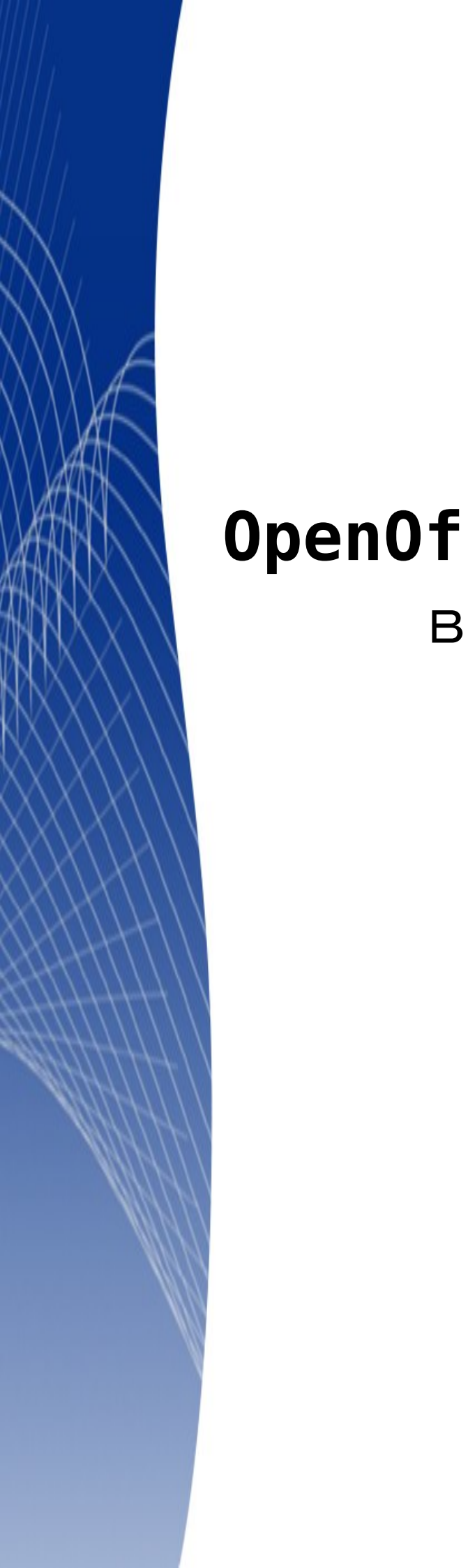

OpenOffice.org 3.1

Ｂａｓｉｃ ガイド

Copyright

なお、本ドキュメントのメンテナンスは、下記の場所で実施しています。

http://wiki.services.openoffice.org/wiki/JA/Documentation/BASIC_Guide

目次

BASIC プログラミングガイド

この章では、OpenOffice.org Basic を使用したプログラミングについて説明します。なお本ガイドの読者には、プログラミング言語に関する基本知識があることを想定しています。また、OpenOffice.org Basic プログラムを開発する際に参考となる各種のサンプルコードも紹介します。

このガイドでは、OpenOffice.org 管理に関する情報を次の章に分けています。最初の 3 つの章では、OpenOffice.org Basic について紹介します。

・OpenOffice.org Basic プログラミング用の言語

・実行時ライブラリ

・API について

これらの章では OpenOffice.org Basic の概要を説明しています。OpenOffice.org Basic プログラムを記述する場合は、必ず目を通しておく必要があります。これ以降の章は、各 OpenOffice.org API の個別の説明です。必要に応じて目を通すようにしてください。

・ドキュメントの使い方

・文書ドキュメント

・表計算ドキュメント

・図形描画とプレゼンテーション

・グラフ (図)

・データベース

・ダイアログ

・フォーム

OpenOffice.org Basic について

OpenOffice.org Basic のプログラミング言語は、OpenOffice.org 専用に開発されたもので、Office パッケージとの親和性に優れています。

その名前が示すように、OpenOffice.org Basic もいわゆる Basic プログラミング言語の一種です。そのため、過去に Basic 言語を使用した経験があり、特に Microsoft 社の Visual Basic や Visual Basic for Applications (VBA) によるプログラミングが行えるのであれば、すぐに OpenOffice.org Basic をマスターできるはずです。OpenOffice.org Basic には、Visual Basic と共通する部分が多くあります。

OpenOffice.org Basic のプログラミング言語を構成する要素は、大きく分けて次の 4 つに分類できます。

・OpenOffice.org Basic プログラミング用の言語: 変数宣言、ループ、関数などの基本言語構成を定義します。

・実行時ライブラリ: OpenOffice.org の基本的な機能を提供するもので、たとえば数字、文字列、日付、ファイルなどの編集がこれに該当します。

・OpenOffice.org API (Application Programming Interface): OpenOffice.org ドキュメントへのアクセスを許可し、ドキュメントの作成、変更、および印刷を許可します。

・ダイアログエディタ: ダイアログウィンドウを定義するためのもので、必要なコントロールの配置や、イベントハンドラの割り当てなどを行います。

OpenOffice.org Basic と VBA の互換性については、OpenOffice.org Basic のプログラミング言語だけでなく、実行時ライブラリも関係します。また OpenOffice.org API およびダイアログエディタは、VBA と互換性が ありません (これらのインターフェースまでも共通化すると、OpenOffice.org に用意された多くの機能が使用不可能となるためです)。

OpenOffice.org Basic の対象ユーザー

OpenOffice.org Basic を用いたアプリケーションは、OpenOffice.org が備えている標準的な機能では対応できない操作を実行する場合に使用されます。たとえば OpenOffice.org Basic を使うことで、ルーチンで実行するタスクを自動操作したり、データベースサーバーなど他のプログラムへのリンクを作成したり、また複雑な処理をスクリプトに記述し、ボタンをクリックするだけで実行させることもできます。

OpenOffice.org Basic を使えば OpenOffice.org の全機能にアクセスできるだけでなく、各種の関数を利用した操作、およびドキュメントの種類の変更や、ダイアログウィンドウをユーザー定義することもできます。

OpenOffice.org Basic の使用

OpenOffice.org Basic は、すべての OpenOffice.org プログラムから実行が可能で、特別なサポートプログラムなどを追加する必要はありません。また標準インストールを行なった場合でも、Basic マクロの作成に必要となる次の OpenOffice.org Basic 用コンポーネントがすべてインストールされます。

・統合開発環境 (integrated development environment: IDE): マクロの入力およびテストを行うためのエディタとして機能します。

・インタプリタ: OpenOffice.org Basic マクロの実行に必要です。

・各種の OpenOffice.org アプリケーションに対するインターフェース: Office ドキュメントに直接アクセスする際に必要となります。

詳細情報

このマニュアルで説明する OpenOffice.org API コンポーネントは、OpenOffice.org Basic のプログラマに必要とされる情報の重要度に応じて選択してあります。また、多くのコンポーネントでは、インターフェース機能の一部のみが説明されています。詳細については、『 API reference』を参照してください。

OpenOffice.org API については、『デベロッパ向けガイド』により詳しい説明がありますが、これらは基本的に Java および C++ でのプログラミングを念頭に記述されています。ただし、OpenOffice.org Basic にある程度習熟していれば、OpenOffice.org Basic および OpenOffice.org でのプログラミングで役立つ情報を『デベロッパ向けガイド』から入手できます。

OpenOffice.org Basic ではなく、Java または C++ を使用して直接プログラミングを行う場合は、このマニュアルの代わりに『OpenOffice.org デベロッパ向けガイド』を参照してください。ただし、Java または C++ による OpenOffice.org プログラミングは、OpenOffice.org Basic よりもさらに複雑なプログラミングを必要とします。

OpenOffice.org Basic プログラミング言語

OpenOffice.org Basic は、いわゆる Basic プログラミング言語の一種です。また OpenOffice.org Basic には、Microsoft Visual Basic for Applications および Microsoft Visual Basic と共通する部分が多くあります。これらの言語を扱った経験があれば、OpenOffice.org Basic は簡単にマスターできるでしょう。

また Java、C++、Delphi など、他のプログラミング言語のプログラマにとっても、OpenOffice.org Basic は簡単にマスターできるはずです。OpenOffice.org Basic は、完全な手続き型のプログラミング言語であり、従来使われていた `GoTo` や `GoSub` などを必要としません。

また OpenOffice.org Basic はオブジェクト指向型のプログラミング言語でもあり、外部オブジェクトライブラリを使用するためのインターフェースも用意されています。OpenOffice.org API のインターフェースの詳細については、後述します。

本章では、OpenOffice.org Basic プログラミング言語の主要コンポーネント、および OpenOffice.org Basic を用いたアプリケーションやライブラリの概要について説明します。

OpenOffice.org Basic 概要

OpenOffice.org Basic はインタプリタ型の言語です。C++ や Delphi とは異なり、OpenOffice.org Basic のコンパイラは、自動的に実行できる実行可能ファイルまたは自動抽出ファイルを作成しません。代わりに、OpenOffice.org の内部で OpenOffice.org Basic プログラムを実行します。これらのコードは、事前にチェックが行われてから、1 行ずつ実行されます。

プログラム行

Basic インタプリタには、コードを 1 行ずつ実行していくという点で、他のプログラミング言語と大きく異なる点があります。たとえばソースコード内に改行記号がある場合、Java、C++、Delphi などのプログラミング言語では、このような改行記号は無視されますが、Basic 言語の場合は、改行コードまでの 1 行が 1 つのプログラミングコードとして完結している必要があります。同様に、関数呼び出しや数値演算および、関数やループのヘッダ部なども、1 行の中に収まっている必要があります。

ただし、コードを記入するスペースが足りなかったり、極端に長い行になるような場合は、下線記号 (_) を末尾に追加することで、複数の行を 1 行と認識させることができます。次のサンプルコードでは、実質的に 1 行の数値演算行を、見かけ上 4 行に分割しています。

```
LongExpression = (Expression1 * Expression2) + _(Expression3 *
Expression4) + _ (Expression5 * Expression6) + _(Expression7 *
Expression8)
```

下線記号は、行の末尾に記入する必要があり、スペース記号やタブ記号を続けることはできず、そのような場合はエラーが発生します。

このような行の分割機能に加えて、OpenOffice.org Basic では、コロン記号を挿入することで、1 行を複数のセクションに分割することができます。この機能はコードの表示スペースを整える場合に有用です。たとえば次のコードは、

```
a = 1 a = a + 1 a = a + 1
```

次のように 1 行にまとめることができます。

```
a = 1  :  a = a + 1  :  a = a + 1
```

コメント

OpenOffice.org Basic のプログラムコード内には、通常の実行行以外にコメント行を記述することができ、プログラム各部の説明および、各種の参考情報などを記入しておくことができます。

OpenOffice.org Basic の場合、プログラムコード内へのコメント行の記述は、次の 2 通りの方法が可能です。

・アポストロフィー記号に続く部分は、すべてコメントと見なされます。

```
Dim A    ' This is a comment for variable A
```

・キーワード Rem に続く部分もコメントと見なされます。

```
Rem This comment is introduced by the keyword Rem.
```

通常これらのコメントは、該当行の末尾までがコメントの内容となります。OpenOffice.org Basic は、それ以降の行を再び通常の実行行として解釈します。複数行にわたるコメントを記述する場合は、各行ごとにコメント指定記号を付ける必要があります。

```
Dim B     ' This comment for variable B is relatively long
          ' and stretches over several lines. The
          ' comment character must therefore be repeated
          ' in each line.
```

マーカー

OpenOffice.org Basic のプログラム内には、数十から数千個のマーカー (変数、定数、関数などの名前) を設定できます。マーカーの名前は、次の命名規則に従う必要があります。

・マーカーに使える文字は、アルファベット、数字、下線記号 (_) だけです。

・マーカーの先頭は、アルファベッドか下線記号で始める必要があります。

・マーカーには、ä â î ß などの特殊文字は使用できません。

・マーカーの長さは、最大 255 文字です。

・大文字と小文字は区別されません。たとえば、OneTestVariable というマーカーは、onetestVariable や ONETESTVARIABLE と同じ変数を定義します。
ただし、この規則には例外が 1 つあり、UNO-API 定数については、大文字と小文字が区別されます。UNO の詳細については、「OpenOffice.org API の概要」を参照してください。

OpenOffice.org Basic と VBA では、マーカーの記述規則が異なります。たとえば、OpenOffice.org Basic では、Option Compatible を指定した場合のみ、マーカーに特殊文字が使えるようになっていますが、これは多言語プロジェクトを扱う際に問題が生じる可能性があるためです。

次に、使用できるマーカーと使用できないマーカーの例をいくつか示します。

```
Surname      ' Correct
Surname5     ' Correct (number 5 is not the first digit)
First Name   ' Incorrect (spaces are not permitted)
DéjàVu       ' Incorrect (letters such as é, à are not permitted)
5Surnames    ' Incorrect (the first character must not be a number)
First,Name   ' Incorrect (commas and full stops are not permitted)
```

変数の使用法

変数の暗黙的宣言

一般に Basic 言語は、簡易的な記法を許可するように設計されています。このような流れを受けて、OpenOffice.org Basic でも変数の使用法が簡易化されており、明示的な宣言をすることなく変数を使用できます。このためコード内に使われる変数名は、最初に記述された段階で実態として存在するようになります。既にどのような変数が存在するかにもよりますが、次のサンプルコードを記述するだけで、最大 3 つの変数が宣言されたことになります。

```
a = b + c
```

ただし、このような暗黙的な変数宣言は、タイプミスなどによる混乱を招く場合もあるため、推奨されるプログラミングスタイルではありません。たとえば、間違った変数名を入力しても、エラーメッセージは出力されません。インタプリタは、新しい変数名が宣言されたと判断して、初期値として 0 を割り当てて処理を継続します。このようなバグは、コード内の該当箇所を特定するのが困難です。

変数の明示的宣言

変数の暗黙的宣言に伴うエラーを防止する観点から、OpenOffice.org Basic には次のような機能が用意されています。

```
Option Explicit
```

このコードは、各モジュールのプログラム先頭行に記述する必要があります。この機能を使用することで、明示的に宣言されていない変数があった場合にエラーメッセージが表示されるようになります。また、この `Option Explicit` は、すべての Basic モジュールに記述する必要があります。

変数の明示的な宣言方法として、次のようなコマンドを記述するのが最も簡単です。

```
Dim MyVar
```

このサンプルコードでは、`MyVar` という名前の変数を宣言し、その変数型を `variant` としています。バリアント型の変数とは、文字列型、整数型、浮動小数点型、ブール型など、使用可能なすべての変数型データを格納できる変数のことです。ここでは、バリアント型変数のいくつかの例を紹介します。

```
MyVar = "Hello World"      ' Assignment of a string
MyVar = 1                  ' Assignment of a whole number
MyVar = 1.0                ' Assignment of a floating point number
MyVar = True               ' Assignment of a Boolean value
```

上記のようにして宣言された変数には、同一プログラム内でほかの変数型のデータを割り当てることもできます。ただし、バリアント型変数を使うと柔軟な操作が行えますが、変数は特定の変数型に固定して使用する方が賢明です。また、変数型の一致しないデータ操作が行われると、OpenOffice.org Basic はエラーメッセージを表示します。

特定の変数型に固定して変数宣言を行うには、次のようにします。

```
Dim MyVar As Integer   ' Declaration of a variable of the integer type
```

この変数は整数型として宣言され、整数型データを格納することができます。数値型の変数宣言は、次のようにしても行えます。

```
Dim MyVar%          ' Declaration of a variable of the integer type
```

Dim による変数宣言では、複数の変数宣言を一度に行えます。

```
Dim MyVar1, MyVar2
```

特定の変数型を指定して変数宣言する場合は、各変数に変数型を指定する必要があります。

```
Dim MyVar1 As Integer, MyVar2 As Integer
```

変数型を指定しないと、OpenOffice.org Basic はバリアント型として変数宣言を行います。たとえば、次のサンプルコードでは、MyVar1 はバリアント型、MyVar2 は整数型として宣言しています。

```
Dim MyVar1, MyVar2 As Integer
```

以降の節では、OpenOffice.org Basic で使用可能な変数型を示し、それらの使用法および宣言法を説明します。

文字列変数

文字列型は、数値型と共に、OpenOffice.org Basic で最も多用される変数型です。文字列とは、一連の文字を並べた形で構成されるデータです。コンピュータ内の処理において文字列は、個々の文字を示す固有の数値として記録されます。

ASCII コードから Unicode コードまで

文字コードセットとは、文字列を構成する各文字とそのコード番号の対応関係を示すもので、コンピュータによる画面への文字出力も、この文字コードセットに応じて処理されます。

ASCII 文字コードセット

ASCII 文字コードセットとは、数字、アルファベット、特殊記号に 1 バイトのコード番号を割り当てたものです。ASCII コードでは 0 から 127 までのコード番号がアルファベットおよびその他の記号 (ピリオド、カッコ、コンマなど) に割り当てられており、その中には画面およびプリンタ制御用の特殊コードなども定義されています。通常、コンピュータ間でテキストデータを交換する際には、この ASCII 文字コードセットが標準的なフォーマットとして利用されています。

ただし、この文字コードセットには、â、ä、î などの欧文特殊文字や、キリル文字などは含まれていません。

ANSI 文字セット

Microsoft 社の Windows 製品では、American National Standards Institute (ANSI) 文字コードセットが採用されていますが、この中には ASCII 文字コードセット内に存在しない文字も含めるよう拡張されています。

コードページ

ISO 8859 文字コードセットは、国際標準となるコードセットを取り決めたものです。この ISO 文字コードセットの最初の 128 文字は、ASCII 文字コードセットと同じものです。ISO 標準による新規文字コードセット (コードページ) の登場により、さまざまな言語をより正しく表示できるようになりました。ただし、この場合、特定の文字コードが、使用する言語ごとに異なる文字を示すという欠点があります。

Unicode

Unicode は、文字コードを 4 バイトに拡張して複数の文字コードセットを組み合わせることで、可能な限り広範な言語に対応した標準コードを規定しています。現在 Unicode の Version 2.0 は、OpenOffice.org および OpenOffice.org Basic をはじめとする多数のソフトウェアでサポートされています。

文字列変数

OpenOffice.org Basic では、文字列変数を Unicode で保存します。1 つの文字列変数には、最大 65535 文字を格納できます。OpenOffice.org Basic はこれらの個々の文字を、該当する Unicode 値として内部に格納しています。そのときに使用する作業用メモリ量は、扱う文字列の長さに依存します。

文字列変数の宣言は、次の形式で行います。

```
Dim Variable As String
```

この変数宣言は、次のように記述することもできます。

```
Dim Variable$
```

VBA アプリケーションを移植するときは、OpenOffice.org Basic での最大文字数 (65535 文字) を超過しないよう確認する必要があります。

文字列の明示的指定

文字列変数に対して明示的に文字列を代入する際には、文字列を引用符 (") で囲みます。

```
Dim MyString As String
MyString = ” This is a test”
```

文字列を 2 行にわたって記述する場合は、1 行目の最後にプラス記号を付けます。

```
Dim MyString As String
MyString =   ”This string is so long that it ” + _
             ”has been split over two lines.”
```

文字列中に引用符 (") そのものを含めたい場合は、該当する位置に引用符を 2 個続けて記述します。

```
Dim MyString As String
MyString = "a ""-quotation mark."    ' produces a "-quotation mark
```

数値変数

OpenOffice.org Basic は、数値処理用に 5 つの基本的な型をサポートしています。

・整数

・ロング整数型

・単精度変数

・倍精度型

・通貨変数

整数変数

整数変数には、**-32768** から **32767** までの整数を格納することができます。1 つの整数変数が消費するメモリ量は、2 バイトです。整数変数の型宣言子は % です。整数変数には、非常に高速に計算できるというメリットがあり、ループカウンタ用の変数として適しています。整数変数に浮動小数点型の数値を代入すると、小数点以下を丸めた整数値が収められます。

整数変数の宣言は、次の形式で行います。

```
Dim Variable As Integer
Dim Variable%
```

ロング整数変数

ロング整数変数には –**2147483648** から **2147483647** までの整数を格納することができます。1 つのロング整数変数が消費するメモリ量は、4 バイトです。ロング整数変数の型宣言子は & です。ロング整数変数には、非常に高速に計算できるというメリットがあり、ループカウンタ用の変数として適しています。ロング整数変数に浮動小数点型の数値を代入すると、小数点以下を丸めた整数値が収められます。

ロング整数変数の宣言は、次の形式で行います。

```
Dim Variable as Long
Dim Variable&
```

単精度変数

単精度変数には **3.402823 × 10**^{38} から **1.401298 × 10**$^{-45}$ までの正および負の浮動点小数を収めることができます。1 つの単精度変数が消費するメモリ量は、4 バイトです。単精度変数の型宣言子は ！です。

従来このような単精度変数は、より精度の高い倍精度変数に対して、必要な計算処理時間を短縮するために使われてきました。ただし近年は計算時間がさほど大きな要素にならなくなってきたため、単精度変数の必要性も低くなっています。

単精度変数の宣言は、次の形式で行います。

```
Dim Variable as Single
Dim Variable!
```

倍精度変数

倍精度変数には **1.79769313486232 × 10**^{308} から **4.94065645841247 × 10**$^{-324}$ までの正および負の浮動点小数を収めることができます。倍精度変数が消費するメモリ量は、最大で 8 バイトです。倍精度変数は、高い精度を必要とする計算処理に適しています。倍精度変数の型宣言子は **#** です。

倍精度変数の宣言は、次の形式で行います。

```
Dim Variable As Double
Dim Variable#
```

通貨変数

通貨変数では、ほかの変数型と異なる方式で値が格納されます。この場合の数値は、小数部 4 桁の固定小数点方式として扱われます。整数部は最大 15 桁までが格納できます。通貨変数には **-922337203685477.5808** から **+922337203685477.5807** までの数値を収めることができ、1 つの通貨変数が消費するメモリ量は、8 バイトです。通貨変数の型宣言子は @ です。

