に従ってフィルタ済み）から直接取得されます。提供された場合、**RecordSource**は親レポートの接続文字列上で使用されます。

# トラッキング

［マップ］フィールドによって表示されるマップは、マップ上に表示されるデータの自動的な中央揃えとズームイン／アウトを行います。この動作は次の2つの係数によって決定されます。

- ［マップ］フィールド全体に指定された AutoCenterと AutoZoomプロパティの値、および中央揃え／ズームを自動的に微調整するいくつかの関連プロパティ（**AutoZoomPadLon**、**AutoZoomPadLat**、**MaxAutoZoom**、**RoundAutoZoom**）。
- レイヤによって表される空間データ（データを「トラッキング」する場合）。トラッキング（特定部分の空間データを自動的な中央揃えとズームに使用するかどうかなど）は、レイヤの **Track**プロパティによって決定されます。また、KML レイヤの場合、特定の KML 項目をトラッキングするかどうかを決定する式を指定できます。

# スタイル

マップ要素の視覚属性は主にスタイルによって定義されます。いくつかのスタイルタイプ（ポイントマーカースタイル、ラインスタイル、および KML 項目スタイル）があります。適用可能なタイプはコンテキストによって決定され、たとえば、ポイントレイヤはポイントマーカースタイルを使用し、ラインレイヤはラインスタイルを使用するのようになります。通常、スタイルは、データドリブン式として指定でき（実際のスタイルは実行時のデータによって決定）、デフォルトではフォールバックスタイルが使用されます。スタイル式の指定と評価方法は次に説明されます。

［マップ］カスタムフィールドには、次の3つのスタイルコレクションが含まれます。

- MarkerStyles
- LineStyles
- KmlItemStyles


Copyright © GrapeCity, Inc. All rights reserved.

これらのスタイルは、[**マップ**]上で定義されたすべてのレイヤや、現在のレポート内の他の[**マップ**]フィールドにも使用可能です。各コレクション内のスタイルは、インデックス、または名前(**Name**プロパティを使用。推奨)のいずれかによってアドレス可能です。スタイル式が特定の文字列に評価される場合、この文字列が一致するスタイルの検索に使用されます。最初は現在のマップを検索し、これが失敗した場合は現在のレポート上の他のすべてのマップを検索します(一致するタイプのスタイルのみを検索します。たとえば、**MarkerStyles** コレクションはポイントマーカースタイルが検索されるだけです)。

## 空間位置

ポイントおよびラインレイヤは、データの空間位置を指定するために、次の2つの異なる方法を提供します。

- 実行時に経度/緯度ペアに評価されるペア式として。通常、これらはデータソースに格納された対応するデータフィールド(経度と緯度)を直接参照します。
- **MapLocation**として。これは、外部のオンラインサービス(Google マップ)を使用して対応する空間位置の取得に使用できる特定の文字列に評価される式(または式のリスト)です。指定した **MapLocation**がセミコロンを含む場合、セミコロン区切りの式のリストとして処理され、式はそれぞれ、個別に評価された後、クエリーとして使用するために結合されます。標準的な **MapLocation**は以下のようになります。

  ```
  "Address;City;PostalCode;Country"
  ```

  これは、*Address*、*City*、*PostalCode*、*Country* の各フィールドをデータソースから取得した後、外部サービスへのクエリーを実行するためにこれらを結合します。

  Internetアクセスのため **MapLocation**の使用は非常に時間がかかることに注意してください。そのため、デフォルトでは、取得されたデータはローカルディスクファイルに格納されます。このファイルへのパスは **Map.GeoCachePath**プロパティで指定されます。デフォルトでは、ファイルの名前は「geocache.xml」で、レポート定義と同じディレクトリに格納されます。ジオキャッシングの無効化は推奨されません。

## ポイントレイヤ

ポイントレイヤを使用し、ポイント位置マーカーを表示します。データソースのレコードごとに1つのマーカーを表示します。マーカーの位置は、「空間位置」で説明したように、経度/緯度ペア、または **MapLocation**によって指定されます。以下に、ポイントレイヤの重要な側面を説明します。

- **データアクセス**:実行時にポイントレイヤを処理すると、レコードソース(指定した場合はレイヤ独自の **RecordSource**、または現在のグループによってフィルタされたレポートのレコードソース)内をループして、データレコードごとにマークが描画されます。
- **視覚スタイル**:ポイントマーカーの外観は適用されたマーカースタイルによって決定されます。ポイントレイヤは、マーカーの形状、色などを指定できるデフォルトの **MarkerStyle**プロパティを提供します。さらに、**MarkerStyleExpr**式を指定できます。この場合、実行時にデータレコードごとにこの式が評価され、一致するマーカースタイルが現在のマップの **MarkerStyles**コレクション内に見つかったか、レポート内で失敗した場合、デフォルトではなく、このスタイルが適用されます(上述のように、スタイル式はスタイルコレクション内の特定のスタイル名に一致する文字列に評価される必要があります)。
- **クラスタリング**:複数のポイントマーカーが相互に近接して配置されている場合、これらを単一マーカーに「クラスタ化」できます。このマーカーは常に、表しているクラスタ化したポイントマーカーの数を表示します。クラスタマーカーの視覚スタイルは、ポイントマーカーのスタイルと異なるものにすることができ、表しているポイントの数に応じて変えることもできます。クラスタスタイルはポイントレイヤの **ClusterStyles**コレクションによって指定されます。複数のスタイルが提供される場合は、特定のスタイルがクラスタサイズによって決定されます。関連するポイントレイヤのプロパティは、**ClusterDistance**、**ClusterDistribution**、および **ClusterStyles**です。
- **トラッキング**:**Track**プロパティが **True**の場合、自動的な中央揃えとズームはすべてのレイヤのポイントを包含します。


Copyright © GrapeCity, Inc. All rights reserved.

# ラインレイヤ

ラインレイヤを使用し、マップ上のポイント間の線を描画します。データレコードごとに2点を結ぶ1本の線を描画します。各ポイントの空間位置は、「ポイントレイヤ」の場合と同じ方法、つまり2つの **経度／緯度**ペア（ラインの各端に1つ）によるか、オンラインサービスから位置の要求に使用する2つの **MapLocations** によって指定できます。以下に、ラインレイヤの重要な側面を説明します。

- **データアクセス**：ポイントレイヤと同様に、ラインレイヤでは、独自の **RecordSource**を指定するか、現在のグループによってフィルタされたレポートのレコードソースを使用できます。
- **視覚スタイル**：スタイルは大体、ポイントレイヤと同じ方法で処理されますが、**MarkerStyles**ではなく、**LineStyles** コレクションが使用されます。
- **トラッキング**：**Track** プロパティが **True** の場合、自動的な中央揃えとズームはすべてのレイヤのラインを包含します。

# KMLレイヤ

KMLレイヤは、KML（Keyhole マークアップ言語）または KMZ（圧縮した KML）ファイルをマップ上にレンダリングします。ファイル名は、レイヤ上の **KmlFileName**プロパティによって指定されます。このファイルは、URL やローカルディスクファイルからロードしたり、レポートに埋め込むことができます。このファイルが埋め込まれていなくて（**EmbedKmlFileが False**）、ディレクトリが指定されていない場合、このファイルはレポート定義を格納するディレクトリからロードされます。

- **KML 項目式**：KMLレイヤがレンダリングされると、KML ファイル内にある項目が順に処理されます。各項目がロードされると、レイヤ上で指定されたいくつかの式が評価され、処理を制御できます。たとえば、さまざまな条件に基づいて特定の項目だけをロードするか、項目の視覚属性だけを変更します。さらに、KML レイヤの **RecordSource**が 指定された場合、項目式を評価する前に各 KML 項目のデータをフィルタできます。以下に、KML 項目式の評価に関係するプロパティを詳しく説明します。これらのすべての式で、特別な変数 **kmlItemName**を使用でき、現在処理されている KML 項目名を参照することに注意してください。
- **ItemFilterExpr: RecordSource**が KML レイヤ上に指定された場合（のみ）、他の式を評価する前に、このフィルタが取得されたデータに適用されます。たとえば、レイヤのレコードソースが「国」フィールドを含み、KML ファイルが国の項目を含む場合、次のフィルタは
  `kmlItemName=Country`
  KML項目ごとに、他の項目式が現在の項目の国に対応するデータで評価されるようにします。
- **ItemTrackExpr:** 指定された場合、自動的にマップを中央揃え／ズームするために項目が使用されるかどうかを決定します。空のままにすると、true と見なされます。
- **ItemVisibleExpr:** 指定された場合、項目の表示／非表示を決定します。空のままにすると、true と見なされます。
- **ItemStyleExpr:** iこの式が（現在の、またはレポート内の他の任意のマップの） **KmlItemStyles**コレクション内の有効なスタイル名に評価された場合、このスタイルが項目に適用されます。これは、たとえば、州の注文合計などのデータ値に応じて、異なる州を異なる色で塗りつぶす場合に使用できます。
- **ItemStyle.ItemNameExpr:** KML項目スタイルはそれ自身、1つの計算されたプロパティ、つまり項目名を含みます。これにより、マップ上にレンダリングされた名前を隠したり、それをレポートデータ（注文合計など）に置き換えることができます。

# 凡例

マップは、複数の関連した凡例を持つことができ、その境界内にレンダリングされます。凡例をマップの境界の外に簡単に配置できるようにするために、凡例をレポート内の任意のマップフィールドに関連付けることができます。これにより、空のマップフィールドを追加して、別のマップを説明する凡例を保持できます。

凡例は[**マップ**]フィールドの **Legends**コレクション内に格納されます。凡例を追加するには、項目をこのコレクションに追加します。マップの境界内の凡例の位置は **LegendAlignment**プロパティによって決定されます。向きは、凡例内の項目を垂直（デフォルト）または水平に配置するかどうかを決定します。いくつかの他のプロパティにより、凡例の外観を微調整できます。

凡例内の項目は Items コレクションによって表されます。凡例の **Automatic**プロパティが **True**に設定された場合は、このコレクションは現在のマップの KML 以外のレイヤからデータを自動的に移植できます。この場合、Items コレクションを編集できません。それ以外の場合は、凡例項目は手動で追加する必要があります。


Copyright © GrapeCity, Inc. All rights reserved.

次の凡例項目のタイプがサポートされます。:

- **LegendLayerStyleItem**:レイヤスタイルを表します。このデザイナでは、凡例項目によって表される既存のレイヤまたはスタイルを選択できます。選択したレイヤスタイルに応じて、凡例項目はポイントマーカー(ポイントレイヤ/スタイルの場合)、ライン(ラインレイヤ/スタイルの場合)、または色見本(KML 項目スタイルの場合)を表すことができます。
- **LegendColorSwatchItem**: 任意の色見本を表します。
- **LegendTextItem**:任意のテキストを表します。

## マップのウォークスルー

このウォークスルーでは、米国の州の注文合計を州別にまとめて表示するレポートにマップを追加します。マップを含まないレポートは非常に単純で、州ごとのすべての注文の合計をリストしているだけです。以下にその全体を示します。

注文 州名

16325.15 アラスカ

3490.02 カリフォルニア

115673.39 アイダホ

1947.24 モンタナ

52245.9 ニューメキシコ

30393.93 オレゴン

31001.65 ワシントン

12489.7 ワイオミング

マップをこのレポートに追加し、それぞれの州を、緑(注文のない州)から黄と赤の陰影(合計注文額で決定)までの色で塗りつぶします。さらに、各州は合計に比例する直径の円形マーク、およびこの合計を示すラベルを持ちます。最後に、アラスカとハワイを表示する2つの小さいインセット地図を追加します。

このウォークスルーでは次のファイルを使用することに注意してください。

- C1NWindMaps.mdb:空間とその他のデータが追加された C1NWind データベース。
- us_states_abbr.kmz:略称の州名を付けた米国の州境界を含む圧縮 KML ファイル。

以下の手順を実行します。

1. **基本レポートを作成します。**

   デザイナで新しいレポートを追加して、データソースとして **C1NWindMaps.mdb**を指定し、次の SQL クエリーを使用します。

```
SELECT Orders.ShipRegion, Orders.ShipCountry, StateNamesGeo.StateName,
Sum([Order Details].UnitPrice*[Order Details].Quantity) AS OrderValue,
(select Longitude from StateNamesGeo where StateNamesGeo.Abbr = Orders.ShipRegion) as Longitude,
(select Latitude from StateNamesGeo where StateNamesGeo.Abbr = Orders.ShipRegion) as Latitude
FROM ((Categories INNER JOIN Products ON Categories.CategoryID = Products.CategoryID)
INNER JOIN (Orders INNER JOIN [Order Details] ON Orders.OrderID = [Order Details].OrderID)
ON Products.ProductID = [Order Details].ProductID)
INNER JOIN StateNamesGeo on Orders.ShipRegion = StateNamesGeo.Abbr
WHERE Orders.ShipCountry = "USA"
GROUP BY Orders.ShipRegion, Orders.ShipCountry, StateNamesGeo.StateName
ORDER BY Orders.ShipRegion;
```

   レポートウィザードを使用し、**OrderValue** と **StateName** フィールドを[詳細]セクションに追加します。レポートを実行し、上に示したデータを印刷することを確認します。

2. **メインマップを追加します。**

   マップをレポートのヘッダーに追加します。

   - レポートデザイナで、ヘッダーの下端を下方向にドラッグして、マップ用のスペースを空けます。
   - マップカスタムフィールドアイコン(地球儀)をクリックし、ヘッダーにドラッグします。
   - マップのサイズを約 12870 x 7620 twips に設定し、必要に応じて調整します。

3. **マップのプロパティを調整します。**

   マップのプロパティを次のように設定します(デフォルト以外のみを次に示します)。

   - **AutoCenter**: false
   - **AutoZoom**: false
   - **CenterLatitude**: 38
   - **CenterLongitude**: -103


Copyright © GrapeCity, Inc. All rights reserved.

- **ShowScale**: false
- **TileSource**: None
- **ZoomLevel**: 3

米国のマップを表示しているため、必要に応じて、座標を手動で設定する（特に、アラスカとハワイのインセット用に右側に十分なスペースを残す）ことに注意してください。

4. **ポイントマーカースタイルを追加します。**

   **MarkerStyles** コレクションを開き、1つのスタイルを追加します。そのコントロールのプロパティを次のように設定します。

   - **CaptionExpr**: StateName & ":" & vbCr & "$" & OrderValue
   - **FillColor**: 120, 255, 128, 0
   - **Name**: msTotalSales
   - **SizeExpr**: sqr(OrderValue / 100)

   このスタイルは、注文合計を示すサイズの円形マーカーを州上に描画するために使用されます。塗りつぶし色はこの場合に効果的な半透明であることに注意してください。また、マーカーのサイズが注文額（合計）の平方根に比例することにも注意してください。スタイル名（msTotalSales）はこのスタイルを参照するために使用されます。

   必要に応じて、他のプロパティ（フォント、ストローク、およびテキスト色）も設定できます。

5. **KML項目スタイルを追加します。**

   注文合計に応じて、すべての州を6つのグループに分けます。

   - 注文が全くない州
   - 注文合計が最大1万ドルの州
   - 注文合計が1万ドル～3万ドルの州
   - 注文合計が3万ドル～5万ドルの州
   - 注文合計が5万ドル～10 万ドルの州
   - 注文合計が 10 万ドルを超える州

   したがって、これらのグループのそれぞれに KML 項目スタイルを作成する必要があります。**KmlItemStyles**コレクションを開き、上記のグループに対応する名前を持つ6つのスタイル、およびマップ上の州を区別する **FillColor**値を追加します。

   - **Name**: ksNoOrders, **FillColor**: 143, 188, 139
   - **Name**: ks0k10k, **FillColor**: 255, 250, 205
   - **Name**: ks10k30k, **FillColor**: 255, 222, 173
   - **Name**: ks30k50k, **FillColor**: 255, 160, 122
   - **Name**: ks50k100k, **FillColor**: 205, 92, 92
   - **Name**: ks100kup, **FillColor**: 178, 34, 34

   これらの名前は、ここでは非常に重要です。KML 項目式でこれらの名前を使用し、州の注文合計値に応じてスタイルを選択するからです。

6. **KML レイヤを追加します。**

   マップの最も重要な部分は、州境界を表示して適切な色で州を塗りつぶす KML レイヤです。これを追加するには、マップの **Layers**コレクションエディタに移動し、KML レイヤを追加します。そのコントロールのプロパティを次のように設定します。

   - **KmlFileName:** us_states_abbr.kmz（パス付きでないファイル名を指定すると、レポート定義と同じ場所からファイルがロードされます。
   - **RecordSource:**

     ```
     SELECT Orders.ShipRegion,
         Sum([Order Details].UnitPrice*[Order Details].Quantity) AS OrderValue
         FROM (Categories INNER JOIN Products ON Categories.CategoryID = Products.CategoryID)
         INNER JOIN (Orders INNER JOIN [Order Details] ON Orders.OrderID = [Order Details].OrderID)
         ON Products.ProductID = [Order Details].ProductID
         WHERE Orders.ShipCountry = "USA"
         GROUP BY Orders.ShipRegion
         ORDER BY Orders.ShipRegion;
     ```

   - **ItemFilterExpr**: kmlItemName=ShipRegion（KML 項目式、特に、KML 項目の描画に使用するスタイルが現在の KML 項目に対応する州のデータで評価されるようにします）。
   - **KmlVisibleExpr:** kmlItemName<>"AK"（メインマップがアラスカを表示しないようにします）。
   - **ItemStyleExpr**:

     ```
     iif (OrderValue > 100000, "ks100kup",
         iif (OrderValue > 50000, "ks50k100k",
         iif (OrderValue > 30000, "ks30k50k",
         iif (OrderValue > 10000, "ks10k30k",
     ```


Copyright © GrapeCity, Inc. All rights reserved.

```
iif (OrderValue > 0, "ks0k10k",
"ksNoOrders"
)))))
```

上の式は、**RecordSource**どおり、1つの州の全注文の合計である現在の **OrderValue**に応じて、KML 項目スタイル名（上を参照）の1つに評価されます。前記の **ItemFilterExpr**は、この式が KML ファイルから現在ロード中の州に対して評価されるようにします。

7. **円形マーカーを追加します。**

マップの視覚的効果をさらに高めるには、州の合計注文に比例してサイズ変更して州上に配置されたた円形マーカーを含むポイントレイヤも追加します。それには、**PointsLayer**を **Layers**コレクションに追加し、KML レイヤの上に移動して、マップがレンダリングされるときにポイントレイヤが KML レイヤの上に表示されるようにします。レイヤのプロパティを次のように設定します。

**MarkerStyleExpr:** 「msTotalSales」（これは既にマップの MarkerStyles コレクションに追加したスタイルを使用します。比例的なサイズ変更はこのスタイルに含まれているため、ここでは他に何もする必要がありません）。

**Latitude:** 緯度（レコードソースによって提供される空間データを使用します）。

**Longitude:** 経度（レコードソースによって提供される空間データを使用します）。

8. **凡例を追加します。**

タイトルを表す凡例（右上端）、および KML 項目スタイルへの色キー（右下）の2つの凡例をマップに追加します。

- 凡例を追加するには、**Legends**コレクションエディタを開きます。
- タイトルを追加するには、項目を追加して、**Caption**を「州別の注文合計」に設定し、**LegendAlignment**をデフォルトの **TopRight**値のままにします。他のプロパティを適当と思うように調整します。
- 色キーを追加するには、凡例をもう1つ追加して、配置を **BottomRight**に設定し、その **Items**コレクションを開き、次の項目を追加します。
  - **[Text]項目。**テキスト「色キー」を含み、凡例のキャプションとなります。
  - 6[**LayerStyle**]項目。KML 項目スタイルのそれぞれに1つ追加します。項目ごとに、**LayerStyle**ドロップダウンから、その項目が記述するスタイルを選択します。これにより、大部分の他のプロパティに選択されたスタイルからの値が自動的に入力されます。手動で設定する必要があるのは、項目のテキストだけです。「注文なし」から「3万ドル未満」、「1万ドル～3万ドル」というように「10万ドル以上」までを、スタイルごとに適切に設定します。

9. **インセット地図を追加します。**

アラスカとハワイの2つのインセット地図を追加します。それには、次の手順を実行します。

- [マップ]カスタムフィールドアイコンをクリックし、アラスカのメインマップの左上部分とハワイの左下部分に、2つの小さいマップを描画します。

  インセット地図はどちらもメインマップで定義されたスタイルを再使用するため、スタイルをインセット地図のスタイルコレクションに追加する必要はありません。

- レイヤはメインマップのレイヤを複製します。したがって、2つのレイヤを各インセット地図の Layers コレクションに追加します。
  - ポイントレイヤ。MarkerStyleExpr は「msTotalSales」に設定され、MarkerVisibleExpr は

    ```
    StateName = "Alaska"
    ```

    （ハワイインセットの場合は「Hawaii」）のように設定され、すべての他のプロパティはメインマップのポイントレイヤと同様です。
  - KMLレイヤ。すべてのプロパティ（RecordSource を含む）はメインマップの KML レイヤの対応するプロパティからコピーされますが、ItemVisibleExpr は

    ```
    kmlItemName="AK"
    ```

    （ハワイの場合は kmlItemName="HI"）に設定されます。特に、**ItemStyleExpr**は、評価されるスタイル名が別のマップのスタイルコレクション内のスタイルを参照する場合でも機能することに注意してください。
- 最後に、州の名前を使用して、単一の固定テキストを持つ凡例を各インセット地図に追加します。

  これで完了です。では、レポートを実行して、正しく機能することを確認します。


Copyright © GrapeCity, Inc. All rights reserved.

# レポート内バーコードの使用

**Reports for WinForms** のバーコード機能を使用すると、さまざまな業界標準のバーコードをレポート内のバーコードフィールドに統合できます。バーコードフィールドはすばやく簡単に生成できます。バーコードフィールドをレポートにドロップし、バーコードシンボルを選択し、テキストを指定すれば完了です。

**Reports for WinForms** のバーコード機能は、バーコードに関連付けられたプロパティによって拡張されています。読み取りエラーをなくすために、チェックサムを自動的に追加しながら値をエンコードしています。

> 2015 v2 リリース以降では、新しいバーコードエンジンが Reports に追加されています。新しいレポートで古いバーコードの互換性を維持するには、**UseCompatibleBarcode** プロパティをデフォルト値 true で使用します。新しいバーコードの場合、**UseCompatibleBarcode** プロパティのデフォルト値は false になります。

# バーコードシンボル

バーコードシンボルは、文字データをバーコードの太いバー、細いバー、およびスペースから成るパターンに変換するために使用されるエンコードスキームを指定します。**Report for WinForms** で使用されるバーコードシンボルを次の表に示します。

| スタイル名 | 例 | 説明 |
|---|---|---|
| Ansi39 | | ANSI 3 of 9（Code 39）は、大文字、数字、- , * $ / + % を使用します。これがデフォルトのバーコードスタイルです。 |
| Ansi39x | | ANSI Extended 3 of 9（Extended Code 39）は、ASCII 文字セット全体を使用します。 |
| Codabar | | Codabar は、A B C D + - : . / $ および数字を使用します。 |
| Code_128_A | | Code 128 A は、制御文字、数字、句読点、大文字を使用します。 |
| Code_128_B | | Code 128 B は、数字、句読点、大文字、小文字を使用します。 |
| Code_128_C | | Code 128 C は数字だけを使用します。 |
| Code_128auto | | Code 128 Auto は、ASCII 文字セット全体を使用します。最小のバーコードになるように、Code 128 A、B、および C から 1 つが自動的に選択されます。 |



Copyright © GrapeCity, Inc. All rights reserved.

| | | |
|---|---|---|
| Code_2_of_5 | | Code 2 of 5 は数字だけを使用します。 |
| Code93 | | Code 93 は、大文字、数字、% $ * / , + - を使用します。 |
| Code25intlv | | Interleaved 2 of 5 は数字だけを使用します。 |
| Code39 | | Code 39 は、大文字、数字、% * $ /. , - + を使用します。 |
| Code39x | | Extended Code 39 は、ASCII 文字セット全体を使用します。 |
| Code49 | | Code 49 は、各段 8 文字の 2 ～ 8 段から成る 2 次元高密度スタック型バーコードです。各段には、スタートコードとストップコードが含まれます。ASCII 文字セット全体をエンコードします。 |
| Code93x | | Extended Code 93 は、ASCII 文字セット全体を使用します。 |
| DataMatrix | | Data Matrix は、正方形または長方形のマトリックスパターンに並べられた正方形モジュールから成る 2 次元高密度バーコードです。 |
| EAN_13 | | EAN-13 は、数字（12 個の数字とチェックデジット）だけを使用します。12 個の数字だけを文字列として受け取ってチェックデジット（CheckSum）を計算し、それを 13 番目の位置に追加します。チェックデジットは、バーコードが正しくスキャンされたかどうかを検証するために使用される追加の桁です。チェックデジットは、CheckSum プロパティが true に設定されると自動的に追加されます。 |
| EAN8 | | EAN-8 は、数字（7 個の数字とチェックデジット）だけを使用します。 |


Copyright © GrapeCity, Inc. All rights reserved.

| | | |
|---|---|---|
| EAN128FNC1 | | EAN-128 は、運送業で梱包物を識別するためのアプリケーション識別子（AI）データの 1 次元英数字表現です。<br><br>このタイプのバーコードには、以下のセクションが含まれています。<br><br>● 先頭のクワイエットゾーン（空白領域）<br>● Code 128 のスタート文字<br>● スキャナに EAN-128 バーコードであることを識別させる FNC（ファンクション）1 文字<br>● データ（AI + データフィールド）<br>● シンボルチェック文字（「スタートコード値 + 各文字位置と各文字値の積」を 103 で除算。この剰余値がチェックサム）。<br>● ストップ文字<br>● 末尾のクワイエットゾーン（空白領域）<br><br>データセクションの AI は、後に続くデータのタイプ（ID、日付、数量、計量単位など）を設定します。データのタイプごとに固有のデータ構造があります。この AI があることが、EAN-128 コードが Code 128 と異なる点です。<br><br>1 つのバーコードで複数の AI（およびそのデータ）を組み合わせることができます。<br><br>EAN128FNC1 は UCC/EAN-128（EAN128）バーコードタイプの 1 つで、任意の場所に FNC1 文字を挿入したり、バーサイズを調整することができるなど、UCC/EAN-128 にはない機能をサポートしています。<br><br>FNC1 文字を挿入するには、実行時に Text プロパティに "\n"（C# の場合）または "vbLf"（VB の場合）を設定します。 |
| IntelligentMail | | Intelligent Mail（以前の 4-State Customer Barcode）は、米国内の郵便に使用される 65 本のバーコードです。 |
| JapanesePostal | | これは、日本の郵便制度で使用されるバーコードです。18 桁の英数字をエンコードします。これには、7 桁の郵便番号の後に、必要に応じて番地や部屋番号などの情報が含まれます。エンコードするデータにはハイフンを含めることができます。 |


Copyright © GrapeCity, Inc. All rights reserved.

| | | |
|---|---|---|
| Matrix_2_of_5 | | Matrix 2 of 5 は、3 本の黒バーと 2 本の白バーから成る比較的高密度のバーコードです。 |
| MicroPDF417 | | MicroPDF417 は、PDF417 から派生された 2 次元多段バーコードです。Micro-PDF417 は、データをできる限り小さなサイズの 2 次元シンボル（最大 150 バイト、英数字 250 文字、数字 366 桁）にエンコードする必要があるアプリケーション用に設計されています。<br>MicroPDF417 では、可変長のアプリケーション識別子（AI）のフィールド区切り文字として FNC1 文字を挿入することができます。<br>FNC1 文字を挿入するには、実行時に Text プロパティに "\n"（C# の場合）または "vbLf"（VB の場合）を設定します。 |
| MSI | | MSI Code は数字だけを使用します。 |
| Pdf417 | | Pdf417 は、よく使用されている 2 次元高密度バーコードです。最大 1,108 バイトの情報をエンコードできます。このバーコードは、小さなバーコードを積み重ねて構成されます。ASCII 文字セット全体をエンコードします。10 のエラー修正レベルと 3 つのデータコンパクションモード（Text、Byte、および Numeric）があります。このバーコードシンボルは、英数字で最大 1,850 文字、数字で最大 2,710 桁をエンコードできます。 |
| PostNet | | PostNet は、数字とチェックデジットを使用します。 |
| QRCode | | QRCode は、英数字やバイトデータに加えて、日本語の漢字やかな文字も扱うことができる 2 次元バーコードです。このバーコードは最大 7,366 文字をエンコードできます。 |
| RM4SCC | | Royal Mail RM4SCC は、文字と数字（チェックデジット付き）だけを使用します。これは、イギリスのロイヤルメールで使用されるバーコードです。 |
| RSS14 | | RSS14 は、全方向型 POS スキャンに EAN.UCC 品目識別を使用する 14 桁の省スペースシンボル（Reduced Space |


Copyright © GrapeCity, Inc. All rights reserved.

| | | |
|---|---|---|
| | | Symbology：RSS）です。 |
| RSS14Stacked | | RSS14Stacked は、RSS14Truncated と同様にインジケータデジットを含む EAN.UCC 情報を使用しますが、幅が狭い場合、2 段に積み重ねられます。<br>RSS14Stacked では合成オプションを設定できます。**[タイプ]**ドロップダウンリストからバーコードのタイプを選択し、**[値]**フィールドで合成バーコードの値を指定できます。 |
| RSS14StackedOmnidirectional | | RSS14StackedOmnidirectional は、RSS14 と同様に全方向型スキャンを含む EAN.UCC 情報を使用しますが、幅が狭い場合、2 段に積み重ねられます。 |
| RSS14Truncated | | RSS14Truncated は、RSS14 と同様に EAN.UCC 情報を使用しますが、POS でスキャンされない小さな品目に使用するために、0 または 1 のインジケータデジットを含みます。 |
| RSSExpanded | | RSSExpanded は、RSS14 と同様に EAN.UCC 情報を使用しますが、重量や賞味期限などの AI 要素を追加します。<br>RSSExpanded では、可変長のアプリケーション識別子（AI）のフィールド区切り文字として FNC1 文字を挿入することができます。<br>FNC1 文字を挿入するには、実行時に Text プロパティに "\n"（C# の場合）または "vbLf"（VB の場合）を設定します。 |
| RSSExpandedStacked | | RSSExpandedStacked は、RSSExpanded と同様に AI 要素を含む EAN.UCC 情報を使用しますが、幅が狭い場合、2 段に積み重ねられます。<br>RSSExpandedStacked では、可変長のアプリケーション識別子（AI）のフィールド区切り文字として FNC1 文字を挿入することができます。<br>FNC1 文字を挿入するには、実行時に Text プロパティに "\n"（C# の場合）または "vbLf"（VB の場合）を設定します。 |
| RSSLimited | | RSS Limited は、RSS14 と同様に EAN.UCC 情報を使用しますが、POS でス |


Copyright © GrapeCity, Inc. All rights reserved.

| | | |
|---|---|---|
| | | キャンされない小さな品目に使用するために、0 または 1 のインジケータデジットを含みます。<br>RSSLimited では合成オプションを設定できます。**[タイプ]**ドロップダウンリストからバーコードのタイプを選択し、**[値]**フィールドで合成バーコードの値を指定できます。 |
| UCCEAN128 | | UCC/EAN –128 は、ASCII 文字セット全体を使用します。HIBC アプリケーションで使用される特殊な Code 128 です。 |
| UPC_A | | UPC-A は、数字（11 個の数字とチェックデジット）だけを使用します。 |
| UPC_E0 | | UPC-E0 は数字だけを使用します。Zero Suppression の UPC シンボルに使用されます。Caption プロパティには、6 桁の UPC-E コードまたは完全な 11 桁（必須のコードタイプ 0 を含む）の UPC-A コードを入力する必要があります。11 桁のコードが入力された場合、バーコードコントロールは、可能であればそれを 6 桁の UPC-E コードに変換します。11 桁のコードを 6 桁のコードに変換できない場合は、何も表示されません。 |
| UPC_E1 | | UPC-E1 は数字だけを使用します。主に小売業界で在庫ラベルに使用されます。<br>UPC-E1 の入力文字列の長さは数字 6 個です。 |

バーコードフィールドは、プロパティウィンドウで **BarCode** プロパティを使用することで、**C1ReportDesigner** に直接挿入できます。**Barcode** を使用して、バーコードフィールドのバーコードのタイプを設定することもできます。

以下のバーコードは、FNC1 文字をサポートしています。

- EAN128FNC1
- MicroPDF417
- RSSExpanded
- RSSExpandedStacked

## バーコードのプロパティ

**BarcodeOptions** は、**Reports for WinForms** でバーコードをレンダリングするためのプロパティを追加します。**BarcodeOptions** から公開される共通のプロパティは次のとおりです。

- **BarDirection**：バーコードの方向（水平または垂直）を選択します。使用可能なオプションは、LeftToRight、RightToLeft、TopToBottom、BottomToTop です。バーコードの方向は、**BarDirectionEnum** を使用して設定することもできます。
- **CaptionGrouping**：キャプションのテキストを、サポートするバーコードタイプに応じて複数のグループに分割します。値は True または False です。
- **CaptionPosition**：バーコードシンボルに対するキャプションの相対的な垂直位置を選択します。使用可能なオプションは、None、Above、Below です。


Copyright © GrapeCity, Inc. All rights reserved.