主として通貨変数は、金銭計算などを行う際に、浮動小数点に起因する丸め誤差を避ける場合に使用されます。

通貨変数の宣言は、次の形式で行います。

```
Dim Variable As Currency
Dim Variable@
```

浮動小数点数値

単精度、倍精度、および通貨の各型は、まとめて浮動小数点、または浮動小数点数型と呼ばれることがよくあります。浮動小数点数は、さまざまな長さの小数部分を持つ数値を格納でき、そのためにこのような名前が付いています。小数点が数値の中を「浮動」できるように見えます。

float 型の変数を宣言できます。変数の実際の型 (single、long、currency) は、値が変数に代入される時点で決まります。

```
Dim A As Float
A = 1210.126
```

数値の明示的指定

数値の表示方式には、小数点表示や指数表示などの各種の方式があり、その他に 10 進数以外の表示を行うことも可能です。OpenOffice.org Basic で扱う数値には、次の規則が適用されます。

整数

数値としての扱い方が最も簡単なのが整数です。なおソースコード内にこれら数値を記述する際には、千単位の桁区切りを示すコンマを付ける必要はありません。

```
Dim A As Integer
Dim B As Float

A = 1210
B = 2438
```

数値の前にはプラス記号 (+) またはマイナス記号 (-) を指定できます (これらの記号と数値の間にスペース記号を付けても付けなくても同じ)。

```
Dim A As Integer
Dim B As Float

A = + 121
B = - 243
```

小数

小数を入力するときは、ピリオド (.) を小数点として使用します。これは、ほかの記号を小数点として使用している地域においても、ソースコードをそのまま使用するための規則です。

```
Dim A As Integer
Dim B As Integer
Dim C As Float

A = 1223.53      ' is rounded
B = - 23446.46   ' is rounded
C = + 3532.76323
```

小数値の前にはプラス記号 (+) またはマイナス記号 (-) を指定できます (この場合も記号と数値の間にスペース記号を付けても付けなくても同じ)。

整数変数に 10 進数が代入される場合、OpenOffice.org Basic は数字を切り上げまたは切り捨てます。

指数表示

OpenOffice.org Basic では、数値を指数 (累乗、べき乗) 形式で指定することができ、たとえば 1.5×10^{-10} (0.00000000015) という値は `1.5e-10` と表記できます。この場合の「e」は大文字でも小文字でもよく、先頭にプラス記号 (+) を付けても付けなくても同じです。

ここでは、正しい指数表記と正しくない指数表記の例をいくつか見てみます。

```
Dim A As Double

A = 1.43E2    ' Correct
A = + 1.43E2  ' Correct (space between plus and basic number)
A = - 1.43E2  ' Correct (space between minus and basic number)
A = 1.43E-2   ' Correct (negative exponent)
A = 1.43E -2  ' Incorrect (spaces not permitted within the number)
A = 1,43E-2   ' Incorrect (commas not permitted as decimal points)
A = 1.43E2.2  ' Incorrect (exponent must be a whole number)
```

上記の無効な例のうち 1 番目と 3 番目のものでは、エラーメッセージは表示されませんが、結果として間違った値が返されるので注意が必要です。配列は次のようにして宣言します。

```
A = 1.43E -2
```

これは 1.43 マイナス 2 として解釈され、値 -0.57 に対応します。しかし、意図した値は 1.43×10^{-2} (0.0143 に対応) です。次のような値があります。

```
A = 1.43E2.2
```

OpenOffice.org Basic は指数部の数値の小数点以下を無視するので、この数式は次のものとして処理されます。

```
A = 1.43E2
```

16 進数

16 進数による数値表記では、2 桁の数値が 1 バイトの値と正確に対応します。この表記法は、コンピュータ内部で直接処理されるデータ値を扱う際に有用です。数値の 16 進数表記では、0 から 9 までの数字と A から F までのアルファベットの組合せで個々の値を表現します。この場合の A は 10 進数の 10 に該当し、F は 10 進数の 15 に該当します。
OpenOffice.org Basic では、&H を数値の先頭に付けることで、16 進数であることを指定します。

```
Dim A As Long
A = &HFF ' Hexadecimal value FF, corresponds to the decimal value 255
A = &H10 ' Hexadecimal value 10, corresponds to the decimal value 16
```

8 進数

OpenOffice.org Basic では 8 進数も使用できます。使用できる数値は 0 ～ 7 です。整数の前に &O を付けて表す必要があります。

```
Dim A As Long
A = &O77 ' Octal value 77, corresponds to the decimal value 63
A = &O10 ' Octal value 10, corresponds to the decimal value 8
```

ブール型変数

ブール型変数には、True (真) および False (偽) という 2 つの値の一方を収めることができます。この変数型は、2 つの状態の一方のみを取る情報を扱う場合に適しています。ブール型の値は、内部的には 2 バイトの整数値として格納され、0 が False に該当し、その他すべての値は True として扱われます。ブール型変数には、型宣言子は存在しません。この場合は、As Boolean による型宣言のみが行えます。

ブール型変数の宣言は、次の形式で行います。

```
Dim Variable As Boolean
```

日付および時刻変数

日付変数には、日付と時刻を示す値のみを収めることができます。OpenOffice.org Basic では日付変数の値を格納する際に、時刻間の比較や計算処理を行えるよう、これらの値を内部形式に変換します。日付変数には、型宣言子は存在しません。この場合は、`As Date` による型宣言のみが行えます。

日付変数の宣言は、次の形式で行います。

```
Dim Variable As Date
```

配列

OpenOffice.org Basic では、単純型変数 (スカラー) の他に、配列 (データフィールド) もサポートしています。1 つのデータフィールド内には複数の値を格納することが可能で、これらの各要素に対してはインデックス指定によりアクセスします。

1 次元配列

配列の宣言法は、通常の変数と基本的には同じです。ただし、配列名に続けて、配列の要素数をかっこで囲んで指定する点が異なります。配列は次のようにして宣言します。

```
Dim MyArray(3)
```

この場合の配列は、MyArray(0)、MyArray(1)、MyArray(2)、MyArray(3) という 4 つの要素を持つ、バリアント型変数として宣言されます。

また配列を宣言する際には、特定の変数型を指定することも可能です。たとえば次の例では 4 つの整数変数をとるよう配列を宣言しています。

```
Dim MyInteger(3) As Integer
```

これまでに説明した配列宣言の例では、インデックスの開始値として、標準値である 0 が使われています。このようなインデックスの開始および終了値は、配列 (データフィールド) の宣言時に指定することができます。次の例は 6 つの整数値を取るデータフィールドを作成するものですが、宣言をする際にインデックス範囲を 5 から 10 とするよう指定しています。

```
Dim MyInteger(5 To 10)
```

インデックス値には、正の値以外を使うことも可能です。次の例では、インデックス範囲に負の値を指定していますが、これも有効な宣言として処理されます。

```
Dim MyInteger(-10 To -5)
```

この場合、データフィールド (配列) のインデックス値は -10 から -5 までの整数値を取り、合計 6 つの要素を持つことになります。

データフィールド (配列) のインデックスについては、次の 3 つの制限があります。

・インデックスとして使用可能な最小値は -32768 まで。

・インデックスとして使用可能な最大値は 32767 まで。

・要素数 (データフィールドの次元) の最大値は 16368 まで。

VBA のデータフィールドでは、これ以外の制限が課されている場合もあります。次元ごとに取りうる最大の要素数についても、同様の制限が当てはまります。実際に有効とされる値については、VBA の関連マニュアルを参照してください。

インデックスの開始値に関する設定

通常、データフィールド (配列) のインデックスには、0 から始まる値が割り当てられます。この開始値については、次のように指定することにより、すべてのデータフィールド宣言において 1 とするよう変更できます。

```
Option Base 1
```

この変更をモジュール内のすべての配列宣言に対して適用させるには、モジュールのヘッダ部で指定する必要があります。ただしこの変更は、OpenOffice.org API で指定する UNO シーケンスに対しては無効で、これらのインデックスは常に 0 から始められます。このため Option Base 1 を指定すると、インデックスの開始値が混在する危険性があります。

`Option Base 1` による設定は、インデックスの開始値を変更するだけであり、配列の要素数には影響しません。たとえば、次のサンプルコードのように配列を宣言したとします。

```
Option Base 1
' ...
Dim MyInteger(3)
```

この場合、`MyInteger(1)`、`MyInteger(2)`、`MyInteger(3)`、`MyInteger(4)` という 4 つの要素を持つ配列が作成されます。

VBA の場合とは異なり、OpenOffice.org Basic での `Option Base 1` による設定は、配列の要素数には影響しません。OpenOffice.org Basic の場合、この設定は単にインデックスの開始値を変更するだけです。たとえば `MyInteger(3)` と宣言すると、VBA では 1 から 3 のインデックス値を取る 3 つの要素が確保されますが、OpenOffice.org Basic では 1 から 4 のインデックス値を取る 4 つの要素が確保されます。`Option Compatible` を使用すると、OpenOffice.org Basic は、VBA と同様に動作します。

多次元データフィールド

OpenOffice.org Basic では 1 次元データフィールド (配列) の他に、多次元データフィールドもサポートしています。個々の次元の指定は、コンマ記号で区切ります。次のサンプルコードは、これらの使用例です。

```
Dim MyIntArray(5, 5)
```

ここでは 2 次元の配列を作成し、個々の次元に 6 つの要素 (インデックス値の 0 から 5) を確保しています。この配列全体としては 6 × 6 = 36 個の値を格納することができます。

OpenOffice.org Basic では、数百次元にも及ぶ多次元配列を宣言することも可能ですが、実際には使用可能なメモリ量により、次元数に制限が課されることになります。

データフィールドのサイズの動的変更

これまでに説明した例では、特定のサイズを持つデータフィールド (配列) を扱ってきました。このような配列以外にも、データフィールドのサイズを動的に変更させることが可能です。たとえば、A という文字で開始するテキストで、すべての単語を含む配列を定義することができます。これらの単語の数は最初はわからないため、後でフィールドの上限を変更する必要があります。このような配列を作成するには、OpenOffice.org Basic では次のように宣言します。

```
ReDim MyArray(10)
```

VBA の場合は `Dim MyArray()` による動的配列のサイズ変更のみが行えますが、OpenOffice.org Basic の場合は `ReDim` により動的および静的配列のサイズを変更できます。

次のサンプルコードでは、最初に作成した配列のサイズを何度か変更して、それぞれ 11 個および 21 個の値を格納できるようにします。

```
Dim MyArray(4) As Integer ' Declaration with five elements
' ...
ReDim MyArray(10) As Integer ' Increase to 11 elements
' ...
ReDim MyArray(20) As Integer ' Increase to 21 elements
```

配列のサイズを変更する際には、これまでの節で説明したすべてのオプションを指定できます。これには、多次元データフィールド化するための指定や、インデックスの開始および終了値の指定などが該当します。なお、データフィールドのサイズを変更すると、格納していたすべてのデータが消失されます。変更前の値を保持させるには、`Preserve` コマンドを使用します。

```
Dim MyArray(10) As Integer ' Defining the initial
' dimensions
' ...
ReDim Preserve MyArray(20) As Integer ' Increase in
' data field, while
' retaining content
```

`Preserve` コマンドを使用する場合は、配列の次元数および変数型が、サイズ変更の前後で同じになっている必要があります。

VBA で `Preserve` コマンドを使用すると、データフィールドの最終次元の上限値だけしか変更できませんが、OpenOffice.org Basic では他の次元も変更できます。

`ReDim` と `Preserve` を併用する場合、データフィールドのデータ型はオリジナルのものから変更することはできません。

変数の有効範囲と有効期限

OpenOffice.org Basic で確保される個々の変数は、プログラムのすべての領域で使用可能というわけではなく、それぞれ有効となる範囲と期限があります。変数が確保され続ける期限および利用可能なプログラム範囲は、各変数の種類と宣言された位置に依存します。

局所変数

関数や手続きの内部で宣言された変数は、局所変数と呼ばれます。

```
Sub Test
  Dim MyInteger As Integer
  ' ...
End Sub
```

このような局所変数は、該当する関数や手続きが実行されている間は確保され続けますが、実行が終了した段階で消去されます。このため関数を呼び出す際には、以前の呼び出し時に代入された値などを利用することはできません。

このような値を保持しておくには、変数を `Static` として定義しておく必要があります。

```
Sub Test
  Static MyInteger As Integer
  ' ...
End Sub
```

VBA とは異なり、OpenOffice.org Basic の局所変数には、モジュールヘッダにある大域変数やプライベート変数と同じ名前を付けることはできません。このため、VBA アプリケーションを OpenOffice.org Basic に移植する際には、重複した変数名を変更する必要があります。

パブリックドメイン変数

パブリックドメイン変数は、モジュールのヘッダセクションで `Dim` キーワード を使って定義します。この変数は、ライブラリ内のすべてのモジュールで利用可能となります。

モジュール A:

```
Dim A As Integer
Sub Test
  Flip
  Flop
End Sub

Sub Flip
 A = A + 1
End Sub
```

モジュール B:

```
Sub Flop
  A = A - 1
End Sub
```

変数 A の値は、関数 Test では直接変更されませんが、間接的に関数 Flip で 1 増やされ、関数 Flop で 1 減らされています。この両者の関数による変更は、大域的 (グローバル) に実施されています。

パブリックドメイン変数の宣言用キーワードには、Dim の代わりに Public も使用できます。

```
Public A As Integer
```

パブリックドメイン変数は、該当するマクロが実行されている間は確保され続けますが、実行が終了した段階で消去されます。

大域変数

大域変数はパブリック変数と同等の機能を担っていますが、該当するマクロの実行終了後もその値が確保され続ける点が異なります。大域変数は、モジュールのヘッダセクションでキーワード Global を使って定義します。

```
Global A As Integer
```

プライベート変数

Private 変数は、定義されたモジュール内部でのみ有効となります。プライベート変数の定義には、キーワード Private を使用します。

```
Private MyInteger As Integer
```

複数のモジュールで同じ名前の Private 変数が使われている場合、OpenOffice.org Basic では、これらの変数はそれぞれ個別の変数として確保されます。次のサンプルコードでは、モジュール A およびモジュール B の両方で、C という名前の Private 変数を確保します。関数 Test は、最初にモジュール A の Private 変数を設定し、次にモジュール B の Private 変数を設定します。

モジュール A:

```
Private C As Integer

Sub Test
  SetModuleA      ' Sets the variable C from module A
  SetModuleB      ' Sets the variable C from module B
  ShowVarA        ' Shows the variable C from module A (= 10)
  ShowVarB        ' Shows the variable C from module B (= 20)
End Sub

Sub SetmoduleeA
  C = 10
End Sub

Sub ShowVarA
  MsgBox C        ' Shows the variable C from module A.
End Sub
```

モジュール B:

```
Private C As Integer

Sub SetModuleB
  C = 20
End Sub

Sub ShowVarB
  MsgBox C        ' Shows the variable C from module B.
End Sub
```

定数

定数とはプログラムで使用できますが、変更できない値です。

定数の定義

OpenOffice.org Basic で定数を宣言するには、キーワード `Const` を使用します。

```
Const A = 10
```

必要であれば、宣言時にデータ型を指定しておくこともできます。

```
Const B As Double = 10
```

定数の範囲

定数の範囲は変数と同じです (「変数の有効範囲と寿命」参照)。ただし、構文が若干異なります。モジュールヘッダの `Const` 定義は、そのモジュールのコードに対して使用できます。定義を他のモジュールでも使用可能にするには、キーワード `Public` を追加します。

```
Public Const one As Integer = 1
```

演算子

OpenOffice.org Basic には、一般的な算術、論理、比較用の演算子が用意されています。

算術演算子

算術演算子は数値を対象とした処理を行うものですが、+ 演算子は文字列を結合させる場合にも使用します。

+	数値および日付の値の加算 (足し算)、および文字列の結合。
-	数値および日付の値の減算 (引き算)。
*	数値の乗算 (掛け算)。
/	数値の除算 (割り算)。
\	数値の整数による除算 (結果を丸める)。
^	数値の指数演算 (累乗、べき乗)。
MOD	整数除算を実行した際の余りの値。

論理演算子

論理演算子は、ブール演算を実行するためのものです。このタイプの演算子をブール型のデータに対して適用すると、該当する論理演算を行った結果が返されます。また、整数およびロング整数のデータに対して適用すると、ビット単位の演算結果が返されます。

AND	論理積 (And) 演算。
OR	論理和 (Or) 演算。
XOR	排他的論理和 (Xor) 演算。
NOT	論理否定 (Not) 演算。
EQV	等価性検査 (両方の部分が `True` または `False`)。
IMP	論理包含 (Imp) 演算 (最初の値が True であれば次の値も True かを判定)。

比較演算子

比較演算子は、基本的なすべてのタイプの変数 (数値、日付、文字列、ブール型) に対して適用できます。

=	数値、日付、文字列のデータについて、両者が等しいかを判定。
<>	数値、日付、文字列のデータについて、両者が等しくないかを判定。
>	数値、日付、文字列のデータについて、左辺が右辺より大きいかを判定。
>=	数値、日付、文字列のデータについて、左辺が右辺より大きいまたは等しいかを判定。
<	数値、日付、文字列のデータについて、左辺が右辺より小さいかを判定。
<=	数値、日付、文字列のデータについて、左辺が右辺より小さいまたは等しいかを判定。

OpenOffice.org Basic は、VBA の `Like` 比較演算子をサポートしていません。

分岐処理

指定した条件に応じて、プログラム内の特定のコードブロックのみを実行させたい場合などは、分岐ステートメントを使用します。

If...Then...Else

使用頻度の高い分岐ステートメントの 1 つとして If ステートメントがあり、これは次のように使用します。

```
If A > 3 Then
  B = 2
End If
```

このサンプルコードで B = 2 の行が実行されるのは、変数 A の値が 3 より大きい場合だけです。If ステートメントのバリエーションとして、If/Else 句があります。

```
If A > 3 Then
  B = 2
Else
  B = 0
End If
```

このサンプルコードで変数 B に代入される値は、変数 A が 3 以上の場合は 2 となり、それ以外の場合 B に代入される値は、0 となります。

If ステートメントをカスケード化して、次のようなより複雑な条件分岐を行わせることもできます。

```
If A = 0 Then
  B = 0
ElseIf A < 3 Then
  B = 1
Else
  B = 2
End If
```

このサンプルコードでは、変数 A の値が 0 であれば、変数 B には 0 が代入されます。変数 A の値が 3 よりも小さければ (ただし 0 とは等しくない)、変数 B には 1 が代入されます。これら以外の場合 (つまり変数 A の値が 3 以上の場合)、変数 B には 2 が代入されます。

Select...Case

Select...Case ステートメントは、If ステートメントのカスケード化と同等の機能を果たすもので、複数の条件に対する分岐を行う際に使用します。

```
Select Case DayOfWeek
  Case 1:
    NameOfWeekday = "Sunday"
  Case 2:
    NameOfWeekday = "Monday"
  Case 3:
    NameOfWeekday = "Tuesday"
  Case 4:
    NameOfWeekday = "Wednesday"
  Case 5:
    NameOfWeekday = "Thursday"
  Case 6:
    NameOfWeekday = "Friday"
  Case 7:
    NameOfWeekday = "Saturday"
End Select
```

このサンプルコードでは、変数 DayOfWeek の値が 1 であれば Sunday 、2 であれば Monday というように、各曜日の名前を番号で識別させています。

Select コマンドによる条件判定は、単純な 1 対 1 に限定されるものではなく、Case による個々の分岐指定部には比較演算子を使ったり、複数の条件式をリスト化して一括指定することが可能です。次のサンプルコードでは、特に多用される条件判定の例を示します。

```
Select Case Var
  Case 1 To 5
    ' ... Var is between the numbers 1 and 5
  Case 6, 7, 8
    ' ... Var is 6, 7 or 8
  Case Var > 8 And Var < 11
    ' ... Var is greater than 8 and less than 11
  Case Else
    ' ... all other instances
End Select
```

ループ

ループは、特定のコードブロックを指定した回数繰り返し実行させる場合などに使用します。またループの実行回数は、不特定値とすることも可能です。

For...Next

For...Next ループは、特定回数の繰り返し実行を行うためのものです。この場合の繰り返し回数は、ループカウンタを使って指定します。このサンプルコードでは、変数 I はループカウンタで、初期値は 1 です。このカウンタの値は、ループを 1 回実行するごとに 1 ずつ増加されます。最終的に変数 I の値が 10 に等しくなった段階で、ループは終了します。

```
Dim I
For I = 1 To 10
  ' ...  Inner part of loop
Next I
```

ループカウンタの増分値を 1 以外にする場合は、次のように Step を使用します。

```
Dim I
For I = 1 To 10 Step 0.5
  ' ... Inner part of loop
Next I
```

上記のサンプルコードでは、ループを 1 回実行するごとにカウンタの値は 0.5 ずつ増加されるため、ループは最終的に 19 回実行されることになります。

ループの増分値には、負の値を指定することもできます。

```
Dim I
For I = 10 To 1 Step -1
  ' ... Inner part of loop
Next I
```

上記のサンプルコードではカウンタの初期値を 10 として、ループを 1 回実行するごとにカウンタの値を 1 ずつ減算させ、最終的に 1 となるまでループを実行しています。

Exit For ステートメントを使用すると、For ループを強制的に終了させることができます。次のサンプルコードでは、ループの 5 巡目で強制的に終了します。

```
Dim I
For I = 1 To 10
  If I = 5 Then
    Exit For
  End If
  ' ... Inner part of loop
Next I
```

For Each

VBA の For Each...Next ループバリアントは OpenOffice.org Basic でサポートされています。For Each ループは For...Next ループのように明示的なカウンタは使用しません。For Each ループは、「n 回繰り返す」ではなく、「このセット内のすべてに対して実行する」ということです。たとえば、以下の通りです。

```
Const d1 = 2
Const d2 = 3
Const d3 = 2
Dim a(d1, d2, d3)
For Each i In a()
    ' ... Inner part of loop
Next i
```

Do...Loop

Do...Loop は、特定回数のループを行うものではありません。Do...Loop によるループ処理は、特定の条件が満たされるまで繰り返し実行されます。Do...Loop には、4 つのバージョンがあります。最初の 2 つのサンプルコードでは、ループ内のコードがまったく実行されない可能性があります (「0 回実行」というロジック)。その後のサンプルコードでは、ループ内のコードが最低 1 回実行されます。(次のサンプルコードで、A > 10 は任意の条件を示します)。

1. Do While...Loop バージョン

```
Do While A > 10
   ' ... loop body
Loop
```

ここではすべてのパスの前に、While の後の条件が、**true** であるかどうかが確認され、その場合のみループが実行されます。

2. Do Until...Loop バージョン

```
Do Until A > 10
  ' ... loop body
Loop
```

ここでは、Until の後の条件が **false** であると評価されれば、ループが実行されます。

3. Do...Loop While バージョン

```
Do
  ' ... loop body
Loop While A > 10
```

ここでは最初のループパスの後の条件のみが確認され、While の後の条件が、**false** であると評価されると終了します。

4. Do...Loop Until バージョン

```
Do
  ' ... loop body
Loop Until A > 10
```

ここでも最初のパスの後の条件が確認されますが、Until の後の条件が **true** であると評価されると終了します。

For...Next ループと同様に、Do...Loop にも強制終了用のコマンドが用意されています。この場合は Exit Do コマンドにより、ループ内の任意の位置で強制終了が行えます。

```
Do
  If A = 4 Then
    Exit Do
  End If
  ' ... loop body
Loop While A > 10
```

プログラミングの例: 埋め込みループを使用したソート

ループの用途としては、リストの検索、値の取得、複雑な数値計算など、様々な処理で利用されています。次のサンプルコードは、2 つのループ処理を用いてリストを名前でソートするアルゴリズムです。

```
Sub Sort
  Dim Entry(1 To 10) As String
  Dim Count As Integer
  Dim Count2 As Integer
  Dim Temp As String

  Entry(1) = "Patty"
  Entry(2) = "Kurt"
  Entry(3) = "Thomas"
  Entry(4) = "Michael"
  Entry(5) = "David"
  Entry(6) = "Cathy"
  Entry(7) = "Susie"
  Entry(8) = "Edward"
  Entry(9) = "Christine"
  Entry(10) = "Jerry"

  For Count = 1 To 9
    For Count2 = Count + 1 To 10
      If Entry(Count) > Entry(Count2) Then
        Temp = Entry(Count)
        Entry(Count) = Entry(Count2)
        Entry(Count2) = Temp
      End If
    Next Count2
  Next Count

  For Count = 1 To 10
    Print Entry(Count)
  Next Count

End Sub
```

ここでは 2 つの値を 1 組にして順序の入れ替え作業を行い、最終的に昇順で並ぶまでこの作業を繰り返しています。その際に個々の変数値は 1 つずつ位置をずらしていきますが、この動きはちょうど泡が移動する様子に似ています。このような理由から、このタイプのアルゴリズムは一般に バブルソート と呼ばれています。

プロシージャと関数

プロシージャと関数こそは、プログラム構造の中心的な役割を担うものです。これらを利用することで、複雑なプログラムを個々のタスクごとにブロック化することができます。

プロシージャ

プロシージャは、プログラム内の特定の処理を実行するユニットのことですが、戻り値を返すことはありません。構文は次のようになります。

```
Sub Test
  ' ... here is the actual code of the procedure
End Sub
```

このサンプルコードでは Test という名前のプロシージャを定義していますが、このプロシージャによる処理は、プログラム内の任意の位置から実行させることができます。プロシージャを呼び出すには、次のようにプログラム内の該当行にプロシージャ名を記述するだけです。

関数

関数も、プロシージャと同様に、特定のプログラムブロックを 1 つのユニットとしてまとめたものです。プロシージャとの相違点として、関数は戻り値を返すという点があります。

```
Function Test
  ' ... here is the actual code of the function
  Test = 123
End Function
```

戻り値は、簡単な代入を使用して割り当てます。この代入は、関数の末尾で実行する必要はなく、関数内の任意の位置で行えます。

上記のようにして定義した関数は、プログラム内の任意の位置から呼び出すことができます。

```
Dim A
A = Test
```

このコードは、変数 A を宣言してから、関数 Test の戻り値をその中に代入しています。

戻り値として返す値は、関数内で何回も書き換えることが可能です。通常の変数への代入操作と同様に、次のサンプルでも実際に返される関数の戻り値は、最後に代入した値となります。

```
Function Test
  Test = 12
  ' ...
  Test = 123
End Function
```

このサンプルコードの場合、関数の戻り値は 123 となります。

戻り値の代入を行わなかった場合、その関数からは zero 値が返されます (数値関数の場合は 0 、文字列関数の場合は空白文字列)。

関数の戻り値のデータ型は、任意の種類を指定できます。この場合のデータ型宣言は、通常の変数宣言と同様の手順で行えます。

```
Function Test As Integer
  ' ... here is the actual code of the function
End Function
```

プロシージャと関数の強制終了

OpenOffice.org Basic では、Exit Sub および Exit Function コマンドを使用して、実行途中のプロシージャや関数を強制終了させることができ、エラーの処理などに便利です。これらのコマンドは、プロシージャや関数を強制終了させてから、これらを呼び出した行にまでプログラムの実行行を戻します。

次のサンプルコードでは、変数 ErrorOccured の値が True になった場合に、プロシージャを強制終了します。

```
Sub Test
  Dim ErrorOccured As Boolean
  ' ...
  If ErrorOccured Then
    Exit Sub
  End If
  ' ...
End Sub
```

パラメータの渡し方

関数やプロシージャには、1 つまたは複数のパラメータ (引数) を渡すことができます。パラメータは、関数やプロシージャの名前の後に、かっこで囲んで指定する必要があります。次の例では、整数値 A と文字列 B をパラメータとして予想するプロシージャを定義しています。

```
Sub Test (A As Integer, B As String)
  ' ...
End Sub
```

通常、OpenOffice.org Basic では、パラメータの 渡し方は参照 です。この場合、呼び出したプロシージャや関数が終了しても、これらの変数値に対して行われた変更はそのまま維持されます。

```
Sub Test
  Dim A As Integer
  A = 10
  ChangeValue(A)
  ' The parameter A now has the value 20
End Sub

Sub ChangeValue(TheValue As Integer)
  TheValue = 20
End Sub
```

この例では、値 A は Test 関数で定義されており、それがパラメータとして ChangeValue 関数に渡されます。その後、値は 20 に変更されて TheValue に渡されますが、これは関数が終了しても維持されます。

またパラメータに対する変更をオリジナルの変数の内容に反映させたくない場合は、パラメータを値渡しで与えることも可能です。パラメータを値で渡すよう指定するには、関数ヘッダで変数宣言の前にキーワード ByVal を指定する必要があります。

前の例では、関数 ChangeValue を置き換えた場合、上位の変数 A はこの変更による影響を受けません。関数 ChangeValue を呼び出した後も、変数 A の値は 10 のままです。

```
Sub ChangeValue(ByVal TheValue As Integer)
  TheValue = 20
End Sub
```

OpenOffice.org におけるプロシージャや関数へのパラメータの渡し方は、基本的に VBA と同じです。標準でパラメータは、参照渡しとして与えられます。パラメータを値として渡すには、ByVal キーワードを使用します。VBA では、キーワード ByRef を使用して、強制的に参照でパラメータを渡すこともできます。OpenOffice.org Basic では参照渡しが既に標準的な方法なので、OpenOffice.org Basic はこのキーワードを認識しますが無視します。

オプションパラメータ

関数やプロシージャを呼び出す際には、必要なパラメータをすべて指定する必要があります。

OpenOffice.org Basic では、パラメータをオプションとして定義できます。つまり、対応する値が呼び出しに含まれない場合、OpenOffice.org Basic は空のパラメータを渡します。次の例では、パラメータ A は必須ですが、パラメータ B はオプションです。

```
Sub Test(A As Integer, Optional B As Integer)
  ' ...
End Sub
```

IsMissing 関数を使うと、呼び出し時にパラメータが指定されているかをチェックできます。

```
Sub Test(A As Integer, Optional B As Integer)
  Dim B_Local As Integer
  ' Check whether B parameter is actually present
  If Not IsMissing (B) Then
    B_Local = B      ' B parameter present
  Else
    B_Local = 0      ' B parameter missing -> default value 0
  End If
  ' ... Start the actual function
End Sub
```

上記のサンプルコードでは、まずパラメータ B に値が指定されているかをチェックしてから、チェック結果に応じて、局所変数 B_Local に渡す値を変えています。対応するパラメータが存在しない場合、B_Local には、渡されたパラメータではなく、デフォルト値 (この例では 0) が代入されます。

VBA のキーワード ParamArray は、OpenOffice.org Basic ではサポートされていません。

再帰処理

再帰処理とは、関数やプロシージャが処理中に自分自身を呼び出すことで、特定の終了条件を満たすまで、このような処理を実行し続けます。たとえば再帰関数の場合、終了条件が満たされた段階で、戻り値を返します。

次のサンプルコードは再帰関数の使用例で、42、-42、3.14 の各数値の階乗を再帰処理で求めています。

```
Sub Main
  Msgbox CalculateFactorial(  42 )    ' Displays 1,40500611775288E+51
  Msgbox CalculateFactorial( -42 )    ' Displays "Invalid number for
factorial!"
  Msgbox CalculateFactorial( 3.14 )   ' Displays "Invalid number for
factorial!"
End Sub

Function CalculateFactorial( Number )
  If Number < 0 Or Number <> Int( Number ) Then
    CalculateFactorial = "Invalid number for factorial!"
  ElseIf Number = 0 Then
    CalculateFactorial = 1
  Else
    ' This is the recursive call:
    CalculateFactorial = Number * CalculateFactorial( Number - 1 )
  Endif
End Function
```

上記のサンプルコードでは、数値 42 の階乗を計算する際に、関数 CalculateFactorial を再帰的に呼び出し、基本条件 0! = 1 になった時点で処理を終了しています。

再帰回数のレベルは、ソフトウェアのプラットフォームごとに異なります。Windows の場合、再帰回数のレベルは 5800 です。Solaris および Linux の場合、再帰回数のレベルは、スタックサイズの計算結果を基に計算されます。

エラー処理

プログラミングを進める際に大きな問題となるのが、エラーに対する修正タスクです。OpenOffice.org Basic には、エラー処理用に機能が各種用意されています。

On Error 命令

On Error 命令は、エラー処理の中心となる機能です。

```
Sub Test
  On Error Goto ErrorHandler
  ' ... undertake task during which an error may occur
  Exit Sub
  ErrorHandler:
    ' ... individual code for error handling
End Sub
```

ここで On Error Goto ErrorHandler 行により、OpenOffice.org Basic が、エラー発生時にどのように動作するか指定しています。たとえばこの場合の Goto ErrorHandler は、現在の実行行を中断させて、ErrorHandler: で指定するコードブロックを OpenOffice.org Basic に実行させます。

Resume コマンド

Resume Next コマンドを使うと、エラーハンドラ用コードを実行した後で、エラー発生行の次の行にプログラム実行を戻して、処理を再開させることができます。

```
ErrorHandler:
  ' ... individual code for error handling
  Resume Next
```

また Resume Proceed コマンドを使用すると、エラーハンドラ用コードを実行した後のプログラムの実行開始位置を指定することができます。

```
ErrorHandler:
  ' ... individual code for error handling
  Resume Proceed

Proceed:
  ' ... the program continues here after the error
```

エラーが発生した際に、エラーメッセージを表示させずにプログラムを継続させるには、次のように記述します。

```
Sub Test
  On Error Resume Next
  ' ... perform task during which an error may occur
End Sub
```

この On Error Resume Next コマンドの作用範囲はプログラム全域に及ぶので、使用する際には注意が必要です。

エラーの関連情報の取得

エラー処理を行う場合、エラーの内容と発生箇所の情報が確認できると有用です。

・Err 変数には、発生したエラー番号が格納されます。

・Error$ 変数には、発生したエラーの内容が格納されます。

・Erl 変数には、エラーの発生した行番号が格納されます。

このメソッドは、次のような形式で使用します。

```
MsgBox "Error " & Err & ": " & Error$ & " (line : " & Erl & ")"
```

この場合は、メッセージウィンドウにエラーの内容が表示されます。

エラーの内容は、OpenOffice.org Basic では Err, Error$、Erl 変数に格納されますが、VBA では Err という名前のオブジェクトにまとめられます。

これらのエラー情報は、次に Resume または On Error コマンドを実行するまで維持され、これらを実行した段階でリセットされます。

VBA では、Err オブジェクトに対する Err.Clear メソッドにより、エラー情報をリセットします。
OpenOffice.org Basic では、On Error または Resume コマンドがこの機能を果たしています。

効率的なエラー処理のヒント

エラーハンドラを設定する On Error コマンドも、実行行を復帰させる Resume コマンドも、いわゆる Goto コマンドの一種です。

この種の実行行をジャンプさせるコマンドは、エラーの発生を予防する観点からも、コード内での多用を避けるべきです。

また On Error Resume Next コマンドは、エラーの関連情報をリセットしてしまうので、その使用に当たっては注意が必要です。

最善の方法は、プログラム内でエラー処理を行うブロックを一カ所にまとめておくことです。つまり、エラー処理ブロックをプログラム本体のコード部から分離しておき、エラー処理が終わっても実行行はエラー発生行にジャンプさせないようにします。

次のサンプルコードは、エラー処理の流れを示しています。

```
Sub Example
  ' Define error handler at the start of the function
  On Error Goto ErrorHandler
    ' ... Here is the actual program code
  On Error Goto 0           ' Deactivate error handling
    ' End of regular program implementation
  Exit Sub

 ' Start point of error handling
 ErrorHandler:
   ' Check whether error was expected
    If Err = ExpectedErrorNo Then
      ' ... Process error
    Else
      ' ... Warning of unexpected error
    End If
 On Error Goto 0            ' Deactivate error handling
End Sub
```

この手続きでは、一番最初にエラーハンドラを設定してから、プログラム本体のコードを記述しています。そしてプログラム本体のコードの末尾で `On Error Goto 0` によりエラーハンドラ機能を解除し、`Exit Sub` コマンドにより手続きの実行を終了させるようにしています (`End Sub` との違いに注意)。

このサンプルコードでは、あらかじめ想定されたエラーが発生したのかを、エラー番号をチェックすることで判定して (この例では判定用に `ExpectedErrorNo` という定数を使用) 、その結果に応じてエラー処理の内容を分岐させています。想定外のエラーが発生していた場合は、警告を表示します。このように、想定外のエラーの発生を検出するには、エラー番号を使ってチェックが行えます。

コード末尾に `On Error Goto 0` を記述してあるのは、エラー情報 (エラー管理用のシステム変数 `Err` に記録されたエラーコード) をリセットして、次回以降に発生するエラーを正確に記録させるためです。

実行時ライブラリ

以降の節では、実行時ライブラリの主要な関数について説明します。

・変換関数

・文字列

・日付および時刻

・ファイルおよびディレクトリの操作

・メッセージボックスとインプットボックス

・その他の関数

変換関数

プログラムを構築する場合、ある変数の変数型を他の種類に変換する必要がよくあります。

変数型の暗黙的変換と明示的変換

変数型を変換する一番簡単な方法は、代入処理を利用することです。

```
Dim A As String
Dim B As Integer

B = 101
A = B
```

このサンプルコードでは、変数 A は文字列型、変数 B は整数型として宣言しています。そしてこのような状況で変数 B に変数 A を代入すると、自動的に OpenOffice.org Basic が変数 B の変数型を文字列型に変換します。このような変換では、見た目よりも複雑な処理が行われており、まず変数 B の整数値が、2 バイト長のデータとして作業用メモリに格納されます。A 一方の変数は文字列型であるので、各文字 (文字や数字) ごとに 1 または 2 バイト長のデータとしてメモリ内に格納されています。このため変数 B に変数 A を代入する際には、変数 B 側の内部フォーマットに一致するよう、変数 A のデータを変換しておく必要があります。

他の多くのプログラミング言語とは異なり、Basic の場合、変数型の変換は自動的に実行されます。ただし、予想外の結果が生じることがあります。ここで、どのような問題が生じるかについて、次のサンプルコードを使って検討をしてみます。

```
Dim A As String
Dim B As Integer
Dim C As Integer

B = 1
C = 1
A = B + C
```

一見するとこのコードに特に問題はなさそうですが、目に見えない形で落とし穴が潜んでいます。Basic インタプリタは、最初に加算演算を行なってから、その結果を文字列変数に代入するので、ここで得られる結果は 2 という文字列になります。

逆に、Basic インタプリタが、最初に変数 B と変数 C の値を文字列に変換し、その結果に対してプラス記号による演算を適用するのであれば、得られる結果は 11 という文字列になります。

同様の問題は、バリアント型変数を使用する際にも生じます。

```
Dim A
Dim B
Dim C

B = 1
C = ”1”
A = B + C
```

バリアント型変数には数値と文字列の両方のデータを格納できるため、この変数 A に代入されるのが数値 2 なのか文字列 11 なのかは不明です。

このような変数型の暗黙的変換に起因するエラーを回避するには、プログラム内にバリアント型変数を使用しないなど、このような点に注意してプログラミングを行う必要があります。

変数型の暗黙的変換に起因するエラーを防止する観点から、OpenOffice.org Basic には各種の変換関数が用意されており、様々なデータ型に対する変換操作を行えるようになっています。

CStr(Var)

任意の型のデータを文字列に変換します。

CInt(Var)

任意の型のデータを整数値に変換します。

CLng(Var)

任意の型のデータをロング整数値に変換します。

CSng(Var)

任意の型のデータを単精度値に変換します。

CDbl(Var)

任意の型のデータを倍精度値に変換します。

CBool(Var)

任意の型のデータをブール値に変換します。

CDate(Var)

任意の型のデータを日付データに変換します。

これらの変換関数を使用することで、OpenOffice.org Basic による型変換を明示的に実施することができます。

```
Dim A As String
Dim B As Integer
Dim C As Integer

B = 1
C = 1
A = CStr(B + C)         ’ B and C are added together first, then
                        ’ converted to the string ”2”
A = CStr(B) + CStr(C)   ’ B and C are converted into a string,then
                        ’ combined to produce the string ”11”
```

このサンプルコードの最初の変換操作では、OpenOffice.org Basic に整数値の加算を行わせてから、その結果を文字列に変換しています。変数 A に代入される値は、文字列の 2 となります。2 番目の変換操作では、整数変数の値を文字列に変換してから、代入時に結合処理を行なっています。変数 A に代入される値は、文字列の 11 となります。

数値変換用の CSng および CDbl 関数は、小数を扱うこともできます。ただしこの場合の小数点には、各地域ごとの各ロケール設定で指定されている小数点記号を使用する必要があります。また逆に、CStr メソッドで数値、日付、時刻のデータを変換する際には、各ロケール設定による書式が適用されます。

Val 関数は Csng、Cdbl メソッドおよび Cstr メソッドとは使用法が少し異なります。この関数は文字列を数値に変換しますが、小数点として使える記号はピリオドに固定されています。

```
Dim A As String
Dim B As Double

A = ”2.22”
B = Val(A)      ’ Is converted correctly regardless of the
                ’ country-specific settings
```

変数の内容の確認

場合によってはデータの変換ができないことがあります。

```
Dim A As String
Dim B As Date

A = ”test”
B = A           ’ Creates error message
```

このサンプルコードからも分かるように、日付変数に test という文字列は代入できないので、このような場合 Basic インタプリタはエラーメッセージを表示します。同様のことは、ブール型変数に文字列を代入するような場合にも当てはまります。

```
Dim A As String
Dim B As Boolean

A = ”test”
B = A           ’ Creates error message
```

上記のサンプルコードでも、Basic インタプリタはエラーメッセージを表示します。

このようなエラーは、代入を行う前のプログラム行で、代入させるデータ型が変数型と一致するかをチェックさせることで回避できます。OpenOffice.org Basic にはこのような処理を行うために、次のテスト関数が用意されています。

IsNumeric(Value)

Value の値が数値であるかチェックします。

IsDate(Value)

Value の値が日付であるかチェックします。

IsArray(Value)

Value の値が配列であるかチェックします。

これらの関数は、ユーザーの入力値を受け取る際に有用です。たとえば数値や日付などの、想定される形式のデータをユーザーが入力したかをチェックできます。

```
If IsNumeric(UserInput) Then
  ValidInput = UserInput
Else
  ValidInput = 0
  MsgBox "Error message."
End If
```

上記のサンプルコードでは、変数 `UserInput` の値が数値であれば、その値を変数 `ValidInput` に代入しています。変数 `UserInput` の値が数値でなければ、変数 `ValidInput` には `0` を代入して、エラーメッセージを表示します。

OpenOffice.org Basic の組み込み関数には、このような数値、日付、配列に対するテスト関数は用意されていますが、ブール値に対するテスト関数は用意されていません。このようなチェック機能が必要であれば、次の関数 `IsBoolean` のような関数を記述することで実装できます。

```
Function IsBoolean(Value As Variant) As Boolean
  On Error Goto ErrorIsBoolean:
  Dim Dummy As Boolean
  Dummy = Value
  IsBoolean = True
  On Error Goto 0
  Exit Sub

  ErrorIsBoolean:
  IsBoolean = False
  On Error Goto 0
End Function
```

上記の関数 `IsBoolean` では、ブール型の局所変数 `Dummy` を用意して、与えられたデータをその中に代入させています。この代入が問題なく実行できれば、関数の戻り値として `True` を返します。そして代入に失敗した場合は、実行時エラーが発生するので、このテスト関数の実行は中断され、エラー処理を行います。

OpenOffice.org Basic の文字列に含まれている数値以外の値が数字に割り当てられていると、OpenOffice.org Basic はエラーメッセージを表示せず、最初の無効な文字で文字列の変換を停止します。VBA の場合は、これとは異なる処理が行われます。VBA でこのような代入を行おうとすると、エラーが発生し、処理が中断されます。