- **CheckSumEnabled**：適切な場合にバーコードのチェックサムを計算してバーコードに含めるかどうかを指定します。
- **TextAlign**：キャプションテキストの配置を選択します。使用可能なオプションは、Left、Center、Right です。
- **SupplementNumber**：バーコードデータの補助コードを指定します。補助コードは、EAN または UPC シンボルの場合、2 桁または 5 桁です。
- **SizeOptions**：
  - **BarHeight**：バーコードの高さを指定します（ツイップ単位）。
  - **ModuleSize**：バーコードのモジュール（最も細いバーの幅）を指定します（ツイップ単位）。
  - **NarrowWideRatio**：細いバーの幅と太いバーの幅の比率を指定します。
  - **SizeMode**：バーコードのサイズ変更モードを指定します。使用可能なオプションは次のとおりです。
    - **Normal**：元のサイズと同じバーコードのサイズを維持します。
    - **Scale**：できるだけ広いフィールド領域を使用するようにバーコード画像を拡大縮小します。拡大縮小の方法は、バーコードタイプによって異なります。たとえば、Code128 のような Bar タイプのバーコードでは高さが拡大され、Matrix、Rss、Composite などのバーコードでは高さと幅が同じ割合で拡大縮小されます。
  - **SupplementSpacing**：メインバーコードと補助バーコードの間の間隔を指定します。

さまざまなバーコードスタイルに応じて **BarcodeOptions** から公開されるその他のオプションは次のとおりです。

**Code49:**

- **Grouping**：バーコードでグループ化を使用します。値は True または False です。
- **Group**：バーコードをグループ化する際のグループ番号を取得または設定します。有効な値は 0 ～ 8 です。

**DataMatrix**：

- **EccMode**：ECC モードを選択します。有効な値は、ECC000、ECC050、ECC080、ECC100、ECC140、ECC200 です。
- **Ecc200SymbolSize:** ECC200 シンボルのサイズを選択します。デフォルト値は SquareAuto です。
- **Ecc200EncodingMode:** ECC200 エンコーディングモードを選択します。有効な値は、Auto、ASCII、C40、Text、X12、EDIFACT、Base256 です。
- **Ecc000_140SymbolSize**：ECC000_140 シンボルのサイズを選択します。
- **StructuredAppend**：現在のバーコードシンボルが構造的連接シンボルの一部かどうかを選択します。
- **StructureNumber**：構造的連接シンボル内の現在のシンボルの構造番号を指定します。
- **FileIdentifier**：関連する構造的連接シンボルグループのファイル識別子を指定します。有効なファイル識別子値は、[1,254] 以内である必要があります。ファイル識別子を 0 に設定すると、ファイル識別子が自動的に計算されます。

**GS1Composite:**

- **Type**：合成シンボルタイプを選択します。選択できる値は None または CCA です。CCA（Composite Component - Version A）は、最小タイプの 2 次元コンポジットコンポーネントです。
- **Value**：CCA 文字データを取得または設定します。

**MicroPDF417:**

- **CompactionMode**：CompactionMode のタイプを選択します。有効な値は、Auto、TextCompactionMode、NumericCompactionMode、ByteCompactionMode です。
- **FileID**：構造的連接シンボルのファイル ID を指定します。0 ～ 899 の値を取ります。
- **SegmentCount**：構造的連接シンボルのセグメント数を指定します。0 ～ 99999 の値を取ります。
- **SegmentIndex**：構造的連接シンボルのセグメントインデックスを指定します。0 ～ 99998 の範囲で、セグメント数の値より小さい値を取ります。
- **Version**：シンボルサイズを選択します。デフォルト値は ColumnPriorAuto です。

**PDF417:**

- **Column**：バーコードの列数を指定します。
- **Row**：バーコードの行数を指定します。
- **ErrorLevel**：バーコードのエラー修正レベルを指定します。
- **Type**：PDF417 バーコードのタイプを選択します。使用可能なタイプは Normal および Simple です。


Copyright © GrapeCity, Inc. All rights reserved.

**QRCode**

- **Model**: QRCode のモデルを選択します。使用可能なモデルは Model1 および Model2 です。
- **ErrorLevel**: バーコードのエラー修正レベルを選択します。使用可能なオプションは、Low、Medium、Quality、High です。
- **Version**: バーコードのバージョンを指定します。
- **Mask**: バーコードのマスクに使用されるパターンを選択します。QRCode が正しく読み取られるようにするには、マスク処理で明度のバランスを取る必要があります。使用可能なオプションは、Auto、Mask000、Mask001、Mask010、Mask011、Mask100、Mask101、Mask110、Mask111 です。
- **Connection**: バーコードにコネクションを使用するかどうかを選択します。値は True または False です。
- **ConnectionNumber**: バーコードのコネクション番号を指定します。0 ～ 15 の範囲の整数値を取ります。

**RssExpandedStacked**:

- **RowCount**: 積み重ねる段数を指定します。

バーコードのクワイエットゾーンは、**C1ReportDesigner** のプロパティウィンドウで MarginBottom、MarginLeft、MarginRight、および MarginTop プロパティを使用して簡単に指定できます。


Copyright © GrapeCity, Inc. All rights reserved.

# レポートとドキュメントビューアの概要

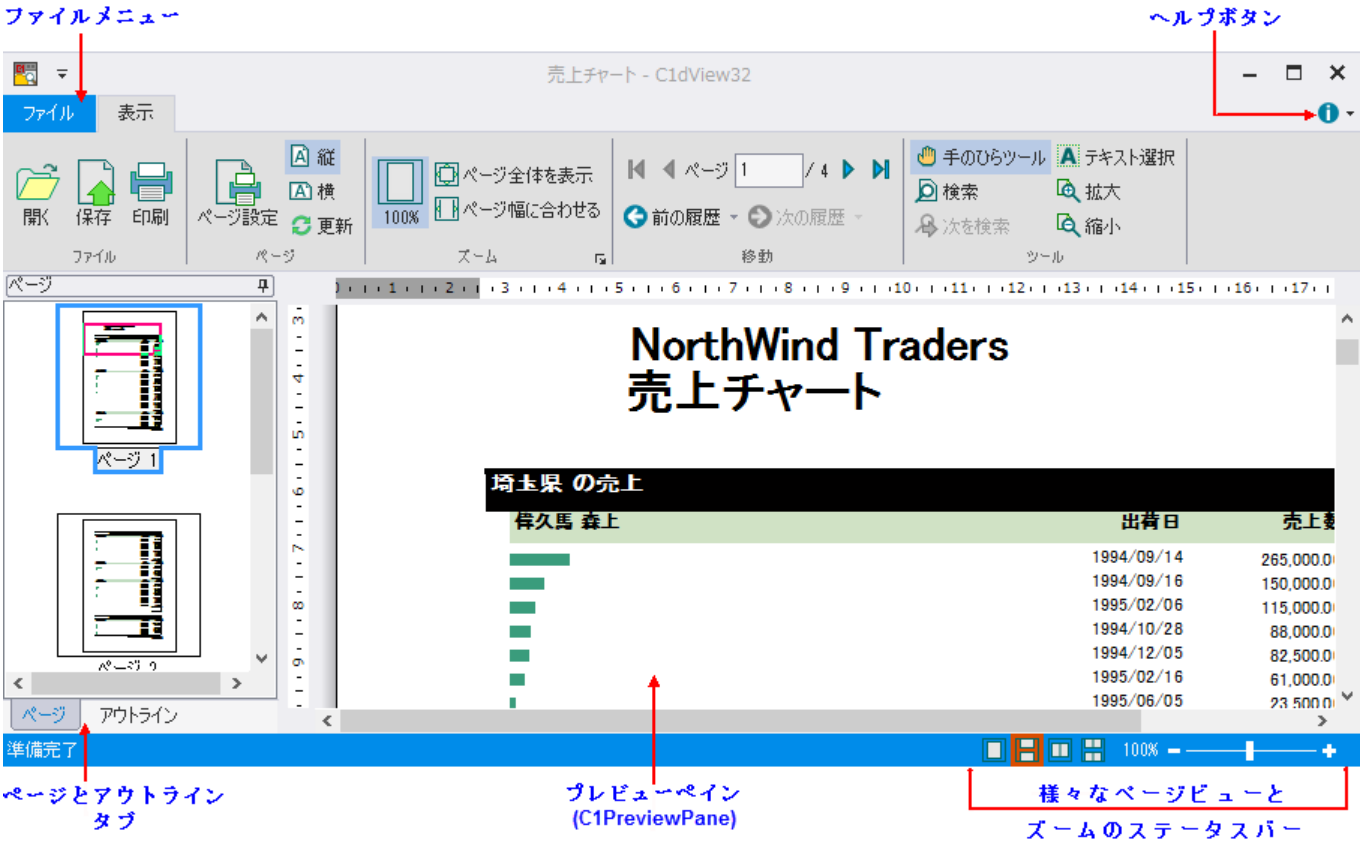
**ComponentOne Report and Document Viewer**（C1dView.exe と C1dView32.exe）は、ComponentOne レポートの生成と表示、または以前に保存した ComponentOne ドキュメント（.c1d、.c1dx、.c1md、.c1db）の表示に使用できる Windows アプリケーションです。

**ComponentOne Report and Documenter ビューア**（C1dView.exe と C1dView32.exe）のユーザーインタフェースは、他の2つの ComponentOne 製品（C1Ribbon と C1Command）に基づいています。

32 ビットプラットフォームと 64 ビットプラットフォームの一方でしか利用できないデータベースプロバイダもあるので、任意の CPU をターゲットにした **C1dView.exe** と x86 をターゲットにした **C1dView32.exe** の2つのアプリケーション実行ファイルが用意されています。特に、64 ビットの .mdb ドライバはないため、64 ビットシステムで .mdb ベースのレポートを開く（生成および表示）場合は、C1dView32.exe を使用してください。

アプリケーションを実行するには、デフォルトで **C:\Program Files (x86)\ComponentOne\Apps\v4** にある **C1dView.exe** ファイルをダブルクリックします。このディレクトリはデフォルトのインストールパスに基づいていますが、インストールパスを変更した場合は、パスが異なる場合があります。

**CommonTasks.xml** ファイルを **C:\Users\<ユーザー名>\ComponentOne Samples\WinForms\C1Report\C1Report\XML** から開いた場合、アプリケーションは次のように表示されます。アプリケーション名 C1dView32 の左にレポートの名前が表示されます。




表示できるレポートのタイプがファイル内に複数ある場合は、**[表示するレポートの選択]**ダイアログボックスが表示され、表示するレポートのタイプを選択できます。**[表示するレポートの選択]**ダイアログボックスには、xml レポート定義ファイル内にあるレポートがリストされます。


Copyright © GrapeCity, Inc. All rights reserved.

Designer のメインウィンドウには、次のコンポーネントがあります。

- **[ファイル]メニュー**:ファイルメニューは、レポート定義ファイルのロードと保存、レポート定義のインポートとエクスポートを行うオプションを提供します。詳細については、「ファイルメニュー」を参照してください。
- **[表示]タブ**:[ファイル]、[ページ]、[ズーム]、[ナビゲーション]、[ツール]の各メニュー機能のショートカットを提供します。詳細については、[表示]タブを参照してください。
- **[プレビュー]タブ**:ドキュメント/レポートのプレビューを表示します。ドキュメント/レポートを開くと、それがプレビューペイン領域に表示されます。
- **[ヘルプ]ボタン**:オンラインソースを開いたり、アプリケーションに関する情報を表示する**[バージョン情報]**画面を表示するためのオプションを提供します。
- **[ページ]タブ**:このタブには、ドキュメント内のすべてのページのサムネイルが含まれます。ドキュメントのすべてのページのサムネイルを作成するには、**C1PreviewThumbnailView** クラスが使用されます。
- **[アウトライン]タブ**:このタブには、ドキュメント/レポートのすべてのページのアウトラインビューが含まれます。ドキュメントのすべてのページのアウトラインビューを作成するには、**C1PreviewOutlineView** クラスが使用されます。
- **ステータスバー**:ステータスバーには、単一ページビュー、連続ビュー、見開きページビュー、連続見開きページビューなど、ドキュメントをさまざまなページビューで表示するリボンボタンが含まれます。ステータスバーの右にあるズームスライダをドラッグして、選択したレポートをズームアウト/ズームインすることができます。

以下のトピックでは、**ComponentOne Report and Document Viewer** アプリケーションを使用して、ComponentOne レポートを生成および表示したり、以前保存した ComponentOne ドキュメント(.c1d、.c1dx、.c1md、.c1db)を表示する方法について説明します。

## [ファイル]メニュー

**[ファイル]**メニューは、レポート定義ファイルのロードと保存、およびレポート定義のインポートとエクスポートを行うためのショートカットメニューを提供します。**[ファイル]**メニューから、**C1ReportDesigner** アプリケーションのオプションにもアクセスできます。

**[ファイル]**をクリックすると、メニュー画面が開きます。次のようになります。


Copyright © GrapeCity, Inc. All rights reserved.

メニューには、次のオプションがあります。

- **開く**: **[開く]**ダイアログボックスを表示します。ここで、既存のレポートまたはドキュメントを選択して開くことができます。
- **閉じる**: 現在のレポートドキュメントファイルを閉じます。
- **保存**: 現在のレポートまたはドキュメントを保存またはエクスポートします。
- **印刷**: 現在のレポートまたはドキュメントを印刷します。
- **ページ設定**: [ページ設定]ダイアログボックスを開きます。ここで、現在のレポートまたはドキュメントのページ設定を変更できます。
- **新規ウィンドウ**: 新しいウィンドウを開きます。
- **最近使用したドキュメント**: 最近開いたドキュメントをリストします。ドキュメントを再度開くには、リストから選択します。
- **終了**: **C1dView** アプリケーションを閉じます。

# [表示]タブ

**[表示]**タブは **C1dView** のリボンメニューに配置され、ファイル、ページ、ズーム、ナビゲーション、ツールの各メニュー機能のショートカットを提供します。詳細については、次のトピックを参照してください。

- [ファイル]グループ
- [ページ]グループ
- **[ズーム]グループ**
- [ナビゲーション]グループ
- [ツール]グループ

# [ファイル]グループ

**[表示]**タブの**[ファイル]**グループは、次のように表示されます。

次のオプションで構成されます。

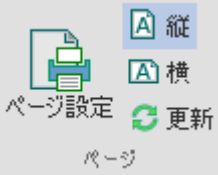
- **開く**: レポートまたはドキュメントを開きます。
- **保存**: 現在のレポートまたはドキュメントを保存またはエクスポートします。


Copyright © GrapeCity, Inc. All rights reserved.

- **印刷**: 現在のレポートまたはドキュメントを印刷します。

## [ページ]グループ

**[表示]**タブの**[ページ]**グループは、次のように表示されます。

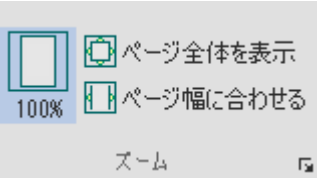



次のオプションで構成されます。

- **ページ設定**: [ページ設定]ダイアログボックスを開きます
- **縦**: レポートまたはドキュメントの方向を縦に変更します。
- **横**: [レポートパラメータの入力]ダイアログボックスを開きます。ここで、現在のレポートまたはドキュメントの方向を横に変更する前に、設定を指定できます。
- **更新**: 現在のドキュメントをリフレッシュするか、レポートを再生成します。

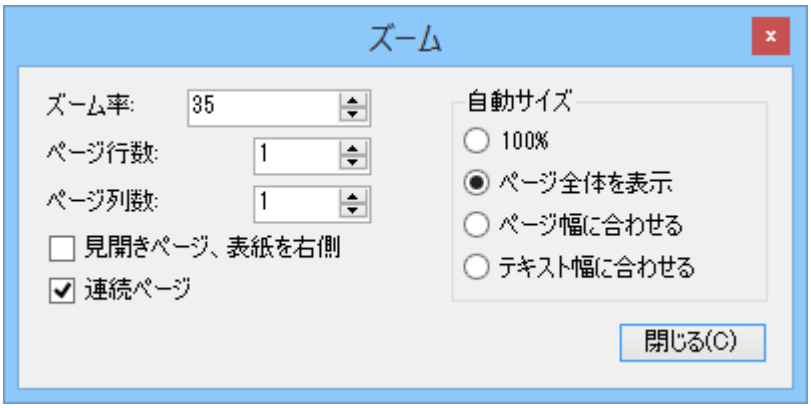
## [ズーム]グループ

**[表示]**タブの**[ズーム]**グループは、次のように表示されます。




次のオプションで構成されます。

- **100%**:ドキュメントを通常サイズの 100% にズームします。
- **ページ全体を表示**:ページをズームしてウィンドウに合わせます。
- **ページ幅に合わせる**:ページをズームしてウィンドウの幅に合わせます。
- **ズーム**:次のような[ズーム]ダイアログボックスを開きます。



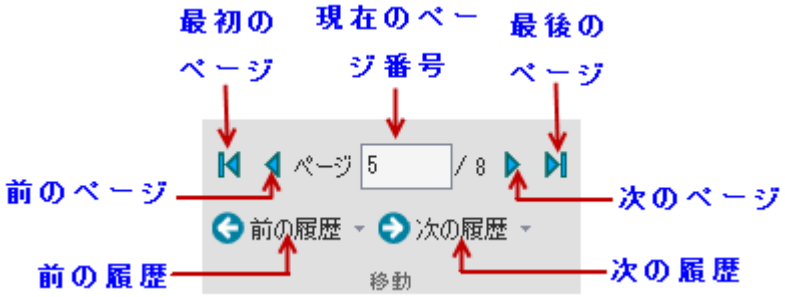

ここで、ズーム率、ページ行数、ページ列数、レポートまたはドキュメントを見開きページビューモードにするかどうか、レポートまたはドキュメントを連続見開きページビューモードにするかどうかを指定できます。

[自動サイズ]グループボックスでは、ドキュメントまたはレポートのズームを[100%]、[ページ全体を表示]、[ページ幅に合わせる]、[テキスト幅に合わせる]のいずれかに指定できます。


Copyright © GrapeCity, Inc. All rights reserved.

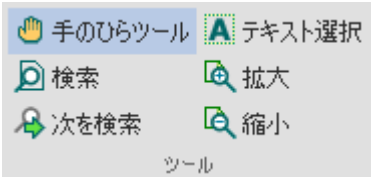
# [移動]グループ

**[表示]**タブの**[移動]**グループは、次のように表示されます。




# [ツール]グループ

**[表示]**タブの**[ツール]**グループは、次のように表示されます。




次のオプションで構成されます。

- **手のひらツール**:マウスを使用してウィンドウ内のドキュメントをスクロールします。
- **検索**: テキストボックスにテキストを入力してドキュメント/レポート内のテキストを検索するための検索バーを開きます。
- **次を検索**:検索テキストの次の項目を検索します。
- **テキスト選択**:ドキュメント内でコピーするテキストを選択します。
- **拡大**:マウスで選択した領域までズームインします。
- **縮小**:マウスで選択した領域までズームアウトします。


Copyright © GrapeCity, Inc. All rights reserved.

# ReportPreview コントロールの概要

2015 v2 リリースでは、**レポートプレビュー**コントロールがベータ版として導入されました。このコントロールは、モダン UI、複数のドキュメントまたはレポートを表示する機能など、完全な機能を備えた単体ビューアです。新しいリボン要素、グループ、サイドバーなどが、カスタマイズ用にユーザーに公開されている API と共に導入されています。

**レポートプレビュー**コントロールは、さまざまなソースから取得したドキュメントやレポートを 1 つのビューアに表示したい場合に、これまで複雑だった実装を簡略化することを目的としています。

Visual Studio にこのコントロールを追加するには、**C1.Win.C1RibbonPreview.4.dll** をツールボックスに追加します。

## ReportPreview コントロールの注目点と機能

### 複数のドキュメントタイプのプレビュー

**C1Report**、**SSRS**（ページ区切り付きモードとページ区切りなしモードの両方）、**C1Document** などの複数のドキュメントまたはレポートタイプをさまざまなソースから 1 つのビューアに表示できます。

### 複数の形式のエクスポート

複数のエクスポート形式オプション（PDF、Doc、Excel（.xls、.xlsx）、MHTML、CSV、Tiff、bmp、emf、gif、jpeg、png、C1 Open XML、C1 バイナリドキュメント、C1 ドキュメント、Open XML Excel、Open XML Word、拡張メタファイル、XML、リッチテキストフォーマット、XPS など）がサポートされているため、より柔軟にドキュメントを保存および共有できます。

### 完全な機能を備えたプレビューコントロール

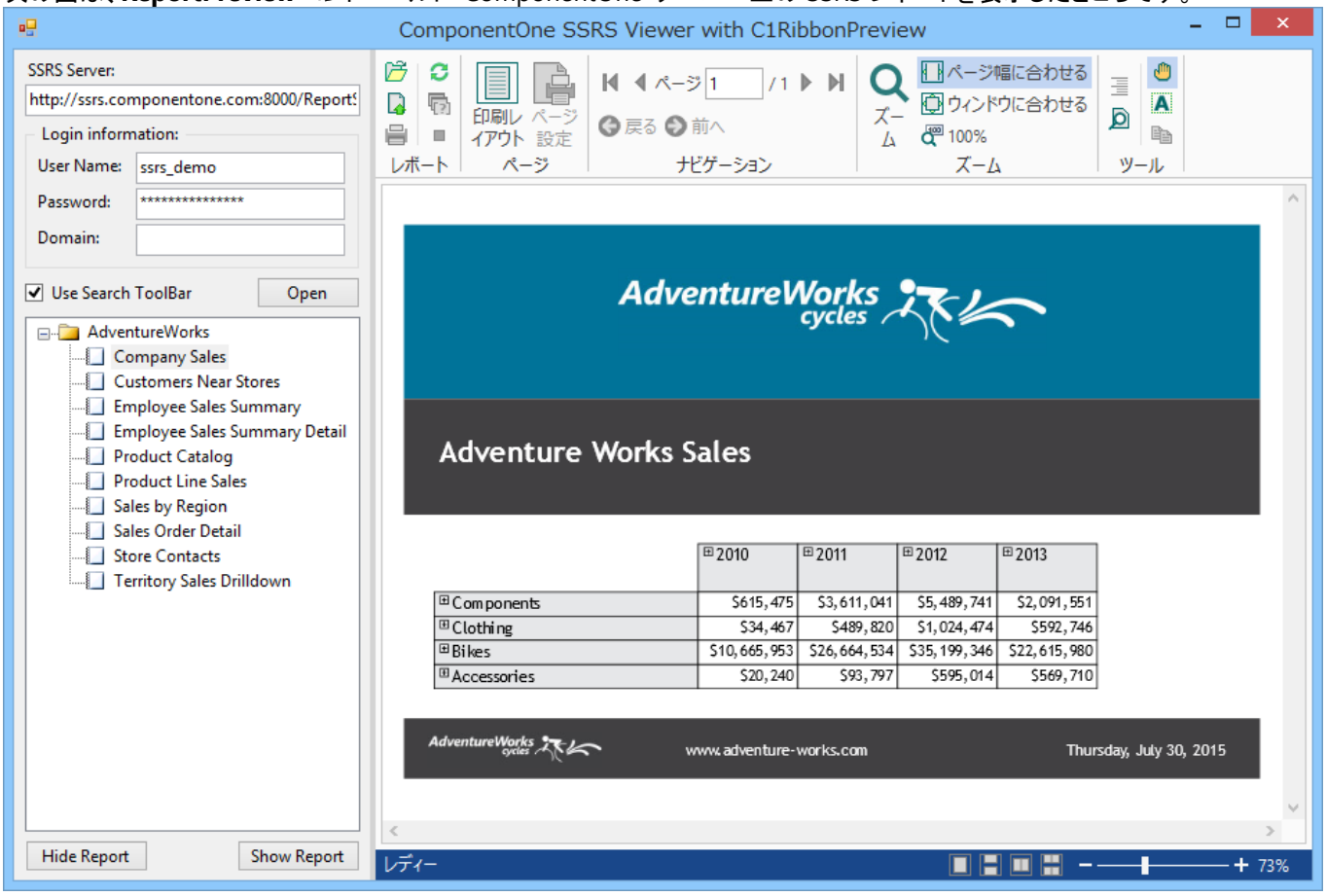
すべての主要な機能がコントロールのリボンにグループ化されています。[レポート]、[ページ]、[ナビゲーション]、[ズーム]、および[ツール]グループで、ドキュメントまたはレポートの開く／閉じる／エクスポート、印刷レイアウトの設定、ページの移動／ズーム、サムネイルのパネルのサポート、アウトライン、テキスト検索などの機能が提供されています。

次の図は、**ReportPreview** コントロールに .c1dx 形式のレポートを表示したところです。

Copyright © GrapeCity, Inc. All rights reserved.

**ReportPreview** コントロールは、**SSRS** レポートを表示する **C1SsrsDocumentSource** もサポートしているため、SSRS サーバー上のレポートツリーを参照して、個々のレポートをプレビューできます。

次の図は、**ReportPreview** コントロールに ComponentOne サーバー上の SSRS レポートを表示したところです。






Copyright © GrapeCity, Inc. All rights reserved.

# C1ReportsScheduler の使い方

以下のトピックには、バックグラウンドでレポート作成をスケジュールするために使用する、**C1ReportsScheduler** に関する重要な情報が含まれています。**C1ReportsScheduler** アプリケーションを使用すると、エクスポートまたは印刷するレポート、レポートのエクスポート形式、レポート作成のスケジュールと頻度を選択できます。

# C1ReportsScheduler について

**C1ReportsScheduler** アプリケーションは、スケジュールされた **ComponentOne Reports for WinForms** タスクをバックグラウンドで実行すること目的としています。以下のレポートタスクタイプがサポートされています。

- XML レポート定義を **C1Report** コンポーネントにロードする。
- XML レポート定義を **C1PrintDocument** コンポーネントにインポートする。
- データがバインドされた **C1PrintDocument** を C1D/C1DX ファイルからロードする。
- 実行可能なユーザープログラムでレポートを生成し、エクスポートまたは印刷する。

各タスクには複数のアクションを指定できます。たとえば、レポートを PDF ファイルと Excel ファイルにエクスポートし、印刷も実行することができます。また、各タスクには、タスクをいつ実行すべきかを指定するスケジュールを関連付けることができます。

**C1ReportsScheduler** アプリケーションは、相互に連携する2つの部分から構成されています。

- **C1ReportsScheduler** フロントエンドアプリケーション（**C1ReportsScheduler.exe**）。
- **C1ReportsScheduler** Windows サービス（**C1ReportsSchedulerService.exe**）。

推奨動作モードでは、サービス（**C1ReportsSchedulerService.exe**）はバックグラウンドで動作し、スケジュールに従って指定されたタスクを実行していきます。フロントエンド（**C1ReportsScheduler.exe**）は、タスクリストの表示または編集、スケジュールの開始または停止、およびサービスの制御に使用できます。フロントエンドはサービスのインストールとセットアップには必要ですが、サービスの実行には必要ないため、通常は、必要に応じてスケジュールされたタスクを調整するために起動し、用が済めば終了します（サービスがマシンにインストールされている場合、フロントエンドは起動時に自動的にサービスに接続します）。

推奨されてはいませんが、スタンドアロンモードの動作も可能です。このモードでは、スケジュールされたタスクを実際に実行するときにもフロントエンドを使用します。この場合はもちろん、フロントエンドアプリケーションを終了すると、すべてのスケジュールが停止します。

サービス固有の操作（インストール、セットアップ、アンインストール）はすべてフロントエンドアプリケーションから実行できます。

# インストールとセットアップ

デフォルトでは、**C1ReportsScheduler** アプリケーションとサービス（**C1ReportsScheduler.exe** と **C1ReportsSchedulerService.exe**）は、**C:\Program Files (x86)\ComponentOne\Apps\v4** ディレクトリにインストールされます。アプリケーションとサービスをインストールしてセットアップするには、次の手順に従います。

1. インストールディレクトリに移動し、**C1ReportsScheduler.exe** アプリケーションをダブルクリックして開きます。**C1ReportsScheduler.exe** アプリケーションをはじめて実行する場合は、ダイアログボックスが開き、**C1ReportsScheduler** サービスをインストールするかどうかを尋ねられます。
2. サービスをインストールするには、〈**はい**〉をクリックします（推奨）。フォームが表示され、フロントエンドがサービスとの通信に使用する WCF アドレス、サービス設定ファイル（.c1rsconf）へのパス、サービススタートアップタイプ（手動または自動）、ログオプションなどのサービスパラメータをセットアップできます。
3. 必要に応じてパラメータを調整し（通常はデフォルト設定で問題ありません）、〈**OK**〉をクリックするとサービスがインストールされます。

進行状況ウィンドウが表示され、それが閉じると、フロントエンドがクライアントモードで起動します。これは、フォームのキャプ


Copyright © GrapeCity, Inc. All rights reserved.

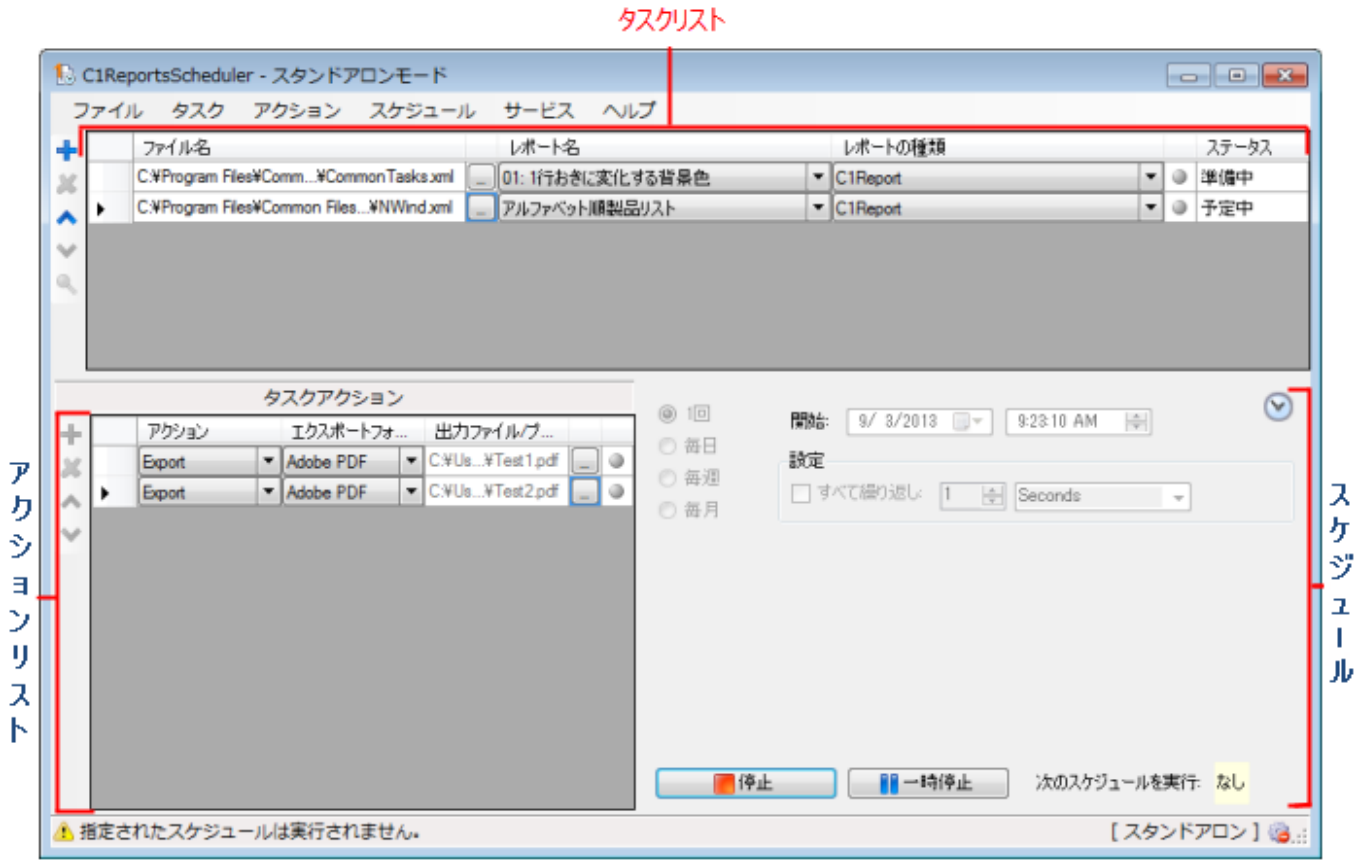
ションに「クライアントモード」、ステータス行に "[ クライアント ]" と表示されることで判断できます（ステータス行には緑のチェックマークの付いた歯車アイコンも表示されます）。

サービスをインストールするかどうかの質問に対して〈**いいえ**〉をクリックすると、フロントエンドはスタンドアロンモードで起動し、フォームのキャプションとステータス行にそのことが示されます。スタンドアロンモードでも、タスクを追加し、アクションとスケジュールを指定し、スケジュールを開始することができます。唯一の違いは、スタンドアロンモードでは、タスクをスケジュールに合わせて実行するには、フロントエンドを実行したままにしておく必要がある点です。

スタンドアロンモードでは、いつでもサービスをインストールして、サービスにタスクリストを移行することができます。サービスをインストールするには、フロントエンドアプリケーションを開き、[**サービス**]メニューから[**サービスのインストール**]を選択します。サービスをアンインストールするには、[**サービス**]メニューから[**サービスのアンインストール**]を選択します。

# ユーザーインタフェース

**C1ReportsScheduler** アプリケーションを開くと、次のような画面が表示されます。




メインの **C1ReportsScheduler** ウィンドウは3つの領域に分けられます。

- メインウィンドウの上半分にはタスクリストが表示されます。各タスクは、生成をスケジュールされたレポートまたはドキュメントを定義します。詳細については、「タスクリスト」を参照してください。
- メインウィンドウの左下部分には、現在のタスクに定義されているアクションのリストが表示されます。詳細については、「 アクションリスト」を参照してください。,
- メインウィンドウの右下部分には、現在のタスクのスケジュールが表示されます。詳細については、「スケジュール」を参照してください。

以下のトピックでは、C1ReportsScheduler アプリケーションのユーザーインタフェースについて説明します。

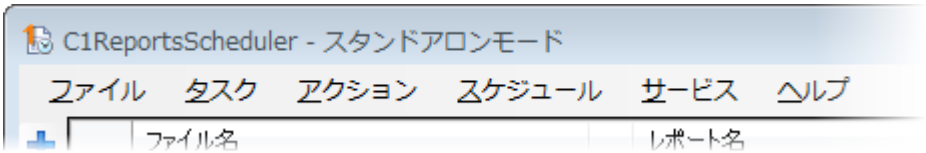
# キャプションとステータスバー

C1ReportsScheduler アプリケーションでの作業中、キャプションとステータスバーにはさまざまなインジケータや情報が表示さ

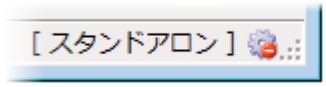

Copyright © GrapeCity, Inc. All rights reserved.