文字列

文字コードの操作

OpenOffice.org Basic の文字列操作では、文字コードとして Unicode が使用されます。Asc および Chr 関数は、Unicode のコード番号と該当する文字との間の変換を行います。次のサンプルコードでは、各種の Unicode 記号を該当するコード番号に変換します。

```
Code = Asc("A")         ' Latin letter A (Unicode-value 65)
Code = Asc("€")         ' Euro character (Unicode-value 8364)
Code = Asc("Л")          ' Cyrillic letter Л (Unicode-value 1083)
```

次のサンプルコードは、これらと逆方向の変換を実行します。

```
MyString = Chr(13)
```

ここでは、コード番号 13 が割り当てられている文字 (改行コード) を、文字列変数 MyString に代入しています。

このように、Chr 関数は、Basic プログラミング内で制御コードを文字列変数に代入する際によく使われます。たとえば、次のサンプルコードのように使用します。

```
MyString = Chr(9) + "This is a test" + Chr(13)
```

ここでは、文字列の前にタブコード (Unicode 値 9) を、後ろに改行コード (Unicode 値 13) を付けています。

文字列の一部の取得

OpenOffice.org Basic には、文字列の一部の取得用に、次の 4 つの関数が用意されています。

Left(MyString, Length)

文字列 MyString の左端から Length 分の文字を取得します。

Right(MyString, Length)

文字列 MyString の右端から Length 分の文字を取得します。

Mid(MyString, Start, Length)

文字列 MyString の左端 Start 文字目から Length 分の文字を取得します。

Len(MyString)

文字列 MyString の文字数を返します。

次のサンプルコードは、これらの使用例です。

```
Dim MyString As String
Dim MyResult As String
Dim MyLen As Integer

MyString = "This is a small test"
MyResult = Left(MyString,5)      ' Provides the string "This "
MyResult = Right(MyString, 5)    ' Provides the string " test"
MyResult = Mid(MyString, 8, 5)   ' Provides the string " a sm"
MyLen = Len(MyString)            ' Provides the value 21
```

検索と置換

OpenOffice.org Basic には、文字列内の部分文字列の検索用に、InStr 関数が用意されています。

```
ResultString = InStr (MyString, SearchString)
```

この関数は、文字列変数 MyString 内に文字列変数 SearchString と一致する部分があるかを調べます。関数の戻り値としては、文字列変数 MyString 内で最初に文字列変数 SearchString が現れる位置を、数値で返します。また、該当する部分文字列が何カ所もあるような場合は、OpenOffice.org Basic が何文字目から検索を始めるかをオプション指定できます。このオプションは、次の形式で記述します。

```
ResultString = InStr(StartPosition, MyString, SearchString)
```

また上記のいずれの場合も、InStr 関数は文字列の大文字と小文字を無視します。InStr 関数で大文字と小文字を区別させるには、次のように最終パラメータとして 0 を追加します。

```
ResultString = InStr(MyString, SearchString, 0)
```

これまでに説明した文字列操作用の関数を組み合わせると、次のような文字列の置換関数を作成できます。

```
Function Replace(Source As String, Search As String, NewPart As String)
  Dim Result As String
  Dim StartPos As Long
  Dim CurrentPos As Long

  Result = ""
  StartPos = 1
  CurrentPos = 1

  If Search = "" Then
    Result = Source
  Else
    Do While CurrentPos <> 0
      CurrentPos = InStr(StartPos, Source, Search)
      If CurrentPos <> 0 Then
        Result = Result + Mid(Source, StartPos, _
        CurrentPos - StartPos)
        Result = Result + NewPart
        StartPos = CurrentPos + Len(Search)
      Else
        Result = Result + Mid(Source, StartPos, Len(Source))
      End If               ' Position <> 0
    Loop
  End If

  Replace = Result
End Function
```

この関数は、ループ内に InStr 関数を配置して、文字列パラメータ Source 内にある文字列パラメータ Search の該当位置を検索します。検索がヒットしたら、該当位置より前の部分を取得して、バッファ用の文字列変数 Result に格納します。そして、文字列パラメータ Search の該当位置を置き換えるよう、新しい Part セクションを挿入します。上記の手順を繰り返し、検索する文字列の残りの部分に該当位置がなくなった段階で、この部分をバッファ用文字列の末尾に追加します。こうして得られた文字列を関数の戻り値として返しますが、これが該当箇所を置換した文字列となります。

このような文字列の一部を置換するという作業は使用頻度が高いため、OpenOffice.org Basic の Mid 関数には、これを行うための拡張機能が用意されています。この例では、文字列 MyString の 6 文字目から 3 文字分を is という文字列に置き換えています。

```
Dim MyString As String

MyString = "This was my text"
Mid(MyString, 6, 3, "is")
```

文字列の書式設定

Format 関数は、数値を文字列として書式設定します。このような処理を行う際には、数値の書式設定用テンプレートを指定する必要がありますが、この関数ではパラメータ Format として指定します。テンプレートは、最終的に出力する文字種に対応したプレースホルダを並べて指定します。使用頻度の高いプレースホルダは、ゼロ (0)、ナンバー記号 (#)、ピリオド (.)、コンマ (,)、ドル記号 ($) の 5 種類です。

0 文字は、該当桁に数字を表示させるための記号です。該当桁に数値が来ない場合は、0 が表示されます。

. は、小数点の位置を指定するための記号で、小数点はオペレーティングシステムのロケール設定に応じたものが使用されます。

次のサンプルコードは、0 と . による書式指定により数値の小数部が処理される例を示します。

```
MyFormat = ”0.00”
MyString = Format(-1579.8, MyFormat)     ’ Provides ”-1579,80”
MyString = Format(1579.8, MyFormat)      ’ Provides ”1579,80”
MyString = Format(0.4, MyFormat)         ’ Provides ”0,40”
MyString = Format(0.434, MyFormat)       ’ Provides ”0,43”
```

また、数値の整数部の桁数がテンプレートよりも小さい場合、該当桁には 0 が表示されます。

```
MyFormat = ”0000.00”
MyString = Format(-1579.8, MyFormat)     ’ Provides ”-1579,80”
MyString = Format(1579.8, MyFormat)      ’ Provides ”1579,80”
MyString = Format(0.4, MyFormat)         ’ Provides ”0000,40”
MyString = Format(0.434, MyFormat)       ’ Provides ”0000,43”
```

, は、千単位の桁区切り位置を指定するための記号で、オペレーティングシステムのロケール設定に応じた桁区切り記号を表示させます。# は、該当桁の数字を表示させるよう指定する記号ですが、該当桁に数値が無い場合は何も表示させません。

```
MyFormat = ”#,##0.00”
MyString = Format(-1579.8, MyFormat)     ’ Provides ”-1.579,80”
MyString = Format(1579.8, MyFormat)      ’ Provides ”1.579,80”
MyString = Format(0.4, MyFormat)         ’ Provides ”0,40”
MyString = Format(0.434, MyFormat)       ’ Provides ”0,43”
```

$ プレースホルダの代わりに、Format 関数は、システムで定義されている関連する通貨記号を表示します (次の例では、ヨーロッパロケールが定義されているものとします)。

```
MyFormat = "#,##0.00 $"
MyString = Format(-1579.8, MyFormat)     ' Provides "-1.579,80 €"
MyString = Format(1579.8, MyFormat)      ' Provides "1.579,80 €"
MyString = Format(0.4, MyFormat)         ' Provides "0,40 €"
MyString = Format(0.434, MyFormat)       ' Provides "0,43 €"
```

VBA で日時の詳細な書式設定用に使用されるフォーマット命令も使用できます。

```
sub main
    dim myDate as date
    myDate = "01/06/98"
    TestStr = Format(myDate, "mm-dd-yyyy") ' 01-06-1998
    MsgBox TestStr
end sub
```

日付および時刻

OpenOffice.org Basic に用意されている Date 型データは、日付および時刻の情報をバイナリ形式で格納しています。

プログラムコード内での日付と時刻の指定

日付変数へ代入する日付データは、文字列として指定します。

```
Dim MyDate As Date
MyDate = ”24.1.2002”
```

このような文字列形式のデータを日付変数へ代入した場合、OpenOffice.org Basic は自動的に必要なデータ変換を実行します。ただし、日付や時刻の表示形式は地域ごとに異なる場合があるので、このような差異に起因したエラーの発生する危険性があります。

OpenOffice.org Basic はオペレーティングシステムのロケール設定に応じて日付データを変換するため、上記のサンプルコードはこの書式の該当する地域でしか正常に動作しません。

このような問題を回避するには、DateSerial 関数を使用して日付データを指定するようにします。

```
Dim MyVar As Date
MyDate = DateSerial (2001, 1, 24)
```

この関数に与えるパラメータは、年、月、日の順番で指定するよう固定されています。そのためこの関数を使用することで、ロケール設定に影響されることなく、一定の形式で日付データの指定が行えます。

時刻の場合は TimeSerial 関数を用いることで、DateSerial 関数と同様の処理が行えます。

```
Dim MyVar As Date
MyDate =  TimeSerial(11, 23, 45)
```

この関数に与えるパラメータは、時、分、秒の順番で指定します。

日付および時刻の取得

次の関数は、DateSerial 関数および TimeSerial 関数と逆方向の操作を行うためのものです。

Day(MyDate)

MyDate に該当する日を返します。

Month(MyDate)

MyDate に該当する月を返します。

Year(MyDate)

MyDate に該当する年を返します。

Weekday(MyDate)

MyDate に該当する曜日を示す数値を返します。

Hour(MyTime)

MyTime に該当する時刻の時を返します。

Minute(MyTime)

MyTime に該当する時刻の分を返します。

Second(MyTime)

MyTime に該当する時刻の秒を返します。

これらの関数は、指定した Date 変数に該当する、日付や時刻の情報を取得します。次のサンプルコードでは、MyDate で保存した日付が 2003 年の日付かどうかを確認しています。

```
Dim MyDate As Date
' ... Initialization of MyDate

If Year(MyDate) = 2003 Then
  ' ... Specified date is in the year 2003
End If
```

同様に、次のサンプルコードでは、MyTime が 12 時と 14 時の間かどうかを確認しています。

```
Dim MyTime As Date
' ... Initialization of MyTime

If Hour(MyTime) >= 12 And Hour(MyTime) < 14 Then
  ' ... Specified time is between 12 and 14 hours
End If
```

Weekday 関数は、与えられた日付データを基に、該当する曜日を示す数値を返します。

```
Dim MyDate As Date
Dim MyWeekday As String
' ... initialize MyDate

Select Case WeekDay(MyDate)
  case 1
    MyWeekday = "Sunday"
  case 2
    MyWeekday = "Monday"
  case 3
    MyWeekday = "Tuesday"
  case 4
    MyWeekday = "Wednesday"
  case 5
    MyWeekday = "Thursday"
  case 6
    MyWeekday = "Friday"
  case 7
    MyWeekday = "Saturday"
End Select
```

ここでは、週の第 1 曜日は日曜日 (Sunday) と仮定しています。

システム日付と時刻の取得

OpenOffice.org Basic には、システム日付と時刻の取得用に、次の関数が用意されています。

Date

現在の日付を取得して返します。

Time

現在の時刻を取得して返します。

Now

現在の日付と時刻を取得して返します (日付と時刻を 1 つに合わせたデータ)。

ファイルおよびディレクトリ

ファイル操作は、アプリケーションの基本タスクの 1 つです。OpenOffice.org API には、Office ドキュメントの作成、オープン、編集用に必要となる、各種のオブジェクトが用意されています。オブジェクトの詳細情報については、「OpenOffice.org API ついて」を参照してください。また状況によっては、ファイルシステムの直接アクセス、ディレクトリの検索、テキストファイルの編集などを行う必要もあります。OpenOffice.org Basic の実行時ライブラリには、このようなタスクを行うための関数が各種用意されています。

ファイルおよびディレクトリ関係の関数のうち DOS 固有のものは、OpenOffice.org でサポートされなくなったか、使用に制限が付くようになりました。たとえば、`ChDir` 関数、`ChDrive` 関数、`CurDir` 関数は現行バージョンでは使用できません。また DOS 固有の属性についても、ファイル属性をパラメータとする関数で使用できなくなっています (たとえば、隠しファイルとシステムファイルの区別など)。これらは、OpenOffice.org のプラットフォーム独立性を確保するために必要な措置の一環です。

ファイル管理

ディレクトリ内のファイル検索

ディレクトリシステム内でファイルやサブディレクトリの検索を行うには、OpenOffice.org Basic の `Dir` 関数を使用します。`Dir` 関数を最初に使用する際には、検索するディレクトリのパスを第 1 パラメータとして指定する必要があります。`Dir` 関数の第 2 パラメータには、検索するファイルやディレクトリの名前を指定します。この関数の戻り値としては、最初に見つかったディレクトリエントリの名前が OpenOffice.org Basic により返されます。次の該当エントリを取得するには、パラメータを付けずに再度 `Dir` 関数を呼び出します。該当するエントリが存在しなかった場合、`Dir` 関数は空白文字列を返します。

次のサンプルコードでは、`Dir` 関数を使って、指定ディレクトリ内に存在するすべてのファイルを一覧表示します。この手続きでは、個々のファイル名を変数 `AllFiles` に記録してゆき、最後の段階でメッセージボックスに一括表示させています。

```
Sub ShowFiles
  Dim NextFile As String
  Dim AllFiles As String

  AllFiles = ""
  NextFile = Dir("C:¥", 0)

  While NextFile  <> ""
    AllFiles = AllFiles & Chr(13) &  NextFile
    NextFile = Dir
  Wend

  MsgBox AllFiles
End Sub
```

Dir 関数の第 2 パラメータに 0 (ゼロ) を指定しているのは、ディレクトリを無視してファイル名のみを Dir 関数に返させるためです。このようなパラメータ値としては、次のものを指定できます。

・0: 通常のファイルのみを対象とする。

・16: サブディレクトリのみを対象とする。

次のサンプルコードでは、基本的に上記のものと同じ処理を行なっていますが、こちらは Dir 関数の第 2 パラメータに 16 を指定しているので、ファイルではなくサブディレクトリを対象にしています。

```
Sub ShowDirs
  Dim NextDir As String
  Dim AllDirs As String

  AllDirs = ""
  NextDir = Dir("C:¥", 16)

  While NextDir <> ""
    AllDirs = AllDirs & Chr(13) &  NextDir
    NextDir  <nowiki>= Dir</nowiki>
  Wend

  MsgBox AllDirs
End Sub
```

OpenOffice.org Basic の Dir 関数にパラメータ値 16 を指定すると、該当ディレクトリ内のサブディレクトリのみが返されます。VBA で同様の処理を行うと、サブディレクトリだけでなく通常のファイルも返されるので、ディレクトリ名のみを取得するには追加の操作が必要です。CompatibilityMode (true) 関数を使用すると、OpenOffice.org Basic は VBA と同様に動作します。つまり、Dir 関数のパラメータに 16 を指定すると、サブディレクトリだけでなく通常のファイルも返されます。

VBA では、隠しファイル、システムファイル、アーカイブファイル、ボリューム名などの属性に基づいた検索オプションを利用できます。OpenOffice.org Basic の場合、このようなファイル機能の存在しな

いオペレーティングシステム上での操作も前提としているため、該当するオプションは用意されていません。

Dir に一覧で指定されたパスでは、* および ? を使用する可能性がありますが、これらは VBA および OpenOffice.org Basic 両方のプレースホルダとして使用できます。ただし VBA とは異なり、OpenOffice.org Basic で * プレースホルダを使用すると、ファイル拡張子かファイル名の最後の文字のみが一致する場合があります。

ディレクトリの作成および削除

OpenOffice.org Basic でディレクトリを作成するには、MkDir 関数を使用します。

```
MkDir ("C:¥SubDir1")
```

この関数は、ディレクトリおよびサブディレクトリを作成します。また必要であれば、指定ディレクトリに対する下層ディレクトリも作成できます。たとえば C:\SubDir1 ディレクトリのみが存在する状況で次のコードを実行したとします。

```
MkDir ("C:¥SubDir1¥SubDir2¥SubDir3¥")
```

この場合、C:\SubDir1\SubDir2 ディレクトリおよび C:\SubDir1\SubDir2\SubDir3 ディレクトリの両方が作成されます。

ディレクトリを削除するには、RmDir 関数を使用します。

```
RmDir ("C:¥SubDir1¥SubDir2¥SubDir3¥")
```

ディレクトリにサブディレクトリまたはファイルが含まれている場合、それらも削除されます。このため、RmDir を使用する場合は、注意が必要です。

VBA の MkDir 関数および RmDir 関数は、現在のディレクトリのみを操作対象とします。これに対して、OpenOffice.org Basic では、MkDir および RmDir を使用して、複数レベルのディレクトリを作成または削除できます。

VBA の RmDir 関数で、ファイルを含むディレクトリを削除しようとするとエラーメッセージが表示されます。OpenOffice.org Basic の場合は、指定ディレクトリおよびその中のすべてのファイルが削除されます。CompatibilityMode (true) 関数を使用すると、OpenOffice.org Basic は、VBA と同じように動作します。

ファイルのコピー、名前変更、削除および存在確認

次の呼び出しでは、Destination という名前で、Source ファイルのコピーが作成されます。

```
FileCopy(Source, Destination)
```

次の関数を使用すると、OldName ファイルを NewName という名前に変更できます。このようにコンマを使わずキーワード As を使用する構文が、Basic 言語の基本形です。

```
Name OldName As NewName
```

次の呼び出しでは、Filename ファイルが削除されます。ディレクトリ (およびその中のファイル) を削除するには、RmDir 関数を使用してください。

```
Kill(Filename)
```

特定のファイルが存在するかを確認するには、FileExists 関数を使用します。

```
If FileExists(Filename) Then
  MsgBox "file exists."
End If
```

ファイル属性の読み取りと変更

ファイル関連の処理を行う際には、最終変更日時、ファイルサイズなどのファイル属性の確認が重要となる場合があります。

次の呼び出しでは、ファイルに関するいくつかの属性が返されます。

```
Dim Attr As Integer
Attr = GetAttr(Filename)
```

戻り値はビットマスクの形式で返され、その中には次のような情報が含まれています。

・1: 読み取り専用のファイル

・16: ディレクトリ名

ここでは、test.txt という名前のファイルのビットマスクを判定して、その情報を基に、読み取り専用であるかどうか、またディレクトリであるかどうかを調べています。どちらにも該当しなかった場合は、変数 FileDescription に「normal」という文字列を代入させています。

```
Dim FileMask As Integer
Dim FileDescription As String

FileMask = GetAttr(”test.txt”)

If (FileMask AND 1) > 0 Then
  FileDescription = FileDescription & ” read-only ”
End IF

If (FileMask AND 16) > 0 Then
  FileDescription = FileDescription & ” directory ”
End IF

If FileDescription = ”” Then
  FileDescription = ” normal ”
End IF

MsgBox FileDescription
```

VBA では、隠しファイル、システムファイル、アーカイブファイル、ボリューム名のファイル属性を示すフラグを利用できますが、これらは Windows 固有の機能であり、OpenOffice.org Basic の場合は、このような機能の存在しないオペレーティングシステム上での操作も前提としているため、これらのフラグはサポートされていません。

ファイル属性を変更するには、SetAttr 関数を使用します。次の呼び出しは、指定ファイルに読み取り専用の属性を設定しています。

```
SetAttr(”test.txt”, 1)
```

逆に、ファイルの読み取り専用属性を解除するには、次のように指定します。

```
SetAttr(”test.txt”, 0)
```

ファイルを最後に変更した日付と時刻を調べるには、FileDateTime 関数を使用します。このようにして得られる日付の書式は、システムのロケール設定に従います。

```
FileDateTime(”test.txt”)   ’ Provides date and time of the last file
amendment.
```

ファイルサイズを調べるには、FileLen 関数を使用します (戻り値はロング整数のバイト値)。

```
FileLen(”test.txt”)      ’ Provides the length of the file in bytes
```

テキストファイルの書き込みと読み取り

OpenOffice.org Basic には、ファイルの書き込みおよび読み取り用の各種メソッドが用意されています。次の説明は、テキストファイル関係の機能をまとめたものです (注意: StarSuite の文書ドキュメントに関するものではありません)。

テキストファイルへの書き込み

テキストファイルにアクセスするには、該当ファイルを事前にオープンしておく必要があります。その際には、フリーのファイルハンドルを使って、アクセスするファイルを特定します。

フリーのファイルハンドルを作成するには、FreeFile 関数を使用します。このハンドルは、Open 命令によるファイルオープン時にパラメータとして指定します。テキストファイルとしてファイルをオープンするには、次のように Open 命令を記述します。

```
Open Filename For Output As #FileNo
```

Filename には、ファイル名を文字列の形で指定します。同じく FileNo には、FreeFile 関数で作成したファイルハンドルを指定します。

オープンしたファイルに対しては、Print 命令により、1 行単位の書き込みが行えます。

```
Print #FileNo, "This is a test line."
```

FileNo はファイルハンドルを示します。第 2 パラメータのテキストは、テキストファイルへ書き込む行の内容を示します。

書き込みの終了したファイルに対しては、Close によるクローズ処理が必要です。

```
Close #FileNo
```

この場合も、ファイルハンドルを指定する必要があります。

次のサンプルコードは、ファイルのオープンから、書き込み、クローズまでの流れを示します。

```
Dim FileNo As Integer
Dim CurrentLine As String
Dim Filename As String

Filename = "c:¥data.txt"            ' Define file name
FileNo = Freefile               ' Establish free file handle

Open Filename For Output As #FileNo         ' Open file (writing mode)
Print #FileNo, "This is a line of text"      ' Save line
Print #FileNo, "This is another line of text"   ' Save line
Close #FileNo                   ' Close file
```

テキストファイルの読み取り

テキストファイルの読み取りは、書き込みと同様の手順で行います。ただし Open 命令によるファイルのオープン時には For Output の代わりに For Input を指定し、Print 命令によるデータの書き込みではなく Line Input 命令によるデータの読み取りを行う点が異なります。

テキストファイルの呼び出し時に、eof 命令を使用して、ファイルの末尾に到達したかどうかを確認します。

```
eof(FileNo)
```

次のサンプルコードは、テキストファイルからのデータの読み取り手順を示します。

```
Dim FileNo As Integer
Dim CurrentLine As String
Dim File As String
Dim Msg as String

’ Define filename
Filename = ”c:¥data.txt”

’ Establish free file handle
FileNo = Freefile

’ Open file (reading mode)
Open Filename For Input As FileNo

’ Check whether file end has been reached
Do While not eof(FileNo)
  ’ Read line
  Line Input #FileNo, CurrentLine
  If CurrentLine <>”” then
    Msg = Msg & CurrentLine & Chr(13)
  end if
Loop


’ Close file

Close #FileNo
Msgbox Msg
```

ここでは、Do While ループを使ってデータを 1 行ずつ読み出しては、変数 Msg に追加することにより格納してゆき、最後にまとめてメッセージボックスに出力させています。

メッセージボックスとインプットボックス

OpenOffice.org Basic には、ユーザーへの入出力用に、MsgBox および InputBox 関数が用意されています。

メッセージの出力

MsgBox を使うと、簡単なメッセージ表示用のダイアログを表示できるだけでなく、その中に 1 つまたは複数のボタンを配置できます。最も簡単なバリアントでは、MsgBox に含まれるのはテキストと OK ボタンだけです。

```
MsgBox "This is a piece of information!"
```

メッセージボックスの表示形態は、パラメータ指定により変更できます。ここでは、ボタンの種類、標準ボタン、追加表示するアイコンをパラメータ指定により設定できます。ボタンの指定は、次の数値を通じて行います。

・0 - OK ボタンのみを表示します。

・1 - OK およびキャンセルの各ボタンを表示します。

・2 - 中止、やり直し、および取消しボタンの各ボタンを表示します。

・3 - はい、いいえ、キャンセルの各ボタンを表示します。

・4 - はい、いいえの各ボタンを表示します。

・5 - やり直しおよびキャンセルの各ボタンを表示します。

なお、表示ボタンの指定用パラメータ値に次の値を加えることで、いずれか 1 つのボタンを標準ボタンに設定できます。たとえば、はい、いいえ、キャンセルの各ボタンを表示させ (指定値 3)、キャンセルを標準ボタンに設定する (指定値 512) には、両者の値を加えた 3 + 512 = 515 を指定します。

・0 - 1 番目のボタンを標準ボタンに設定します。

・256 - 2 番目のボタンを標準ボタンに設定します。

・512 - 3 番目のボタンを標準ボタンに設定します。

また、次の値をパラメータ値に加えることで、該当するアイコンを追加表示できます。

・16 - ストップ記号のアイコンを表示します。

・32 - 疑問符アイコンを表示します。

・48 - 感嘆符アイコンを表示します。

・64 - ヒント記号のアイコンを表示します。

この場合は、はいといいえのボタンを表示させて (指定値 4)、2 番目のボタン (いいえ) を標準ボタンに設定し (指定値 256)、疑問符アイコンを表示させる (指定値 32) ので、パラメータ値として 4 + 256 + 32 = 292 を指定します。

```
MsgBox ”Do you want to continue?”,  292
```

メッセージボックスに複数のボタンを表示した場合、通常は、どのボタンをユーザーが押したかを確認する処理が必要となります。このような処理には、次の戻り値を利用します。

・1 - OK

・2 - キャンセル

・3 - 終了

・4 - やり直し

・5 - 無視する記号・文字

・6 - はい

・7 - いいえ

たとえば先のサンプルコードの場合、戻り値の判定は次のようにして行えます。

```
If MsgBox (”Do you want to continue?”,  292) = 6 Then
  ’ Yes button pressed
Else
  ’ No button pressed
End IF
```

MsgBox では、メッセージとして表示するテキストおよび表示ボタンの指定用パラメータの他に、第 3 のパラメータとしてメッセージボックスのタイトルを設定できます。

```
MsgBox ”Do you want to continue?”,  292, ”Box Title”
```

このメッセージボックスのタイトル指定を省略した場合、「soffice」が標準タイトルとして使用されます。

インプットボックスと簡単な文字列入力

ユーザーからの簡単な文字列入力を受け付けるには、InputBox 関数を使用します。このような処理に関しては、ダイアログを構築するよりも、この関数を利用する方が簡単です。
InputBox 関数には、次の 3 つのパラメータを指定します。

・説明用のテキスト

・インプットボックスのタイトル

・入力フィールドに表示しておく標準値

```
InputVal = InputBox(”Please enter value:”, ”Test”, ”default value”)
```

InputBox 関数の戻り値には、ユーザーの入力した文字列が返されます。

その他の関数

Beep

不正な操作が行われた場合に警告音を鳴らすには、Beep 関数を使ってシステムのビープ音を発生させます。Beep 関数に指定するパラメータは存在しません。

```
Beep     ' creates an informative tone
```

Shell

外部プログラムを起動させるには、Shell 関数を使用します。

```
Shell(Pathname, Windowstyle, Param)
```

Pathname には、実行する外部プログラムのパス名を指定します。

Windowstyle には、プログラム起動時のウィンドウのスタイルを指定します。

ここには、次のいずれかの値を指定できます。

・0 - プログラムウィンドウを非表示にして、フォーカスを移動します。

・1 - プログラムウィンドウを標準サイズにして、フォーカスを移動します。

・2 - プログラムウィンドウを最小化 (アイコン化) して、フォーカスを移動します。

・3 - プログラムウィンドウを最大表示にして、フォーカスを移動します。

・4 - プログラムウィンドウを標準サイズにしますが、フォーカスは移動しません。

・6 - プログラムウィンドウを最小化しますが、フォーカスは現在のウィンドウにとどめておきます。

・10 - プログラムウィンドウを全画面表示にします。

第 3 パラメータ Param には、実行するプログラムに渡すコマンド行パラメータを指定できます。

Wait

プログラムの実行を指定した時間だけ中断させるには、Wait 関数を使用します。待機させる時間は、ミリ秒単位で指定します。これは、次のサンプルコードのように記述します。

```
Wait 2000
```

この場合の待機時間は、2 秒 (2000 ミリ秒) です。

Environ

オペレーティングシステムの環境変数を取得するには、Environ 関数を使用します。こうして得られる情報は、使用するシステムや環境ごとに異なります。次の呼び出しは、オペレーティングシステムの一時ディレクトリに関する環境変数を取得します。

```
Dim TempDir
TempDir=Environ (”TEMP”)
```

OpenOffice.org API について

OpenOffice.org API とは、OpenOffice.org へのアクセスに使用する汎用プログラミングインターフェースです。OpenOffice.org ドキュメントの作成、オープン、変更、印刷を行うには、このような OpenOffice.org API を利用します。またユーザーが定義したマクロやダイアログを使用して、OpenOffice.org の機能を拡張するオプションも用意されています。

OpenOffice.org API の使用は、必ずしも OpenOffice.org Basic に限定されるものではなく、必要であれば Java や C++ などのプログラミング言語から利用することもできます。そのような場合は、各種プログラミング言語との間のインターフェースとして、**Universal Network Objects** (UNO) と呼ばれる機能を利用します。

この章では、UNO を併用することで、OpenOffice.org Basic から OpenOffice.org を制御する方法に焦点を当てて説明します。それにはまず、OpenOffice.org Basic プログラミングの視点から、UNO の主要な概念について理解しておく必要があります。個々の OpenOffice.org API の使用法に関する詳細については、後半の章で説明します。

Universal Network Objects (UNO)

OpenOffice.org には Universal Network Objects (UNO) という形式のプログラミングインターフェースが用意されています。これはオブジェクト指向型のプログラミングインターフェースであり、OpenOffice.org の場合、プログラム内から行う Office パッケージへのアクセスのタイプごとに細分化されています。

OpenOffice.org Basic は手続き型のプログラミング言語であるため、UNO の導入に伴い、いくつかの機能が追加されています。

OpenOffice.org Basic で Universal Network Object を使用するにあたっては、使用するオブジェクトに対応した変数宣言が必要です。この宣言は `Dim` 命令で実行します (詳細は「OpenOffice.org Basic プログラミング用の言語」を参照)。オブジェクトを宣言する際には、宣言型に `Object` を指定します。

```
Dim Obj As Object
```

たとえば上記のサンプルコードは、`Obj` という名前のオブジェクトを宣言しています。

新規作成したオブジェクト変数は、使用する前に初期化する必要があります。このような処理には、`createUnoService` 関数を使用します。

```
Obj = createUnoService("com.sun.star.frame.Desktop")
```

上記のサンプルコードの場合、新規作成したオブジェクトへの参照情報を変数 `Obj` へ割り当てています。`com.sun.star.frame.Desktop` はいわゆるオブジェクト型に類似したものですが、UNO の用語ではこのようなものを「型」ではなく「サービス」と呼んでいます。UNO の流儀に従えば、上記の Obj は「

```
com.sun.star.frame.Desktop
```

サービスをサポートしたオブジェクトに対する参照」と表現できます。つまり OpenOffice.org Basic で使われる「サービス」という用語は、他の言語で言う「型 (タイプ)」や「クラス」に該当します。

ただし Universal Network Object の持つ大きな特徴として、複数のサービスを同時にサポートできる点があります。このため UNO のサービスの中には他のサービスをサポートしているものが存在し、1 つのオブジェクトで複数のサービスを扱うケースがあります。たとえば先のサンプルコードでは

```
com.sun.star.frame.Desktop
```

サービスをサポートしたオブジェクトを作成しましたが、このオブジェクトもその他のサービスとして、ドキュメント読み込み用およびプログラム終了用のサービスをサポートしています。

VBA のオブジェクト構造は、所属するクラスにより定義しますが、OpenOffice.org Basic のオブジェクト構造は、サポートするサービスにより定義します。VBA のオブジェクトは、常に 1 つのクラスに対して割り当てられます。これに対して、OpenOffice.org Basic のオブジェクトは複数のサービスをサポートできます。

プロパティーとメソッド

OpenOffice.org Basic の各オブジェクトからは、各種のプロパティーとメソッドを呼び出すことができます。

プロパティー

「プロパティー」とはオブジェクトのもつプロパティーと同様のもので、たとえば Document オブジェクトには Filename や Title などのプロパティーが存在します。

プロパティーは、値を代入することにより設定されます。

```
Document.Title = "OpenOffice.org Basic Programmer's Guide"
Document.Filename = "basguide.odt"
```

各プロパティーには、通常の変数と同様に、どのような値を格納できるかを規定するタイプ (型) が存在します。前のプロパティー Filename および Title は、文字列タイプの一種です。

リアル属性とイミテーション属性

OpenOffice.org Basic の各オブジェクトのもつプロパティーの大部分は、UNO サービスとして定義されています。OpenOffice.org Basic のプロパティーには、このような「リアル」なプロパティー以外に、UNO レベルの 2 つのメソッドから成るプロパティーが存在します。その一方は、そのプロパティー値を取得する際に、もう一方は値を指定する際に使用します (get メソッドと set メソッド)。つまり、このような属性は、これら 2 つのメソッドの「イミテーション」として存在するとも言えます。たとえば UNO の文字オブジェクトには、該当するキーポイントの取得と変更を行う getPosition および setPosition というメソッドが用意されています。OpenOffice.org Basic のプログラムでこのような値にアクセスするには、Position プロパティーを使用します。そしてこれらとは独立した形で、オリジナルのメソッドも使用できます (ここでの例の場合は getPosition および setPosition)。

メソッド

メソッドとは、特定のオブジェクトを指定して実行する、一種の関数と見なすことができます。たとえば、先に説明した Document オブジェクトには、Save というメソッドが存在し、これは次のような形式で使用します。

```
Document.Save()
```

このようなメソッドを使用する際には、関数を呼び出す場合と同様に、パラメータの指定および戻り値の取得が行えます。実際そのようなメソッドの呼び出しは、通常の関数と同様の形式で行えます。次の呼び出しでは、ドキュメントオブジェクトの Save メソッドを呼び出す際に、パラメータとして True を指定しています。

```
Ok = Document.Save(True)
```

この Save メソッドによる処理が終了すると、その戻り値が変数 Ok に格納されます。

モジュール、サービス、インターフェース

OpenOffice.org には数百のサービスが提供されています。このような膨大な数のサービスを整理するため、これらはモジュールという形にまとめられています。単に機能面で見た場合、OpenOffice.org Basic プログラマにとってモジュールにはそれ以上の意味は特にありません。ただし、サービス名を指定する際には、該当するモジュール名も含める必要があります。完全な形のサービス名は、OpenOffice.org のサービスであることを示す `com.sun.star` 式に続けて、`frame` などのモジュール名の指定が入り、最後に `Desktop` などの実際のサービス名が続きます。つまりこの場合、完全な名前は、次のようになります。

```
com.sun.star.frame.Desktop
```

そして UNO の場合、モジュール名とサービス名の項目に加えて「インターフェース」の項目が続きます。このような項目は Java プログラミングではお馴染みのものですが、通常の Basic プログラミングで使われるものではありません。

インターフェースは、複数のメソッドを結合するものです。厳密には、UNO のサービスはメソッドではなくインターフェースをサポートし、これらのインターフェースが各種のメソッドを提供していると表現できます。つまりメソッドは、インターフェース内のサービスに対して (組み合せとして) 割り当てられているのです。Java や C++ などでプログラミングを行う場合、メソッドを要求するときにインターフェースが必要となるため、このような関係に注意を特に払う必要があります。ただし、OpenOffice.org Basic には関係ありません。この場合メソッドの呼び出しは、該当オブジェクトから直接行います。

ただし、各サービスごとに複数のインターフェースが使用されるため、各種のインターフェースに割り当てられているメソッドの関係を把握しておくと、API の扱い方を理解する助けになります。これは、ある 1 つのインターフェースに関する知識があれば、複数のサービスでその知識を生かせるためです。

一部の中心的なインターフェースは、頻繁に使用され、別のサービスによってトリガされるので、この章の最後で改めて示します。

UNO の関連ツール

UNO で使われるオブジェクトやサービスに関しては、何がどの属性、メソッド、インターフェースをサポートしているのかという点と、それをどのように確認するかという問題が残っています。オブジェクトに関する情報については、このマニュアル以外にも、supportsService メソッドおよびデバッグ用の各種メソッドのほか、『デベロッパ向けガイド』および『API reference』を参照してください。

supportsService メソッド

UNO オブジェクトの多くが supportsService メソッドをサポートしており、このメソッドを使用することで、個々のオブジェクトが特定のサービスをサポートしているかを確認できます。次のサンプルコードでは、TextElement オブジェクトが

```
com.sun.star.text.Paragraph
```

サービスをサポートするかどうかを指定します。

```
Ok = TextElement.supportsService("com.sun.star.text.Paragraph")
```

デバッグの属性

OpenOffice.org Basic では、各 UNO オブジェクトごとに使用可能な属性、メソッド、インターフェースに関する情報が、あらかじめ登録されています。このような情報は属性として取得可能で、該当項目が一覧形式で表示されます。これに該当するのは、次の属性です。

DBG_properties

オブジェクトの全属性を文字列で返します。

DBG_methods

オブジェクトの全メソッドを文字列で返します。

DBG_supportedInterfaces

オブジェクトの全インターフェースを文字列で返します。

次のサンプルコードは、DBG_properties と DBG_methods の使用例です。ここでは、まず

```
com.sun.star.frame.Desktop
```

サービスを作成してから、そのサポートする属性とメソッドをメッセージボックスに表示させます。

```
Dim Obj As Object
Obj = createUnoService("com.sun.star.frame.Desktop")

MsgBox Obj.DBG_Properties
MsgBox Obj.DBG_methods
```

DBG_properties を使用する場合、戻り値として返される属性の中には、単に分類上の都合で該当サービスで使用可能とされているだけの属性もあるので注意が必要です。つまり、これらの属性が実際に該当サービスで使用できるかについては、保証されていません。そのため、このような属性を利用する際には、IsEmpty 関数を使って利用できるかを確認する必要があります。

API Reference

使用可能なサービスおよび、該当するインターフェース、メソッド、属性に関するより詳細な情報は、OpenOffice.org API のリファレンスに収録されています。

中央インターフェースの概要

OpenOffice.org のインターフェースの中には、OpenOffice.org API の各所で使用されるものが存在します。これらは、各種の処理に利用可能な抽象的タスクを実行する一連のメソッドを規定しています。ここではこのようなインターフェースのうち、使用頻度の高いものについて、その概要を説明します。

オブジェクトの出所については、このマニュアルの後半で説明します。ここでは単に、OpenOffice.org API が主要なインターフェースを提供するオブジェクトについて、その抽象的な機能のいくつかを簡単に説明するにとどめておきます。

コンテキスト依存型オブジェクトの作成

OpenOffice.org API でオブジェクトを作成するには、2 通りの方法が存在します。その 1 つは、この章の初めに説明した createUnoService 関数を使用する方法です。createUnoService 関数は、広域的に使用可能なオブジェクトを作成します。このようなオブジェクトやサービスは、コンテキスト非依存型サービスとも呼びます。

このようなコンテキスト非依存型サービスに対するものとして、コンテキスト依存型サービスも存在し、この場合のオブジェクトは他のオブジェクトと併用することでのみ使用できます。たとえば、ある表計算ドキュメントに配置された図形描画オブジェクトは、このドキュメントと共存することにより存在できます。

com.sun.star.lang.XMultiServiceFactory インターフェース

通常、コンテキスト依存型のオブジェクトを作成するには、そのオブジェクトの依存相手のオブジェクトメソッドを使用します。createInstance メソッドは、XMultiServiceFactory インターフェースで規定されているもので、主としてドキュメントオブジェクトに対して使用します。

たとえば、先に説明した図形描画オブジェクトを作成するには、次のサンプルコードのように、オブジェクトとして表計算ドキュメントを使用します。

```
Dim RectangleShape As Object
RectangleShape = _
  Spreadsheet.createInstance("com.sun.star.drawing.RectangleShape")
```

同様に、文書ドキュメントの段落テンプレートは、次のサンプルコードのようにして作成します。

```
Dim Style as Object
Style = Textdocument.createInstance("com.sun.star.style.ParagraphStyle")
```

下位オブジェクトへの名前付きアクセス

下位オブジェクトを持つオブジェクトで自然言語名によるアクセスが可能なものに対しては、XNameAccess および XNameContainer インターフェースを使用します。

このうち XNamedAccess は個々のオブジェクトに対してアクセスを行い、XNameContainer は各要素の挿入、変更、削除を行います。

com.sun.star.container.XNameAccess インターフェース

ここでは XNameAccess の使用例として、表計算ドキュメントの表 (シート) オブジェクトを扱う場合を取り上げます。このオブジェクトは表計算ドキュメント内のすべてのページをまとめたものとして扱われます。そのため各ページへのアクセスには、XNameAccess の getByName メソッドを使用します。

```
Dim Sheets As Object
Dim Sheet As Object

Sheets = Spreadsheet.Sheets
Sheet = Sheets.getByName(”Sheet1”)
```

すべての要素の名前を取得するには、getElementNames メソッドを使用します。これを実行すると、該当する名前を収めたデータフィールド (配列) が返されます。次のサンプルコードは、ループを使用して表計算ドキュメント内のすべての要素名を取得し、表示します。

```
Dim Sheets As Object
Dim SheetNames
Dim I As Integer

Sheets = Spreadsheet.Sheets
SheetNames = Sheets.getElementNames

For I=LBound(SheetNames) To UBound(SheetNames)
  MsgBox SheetNames(I)
Next I
```

基本オブジェクト内に該当する名前の下位オブジェクトが存在するかを確認するには、XNameAccess インターフェースの hasByName メソッドを使用します。次のサンプルコードは、Spreadsheet オブジェクトに Sheet1 という名前のページがあるかを確認して、その結果をメッセージボックスに表示します。

```
Dim Sheets As Object

Sheets = Spreadsheet.Sheets
If Sheets.HasByName("Sheet1") Then
  MsgBox " Sheet1 available"
Else
  MsgBox "Sheet1 not available"
End If
```

com.sun.star.container.XNameContainer インターフェース

基本オブジェクトの下位にある要素に対する挿入、削除、変更を行うには、XNameContainer インターフェースを使用します。ここでは、insertByName、removeByName、replaceByName の各メソッドを使用できます。

これらは通常、次のサンプルコードのように使用します。この場合は、文書ドキュメントの StyleFamilies オブジェクトを使用して、ドキュメントの段落テンプレート (ParagraphStyles) に対する操作を行なっています。

```
Dim StyleFamilies As Object
Dim ParagraphStyles As Object
Dim NewStyle As Object

StyleFamilies = Textdoc.StyleFamilies
ParagraphStyles = StyleFamilies.getByName("ParagraphStyles")
ParagraphStyles.insertByName("NewStyle", NewStyle)
ParagraphStyles.replaceByName("ChangingStyle", NewStyle)
ParagraphStyles.removeByName("OldStyle")
```

ここで insertByName の行は、NewStyle というスタイルを ParagraphStyles オブジェクトと同じ名前で挿入しています。その次の replaceByName の行は、ChangingStyle で指定するオブジェクトを、NewStyle に変更しています。最後に、removeByName の行は、OldStyle で指定するオブジェクトを、ParagraphStyles から削除しています。

インデックス方式による下位オブジェクトへのアクセス

下位オブジェクトを持つオブジェクトでインデックスによるアクセスが可能なものに対しては、XIndexAccess および XIndexContainer インターフェースを使用します。

XIndexAccess には、個々のオブジェクトへアクセスするためのメソッドが用意されています。XIndexContainer には、要素を挿入したり削除するためのメソッドが用意されています。

com.sun.star.container.XIndexAccess インターフェース

XIndexAccess を用いて下位オブジェクトへアクセスするには、getByIndex および getCount メソッドを使用します。このうち getByIndex は、指定したインデックスに該当するオブジェクトを返します。同様に getCount は、使用可能なオブジェクト数を返します。

```
Dim Sheets As Object
Dim Sheet As Object
Dim I As Integer

Sheets = Spreadsheet.Sheets

For I = 0 to Sheets.getCount() - 1
  Sheet = Sheets.getByIndex(I)
  ' Editing sheet
Next I
```