れます。

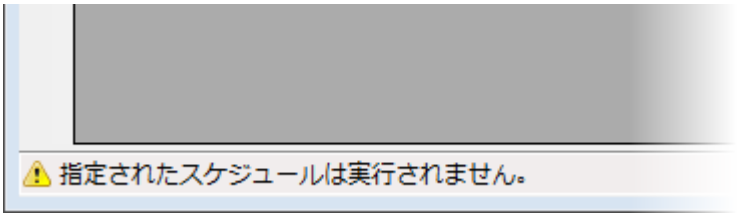
フォームキャプションは、現在のモードを示すために使用されます（クライアントまたはスタンドアロン）。たとえば、次の図のキャプションバーでは、アプリケーションがクライアントモードで実行されていることと、接続先のサービスが表示されています。




ステータスバーには2つの領域があります。ステータスバーの右側には、アイコンと現在のモードの簡単な説明が表示されます。利用可能な2つのモードはクライアントモードとスタンドアロンモードです。たとえば、次の図では、ステータスバーにスタンドアロンモードであることが示されています。

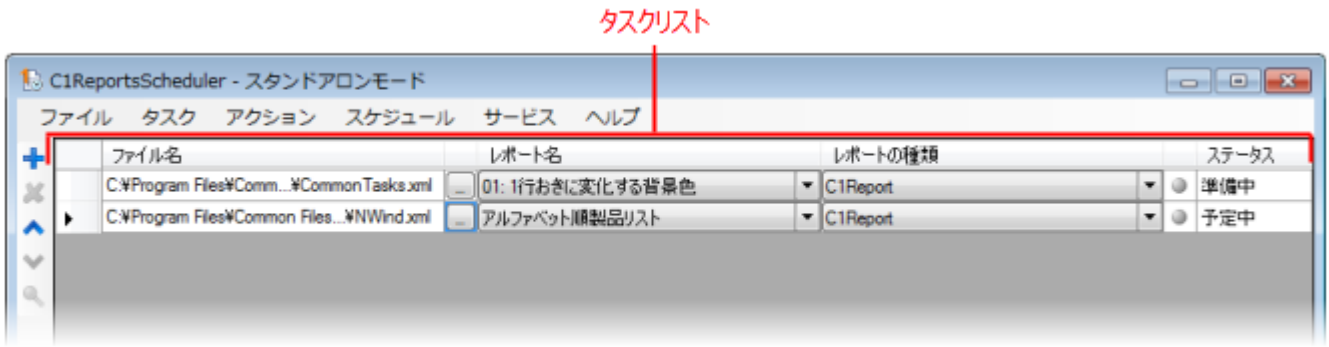



ステータスバーの左側には、警告やエラーメッセージが表示されます。現在のタスクまたはアクションにエラーが発生した場合、メインステータス領域に関連アイコンとエラーの説明が表示されます（「レポート定義ファイルが見つかりません」など）。たとえば、次の図では、ステータスバーに警告が表示されています。




## タスクリスト

タスクリストは、メインウィンドウの上半分に表示され、いくつかの列を持つグリッドから構成されています。グリッドにはさまざまなタスクを追加できます。各タスクは、生成をスケジュールされたレポートまたはドキュメントを定義します。タスクリストは、次のように表示されます。




タスクリストは、以下の列を持つグリッドによって表されます。

- **ファイル名**

[**ファイル名**]列には、アクションの実行対象である **C1Report**、**C1PrintDocument**、または実行ファイルの名前が表示されます。**C1Report**／インポートした **C1Report** タイプのタスクの場合、これはレポート定義ファイルになります。**C1PrintDocument** タイプのタスクの場合、これはドキュメントを含む C1D/C1DX ファイルの名前になります。外部実行ファイルタイプのタスクの場合、これは実行ファイルの名前になります。ファイルを選択するには、ファイル名テキストボックスの右の〈...〉ボタンをクリックします。

- **レポート名**

  **C1Report**／インポートした **C1Report** タスクでのみ使用されます。この列はレポートの名前を指定します。この列は


Copyright © GrapeCity, Inc. All rights reserved.

コンボボックスです。最初の列でレポート定義ファイルが選択されると、コンボドロップダウンボックスに自動的に利用可能なレポートの名前がリストされます。

- **レポートタイプ**

  この列は現在のタスクのタイプを指定します。以下のレポートタスクタイプがサポートされています。

| 状態 | 説明 |
|---|---|
| C1Report | **C1Report** タイプのタスクの場合、レポート定義のロードとレポートの生成に、**C1Report** コンポーネントのインスタンスが使用されます。 |
| インポートした C1Report | **インポートした C1Report** タイプのタスクの場合、レポート定義のロードとレポートの生成に、**C1PrintDocument** コンポーネントのインスタンスが使用されます。 |
| C1PrintDocument | **C1PrintDocument** タイプのタスクの場合、ドキュメントのロードと生成に **C1PrintDocument** が使用されます。 |
| 外部実行ファイル | **外部実行ファイル**タイプのタスクは、外部プログラムによって表されます。これは、レポートを生成するときにコードに依存するアプリケーションを実行するのが目的です。 |

- **タスク状態**

  この列には、タスクの現在の状態を表す小さな画像が表示されます。この列にはキャプションがありません。この列で使われる画像インジケータは次のとおりです。

| | 説明 |
|---|---|
| | グレーのボールは未チェックのタスクを表します。 |
| | 緑のボールは、チェックに成功し、現在実行されていないタスクを表します。 |
| | 黄色のボールは、チェックに成功し、現在実行中のタスクを表します。 |
| | 感嘆符の付いた黄色の三角形は、エラーの発生したタスクを表します。 |

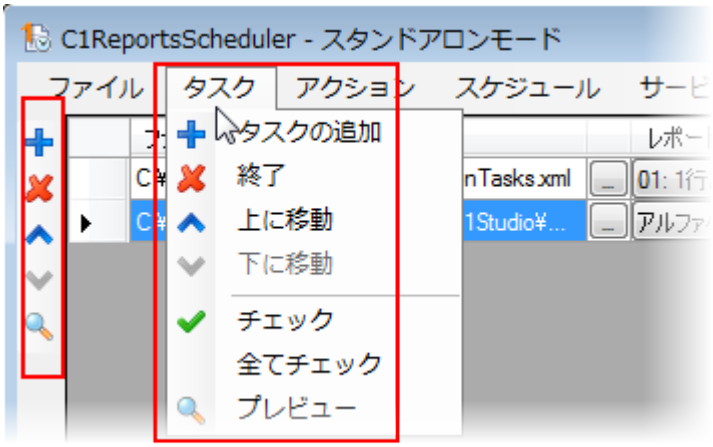
- **ステータス**

  この列には、タスクの現在の状態が表示されます。次のような状態があります。

| 状態 | 説明 |
|---|---|
| 準備完了 | タスクの準備は完了していますが、スケジュールされていません。タスクを編集できる唯一のステータスです。 |
| スケジュール済み | タスクの実行がスケジュールされています。このステータスのタスクは変更できません。 |
| ビジー | 現在実行中のスケジュール済みタスク。このステータスのタスクは変更できません。 |
| 一時停止 | タスクの実行がスケジュールされていますが、スケジュールが一時停止されています。このステータスのタスクは変更できません。 |

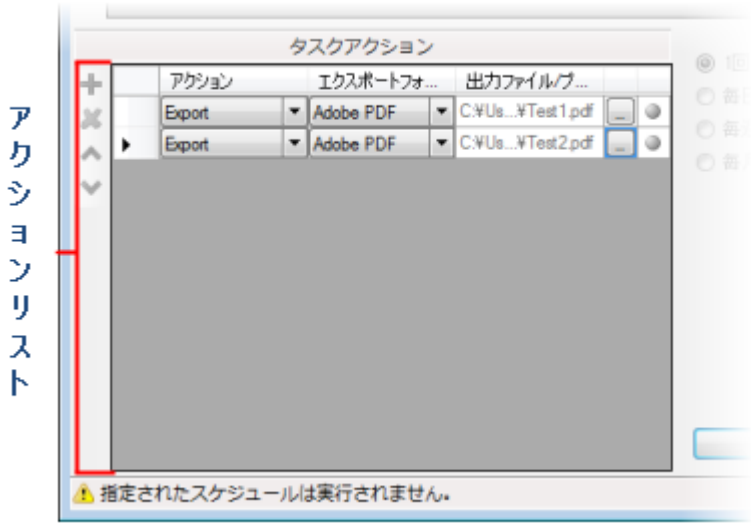
タスクリストを操作するには、[**タスク**]メニューまたはタスクグリッド左のツールバーを使用します。


Copyright © GrapeCity, Inc. All rights reserved.




## アクションリスト

タスクアクションリストは画面の左下に表示され、現在のタスク(タスクリストで選択したタスク)に関連付けられているアクションのリストを表しています。アクションリストは、次のように表示されます。




アクションリストは、以下の列を持つグリッドによって表されます。

- **アクション**

  アクションのタイプです。以下のアクションタイプがサポートされています。

| タイプ | 説明 |
|---|---|
| エクスポート | 現在のタスクによって表されているレポートまたはドキュメントをサポートされている外部形式のいずれかにエクスポートします。<br>このアクションタイプは、外部実行ファイルタイプのタスクには指定できません。 |
| 印刷 | 現在のタスクによって表されているレポートまたはドキュメントを印刷します。<br>このアクションタイプは、外部実行ファイルタイプのタスクには指定できません. |
| 実行 | 行ファイルを実行します。このアクションタイプは、外部実行ファイルタイプのタスクにのみ指定できます。 |

- **エクスポート形式**


Copyright © GrapeCity, Inc. All rights reserved.

エクスポートアクションの場合、この列でエクスポート形式を指定します。C1Report コンポーネントと C1PrintDocument コンポーネントを使用するタスクの場合は、使用できるエクスポート形式のセットが複数あります（これらのコンポーネントのエクスポートがプレビューコントロールで処理される方法と似ています）。

- **出力ファイル／プリンタ名**

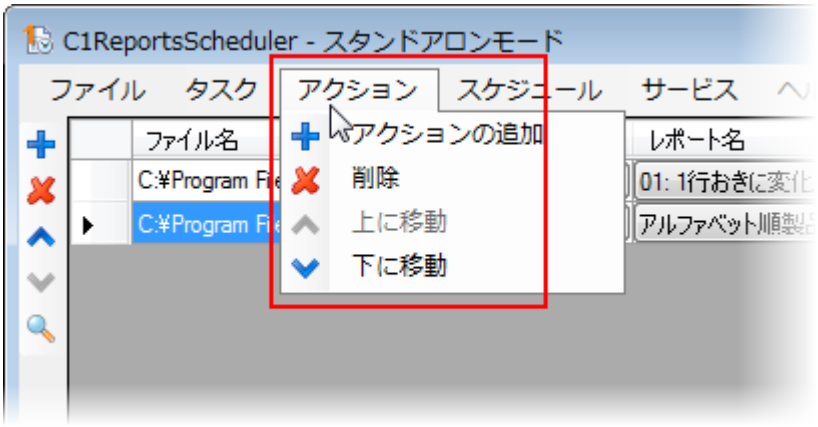
  エクスポートファイルの名前またはプリンタ名を指定します。このテキストボックスの右のボタンをクリックすると、（アクションタイプに応じて）ファイルまたはプリンタ名を選択できます。

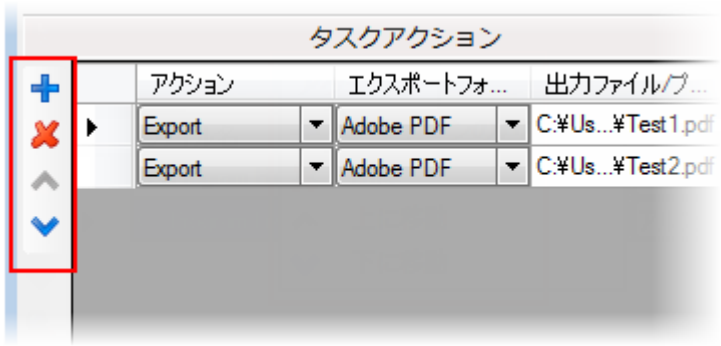
- **現在のステータス**

  最後の列（タイトルなし）は、アクションのステータスを表す小さな画像の表示に使用されます。

| | 説明 |
|---|---|
| | グレーのボールは未チェックのタスクを表します。 |
| | 緑のボールは、チェックに成功し、現在実行されていないタスクを表します。 |
| | A黄色のボールは、チェックに成功し、現在実行中のタスクを表します。 |
| | 感嘆符の付いた黄色の三角形は、エラーの発生したタスクを表します。 |

アクションリストを操作するには、[**アクション**]メニューまたはアクショングリッド左のツールバーを使用します。:

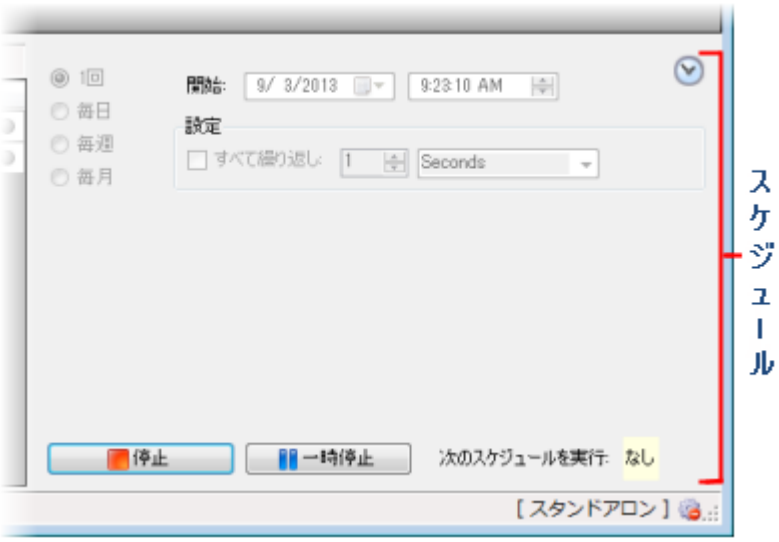



## スケジュール

スケジュールパネルには、現在のタスクに関連付けられているスケジュールが表示され、タスクのをスケジュールし、実行することができます。スケジュール領域は、次のように表示されます。


Copyright © GrapeCity, Inc. All rights reserved.




アプリケーションのスケジュールセクションには以下のオプションが含まれています。

- **頻度**

  パネルの左上には4つのラジオボタンがあり、タスクを実行する頻度を指定できます([**1 回** ]、[**毎日**]、[**毎週**]、[**毎月**])。1つ目のオプション、[**1 回**]は、指定した秒、分、または時間ごとにタスクを繰り返すことができます。その他のオプションは、日、週、または月ごとに繰り返します。選択するラジオボタンに応じて、スケジュールパネルに表示されるスケジュールオプションは変化します。

- **開始日時**

  実行を開始する日時。スケジュールアクションを開始する日時を設定します。

- **開始、終了、一時停止**

  パネルの下部には、現在のタスクの開始、終了、一時停止、再開を行える2つのボタンがあります。使用できるボタンは現在のタスクのステータスに応じて変化します。

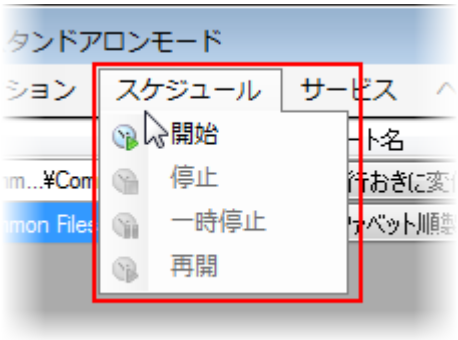
- **次回実行予定**

  ボタンの右に、タスクの次回実行予定時刻が表示されます。

- **繰り返し**

  このセクションは、選択した〈頻度〉ラジオボタンに応じて変化します。たとえば、[**毎日**]オプションには数値ボックスが含まれ、スケジュールタスク間の日数を選択できます。[**毎月**]オプションを選択した場合は、タスクを実行する月と各月の日付などを選択できます。

スケジュールしたタスクの開始、終了、一時停止、再開は、[**Schedule**]メニューから実行することもできます。

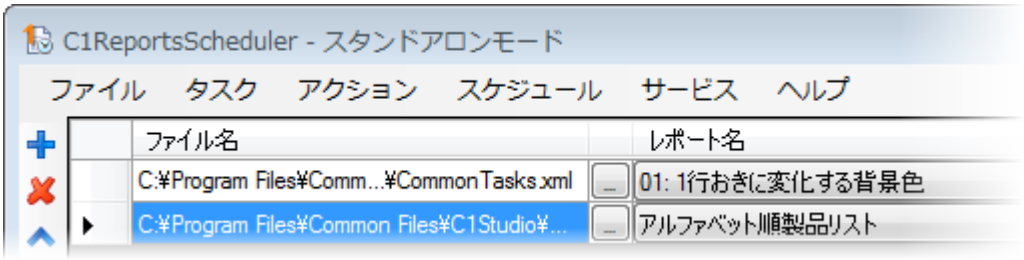




Copyright © GrapeCity, Inc. All rights reserved.

# メニューシステム

**C1ReportsScheduler** アプリケーションにはいくつかのメニューオプションがあります。アプリケーションメニューには、[**ファイル**]、[**タスク**]、[**アクション**]、[**スケジュール**]、[**サービス**]、[**ヘルプ**]オプションがあり、次のように表示されます。




このトピックでは、使用可能な各メニューオプションについて説明します。

## ファイル

[**ファイル**]メニューには、次のオプションが含まれています。

- **新規**

  現在のタスクリストをクリアします。

- **開く**

  既存の C1Reports Scheduler 設定ファイル(.c1rsconf)を開きます。

- **保存**

  現在のタスクリストを保存します。

- **名前を付けて保存**

  現在のタスクリストを C1Reports Scheduler 設定ファイル(.c1rsconf)に保存します。

- **終了**

  プログラムを閉じます。

## タスク

[**タスク**]メニューには、次のオプションが含まれています。

- **タスクを追加**

  タスクリストに新しいタスクを追加します。タスクはリストの最後に追加されますが、適切なコマンドを使用して上下に移動することができます。

- **削除**

  現在のタスクをリストから削除します。

- **上に移動**

  現在のタスクをリストの中で上に移動します。

- **下に移動**


Copyright © GrapeCity, Inc. All rights reserved.

現在のタスクをリストの中で下に移動します。

- **チェック**

  現在のタスクの有効性とそのすべてのアクション仕様を検証します。チェック項目には、レポート定義の存在と有効性、出力ファイル名の正しさなどが含まれます。タスクはスケジュール時に自動的にチェックされます。チェックに成功したタスクは、[状態]列に緑のボールアイコンが表示されます。チェックが失敗した場合、タスクをスケジュールすることはできず、感嘆符アイコンが表示されます。マウスカーソルをアイコンの上に置くとエラーメッセージが表示されます（タスクを選択している場合は、ステータス行にも表示されます）。

- **すべてチェック**

  リストのすべてのタスクの有効性をチェックします。

- **プレビュー**

  現在のタスクのレポートまたはドキュメントを生成し、印刷プレビューダイアログに表示します。このコマンドは、タスクがスケジュールされている場合は無効になります。

## アクション

[アクション]メニューには、次のオプションが含まれています。

- **アクションを追加**

  現在のタスクのアクションリストに新しいアクションを追加します。アクションはリストの最後に追加されますが、適切なコマンドを使用して上下に移動することができます。

- **削除**

  現在のアクションをリストから削除します。

- **上に移動**

  現在のアクションをリストの中で上に移動します。

- **下に移動**

  現在のアクションをリストの中で下に移動します。

## スケジュール

[**スケジュール**]メニューには、次のオプションが含まれています。

- **開始**

  現在のタスクのスケジュールを開始します。スケジュールが開始している場合は、タスクまたはそのアクションを編集することはできません。

- **終了**

  現在のタスクのスケジュールを終了します。

- **一時停止**

  現在のタスクのスケジュールを一時停止します。

- **再開**


Copyright © GrapeCity, Inc. All rights reserved.

現在のタスクのスケジュールが一時停止されている場合に、それを再開します。

## サービス

[サービス]メニューには、次のオプションが含まれています。

- **接続**

  C1Reports Scheduler サービスに接続します。このコマンドは、サービスの実行中にのみ使用できます。

- **切断**

  C1Reports Scheduler サービスから切断します。

- **タスクを転送**

  現在のタスクリストを C1Reports Scheduler サービスに転送します。このコマンドは、サービスは実行中だがフロントエンドアプリケーションがサービスから切断され、フロントエンドに独自のタスクリストがある場合に使用できます。

- **開始**

  C1Reports Scheduler サービスを開始します。このコマンドは、サービスがマシンにインストールされているが実行されていない場合に使用できます。

- **終了**

  C1Reports Scheduler サービスを終了します。このコマンドは、サービスがマシンにインストールされていて実行されている場合に使用できます。

- **一時停止**

  C1Reports Scheduler サービスを一時停止します。このコマンドは、サービスがマシンにインストールされていて実行されている場合に使用できます。

- **再開**

  C1Reports Scheduler サービスを再開します。このコマンドは、サービスがマシンにインストールされていて一時停止されている場合に使用できます。

- **サービスセットアップ**

  C1Reports Scheduler サービスセットアップダイアログを起動します。このダイアログではサービスパラメータを調整できます。ダイアログの〈OK〉をクリックするとサービスが再起動します。このコマンドは、サービスがマシンにインストールされている場合に使用できます。

- **サービスのインストール**

  C1Reports Scheduler サービスをマシンにインストールします。このコマンドは、サービスがマシンにインストールされていない場合に使用できます。

- **サービスのアンインストール**

  C1Reports Scheduler サービスをアンインストールします。このコマンドは、サービスがマシンにインストールされている場合に使用できます。

- **サービスログ**

  C1Reports Scheduler サービスログを含むウィンドウを表示します。


Copyright © GrapeCity, Inc. All rights reserved.

## ヘルプ

[**ヘルプ**]メニューには、次のオプションが含まれています。

- **目次**

  ヘルプファイルを表示します。

- **バージョン情報**

  アプリケーションに関する情報、オンラインリセットへのリンクを含む[バージョン情報]ボックスを表示します。


Copyright © GrapeCity, Inc. All rights reserved.

# C1PrintDocument の使い方

**C1PrintDocument** は、印刷、プレビュー、およびディスクファイルへの保存が可能な複雑なドキュメントを作成するために使用できるコンポーネントです。ドキュメントを PDF、RTF などのさまざまな外部形式にエクスポートすることもできます。

**C1PrintDocument** は、次のような多くの優れた機能を提供します。

- 一貫性のある透過的な階層ドキュメント構造
- 使いやすく効率的なスタイル
- 変更や異なるページ設定に合わせて変更および再レンダリングできるドキュメント
- ドキュメントのプレビュー、印刷、保存、および外部形式へのエクスポート
- 入力コントロールになるドキュメント（プレビューコンポーネントによってサポート）
- 完全なテーブルのサポート（ネストされたテーブルを含む）
- 複数スタイルテキストのサポート（インライン画像を含む）
- True Type フォントの埋め込み
- ハイパーリンク。
- 目次の自動生成

**C1PrintDocument** で使用されるデフォルトの名前空間は、C1.C1Preview です。C1Preview 製品の Windows Forms 部分（ドキュメントを操作するための Windows フォームコントロールを提供）は、デフォルトの名前空間 C1.Win.C1Preview を使用します。

ドキュメント全体は、Component から継承された**C1PrintDocument**クラスによって表されます。

C1PrintDocument は、主に次の部分で構成されます。

- **Body**

  ドキュメントの実際のコンテンツ（テキスト、画像など）。Body（本体）は、ドキュメントの論理構造を表します。この後のページコレクションも参照してください。

- **Pages**

  コンテンツ（本体）と特定のページ設定に基づいて生成されたページのコレクション。通常、ページコレクションは、情報を失うことなく再生成できます。たとえば、異なる用紙サイズで再生成できます。

- **Style**

  ドキュメントのルートスタイル。スタイルは、ドキュメント要素（フォント、色、線スタイルなど）のほとんどのビジュアルプロパティを制御します。

- **Dictionary**

  ドキュメント内の複数の場所で使用される画像は、辞書に格納して再利用することで、パフォーマンスを向上させ、メモリフットプリントを減らすことができます。

- **EmbeddedFonts**

  ドキュメントで使用される埋め込み True Type フォントのコレクションです。

- **Tags**

  ドキュメント内に挿入できるユーザー定義タグのコレクションです。これらのタグは、ドキュメントの生成時に対応する値に置き換えられます。

## C1PrintDocument の使い方

# 描画オブジェクト


Copyright © GrapeCity, Inc. All rights reserved.

次のセクションでは、レンダリングオブジェクトの階層、包含関係、配置、およびスタック規則について説明します。

# 描画オブジェクトの階層

**C1PrintDocument** のすべてのコンテンツは、**描画オブジェクト**によって表されます。さまざまなタイプのコンテンツを表すために、豊富な描画オブジェクト（**RenderObject** クラスに基づく）が階層化されて提供されています。描画オブジェクト型の階層を次に示します。各クラスには簡単な説明が付いており、斜体は抽象クラスを表します。

| 描画オブジェクトタイプ | 説明 |
|---|---|
| **RenderObject** | 階層全体の基本クラス。 |
| **RenderArea** | 他のオブジェクトの汎用コンテナ。 |
| **RenderToc** | 目次を表します。 |
| **RenderReport** | サブレポート（RenderField 内に含まれ、SubReport プロパティで指定された **C1Report**）を表します。 |
| **RenderSection** | インポートされた **C1Report** の1つのセクションを表します。 |
| **RenderC1Printable** | **C1PrintDocument** にシームレスに挿入できる外部コントロールを表します。（オブジェクトはIC1Printable インタフェースに対応することが必須） |
| **RenderEmpty** | 空の描画オブジェクト。ページ区切りなど、コンテンツをレンダリングする必要がない部分に便利なプレースホルダを提供します。 |
| **RenderGraphics** | .NET Graphics オブジェクト内の描画を表します。 |
| **RenderImage** | 画像を表します。 |
| **RenderInputBase** | 入力コントロール（ドキュメントに埋め込まれ、プレビューで表示／処理されるコントロール）の基本クラス。 |
| **RenderInputButtonBase** | ボタン形式の入力コントロールの基本クラス。（ボタン、チェックボックス、ラジオボタンなど） |
| **RenderInputButton** | 入力コントロール（ドキュメントに埋め込まれ、プレビューで表示／処理されるコントロール）の基本クラス。 |
| **RenderInputCheckBox** | チェックボックスを表します。 |
| **RenderInputRadioButton** | ラジオボタンを表します。 |
| **RenderInputComboBox** | コンボボックスを表します。（ドロップダウンリストのあるテキスト入力コントロールなど） |
| **RenderInputText** | テキストボックスコントロールを表します。 |
| **RenderRichText** | リッチテキストを表します。 |
| **RenderShapeBase** | 図形（線、多角形など）の基本クラス |
| **RenderLineBase** | 線および多角形の基本クラス。 |
| **RenderLine** | 線を表します。 |


Copyright © GrapeCity, Inc. All rights reserved.

| 描画オブジェクトタイプ | 説明 |
|---|---|
| **RenderPolygon** | 多角形を表します。 |
| **RenderRectangle** | 四角形を表します。 |
| **RenderEllipse** | 楕円を表します。 |
| **RenderRoundRectangle** | 角丸四角形を表します。 |
| **RenderTable** | テーブルを表します。 |
| **RenderTextBase** | テキストおよび段落オブジェクトの基本クラス。 |
| **RenderParagraph** | 段落を表します。段落は、さまざまなスタイルのテキストフラグメントとインライン画像を含むことができます。 |
| **RenderTocItem** | 目次に挿入される段落を表します。 |
| **RenderText** | 単純な単一スタイルのテキストを表します。 |
| **RenderField** | **C1Report** のフィールドを表します。この型のオブジェクトは C1Report を **C1PrintDocument** にインポートするときに作成されます。 |
| **RenderBarCode** | バーコードを表します。 |

# 描画オブジェクトの包含、配置、スタック規則

**C1PrintDocument** は階層構造を持っています。描画オブジェクトをドキュメントの構成要素とするには、それをドキュメントの階層の特定の位置に追加する必要があります。ドキュメントの主要部分は **Body** と呼ばれます。描画オブジェクトを本体に追加するには、ドキュメントの **Body** プロパティの **Children** コレクションに追加する必要があります。次に例を示します。

**Visual Basic コードの書き方**

Visual Basic

```
Dim doc As New C1PrintDocument()
Dim rt As New RenderText()
rt.Text = "これはテキストです。"
doc.Body.Children.Add(rt)
```

**C# コードの書き方**

C#

```
C1PrintDocument doc = new C1PrintDocument();
RenderText rt = new RenderText();
rt.Text = "これはテキストです";
doc.Body.Children.Add(rt);
```

ドキュメントの **Body.Children** コレクションには、ドキュメントのすべての最上位の描画オブジェクトが含まれています。さらに、その各描画オブジェクトも **Children** コレクションを保持し、そのコレクション内に描画オブジェクトが含まれています。これは、Windows フォームコントロールが次々と内側にネストされる（各コントロールがそれ自身に含まれるコントロールのコレクションを保持する）仕組みに似ています。

ドキュメントの本文の他に、ドキュメント内にオブジェクトを表示するための場所が2つあります。これらは、それぞれ **PageHeader**と **PageFooter** プロパティを使用してアクセスできるページのヘッダーとフッターです。


Copyright © GrapeCity, Inc. All rights reserved.

## 表示領域

任意の表示オブジェクトには、その子のような、その他の表示オブジェクトを含めることができますが、その他のオブジェクトのコンテナとして専用にデザインされた **RenderArea** と呼ばれる1つの表示オブジェクトがあります。表示領域とその他の表示オブジェクト(表示テキストなど)の主な相違は、表示領域では、その幅と高さのどちらも **Auto** として指定される(つまり、対応する寸法が子のサイズによって自動的に決定される)のに対し、その他の種類のオブジェクトでは、自動サイズ調整がオブジェクト自体のコンテンツ(RenderText の文字サイズ、RenderImage の画像サイズなど)によって決定されることです。

デフォルトでは、新しい表示領域を作成すると、その幅はその親の幅に等しくなります(したがって、最上位レベルの表示領域はページ全体、または段組みレイアウトの場合、現在の段落全体に引き伸ばされます)。一方、表示領域の高さは、デフォルトで Auto(**Unit.Auto**)に設定され、表示領域の子の高さの合計によって決定されます。したがって、最上位レベルの表示領域のデフォルトの動作は、ページ全体の幅を占有し、そのコンテンツのすべてに対応するように必要に応じて下方に伸びることです(場合によっては、複数ページにまたがります)。表示領域の幅を自動(**Unit.Auto**)に設定できます。ここでは、その領域の子の幅の合計に対応するように調整されます。この場合、この領域の子の幅の合計がページの幅を超えると、**水平方向**の改ページが行われ、現在のページの右側に拡張されたページが追加されます。**水平方向**の改ページを禁止(必要に応じて右側の領域をクリッピング)するには、領域の **CanSplitHorz** プロパティを **False** に設定します(デフォルトは、**True**)。

## スタック規則

コンテナ内の描画オブジェクトは、デフォルトではスタック規則に従って配置されます。スタック規則は、コンテナの **Stacking** プロパティの値で決定されます。この値は、次のいずれか(**StackingRulesEnum** 列挙型のメンバ)に設定できます。

- **BlockTopToBottom**

  コンテナ内の各オブジェクトはそれぞれ別のオブジェクトの1つ下に配置されます。現在のページの下端に到達すると、新しいページが追加されます。これはデフォルトです。

- **BlockLeftToRight**

  オブジェクトは左から右に配置されます。現在のページの右端に到達すると、新しい「水平方向」ページが追加されます。水平方向ページは論理的に前のページを右に拡張します。**C1PreviewPane** はデフォルトでこの位置を認識し、これらのページを並べて表示します。

- **InlineLeftToRight**

  オブジェクトはインラインで左から右に配置されます。現在のページの右端に到達すると、続きは次の行に折り返されます。現在のページの下端に到達すると、新しいページが追加されます。

スタック規則は、含まれているオブジェクト(子)には伝達されません。つまり、レンダリング領域を定義し、スタックを **BlockLeftToRight** 値(デフォルト以外)に設定した後で、その内部に別のレンダリング領域を追加した場合、そのスタック規則は、明示的に変更しないかぎりデフォルト(**BlockTopToBottom**)になります。

描画オブジェクトの **X** プロパティと **Y**プロパティを使用して、位置を明示的に設定することもできます(詳細については、次のセクションを参照)。この場合、その描画オブジェクトはスタック順にまったく関与しません。つまり、その位置は兄弟オブジェクトの位置に影響せず、兄弟オブジェクトの位置にも影響されません。

## 描画オブジェクトのサイズと位置の指定

描画オブジェクトのサイズと位置を制御する次の4つのプロパティがあります。

- **X** - オブジェクトの X 座標を指定します。
- **Y** - オブジェクトの Y 座標を指定します。
- **Width** - オブジェクトの幅を指定します。
- **Height** - オブジェクトの高さを指定します。

これらのプロパティはすべて Unit 型です。X と Y のデフォルト値は「自動」(静的フィールド **Unit.Auto**)です。つまり、オブジェ


Copyright © GrapeCity, Inc. All rights reserved.

クトは、親から指定されるスタック規則に従って配置されます（詳細については、「スタック規則」を参照）。幅と高さのデフォルト値は、描画オブジェクトの型によって異なります。

すべての描画オブジェクトのデフォルトのサイズを次に示します。

| | 幅 | 高さ | 自動サイズ |
|---|---|---|---|
| **RenderArea RenderToc RenderReport RenderSection RenderC1Printable** | 親の幅 | 自動 | 子のサイズの合計によって決まります。 |
| **RenderEmpty** | 自動 | 自動 | 0 |
| **RenderGraphics** | 自動 | 自動 | コンテンツのサイズによって決定されます。 |
| **RenderImage** | 自動 | 自動 | 画像のサイズによって決定されます。 |
| **RenderInputButton RenderInputCheckBox RenderInputRadioButton RenderInputComboBox RenderInputText** | 自動 | 自動 | コンテンツのサイズによって決定されます。 |
| **RenderRichText** | 親の幅と同じ | 自動 | -- |
| **RenderLine RenderPolygon RenderEllipse RenderArc RenderPie RenderRectangle RenderRoundRectangle** | 自動 | 自動 | 図形のサイズによって決定されます。 |
| **RenderTable** | 親の幅と同じ<br>（自動幅は、列の幅の合計として計算されます。この場合は、すべての列の幅が絶対値で指定されている必要があります。） | 自動 | （すべての行の高さの合計として決定） |
| **RenderParagraph RenderText RenderTocItem** | 親の幅と同じ | 自動 | テキストのサイズによって決定されます。 |
| **RenderField** | 親の幅 | 自動 | コンテンツのサイズによって決まります。 |
| **RenderBarCode** | 自動 | 自動 | コンテンツのサイズによって決まります。 |

これらのプロパティのデフォルト値はカスタム値でオーバーライドできます。「スタック規則」で説明したように、X 座標または Y 座標に「自動」以外の値を指定すると、そのオブジェクトがスタックフローから除外されます。サイズプロパティと位置プロパティは、次のいずれかの方法で設定できます（下のサンプルの「**ro**」は描画オブジェクト）。

- 自動に設定します。その意味は描画オブジェクト型によって異なります。

  ```
  ro.Width = Unit.Auto;
  ro.Height = "auto";
  ```


Copyright © GrapeCity, Inc. All rights reserved.

- 絶対値で設定します。

  **ro.X = new Unit(8, UnitTypeEnum.Mm);**

  **ro.Y = 8**;（単位として C1PrintDocument.DefaultUnit が使用される）

  **ro.Width = "28mm";**

- 親のサイズに対する割合で設定します。この設定を座標に使用しても意味がなく、0になります。

  **ro.Height = new Unit(50, DimensionEnum.Width);**

  **ro.Width = "100%";**

  別のオブジェクトのサイズまたは位置を参照して設定します。オブジェクトは、次のキーワードのいずれかで識別されます。

  **self** - 現在のオブジェクト（デフォルト値。省略可能）。>

  **parent** - オブジェクトの親。

  **prev** - 前の兄弟オブジェクト。

  **next** - 次の兄弟オブジェクト。

  **page** - 現在のページ。

  **column** - 現在の列。

  **page<N>** - 指定された番号のページ。メモ：このページがすでに存在している必要があります。将来生成されるページへの前方参照はサポートされていません。

  **column<M>** - 現在のページの指定された列。

  **page<N>.column<N>** - 指定されたページの指定された列。

  **<object name>** - 指定された名前のオブジェクト。名前は **Name** プロパティの値です。オブジェクトはまず現在の兄弟から検索され、次に子から検索されます。

  参照されるオブジェクトのサイズと位置は、**left**、**top**、**right**、**bottom**、**width**、**height** の各キーワードで識別されます（座標はオブジェクトの親に対する相対位置）。

  いくつかの例を示します。

  **ro.Height = "next.height";** - オブジェクトの高さを次の兄弟オブジェクトの高さに設定します。

  **ro.Width = "page1.width";** - オブジェクトの幅を最初のページの幅に設定します。

  **ro.Height = "width";** - オブジェクトの高さを自分自身の幅に設定します。

  **ro.Y = "prev.bottom";** - オブジェクトの Y 座標を前の兄弟オブジェクトの下端に設定します。

  **ro.Width = "prev.width";** - オブジェクトの幅を前の兄弟オブジェクトの幅に設定します。

- 「Max」および「Min」関数の使用。次に例を示します。

  **ro.Width = "Max(prev.width,6cm)";** - オブジェクトの幅を6 cm または直前の兄弟オブジェクトの幅の大きい方に設定します。

- 式で設定します。式は、上で説明されている方法のいずれかを使用し、**+**、-、*、**/**、**%**、カッコ ( ) で組み合わせて、別のオブジェクトのサイズや位置を参照できます。次に例を示します。

  **ro.Width = "prev.width + 50%prev.width";** – オブジェクトの幅を前の兄弟オブジェクトの幅の 1.5 倍に設定します。

  **ro.Width = "150%prev";** - 上と同じです。設定する要素と同じ要素を参照する場合は、要素（この場合は「width」）を省略できます。

  **ro.Width = "prev*1.5";** - これも上と同じです。ただし、パーセントではなく乗算を使用しています。

サイズや位置が文字列に設定されているサンプルでは、**Unit(string)** コンストラクタを使用する構文も使用できます。次に例


Copyright © GrapeCity, Inc. All rights reserved.

を示します。

**Visual Basic コードの書き方**

```
Visual Basic
ro.Width = New C1.C1Preview.Unit("150%prev")
```

**C# コードの書き方**

```
C#
ro.Width = new Unit("150%prev");
```

文字列の大文字小文字は区別されないため、「prev.width」、「PrEv.WidhT」、および「PREV.WIDTH」はすべて同じです。

# 表示オブジェクトの相対位置決め方式の例

以下に、オブジェクトの相対位置決め方式を使用して、画像とテキストを配置するいくつかの例を示します（サンプル内の「myImage」は、コードの他の場所で宣言された System.Drawing.Image 型のオブジェクトです）。

このコードは、オブジェクトを次々に標準ブロックフローに単純に追加する画像の下にテキストを配置します。

**Visual Basic コードの書き方**

```
Visual Basic
Dim doc as New C1PrintDocument
Dim rt as New RenderText("test")
Dim ri as New RenderImage(myImage)
Dim ra As New RenderArea()
ra.Children.Add(ri)
ra.Children.Add(rt)
doc.Body.Children.Add(ra)
```

**C# コードの書き方**

```
C#
C1PrintDocument doc = new C1PrintDocument();
RenderText rt = new RenderText("test");
RenderImage ri = new RenderImage(myImage);
RenderArea ra = new RenderArea();
ra.Children.Add(ri);
ra.Children.Add(rt);
doc.Body.Children.Add(ra);
```

このコードは、子が領域に逆順で追加される場合に同じ結果（画像の下のテキスト）を生成します（両方のオブジェクトに自動調整されない座標が明示的に指定されているため、どちらもブロックフローに挿入されません）。

**Visual Basic コードの書き方**

```
Visual Basic
Dim doc as New C1PrintDocument
Dim rt as New RenderText("test")
Dim ri as New RenderImage(myImage)
Dim ra As New RenderArea()
```


Copyright © GrapeCity, Inc. All rights reserved.

```
' 親の上部に画像を配置します。
ri.Y = 0
' 次の兄弟の下にテキストを配置します。
rt.Y = "next.bottom"
' テキストの幅を自動調整します。
rt.Width = Unit.Auto
ra.Children.Add(ri)
ra.Children.Add(rt)
doc.Body.Children.Add(ra)
```

**C# コードの書き方**

C#

```
C1PrintDocument doc = new C1PrintDocument();
RenderText rt = new RenderText("test");
RenderImage ri = new RenderImage(myImage);
RenderArea ra = new RenderArea();
// 親の上部に画像を配置します。
ri.Y = 0;
// 次の兄弟の下にテキストを配置します。
rt.Y = "next.bottom";
// テキストの幅を自動調整します。
rt.Width = Unit.Auto;
ra.Children.Add(rt);
ra.Children.Add(ri);
doc.Body.Children.Add(ra);
```

以下のコードは、標準ブロックフローに画像を挿入すると共に、テキストを画像の右側に配置し、画像に対して垂直方向に中央揃えします。

**Visual Basic コードの書き方**

Visual Basic

```
Dim doc as New C1PrintDocument
Dim rt as New RenderText("test")
Dim ri as New RenderImage(myImage)
Dim ra As New RenderArea()
ra.Children.Add(ri)
rt.Width = Unit.Auto
' 画像の次にテキストを追加します。
ra.Children.Add(rt)
rt.X = "prev.right"
rt.Y = "prev.height/2-self.height/2"
doc.Body.Children.Add(ra)
```

**C# コードの書き方**

C#

```
C1PrintDocument doc = new C1PrintDocument();
RenderText rt = new RenderText("test");
RenderImage ri = new RenderImage(myImage);
RenderArea ra = new RenderArea();
```


Copyright © GrapeCity, Inc. All rights reserved.

```
ra.Children.Add(ri);
rt.Width = Unit.Auto;
// 画像の次にテキストを追加します。
ra.Children.Add(rt);
rt.X = "prev.right";
rt.Y = "prev.height/2-self.height/2";
doc.Body.Children.Add(ra);
```

このコードは、画像の右側にテキストを配置し、垂直方向に中央揃えしますが、相対 ID「prev」ではなく位置決め式で **RenderObject.Name** も使用します。また、式で絶対長さを使用した結果、テキストが右に 2 mm 移動します。

**Visual Basic コードの書き方**

Visual Basic

```
Dim doc as New C1PrintDocument
Dim rt as New RenderText("test")
Dim ri as New RenderImage(myImage)
Dim ra As New RenderArea()
ri.Name = "myImage"
rt.Width = "auto"
rt.X = "myImage.right+2mm"
rt.Y = "myImage.height/2-self.height/2"
ra.Children.Add(ri)
ra.Children.Add(rt)
doc.Body.Children.Add(ra)
```

**C# コードの書き方**

C#

```
C1PrintDocument doc = new C1PrintDocument();
RenderText rt = new RenderText("test");
RenderImage ri = new RenderImage(myImage);
RenderArea ra = new RenderArea();
ri.Name = "myImage";
rt.Width = "auto";
rt.X = "myImage.right+2mm";
rt.Y = "myImage.height/2-self.height/2";
ra.Children.Add(ri);
ra.Children.Add(rt);
doc.Body.Children.Add(ra);
```

以下のコードは、組み込み Max 関数を使用してテキストが右に少なくとも 6cm 移動するように同じ例を変更します。

**Visual Basic コードの書き方**

Visual Basic

```
Dim doc as New C1PrintDocument
Dim rt as New RenderText("test")
Dim ri as New RenderImage(myImage)
Dim ra As New RenderArea()
ri.Name = "myImage"
rt.Width = "auto"
rt.X = "Max(myImage.right+2mm,6cm)"
```


Copyright © GrapeCity, Inc. All rights reserved.

```
rt.Y = "myImage.height/2-self.height/2"
ra.Children.Add(ri)
ra.Children.Add(rt)
doc.Body.Children.Add(ra)
```

**C# コードの書き方**

C#

```
C1PrintDocument doc = new C1PrintDocument();
RenderText rt = new RenderText("test");
RenderImage ri = new RenderImage(myImage);
RenderArea ra = new RenderArea();
ri.Name = "myImage";
rt.Width = "auto";
rt.X = "Max(myImage.right+2mm,6cm)";
rt.Y = "myImage.height/2-self.height/2";
ra.Children.Add(ri);
ra.Children.Add(rt);
doc.Body.Children.Add(ra);
```

以下のコードスニペットは、画像をページの右側に配置すると共に（表示領域の幅にデフォルト値である親の幅を使用して）、テキストを左揃えにし、画像に対して垂直方向に中央揃えします。

**Visual Basic コードの書き方**

Visual Basic

```
Dim doc as New C1PrintDocument
Dim rt as New RenderText("test")
Dim ri as New RenderImage(myImage)
Dim ra As New RenderArea()
ri.Name = "myImage"
' 画像を右揃えにします。
ri.X = "parent.right-width"
' テキストを左揃えにします。
rt.X = "0"
rt.Y = "myImage.height/2-height/2"
ra.Children.Add(ri)
ra.Children.Add(rt)
doc.Body.Children.Add(ra)
```

**C# コードの書き方**

C#

```
C1PrintDocument doc = new C1PrintDocument();
RenderText rt = new RenderText("test");
RenderImage ri = new RenderImage(myImage);
RenderArea ra = new RenderArea();
ri.Name = "myImage";
// 画像を右揃えにします。
ri.X = "parent.right-width";
// テキストを左揃えにします。
rt.X = "0";
```


Copyright © GrapeCity, Inc. All rights reserved.

```
rt.Y = "myImage.height/2-height/2";
ra.Children.Add(ri);
ra.Children.Add(rt);
doc.Body.Children.Add(ra);
```

# 描画オブジェクトの影

**Reports for WinForms** には、描画オブジェクトによるシャドウキャストのサポートが含まれます。パブリック **IShadow** インタフェースは、パブリック構造体 **Shadow** によって実装され、非アンビエントパブリックプロパティ **Shadow** によって公開されます。

次のサブプロパティが含まれます。

| プロパティ | 説明 |
|---|---|
| **透明度** | 影の透明度をパーセントで取得または設定します。0の値は塗りつぶした（不透明な）影を定義し、100 の値（デフォルト）は完全に透明（非表示）な影を定義します。 |
| **サイズ** | オブジェクトのサイズに相対的な影のサイズをパーセントで取得または設定します。100 の値（デフォルト）は、影がオブジェクトと同じサイズを持つことを示します。 |
| **Distance** | 影の中心がオブジェクトの中心からのオフセットである距離を取得または設定します。絶対単位値（「0.5in」や「4mm」など）のみがこのプロパティに割り当てられることに注意してください。デフォルトは 2mm です。 |
| **Angle** | 影の角度を度単位で取得または設定します。角度は、3時の位置に相対的に時計回りに測定されます。デフォルトは45です。 |
| **Color** | 影の色を取得または設定します。デフォルトは Black です。 |

次のサンプルコードは、描画オブジェクトの影を定義します。

**Visual Basic コードの書き方**

Visual Basic

```
Dim doc As C1PrintDocument = C1PrintDocument1
Dim rt As New RenderText("Sample Shadow")
rt.Width = Unit.Auto
rt.Style.Shadow.Transparency = 20
rt.Style.Shadow.Color = Color.BurlyWood
doc.Body.Children.Add(rt)
```

**C# コードの書き方**

C#

```
C1PrintDocument doc = c1PrintDocument1;
RenderText rt = new RenderText("Sample Shadow");
rt.Width = Unit.Auto;
rt.Style.Shadow.Transparency = 20;
rt.Style.Shadow.Color = Color.BurlyWood;
doc.Body.Children.Add(rt);
```

影のプロパティの設定時に Shadow オブジェクトを作成する必要はありませんが、たとえば、次のように作成することも選択できます。


Copyright © GrapeCity, Inc. All rights reserved.

**Visual Basic コードの書き方**

```
Visual Basic
Dim doc As C1PrintDocument = C1PrintDocument1
Dim rt As New RenderText("Sample Shadow")
rt.Width = Unit.Auto
rt.Style.Shadow = New Shadow(20, 100, "1mm", 45, Color.CadetBlue)
doc.Body.Children.Add(rt)
```

**C# コードの書き方**

```
C#
C1PrintDocument doc = c1PrintDocument1;
RenderText rt = new RenderText("Sample Shadow");
rt.Width = Unit.Auto;
rt.Style.Shadow = new Shadow(20, 100, "1mm", 45, Color.CadetBlue);
doc.Body.Children.Add(rt);
```

**注意**:レイアウト目的の場合、影はオブジェクトのサイズに影響しません。

# オブジェクトの枠線

**Reports for WinForms** には、新しい方法の枠線のレイアウトと配置へのサポートが含まれています。枠線のレイアウトと配置のこのメソッドは主に、RDL 互換性のために追加されましたが、それ自身にも役立てることができます。たとえば、現在はオプションで、オブジェクトのサイズや周囲のオブジェクトの位置にも影響を及ぼさずに、枠線をオブジェクトの範囲で中央揃えできるようになっています。

次のパブリックタイプによりこの機能が容易に使用できます。

| タイプ | 説明 |
|---|---|
| **BordersModeEnum** | オブジェクトをドキュメントにレイアウトする際の枠線の太さを示すさまざまなモードを指定します。 |

**BordersModeEnum** には次のメンバが含まれます。

- **Default**: 枠線全体はオブジェクトの一部であると見なされます。これは C1PrintDocument のオブジェクトのデフォルト動作です。
- **C1Report**: 枠線の太さの内側 1/2 はオブジェクトの一部と見なされ、枠線の外側 1/2 はオブジェクトのスペース外と見なされます。これは C1Report のオブジェクトのデフォルト動作です(MS Access の場合と同じ)。
- **Rdl**: 枠線の太さはオブジェクトのサイズとレイアウトの計算時にまったく考慮されません。枠線はオブジェクトの範囲に中央揃えにして描画されます

# スタイル

**C1PrintDocument**の基本的な機能の1つにスタイルがあります。スタイルは、ドキュメントのコンテンツの表現に使用されるビジュアルな属性のほとんどを提供します。スタイルを使用する際に留意しておく必要がある主な点を以下に説明します。

# Style プロパティを持つクラス

ドキュメント内で独自のビジュアル表現を持つすべてのオブジェクトには、スタイルが関連付けられています。具体的には、次のクラスが **Style** プロパティを持ちます。


Copyright © GrapeCity, Inc. All rights reserved.

- ドキュメント全体(**C1PrintDocument**).
- 描画オブジェクト(**RenderObject** とすべての派生クラス).
- 段落オブジェクト(**ParagraphObject**から派生される**ParagraphText**, と**ParagraphImage**)
- テーブルセル (**TableCell**).
- テーブル内のセルのユーザー定義グループ(**UserCellGroup**).
- **TableVector**テーブルの行と列(から派生される **TableRow** と **TableCol**).
- テーブルの行や列のグループ - テーブルヘッダー、テーブルフッターなど(**TableVectorGroup**).

## インラインスタイルと非インラインスタイル

**C1PrintDocument**には、インラインと非インラインの2種類のスタイルがあります。オブジェクトが**Style** プロパティを持つ場合、そのプロパティはオブジェクトのインラインスタイルを参照します。インラインスタイルはオブジェクトに不可欠です。インラインスタイルは削除も設定もできません。これは、オブジェクトと共に常に存在するスタイルインスタンスを参照する読み取り専用プロパティです。このため、スタイルのプロパティはオブジェクト自身のプロパティと見なすことができます。ただし、継承のため、スタイルには大きな柔軟性があり、メモリ効率も優れています。

さらに、各 **Style** には、どのオブジェクトにも直接アタッチされていないスタイルのコレクション**Children**が含まれています。代わりに、これらの**非インライン**スタイルを親スタイル（スタイルのプロパティ **Parent** と **AmbientParent** を参照）として使用して、他のスタイル**インラインスタイルを含む**の継承プロパティの値を提供できます。

スタイルオブジェクト自体を独立に作成することはできません。スタイルオブジェクトは常に、描画オブジェクトなどのドキュメントの要素に直接アタッチされたインラインスタイル、または別のスタイルの **Children** コレクションのメンバとして作成されます。

たとえば、次のコードはコンパイルされません。

**Visual Basic コードの書き方**

Visual Basic

```
Dim doc As New C1PrintDocument()
Style s = new Style() 'コンパイルされません。
s.Borders.All = New LineDef("1mm", Color.Red)
Dim rt As New RenderText("My text.")
rt.Style = s
```

**C# コードの書き方**

C#

```
C1PrintDocument doc = new C1PrintDocument();
Style s = new Style(); // コンパイルされません。
s.Borders.All = new LineDef("1mm", Color.Red);
RenderText rt = new RenderText("My text.");
rt.Style = s;
```

一方、このコードはコンパイルされ、希望する結果が得られます。

**Visual Basic コードの書き方**

Visual Basic

```
Dim doc As New C1PrintDocument()
Dim s As doc.Style.Children.Add()
s.Borders.All = New LineDef("1mm", Color.Red)
RenderText rt = New RenderText("My text.")
rt.Style.Parent = s
```


Copyright © GrapeCity, Inc. All rights reserved.

**C# コードの書き方**

C#

```
C1PrintDocument doc = new C1PrintDocument();
Style s = doc.Style.Children.Add();
s.Borders.All = new LineDef("1mm", Color.Red);
RenderText rt = new RenderText("My text.");
rt.Style.Parent = s;
```

アンビエントおよび非アンビエントスタイル属性と親についての詳細は、アンビエントおよび非アンビエントスタイルプロパティを参照してください。

# アンビエントスタイルプロパティと非アンビエントスタイルプロパティ

すべてのスタイルプロパティ(**フォント**、**色**、**境界線**など)は、その意味に基づいてアンビエントと非アンビエントの2つのグループに分類できます。アンビエントプロパティはオブジェクトのコンテンツに影響するプロパティ(テキストフォントなど)で、非アンビエントプロパティはオブジェクトの装飾の描画に使用されるプロパティ(オブジェクトの周囲の境界線など)です。通常、この2つのグループでは必要な動作(**特に継承規則**)が異なるため、この区別は自然で便利です。アンビエントプロパティは、通常、オブジェクトの階層を下方に伝達されることが適しています。つまり、コンテナオブジェクトのアンビエントプロパティの設定が、そのコンテナ内のすべてのオブジェクトに影響するということです。テーブルを例にすれば、テーブル自体にフォントを設定した場合は、通常、テーブル内のすべてのセルのテキストにもそのフォントが適用される方が自然です。非アンビエントプロパティでは、この事情が大きく異なります。たとえば、テーブルの前後に余白を追加する場合は、テーブルの **Spacing** プロパティを設定しますが、その余白がすべてのセルに伝達されると不都合です。**C1Preview** のスタイルでは、このようにスタイルプロパティがアンビエントグループと非アンビエントグループに分類されています。基本的に、通常はオブジェクトのスタイルにプロパティを設定するだけで(あまりアンビエントについて考えなくても)、適切に処理されます。

すべてのスタイルプロパティの完全なリスト(プロパティがアンビエントであるか、そうでないかを示す)については、下記の別個のセクションで示します(スタイルプロパティとそのデフォルト値を参照してください)。ただし、次の一般的なルールが適用されます。

アンビエントスタイルプロパティは、オブジェクトのコンテンツ(テキストフォントなど)の表示を制御します。デフォルトでは、アンビエントプロパティは、オブジェクトの包含階層を下方に伝播します。つまり、コンテナオブジェクトのスタイルに設定されたアンビエントプロパティは、そのコンテナ内のすべてのオブジェクトに適用されます。

非アンビエントスタイルプロパティは、オブジェクトの装飾(**枠線**など)を制御します。デフォルトでは、非アンビエントプロパティは、オブジェクトの包含を通じて伝わりませんが、**Style.Parent** プロパティを介してスタイル全体に伝えることができます。

# スタイルの継承 - Parent と AmbientParent

スタイルが適用されるオブジェクトの外観に影響するすべてのスタイルプロパティ(**Font**、**BackColor** など)は、設定されているか設定されていないかのいずれかの状態です。最初、新しく作成されたスタイルのプロパティはどれも設定されていません。これらの値を参照することはできますが、継承を通して別のスタイルまたはオブジェクトから取得されます(下記を参照)。ただし、プロパティが設定されている場合、その設定値はスタイル自体に保存され、継承を通して他のスタイルの影響を受けることはなくなります。

スタイルには、スタイルの継承をサポートする2つの特別なプロパティ **Parent** と **AmbientParent** があります。**Parent** プロパティは、現在のスタイルに設定されていない非アンビエントプロパティの値を提供するスタイルを取得または設定します。**AmbientParent** プロパティは、現在のスタイルに設定されていないアンビエントプロパティの値を提供するスタイルを取得または設定します。デフォルトでは、新しく作成されたインラインスタイルの親はどちらも空(C# の場合は null 値、VB の場合は Nothing)ですが、スタイルの **Children** コレクションに作成されたスタイルの場合、**Parent** はそのコレクションを保持するスタイルに設定され、**AmbientParent** は空になります。

スタイルの **Parent** が指定されていない場合、非アンビエントプロパティの値は、静的なデフォルトから取得されます(下の「**テーブル**」を参照)。スタイルの **AmbientParent** が指定されていない場合、アンビエントプロパティの値は、現在のスタイルが適用されるオブジェクトを保持するオブジェクトから取得されます。


Copyright © GrapeCity, Inc. All rights reserved.

たとえば、**RenderTextrt** が**RenderArea** ra に含まれている場合は、次の規則に基づいて、テキストのレンダリングに使用されるフォントが取得されます。

- **rt.Style.Font** が設定されている場合は、それが使用されます。
- そうではなく、**rt.Style.AmbientParent** が **null** でない場合は、**Font** がアンビエントプロパティなので、**rt.Style.AmbientParent.Font** が使用されます。
- **rt.Style.AmbientParent** が **null**（デフォルト）の場合は、包含するオブジェクトのフォント（**ra.Style.Font**）が使用されます。

同じ例を使用すると、次の規則に基づいて、**RenderText** オブジェクトの **BackgroundImage** が取得されます。

- **rt.Style.BackgroundImage** が設定されている場合は、それが使用されます。
- そうでなく、**rt.Style.Parent** が null ではない場合は、**rt.Style.Parent.BackgroundImage** が使用されます。
- それ以外の場合は、背景画像のデフォルト値（画像なし）が使用されます。

デフォルトでは、スタイルの **Children** コレクションに含まれるスタイルの **Parent** は、その **Children** コレクションの所有者であるスタイルに設定されていますが、これは変更できることに注意してください。たとえば、**Children** コレクションに含まれる1つのスタイルの親を同じコレクション内の別のスタイルに設定することもできます。**Children** コレクションは、関連するスタイルのグループ化や保存に便利な場所を提供するだけで、実際にスタイルの階層には何も制限が適用されません。

## スタイルプロパティとそのデフォルト値

次の表に、スタイルが適用されるオブジェクトの外観に影響するすべてのスタイルプロパティを示し、各プロパティがアンビエントかどうかと、プロパティのデフォルト値を示します。

| プロパティ名 | アンビエント | デフォルト値 |
|---|---|---|
| ActiveHyperlinkAttrs | はい | |
| BackColor | | 空 |
| BackgroundImage | | なし |
| BackgroundImageAlign | | 左／上に配置、水平／垂直方向に引き伸ばし、縦横比を保持 |
| BackgroundImageName | | なし |
| Borders | | すべて空 |
| Brush | | なし |
| CharSpacing | はい | 0 |
| CharWidth | はい | 100 |
| ClientAreaOnly | | False |
| FlowAlign | | デフォルトフローの配置 |
| FlowAlignChildren | | 近接配置 |
| Font | はい | Arial, 10pt |
| FontBold | はい | False |
| FontItalic | はい | False |
| FontName | はい | Arial |
| FontSize | はい | 10 |
| FontStrikeout | はい | False |


Copyright © GrapeCity, Inc. All rights reserved.

| プロパティ名 | アンビエント | デフォルト値 |
|---|---|---|
| FontUnderline | はい | False |
| GridLines | | なし |
| HoverHyperlinkAttrs | はい | |
| HyperlinkAttrs | はい | 青 |
| ImageAlign | はい | 左／上に配置、水平／垂直方向に引き伸ばし、縦横比を保持 |
| JustifyEndOfLines | はい | True |
| JustifyLastLine | はい | False |
| LineSpacing | はい | 100% |
| MeasureTrailingSpaces | はい | False |
| MinOrphanLines | | 0 |
| Padding | | すべてゼロ |
| ShapeFillBrush | | なし |
| ShapeFillColor | | 透明 |
| ShapeLine | | 黒、0.5pt |
| Spacing | | すべてゼロ |
| TextAlignHorz | はい | Left |
| TextAlignVert | はい | Top |
| TextAngle | | 0 |
| TextColor | はい | 黒 |
| TextIndent | | 0 |
| TextPosition | | 標準 |
| VisitedHyperlinkAttrs | はい | マゼンダ |
| WordWrap | はい | True |

## 複雑なスタイルプロパティのサブプロパティ

スタイルプロパティとそのデフォルト値ピックの表に示すいくつかのプロパティには、個別に設定できるサブプロパティがあります。たとえば、**BackgroundImageAlign** プロパティには、**AlignHorz**、**AlignVert**、およびその他いくつかのサブプロパティがあります。読み取り専用サブプロパティ（変えることができず、個々のサブプロパティを設定できないフォント付きの場合など）を除き、各サブプロパティは個別に設定または継承することができます。

フォントのサブプロパティは変更できませんが、これらのサブプロパティのそれぞれは、**FontBold**、**FontItalic** など、独立したスタイルのルートレベルのプロパティによって表されます。これらの各サブプロパティは個別に設定され、一般的なスタイル継承ルールに従います。ただし、考慮しなければならない微妙な違いがあります。**Font** と独立したフォント関連プロパティ（**FontBold**、**FontItalic** など）の両方が設定される場合、その結果は2つのプロパティが設定される順序によって異なります。最初に **Font** が設定されると、フォント関連プロパティが変更され（つまり、**FontItalic** が **True** に設定され）、その変更が結果に影響を与えます。それとは逆に、最初に **FontItalic** が **True** に設定され、次に **Font** がイタリック以外のフォントに設定されると、**FontItalic** の変更は失われます。


Copyright © GrapeCity, Inc. All rights reserved.

# 計算されたスタイルプロパティ

**Reports for WinForms** の 2009 v3 リリースでは、計算されたスタイルプロパティへのサポートが追加されました。各スタイルプロパティに、「Expr」が付加された同じ名前で文字列が一致するプロパティが追加されました。たとえば、**BackColorExpr** と**TextColorExpr** プロパティが **BackColor** と **TextColor** プロパティに一致するというように、すべてのプロパティで続きます。

複合プロパティのサブプロパティ（**ImageAlign**、**Borders** など）も、式が一致するサブプロパティを持ちます。たとえば、**LeftExpr** プロパティは **Left** プロパティに一致するというように続きます。

## スタイル式

次のオブジェクトはスタイル式内で使用できます。

- **RenderObject**：現在のスタイルのオーナー描画オブジェクト。
- **Document**：現在のドキュメント。
- **Page**（および **PageNo** などの他のページ関連オブジェクト）：オブジェクトを格納するページ（下の注記を参照）。
- **RenderFragment**：現在のフラグメント。
- **Aggregates**。
- **Fields, DataBinding**：現在のデータソースを参照します。スタイルのオーナーがテーブル内にあり、データソースが行と列の両方に指定された場合は、列に定義されたデータソースを参照します。
- **RowNumber**：関連付けられたデータソースの行番号。
- **ColFields, ColDataBinding**：スタイルがテーブル内で使用された場合にのみアクセス可能で、列に定義されたデータソースを参照します。
- **RowFields, RowDataBinding**：スタイルがテーブル内で使用された場合にのみアクセス可能で、行に定義されたデータソースを参照します。

## 変換型

- 計算されたスタイルプロパティ値がオブジェクトのレンダリングに使用される実際の値に割り当てられると、タイプは次の規則に従って変換されます。
- ターゲットプロパティが数値型（**int**、**float** など）の場合、必要に応じて、計算された値は必要な数値型に変換（**string**、**rounded** などから変換）されます。
- ターゲットプロパティが単位（スペースなど）で、式が数値を生成する場合、単位は **new Unit(Document.DefaultUnitType, value)** コンストラクタを使用して作成されます。式が文字列を生成した場合、この文字列は通常の単位規則を使用して解析されます。
- それ以外の場合はすべて、**TypeConverter** を使用してこの値をターゲットタイプに変換する試みが行われます。
- 最後に、式が **null** を生成した場合、プロパティが現在のスタイルに指定されなかったかのように、親のスタイル値が使用されます（未指定のスタイルプロパティのデフォルト動作など）。

## ページ参照

以下は、ページ参照に関連する重要な注記です。スタイル式は現在のページを参照できます。たとえば、次のようになります。

```
ro.Style.BackColor = "iif(PageCount < 3, Color.Red, Color.Blue)";
```

この式はドキュメントの生成時に計算できません。そのため、生成時にこの式は無視（デフォルト値を使用）され、値は、オブジェクトを格納する実際のページがレンダリングされる（プレビューへの描画、エクスポートなど）ときに計算されます。

その結果、改ページ制御に依存しかつドキュメントのレイアウトに影響するスタイル式は、予期しない望ましくない結果を生成する可能性があります。たとえば、次の式がフォントサイズに使用された場合

```
ro.Style.FontSize = "iif(PageCount < 3, 20, 30)";
```

上の式は生成時に無視されます。その結果、レンダリングされたテキストは計算されたオブジェクトのサイズには大きすぎる可能性が非常に高く、クリッピングが発生します。


Copyright © GrapeCity, Inc. All rights reserved.

# 段落オブジェクトのスタイル

**RenderParagraph** オブジェクトは（他の描画オブジェクトと同様に）**Style** プロパティを持ち、このプロパティを使用して、段落全体に適用されるスタイルプロパティを設定できます。個別の段落オブジェクト(**ParagraphText** と **ParagraphImage**) もスタイルを持ちますが、段落オブジェクトに適用されるスタイルプロパティは、次のプロパティに制限されます。

- **BackColor**
- **Brush** (塗りつぶしブラシの場合のみ)
- **Font** (FontBoldなどのフォント関連のプロパティ)
- **HoverHyperlinkAttrs**
- **TextColor**
- **TextPosition**
- **VisitedHyperlinkAttrs**

# テーブルのスタイル

テーブルでは、オブジェクトの表示に影響を与えるスタイルの数が劇的に増加します。通常の包含（テーブルは、ドキュメントのその他すべての要素と同様に、他の表示オブジェクト内、または最上位レベルのドキュメントの本文内のいずれかに含まれます）に加えて、テーブル内のオブジェクトもすべてが独自のスタイルを持つセル、行および列に属します。さらに、オブジェクトは、多くのテーブル要素グループに属することができるので、処理は一層複雑になります。テーブル内でスタイルがどのように動作するかについての詳細は、テーブル内のスタイルトピックを参照してください。

# テーブル

テーブルは、**RenderTable** クラスのインスタンスによって表されます。テーブルを作成するには、次の例に示すように、単にそのコンストラクタを呼び出します。

**Visual Basic コードの書き方**

Visual Basic

```
Dim rt1 As New C1.C1Preview.RenderTable()
```

**C# コードの書き方**

C#

```
RenderTable rt1 = new RenderTable();
```

**C1PrintDocument** テーブルは、Microsoft Excel のモデルに準拠します。新規に作成されたテーブルは、物理的には空（つまり、メモリ内に多くのスペースを占有していません）ですが、論理的には無限です。最初に行または列を追加することなく、テーブルの任意の要素（セル、行、または列）にアクセスし、その要素に書き込むことで、その要素の前に来るすべての要素を論理的に作成できます。たとえば、行インデックスが9、列インデックスが3の空のテーブルのセルのテキストを設定すると、テーブルは 10 行と4列に拡大されます。

テーブルにコンテンツを追加するには、セルにデータを入力します。これは、次の方法のいずれかを使用して実行できます。

- セルの **RenderObject** プロパティを任意の表示オブジェクトに設定する。これにより、指定した表示オブジェクトがそのセルに挿入されます。別のテーブルを含む任意の表示オブジェクトをセルに追加できます。その結果、テーブルをネストできます。
- セルの **Text** プロパティを文字列に設定する。これは、実際にはテキストのみのテーブルを作成するための便利なショートカットです。新しい**RenderText** オブジェクトを作成し、セルの **RenderObject** プロパティをその **RenderText** に設定し、そのオブジェクトの **Text** プロパティを指定した文字列に設定します。

たとえば、次のコードスニペットは、10 行と4列のテーブルを作成します。


Copyright © GrapeCity, Inc. All rights reserved.

**Visual Basic コードの書き方**

```
Visual Basic
Dim rt1 As New C1.C1Preview.RenderTable()
Dim row As Integer = 0
Do While (row < 10)
    Dim col As Integer = 0
    Do While (col < 4)
        rt1.Cells(row, col).Text = String.Format( _
          "Text in cell({0}, {1})", row, col)
        col += 1
    Loop
    row += 1
Loop
```

**C# コードの書き方**

```
C#
RenderTable rt1 = new RenderTable();
for (int row = 0; row < 10; ++row)
{
    for (int col = 0; col < 4; ++col)
        rt1.Cells[row, col].Text = string.Format(
          "Text in cell({0}, {1})", row, col);
}
```

**Cols.Count** プロパティ(現在の列の数を返します)と **Rows.Count** プロパティ(現在の行の数を返します)の値を照会することによって、実際のテーブルの現在のサイズをいつでも確認できます。

# セル、列、および行へのアクセス

テーブルトピックのサンプルコードからわかるように、テーブル内のすべてのセルは、**TableCellCollection** 型を持つ**Cells** コレクションによって表されます。この個々のセルを表すコレクション内の要素は、**TableCell** 型を持っています。テーブル内の任意のセルにアクセスするために、**Cells** コレクションは、次のコードに示すようにセルの行と列にインデックスを付けることができます。

**Visual Basic コードの書き方**

```
Visual Basic
Dim rt As New C1.C1Preview.RenderTable()
…
' 行 10、列 4 のセルを取得します。
Dim tc as TableCell = rt.Cells(10, 4)
```

**C#コードの書き方**

```
C#
RenderTable rt = new RenderTable();
…
// 行 10、列 4 のセルを取得します。
TableCell tc = rt.Cells[10, 4];
```


Copyright © GrapeCity, Inc. All rights reserved.