このサンプルコードでは、ループを使用して個々の表 (シート) 要素にアクセスし、各要素をオブジェクト変数 Sheet に取得しています。getCount は正味の要素数を返すため、インデックスによるアクセスを行う場合、その扱いには注意が必要です。これは、getByIndex に指定するインデックスは、0 から始まるものとして処理されるためです。このため、ループカウンタの指定は 0 から getCount()-1 などのように記述する必要があります。

com.sun.star.container.XIndexContainer インターフェース

XIndexContainer インターフェースにより、関数 insertByIndex および removeByIndex が使用できます。これらに指定するパラメータは、XNameContainer の該当するメソッドと同様の構成になっています。

下位オブジェクトへの反復アクセス

インスタンスによっては、名前やインデックスではアクセスできない下位オブジェクトを持つオブジェクトが存在します。このような場合は、XEnumeration および XenumerationAccess インターフェースを使用します。これらを用いると、直接アドレスを指定することなく、各オブジェクトに存在するすべての下位要素にアクセスできます。

com.sun.star.container.XEnumeration および XenumerationAccess インターフェース

基本オブジェクトには XEnumerationAccess インターフェースが用意されていますが、このインターフェースからは createEnumeration メソッドのみが使用できます。これを使用して得られる補助オブジェクトには、hasMoreElements と nextElement メソッドを持つ XEnumeration インターフェースが用意されています。下位オブジェクトのアクセスには、これらのメソッドを使用します。

次のサンプルコードは、文書ドキュメント上のすべての段落にアクセスします。

```
Dim ParagraphEnumeration As Object
Dim Paragraph As Object

ParagraphEnumeration = Textdoc.Text.createEnumeration

While ParagraphEnumeration.hasMoreElements()
  Paragraph = ParagraphEnumeration.nextElement()
Wend
```

上記のサンプルコードでは、最初に ParagraphEnumeration という名前で補助オブジェクトを作成しています。そしてループに入り、この補助オブジェクトを用いて、文章中の各段落に順次アクセスします。テキストの末尾に到達すると hasMoreElements メソッドは False 値を返すため、これをループの終了条件に利用します。

OpenOffice.org ドキュメントの使い方

OpenOffice.org API は、可能な限り多数のタスクで共通した操作が行えることを念頭に設計されています。このような方針は、ドキュメントの作成、オープン、保存、変換、印刷およびテンプレート管理を行うためのインターフェースやサービスについても該当します。このような機能は、あらゆるドキュメントで行われる操作であるため、本章で最初に説明します。

・StarDesktop

・テンプレートとスタイル

StarDesktop

ドキュメント関係の操作で使用頻度の高いものとして、次の 2 つのサービスが挙げられます。

.

```
com.sun.star.frame.Desktop
```

サービスは、OpenOffice.org のコアサービスとよく似た機能を備えています。これにより、すべてのドキュメントウィンドウを管理する OpenOffice.org のフレームオブジェクトを扱うための機能が提供されます。またドキュメントの作成、オープン、インポートをする際にも、このサービスを使用します。

・個々のドキュメントオブジェクトの基本機能は、

```
com.sun.star.document.OfficeDocument
```

サービスで提供されます。たとえばこのサービスには、ドキュメントの保存、エクスポート、印刷を行うためのメソッドが用意されています。

```
com.sun.star.frame.Desktop
```

サービスは、OpenOffice.org が起動すると自動的に作成されます。OpenOffice.org Basic では、大域名 `StarDesktop` を使用して、このサービスを操作できます。

`StarDesktop` の最も重要なインターフェースは

```
com.sun.star.frame.XComponentLoader
```

です。これは主として、ドキュメントの作成、インポート、オープンの操作を担う `loadComponentFromURL` メソッドをカバーしています。

`StarDesktop` というオブジェクト名の起源は、StarOffice 5 の時代にまで遡るもので、当時は `StarDesktop` という 1 つのアプリケーションにすべてのドキュメントを埋め込む形で処理していました。OpenOffice.org の現行バージョンでは、この `StarDesktop` を目に見える形で使用することはありません。ただし、`StarDesktop` という名称は、アプリケーション全体の基本オブジェクトであることが直感的に分かりやすいため、現在でも OpenOffice.org のフレームオブジェクトに対して使われています。

StarDesktop オブジェクトは、以前はルート オブジェクトとして適用されていた StarOffice 5 の Application オブジェクトに代わるものです。ただし従来の Application オブジェクトとは異なり、こちらは主として新規ドキュメントのオープン処理を担当しています。従来の Application オブジェクトで使われていた、OpenOffice.org のオンスクリーン描画コントロール用の機能 (たとえば FullScreen、FunctionBarVisible、Height、Width、Top、Visible など) は、現在では使用されていません。

アクティブドキュメントへアクセスする際に、Word では Application.ActiveDocument を使用し、Excel では Application.ActiveWorkbook を使用するのに対して、OpenOffice.org の場合は、StarDesktop がこれらの役割を果たします。OpenOffice.org でアクティブドキュメントオブジェクトへアクセスするには、StarDesktop.CurrentComponent プロパティーまたは ThisComponent を使用します。

ThisComponent

ThisComponent は StarDesktop.CurrentComponent とほぼ同じオブジェクトを返しますが、1 つ大きな利点があります。Basic IDE 内から実行、デバッグ、または調査している場合、StarDesktop は Basic IDE 自体を返します。これはおそらく意図したものとは異なります。ThisComponent は、最後のアクティブなドキュメントを返します。

OpenOffice.org でのドキュメントに関する基本知識

OpenOffice.org ドキュメントを操作する際に、OpenOffice.org の行うドキュメント管理に関する基本的な知識があると有用です。このような知識としては、OpenOffice.org ドキュメントでのファイル名の扱い方や、ファイル保存時に使用されるフォーマットなどが該当します。

ファイル名の URL 指定

OpenOffice.org は、個々のプラットフォームに拘束されないアプリケーションとして設計されているため、ファイル名の取り扱いについても、Internet Standard RFC 1738 に準拠した URL 指定法を採用しています (URL 指定はオペレーティングシステムに依存しない)。この規則を使用する標準ファイル名は、最初に接頭辞 file:/// があり、その後にローカルパスが続きます。ファイル名にサブディレクトリを含む場合、ディレクトリ間の区切り記号は、Windows で用いられるバックスラッシュ (日本語フォントでは半角円記号) ではなく、フォワード スラッシュで区切ります。たとえば次のパスは、C ドライブ直下の doc ディレクトリにある test.odt ファイルを示しています。

```
file:///C:/doc/test.odt
```

ローカルのファイル名を URL に変換するには、OpenOffice.org に用意されている ConvertToUrl 関数を使用します。逆に URL をローカルファイル名に変換するには、OpenOffice.org に用意されている ConvertFromUrl 関数を使用します。

```
MsgBox ConvertToUrl("C:¥doc¥test.odt")
  ' supplies file:///C:/doc/test.odt
MsgBox ConvertFromUrl("file:///C:/doc/test.odt")
  '  supplies (under Windows) c:¥doc¥test.odt
```

このサンプルコードでは、ローカルのファイル名を URL に変換して、メッセージボックスに表示しています。その次に、URL をローカルファイル名に変換して、メッセージボックスに表示しています。

Internet Standard RFC 1738 をベースにしたため、ファイル名に 0-9、a-z、A-Z の文字を使用できます。URL でこれ以外の文字を使用する場合は、エスケープ処理をする必要があります。この処理は、エスケープする文字を Unicode の UTF-8 エンコーディング内の該当 16 進値に変換して、パーセント記号を前に付けることで行います。たとえば、ローカルファイル名に半角スペース記号が含まれている場合、URL 内で半角スペース記号は %20 と表記します。

XML ファイルフォーマット

OpenOffice.org は、XML ベースのファイルフォーマットを使用します。XML を使用しているので、ファイルを他のプログラムで開いて編集できます。

ファイルの圧縮

XML は通常のテキストファイルをベースとしているので、一般に最終的なファイルサイズは非常に大きくなります。このため、OpenOffice.org では、ファイルを ZIP 形式で圧縮して保存するようにしています。ただし、storeAsURL メソッドのオプションを使用すると、XML ファイル形式で直接保存することができます。詳細は、「storeAsURL メソッドオプション」を参照してください。

ドキュメントの作成、オープン、インポート

ドキュメントのオープン、インポート、作成は、次のメソッドを使用して行います。

```
StarDesktop.loadComponentFromURL(URL, Frame, SearchFlags,
FileProperties)
```

ここで、loadComponentFromURL の第 1 パラメータには、対象とするファイルの URL を指定します。

loadComponentFromURL の第 2 パラメータには、管理用に OpenOffice.org が内部的に作成するウィンドウのフレームオブジェクト名を指定します。通常ここには事前定義されている _blank という名前を使用しますが、この場合 OpenOffice.org は新規ウィンドウを作成します。その他にも、_hidden という名前も指定できます。この場合は該当ドキュメントを読み込んで非表示状態にします。

実際、OpenOffice.org ドキュメントを開くのに必要なのは上記 2 つのパラメータだけで、残り 2 つのパラメータは単なるプレースホルダ (ダミー値) として指定するだけです。

```
Dim Doc As Object
Dim Url As String
Dim Dummy() 'It is an (empty) array of PropertyValues

Url = "file:///C:/test.odt"

Doc = StarDesktop.loadComponentFromURL(Url, "_blank", 0, Dummy)
```

上記のサンプルコードでは、text.odt ファイルを開いて、新規ウィンドウとして表示しています。

OpenOffice.org Basic では、ここで説明した方式により任意の数のドキュメントを開くことができ、これらのドキュメントオブジェクトを操作することにより各ドキュメントを編集することもできます。

StarDesktop.loadComponentFromURL は、OpenOffice.org API の従来バージョンにあった Documents.Add および Documents.Open メソッドに代わるものです。

ドキュメントウィンドウの内容の書き換え

パラメータ Frame の値として、事前定義された _blank および _hidden という名前を指定すると、OpenOffice.org は loadComponentFromURL の呼び出し時に新規ウィンドウを作成します。ただし状況によっては、既存ウィンドウの内容を書き換える方が便利な場合もあります。このような処理を行うには、ウィンドウのフレームオブジェクトに明示的な名前が付いている必要があります。ただし、この名前の先頭には下線記号は使えないので注意が必要です。また、該当するフレームワークが存在しない場合、これを作成するにはパラメータ SearchFlags を指定する必要があります。パラメータ SearchFlags には、次の定数値を指定できます。

```
SearchFlags = com.sun.star.frame.FrameSearchFlag.CREATE + _
              com.sun.star.frame.FrameSearchFlag.ALL
```

次のサンプルコードは、フレームパラメータと SearchFlags を使って、オープン済みのウィンドウを書き換える方法を示しています。

```
Dim Doc As Object
Dim Dummy()
Dim Url As String
Dim SearchFlags As Long

SearchFlags = com.sun.star.frame.FrameSearchFlag.CREATE + _
              com.sun.star.frame.FrameSearchFlag.ALL
Url = "file:///C:/test.odt"
Doc = StarDesktop.loadComponentFromURL(Url, "MyFrame", SearchFlags,
Dummy)
MsgBox "Press OK to display the second document."

Url = "file:///C:/test2.odt"
Doc = StarDesktop.loadComponentFromURL(Url, "MyFrame", _
      SearchFlags, Dummy)
```

この例ではまず最初に、フレーム名を MyFrame とした新規ウィンドウの中に、test.odt という名前のファイルを開いています。そしてメッセージボックスで操作の確認をしてから、ウィンドウの内容を test2.odt というファイルに書き換えています。

loadComponentFromURL メソッドのオプション

loadComponentFromURL 関数の 4 番目のパラメータは、PropertyValue データフィールドです。このパラメータは、ドキュメントのオープンと作成に対するさまざまなオプションを OpenOffice.org に提供します。このデータフィールドについては、個々のオプションごとに PropertyValue 構造体を用意して、オプション名を示す文字列や必要な設定値を格納する必要があります。

loadComponentFromURL には次のオプションを指定できます。

AsTemplate (Boolean)

これが True の場合、指定 URL からドキュメントを読み込み、新規の無題ドキュメントとして表示します。False の場合は、テンプレートファイルを編集モードで読み込みます。

CharacterSet (String)

ドキュメントの基になっている文字セットを定義します。

FilterName (String)

loadComponentFromURL 関数用の特殊なフィルタを指定します。指定可能なフィルタ名は、ファイル \share\config\registry\instance\org\openoffice\office\TypeDetection.xml で定義されています。

FilterOptions (String)

フィルタ用の追加オプションを定義します。

JumpMark (String)

ドキュメントのオープン後に、JumpMark の指定位置にジャンプします。

Password (String)

保護されたファイルのパスワードを転送します。

ReadOnly (Boolean)

読み取り専用のドキュメントを読み込みます。

次のサンプルコードは、FilterName オプションを使用して、コンマ区切りのテキストファイルを OpenOffice.org Calc で開く方法を示しています。

```
Dim Doc As Object
Dim FileProperties(0) As New com.sun.star.beans.PropertyValue
Dim Url As String

Url = "file:///C:/csv.doc"
FileProperties(0).Name = "FilterName"
FileProperties(0).Value ="scalc: Text - txt - csv ({{OOo}} Calc)"

Doc = StarDesktop.loadComponentFromURL(Url, "_blank", 0,
FileProperties())
```

この場合の FileProperties データフィールドは 1 つの値しか使っていませんが、これは指定するオプションが 1 つだけだからです。Filtername プロパティーは、OpenOffice.org が OpenOffice.org Calc テキストフィルタを使用してファイルを開くかどうかを指定しています。

新規ドキュメントの作成

URL 指定されたものがテンプレートであった場合、OpenOffice.org は自動的に新規ドキュメントを作成します。

このような場合に空白ドキュメントが必要であれば、次のような形式で URL に private:factory と指定します。

```
Dim Dummy()
Dim Url As String
Dim Doc As Object

Url = "private:factory/swriter"
Doc = StarDesktop.loadComponentFromURL(Url, "_blank", 0, Dummy())
```

上記のサンプルコードを実行すると、OpenOffice.org Writer の空白ドキュメントが作成されます。

ドキュメントオブジェクト

前節で説明した loadComponentFromURL 関数は、戻り値としてドキュメントオブジェクトを返します。そのサポートする

```
com.sun.star.document.OfficeDocument
```

サービスは、次の 2 つの主要なインターフェースを提供しています。

・

```
com.sun.star.frame.XStorable
```

インターフェースは、ドキュメントの保存を担当します。

・

```
com.sun.star.view.XPrintable
```

インターフェースは、ドキュメント印刷用のメソッドを含みます。

ドキュメントの保存とエクスポート

OpenOffice.org ドキュメントの保存処理は、ドキュメントオブジェクトを通じて直接実行されます。このような処理を行うには、

```
com.sun.star.frame.XStorable
```

インターフェースの store メソッドを、次のような形で使用します。

```
Doc.store()
```

このメソッドは、ドキュメントへのメモリ割り当てがすでに終了している場合にのみ機能します。ただしこの条件は、新規ドキュメントの場合には該当しません。そのような場合は、storeAsURL メソッドを使用します。このメソッドは

```
com.sun.star.frame.XStorable
```

でも定義されており、次のようにドキュメントの保存位置を指定する際に使用できます。

```
Dim URL As String
Dim Dummy()

Url = "file:///C:/test3.odt"
Doc.storeAsURL(URL, Dummy())
```

これらのメソッドの他に

```
com.sun.star.frame.XStorable
```

には、ドキュメントの保存に関するいくつかのサポート用メソッドが用意されています。以下にその属性を示します。

hasLocation()

ドキュメントに既に URL が割り当てられているかどうかを指定します。

isReadonly()

ドキュメントが読み取り専用であるかを指定します。

isModified()

前回保存時からドキュメントが変更されているかを指定します。

ドキュメント保存用のプログラムコードを作成する際には、これらのオプションを利用することで、変更箇所があるドキュメントのみ保存処理を進めたり、名前が指定されていない場合のみファイル名の指定を要求するなどの機能を組み込むことができます。

```
If (Doc.isModified) Then
  If (Doc.hasLocation And (Not Doc.isReadOnly)) Then
    Doc.store()
  Else
    Doc.storeAsURL(URL, Dummy())
  End If
End If
```

上記のサンプルコードでは、まず最初に、前回保存時からドキュメントが変更されているかを確認しています。そして変更箇所がある場合のみ、残りの保存処理を継続します。次に、ドキュメントの URL 指定の有無と、読み取り専用でないことを確認し、双方の条件を満たしている場合のみ、既存の URL に対してドキュメントを保存します。URL 指定が完了していない場合、あるいは読み取り専用で開かれている場合は、URL を新規指定して保存する処理を行います。

storeAsURL メソッドのオプション

loadComponentFromURL メソッドと同様に、storeAsURL メソッドも、PropertyValue データフィールドを使ったいくつかのオプションを指定できます。これらのオプションは、OpenOffice.org によるドキュメント保存に関する指定を行います。storeAsURL メソッドで指定できるのは、次のオプションです。

CharacterSet (String)

ドキュメントの基になっている文字セットを定義します。

FilterName (String)

loadComponentFromURL 関数用の特殊なフィルタを指定します。指定可能なフィルタ名は、ファイル \share\config\registry\instance\org\openoffice\office\TypeDetection.xml で定義されています。

FilterOptions (String)

フィルタ用の追加オプションを定義します。

Overwrite (Boolean)

既存ファイルを警告なしで上書きするかを指定します。

Password (String)

保護されたファイルのパスワードを転送します。

Unpacked (Boolean)

ドキュメントを (圧縮しないで) サブディレクトリに保存します。

次のサンプルコードは、storeAsURL での Overwrite オプションの使用例です。

```
Dim Doc As Object
Dim FileProperties(0) As New com.sun.star.beans.PropertyValue
Dim Url As String
' ... Initialize Doc

Url = "file:///c:/test3.odt"
FileProperties(0).Name = "Overwrite"
FileProperties(0).Value = True
Doc.storeAsURL(sUrl, mFileProperties())
```

このサンプルコードでは、同名のファイルが存在した場合、指定ファイル名で Doc を保存します。

ドキュメントの印刷

ドキュメントの保存と同様に、ドキュメントの印刷も該当オブジェクトから直接行えます。このような操作には、

```
com.sun.star.view.Xprintable
```

インターフェースの Print メソッドを使用します。次のサンプルコードは、print メソッドを呼び出す際の基本パターンです。

```
Dim Dummy()

Doc.print(Dummy())
```

loadComponentFromURL メソッドの場合と同様に、パラメータの Dummy には PropertyValue データフィールドを使用し、この値を通じて OpenOffice.org の印刷オプションを指定します。

print メソッドのオプション

print メソッドのパラメータは、PropertyValue データフィールドの形で与えますが、その値には OpenOffice.org の印刷用ダイアログの項目に対応した内容を指定します。

CopyCount (Integer)

印刷する部数を指定します。

FileName (String)

指定したファイルのドキュメントを印刷します。

Collate (Boolean)

複数部数の印刷でページの丁合をとるよう、プリンタに指示します。

Sort (Boolean)

複数の部数を印刷するときに (CopyCount > 1)、ページをソートします。

Pages (String)

印刷するページを指定します (表記規則は印刷用ダイアログのものと同じ)。

次のサンプルコードでは、Pages オプションを使って特定ページのみを印刷します。

```
Dim Doc As Object
Dim PrintProperties(0) As New com.sun.star.beans.PropertyValue

PrintProperties(0).Name="Pages"
PrintProperties(0).Value="1-3; 7; 9"
Doc.print(PrintProperties())
```

プリンタの選択と設定

プリンタの選択には、

```
com.sun.star.view.XPrintable
```

インターフェースの Printer プロパティーを使用します。このプロパティーには、PropertyValue データフィールドを使って、次の内容を設定します。

Name (String)

プリンタの名前を指定します。

PaperOrientation (Enum)

用紙の向きを指定します (

```
com.sun.star.view.PaperOrientation.PORTRAIT
```

は縦、

```
com.sun.star.view.PaperOrientation.LANDSCAPE
```

は横)。

PaperFormat (Enum)

用紙の形式を指定します (たとえば、

```
com.sun.star.view.PaperFormat.A4
```

は DIN A4、

```
com.sun.star.view.PaperFormat.Letter
```

は US レター)。

PaperSize (Size)

用紙サイズを 100 分の 1 ミリ単位で指定します。

次のサンプルコードでは、Printer プロパティーを使ってプリンタを選択し、用紙サイズを設定します。

```
Dim Doc As Object
Dim PrinterProperties(1) As New com.sun.star.beans.PropertyValue
Dim PaperSize As New com.sun.star.awt.Size

PaperSize.Width = 20000   ' corresponds to 20 cm
PaperSize.Height = 20000   ' corresponds to 20 cm
PrinterProperties (0).Name="Name"
PrinterProperties (0).Value="My HP Laserjet"
PrinterProperties (1).Name="PaperSize"
PrinterProperties (1).Value=PaperSize
Doc.Printer = PrinterProperties()
```

ここでは、PaperSize という名前で

```
com.sun.star.awt.Size
```

型のオブジェクトを作成します。用紙サイズの指定には、このタイプのオブジェクトが必要です。さらに、PrinterProperties という名前の 2 つの PropertyValue エントリに対するデータフィールドを作成します。このデータフィールドには、初期化もかねて Printer プロパティーに渡す値を代入します。なお UNO では、プリンタはリアル属性ではなくイミテーション属性として扱われます。

テンプレート

テンプレートとは、さまざまな書式設定情報を 1 つにまとめて、名前を付けたものです。OpenOffice.org アプリケーションでは、各種のテンプレートを利用して、書式設定の操作を簡略化します。テンプレートの登録情報が変更されると、OpenOffice.org ドキュメントの該当セクションも自動的に更新されます。このような機能を利用すると、たとえばテンプレートを変更して、すべてのレベル 1 ヘッダの表示フォントを一括変更する、などの操作が行えます。OpenOffice.org では、ドキュメントの種類ごとに該当するテンプレートが自動判別されます。