テーブルの列は、**TableColCollection** 型を持ち、**TableCol**型の要素を含む**Cols** コレクションを通じてアクセスされます。セルと同様に、列を「タッチ」するだけで列が作成されます。たとえば、列の**Style**プロパティを設定すると、列が存在しない場合でも、その列が作成されます。

テーブルの行は、**TableRowCollection** 型を持ち、**TableRow**型の要素を含む**Rows** コレクションを通じてアクセスされます。セルおよび列と同様に、行を「タッチ」するだけで行が作成されます。たとえば、行の **Height** を設定すると、行（およびその前のすべての行）が自動的に作成されます。

ただし、実際のコンテンツを含まないテーブルのすべての行は、高さがゼロとなり、そのためテーブルを表示しても目に見えません。

## テーブルと列の幅、行の高さ

**C1PrintDocument** テーブルの行と列のいずれもサイズが自動調整されますが、デフォルトの動作は行と列で異なります。デフォルトでは、行の高さは自動調整されますが（行内のセルのコンテンツに基づいて計算されます）、列の幅は固定されます。**RenderTable** のデフォルトの幅は、その親の幅と同じであり、その幅はすべての列で均等に共有されます。次のコードは、同じ幅の3列、セルのコンテンツの高さに自動調整された高さの 10 行を含む、ページ全体にわたるテーブルを作成します。

**Visual Basic コードの書き方**

Visual Basic

```
Dim rt As New C1.C1Preview.RenderTable()
rt.Style.GridLines.All = LineDef.Default
Dim row As Integer = 0
Do While (row < 10)
  Dim col As Integer = 0
    Do While (col < 3)
      rt.Cells(row, col).Text = String.Format( _
          "Cell({0},{1})", row, col)
      col += 1
    Loop
  row += 1
Loop
doc.Body.Children.Add(rt)
```

**C#コードの書き方**

C#

```
RenderTable rt = new RenderTable();
rt.Style.GridLines.All = LineDef.Default;
for (int row = 0; row < 10; ++row)
  for (int col = 0; col < 3; ++col)
    rt.Cells[row, col].Text = string.Format(
        "Cell({0}, {1})", row, col);
doc.Body.Children.Add(rt);
```

サイズが完全に自動調整されるテーブルを作成するには、デフォルト設定に比べて、次の2つの処理を行う必要があります。

- テーブル全体の幅を **Auto** に設定する必要があります（文字列を**auto**に設定するか、スタティックフィールドを **Unit.Auto** に設定します）。

  および


Copyright © GrapeCity, Inc. All rights reserved.

- テーブルの **RenderTable.ColumnSizingMode** を **TableSizingModeEnum.Auto** に設定する必要があります。

以下に変更されたコードを示します。

**Visual Basic コードの書き方**

Visual Basic

```
Dim rt As New C1.C1Preview.RenderTable()
rt.Style.GridLines.All = LineDef.Default
Dim row As Integer = 0
Do While (row < 10)
  Dim col As Integer = 0
    Do While (col < 3)
      rt.Cells(row, col).Text = String.Format( _
          "Cell({0},{1})", row, col)
      col += 1
    Loop
  row += 1
Loop
rt.Width = Unit.Auto
rt.ColumnSizingMode = TableSizingModeEnum.Auto
doc.Body.Children.Add(rt)
```

**C# コードの書き方**

C#

```
RenderTable rt = new RenderTable();
rt.Style.GridLines.All = LineDef.Default;
for (int row = 0; row < 10; ++row)
  for (int col = 0; col < 3; ++col)
    rt.Cells[row, col].Text = string.Format(
        "Cell({0}, {1})", row, col);
rt.Width = Unit.Auto;
rt.ColumnSizingMode = TableSizingModeEnum.Auto;
doc.Body.Children.Add(rt);
```

変更されたコードは、テーブルの各列の幅を、セル内のすべてのテキストに対応する必要がある限度に設定します。

# 行と列のグループ - ヘッダーとフッター

テーブルの強力な機能の1つに、要素のグループ化があります。グループ化を使用すると、テーブルの複数の要素にまとめてアクセスできます。たとえば、それが1つの要素であるかのようにグループにスタイルを設定できます。列のグループ化、行のグループ化、およびセルのグループ化がサポートされています。

行のグループにアクセスするには、**RowGroups** コレクション (**TableVectorGroupCollection** 型).を使用します。そのコレクションの要素は**TableVectorGroup**型で、この型にいくつかの便利なプロパティが定義されています。このプロパティの中でも注目されるプロパティは、**ColumnHeader**です。このプロパティを使用すると、行のグループをテーブルヘッダーに指定して、それをページまたはページ列ごとに先頭に繰り返し表示できます。関連するプロパティは**ColumnFooter**です。これを使用すると、行のグループをテーブルフッターに指定して、それをページまたはページ列ごとに末尾に繰り返し表示できます。

次のコードは、テーブルの最初の2行をテーブルヘッダーに指定して、それをページ区切りまたは列区切りの後に繰り返し表示する方法を示します(rt1 は **RenderTable** オブジェクト)。


Copyright © GrapeCity, Inc. All rights reserved.

**Visual Basic コードの書き方**

```
Visual Basic
rt1.RowGroups(0, 2).Header = C1.C1Preview.TableHeaderEnum.Page
```

**C# コードの書き方**

```
C#
rt1.RowGroups[0, 2].Header = C1.C1Preview.TableHeaderEnum.Page;
```

上に示したように、**TableVectorGroupCollection** クラスのインデクサは2つの整数を受け取ります。最初の値は、グループに含まれる最初の行のインデックスです（上のサンプルでは0）。2番目の値は、グループ内の行数です（上のサンプルでは2）。

列のグループにアクセスするには、**ColGroups** コレクションを使用する必要があります。これは、行グループのコレクション（**TableVectorGroupCollection**）と同じ型で、同じ機能を提供します。特に注目される点は、列のグループを垂直方向のテーブルヘッダーまたはテーブルフッターに指定する機能です。**C1PrintDocument** は「水平方向」（または「拡張」）ページをサポートします。これを使用すると、広いオブジェクトを水平方向の複数のページにまたがって表示できます。オブジェクト（テーブルなど）を水平方向の複数のページにまたがって表示するには、**SplitHorzBehavior** を true に設定します。オブジェクトの幅がページ幅より広い場合は、複数の水平方向ページに分割されます。特に、この方法で広いテーブルを分割することができます。列のグループを各ページの左端に沿って繰り返し表示するには、グループの **ColumnHeader** プロパティを **true** に設定します。列のグループを各ページの右端に沿って繰り返し表示するには、グループの **ColumnFooter** プロパティを **true** に設定します

> **メモ**:テーブルの任意の行（または列）のグループをフッターに割り当てることができますが、通常は、テーブルの最後の何行（または列）かをフッターグループに入れます。これで、フッターが通常のフッターとして動作するようになります。つまり、フッターは各ページの下部（または右端）にだけ表示され、さらにテーブルの最後にも表示されます。たとえば、テーブルの最初の行をフッターに指定すると、テーブルの先頭にも表示され、テーブルの最後には表示されなくなります。

次のコード例では、100 行 10 列のテーブルを作成し、テーブルの幅を自動に設定します。さらに、各列の幅を明示的に1インチに設定し、水平方向および垂直方向のテーブルヘッダーとテーブルフッターも割り当てます。

**Visual Basic コードの書き方**

```
Visual Basic
'テーブルを作成し、内容を挿入します
Dim rt1 As C1.C1Preview.RenderTable = New C1.C1Preview.RenderTable()
Dim row As Integer = 0
Dim col As Integer
Do While (row < 100)
    col = 0
    Do While (col < 6)
        rt1.Cells(row, col).Text = String.Format("Text in cell({0}, {1})", row, col)
        col += 1
    Loop
    row += 1
Loop

' テーブルと列の幅を設定します
rt1.Width = C1.C1Preview.Unit.Auto
col = 0
Do While (col < 6)
```


Copyright © GrapeCity, Inc. All rights reserved.

```
    rt1.Cols(col).Width = "1in"
    col += 1
Loop

' 最初の2行をヘッダーとして割り当てます。背景を設定します
rt1.RowGroups(0, 2).PageHeader = True
rt1.RowGroups(0, 2).Style.BackColor = Color.Red

'最後の2行をフッターとして割り当てます。背景を設定します
rt1.RowGroups(98, 2).PageFooter = True
rt1.RowGroups(98, 2).Style.BackColor = Color.Blue

'最初の列をヘッダーとして割り当てます
rt1.ColGroups(0, 1).PageHeader = True
rt1.ColGroups(0, 1).Style.BackColor = Color.BlueViolet

' 最後の列をフッターとして割り当てます
rt1.ColGroups(5, 1).PageFooter = True
rt1.ColGroups(5, 1).Style.BackColor = Color.BurlyWood
```

**C# コードの書き方**

C#

```
// テーブルを作成し、内容を挿入します
RenderTable rt1 = new RenderTable();
for (int row = 0; row < 100; ++row)
{
    for (int col = 0; col < 6; ++col)
    {
        rt1.Cells[row, col].Text = string.Format("Text in cell({0}, {1})", row, col);
    }
}

// テーブルと列の幅を設定します
rt1.Width = Unit.Auto;
for (int col = 0; col < 6; ++col)
{
    rt1.Cols[col].Width = "1in";
}

//最初の2行をヘッダーとして割り当てます。背景を設定します
rt1.RowGroups[0, 2].PageHeader = true;
rt1.RowGroups[0, 2].Style.BackColor = Color.Red;

// 最後の2行をフッターとして割り当てます。背景を設定します
rt1.RowGroups[98, 2].PageFooter = true;
rt1.RowGroups[98, 2]. Style.BackColor = Color.Blue;

//最初の列をヘッダーとして割り当てます
rt1.ColGroups[0, 1].PageHeader = true;
rt1.ColGroups[0, 1].Style.BackColor = Color.BlueViolet;
```


Copyright © GrapeCity, Inc. All rights reserved.

```
// 最後の列をフッターとして割り当てます
rt1.ColGroups[5, 1].PageFooter = true;
rt1.ColGroups[5, 1].Style.BackColor = Color.BurlyWood;
```

このサンプルでは、背景色を使って行グループと列グループを強調表示しています。

## ユーザーセルグループ

セルは、離れていてもグループにまとめることができます。これで、1つのコマンドで簡単にグループ内のすべてのセルにスタイルを設定できます。ユーザーセルグループを定義するには、次の手順に従います。

1. **UserCellGroup**型のオブジェクトを作成します。このクラスには、いくつかのオーバーロードコンストラクタがあり、これを使用して、グループに入れるセルの座標を指定できます。すべてのセルをコンストラクタでグループに追加する必要があります。
2. 作成した **UserCellGroup** オブジェクトをテーブルの**UserCellGroups** コレクションに追加します。
3. これで、グループにスタイルを設定できます。スタイルの設定はグループ内のすべてのセルに影響します。

## テーブル内のスタイル

テーブルのセル、列、および行は表示オブジェクトではありませんが（これらは、**RenderObject** クラスから派生しません）、表示オブジェクトのプロパティに類似したいくつかのプロパティを備えています。特に、すべてに **Style** プロパティが用意されています。

スタイルの操作は、対応する要素とそのすべてのコンテンツに影響を与えます。行のスタイルの設定は、その行内のすべてのセルに影響を与えます。列のスタイルの設定は、その列内のすべてのセルに影響を与えます。行と列の交点にあるセルのスタイルは、行と列に指定されたスタイルの組み合わせになります。行と列の両方に同じスタイルプロパティが設定されている場合、列が優先します。

さらに、グループ（行のグループ、列のグループ、およびユーザーのセルグループ）のすべてが独自のスタイルを持ち、セル内のデータの表示とテーブルの行と列の表示にも影響を与えます。

以下のルールは、テーブル内のスタイルのアプリケーションを制御します。

アンビエントスタイルプロパティは、「幾何学的な」包含に基づいて、テーブル要素（テーブル全体、行と列のグループ、セルのグループ、個別の行とセル、および個別のセル）を下方に伝播します。この動作は、アンビエントスタイルプロパティがテーブルの外部にある表示オブジェクトの包含を下方に伝播する方法に似ています。

アンビエントスタイルプロパティは、セルのコンテナの要素に影響を与えることなく、これらのコンテンツに影響を与えます。たとえば、テーブル全体のスタイルにフォントを設定すると、下位レベルにフォントが明示的に設定されていないかぎり、そのテーブル内のすべてのテキストに影響を与えます。同様に、そのテーブル内の行のスタイルにフォントを設定すると、その行内のすべてのセルに影響を与えます。

関連する複数のテーブル要素で特定のアンビエントプロパティを変更する場合、以下の優先順位が使用して、セルの表示に使用するプロパティの実効値が計算されます。

- セル自体のスタイル（最も高い優先順位）
- UserCellGroup スタイル（セルが含まれている場合）
- 列のスタイル
- 列グループのスタイル（存在する場合）
- 行のスタイル
- 行グループのスタイル（存在する場合）
- テーブルのスタイル（最も低い優先順位）

上記にリストしたテーブル要素のスタイルに設定された非アンビエントプロパティ（テーブル全体、行、列とセルのグループ、行、列とセル）は、これらの要素（テーブル全体を除き）が表示オブジェクトでない場合でも、セルのコンテンツに影響を与えることなく、これらの要素そのものに適用されます。たとえば、テーブル内の行の周囲に枠線を表示するには、行の **Style.Borders** を希望する値に設定します。


Copyright © GrapeCity, Inc. All rights reserved.

テーブル内のすべてのセルに非アンビエントプロパティを設定するには、**RenderTable.CellStyle** を使用します。スタイルが指定されると、そのスタイルはセル内の表示オブジェクトのスタイルの親として有効に使用されます。

また、**CellStyle** プロパティが行、列、およびテーブル要素のグループに対して定義され、指定された場合、これらのスタイルのすべてがセル内のオブジェクトの非アンビエントプロパティに影響を与えます。たとえば、テーブル内のすべてのセルの背景画像を設定するには、テーブルの **CellStyle.BackgroundImage** を設定します。この操作は、テーブル内のすべてのセルの画像を繰り返す他、同じ画像をテーブルの **Style.BackgroundImage** に割り当てることで、その画像をテーブル全体の背景とします（相違は、それぞれの場合の画像を引き伸ばせば明らかになります）。

# アンカーとハイパーリンク

**C1Preview** はハイパーリンクをサポートします。ハイパーリンクは、描画オブジェクト（**RenderObject** と派生クラス）および段落オブジェクト（**ParagraphObject**と派生クラス）にアタッチされ、次の要素にリンクできます。

- 現在のドキュメント内のアンカー
- 別の**C1PrintDocument**内のアンカー
- 現在のドキュメント内の場所
- 外部ファイル
- 現在のドキュメント内のページ
- ユーザーイベント

ハイパーリンクは、**C1.Win.C1Preview** アセンブリに含まれるプレビューコントロール（**C1PreviewPane**、**C1PrintPreviewControl**、**C1PrintPreviewDialog**）によってサポートされます。ハイパーリンクを含むドキュメントをプレビューし、マウスポインタをハイパーリンクの上に置くと、カーソルが手の形に変わります。ハイパーリンクをクリックすると、リンクのターゲットに基づいて、ドキュメント内の別の場所にジャンプするか、別のドキュメントを開き、そのドキュメント内の場所にジャンプします。または、外部ファイルを開いたり、ユーザーイベントを起動します。

> **メモ**:次のトピック内のサンプルコードは、「**using C1.C1Preview**」ディレクティブ（C# の場合。他の言語の場合は相当する構文）がファイルに挿入されていることを前提としています。したがって、完全修飾された型名（**C1.C1Preview.RenderText** など）ではなく、クラス名部分（**RenderText**）だけを使用します。

# 同じドキュメント内のアンカーへのハイパーリンクを追加する

ドキュメントの一部分を別の部分にリンクするには、次の2つの作業を行う必要があります。

- リンク先の場所（**アンカー**）をマークします。
- その場所へのリンク**ハイパーリンク**をドキュメントの別の部分に追加します。複数のハイパーリンクが同じアンカーをポイントすることができます。

描画オブジェクトにアンカーを作成するには、その描画オブジェクトの **Anchors**コレクションに **C1Anchor**型の要素を追加します。たとえば、rt が **RenderTable**である場合は、次のように記述できます。

**Visual Basic コードの書き方**

```
Visual Basic
rt.Anchors.Add(New C1.C1Preview.C1Anchor("anchor1"))
```

**C# コードの書き方**

```
C#
rt.Anchors.Add(new C1Anchor("anchor1"));
```

これで、このレンダリングテーブルに**anchor1**という名前のアンカーが定義されます。この名前は、アンカーを参照するために


Copyright © GrapeCity, Inc. All rights reserved.

使用されます。

このアンカーに別の描画オブジェクト(たとえば、**RenderText**)をリンクするには、次のように記述します。

**Visual Basic コードの書き方**

Visual Basic

```
Dim rtxt As New C1.C1Preview.RenderText()
rtxt.Text = "Link to anchor1"
rtxt.Hyperlink = New C1.C1Preview.C1Hyperlink("anchor1")
```

**C# コードの書き方**

C#

```
RenderText rtxt = new RenderText();
rtxt.Text = "Link to anchor1";
rtxt.Hyperlink = new C1Hyperlink("anchor1");
```

もちろん、ドキュメントには、この両方の描画オブジェクト(アンカーを含むオブジェクトとハイパーリンクを含むオブジェクト)を追加する必要があります。

**Hyperlink** は、**RenderObject** クラスのプロパティで、このクラスはすべての描画オブジェクトの基本クラスです。したがって、先ほどの説明とまったく同じ方法でプロパティを設定することで、すべての描画オブジェクトをハイパーリンクに変更できます。

# 別のC1PrintDocument内のアンカーへのハイパーリンクを追加する

ドキュメント内の場所を別のドキュメント内の場所にリンクするには、以下の作業を行う必要があります。

- 先ほど説明したように、ターゲットドキュメントにアンカーを追加し、そのドキュメントを生成して、ディスクに **C1D** ファイルとして保存します。プレビューコントロールの[保存]ボタンを使用するか、コード内でドキュメント自身の **Save** メソッドを使用して、ドキュメントを保存します。
- 内部リンクを追加する場合と同様の方法で、そのアンカーをポイントするリンクを別のドキュメントに追加します。異なる点は、ターゲットアンカーの名前の他に、ドキュメントのファイル名も提供する必要があることだけです。

以下に、完全なプログラムのコードを示します。このプログラムは、アンカーを含むドキュメントを作成し、そのドキュメントをディスクファイル(**myDocument1.c1d**)に保存します。次に、別のドキュメントを作成し、そのドキュメントに最初のドキュメント内のアンカーへのリンクを追加して、プレビューダイアログで2番目のドキュメントを表示します。

**Visual Basic コードの書き方**

Visual Basic

```
'   アンカーを含むターゲットドキュメントを作成します
Dim targetDoc As New C1.C1Preview.C1PrintDocument
Dim rt1 As New C1.C1Preview.RenderText("This is anchor1 in myDocument1.")
rt1.Anchors.Add(New C1.C1Preview.C1Anchor("anchor1"))
targetDoc.Body.Children.Add(rt1)
targetDoc.Generate()
targetDoc.Save("c:\myDocument1.c1d")

' アンカーへのハイパーリンクを含むドキュメントを作成します
Dim sourceDoc As New C1.C1Preview.C1PrintDocument
Dim rt2 As New C1.C1Preview.RenderText("This is hyperlink to myDocument1.")
Dim linkTarget As C1.C1Preview.C1LinkTarget = New
```


Copyright © GrapeCity, Inc. All rights reserved.

```
C1.C1Preview.C1LinkTargetExternalAnchor("c:\myDocument1.c1d", "anchor1")
rt2.Hyperlink = New C1.C1Preview.C1Hyperlink(linkTarget)
sourceDoc.Body.Children.Add(rt2)
sourceDoc.Generate()

' ハイパーリンクを含むドキュメントをプレビューで表示します
Dim preview As New C1.Win.C1Preview.C1PrintPreviewDialog()
preview.Document = sourceDoc
preview.ShowDialog()
```

**C# コードの書き方**

C#

```
//  アンカーを含むターゲットドキュメントを作成します
C1PrintDocument targetDoc = new C1PrintDocument();
RenderText rt1 = new RenderText("This is anchor1 in myDocument1.");
rt1.Anchors.Add(new C1Anchor("anchor1"));
targetDoc.Body.Children.Add(rt1);
targetDoc.Generate();
targetDoc.Save(@"c:\myDocument1.c1d");

//アンカーへのハイパーリンクを含むドキュメントを作成します
C1PrintDocument sourceDoc = new C1PrintDocument();
RenderText rt2 = new RenderText("This is hyperlink to myDocument1.");
C1LinkTarget linkTarget = new C1LinkTargetExternalAnchor(@"c:\myDocument1.c1d",
"anchor1");
rt2.Hyperlink = new C1Hyperlink(linkTarget);
sourceDoc.Body.Children.Add(rt2);
sourceDoc.Generate();

// ハイパーリンクを含むドキュメントをプレビューで表示します
C1PrintPreviewDialog preview = new C1PrintPreviewDialog();
preview.Document = sourceDoc;
preview.ShowDialog();
```

次の点に注意してください。

- アンカーは、同じドキュメントの場合と同じ方法で作成されます。実際に違いはありません。1つのアンカーを同じドキュメントと別のドキュメントの両方からリンクターゲットにすることもできます。
- ドキュメントを保存するために、**Save** メソッドが使用されます。このメソッドは、ネイティブな**C1PrintDocument**形式でドキュメントを保存します。デフォルトの拡張子は C1D です。この形式で保存されたファイルは、後で**C1PrintDocument**オブジェクトにロードして、さらに処理を実行したり、C1 印刷プレビューコントロールを使ってプレビューすることができます。
- ハイパーリンクを作成する前に、ハイパーリンクのコンストラクタに渡すリンクターゲットオブジェクトを作成する必要があります。**C1LinkTarget** 基本クラスから派生したいくつかのタイプのリンクターゲットがあります。外部アンカーの場合は、**C1LinkTargetExternalAnchor** 型を使用します。リンクターゲットには、リンクへのジャンプを処理するために必要な情報が含まれます。この例では、ドキュメントのファイル名とアンカー名が含まれます。

## 現在のドキュメント内の場所へのハイパーリンクを追加する

アンカーを作成しなくても、現在のドキュメント内のオブジェクトへのリンクを追加できます。アンカーを作成する代わりに、次のように、描画オブジェクト上に直接作成される **C1LinkTargetDocumentLocation** リンクターゲットを使用します。


Copyright © GrapeCity, Inc. All rights reserved.

**Visual Basic コードの書き方**

```
Visual Basic
Dim linkTarget = New C1.C1Preview.C1LinkTargetDocumentLocation(ro1)
```

**C# コードの書き方**

```
C#
C1LinkTarget linkTarget = new C1LinkTargetDocumentLocation(ro1);
```

**ro1** は、現在のドキュメント内の任意の描画オブジェクトです。ハイパーリンクにこのリンクターゲットを設定した場合、そのハイパーリンクを所有するオブジェクトがクリックされると、ハイパーリンクは指定された描画オブジェクトにジャンプします。たとえば、ハイパーリンクに変換する描画オブジェクトを **ro2** とすると、次のコードは、このオブジェクトを **ro1** の場所にリンクします。**ro1** は、上のコードで示したように **linkTarget** が設定された場所です。

**Visual Basic コードの書き方**

```
Visual Basic
rt2.Hyperlink = New C1.C1Preview.C1Hyperlink()
rt2.Hyperlink.LinkTarget = linkTarget
```

**C# コードの書き方**

```
C#
rt2.Hyperlink = new C1Hyperlink();
rt2.Hyperlink.LinkTarget = linkTarget;
```

この例では、ハイパーリンクの, **LinkTarget** プロパティがハイパーリンクの作成後に設定されていることに注意してください。

## 外部ファイルへのハイパーリンクを追加する

外部ファイルへのハイパーリンクは、リンクターゲットに基づく外部アンカーへのリンクとは異なります。外部ファイル用のリンクターゲットクラスは、**C1LinkTargetFile**という名前です。このようなリンクをクリックすると、Windows シェルを使ってファイルが開かれます。たとえば、前のセクションのサンプルを使用する場合は、外部アンカーリンクターゲットを作成する行を次の行に置き換えます。

**Visual Basic コードの書き方**

```
Visual Basic
Dim linkTarget = New C1.C1Preview.C1LinkTargetFile("c:\")
```

**C# コードの書き方**

```
C#
C1LinkTarget linkTarget = new C1LinkTargetFile(@"c:\");
```

このリンクをクリックすると、Windows エクスプローラでドライブ C のルートディレクトリが開かれます。

完全なプログラムを次に示します。

**Visual Basic コードの書き方**


Copyright © GrapeCity, Inc. All rights reserved.

Visual Basic

```
'  外部ファイルへのハイパーリンクを含むドキュメントを作成します
Dim doc As New C1.C1Preview.C1PrintDocument
Dim rt As New C1.C1Preview.RenderText("Explore drive C:...")
Dim linkTarget As C1.C1Preview.C1LinkTarget = New
C1.C1Preview.C1LinkTargetFile("c:\")
rt.Hyperlink = New C1.C1Preview.C1Hyperlink(linkTarget)
doc.Body.Children.Add(rt)
doc.Generate()

' ハイパーリンクを含むドキュメントをプレビューで表示します
Dim preview As New C1.Win.C1Preview.C1PrintPreviewDialog()
preview.Document = doc
preview.ShowDialog()
```

**C# コードの書き方**

C#

```
// 外部ファイルへのハイパーリンクを含むドキュメントを作成します
C1PrintDocument doc = new C1PrintDocument();
RenderText rt = new RenderText("Explore drive C:...");
C1LinkTarget linkTarget = new C1LinkTargetFile(@"c:\");
rt.Hyperlink = new C1Hyperlink(linkTarget);
doc.Body.Children.Add(rt);
doc.Generate();

// ハイパーリンクを含むドキュメントをプレビューで表示します
C1PrintPreviewDialog preview = new C1PrintPreviewDialog();
preview.Document = doc;
preview.ShowDialog();
```

## 同じドキュメント内のページへのハイパーリンクを追加する

アンカーを定義しなくても、**C1LinkTargetPage** リンクターゲットを使用して、同じドキュメント内の別のページへのハイパーリンクを追加できます。以下のページへのジャンプがサポートされます。

- ドキュメントの最初のページへのジャンプ。
- ドキュメントの最後のページへのジャンプ。
- 前のページへのジャンプ。
- 次のページへのジャンプ。
- 絶対ページ番号で指定されたページへのジャンプ。
- 現在のページに対するオフセットで指定されたページへのジャンプ。

たとえば、リンクターゲットからドキュメント内の最初のページにジャンプする場合は、次のコードを使用します。

**Visual Basic コードの書き方**

Visual Basic

```
Dim linkTarget = New
C1.C1Preview.C1LinkTargetPage(C1.C1Preview.PageJumpTypeEnum.First)
```

**C# コードの書き方**


Copyright © GrapeCity, Inc. All rights reserved.

C#

```
C1LinkTarget linkTarget = new C1LinkTargetPage(PageJumpTypeEnum.First);
```

ここで、**PageJumpTypeEnum** は、ページジャンプのタイプを指定します。もちろん、絶対または相対ページ番号を必要とするジャンプの場合は、その数値を受け取るコンストラクタを使用する必要があります。

この機能を使用すると、ドキュメント自身でドキュメント内を移動する簡単な方法を提供できます。たとえば、ビデオデッキのような操作ボタン（最初のページに移動、前のページに移動、次のページに移動、最後のページに移動）をドキュメントのフッターに追加できます。

## ユーザーイベントへのハイパーリンクを追加する

最後に、**C1PreviewPane**のイベントを発生させてコードによって処理するためのハイパーリンクを追加します。それには、**C1LinkTargetUser** を使用する必要があります。この概念を示す完全な例を次に示します。

**Visual Basic コードの書き方**

Visual Basic

```
Private Sub UserLinkSetup()

    ' ユーザーハイパーリンクを含むドキュメントを作成します
    Dim doc As New C1.C1Preview.C1PrintDocument
    Dim rt As New C1.C1Preview.RenderText("Click this to show message box...")
    Dim linkTarget As C1.C1Preview.C1LinkTarget = New C1.C1Preview.C1LinkTargetUser
    rt.Hyperlink = New C1.C1Preview.C1Hyperlink(linkTarget)
    rt.Hyperlink.UserData = "My hyperlnk user data"
    doc.Body.Children.Add(rt)
    doc.Generate()

    ' プレビューを作成します
    Dim preview As New C1.Win.C1Preview.C1PrintPreviewDialog()

    ' イベントハンドラを UserHyperlinkJump イベントにアタッチします
    AddHandler preview.PreviewPane.UserHyperlinkJump, New
C1.Win.C1Preview.HyperlinkEventHandler(AddressOf Me.C1PreviewPane1_UserHyperlinkJump)

    ' ドキュメントをプレビューします
    preview.Document = doc
    preview.ShowDialog()End Sub

Private Sub C1PreviewPane1_UserHyperlinkJump(ByVal sender As Object, ByVal e As
C1.Win.C1Preview.HyperlinkEventArgs) Handles C1PreviewPane1.UserHyperlinkJump
    MessageBox.Show(e.Hyperlink.UserData.ToString())
End Sub
```

**C# コードの書き方**

C#

```
private void UserLinkSetup()
{
```


Copyright © GrapeCity, Inc. All rights reserved.

```
    // ユーザーハイパーリンクを含むドキュメントを作成します
    C1PrintDocument doc = new C1PrintDocument();
    RenderText rt = new RenderText("Click this to show message box...");
    C1LinkTarget linkTarget = new C1LinkTargetUser();
    rt.Hyperlink = new C1Hyperlink(linkTarget);
    rt.Hyperlink.UserData = "My hyperlnk user data";
    doc.Body.Children.Add(rt);
    doc.Generate();

    // プレビューを作成します
    C1PrintPreviewDialog preview = new C1PrintPreviewDialog();

    //イベントハンドラを UserHyperlinkJump イベントにアタッチします
    preview.PreviewPane.UserHyperlinkJump += new
HyperlinkEventHandler(PreviewPane_UserHyperlinkJump);

    // ドキュメントをプレビューします
    preview.Document = doc;
    preview.ShowDialog();
}

private void PreviewPane_UserHyperlinkJump(object sender, HyperlinkEventArgs e)
{
    MessageBox.Show(e.Hyperlink.UserData.ToString());
}
```

この例では、ハイパーリンクがクリックされると、ハイパーリンクの**UserData** プロパティに割り当てられた文字列（この例では「ハイパーリンクのユーザーデータ」）を含むメッセージボックスが表示されます。

## リンクターゲットクラスの階層

ハイパーリンクに関するセクションの最後に、リンクターゲットクラスの階層を次に示します。

| クラス名 | 詳細 |
|---|---|
| **C1LinkTarget** | 階層全体の基本クラス。 |
| **C1LinkTargetAnchor** | 現在のドキュメント内のアンカーであるターゲットを記述します。 |
| **C1LinkTargetExternalAnchor** | 別のドキュメント内のアンカーであるターゲットを記述します。 |
| **C1LinkTargetDocumentLocation** | 描画オブジェクトであるターゲットを記述します。 |
| **C1LinkTargetFile** | OS シェルによって開かれる外部ファイルであるターゲットを記述します。 |
| **C1LinkTargetPage** | 現在のドキュメント内のページであるターゲットを記述します。 |
| **C1LinkTargetUser** | C1PreviewPane のユーザーイベントハンドラを起動するターゲットを記述します。 |

## 式、スクリプト、タグ

式（またはスクリプト – ここでは2つの用語が同じ意味で使用されます）は、ドキュメントを通じて随所に使用されます。式は、次のコードに示すように角括弧で囲んで表されます。

**Visual Basic コードの書き方**


Copyright © GrapeCity, Inc. All rights reserved.

Visual Basic

```
Dim doc As New C1PrintDocument()
doc.Body.Children.Add(New RenderText("2 + 2 = [2+2]"))
```

**C# コードの書き方**

C#

```
C1PrintDocument doc = new C1PrintDocument();
doc.Body.Children.Add(new RenderText("2 + 2 = [2+2]"));
```

このコードでは、生成されるドキュメントに次のテキストが表示されます。

```
2 + 2 = 4
```

この場合、"[2+2]" は式として解釈され、計算が行われて結果として表示されるテキストを生成します。

**注意**:式パーサーが角括弧内のテキストを解析できない場合、このテキストは式として解釈されず、「そのまま」ドキュメント内に挿入されます。

## タグ

タグは式に密接に関連しています。実際、タグは式内で使用するか、または単に式として使用できる変数です。最も単純な場合、タグにより、特定の文字列を挿入したいが、その文字列の値がわからない場所にプレースホルダを使用できます。タグの一般的な例は、ドキュメントを再生成するときに、ページ番号が何ページになるかわからない場合、または変わる可能性がある場合にページ番号を出力する場合です。

タグには、**Name** と **Value** の2つの主なプロパティがあります。名前はタグの識別に使用され、値はタグが置き換えられる内容です。

**C1PrintDocument** には、定義済みタグとカスタムタグの2種類のタグが用意されています。定義済みタグには、以下のものがあります。

- [PageNo] – 現在のページ番号に置き換えられます。
- [PageCount] – 合計ページ数に置き換えられます。
- [PageX] – 現在の横方向のページ番号に置き換えられます。
- [PageXCount] – 横方向のページの合計ページ数に置き換えられます。
- [PageY] – 現在の縦方向のページ番号に置き換えられます(横方向のページが存在しない場合、[PageNo] と同じです)。
- [PageYCount] – 現在の縦方向のページの合計ページ数に置き換えられます(横方向のページが存在しない場合、[PageCount] と同じです)。

カスタムタグは、ドキュメントの **Tags** コレクションに格納されています。このコレクションにタグを追加するには、次のコードを使用できます。

**Visual Basic コードの書き方**

Visual Basic

```
doc.Tags.Add(New C1.C1Preview.Tag("tag1", "tag value"))
```

**C# コードの書き方**

C#

```
doc.Tags.Add(new C1.C1Preview.Tag("tag1", "tag value"));
```


Copyright © GrapeCity, Inc. All rights reserved.

タグの作成時にタグの値を未指定のままにし、後のある時点で（タグがドキュメント内で使用された後でも）指定できます。

タグを使用するには、次の例に示すように、タグの値を表示するテキスト内に、その名前を角括弧で囲んで挿入します。

**Visual Basic コードの書き方**

Visual Basic

```
Dim rt As New C1.C1Preview.RenderText()
rt.Text = "The value of tag1 will appear here: [tag1]."
```

**C# コードの書き方**

C#

```
RenderText rt = new RenderText();
rt.Text = "The value of tag1 will appear here: [tag1].";
```

# タグ/式の構文

必要に応じて、ドキュメントの **TagOpenParen** および **TagCloseParen** プロパティを使用して、タグまたはスクリプトをテキストに取り込むために使用する角括弧を任意の文字列に変更できます。これは、ドキュメントの多くの角括弧を含める場合に役立ちます。デフォルトでは、これらは、結果の表示に影響を与えない場合でも、多くのリソースを消費する可能性のあるすべての式の解析を起動します。次に例を示します。

**Visual Basic コードの書き方**

Visual Basic

```
doc.TagOpenParen = "@@@["
doc.TagCloseParen = "@@@]"
```

**C# コードの書き方**

C#

```
doc.TagOpenParen = "@@@[";
doc.TagCloseParen = "@@@]";
```

これらの例では、「@@@[」と「@@@]」で囲まれた文字列だけが式として解釈されます。

式の括弧は、次の例に示すように、前にバックスラッシュ記号を置くことによってエスケープすることもできます。

**Visual Basic コードの書き方**

Visual Basic

```
Dim doc As New C1PrintDocument()
doc.Body.Children.Add(new RenderText("2 + 2 = \[2+2\]"))
```

**C# コードの書き方**

C#

```
C1PrintDocument doc = new C1PrintDocument();
doc.Body.Children.Add(new RenderText("2 + 2 = \[2+2\]"));
```

このコードでは、生成されるドキュメントに次のテキストが表示されます。

```
2 + 2 = [2+2]
```


Copyright © GrapeCity, Inc. All rights reserved.

これは、括弧がエスケープされるためです。

ドキュメントで **TagEscapeString** プロパティを使用して、エスケープ文字列を任意の文字列に変更できます。

# 実行時のタグ値の編集

エンドユーザーが**C1PrintDocument**コンポーネントの **Tags** コレクションに含まれているタグのタグ値を表示および編集するフォームを表示することができます。複数の新しいメンバが、特定のタグが表示されるカスタマイズされたフォームを作成できる、このオプションをサポートします。

作成する **Tags** フォームを使用するいくつかのオプションがあります。以下を実行できます。

- **C1PrintDocument** を生成するたびに、ユーザーがすべてのタグを編集できるようにします。詳細については、すべてのタグの表示を参照してください。
- ユーザーがすべてのタグではなく、一部のタグを編集できるようにします。ユーザーに編集されたくないタグについては、タグの **Flags** プロパティを **None** に設定します。
- 最後に、エンドユーザーが[タグ]ダイアログボックスを表示する場合を選択できます。ドキュメントの**ShowTagsInputDialog** を **False** に設定し、ユーザーがタグ入力ダイアログを表示する必要がある場合に独自のコードで **EditTags**() メソッドを呼び出します。

# すべてのタグの表示

デフォルトでは、**ShowTagsInputDialog** プロパティは **False** に設定され、[***タグ***]ダイアログボックスは表示されません。**C1PrintDocument** を生成するたびに、ユーザーがすべてのタグを入力できるようにするには、ドキュメントの **ShowTagsInputDialog** プロパティを **True** に設定します。ドキュメントの **Tags** コレクションに追加したタグは、ドキュメントを生成しようとするたびに、ユーザーのダイアログボックスに自動的に表示されます。この結果、エンドユーザーが[***タグ***]ダイアログボックスでタグの値を編集できるようになります。

たとえば、次のコードを **Form_Load** イベントに追加して、これらのタグのドキュメントとテキストの値に3つのタグを追加します。

**Visual Basic コードの書き方**

Visual Basic

```
Dim doc As New C1PrintDocument()
Me.C1PrintPreviewControl1.Document = doc
' ドキュメントの生成時に[タグ]ダイアログボックスを表示します。
doc.ShowTagsInputDialog = True
' [タグ]ダイアログボックスに表示されるタグを作成します。
doc.Tags.Add(New C1.C1Preview.Tag("Statement", "Hello World!"))
doc.Tags.Add(New C1.C1Preview.Tag("Name", "ComponentOne"))
doc.Tags.Add(New C1.C1Preview.Tag("Location", "Pittsburgh, PA"))
' タグをドキュメントに追加し、生成します。
Dim rt As New C1.C1Preview.RenderText()
rt.Text = "[Statement] My name is [Name] and my current location is [Location]."
doc.Body.Children.Add(rt)
doc.Generate()
```

**C#コードの書き方**

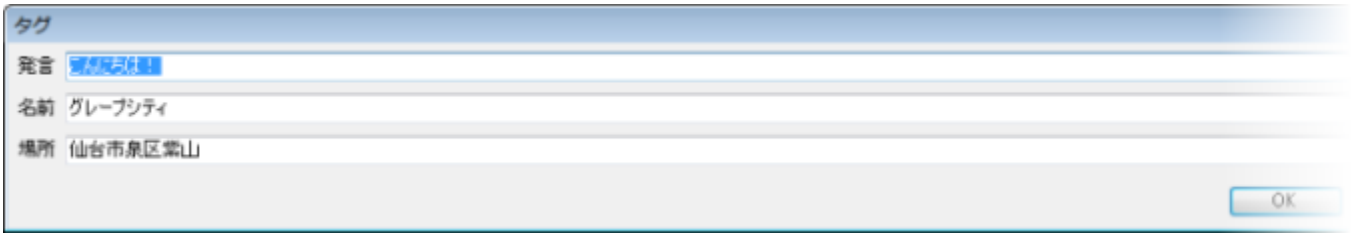
C#

```
C1PrintDocument doc = new C1PrintDocument();
this.c1PrintPreviewControl1.Document = doc;
// ドキュメントの生成時に[タグ]ダイアログボックスを表示します。
```


Copyright © GrapeCity, Inc. All rights reserved.

```
doc.ShowTagsInputDialog = true;
// [タグ]ダイアログボックスに表示されるタグを作成します。
doc.Tags.Add(new C1.C1Preview.Tag("Statement", "Hello World!"));
doc.Tags.Add(new C1.C1Preview.Tag("Name", "ComponentOne"));
doc.Tags.Add(new C1.C1Preview.Tag("Location", "Pittsburgh, PA"));
// タグをドキュメントに追加し、生成します。
C1.C1Preview.RenderText rt = new C1.C1Preview.RenderText();
rt.Text = "[Statement] My name is [Name] and my current location is [Location].";
doc.Body.Children.Add(rt);
doc.Generate();
```

アプリケーションを実行すると、ドキュメントを生成する前に、以下のダイアログボックスが表示されます。




[**タグ**]ダイアログボックスのテキストボックスのいずれかでテキストを変更すると、生成されたドキュメントに表示されるテキストが変更されます。デフォルトのテキストがそのまま残された場合、生成されたドキュメントに以下のテキストが表示されます。

こんにちは！　私の名前はグレープシティです。私の現在地は　仙台市泉区紫山　です。

# 特定のタグの表示

**ShowTagsInputDialog** プロパティを **True** に設定すると、デフォルトで、すべてのタグが[**タグ**]ダイアログボックスに表示されます。**Tag.ShowInDialog** プロパティを使用して、ユーザーが特定のタグを編集できないようにすることができます。ユーザーがすべてのタグではなく、一部のタグを編集できるようにするには、編集する必要がないタグの **Flags** プロパティを **None** に設定します。

たとえば、次のコードを **Form_Load** イベントに追加して、編集できないドキュメントのいずれかに3つのタグを追加します。

**Visual Basic コードの書き方**

Visual Basic

```
Dim doc As New C1PrintDocument()
Me.C1PrintPreviewControl1.Document = doc
' ドキュメントの生成時に[タグ]ダイアログボックスを表示します。
doc.ShowTagsInputDialog = True
' タグを作成し、そのタグが[タグ]ダイアログボックスに表示されないようにします。
doc.Tags.Add(New C1.C1Preview.Tag("発言", "こんにちは！"))
doc.Tags("発言").ShowInDialog = False
' 表示されるタグを作成します。
doc.Tags.Add(New C1.C1Preview.Tag("名前", "グレープシティ"))
doc.Tags.Add(New C1.C1Preview.Tag("場所", "仙台市泉区紫山"))
' タグをドキュメントに追加し、生成します。
Dim rt As New C1.C1Preview.RenderText()
rt.Text = "[発言] 私の名前は [名前] です。私の現在地は [場所] です。"
doc.Body.Children.Add(rt)
doc.Generate()
```

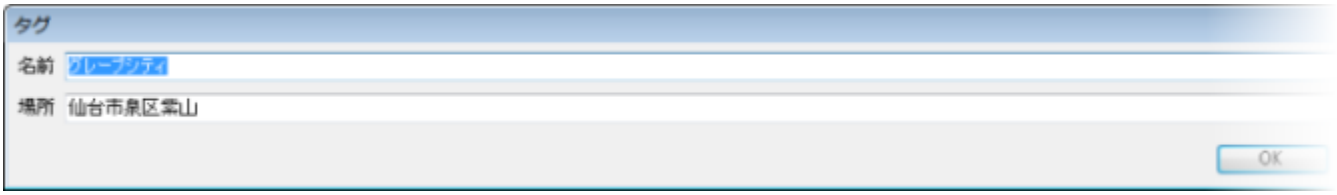
**C# コードの書き方**


Copyright © GrapeCity, Inc. All rights reserved.

C#

```
C1PrintDocument doc = new C1PrintDocument();
this.c1PrintPreviewControl1.Document = doc;
//ドキュメントの生成時に[タグ]ダイアログボックスを表示します。
doc.ShowTagsInputDialog = true;
// タグを作成し、そのタグが[タグ]ダイアログボックスに表示されないようにします。
doc.Tags.Add(new C1.C1Preview.Tag("発言", "こんにちは！"));
doc.Tags["発言"].ShowInDialog = false;
//表示されるタグを作成します。
doc.Tags.Add(new C1.C1Preview.Tag("名前", "グレープシティ"));
doc.Tags.Add(new C1.C1Preview.Tag("場所", "仙台市泉区紫山"));
// タグをドキュメントに追加し、生成します。
C1.C1Preview.RenderText rt = new C1.C1Preview.RenderText();
rt.Text = "[発言] 私の名前は [名前] です。私の現在地は [場所] です。";
doc.Body.Children.Add(rt);
doc.Generate();
```

アプリケーションを実行すると、ドキュメントを生成する前に、以下のダイアログボックスが表示されます。




[**タグ**]ダイアログボックスのテキストボックスのいずれかでテキストを変更すると、生成されたドキュメントに表示されるテキストが変更されます。**Statement** タグは表示されず、ダイアログボックスで変更できないことに注意してください。デフォルトのテキストがそのまま残された場合、生成されたドキュメントに以下のテキストが表示されます。

こんにちは！ 私の名前は グレープシティ です。私の現在地は 仙台市泉区紫山 です。

## [タグ]ダイアログボックスを表示する場合の指定

**ShowTagsInputDialog**プロパティを **True** に設定すると、ドキュメントを生成する直前に[**タグ**]ダイアログボックスが表示されます。**EditTags** メソッドを呼び出すことで、必要なときは(**ShowTagsInputDialog** プロパティの値に関係なく)いつでも、プログラム操作によって、このダイアログボックスを表示できます。

たとえば、次のコードは、ボタンをクリックしたときに、タグ入力ダイアログボックスを表示します。

**Visual Basic コードの書き方**

Visual Basic

```
Public Class Form1
    Dim doc As New C1PrintDocument()
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        Me.C1PrintPreviewControl1.Document = doc
        '  表示されるタグを作成します。
        doc.Tags.Add(New C1.C1Preview.Tag("Statement", "Hello World!"))
        doc.Tags("Statement").ShowInDialog = True
        doc.Tags.Add(New C1.C1Preview.Tag("Name", "ComponentOne"))
        doc.Tags.Add(New C1.C1Preview.Tag("Location", "Pittsburgh, PA"))
        ' タグをドキュメントに追加します。
        Dim rt As New C1.C1Preview.RenderText()
```


Copyright © GrapeCity, Inc. All rights reserved.

```
        rt.Text = "[Statement] My name is [Name] and my current location is
[Location]."
        doc.Body.Children.Add(rt)
    End Sub
    Private Sub EditTagsNow_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles EditTagsNow.Click
        ' ボタンのクリックにより[タグ]ダイアログボックスを表示します。
        doc.ShowTagsInputDialog = True
        doc.EditTags()
    End Sub
    Private Sub GenerateDocNow_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles GenerateDocNow.Click
        doc.ShowTagsInputDialog = False
        ' ボタンのクリックによりドキュメントを生成します。
        doc.Generate()
    End Sub
End Class
```

**C# コードの書き方**

C#

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }
    C1PrintDocument doc = new C1PrintDocument();
    private void Form1_Load(object sender, EventArgs e)
    {
        this.c1PrintPreviewControl1.Document = doc;
        //  表示されるタグを作成します。
        doc.Tags.Add(new C1.C1Preview.Tag("Statement", "Hello World!"));
        doc.Tags["Statement"].ShowInDialog = true;
        doc.Tags.Add(new C1.C1Preview.Tag("Name", "ComponentOne"));
        doc.Tags.Add(new C1.C1Preview.Tag("Location", "Pittsburgh, PA"));
        // タグをドキュメントに追加します。
        C1.C1Preview.RenderText rt = new C1.C1Preview.RenderText();
        rt.Text = "[Statement] My name is [Name] and my current location is
[Location].";
        doc.Body.Children.Add(rt);
    }
    private void EditTagsNow_Click(object sender, EventArgs e)
    {
        // ボタンのクリックにより[タグ]ダイアログボックスを表示します。
        doc.ShowTagsInputDialog = true;
        doc.EditTags();
    }
    private void GenerateDoc_Click(object sender, EventArgs e)
    {
        doc.ShowTagsInputDialog = false;
        // ボタンのクリックによりドキュメントを生成します。
```


Copyright © GrapeCity, Inc. All rights reserved.

```
        doc.Generate();
    }
}
```

上記の例では、〈**EditTagsNow**〉ボタンをクリックすると、[**タグ**]ダイアログボックスが表示されます。

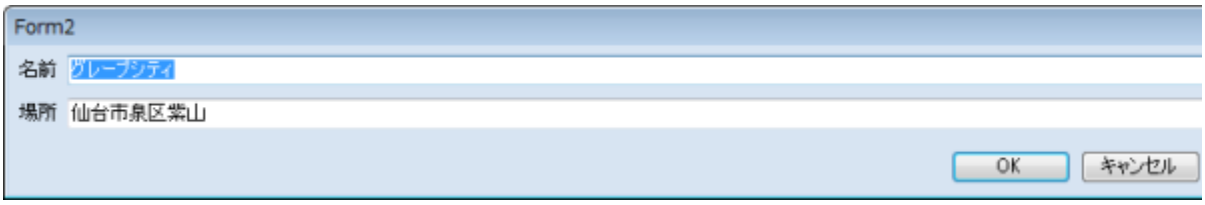
# デフォルトの[タグ]ダイアログボックスの定義

**TagsInputForm** または **TagsInputFormBase** のいずれかに基づくプロジェクトに継承されたフォームを追加することで、[タグ]ダイアログボックスを簡単にカスタマイズできます。フォームに小規模な変更を加えるか、フォームを完全に再設計するかによって、対応する方法が異なります。

## 小規模な変更を加える場合

デフォルトのフォームに小規模な変更を加えるだけの場合(〈ヘルプ〉ボタンを追加する場合など)、**TagsInputForm** に基づくプロジェクトに継承されたフォームを追加し、必要に応じて調整して、そのフォームの型名をドキュメントの **TagsInputDialogClassName** プロパティに割り当てることができます。

たとえば、次のフォームでは、キャプションバーに〈ヘルプ〉ボタンが追加され、フォームの背景色が変更されました。




## フォームを完全に変更する場合

希望する場合、デフォルトのフォームを完全に変更することができます。たとえば、独自のコントロールを用意したり、タグの値を入力したりできます。これを行うには、TagsInputFormBase の継承されたフォームに基づき、必要に応じて変更し、**EditTags** メソッドをオーバーライドします。

# スクリプト/式の言語

式で使用する言語は、**C1PrintDocument.ScriptingOptions.Language** プロパティの値によって決まります。このプロパティは、以下の2つの値のいずれかを持つことができます。

- **VB**。これはデフォルト値で、標準 **VB.NET** がスクリプト言語として使用されることを示します。
- **C1Report**。この値は、**C1Report** スクリプト言語が使用されることを示します。この言語は VB に似ていますが、わずかな違いがあります。このオプションは、主として後方互換性を提供します。
- **CSharp**。この値は、標準 C# がスクリプト言語として使用されることを示します。

**VB** が式言語として使用される場合、ドキュメントを生成すると、**ScriptExpressionBase** から派生した単一クラスを含む各式に対して、別個のアセンブリが内部的にビルドされます。このクラスには、式に使用できる保護されたプロパティが含まれています。式自体は、このクラスの関数として実装されます。

たとえば、

**Visual Basic コードの書き方**

Visual Basic

```
Dim doc As New C1PrintDocument()
doc.ScriptingOptions.Language = ScriptLanguageEnum.CSharp
```


Copyright © GrapeCity, Inc. All rights reserved.

```
Dim rt As New RenderText("[PageNo == 1 ? ""First"" : ""Not first""]")
doc.Body.Children.Add(rt)
```

**C# コードの書き方**

C#

```
C1PrintDocument doc = new C1PrintDocument();
doc.ScriptingOptions.Language = ScriptLanguageEnum.CSharp;
RenderText rt = new RenderText("[PageNo == 1 ? \"First\" : \"Not first\"]");
doc.Body.Children.Add(rt);
```

# アセンブリと名前空間

デフォルトでは、以下のアセンブリがスクリプトに使用（参照）可能です。

- System
- System.Drawing

スクリプトで参照されるアセンブリのリストに別の（システムまたはカスタム）アセンブリを追加するには、それをドキュメントの **C1PrintDocument.ScriptingOptions.ExternalAssemblies** コレクションに追加します。たとえば、次のコードは、**System.Data** アセンブリへの参照をドキュメントのスクリプトに追加します。

**Visual Basic コードの書き方**

Visual Basic

```
Dim doc As New C1PrintDocument()
doc.ScriptingOptions.ExternalAssemblies.Add("System.Data.dll")
```

**C# コードの書き方**

C#

```
C1PrintDocument doc = new C1PrintDocument();
doc.ScriptingOptions.ExternalAssemblies.Add("System.Data.dll");
```

以下の名前空間は、デフォルトでスクリプトに使用（インポート）可能です。

- System
- System.Collections
- System.Collections.Generic
- System.Text
- Microsoft.VisualBasic
- System.Drawing

名前空間を追加するには、それをドキュメントの **C1PrintDocument.ScriptingOptions.Namespaces** コレクションに追加します。たとえば、ドキュメントのスクリプト内の **System.Data** 名前空間で宣言された型を、名前空間によって完全修飾することなく使用することができます。

**Visual Basic コードの書き方**

Visual Basic

```
Dim doc As New C1PrintDocument()
doc.ScriptingOptions. Namespaces.Add("System.Data")
```


Copyright © GrapeCity, Inc. All rights reserved.

**C# コードの書き方**

```
C#
C1PrintDocument doc = new C1PrintDocument();
doc.ScriptingOptions. Namespaces.Add("System.Data");
```

# テキスト式でアクセス可能な ID

前述のとおり、括弧内の式は、**RenderText** および**ParagraphText** オブジェクトの **Text** プロパティ内で使用できます。これらの式では、以下のオブジェクト **ID** を使用できます。

- Document (**C1PrintDocument**型)

  この変数は、生成されるドキュメントを参照します。これは、多くの方法で使用できます。たとえば、次のコードは、現在のドキュメントの作成者を出力します。

  **Visual Basic コードの書き方**

```
Visual Basic
Dim doc As New C1PrintDocument()
Dim rt As New RenderText("Landscape is " + _
  "[Iif(Page.PageSettings.Landscape,\"TRUE\",\"FALSE\")].")
doc.Body.Children.Add(rt)
```

  **C#コードの書き方**

```
C#
C1PrintDocument doc = new C1PrintDocument();
RenderText rt = new RenderText("Landscape is " +
  "[Iif(Page.PageSettings.Landscape,\"TRUE\",\"FALSE\")].");
doc.Body.Children.Add(rt);
```

- RenderObject (**RenderObject**型)

  この変数は、現在の表示オブジェクトを参照します。たとえば、次のコードは、現在の表示オブジェクトの名前を出力します。

  **Visual Basic コードの書き方**

```
Visual Basic
Dim doc As New C1PrintDocument()
Dim rt As New RenderText( _
  "The object's name is [RenderObject.Name]")
rt.Name = "MyRenderText"
doc.Body.Children.Add(rt)
```

  **C# コードの書き方**

```
C#
C1PrintDocument doc = new C1PrintDocument();
RenderText rt = new RenderText(
  "The object's name is [RenderObject.Name]");
rt.Name = "MyRenderText";
```


Copyright © GrapeCity, Inc. All rights reserved.

```
doc.Body.Children.Add(rt);
```

- Page (**C1Page**型)

  この変数は、現在のページ（**C1Page** 型のオブジェクト）を参照します。ページオブジェクトのスクリプトメンバで最もよく使用される変数には直接アクセスできますが（後述の **PageNo**、**PageCount** などを参照してください）、現在のページ設定など、Page 変数を介してアクセスできる他のデータがあります。たとえば、次のコードは、現在のページレイアウトが横長の向きの場合「**Landscape is TRUE**」を出力し、それ以外の場合「**Landscape is FALSE**」を返します。

  **Visual Basic コードの書き方**

  Visual Basic

```
Dim doc As New C1PrintDocument()
Dim rt As New RenderText("Landscape is " + _
  "[Iif(Page.PageSettings.Landscape,\"TRUE\",\"FALSE\")].")
doc.Body.Children.Add(rt)
```

  **C# コードの書き方**

  C#

```
C1PrintDocument doc = new C1PrintDocument();
RenderText rt = new RenderText("Landscape is " +
  "[Iif(Page.PageSettings.Landscape,\"TRUE\",\"FALSE\")].");
doc.Body.Children.Add(rt);
```

- PageNo (Integer型)

  この名前は、現在のページ番号を1ページ単位で解決します。**Page.PageNo** と同等です。

- PageCount (Integer型)

  この名前は、ドキュメントの合計ページ数を解決します。**Page.PageCount** と同等です。たとえば、次のコードは、一般的な「**Page X of Y**」ページヘッダーの生成に使用できます。

  **Visual Basic コードの書き方**

  Visual Basic

```
Dim doc As New C1PrintDocument()
doc.PageLayout.PageHeader = New RenderText( _
  "Page [PageNo] of [PageCount]")
```

  **C# コードの書き方**

  C#

```
C1PrintDocument doc = new C1PrintDocument();
doc.PageLayout.PageHeader = new RenderText(
  "Page [PageNo] of [PageCount]");
```

- PageX (Integer型)

  この名前は、現在の横方向のページ番号を1ページ単位で解決します（横方向の改ページがないドキュメントの場合、1 を返します）。

- PageY (Integer型)

  この名前は、現在の縦方向のページ番号を1ページ単位で解決します（縦方向の改ページがないドキュメントの場合、**PageNo** と同等です）。


Copyright © GrapeCity, Inc. All rights reserved.

- PageXCount (Integer型)

  この名前は、ドキュメントの合計ページ数を解決します（横方向の改ページがないドキュメントの場合、1 を返します）。

- PageYCount (Integer型)

  この名前は、ドキュメントの合計ページ数を解決します（横方向の改ページがないドキュメントの場合、**PageCount** と同等です）。ここで説明するページ番号に関連するどの変数も、必ずしもページヘッダーやページフッターに限らず、ドキュメント内の任意の場所で使用できることに注意することが重要です。

- Fields (**FieldCollection**型)

  この変数は、使用可能なデータベースフィールドのコレクションを参照し、**C1.C1Preview.DataBinding.FieldCollection** 型を持ちます。データがバインドされたドキュメントでのみ使用できます。たとえば、次のコードは、**NWIND** データベースの **Products** テーブルに含まれる商品名のリストを出力します。

  **Visual Basic コードの書き方**

  Visual Basic

```
Dim doc As New C1PrintDocument()
Dim dSrc As New DataSource()
dSrc.ConnectionProperties.DataProvider = DataProviderEnum.OLEDB
dSrc.ConnectionProperties.ConnectString = _
  "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=NWIND.MDB"
Dim dSet1 As New C1.C1Preview.DataBinding.DataSet( _
  dSrc, "select * from Products")
doc.DataSchema.DataSources.Add(dSrc)
doc.DataSchema.DataSets.Add(dSet1)
Dim rt As New RenderText()
rt.DataBinding.DataSource = dSet1
rt.Text = "[Fields!ProductName.Value]"
doc.Body.Children.Add(rt)
```

  **C#コードの書き方**

  C#

```
C1PrintDocument doc = new C1PrintDocument();
DataSource dSrc = new DataSource();
dSrc.ConnectionProperties.DataProvider = DataProviderEnum.OLEDB;
dSrc.ConnectionProperties.ConnectString =
  @"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=NWIND.MDB";
C1.C1Preview.DataBinding.DataSet dSet1 =
  new C1.C1Preview.DataBinding.DataSet(dSrc,
  "select * from Products");
doc.DataSchema.DataSources.Add(dSrc);
doc.DataSchema.DataSets.Add(dSet1);
RenderText rt = new RenderText();
doc.Body.Children.Add(rt);
rt.DataBinding.DataSource = dSet1;
rt.Text = "[Fields!ProductName.Value]";
```

  最終行でフィールド配列の要素にアクセスするための「!」の使用に注意してください。代わりに、次のように記述できます。

  **Visual Basic コードの書き方**


Copyright © GrapeCity, Inc. All rights reserved.

Visual Basic

```
rt.Text = "[Fields(\"ProductName\").Value]"
```

**C# コードの書き方**

C#

```
rt.Text = "[Fields(\"ProductName\").Value]";
```

ただし、「!」を使用する表記の方が短くて、読みやすくなります。

- Aggregates (AggregateCollection型)

この変数を使用して、ドキュメントで定義された集計のコレクションにアクセスできます。このコレクションは、**C1.C1Preview.DataBinding.AggregateCollection** 型を持ち、その要素は **C1Preview.DataBinding.Aggregate** 型を持ちます。たとえば、次のコードは、商品のリストの後に平均単価を出力します。

**Visual Basic コードの書き方**

Visual Basic

```
Dim doc As New C1PrintDocument()
Dim dSrc As New DataSource()
dSrc.ConnectionProperties.DataProvider = DataProviderEnum.OLEDB
dSrc.ConnectionProperties.ConnectString = _
  "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=NWIND.MDB"
C1.C1Preview.DataBinding.DataSet dSet1 = _
  new C1.C1Preview.DataBinding.DataSet(dSrc, _
  "select * from Products")
doc.DataSchema.DataSources.Add(dSrc)
doc.DataSchema.DataSets.Add(dSet1)
Dim rt As New RenderText()
doc.Body.Children.Add(rt)
rt.DataBinding.DataSource = dSet1
rt.Text = "[Fields!ProductName.Value]"
doc.DataSchema.Aggregates.Add(new Aggregate( _
  "AveragePrice", "Fields!UnitPrice.Value", _
  rt.DataBinding, RunningEnum.Document, _
  AggregateFuncEnum.Average))
doc.Body.Children.Add(new RenderText( _
  "Average price: [Aggregates!AveragePrice.Value]"))
```

**C# コードの書き方**

C#

```
C1PrintDocument doc = new C1PrintDocument();
DataSource dSrc = new DataSource();
dSrc.ConnectionProperties.DataProvider = DataProviderEnum.OLEDB;
dSrc.ConnectionProperties.ConnectString =
  @"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=NWIND.MDB";
C1.C1Preview.DataBinding.DataSet dSet1 =
  new C1.C1Preview.DataBinding.DataSet(dSrc,  "select * from Products");
doc.DataSchema.DataSources.Add(dSrc);
```


Copyright © GrapeCity, Inc. All rights reserved.

```
doc.DataSchema.DataSets.Add(dSet1);
RenderText rt = new RenderText();
doc.Body.Children.Add(rt);
rt.DataBinding.DataSource = dSet1;
rt.Text = "[Fields!ProductName.Value]";
doc.DataSchema.Aggregates.Add(new Aggregate(
  "AveragePrice", "Fields!UnitPrice.Value",
  rt.DataBinding, RunningEnum.Document,
  AggregateFuncEnum.Average));
doc.Body.Children.Add(new RenderText(
  "Average price: [Aggregates!AveragePrice.Value]"));
```

- DataBinding (**C1DataBinding型**)

  この変数を使用して、**C1.C1Preview.DataBinding.C1DataBinding** 型の現在の表示オブジェクトの **DataBinding** プロパティにアクセスできます。たとえば、次のコード（**Fields** 変数の使用を示すサンプルから変更されました）は、テキスト式内で表示オブジェクトの **DataBinding** プロパティの **RowNumber** メンバを使用して、商品の番号付きリストを生成します。

  **Visual Basic コードの書き方**

  Visual Basic

```
Dim doc As New C1PrintDocument()
Dim dSrc As New DataSource()
dSrc.ConnectionProperties.DataProvider = DataProviderEnum.OLEDB
dSrc.ConnectionProperties.ConnectString = _
  "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=NWIND.MDB"
C1.C1Preview.DataBinding.DataSet dSet1 = _
  new C1.C1Preview.DataBinding.DataSet(dSrc, _
  "select * from Products")
doc.DataSchema.DataSources.Add(dSrc)
doc.DataSchema.DataSets.Add(dSet1)
Dim rt As New RenderText()
rt.DataBinding.DataSource = dSet1
rt.Text = "[DataBinding.RowNumber]: [Fields!ProductName.Value]"
doc.Body.Children.Add(rt)
```

  **C#コードの書き方**

  C#

```
C1PrintDocument doc = new C1PrintDocument();
DataSource dSrc = new DataSource();
dSrc.ConnectionProperties.DataProvider = DataProviderEnum.OLEDB;
dSrc.ConnectionProperties.ConnectString =
  @"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=NWIND.MDB";
C1.C1Preview.DataBinding.DataSet dSet1 =
  new C1.C1Preview.DataBinding.DataSet(dSrc,
  "select * from Products");
doc.DataSchema.DataSources.Add(dSrc);
doc.DataSchema.DataSets.Add(dSet1);
RenderText rt = new RenderText();
rt.DataBinding.DataSource = dSet1;
rt.Text = "[DataBinding.RowNumber]: [Fields!ProductName.Value]";
doc.Body.Children.Add(rt);
```


Copyright © GrapeCity, Inc. All rights reserved.

## フィルタ、グループ化、およびソート内で式にアクセス可能な ID

フィルタ、グループ化、およびソート式では、以下の ID のサブセットにアクセスできます。

- Document (**C1PrintDocument**型)
- DataBinding (**C1DataBinding**型)
- Fields (**FieldCollection**型)

これらのオブジェクト型についての詳細は、テキスト式でアクセス可能な IDを参照してください。

## DataSet 内の集計フィールドの指定に使用する式でアクセス可能な ID

データセット内の集計フィールドを指定するために使用する式では、以下の ID のサブセットにアクセスできます。

- Document (**C1PrintDocument**型)
- Fields (**FieldCollection**型)

これらのオブジェクト型についての詳細は、テキスト式でアクセス可能な IDを参照してください。

## データ連結

**C1PrintDocument** にデータ連結が追加されたことで、データ連結形式を **C1PrintDocument**（**RDL**、**C1Report** レポート定義など）にインポートしたり、**C1PrintDocument** をレポートジェネレータとして活用することができます。

**C1PrintDocument** にデータ連結を追加すると、**C1DataBinding** 型の **RenderObject** クラスに新しい複合プロパティが追加されます。これにより、レンダリングオブジェクトで表示されるデータのデータソースを指定できます。また、データ連結では、データソース内のすべてのレコードに対してレンダリングオブジェクトを繰り返すように指定できます。この場合、レンダリングオブジェクトは、水平区切りレポートジェネレータの「領域」と同じになります。これは、Microsoft の RDL 定義と同じです。

このように、データ連結が追加されたことで、ドキュメントの生成には次の2つの段階が含まれます。

- すべてのデータ連結レンダリングオブジェクトが選択され、これらを（テンプレートとして）使用して、データに基づいて「実際の」レンダリングオブジェクトが作成されます。
- 生成されるドキュメントは、以前のバージョンと同様にページ区切りされます。

ドキュメントには、データベーススキーマ（**C1DataSchema**クラスによって表され、データベース接続情報、SQL クエリーなどを含む）を含めることができます。ドキュメント内の **C1DataBinding** オブジェクトは、そのスキーマのプロパティを参照できます。ドキュメント内のすべてのデータ連結オブジェクトがドキュメント自体の **C1DataSchema** のプロパティだけを参照する場合、そのドキュメントは「データのリフロー可能」になります。つまり、ドキュメントを作成したプログラムとは関係なく、データを完全に更新してドキュメントを再生成することができます。

また、**C1DataBinding** は既存のデータソース（**DataTable** など）を参照できます。これは、フォーム上またはその**C1PrintDocument**を作成したプログラムの任意の場所で作成されたものです。この場合は、そのプログラムのコンテキスト内でのみデータを更新してドキュメントを再生成できます。そのドキュメントを保存した後で（.C1D ファイルとして）読み込むと、そのデータとの接続はすべて失われます。

## レンダリングオブジェクトのデータ連結

レンダリングオブジェクトが作成されても、そのレンダリングオブジェクトに最初からデータ連結は作成されません。データ連結は、ユーザーコードで **DataBinding** プロパティが参照されたときに作成されます。たとえば、次のように記述します。

**Visual Basic コードの書き方**


Copyright © GrapeCity, Inc. All rights reserved.

Visual Basic

```
Dim rt As RenderText = New RenderText
' ...
If Not (rt.DataBinding Is Nothing) Then
MessageBox.Show("Data binding defined.")
End If
```

**C# コードの書き方**

C#

```
RenderText rt = new RenderText();
// ...
if (rt.DataBinding != null)
{
MessageBox.Show("Data binding defined.");
}
```

上のコードの条件は、常に True に評価されます。したがって、特定のレンダリングオブジェクトにデータ連結が存在するかどうかを確認するだけであれば、代わりに **DataBindingDefined** プロパティを使用します。

**Visual Basic コードの書き方**

Visual Basic

```
Dim rt As RenderText = New RenderText
' ...
If rt.DataBindingDefined Then
MessageBox.Show("Data binding defined.")
End If
```

**C# コードの書き方**

C#

```
RenderText rt = new RenderText();
// ...
if (rt.DataBindingDefined)
{
MessageBox.Show("Data binding defined.");
}
```

**メモ**:これは、**Windows Forms** コントロールクラスの **Handle** プロパティと **IsHandleCreated** プロパティに似ています。

ドキュメント生成時に、**RenderObjectsList** コレクションが形成されます。その結果として、次の3つの状況が考えられます。

- **Copies** が **null**。これは、オブジェクトがデータ連結ではなく、通常どおり処理されることを意味します。オブジェクトの **Fragments** プロパティを使用して、生成されたページに実際にレンダリングされたオブジェクトのフラグメントにアクセスできます。
- **Copies** が **null** ではなく、**Copies.Cound** が0。これは、オブジェクトはデータ連結だが、データセットが空である(レコードを含まない)ことを意味します。このような場合、オブジェクトはドキュメントにまったく表示されません。つまり、オブジェクトのフラグメントは生成されません。例については、**DataBinding** サンプルの「空のリストへの連結」を参照してください。
- **Copies** が **null** ではなく、**Copies.Cound** が0より大きい。これは、オブジェクトはデータ連結で、データソースが空で


Copyright © GrapeCity, Inc. All rights reserved.

はない(レコードを含む)ことを意味します。ドキュメント生成時に、レンダリングオブジェクトのいくつかのコピーが作成されて、このコレクションに配置されます。その後、これらのコピーが処理され、通常どおりコピーごとにフラグメントが生成されます。

## RenderTable によるデータ連結の使用

テーブル行／列グループの基本クラスである **TableVectorGroup**の **DataBinding** プロパティを使用して、RenderTable をデータ連結にできます。この例については、**DataBinding** サンプルを参照してください。

**使用可能なサンプルプロジェクト**:**RenderTable**へのバインディングの例については、**DataBinding** のサンプルを参照してください。

行のグループだけでなく、列のグループもデータ連結できることに注目してください。つまり、テーブルを下方向だけでなく、横方向にも拡大できます。

次のコードに示すように、グループ化は機能しますが、グループ階層は **TableVectorGroup** オブジェクトの階層に基づいていません。

**Visual Basic コードの書き方**

Visual Basic

```
Dim rt As C1.C1Preview.RenderTable = New C1.C1Preview.RenderTable
rt.Style.GridLines.All = C1.C1Preview.LineDef.Default

' テーブルヘッダー:
Dim c As C1.C1Preview.TableCell = rt.Cells(0, 0)
c.SpanCols = 3
c.Text = "Header"

' グループヘッダー:
c = rt.Cells(1, 0)
c.Text = "GroupId = [Fields!GroupId.Value]"
c.SpanCols = 3
c.Style.TextAlignHorz = C1.C1Preview.AlignHorzEnum.Center
c.Style.TextAlignVert = C1.C1Preview.AlignVertEnum.Center

' サブグループヘッダー:
c = rt.Cells(2, 0)
c.Text = "GroupId = [Fields!GroupId.Value] SubGroupId = [Fields!SubGroupId.Value]"
c.SpanCols = 3
c.Style.TextAlignHorz = C1.C1Preview.AlignHorzEnum.Center
c.Style.TextAlignVert = C1.C1Preview.AlignVertEnum.Center

'  サブグループデータ:
rt.Cells(3, 0).Text = "GroupId=[Fields!GroupId.Value]"
rt.Cells(3, 1).Text = "SubGroupId=[Fields!SubGroupId.Value]"
rt.Cells(3, 2).Text = "IntValue=[Fields!IntValue.Value]"

'GroupId フィールドによってグループ化されたデータ連結線のグループを作成します:
Dim g As C1.C1Preview.TableVectorGroup = rt.RowGroups(1, 3)
g.CanSplit = True
g.DataBinding.DataSource = MyData.Generate(20, 0)
g.DataBinding.Grouping.Expressions.Add("Fields!GroupId.Value")
g.Style.BackColor = Color.LightCyan
```


Copyright © GrapeCity, Inc. All rights reserved.