OpenOffice.org Writer は次のテンプレートをサポートしています。

・文字テンプレート

・段落テンプレート

・フレームテンプレート

・ページテンプレート

・番号付けテンプレート

OpenOffice.org Calc は次のテンプレートをサポートしています。

・セルテンプレート

・ページテンプレート

OpenOffice.org Impress は次のテンプレートをサポートしています。

・文字要素テンプレート

・プレゼンテーションテンプレート

OpenOffice.org で利用する各種のテンプレートは、

```
com.sun.star.style.StyleFamily
```

サービスに基づいて、StyleFamilies として管理されています。StyleFamilies にはドキュメントオブジェクトを通じてアクセスします。

```
Dim Doc As Object
Dim Sheet As Object
Dim StyleFamilies As Object
Dim CellStyles As Object

Doc = StarDesktop.CurrentComponent
StyleFamilies = Doc.StyleFamilies
CellStyles = StyleFamilies.getByName("CellStyles")
```

このサンプルコードでは、表計算ドキュメントの StyleFamilies 属性を用いて、使用可能なすべてのセルテンプレートの一覧を取得します。

個々のテンプレートには、インデックスを使って直接アクセスできます。

```
Dim Doc As Object
Dim Sheet As Object
Dim StyleFamilies As Object
Dim CellStyles As Object
Dim CellStyle As Object
Dim I As Integer

Doc = StarDesktop.CurrentComponent
StyleFamilies = Doc.StyleFamilies
CellStyles = StyleFamilies.getByName("CellStyles")

For I = 0 To CellStyles.Count - 1
  CellStyle = CellStyles(I)
  MsgBox CellStyle.Name
Next I
```

1 つ前のサンプルコードでは、すべてのセルテンプレートをメッセージボックスに一括表示しましたが、ここではループを使って個別に表示します。

書式設定オプションの詳細

各テンプレートには、独自の書式設定属性が存在します。以下に、主要な書式設定属性とその説明箇所を示します。

・文字属性、

```
com.sun.star.style.CharacterProperties
```

サービス

・段落属性、

```
com.sun.star.text.Paragraph
```

サービス

・セル属性、

```
com.sun.star.table.CellProperties
```

サービス

・ページ属性、

```
com.sun.star.style.PageProperties
```

サービス

・文字要素属性、さまざまなサービス

書式設定属性は、ここでの説明に用いたアプリケーションにのみ限定されるわけでなく、より広範に利用されます。たとえば、「スプレッドシート」で説明するページ属性の大部分は、OpenOffice.org Calc だけでなく OpenOffice.org Writer でも使用できます。

テンプレートの使用法に関する詳細情報については、「文書ドキュメント」の「文字と段落属性のデフォルト値」を参照してください。

文書ドキュメント

文書ドキュメントには、プレーンテキストだけでなく、書式設定に関する情報も記録されます。通常このような文字装飾は、テキストの各所に施されます。またテーブル (表) を作成するような場合、このような構造はより複雑化します。これは単なる 1 次元の文字列情報ではなく、2 次元フィールドと成るためです。更に現在のワードプロセッサでは、図形描画オブジェクトを始め、テキスト枠やその他各種のオブジェクトを文章中に配置できるようになっています。またこれらのオブジェクトの配置位置も、テキストの間に限定されるものではなく、ページ上の任意の位置にレイアウトされる場合もあります。

本章では、文書ドキュメントで使われる主なインターフェースとサービスを取り上げます。

・文書ドキュメントの構造

・文書ドキュメントの編集

・テキスト以外のオブジェクト

最初の節では、文書ドキュメントの構造について説明し、OpenOffice.org ドキュメント上で繰り返し行うタイプの処理を OpenOffice.org Basic プログラムで自動化する方法を説明します。ここでは特に、段落や段落部位およびこれらの書式設定に焦点を当てます。

次の節では、文書ドキュメントでの処理の効率化を検討します。OpenOffice.org には `TextCursor` オブジェクトなど各種のサポートオブジェクトが用意されており、これらを利用することで、最初の節で説明した以上の処理を行えるようになります。

3 番目の節では、テキスト以外の処理を説明します。対象となるのは、テーブル、テキスト枠、テキストフィールド、テキストマーク、コンテンツディレクトリなどです。

ドキュメントの作成、オープン、保存、印刷法については、文書ドキュメントだけでなく他のドキュメントにも共通する内容であるため、「ドキュメントの使い方」で説明しています。

文書ドキュメントの構造

文書ドキュメントには、次の 4 種類の情報が記録されています。

・テキスト本体

・文字、段落、ページを対象とした書式設定用テンプレート

・テーブル、画像、図形描画オブジェクトなど、テキスト以外の要素

・文書ドキュメント全体の設定

本節では特に、テキストおよびその書式設定オプションについて説明します。

段落と段落部位

文書ドキュメントは、段落の集合であるとも言えます。しかしこのような段落は、個別的な名前やインデックスが付けられているわけでもないため、直接アクセスする方法はありません。その代わり段落へのアクセスは、「API について」で説明されている Enumeration オブジェクトを用いた順次アクセスが行えます。段落を編集する場合は、このような機能を利用します。

ただし Enumeration オブジェクトで取得される対象には、段落だけでなくテーブルも含まれるので、注意が必要です (OpenOffice.org Writer では、テーブルを特殊な段落として処理)。このため、取得したオブジェクトへアクセスする際には、そのオブジェクトが段落を示す

```
com.sun.star.text.Paragraph
```

サービスをサポートしているのか、テーブルを示す

```
com.sun.star.text.TextTable
```

サービスをサポートしているのかを確認する必要があります。

次のサンプルコードでは、ループを使って文書ドキュメントの内容に順次アクセスして、各インスタンスごとに該当オブジェクトが段落であるかテーブルであるかを、メッセージ表示します。

```
Dim Doc As Object
Dim Enum As Object
Dim TextElement As Object

' Create document object
Doc = StarDesktop.CurrentComponent
' Create enumeration object
Enum = Doc.Text.createEnumeration
' loop over all text elements

While Enum.hasMoreElements
  TextElement = Enum.nextElement

  If TextElement.supportsService("com.sun.star.text.TextTable") Then
    MsgBox "The current block contains a table."
  End If

  If TextElement.supportsService("com.sun.star.text.Paragraph") Then
    MsgBox "The current block contains a paragraph."
  End If

Wend
```

このサンプルコードでは、Doc というドキュメントオブジェクトを作成して、現在の OpenOffice.org ドキュメントを参照しています。次にこの Doc オブジェクトを使って Enumeration オブジェクトを作成して、テキストの各部 (段落およびテーブル) に順次アクセスして、TextElement というオブジェクトに現在の要素を取得します。そして supportsService メソッドを使って、TextElement の内容が段落かテーブルかを判定しています。

段落

段落の内容にアクセスするには、

```
com.sun.star.text.Paragraph
```

サービスを使用します。そして段落中のテキストの取得および変更には、String 属性を使用します。

```
Dim Doc As Object
Dim Enum As Object
Dim TextElement As Object

Doc = StarDesktop.CurrentComponent
Enum = Doc.Text.createEnumeration

While Enum.hasMoreElements
  TextElement = Enum.nextElement

  If TextElement.supportsService("com.sun.star.text.Paragraph") Then
    TextElement.String = Replace(TextElement.String, "you", "U")
    TextElement.String = Replace(TextElement.String, "too", "2")
    TextElement.String = Replace(TextElement.String, "for", "4")
  End If

Wend
```

このサンプルコードでは、現在の文書ドキュメントを開いて、その内容に Enumeration オブジェクトを用いて順次アクセスしています。そして各段落の `TextElement.String` 属性を使用して、単語 `you`、`too` と `for` をそれぞれ文字 `U`、`2` と `4` に置き換えていますなお、ここで使っている `Replace` という関数は、OpenOffice.org Basic に用意されているものではありません。これは、「検索と置換」で説明されている実行例です。

ここで説明した段落テキストへのアクセス手順は、VBA の場合の Paragraphs によるリスト作成に該当するもので、これらは `Range` および `Document` オブジェクトから使用できます。VBA の場合、段落へのアクセスは番号指定で行えますが (たとえば `Paragraph(1)` など)、OpenOffice.org Basic 場合は、ここで説明した `Enumeration` オブジェクトを使用する必要があります。

VBA の `Characters, Sentences` および `Words` リストに直接該当する機能は、OpenOffice.org Basic には用意されていません。ただし、`TextCursor` の機能を利用することで、文字、段落、ワード単位での操作を行えます。

段落部位

上記のサンプルコードを実行すると、テキストの置換は成功しても、書式設定が崩れるような場合があります。

このような現象は、個々の段落が独立したサブオブジェクトから構成されていることに原因があります。これら各サブオブジェクトは、独自の書式設定情報を保持しています。たとえば、中央部の 1 つの単語だけに太字の書式設定が行われた段落がある場合、OpenOffice.org はこの段落を、太字テキストよりも前の部分、太字テキストの部分、太字テキストよりも後の通常テキストの部分という、3 つの段落部位として扱います。

この段落のテキストを `String` 属性を使って変更する場合、OpenOffice.org は該当する段落部位をいったん削除してから新規に段落部位を挿入するという方法で処理を進めます。この際に、変更前の書式設定は失われてしまいます。

このような現象を回避するには、段落全体ではなく該当する段落部位にアクセスするようにします。このような処理を行うため、各段落には Enumeration オブジェクトが用意されています。次のサンプルコードは、先のサンプルコードと同様の置換処理を実行しますが、ここではループを二重にすることで、文書ドキュメント内のすべての段落および、各段落を構成するすべての段落部位にアクセスするようにしています。

```
Dim Doc As Object
Dim Enum1 As Object
Dim Enum2 As Object
Dim TextElement As Object
Dim TextPortion As Object

Doc = StarDesktop.CurrentComponent
Enum1 = Doc.Text.createEnumeration

' loop over all paragraphs
While Enum1.hasMoreElements
  TextElement = Enum1.nextElement

  If TextElement.supportsService("com.sun.star.text.Paragraph") Then
    Enum2 = TextElement.createEnumeration
    ' loop over all sub-paragraphs

    While Enum2.hasMoreElements
      TextPortion = Enum2.nextElement
      MsgBox "'" & TextPortion.String & "'"
      TextPortion.String = Replace(TextPortion.String, "you", "U")
      TextPortion.String = Replace(TextPortion.String, "too", "2")
      TextPortion.String = Replace(TextPortion.String, "for", "4")
    Wend

  End If
Wend
```

このサンプルコードでは、二重ループを使って文書ドキュメントの内容に順次アクセスしています。外周部のループは、段落単位のアクセスを担当しています。そして内周部のループは、各段落を構成する段落部位のアクセスを担当しています。次のサンプルコードでは、文字列の String 属性を使用して、各段落部位の内容を変更します。段落については、先のサンプルコードの場合と同じです。ただしこのサンプルコードでは、個々の段落部位ごとに変更するようにしているので、テキストの置換を行なっても書式設定情報は保持されます。

書式設定

テキストの書式設定を行う方法は、複数あります。一番簡単な方法は、書式設定用属性をテキストに直接指定することです。このような方式は、ダイレクトフォーマッティングと呼ばれます。通常このようなダイレクトフォーマッティングが使われるのは、マウス操作による逐次的な書式設定が行えるような、比較的小さめのドキュメントです。より具体的に言えば、ユーザーがマウスを直接操作してテキスト内の特定の単語を太字にしたり、1 行だけを中央揃えにする場合が、これに該当します。

書式設定は、ダイレクトフォーマッティング以外にも、テンプレートを使って行うことができます。このような方式を、インダイレクトフォーマッティングと呼びます。インダイレクトフォーマッティングは、あらかじめ定義しておいたテンプレートを、該当テキストに適用することで実施します。このようなテキストの書式設定を後から変更する場合は、テンプレートを変更するだけで済みます。OpenOffice.org はテンプレートへの変更を、該当するすべてのテキストに対して一括適用します。

通常 VBA では、書式設定用属性は複数のサブオブジェクトに分散しています (たとえば `Range.Font`、`Range.Borders`、`Range.Shading`、`Range.ParagraphFormat` など) これらの属性はカスケード式に指定します (たとえば `Range.Font.AllCaps`)。これに対して OpenOffice.org Basic では、該当オブジェクトに対して書式設定用属性を直接指定できます (`TextCursor`、`Paragraph` など)。OpenOffice.org に用意されている文字および段落関係の属性については、次の 2 つの節で説明しています。

個々のオブジェクト (`Paragraph`、`TextCursor` など) に書式設定用の属性が用意されており、これらの指定を直接行えるようになっています。

文字属性

ここでは、個々の文字に対する書式設定属性を、文字属性と総称します。このようなものには、太字やフォントなどの書体指定が該当します。文字属性を使えるオブジェクトは、

```
com.sun.star.style.CharacterProperties
```

サービスをサポートしているものに限られます。OpenOffice.org には、これに該当する各種のサービスが存在します。たとえば先に説明した、段落に対する

```
com.sun.star.text.Paragraph
```

サービスや、段落部位に対する

```
com.sun.star.text.TextPortion
```

サービスなども、その中に含まれます。

```
com.sun.star.style.CharacterProperties
```

サービスは、何らかのインターフェースを提供するものではなく、文字属性の指定と取得を行う各種の属性を提供します。すべての文字属性のリストについては、OpenOffice.org の『API reference』を参照してください。以下に主要な属性を示します。

CharFontName (String)

選択したフォントの種類の名前。

CharColor (Long)

テキストの色。

CharHeight (Float)

ポイント単位で指定した文字の高さ。

CharUnderline (Constant group)

下線の種類 (

```
com.sun.star.awt.FontUnderline
```

に定められた定数)。

CharWeight (Constant group)

フォントの太さ (

```
com.sun.star.awt.FontWeight
```

に定められた定数)。

CharBackColor (Long)

背景色。

CharKeepTogether (Boolean)

自動行ブレークを抑制する指定。

CharStyleName (String)

文字テンプレートの名前。

段落属性

個々の文字に対してではなく、段落全体に対して施される書式設定の情報は、段落属性と総称されます。このような情報としては、用紙と段落の間の余白や行間の大きさなどが該当します。段落属性は、

```
com.sun.star.style.ParagraphProperties
```

サービスを通じて使用します。

段落属性は、各種のオブジェクトで利用できます。

```
com.sun.star.text.Paragraph
```

サービスをサポートするすべてのオブジェクトは、

```
com.sun.star.style.ParagraphProperties
```

の段落属性もサポートしています。

すべての段落属性のリストについては、『OpenOffice.org API reference』を参照してください。以下に主要な段落属性を示します。

ParaAdjust (enum)

垂直の文字の方向 (

```
com.sun.star.style.ParagraphAdjust
```

に定められた定数)。

ParaLineSpacing (struct)

行間 (

```
com.sun.star.style.LineSpacing
```

に定められた構造体)。

ParaBackColor (Long)

背景色。

ParaLeftMargin (Long)

100 分の 1 ミリ単位で指定した左マージン。

ParaRightMargin (Long)

100 分の 1 ミリ単位で指定した右マージン。

ParaTopMargin (Long)

100 分の 1 ミリ単位で指定した上マージン。

ParaBottomMargin (Long)

100 分の 1 ミリ単位で指定した下マージン。

ParaTabStops (Array of struct)

タブの種類と位置 (Typs 構造

```
com.sun.star.style.TabStop
```

を持つ配列)。

ParaStyleName (String)

段落テンプレートの名前。

例: HTML の簡易エクスポート

次のサンプルコードは、書式設定情報の操作例です。ここでは、文書ドキュメントを順次読み取り、HTML 形式ファイルへの簡易的な変換を行なっています。基本的な変換処理としては、個々の段落の先頭に HTML タグの <P> を付加します。同様に、太字指定のされた段落部位には HTML タグの <B> を付けるよう処理しています。

```
Dim FileNo As Integer, Filename As String, CurLine As String
Dim Doc As Object
Dim Enum1 As Object, Enum2 As Object
Dim TextElement As Object, TextPortion As Object

Filename = "c:¥text.html"
FileNo = Freefile
Open Filename For Output As #FileNo
Print #FileNo, "<HTML><BODY>"
Doc = StarDesktop.CurrentComponent
Enum1 = Doc.Text.createEnumeration

' loop over all paragraphs
While Enum1.hasMoreElements
 TextElement = Enum1.nextElement

 If TextElement.supportsService("com.sun.star.text.Paragraph") Then
   Enum2 = TextElement.createEnumeration
   CurLine = "<P>"

    ' loop over all paragraph portions
   While Enum2.hasMoreElements
     TextPortion = Enum2.nextElement

     If TextPortion.CharWeight = com.sun.star.awt.FontWeight.BOLD THEN
       CurLine = CurLine & "<B>" & TextPortion.String & "</B>"
      Else
        CurLine = CurLine & TextPortion.String
      End If

     Wend

    ' output the line
     CurLine = CurLine & "</P>"
     Print #FileNo, CurLine
   End If

 Wend

' write HTML footer
Print #FileNo, "</BODY></HTML>"
Close #FileNo
```

このサンプルコードの基本構造は、先に説明した段落部位へアクセスするサンプルコードと同じものです。今回追加されたものは、HTML ファイルの書き出し処理および、テキストが太字であるかをチェックして該当する段落部位を HTML タグで囲むよう処理するコードです。

文字と段落属性のデフォルト値

ダイレクトフォーマッティングは、常にインダイレクトフォーマッティングに優先します。これはつまりテンプレートによる書式設定よりも、テキストへの直接操作による書式設定の方が優先されるということになります。

ドキュメントの特定セクションが、ダイレクトフォーマッティングされたのか、インダイレクトフォーマッティングされたのかは、簡単には確認できません。OpenOffice.org のオブジェクトバーには、本文テキストに設定されたフォント、太さ、サイズなどの属性が表示されます。しかし、このような書式設定がテンプレートによるものか、直接設定したものかについては表示されません。

特定の属性については、OpenOffice.org Basic の `getPropertyState` メソッドを使うことで、どのように書式設定されているかを確認することができます。この場合に渡すパラメータには属性名を指定し、戻り値としては書式設定の出所を示す定数が返されます。このような定数としては、

```
com.sun.star.beans.PropertyState
```

に定義された次の値が使われます。

```
com.sun.star.beans.PropertyState.DIRECT_VALUE
```

テキストへの直接操作により設定された属性 (ダイレクトフォーマッティング)

```
com.sun.star.beans.PropertyState.DEFAULT_VALUE
```

テンプレートにより設定された属性 (インダイレクトフォーマッティング)

```
com.sun.star.beans.PropertyState.AMBIGUOUS_VALUE
```

属性が不明確。このような状況が発生するのは、たとえばプレーンテキストと太字テキストの部分が混在する段落で、太字テキストの部分を調べようとした場合などです。

次のサンプルコードでは、書式設定用属性が OpenOffice.org 内でどのように編集されているかを調べます。ここでは、個々の段落部位にダイレクトフォーマッティングにより太字にされたものがあるかをチェックします。そして該当する段落部位があると、`setPropertyToDefault` メソッドを用いてダイレクトフォーマッティングによる書式指定を解除して、`MyBold` という文字テンプレートを適用させています。

```
Dim Doc As Object
Dim Enum1 As Object
Dim Enum2 As Object
Dim TextElement As Object
Dim TextPortion As Object

Doc = StarDesktop.CurrentComponent
Enum1 = Doc.Text.createEnumeration

' loop over all paragraphs
While Enum1.hasMoreElements
  TextElement = Enum1.nextElement

 If TextElement.supportsService("com.sun.star.text.Paragraph") Then
    Enum2 = TextElement.createEnumeration
    ' loop over all paragraph portions

 While Enum2.hasMoreElements
      TextPortion = Enum2.nextElement

      If TextPortion.CharWeight = _
        com.sun.star.awt.FontWeight.BOLD AND _
        TextPortion.getPropertyState("CharWeight") = _
        com.sun.star.beans.PropertyState.DIRECT_VALUE Then
        TextPortion.setPropertyToDefault("CharWeight")
        TextPortion.CharStyleName = "MyBold"
      End If
    Wend
  End If
Wend
```

文書ドキュメントの編集

文書ドキュメントの編集に関しては、すでに前節で、段落および段落部位へのアクセスを行う

```
com.sun.star.text.TextPortion
```

と

```
com.sun.star.text.Paragraph
```

サービスを中心に説明しました。これらのサービスの利用が適しているのは、各ループごとに 1 度ずつテキストの編集を行うタイプの作業です。しかし、このような方式では対処し得ない処理も多く存在します。OpenOffice.org に用意されている

```
com.sun.star.text.TextCursor
```

サービスは、ドキュメントを逆方向に遡って処理したり、センテンスや単語単位で操作するといった、より複雑な処理を行うためのもので、`TextPortions` を使うよりもこのような操作に適しています。

TextCursor

OpenOffice.org API の `TextCursor` は、OpenOffice.org ドキュメント上の操作で表示されるカーソルに該当するものです。これを使用すると、文書ドキュメント上の特定位置を操作対象に指定して、コマンド指定による各種の選択処理を行うことができます。ただし、このような OpenOffice.org Basic の `TextCursor` オブジェクトを、通常のカーソルと混同してはいけません。両者は、本質的に異なるものです。

VBA の Range オブジェクトの機能に該当するものは、OpenOffice.org の TextCursor オブジェクトであって、同じ名前を持つ OpenOffice.org の Range オブジェクトに相当するものではありません。

たとえば、OpenOffice.org の TextCursor オブジェクトはドキュメント内の移動やテキストの変更という機能を担っていますが、このような処理を VBA では Range オブジェクトで処理します (MoveStart、MoveEnd、InsertBefore、InsertAfter など)。OpenOffice.org の TextCursor オブジェクトに該当する機能は、次の節で説明しています。