```
'SubGroupId によってグループ化されたネストされたグループを作成します：
Dim ng As C1.C1Preview.TableVectorGroup = rt.RowGroups(2, 2)
ng.CanSplit = True
ng.DataBinding.DataSource = g.DataBinding.DataSource
ng.DataBinding.Grouping.Expressions.Add("Fields!SubGroupId.Value")
ng.Style.BackColor = Color.LightPink

' Create yet deeper nested data bound group:
Dim ng2 As C1.C1Preview.TableVectorGroup = rt.RowGroups(3, 1)
ng2.DataBinding.DataSource = g.DataBinding.DataSource
ng2.Style.BackColor = Color.LightSteelBlue
```

**C# コードの書き方**

C#

```
RenderTable rt = new RenderTable();
rt.Style.GridLines.All = LineDef.Default;

// テーブルヘッダー:
TableCell c = rt.Cells[0, 0];
c.SpanCols = 3;
c.Text = "Header";

//グループヘッダー:
c = rt.Cells[1, 0];
c.Text = "GroupId = [Fields!GroupId.Value]";
c.SpanCols = 3;
c.Style.TextAlignHorz = AlignHorzEnum.Center;
c.Style.TextAlignVert = AlignVertEnum.Center;

// サブグループヘッダー:
c = rt.Cells[2, 0];
c.Text = "GroupId = [Fields!GroupId.Value] SubGroupId = [Fields!SubGroupId.Value]";
c.SpanCols = 3;
c.Style.TextAlignHorz = AlignHorzEnum.Center;
c.Style.TextAlignVert = AlignVertEnum.Center;

//  サブグループデータ:
rt.Cells[3, 0].Text = "GroupId=[Fields!GroupId.Value]";
rt.Cells[3, 1].Text = "SubGroupId=[Fields!SubGroupId.Value]";
rt.Cells[3, 2].Text = "IntValue=[Fields!IntValue.Value]";

// GroupId フィールドによってグループ化されたデータ連結線のグループを作成します：
TableVectorGroup g = rt.RowGroups[1, 3];
g.CanSplit = true;
g.DataBinding.DataSource = MyData.Generate(20, 0);
g.DataBinding.Grouping.Expressions.Add("Fields!GroupId.Value");
g.Style.BackColor = Color.LightCyan;

// SubGroupId によってグループ化されたネストされたグループを作成します：
TableVectorGroup ng = rt.RowGroups[2, 2];
```


Copyright © GrapeCity, Inc. All rights reserved.

```
ng.CanSplit = true;
ng.DataBinding.DataSource = g.DataBinding.DataSource;
ng.DataBinding.Grouping.Expressions.Add("Fields!SubGroupId.Value");
ng.Style.BackColor = Color.LightPink;

// さらに深くネストされたデータ連結グループを作成します：
TableVectorGroup ng2 = rt.RowGroups[3, 1];
ng2.DataBinding.DataSource = g.DataBinding.DataSource;
ng2.Style.BackColor = Color.LightSteelBlue;
```

上記のコードを図示すると、次の表のようになります。

<table>
<tr><td></td><td></td><td></td><td colspan="3">ヘッダー</td></tr>
<tr><td rowspan="3">グループ<br>1-3</td><td></td><td></td><td colspan="3">GroupId = [Fields!GroupId.Value]</td></tr>
<tr><td rowspan="2">グループ<br>2-2</td><td></td><td colspan="3">GroupId = [Fields!GroupId.Value]  SubGroupId = [Fields!SubGroupId.Value]</td></tr>
<tr><td>グループ<br>3-1</td><td>GroupId=[Fields!<br>GroupId.Value]</td><td>SubGroupId=[Fields!<br>SubGroupId.Value]</td><td>IntValue=[Fields!<br>IntValue.Value]</td></tr>
</table>

# データ連結のサンプル

**DataBinding** サンプルには、いくつかのデータ連結ドキュメントのサンプルが含まれます。以下に、いくつかの重要なコード例について説明します。

# グループの使用

次のコードは、典型的なグループ化の使用方法を示します。:

**Visual Basic コードの書き方**

Visual Basic

```
' グループごとに繰り返されるレンダリング領域が作成されます。
Dim ra As C1.C1Preview.RenderArea = New C1.C1Preview.RenderArea
ra.Style.Borders.All = New C1.C1Preview.LineDef("2mm", Color.Blue)

' オブジェクトの MyData 配列がデータソースとして使用されます：
ra.DataBinding.DataSource = MyData.Generate(100, 0)

'GroupId フィールドでデータがグループ化されます：
ra.DataBinding.Grouping.Expressions.Add("Fields!GroupId.Value")

' グループヘッダーとして機能する RenderText を作成します。通常、このヘッダーは複雑になり、ヘッダー自体がデータ連結になります。
Dim rt As C1.C1Preview.RenderText = New C1.C1Preview.RenderText

' グループヘッダーは、「GroupId = XXX」のようになります。XXX は、グループの GroupId フィールドの値です：
rt.Text = "GroupId: [Fields!GroupId.Value]"
```


Copyright © GrapeCity, Inc. All rights reserved.

```
rt.Style.BackColor = Color.Yellow

'ヘッダーをグループ領域に追加します:
ra.Children.Add(rt)

' この RenderText は、各グループ内のレコードを出力します:
rt = New C1.C1Preview.RenderText

' レコードごとに出力するテキスト:
rt.Text = "GroupId: [Fields!GroupId.Value]" & Microsoft.VisualBasic.Chr(13) &
"IntValue: [Fields!IntValue.Value]"
rt.Style.Borders.Bottom = C1.C1Preview.LineDef.Default
rt.Style.BackColor = Color.FromArgb(200, 210, 220)

' テキストのデータソースを外側の RenderArea のデータソースに設定します。これは、レンダリングオブジェクトが指定されたオブジェクト内の現在のグループに連結されていることを示します:
rt.DataBinding.DataSource = ra.DataBinding.DataSource

' テキストを領域に追加します:
ra.Children.Add(rt)
```

**C# コードの書き方**

C#

```
//グループごとに繰り返されるレンダリング領域が作成されます。
RenderArea ra = new RenderArea();
ra.Style.Borders.All = new LineDef("2mm", Color.Blue);

// オブジェクトの MyData 配列がデータソースとして使用されます:
ra.DataBinding.DataSource = MyData.Generate(100, 0);

// GroupId フィールドでデータがグループ化されます:
ra.DataBinding.Grouping.Expressions.Add("Fields!GroupId.Value");

//グループヘッダーとして機能する RenderText を作成します。通常、このヘッダーは複雑になり、ヘッダー自体がデータ連結になります。
RenderText rt = new RenderText();

// グループヘッダーは、「GroupId = XXX」のようになります。XXX は、グループの GroupId フィールドの値です:
rt.Text = "GroupId: [Fields!GroupId.Value]";
rt.Style.BackColor = Color.Yellow;

// ヘッダーをグループ領域に追加します:
ra.Children.Add(rt);

// この RenderText は、各グループ内のレコードを出力します:
rt = new RenderText();

// レコードごとに出力するテキスト:
rt.Text = "GroupId: [Fields!GroupId.Value]\rIntValue: [Fields!IntValue.Value]";
rt.Style.Borders.Bottom = LineDef.Default;
rt.Style.BackColor = Color.FromArgb(200, 210, 220);
```


Copyright © GrapeCity, Inc. All rights reserved.

```
//テキストのデータソースを外側の RenderArea のデータソースに設定します。これは、レンダリングオブジェクトが指定されたオブジェクト内の現在のグループに連結されていることを示します:
rt.DataBinding.DataSource = ra.DataBinding.DataSource;

// テキストを領域に追加します:
ra.Children.Add(rt);
```

# 集計の使用

次のコードは、典型的なグループ化の使用方法を示します。

**Visual Basic コードの書き方**

Visual Basic

```
' 各グループで繰り返されるレンダリング領域を作成します:
Dim ra As C1.C1Preview.RenderArea = New C1.C1Preview.RenderArea
ra.Style.Borders.All = New C1.C1Preview.LineDef("2mm", Color.Blue)
ra.DataBinding.DataSource = MyData.Generate(20, 0, True)
ra.DataBinding.Grouping.Expressions.Add("Fields!GroupId.Value")

' 各グループ内の IntValue フィールドの合計を計算する集計を作成します:
Dim agg As C1.C1Preview.DataBinding.Aggregate = New
C1.C1Preview.DataBinding.Aggregate("Group_IntValue")

' 合計を計算する式を定義します:
agg.ExpressionText = "Fields!IntValue.Value"

'集計がグループ範囲を持つことを指定します:
agg.Running = C1.C1Preview.DataBinding.RunningEnum.Group

'  集計のデータソースを指定します:
agg.DataBinding = ra.DataBinding

' 集計をドキュメントに追加します。
doc.DataSchema.Aggregates.Add(agg)

' ドキュメント全体の IntValue フィールドの合計を計算する集計を作成します:
agg = New C1.C1Preview.DataBinding.Aggregate("Total_IntValue")

' 合計を計算する式を定義します:
agg.ExpressionText = "Fields!IntValue.Value"

' 集計がドキュメント範囲を持つことを指定します:
agg.Running = C1.C1Preview.DataBinding.RunningEnum.All

' 集計のデータソースを指定します:
agg.DataBinding = ra.DataBinding

' 集計をドキュメントに追加します。
doc.DataSchema.Aggregates.Add(agg)
```


Copyright © GrapeCity, Inc. All rights reserved.

```
' グループヘッダーを作成します:
Dim rt As C1.C1Preview.RenderText = New C1.C1Preview.RenderText
rt.Text = "GroupId: [Fields!GroupId.Value]"
rt.Style.BackColor = Color.Yellow
ra.Children.Add(rt)

' このレンダリングテキストは、グループレコードを出力します。コードからわかるように、グループ集計値はグループフッターだけでなく、グループヘッダーおよびグループ詳細でも参照できます:
rt = New C1.C1Preview.RenderText
rt.Text = "GroupId:
[Fields!GroupId.Value]\rIntValue:[Fields!IntValue.Value]\rGroup_IntValue:
[Aggregates!Group_IntValue.Value]\rTotal_IntValue:
[Aggregates!Total_IntValue.Value]\rTatalNested_IntValue:
[Aggregates!TatalNested_IntValue.Value]"

rt.Style.Borders.Bottom = C1.C1Preview.LineDef.Default
rt.Style.BackColor = Color.FromArgb(200, 210, 220)
rt.DataBinding.DataSource = ra.DataBinding.DataSource
ra.Children.Add(rt)

' この集計もグループに対して計算されますが、ネストされたオブジェクトのデータ連結に接続されます。
agg = New C1.C1Preview.DataBinding.Aggregate("TotalNested_IntValue")
agg.ExpressionText = "Fields!IntValue.Value"
agg.Running = RunningEnum.All
agg.DataBinding = rt.DataBinding
doc.DataSchema.Aggregates.Add(agg)

'領域をドキュメントに追加します。
doc.Body.Children.Add(ra)
```

**C# コードの書き方**

C#

```
//各グループで繰り返されるレンダリング領域を作成します:
RenderArea ra = new RenderArea();
ra.Style.Borders.All = new LineDef("2mm", Color.Blue);
ra.DataBinding.DataSource = MyData.Generate(20, 0, true);
ra.DataBinding.Grouping.Expressions.Add("Fields!GroupId.Value");

//各グループ内の IntValue フィールドの合計を計算する集計を作成します:
Aggregate agg = new Aggregate("Group_IntValue");

// 合計を計算する式を定義します:
agg.ExpressionText = "Fields!IntValue.Value";

// 集計がグループ範囲を持つことを指定します:
agg.Running = RunningEnum.Group;

//  集計のデータソースを指定します:
agg.DataBinding = ra.DataBinding;

//  集計をドキュメントに追加します。
```


Copyright © GrapeCity, Inc. All rights reserved.

```
doc.DataSchema.Aggregates.Add(agg);

// ドキュメント全体の IntValue フィールドの合計を計算する集計を作成します:
agg = new Aggregate("Total_IntValue");

//合計を計算する式を定義します:
agg.ExpressionText = "Fields!IntValue.Value";

// 集計がドキュメント範囲を持つことを指定します:
agg.Running = RunningEnum.All;

// 集計のデータソースを指定します:
agg.DataBinding = ra.DataBinding;

//集計をドキュメントに追加します。
doc.DataSchema.Aggregates.Add(agg);

// グループヘッダーを作成します:
RenderText rt = new RenderText();
rt.Text = "GroupId: [Fields!GroupId.Value]";
rt.Style.BackColor = Color.Yellow;
ra.Children.Add(rt);

// このレンダリングテキストは、グループレコードを出力します。コードからわかるように、グループ集計値はグループフッターだけでなく、グループヘッダーおよびグループ詳細でも参照できます:
rt = new RenderText();
rt.Text = "GroupId:
[Fields!GroupId.Value]\rIntValue:[Fields!IntValue.Value]\rGroup_IntValue:
[Aggregates!Group_IntValue.Value]\rTotal_IntValue:
[Aggregates!Total_IntValue.Value]\rTatalNested_IntValue:
[Aggregates!TatalNested_IntValue.Value]";

rt.Style.Borders.Bottom = LineDef.Default;
rt.Style.BackColor = Color.FromArgb(200, 210, 220);
rt.DataBinding.DataSource = ra.DataBinding.DataSource;
ra.Children.Add(rt);

// この集計もグループに対して計算されますが、ネストされたオブジェクトのデータ連結に接続されます。
agg = new Aggregate("TotalNested_IntValue");
agg.ExpressionText = "Fields!IntValue.Value";
agg.Running = RunningEnum.All;
agg.DataBinding = rt.DataBinding;
doc.DataSchema.Aggregates.Add(agg);

//領域をドキュメントに追加します。
doc.Body.Children.Add(ra);
```

ドキュメントの aggregate コレクション(**Aggregates**)内で宣言せずに、データバインドされた **C1PrintDocument** で使用できる集計タイプもあることに注意してください。詳細や例については、データ集計トピックを参照してください。

# データ集計



Copyright © GrapeCity, Inc. All rights reserved.

2010 v1 リリースで、新しい集計が **Reports for WinForms** に追加されました。これらの集計タイプは、ドキュメントの aggregates コレクション（**Aggregates**）内で宣言する必要なく、データバインドされた **C1PrintDocument** 内で使用できます。

たとえば、「残高」がデータバインドされたドキュメントのデータフィールドである場合、次の**RenderText** を使用し、このデータセットの合計残高を印刷できます。

**Visual Basic コードの書き方**

Visual Basic

```
Dim rt As New RenderText("[Sum(""Fields!Balance.Value"")]")
```

**C# コードの書き方**

C#

```
RenderText rt = new RenderText("[Sum(\"Fields!Balance.Value\")]");
```

この機能をサポートするために次の新しいプロパティとメソッドが **DataSet** および **C1DataBinding** 型に追加されました。

| クラス | メンバ | 説明 |
|---|---|---|
| **C1DataBinding** | **Name** プロパティ | 現在の C1DataBinding の名前を取得または設定します。この名前は、aggregate 関数で使用でき、aggregate が参照するデータバインディングを示します。 |
| **DataSet** | **Name** プロパティ | 現在の DataSet の名前を取得または設定します。この名前は、aggregate 関数で使用でき、aggregate が参照するデータセットを示します。 |

すべての aggregate 関数は次の書式を持ちます。

AggFunc(expression, scope)

ここで

- 式は、行グループまたはデータセット行ごとに計算された式を定義する文字列です。
- 範囲は、aggregate が計算されるデータセットを識別する文字列です。省略された場合、aggregate は現在のデータセット（現在のグループ、データセットなど）に計算されます。
  指定された場合、ターゲットグループまたはデータセットの名前となります。

たとえば、データセットが ID、GroupID、SubGroupID、NAME、Q のフィールドを持ち、レコードが、GroupID と SubGroupID によってグループ化される場合、次のドキュメントが生成されます。

**Visual Basic コードの書き方**

Visual Basic

```
Dim doc As New C1PrintDocument()
Dim raGroupId As New RenderArea()
' ...必要に応じて、raGroupId プロパティを設定します。...
raGroupID.DataBinding.DataSource = dataSet
raGroupID.DataBinding.Name = "GroupID"
raGroupID.DataBinding.Grouping.Expressions.Add("Fields!GroupID.Value")

Dim raSubGroupID As New RenderArea()
'...必要に応じて、raSubGroupID プロパティを設定します。...
raSubGroupID.DataBinding.DataSource = dataSet
raSubGroupID.DataBinding.Grouping.Expressions.Add("Fields!SubGroupID.Value")
```


Copyright © GrapeCity, Inc. All rights reserved.

```
raGroupID.Children.Add(raSubGroupID)

Dim raDetail As New RenderArea()
'  ...必要に応じて、raDetail プロパティを設定します。...
raDetail.DataBinding.DataSource = dataSet
raSubGroupID.Children.Add(raDetail)

'Q フィールドの値を表示します。
Dim rtQ As New RenderText()
rtQ.Text = "[Fields!Q.Value]"
raDetail.Children.Add(rtQ)

' ネストされたグループ(SubGroupID)の Q フィールドの合計を表示します。
Dim rtSumQ1 As New RenderText()
rtSumQ1.Text = "[Sum(""Fields!Q.Value"")]"
raDetail.Children.Add(rtSumQ1)
' GroupID の Q フィールドの合計を表示します。
Dim rtSumQ2 As New RenderText()
rtSumQ2.Text = "[Sum(\"Fields!Q.Value\", "\"GroupID\"")]"
raDetail.Children.Add(rtSumQ2)
' show TOTAL sum of Q field for the entire dataset:
Dim rtSumQ3 As New RenderText()
rtSumQ3.Text = "[Sum(\"Fields!Q.Value\", "\"DataSet\"")]"
raDetail.Children.Add(rtSumQ3)
doc.Body.Children.Add(raGroupId)
```

**C# コードの書き方**

C#

```
C1PrintDocument doc = new C1PrintDocument();
RenderArea raGroupId = new RenderArea();
// ...必要に応じて、raGroupId プロパティを設定します。...
raGroupID.DataBinding.DataSource = dataSet;
raGroupID.DataBinding.Name = "GroupID";
raGroupID.DataBinding.Grouping.Expressions.Add("Fields!GroupID.Value");

RenderArea raSubGroupID = new RenderArea();
// ...必要に応じて、raSubGroupID プロパティを設定します。...
raSubGroupID.DataBinding.DataSource = dataSet;
raSubGroupID.DataBinding.Grouping.Expressions.Add("Fields!SubGroupID.Value");
raGroupID.Children.Add(raSubGroupID);

RenderArea raDetail = new RenderArea();
//  ...必要に応じて、raDetail プロパティを設定します。...
raDetail.DataBinding.DataSource = dataSet;
raSubGroupID.Children.Add(raDetail);

// Q フィールドの値を表示します。
RenderText rtQ = new RenderText();
rtQ.Text = "[Fields!Q.Value]";
raDetail.Children.Add(rtQ);
// ネストされたグループ(SubGroupID)の Q フィールドの合計を表示します。
```



Copyright © GrapeCity, Inc. All rights reserved.

```
RenderText rtSumQ1 = new RenderText();
rtSumQ1.Text = "[Sum(\"Fields!Q.Value\")]";
raDetail.Children.Add(rtSumQ1);
// GroupID の Q フィールドの合計を表示します。
RenderText rtSumQ2 = new RenderText();
rtSumQ2.Text = "[Sum(\"Fields!Q.Value\", "\"GroupID\"")]";
raDetail.Children.Add(rtSumQ2);
//データセット全体の Q フィールドの合計を表示します。
RenderText rtSumQ3 = new RenderText();
rtSumQ3.Text = "[Sum(\"Fields!Q.Value\", "\"DataSet\"")]";
raDetail.Children.Add(rtSumQ3);

doc.Body.Children.Add(raGroupId);
```

上のドキュメントが生成されるとき、raDetail グループの各インスタンスは次の4つの値を表示します。

- 「Q」フィールドの現在の値。
- 現在の SubGroupID に対する「Q」フィールドの合計
- 現在の GroupID に対する「Q」フィールドの合計
- ドキュメント全体に対する「Q」フィールドの合計

# 目次

**C1PrintDocument** は、目次（**TOC**）の自動生成をサポートします。目次そのものは、専用表示オブジェクトの**RenderToc**, によって表されます。このオブジェクトは、**RenderArea** から派生し、**TOC** 固有の機能を追加します。**TOC** 内の個別項目は、**RenderTocItem** (**RenderParagraph**から派生）によって表されます。**TOC** の各項目は、ドキュメント内の場所（アンカーによって表される）をポイントするハイパーリンク（**RenderTocItem.Hyperlink** プロパティ）を保持します。そこで、同じメカニズムが、**TOC** 項目をハイパーリンクが指定するドキュメントのコンテンツに連結するために使用されます。以下に示すように、**TOC** の作成には便利な方法が用意されています。

ドキュメントに目次を追加するには、以下の手順を実行します。

1. **RenderToc** クラスのインスタンスを作成し、ドキュメントの TOC を表示する位置に追加します。
2. 個別項目（**RenderTocItem** 型）を **RenderToc** インスタンスに追加します。このためには、以下のいずれかの方法（またはこれらの組み合わせ）を使用できます。
    - 項目のテキスト、ポイントする場所、およびオプションで **TOC** 内のレベルを提供する **RenderToc.AddItem** メソッドの任意のオーバーロードを使用して、**TOC** 項目を **TOC** に追加できます。
    - **RenderTocItem** クラスのインスタンスをコードで作成し、プロパティを設定して、それを TOC 表示オブジェクトの **Children** コレクションに追加できます。
    - **RenderToc** のインスタンスを引数として受け入れる **RenderTocItem** コンストラクタのオーバーロードが用意されています。このコンストラクタを使用すると、新しく作成された **TOC** 項目がコンストラクタ内の TOC に追加されます。このため、それを **TOC** オブジェクトの **Children** コレクションに手動で追加する必要はありません。

使用可能なサンプル：専用の **RenderToc** 表示オブジェクトを使用してドキュメントの目次を作成する方法の完全なサンプルについては、**RenderTOC** のサンプルを参照してください。

# 用語索引

**C1PrintDocument**を使用して、自動的に索引を生成できるようになりました。各索引（1つのドキュメント内に複数の索引を生成できます）は、アルファベット順にソートされたエントリを含む文字見出しのリストの後にエントリが含まれるページ番号が続くリストで構成されます。

索引は、**RenderArea** から派生した表示オブジェクトである **RenderIndex** のインスタンスによって表されます。このオブジェクトは、他の表示オブジェクトと同じようにドキュメントに追加できますが、索引に含まれるエントリ（用語）は、すべて出現した後にドキュメントに表示する必要があるという重要な制限があります。したがって、従来の索引と同様、ここで説明する索引が最


Copyright © GrapeCity, Inc. All rights reserved.

も良く機能するのはドキュメントの末尾に表示された場合です。

用語は、ドキュメント内に出現し、索引のエントリとして表示される単語または単語の組み合わせです。ドキュメントが作成されると、これらの用語は、出現した場所に関する情報（通常、用語を含む**RenderText** または**RenderParagraph** ）と共にRenderIndex オブジェクトに追加されます。その後、ドキュメントが生成されると、RenderIndex オブジェクトが実際の索引を生成します。

## 索引生成機能をサポートするクラス

以下の専用クラスが索引生成機能をサポートします。

- **RenderIndex**: このクラスは、**RenderArea**から派生し、**C1PrintDocument** に挿入され、そのドキュメントが生成されたときに索引を生成します。
  - RenderIndex は、すべての索引エントリが出現した後に、ドキュメントに表示される必要があります。この制限が行われる理由は、索引の実際の内容（したがって、索引が占有するスペース量）がエントリの出現に応じて大幅に変わる可能性があるからです。
- **IndexEntry**: このクラスは、索引内の索引エントリ（用語）を記述するために使用されます。
  - 各エントリは、複数の出現（用語が記述または参照されるドキュメント内の場所）が関連付けられている場合があります。エントリのすべての出現のコレクションは、IndexEntry の**Occurrences** プロパティを使用して表示されます。
  - 出現するたびに、ドキュメントが生成されるときに、索引内にハイパーリンク付きページ番号が生成されます。さらに、エントリには、サブエントリのリストが含まれる場合があります（**Children**によって表示されます）。ネストされるレベルは無制限ですが、通常、最大3レベルが使用されます。
  - 最後に、エントリの**SeeAlso** プロパティに、生成される索引内の現在のエントリの参照として表示される索引エントリのリストを含めることにより、エントリを索引内の他のエントリにリンクさせることができます。
- **IndexEntryOccurrence**: このクラスは、ドキュメント内のエントリの単一の出現を記述します。
  - この型の要素は、IndexEntry の Occurrences コレクション内に含まれています。
  - 1つの索引エントリのインスタンスを作成するときに、1回または複数回の出現を指定し（コンストラクタのパラメータとして）、後でさらなる出現をエントリに追加できます。
  - このクラスの主な機能プロパティは、出現の場所をポイントする**C1LinkTarget**,型の **Target**です。

## コードによる索引の生成

通常、コードで作成されるドキュメント内に簡単な1レベルの索引を作成する場合、以下の手順が使用されます。

1. **RenderIndex** クラスのインスタンスを作成し、ローカル変数内に格納する必要があります（上述のとおり、索引はそのエントリの出現の前に作成することはできません）。
2. コンテンツ（表示オブジェクト）がドキュメントに追加されると、いくつかのプログラムロジックが索引内のエントリ（用語）となる文字列を識別します。これらの各文字列は、手順1で作成した索引オブジェクトの Entries コレクションにすでに追加されているかどうかがテストされます。これが新規エントリである場合、それに対応する新しい **IndexEntry** オブジェクトが作成され、索引に追加されます。
3. ドキュメント内の出現の場所をポイントするために、エントリの出現(**IndexEntryOccurrence**) を、既存または新規に作成されたエントリに追加する必要があります。通常、この場所は、それを含み、ドキュメントに追加される **RenderObject** によって識別されます。
4. エントリのすべての出現がドキュメントに追加されると、手順1で作成した RenderIndex がドキュメントの本文に追加されます。
5. ドキュメントが生成されると、RenderIndex オブジェクトが、それに追加されたエントリのハイパーリンク付き索引を生成します。エントリは自動的にソートされ、各文字に対応するグループに分割され、文字見出しが追加されます。

もちろん、これは、索引の作成に関連する主要なオブジェクトの関係を示すために設計された、考えられる簡単なシナリオの1つに過ぎません。上記以外の可能性として、ドキュメントを作成する前に（たとえば、外部ディクショナリの用語に基づいて）索引付けされた用語を作成したり、ネストされたエントリ（サブエントリ）を追加したりすることができます。


Copyright © GrapeCity, Inc. All rights reserved.

# 索引の外観のカスタマイズ

以下のプロパティは、生成される索引の外観をカスタマイズする方法を提供します。

**スタイル**(スタイルも参照):

- **Style**: 索引全体のスタイル(見出し、エントリなど)を指定します。
- **HeadingStyle**: 文字見出し(見出しは、その文字で始まるエントリのグループの前に来る文字です)に使用するスタイルを指定します。生成される索引では、各見出し(通常、その文字で始まるエントリのグループの前に来る大文字です)は、スタイルが適用された別個の表示オブジェクト(**RenderText**)によって表されます。
- **EntryStyles**: さまざまなレベルのエントリのスタイルを指定する索引付けされたプロパティです。たとえば、EntryStyles[0](VB では EntryStyles(0))は、最上位レベルのエントリのスタイルを指定し、EntryStyles[1](VB では EntryStyles(1))は、サブエントリのスタイルを指定します(索引内のネストされたレベルの番号が EntryStyles コレクション内の要素の数よりも大きい場合、コレクション内の最後のスタイルがネストされたスタイルに使用されます)。
- 生成される索引では、各エントリ(用語の後に、それが表示されるページのリストが続く)は、別個の**RenderParagraph** オブジェクトによって表され、エントリのネストされたレベルによって索引付けされたこのプロパティによって指定されたスタイルが適用されます。たとえば、このスタイルでは、改ページを挿入できるようにする(**MinOrphanLines**を使用して)前に、エントリのテキストの最小行数を指定できます。
- **EntryStyle**: これは、**EntryStyles** コレクションの最初の(索引ゼロの)要素に対するショートカットです。
- **SeeAlsoStyle**: エントリ間の相互参照の前に来る「**関連項目**」テキストのスタイルを指定できます(**SeeAlso**を参照)。.
- **Style**: 特定のエントリのスタイルをオーバーライドできます。
- **SeeAlsoStyle**: 特定の「**関連項目**」テキストのスタイルをオーバーライドできます。

**その他のプロパティ:**

- **RunIn**: **True** の場合、サブエントリがインデントされた別個の行ではなく、主要見出しに沿って表示されることを示すブール型プロパティ(デフォルトでは、**False**)です。
- **EntryIndent**: 主要エントリに対するサブエントリの相対的なインデントを指定する **Unit** プロパティです。デフォルトは 0.25 インチです。
- **EntryHangingIndent**: 次の行に対するエントリのテキストの最初の行の相対的な(左側への)突き出しインデントを指定する Unit プロパティです(参照のリストが1行に収まらない場合に使用されます)。デフォルトは -0.125 インチです。
- **LetterSplitBehavior**: 文字グループ(同じ文字で始まるエントリ)を縦方向に分割する方法を指定する**SplitBehaviorEnum**プロパティです。デフォルトは、**SplitBehaviorEnum.SplitIfNeeded** です。見出し(デフォルトでは、その文字によって表される)は、常に最初のエントリと一緒に出力されることに注意してください。
- **Italic**: に似ていますが、ボールドではなくイタリック体を使用します。
- **LetterFormat**: 文字見出しの書式設定に使用する文字列です。デフォルトは、「{0}」です。
- **TermDelimiter**:エントリ用語と用語の出現リスト(ページ番号)を区切るために使用する文字列です。デフォルトでは、カンマの後にスペースを続けます。
- **RunInDelimiter**: 連続(**RunIn** を参照)索引を生成するときに、エントリの区切りに使用する文字列です。デフォルトは、セにコロンです。
- **OccurrenceDelimiter**: エントリの出現リスト(ページ番号)を区切るために使用する文字列です。デフォルトでは、カンマの後にスペースを続けます。
- **PageRangeFormat**: エントリの出現のページ範囲を書式設定するために使用する書式設定文字列です。デフォルトは、「{0}-{1}」です。
- **SeeAlsoFormat**:「**関連項目**」参照の書式設定に使用する文字列です。デフォルトは、「 (see {0})」です(スペースの後に、左小括弧、参照の出力に使用する書式項目、右小括弧が順に続きます)。
- **FillChar**: ページ番号が右揃えされる場合(**PageNumbersAtRight** が **True** の場合)に、充てん文字として使用する文字です。このプロパティのデフォルト値はドットです。
- **PageNumbersAtRight**: ページ番号を右揃えにするかどうかを指定するブール型プロパティです。デフォルトは False です。
- **EntrySplitBehavior**: 単一のエントリを縦方向に分割する方法を指定する **SplitBehaviorEnum** プロパティです。デフォルトは、**SplitBehaviorEnum.SplitIfLarge** です。このプロパティは、すべてのレベルのエントリに適用されます。
- **Bold**:特定のボールド体を使用するエントリの出現に対応するページ番号をハイライト表示できるブール型プロパティです(たとえば、用語の主な定義を提供する場所をハイライト表示するために使用できます)。


Copyright © GrapeCity, Inc. All rights reserved.

# 索引のスタイルの階層

索引固有のスタイルの階層は、以下のとおりです。

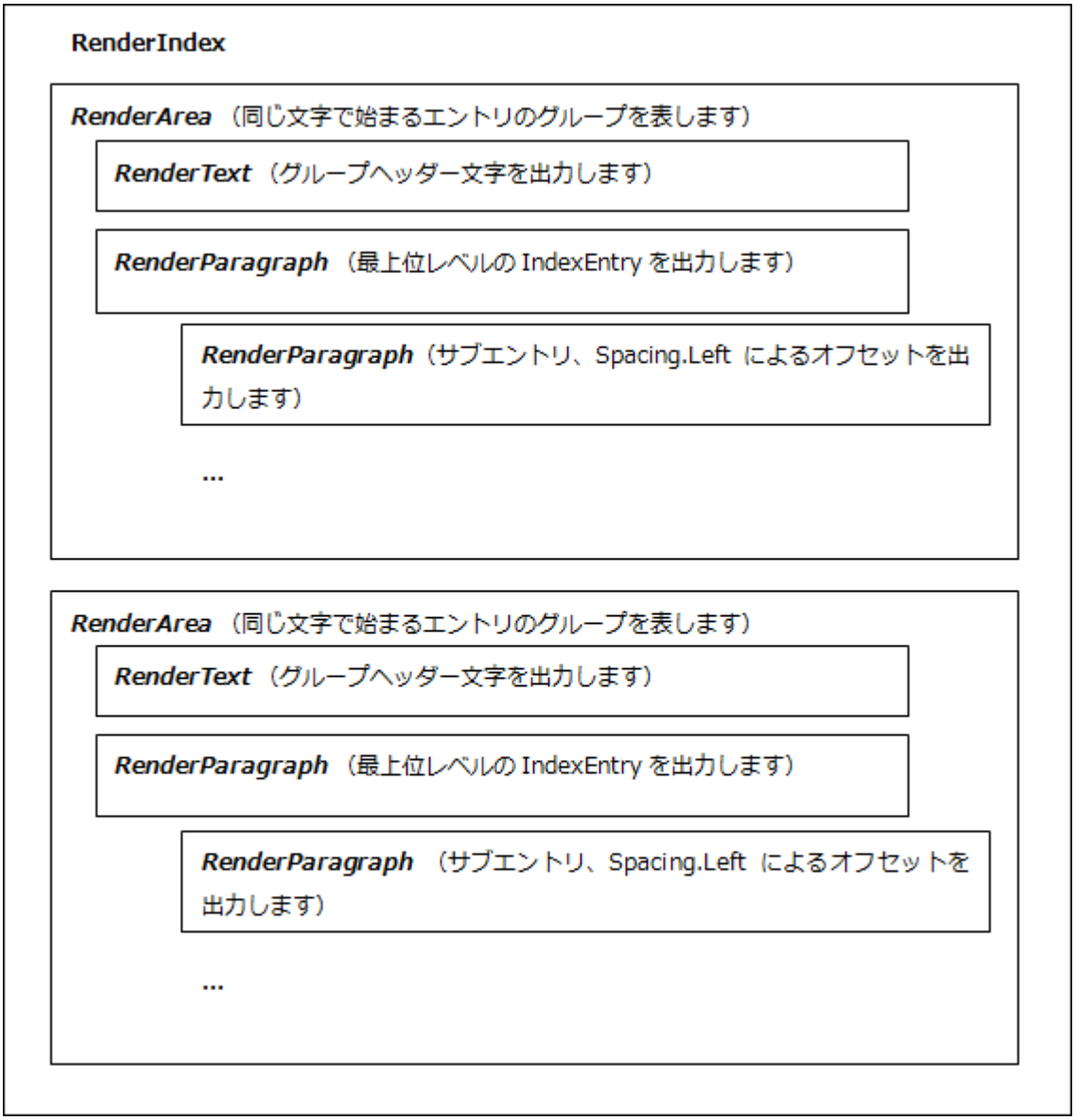
- **RenderIndex** の **Style** は、他のすべての索引固有のスタイルに対する AmbientParent として機能します。
- **HeadingStyle**
- **EntryStyles**
  - **Style**
- **SeeAlsoStyle**
  - **SeeAlsoStyle**

RenderIndex の Style を除き、上記に示したすべてのスタイルは、関連するオブジェクトのインラインスタイルの**Parent** と **AmbientParent** の両方として機能します。たとえば、RenderIndex の Style でのフォントの設定は、その索引のすべての要素に影響を与えますが（下位レベルのスタイルによってオーバーライドされないかぎり）、そのスタイルに枠線を指定すると、索引全体の周囲には表示されますが、索引内の個々の要素には表示されません。これに対して、たとえば SeeAlsoStyle に枠線を指定すると、索引内の各「関連項目」要素の周囲に枠線が表示されます。

# 生成される索引の構造

下図は、ドキュメントが生成されるときに、**RenderIndex** オブジェクトによって作成される表示オブジェクトのツリー構造と階層を示しています（この図では、**RenderIndex** オブジェクトのみがユーザーコードによって最上位レベルに作成され、その他すべてのオブジェクトは自動的に作成されることに注意してください）。


Copyright © GrapeCity, Inc. All rights reserved.




## アウトライン

**C1PrintDocument** はアウトラインをサポートしています。ドキュメントのアウトラインは、ドキュメント内の位置をポイントするノード(**OutlineNode**型）から成るツリーとして、**Outlines** プロパティで指定されます。アウトラインは、プレビューのナビゲーションパネル内のタブに表示されます。項目をクリックすると、その項目に対応する位置に移動できます。アウトラインは、同じ構造をサポートする形式（PDF など）にもエクスポートされます。

アウトラインノードを作成するには、オーバーロード **Outline** コンストラクタのいずれかを使用します。アウトラインのテキスト、ドキュメント内の位置（描画オブジェクトまたはアンカー）、およびプレビュー内のアウトラインツリーパネルに表示されるアイコンを指定できます。最上位のノードは、ドキュメントの **Outlines** コレクションに追加します。さらに、各アウトラインノードの **Children** コレクションに子ノードのコレクションを入れ、以下同様に繰り返すことができます。

**ヒント**:ドキュメントにレンダリングされる各項目のアウトラインノードはクリックできます。ノードをクリックすると、そのノードにアタッチされている項目が表示されます。アウトラインノードを追加する例については、**アウトライン]タブへのアウトラインエントリの追加**を参照してください。

## 埋め込みフォント


Copyright © GrapeCity, Inc. All rights reserved.

**C1PrintDocument** が C1DX または C1D ファイルとして保存される場合、ドキュメントで使用するフォントがそのファイルに埋め込まれます。この場合、そのドキュメントが別のシスタム上のファイルからロードされる場合、現在のシステムが元のフォントをすべてインストールしていない場合でも、埋め込まれているフォントで表示されるテキストは正しく表示されるように保証されています。フォントの埋め込みは、珍しいフォントや専用フォントが使用される場合（バーコードを表示するフォントなど）特に有用です。**C1PrintDocument** にフォントを埋め込む場合、そのフォントからすべてのグリフが埋め込まれる訳ではないことに注意してください。この場合、実際にフォントで使用されるグリフだけが埋め込まれます。

以下の **C1PrintDocument** プロパティがフォントの埋め込みに関連します。

- **EmbeddedFonts** - これは、ドキュメントに埋め込まれたフォントを含むコレクションです。フォントは自動的に作成される一方、フォント埋め込みのカスタムコントロールによって手動で変更できることにも注意してください（詳細については、後述の説明を参照してください）。
- **DocumentFonts** - このコレクションは、**C1PrintDocument.FontHandling** プロパティの値に応じて自動的に作成されます。ドキュメントで使用されるフォントを見つけるために使用できます。
- **FontHandling** - このプロパティは、2つの関連コレクション（**EmbeddedFonts** と **DocumentFonts**）が作成されるかどうかとどのように作成されるかを決定します。

## フォントの代替

テキストがフォントを使用して表示され、グリフが指定したフォントでは存在しないテキストに表示される場合、グリフを表示する代替フォントを選択することができます。たとえば、日本語象形文字を表示するために **Arial** フォントが使用される場合、実際のテキストの表示には **Arial Unicode MS** フォントが使用される場合があります。**C1PrintDocument** は、これを分析し、**DocumentFonts** および**EmbeddedFonts** コレクションに使用する実際のフォント（指定したフォントではなく）を追加できます。これを行うには、**FontHandling**を **FontHandling.BuildActualDocumentFonts** または **FontHandling.EmbedActualFonts** に設定する必要があります。これらの設定の欠点は、ドキュメントの生成速度が低下し、時間がかかることです。このため、指定したフォントから欠落している可能性のある文字がドキュメントに含まれている場合にのみ（一般的なラテンフォントを使用した極東言語のテキストなど）、これらの設定を使用することをお勧めします。

フォントの代替を分析する場合、以下の定義済みフォントのセットを検索し、欠落したグリフを含む最もよく一致したフォントを見つけます。

- MS UI Gothic
- MS 明朝
- Arial Unicode MS
- Batang
- Gulim
- Microsoft YaHei
- Microsoft JhengHei
- MingLiU
- SimHei
- SimSun

## 選択フォントの埋め込み

一般的に使用可能な **Arial** などのフォントと共に専用フォントをドキュメントで使用している場合、ドキュメントで使用するすべてのフォントを埋め込むのではなく、その専用フォントのみ、または複数の専用フォントを埋め込みたい場合があります。これを行うには、以下の手順を実行します。

1. **FontHandling** を **FontHandling.EmbedFonts** または **FontHandling.EmbedActualFonts** 以外の値に設定します。これにより、ドキュメントを生成するときに、**EmbeddedFonts** コレクションが空のままになります。
2. コードを使用して、専用フォント（現在のシステムにインストールされている必要があります）をドキュメントの **EmbeddedFonts** コレクションに追加します。**EmbeddedFont** を作成するには、カスタムフォントに対応する **.NET Font** オブジェクトを **EmbeddedFont** のコンストラクタに渡します。**EmbeddedFont.AddGlyphs** メソッドを使用して、必要なグリフをフォントに追加します（複数のオーバーロードが用意されています）。

この方法で作成した **EmbeddedFonts** コレクションを含む **C1PrintDocument** を保存すると（**C1DX** または **C1D** ファイルと


Copyright © GrapeCity, Inc. All rights reserved.

して)、このコレクションで指定されたフォントだけがドキュメントに埋め込まれます。

## 辞書

同じ項目(画像またはアイコン)がドキュメント内の多くの場所で使用される場合は、同じ画像またはアイコンをそれらすべての場所に挿入するのではなく、その項目をドキュメント全体からアクセスできる位置に一度保存し、そのインスタンスを参照するようにできると便利です。そのために、**C1PrintDocument** には辞書が用意されています。辞書にアクセスするには、**Dictionary** プロパティを使用します。辞書には、基本抽象型 **DictionaryItem**の項目が含まれています。画像とアイコンを保存するために、**DictionaryImage** と **DictionaryIcon**の2つの派生クラスが用意されています。辞書内の項目は名前で参照されます。項目を使用するには、項目に名前を割り当てる必要があります。名前は辞書内で一意である必要があります。辞書項目はドキュメント内の次の場所で使用できます。

- スタイルの背景画像(**BackgroundImageName** プロパティを使用)。
- **RenderImage**オブジェクト内の画像(**ImageName**プロパティを使用)。

同じ画像をドキュメント内の複数の場所で使用するには、次の手順に従います。

- 画像を辞書に追加します。次に例を示します。

  **Visual Basic コードの書き方**

  Visual Basic
  ```
  Me.C1PrintDocument1.Dictionary.Add(New C1.C1Preview.DictionaryImage("image1",
  Image.FromFile("myImage.jpg")))
  ```

  **C# コードの書き方**

  C#
  ```
  this.c1PrintDocument1.Dictionary.Add(new DictionaryImage("image1",
  Image.FromFile("myImage.jpg")));
  ```

- スタイルの **BackgroundImageName** プロパティまたはレンダリング画像の **ImageName** プロパティを設定して、その画像を使用します。次に例を示します。

  **Visual Basic コードの書き方**

  Visual Basic
  ```
  Dim ri As New C1.C1Preview.RenderImage()
  ri.ImageName = "image1"
  ```

  **C# コードの書き方**

  C#
  ```
  RenderImage ri = new RenderImage();
  ri.ImageName = "image1";
  ```

## C1Report 定義

C1Previewには、**C1Report** 定義をインポートおよび生成する機能があります。.


Copyright © GrapeCity, Inc. All rights reserved.

C1Reportを **C1PrintDocument**にインポートするには、**ImportC1Report** メソッドを使用します。たとえば、次のコードを使用して、**C1Report** に付属する **ReportBrowser** サンプルの「**Alphabetical List of Products**」(製品のアルファベット順リスト)をインポートしてプレビューできます。

**Visual Basic コードの書き方**

```
Visual Basic
Dim doc As C1PrintDocument = New C1PrintDocument()
doc.ImportC1Report("NWind.xml", "Alphabetical List of Products")
Dim pdlg As C1PrintPreviewDialog = New C1PrintPreviewDialog()
pdlg.Document = doc
pdlg.ShowDialog()
```

**C# コードの書き方**

```
C#
C1PrintDocument doc = new C1PrintDocument();
doc.ImportC1Report("NWind.xml", "Alphabetical List of Products");
C1PrintPreviewDialog pdlg = new C1PrintPreviewDialog();
pdlg.Document = doc;
pdlg.ShowDialog();
```

**C1Report** が**C1PrintDocument**にインポートされると、ドキュメントは次のような構造になります。

## グループがないレポートの場合

- ページフッターは **RenderSection** オブジェクトによって表され、ドキュメントの **PageLayouts.Default.PageFooter** に割り当てられます。
- レポートの各セクションでは、RenderSection オブジェクトが作成されて、ドキュメント本体に次の順序で追加されます。
  - Header(レポートヘッダー、**SectionTypeEnum.Header**)
  - PageHeader (ページヘッダー、SectionTypeEnum.**PageHeader**)
  - Detail (データセクション、SectionTypeEnum.**Detail**)
  - Footer (レポートフッター、SectionTypeEnum.**Footer**)

**PageHeader** のコピーも **PageHeader.LayoutChangeAfter.PageLayout.PageHeader** に割り当てられます。**C1Report** では、レポートヘッダーの後に最初のページヘッダーが出力されるため、このような複雑な構造が必要になります。

## グループがあるレポートの場合

各グループで、**RenderArea** が作成され、**PageHeader** および **Footer** の間に次のオブジェクトツリーが配置されます(2つのグループの場合)。

- 最上位グループを表す**RenderArea**
  - GroupHeader1 を表す**RenderSection**
  - ネストされたグループを表す RenderArea
    - GroupHeader2 を表す RenderSection
    - Detail
    - GroupFooter2 を表す RenderSection
  - GroupFooter1 を表す RenderSection

# C1Report インポートの制限

**C1Report** を **C1PrintDocument** にインポートする際は、次の制限があります。


Copyright © GrapeCity, Inc. All rights reserved.

- XML ファイルに含まれるレポート定義だけをインポートできます。つまり、アプリケーションが **C1Report** イベントにアタッチされた C# または VB.NET ハンドラを介してコードでレポートを生成している場合、そのレポートを**C1PrintDocument**にインポートすることはできません。
- **C1Report** では、プリンタがインストールされておらず、**CustomWidth** および **CustomHeight** がどちらも0に設定されている場合、用紙サイズは常にレター(8.5×11 インチ)に設定されます。**C1PrintDocument** では、現在のロケールに基づいて用紙サイズが決定されます。たとえば、ヨーロッパのほとんどの国では、用紙サイズは A4 (210cm×297cm)に設定されます。
- スクリプトの制限:

  **C1PrintDocument**:

  - Font プロパティは読み取り専用です。

  **Field**:

  - **Section** プロパティは読み取り専用です。
  - **Font** プロパティは読み取り専用です。
  - **LineSpacing** プロパティは存在しません。
  - Field.Subreport プロパティは読み取り専用です。
  - **LinkTarget** プロパティに式が含まれる場合は、評価されずに文字列として使用されます。
  - **SubreportHasData** プロパティは存在しません。
  - **LinkValue** プロパティは存在しません。

  **Layout**:

  - **ColumnLayout** プロパティはサポートされません。列は常に上から下、左から右に並べられます。
  - **LabelSpacingX** プロパティは存在しません。
  - **LabelSpacingY** プロパティは存在しません。
  - **OverlayReplacements** プロパティは存在しません。
- **OnFormat** イベントハンドラでは、生成されるドキュメントのページ区切りに影響を及ぼすプロパティを変更しないでください。たとえば、ForcePageBreakを使用することはできません。
- レポートのパラメータを入力するためのダイアログは表示されません。代わりに、デフォルト値が使用されます。デフォルトが指定されていない場合は、パラメータの種類によって決定されます。たとえば、数値には0、文字列には空文字列、日付には現在の日付が使用されます。
- **PageHeader** および **PageFooter** では、データベースフィールドを使用できません。
- **C1Report** では、段組みレポートのレポートヘッダーがすべての列に出力されますが、**C1PrintDocument** では、最初の列だけに出力されます。
- 列では **Across** がサポートされません。

# プリンタドライバでの作業

**Reports for WinForms** の 2009 v1 では、プリンタドライバに起因する特定の問題に対処するために、いくつかの新しいメンバが追加されました。

プリンタドライバの問題を解決するために、以下のメンバが追加されました。

| クラス | メンバ | 説明 |
|---|---|---|
| **C1PreviewPane** | **AdjustPrintPage** イベント | ドキュメントの印刷に使用する C1PrintManager の **PrintPage** イベントハンドラから発生します。 |
| **C1PrintManager** | **AdjustPrintPage** イベント | 実際にページを印刷する前の、現在のプリントマネージャの **PrintDocument.PrintPage** イベントハンドラから発生します。 |
| **C1PrintOptions** | **DrawPrintableAreaBounds** プロパティ | ページの印刷可能領域の周囲に線を表示するかどうかを指定する値を取得または設定します(プリンタの |


Copyright © GrapeCity, Inc. All rights reserved.

| クラス | メンバ | 説明 |
|---|---|---|
| | | 問題のデバッグに便利です)。 |
| | **PrintableAreaBoundsPen** プロパティ | DrawPrintableAreaBounds が True の場合、印刷可能領域の枠線を表示するために使用するペンを取得または設定します。 |
| | **PrintAsBitmap** プロパティ | 印刷前に、ページのメタファイルをビットマップに変換し、プリンタのハードマージンにクリップするかどうかを指定する値を取得または設定します。 |

上記のテーブルに示されたメンバは、特定のプリンタの問題に対処するために使用できます。たとえば、**HP-CP1700** プリンタ(Vista の組み込みプリンタドライバを使用)を装備した **Windows Vista 64** を実行するマシンの以下のシナリオについて検討します。この例では、**ClipPage** が **False**(デフォルト)に設定され、ドキュメントのページがプリンタのマージンより広く、空のページが出力され、ドキュメントのコンテンツは印刷されませんでした。たとえば、以下のドキュメントは、印刷を実行したときに、空の2ページのみ生成しました。

**Visual Basic コードの書き方**

Visual Basic

```
Dim doc As New C1PrintDocument()
doc.Style.Font = New Font("Arial", 32)
For i As Integer = 0 To 19
Dim rtx As New RenderText(i.ToString())
rtx.X = String.Format("{0}in", i)
rtx.Y = "10cm"
rtx.Style.FontSize = 64
doc.Body.Children.Add(rtx)
Next
doc.Generate()
```

**C# コードの書き方**

C#

```
C1PrintDocument doc = new C1PrintDocument();
doc.Style.Font = new Font("Arial", 32);
for (int i = 0; i < 20; ++i)
{
RenderText rtx = new RenderText(i.ToString());
rtx.X = string.Format("{0}in", i);
rtx.Y = "10cm";
rtx.Style.FontSize = 64;
doc.Body.Children.Add(rtx);
}
doc.Generate();
```

この問題は、以下の2つの方法によって対処できるようになりました。

1. **PrintAsBitmap** を True に設定します(たとえば、**C1PreviewPane** 上で).
2. 以下のイベントハンドラを**AdjustPrintPage** イベントに関連付けます(**AdjustPrintPage** 経由でも可能です)。

   **Visual Basic コードの書き方**

   Visual Basic


Copyright © GrapeCity, Inc. All rights reserved.

```
Dim doc As New C1PrintDocument()
doc.Style.Font = New Font("Arial", 32)
Private Sub PreviewPane_AdjustPrintPage(ByVal sender As Object, ByVal e As
AdjustPrintPageEventArgs)
Dim pa As RectangleF = e.PrintableArea
If Not e.PrintPageEventArgs.PageSettings.Landscape Then
pa.Width = 800 ' システムは 824 に設定されます。
pa.X = 25 ' システムは 13 に設定されます。
pa.Y = 13 'システムは 6.666... に設定されます。
Else
pa.X = 13
pa.Y = 0
End If
e.PrintableArea = pa
End Sub
```

**C# コードの書き方**

C#

```
C1PrintDocument doc = new C1PrintDocument();
doc.Style.Font = new Font("Arial", 32);
void PreviewPane_AdjustPrintPage(object sender, AdjustPrintPageEventArgs e)
{
RectangleF pa = e.PrintableArea;
if (!e.PrintPageEventArgs.PageSettings.Landscape)
{
pa.Width = 800; // システムは 824 に設定されます。
pa.X = 25; // システムは 13 に設定されます。
pa.Y = 13; // システムは 6.666... に設定されます。
}
else {
pa.X = 13;
pa.Y = 0;
}
e.PrintableArea = pa;
}
```

このコードは、プリンタドライバによって設定された、間違ったハードページマージンを修正し、上記の問題を回避します。

# レポート定義言語(RDL)ファイル

RDL ファイルインポートのサポートが **C1PrintDocument** に含まれています。レポート定義言語(**RDL**)インポートにより、**RDL** レポート定義を **C1PrintDocument** コンポーネントのインスタンスに読み込むことができます。結果のドキュメントは、インポートされたレポートのデータバインドされた表現です。**C1PrintDocument** の **RDL** サポートは Microsoft RDL Specification for SQL Server 2008に基づいています。

> **注意**: **C1PrintDocument** での **RDL** サポート(**ImportRdl** と **FromRdl** メソッドで提供)は、現在廃止されています。代わりに、**C1RdlReport** を使用する必要があります。詳細については、C1RdlReport の使い方を参照してください。

次のコードを使用し、**RDL** ファイルをインポートできます。

**Visual Basic コードの書き方**


Copyright © GrapeCity, Inc. All rights reserved.

Visual Basic

```
Dim doc As New C1PrintDocument()
doc.ImportRdl("myReport.rdl")
doc.Generate()
```

**C# コードの書き方**

C#

```
C1PrintDocument doc = new C1PrintDocument();
doc.ImportRdl("myReport.rdl");
doc.Generate();
```

現時点で、すべての **RDL** プロパティはサポートされていませんが、将来のリリースでサポートが追加される予定です。詳細については、RDL インポートの制限トピックを参照ください。

# RDL インポートの制限

**Reports for WinFroms** の RDL インポートの現在の実装にはいくつかの制限があります。これらの制限は次のとおりです。

- Gauge、Chart、および SubReport オブジェクトはサポートされません。
- 複合 RDL プロパティのサブプロパティ（枠線の幅など）の式はサポートされません。
- RDL の大部分の aggregate 関数はオプションの「再帰型」パラメータを持っていますが、サポートされません。
- 次の RDL プロパティはまだ完全にはサポートされていません。
- QueryParameter.Value：リテラル値のみ指定できます。
- ReportParameter：データを参照するパラメータはサポートされません。
- Hyperlink：式として指定できません。複数のアクションが1つのハイパーリンクに関連付けられている場合、最初のアクションがサポートされます。
- ReportItem.Visibility：式として指定できません。
- ReportItem.Bookmark：式として指定できません。
- Style.TextAlign.General：左揃えが使用されます。

次の RDL プロパティは現在サポートされていません。

- Document.AutoRefresh
- Document.CustomProperties
- Document.Code
- Document.Width
- Document.Language
- Document.CodeModules
- Document.Classes
- Document.ConsumeContainerWhitespace
- Document.DataTransform
- Document.DataSchema
- Document.DataElementName
- Document.DataElementStyle
- ConnectionProperties.Prompt
- ConnectionProperties.DataProvider
- DataSet.CaseSensitivity
- DataSet.Collation
- DataSet.AccentSensitivity
- DataSet.KanatypeSensitivity
- DataSet.WidthSensitivity
- DataSet.InterpretSubtotalsAsDetails
- Body.Height


Copyright © GrapeCity, Inc. All rights reserved.

- ReportItem.ToolTip
- ReportItem.DocumentMapLabel
- ReportItem.CustomProperties
- ReportItem.DataElementName
- ReportItem.DataElementOutput
- TextBox.HideDuplicates
- TextBox.ToggleImage
- TextBox.UserSort
- TextBox.DataElementStyle
- TextBox.ListStyle
- TextBox.ListLevel
- TextRun.ToolTip
- TextRun.MarkupType
- Style.Format
- Style.LineHeight
- Style.Direction
- Style.Language
- Style.Calendar
- Style.NumeralVariant
- Style.TextEffect

**注意**:**C1PrintDocument** での **RDL** インポート(**ImportRdl** と **FromRdl** メソッドで提供)は、現在廃止されています。代わりに、**C1RdlReport** を使用する必要があります。詳細については、C1RdlReport の使い方を参照してください。


Copyright © GrapeCity, Inc. All rights reserved.

# C1MultiDocument の使い方

**C1MultiDocument**コンポーネントは、メモリの制限により、単一の**C1PrintDocument** オブジェクトでは処理できない大きなドキュメントを作成、保持、エクスポートできるように設計されています。

**C1MultiDocument** オブジェクトは、**C1MultiDocumentItem** 型の要素を複数格納できる **Items** コレクションを提供します。これらの要素はそれぞれ **C1PrintDocument** を表します。圧縮と一時ディスク格納の使用により、すべてのページが単一の **C1PrintDocument** に属していると、メモリ不足を引き起こすような複数の **C1PrintDocument** オブジェクトを、1つの大きなマルチドキュメントにまとめることができます。以下のコードの断片は、マルチドキュメントの作成およびプレビュー方法を示しています。

**Visual Basic コードの書き方**

Visual Basic

```
Dim mdoc As New C1MultiDocument()
mdoc.Items.Add(C1PrintDocument.FromFile("myDoc1.c1dx"))
mdoc.Items.Add(C1PrintDocument.FromFile("myDoc2.c1dx"))
mdoc.Items.Add(C1PrintDocument.FromFile("myDoc3.c1dx"))
Dim pview As New C1PrintPreviewDialog()
pview.Document = mdoc
pview.ShowDialog()
```

**C# コードの書き方**

C#

```
C1MultiDocument mdoc = new C1MultiDocument();
mdoc.Items.Add(C1PrintDocument.FromFile("myDoc1.c1dx"));
mdoc.Items.Add(C1PrintDocument.FromFile("myDoc2.c1dx"));
mdoc.Items.Add(C1PrintDocument.FromFile("myDoc3.c1dx"));
C1PrintPreviewDialog pview = new C1PrintPreviewDialog();
pview.Document = mdoc;
pview.ShowDialog();
```

**C1MultiDocument** は、含まれている文書の間のリンク、共通の目次、共通ページ記数、および総ページ数をサポートします。

**C1MultiDocument** は、追加された **C1PrintDocument** オブジェクトへの参照を格納せず、(.c1d/x などに)シリアル化して、その結果を格納する点に注意してください。したがって、メモリ不足に陥ることなく、本当に大きなマルチドキュメントを作成することができます。もちろん、コード自体が、**C1MultiDocument** に追加された個々の **C1PrintDocument** オブジェクトへの参照を保持していないことが前提です。**C1MultiDocument** を使用する際は、ドキュメントオブジェクトをマルチドキュメントに追加した後、個々のドキュメントオブジェクトを保持していないことを確認してください。

**C1MultiDocument** は、デフォルトの拡張子 .c1mdx で**C1 Open XML Multi Document**として永続化することができます。

**C1MultiDocument** は、**Export** メソッドのオーバーロードを使用して、ほとんどの形式にエクスポートすることができます。詳細については、C1MultiDocument ファイルのエクスポートを参照してください。

**C1MultiDocument** は、**Print**メソッドおよび **PrintDialog** メソッドのオーバーロードを使用して印刷することができます。詳細については、C1MultiDocument ファイルの印刷を参照してください。

# C1MultiDocument の制限

**C1MultiDocument**コンポーネントの主な目的は、このコンポーネントがなければ、メモリ制限により作成／エクスポート／印刷が不可能だった大きなドキュメントの処理を可能にすることです。ドキュメントのサイズと数に制限はありません。しかし、(デフォルトの)メモリではなくディスクストレージを使用しなければならない場合があります。詳細については、**SetStorage**メ


Copyright © GrapeCity, Inc. All rights reserved.

ソッドを参照してください。

マルチドキュメントには一部制限があります。それが大きな問題になるかどうかはアプリケーションによります。中に含まれている **C1PrintDocument** のオブジェクトモデルにアクセスすることができません。言い換えれば、次のコードは実行可能ですが、

**Visual Basic コードの書き方**

```
Visual Basic
Dim mdoc As New C1MultiDocument()
mdoc.Items.Add(C1PrintDocument.FromFile("file1.c1dx"))
mdoc.Items.Add(C1PrintDocument.FromFile("file2.c1dx"))
mdoc.Items.Add(C1PrintDocument.FromFile("file3.c1dx"))
```

**C# コードの書き方**

```
C#
C1MultiDocument mdoc = new C1MultiDocument();
mdoc.Items.Add(C1PrintDocument.FromFile("file1.c1dx"));
mdoc.Items.Add(C1PrintDocument.FromFile("file2.c1dx"));
mdoc.Items.Add(C1PrintDocument.FromFile("file3.c1dx"));
```

後から、次のようなコードを追加することはできません。

**Visual Basic コードの書き方**

```
Visual Basic
Dim doc As C1PrintDocument = mdoc.Items(1)
```

**C# コードの書き方**

```
C#
C1PrintDocument doc = mdoc.Items[1];
```

アプリケーションでこの制限が問題にならない場合は、メモリ問題がなくても、**C1MultiDocument** コンポーネントを使用して、アプリケーションを「モジュラー化」することさえ可能です。

# C1MultiDocument ファイルの作成とプレビュー

**C1MultiDocumentItemCollection**に項目を追加するには、**Add** メソッドを使用します。ファイルを**C1MultiDocument** コンポーネントにロードするには、**Load** メソッドを使用します。ファイルを削除するには、**Clear**メソッドを使用します。このメソッドは、それまで **C1MultiDocument** コンポーネントにロードされていたファイルをすべてクリアします。

**C1MultiDocumentItemCollection** に項目を追加するには、**Add** メソッドを使用します。以下の手順を実行します。

1. デザインビューで、フォームをダブルクリックして、コードエディタを開きます。
2. 次のコードを **Load** イベントに追加します。

   **Visual Basic コードの書き方**

   ```
   Visual Basic
   Dim ppc As New C1PrintPreviewControl
   Controls.Add(ppc)
   ppc.Dock = DockStyle.Fill
   ```


Copyright © GrapeCity, Inc. All rights reserved.

```
Dim pdoc As New C1PrintDocument
Dim pdoc2 As New C1PrintDocument
Dim mdoc As New C1MultiDocument
pdoc.Body.Children.Add(New C1.C1Preview.RenderText("Hello!"))
pdoc2.Body.Children.Add(New C1.C1Preview.RenderText("World!"))
mdoc.Items.Add(pdoc)
mdoc.Items.Add(pdoc2)
ppc.Document = mdoc
mdoc.Generate()
```

**C# コードの書き方**

C#

```
C1PrintPreviewControl ppc = new C1PrintPreviewControl();
Controls.Add(ppc);
ppc.Dock = DockStyle.Fill;
C1PrintDocument pdoc = new C1PrintDocument();
C1PrintDocument pdoc2 = new C1PrintDocument();
C1MultiDocument mdoc = new C1MultiDocument();
pdoc.Body.Children.Add(new C1.C1Preview.RenderText("Hello!"));
pdoc2.Body.Children.Add(new C1.C1Preview.RenderText("World!"));
mdoc.Items.Add(pdoc);
mdoc.Items.Add(pdoc2);
ppc.Document = mdoc;
mdoc.Generate();
```

上記のコードは、**C1MultiDocument** コンポーネントに2つの **C1PrintDocument** をロードし、実行時に **C1PrintPreviewControl** でファイルを表示します。

# C1MultiDocument ファイルのエクスポート

**C1MultiDocument** は、**Export**メソッドのオーバーロードを使用して、ほとんどの形式にエクスポートすることができます。たとえば、次の例では、**C1MultiDocument** を PDF ファイルにエクスポートしています。ブール値の **True** は、進行状況ダイアログボックスを表示することを意味します。

**Visual Basic コードの書き方**

Visual Basic

```
Me.C1MultiDocument1.Export("C:\exportedfile.pdf", True)
```

**C# コードの書き方**

C#

```
this.c1MultiDocument1.Export(@"C:\exportedfile.pdf", true);
```

上記のコードをボタンの **Click** イベントハンドラに組み込むと、実行時にそのボタンをクリックしたときに、**C1MultiDocument** のコンテンツが **PDF** ファイルにエクスポートされます。

# C1MultiDocument ファイルの印刷

**C1MultiDocument** は、**Print** メソッドおよび **PrintDialog** メソッドのオーバーロードを使用して印刷することができます。た


Copyright © GrapeCity, Inc. All rights reserved.

とえば、次のコードは、**印刷**ダイアログボックスを開きます。

**Visual Basic コードの書き方**

```
Visual Basic
Me.C1MultiDocument1.PrintDialog()
```

**C# コードの書き方**

```
C#
this.c1MultiDocument1.PrintDialog();
```

上記のコードをボタンの **Click** イベントハンドラに組み込むと、実行時にそのボタンクリックしたときに、**印刷**ダイアログボックスが表示されるようになります。

# C1MultiDocument のアウトライン

**C1MultiDocument** にはアウトラインサポートが含まれています。**Outlines** プロパティからマルチドキュメント固有のアウトラインノードのコレクションを指定することができます。結果のアウトライン（プレビューのアウトラインなど）は、そのコレクション内のアウトラインノードと、中のドキュメント内のアウトラインノードの組み合わせとして構築されます。このアウトラインは、**MakeOutlines**() メソッドを使用してプログラムで構築することもできます。

マルチドキュメント自体の **Outlines** コレクションが最初に処理され、そのコレクションに由来するノードは、結果のアウトラインに含まれます。ノードが中の**C1MultiDocumentItem** の **OutlineNode** の値としても指定されている場合（2つのプロパティ参照が同一のオブジェクトを指している場合など）、その項目によって表されるドキュメントまたはレポートのアウトライン全体が結果のアウトラインに挿入されます。マルチドキュメント項目の **NestedOutlinesMode** プロパティの値に応じて、ドキュメントまたはレポートのアウトラインはアウトラインノード内でネストされるか、そのノードを置き換えます。最後に、マルチドキュメントの **Outlines** コレクションに含まれない項目によって表されるドキュメントおよびレポートのアウトラインは、結果のアウトラインに自動的に順番に追加されます。

アウトラインセットアップは、以下のプロパティとメソッドによって提供されます。

- **Outlines** プロパティ
- **MakeOutlines**メソッド
- **Outlines** プロパティ
- **OutlineNode** プロパティ


Copyright © GrapeCity, Inc. All rights reserved.