テキスト内の移動

OpenOffice.org Basic の TextCursor オブジェクトは、文書ドキュメント上に表示される通常のカーソルとは異なるものです。このため、TextCursor オブジェクトの表示位置をプログラム制御で変更しても、通常のカーソルは何の影響も受けません。また TextCursor オブジェクトは、同一オブジェクト上の異なる位置に複数オープンすることが可能で、相互に独立した形で個別制御できます。

TextCursor オブジェクトを作成するには、createTextCursor 呼び出しを使用します。

```
Dim Doc As Object
Dim Cursor As Object

Doc = StarDesktop.CurrentComponent
Cursor = TextDocument.Text.createTextCursor()
```

ここで作成した Cursor というオブジェクトは、com.sun.star.text.TextCursor サービスをサポートしており、文書ドキュメント内のテキストを移動するための各種メソッドを利用できます。たとえば次のサンプルコードは、最初に TextCursor を 10 文字分左に移動してから、3 文字分右に移動します。

```
Cursor.goLeft(10, False)
Cursor.goRight(3, False)
```

TextCursor は、選択範囲の強調表示にも使用できます。このような処理は、マウスでテキストの一部を選択して強調させる操作に相当します。上記のサンプルコードでパラメータとして渡した False は、オブジェクトの移動に伴う通過部分を強調表示させるかどうかの指定です。このような TextCursor の挙動は、次のサンプルコードと対比させると分かりやすいでしょう。

```
Cursor.goLeft(10, False)
Cursor.goRight(3, True)
```

この場合も最初に 10 文字分左に移動させてますが、このときは強調表示しないで、次に 3 文字分右に移動させる際には、強調表示するようにしています。つまりこのサンプルコードで TextCursor が強調表示するのは、最初の位置から 8 番目から 10 番目の文字までとなります。

以下に、

```
com.sun.star.text.TextCursor
```

サービスの移動操作に用いる主要なメソッドを示します。

goLeft (Count, Expand)

Count 分の文字だけ左へ移動します。

goRight (Count, Expand)

Count 分の文字だけ右へ移動します。

gotoStart (Expand)

文書ドキュメントの先頭に移動します。

gotoEnd (Expand))

文書ドキュメントの末尾に移動します。

gotoRange (TextRange, Expand)

TextRange の指定オブジェクトに移動します。

gotoStartOfWord (Expand)

現在位置にあるワード (単語) の先頭に移動します。

gotoEndOfWord (Expand)

現在位置にあるワードの末尾に移動します。

gotoNextWord (Expand)

後ろにあるワードの先頭に移動します。

gotoPreviousWord (Expand)

前にあるワードの先頭に移動します。

isStartOfWord ()

TextCursor の位置がワードの先頭であれば True を返します。

isEndOfWord ()

TextCursor の位置がワードの末尾であれば True を返します。

gotoStartOfSentence (Expand)

現在位置にあるセンテンス (文章) の先頭に移動します。

gotoEndOfSentence (Expand)

現在位置にあるセンテンスの末尾に移動します。

gotoNextSentence (Expand)

後ろにあるセンテンスの先頭に移動します。

gotoPreviousSentence (Expand)

前にあるセンテンスの先頭に移動します。

isStartOfSentence ()

TextCursor の位置がセンテンスの先頭であれば True を返します。

isEndOfSentence ()

TextCursor の位置がセンテンスの末尾であれば True を返します。

gotoStartOfParagraph (Expand)

現在位置にあるパラグラフ (段落) の先頭に移動します。

gotoEndOfParagraph (Expand)

現在位置にあるパラグラフの末尾に移動します。

gotoNextParagraph (Expand)

後ろにあるパラグラフの先頭に移動します。

gotoPreviousParagraph (Expand))

前にあるパラグラフの先頭に移動します。

isStartOfParagraph ()

TextCursor の位置がパラグラフの先頭であれば True を返します。

isEndOfParagraph ()

TextCursor の位置がパラグラフの末尾であれば True を返します。

個々のセンテンス (文章) 間の区切りは、センテンス末を示す記号を基準に処理されます。たとえばピリオドは、このようなセンテンス末の記号として認識されます。

Expand パラメータは、オブジェクトの移動に伴う通過部分を強調表示させるかを、ブール値を用いて指定します。またこれらの移動操作用メソッドは、移動処理に成功したか、あるいは移動先となるテキスト位置の不在により処理が中断されたかを、戻り値として返します。

以下に、

```
com.sun.star.text.TextCursor
```

サービスをサポートし、TextCursor を使用して強調表示した範囲を編集する代表的なメソッドを示します。

collapseToStart ()

TextCursor の位置を強調表示部分の先頭に移動し、強調表示を解除します。

collapseToEnd ()

TextCursor の位置を強調表示部分の末尾に移動し、強調表示を解除します。

isCollapsed ()

TextCursor による強調表示部分がなければ、True を返します。

TextCursor によるテキストの書式設定

```
com.sun.star.text.TextCursor
```

サービスは、本章の冒頭で説明した文字および段落関係の属性をすべてサポートしています。

次のサンプルコードは、TextCursor を用いたこれらの使用例です。ここでは、ドキュメントの内容に順次アクセスして、各センテンスの先頭ワードに対して太字の書式を設定します。

```
Dim Doc As Object
Dim Cursor As Object
Dim Proceed As Boolean

Doc = StarDesktop.CurrentComponent
Cursor = Doc.Text.createTextCursor

Do
  Cursor.gotoEndOfWord(True)
  Cursor.CharWeight = com.sun.star.awt.FontWeight.BOLD
  Proceed = Cursor.gotoNextSentence(False)
  Cursor.gotoNextWord(False)
Loop While Proceed
```

このサンプルコードではまず、テキストの取得に使うドキュメントオブジェクトを作成しています。そして、ループを使ってセンテンス単位でテキストを読み取り、先頭ワードを強調表示して、太字の書式を設定します。

テキストの取得と変更

TextCursor による強調表示部分に対しては、TextCursor オブジェクトの String 属性によるテキスト操作が可能です。次のサンプルコードでは、String 属性を用いて、センテンスの先頭ワードをメッセージボックスに表示します。

```
Dim Doc As Object
Dim Cursor As Object
Dim Proceed As Boolean

Doc = StarDesktop.CurrentComponent
Cursor = Doc.Text.createTextCursor

Do
  Cursor.gotoEndOfWord(True)
  MsgBox Cursor.String
  Proceed = Cursor.gotoNextSentence(False)
  Cursor.gotoNextWord(False)
Loop While Proceed
```

このような String 属性を使った手法は、センテンスの先頭ワードを変更する場合にも利用できます。

```
Dim Doc As Object
Dim Cursor As Object
Dim Proceed As Boolean

Doc = StarDesktop.CurrentComponent
Cursor = Doc.Text.createTextCursor

Do
  Cursor.gotoEndOfWord(True)
  Cursor.String = "Ups"
  Proceed = Cursor.gotoNextSentence(False)
  Cursor.gotoNextWord(False)
Loop While Proceed
```

TextCursor によりテキストを強調表示した状態で、String 属性に文字列を代入すると、該当部のテキストが新規の文字列に置き換えられます。強調表示されていない場合、文字列は TextCursor 位置に挿入されます。

制御コードの挿入

状況によっては、ドキュメント上に表示される実際のテキストではなく、表示形態の方を変更したい場合もあります。OpenOffice.org は、このような表示形態の一部を制御コードを用いて処理します。これらの制御コードは、テキスト内に挿入することで、その表示形態を整えます。個々の制御コードは

```
com.sun.star.text.ControlCharacter
```

で定数として定義されています。以下に、OpenOffice.org で使用可能な制御コードを示します。

PARAGRAPH_BREAK

段落区切り。

LINE_BREAK

段落内の行ブレーク。

SOFT_HYPHEN

ハイフネーションが可能な位置。

HARD_HYPHEN

ハイフネーションを強制する位置。

HARD_SPACE

ハードスペーステキストの行末調整に影響されないスペース (ハードスペース)。

制御コードを挿入するには、挿入位置だけではなく、該当するドキュメントオブジェクトも指定する必要があります。次のサンプルコードでは、20 番目の文字の次に段落区切りを挿入します。

```
Dim Doc As Object
Dim Cursor As Object
Dim Proceed As Boolean

Doc = StarDesktop.CurrentComponent
Cursor = Doc.Text.createTextCursor
Cursor.goRight(20, False)
Doc.Text.insertControlCharacter(Cursor, _
    com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK, False)
```

insertControlCharacter メソッドの呼び出しで、パラメータに False を指定しているのは、挿入処理後も TextCursor による強調表示部分を保持させるためです。ここでパラメータに True を指定すると、insertControlCharacter は、該当部のテキストを置き換えます。

テキスト部位の検索

使用頻度の高い操作として、文書ドキュメント内にある特定の文字列を検索して、その位置にあるテキストを編集するという処理があります。このような処理を行うため、すべての OpenOffice.org ドキュメントには特殊なインターフェースが用意されていますが、このインターフェースによる検索を行う際は、SearchDescriptor と呼ばれるオブジェクトを事前に作成しておく必要があります。これにより、ドキュメントの検索する対象が OpenOffice.org に指定されます。SearchDescriptor は com.sun.star.util. SearchDescriptor サービスをサポートしたオブジェクトで、次のサンプルコードのように createSearchDescriptor メソッドを用いて作成します。

```
Dim SearchDesc As Object
SearchDesc = Doc.createSearchDescriptor
```

作成した SearchDescriptor に対しては、次のようにして検索するテキストを指定します。

```
SearchDesc.searchString=”any text”
```

このように SearchDescriptor は、通常の OpenOffice.org 操作で使う検索ダイアログに相当する機能を担っています。また、検索ダイアログに各種の検索設定があるように、同様の設定を SearchDescriptor オブジェクトに対しても指定できます。

このような属性は、

```
com.sun.star.util.SearchDescriptor
```

サービスに用意されています。

SearchBackwards (Boolean)

テキストを前方ではなく後方に向かって検索する指定。

SearchCaseSensitive (Boolean)

検索をする際に大文字と小文字を区別する指定。

SearchRegularExpression (Boolean)

検索式を正規表現として扱う指定。

SearchStyles (Boolean)

指定した段落テンプレートを検索する指定。

SearchWords (Boolean)

完全なワードのみを検索する指定。

OpenOffice.org Basic では、OpenOffice.org SearchSimilarity (いわゆる「ファジーマッチ」) 機能を利用できます。この機能を使うと、指定文字列と類似した文字列を OpenOffice.org に検索させることができます。このような検索を行う際には、オリジナルの検索文字列に対して追加、削除、変更可能な文字数をそれぞれ指定できます。このような指定には、以下に示す com.sun.star.util.SearchDescriptor サービスの関連属性を使用します。

SearchSimilarity (Boolean)

類似検索を実行する指定。

SearchSimilarityAdd (Short)

類似検索での追加可能な文字数の指定。

SearchSimilarityExchange (Short)

類似検索での置換可能な文字数の指定。

SearchSimilarityRemove (Short)

類似検索での削除可能な文字数の指定。

SearchSimilarityRelax (Boolean)

類似検索の変動規則を適用する指定。

SearchDescriptor に関する必要な指定の終了後、文書ドキュメントに対する検索を実行します。このような処理には、OpenOffice.org に用意されている findFirst および findNext メソッドを使用します。

```
Found = Doc.findFirst (SearchDesc)

Do While Found
  ' Suchergebnis bearbeiten
  Found = Doc.findNext( Found.End, Search)
Loop
```

このサンプルコードでは、ループを使って検索該当箇所をすべて探し、検索にヒットしたテキストを参照する TextRange オブジェクトを取得しています。

例：類似検索

次のサンプルコードは、「turnover」という単語を検索して、該当箇所に太字の書式を設定します。ここでは類似検索を用いて、「turnover」に完全に一致するものだけでなく、複数形の「turnovers」を始め「turnover's」なども検索にヒットするようにしています。なお類似性の度合いとしては、オリジナルの検索文字列に対して 2 文字までの違いを許容させることにします。

```
Dim SearchDesc As Object
Dim Doc As Object

Doc = StarDesktop.CurrentComponent
SearchDesc = Doc.createSearchDescriptor
SearchDesc.SearchString=”turnover”
SearchDesc.SearchSimilarity = True
SearchDesc.SearchSimilarityAdd = 2
SearchDesc.SearchSimilarityExchange = 2
SearchDesc.SearchSimilarityRemove = 2
SearchDesc.SearchSimilarityRelax = False
Found = Doc.findFirst (SearchDesc)

Do While Found
  Found.CharWeight = com.sun.star.awt.FontWeight.BOLD
  Found = Doc.findNext( Found.End, Search)
Loop
```

OpenOffice.org で検索と置換の処理を行う場合、その基本的な考えは VBA と同じです。どちらも、検索と置換に使用する属性を 1 つのオブジェクトに収めることにより指定します。そして、このオブジェクトを処理対象のテキスト範囲に対して渡すことにより検索や置換の実際の処理を開始します。ただしこのような補助オブジェクトは、VBA では Range オブジェクトの Find 属性で処理できるのに対して、OpenOffice.org Basic では、ドキュメントオブジェクト側から `createSearchDescriptor` または `createReplaceDescriptor` を呼び出して作成する必要があります。その他、検索用の属性やメソッドにも違いがあります。

従来の OpenOffice.org API 同様に現行の API でも、テキストの検索や置換は、ドキュメントオブジェクトを通じて実行します。ただし従来は、検索オプションの指定などを `SearchSettings` というオブジェクトで行なっていましたが、現在はテキストの置換処理用に用意された `SearchDescriptor` または `ReplaceDescriptor` オブジェクトを使用します。これらのオブジェクトには、検索オプションだけでなく、検索文字列を指定することが可能で、必要であれば置換文字列も格納できます。これらのオプション指定用オブジェクトは、ドキュメントオブジェクトを用いて作成してから、必要なオプション値を代入し、検索メソッド用のパラメータの形でドキュメントオブジェクトに引き渡します。

テキスト部位の置換

検索の場合と同様、OpenOffice.org で行う通常の置換操作も、OpenOffice.org Basic 上で実行できます。置換処理に必要な手順は、検索処理の場合と基本的に同じです。置換の場合も、最初にオプション指定用の特殊オブジェクトを作成します。このオブジェクトは ReplaceDescriptor と呼ばれ、

```
com.sun.star.util.ReplaceDescriptor
```

サービスをサポートしています。これまでの節で説明した SearchDescriptor のすべての属性は、ReplaceDescriptor によってサポートされています。たとえば、置換処理中、大文字と小文字の区別は有効にも無効にもすることができ、類似検索を実行できます。

次のサンプルコードは、ReplaceDescriptors を使用した OpenOffice.org ドキュメント上での検索処理を示します。

```
Dim I As Long
Dim Doc As Object
Dim Replace As Object
Dim BritishWords(5) As String
Dim USWords(5) As String

BritishWords() = Array("colour", "neighbour", "centre", "behaviour", _
   "metre", "through")
USWords() = Array("color", "neighbor", "center", "behavior", _
   "meter", "thru")

Doc = StarDesktop.CurrentComponent
Replace = Doc.createReplaceDescriptor

For O = 0 To 5
  Replace.SearchString = BritishWords(I)
  Replace.ReplaceString = USWords(I)
  Doc.replaceAll(Replace)
Next n
```

検索文字列および置換文字列は、ReplaceDescriptors の SearchString および ReplaceString 属性を使って指定します。実際の置換処理では、ドキュメントオブジェクトの replaceAll メソッドを使用することで、該当文字列を一括置換できます。

例：正規表現による検索と置換

OpenOffice.org の置換機能は、正規表現と併用することで、より複雑な処理を行えるようになります。正規表現とは、通常の固定された検索文字列の代わりに、プレースホルダや特殊記号から成る検索式を用いた、いわゆるパターンマッチングのことです。

OpenOffice.org で使用可能な正規表現の詳細情報については、OpenOffice.org のオンラインヘルプを参照してください。ここでは、いくつかの例を紹介します。

・検索式内のピリオド記号は、任意の文字に一致します。たとえば sh.rt という検索式は、shirt にも short にも一致します。

・検索式内の ^ 記号は、段落の先頭に一致します。たとえば ^Peter という検索式は、Peter が先頭にあるすべての段落に一致します。

・検索式内の $ 記号は、段落の末尾に一致します。たとえば Peter$ という検索式は、Peter が末尾にあるすべての段落に一致します。

・検索式内の * 記号は、直前の文字の任意回数の繰り返しを意味します。これをピリオド記号の次に置くと、任意の文字列に一致するプレースホルダとなります。たとえば temper.*e という検索式は、temperance にも temperature にも一致します。

次のサンプルコードでは、^$ という正規表現を用いて、文書ドキュメント内の空白行を削除します。

```
Dim Doc As Object
Dim Replace As Object
Dim I As Long

Doc = StarDesktop.CurrentComponent
Replace = Doc.createReplaceDescriptor
Replace.SearchRegularExpression = True
Replace.SearchString = "^$"
Replace.ReplaceString = ""

Doc.replaceAll(Replace)
```

テキスト以外のオブジェクト

本章のここまでの説明は、テキストの段落および段落部位のみを扱ってきました。しかし文書ドキュメントには、テキスト以外のオブジェクトも存在します。これに該当するのは、テーブル、テキストフィールド、ディレクトリ、図形描画オブジェクトなどです。このようなオブジェクトは、テキスト内の任意の位置に配置することができます。

基本機能と共に、これらの OpenOffice.org オブジェクトはすべて、

```
com.sun.star.text.TextContent
```

という共通基本サービスをサポートしています。このサービスでは、次の属性が利用できます。

AnchorType (Enum)

TextContent オブジェクトのアンカーの種類を決定します (

```
com.sun.star.text.TextContentAnchorType
```

列挙型に従うデフォルト値)。

AnchorTypes (sequence of Enum)

特殊な TextContent オブジェクトをサポートするすべての AnchorTypes の列挙。

TextWrap (Enum)

TextContent オブジェクト周囲のテキストの折り返しの種類を特定します (

```
com.sun.star.text.WrapTextMode
```

列挙型に従ったデフォルト値)。

TextContent のオブジェクトは、オブジェクトの作成、挿入、削除に関するものなど、いくつかのメソッドを共有しています。

・新しい TextContent オブジェクトを作成するには、ドキュメントオブジェクトの createInstance メソッドを使用します。

・オブジェクトを挿入するには、テキストオブジェクトの insertTextContent メソッドを使用します。

・TextContent オブジェクトを削除するには、removeTextContent メソッドを使用します。

これらのメソッドの使用法については、次の節で例をいくつか説明します。

テーブル

次のサンプルコードでは、先に説明した createInstance メソッドを利用してテーブル (表) を作成します。

```
Dim Doc As Object
Dim Table As Object
Dim Cursor As Object

Doc = StarDesktop.CurrentComponent
Cursor = Doc.Text.createTextCursor()

Table = Doc.createInstance("com.sun.star.text.TextTable")
Table.initialize(5, 4)

Doc.Text.insertTextContent(Cursor, Table, False)
```

Python のサンプル:

```
def inset_img():
   # abstract the objects in varialbes, doc, text and img, note we used
the XSCRIPTCONTEXT
   doc = XSCRIPTCONTEXT.getDocument()
   text = doc.getText()
   cursor = text.createTextCursor()  # not sure
   Table = doc.createInstance(u'com.sun.star.text.TextTable')

   # Verify code
   Table.initialize(5, 4)

   #inser the image in the text area location
   text.insertTextContent(text.getEnd(), Table, False)
```

テーブルを作成した後、`initialize` の呼び出しを使用してテーブルの行数と列数を設定し、`insertTextContent` を使用して文書ドキュメントに挿入します。

前の例を見るとわかるように、`insertTextContent` メソッドには、挿入する `Content` オブジェクト以外に、2 つのパラメータを渡す必要があります。

・挿入位置を指定する `Cursor` オブジェクト。

・`Content` オブジェクトをカーソルの現在の選択範囲と置き換えるか (`True`) または現在の選択テキストの直前に挿入するか (`False`) を指定するブール値。

テーブルを作成して文書ドキュメントに挿入するときは、VBA で使用するものと似たオブジェクトを OpenOffice.org Basic でも使用します。OpenOffice.org Basic ではドキュメントオブジェクトと `TextCursor` オブジェクトを使用し、VBA では `Range` オブジェクトを使用します。VBA ではテーブルの作成と設定は `Document.Tables.Add` メソッドが処理しますが、OpenOffice.org Basic では、前の例のように、テーブルの作成は `createInstance` を使用して行い、初期化とドキュメントへの挿入には `insertTextContent` を使用します。

文書ドキュメントに挿入されたすべてのテーブルの取得は、簡単なループで処理できます。この処理には、文書ドキュメントオブジェクトの `getTextTables()` メソッドを使用します。

```
Dim Doc As Object
Dim TextTables As Object
Dim Table As Object
Dim I As Integer
Doc = StarDesktop.CurrentComponent
TextTables = Doc.getTextTables()
For I = 0 to TextTables.count - 1

   Table = TextTables(I)
   ' Editing table

Next I
```

OpenOffice.org では、ドキュメントオブジェクトの TextTables リストを使用してテキストテーブルにアクセスできます。先に見たサンプルコードでは、テキストテーブルの作成法について説明しました。テキストテーブルへのアクセスについては、次の節で説明します。

テーブルの編集

テーブルは、個別の行で構成されています。そして各行は、いくつかのセルに分割されています。厳密に言うと、OpenOffice.org にはテーブルの列は存在しません。ここでのテーブル列は、複数の行を上下方向に並べた結果として形成された、いわば見かけ上の存在です。ただし、OpenOffice.org には、テーブルに簡単にアクセスできるよう、列を操作するメソッドがいくつか用意されています。これらのメソッドは、セルを結合していないテーブルを扱う場合に有用です。

ここではまず、テーブル自体のプロパティーについて説明します。これらは、

```
com.sun.star.text.TextTable
```

サービスで定義されています。以下に、重要度の高いテーブルオブジェクトの属性を示します。

BackColor (Long)

テーブルの背景色。

BottomMargin (Long)

100 分の 1 ミリ単位で指定した下部マージン。

LeftMargin (Long)

100 分の 1 ミリ単位で指定した左マージン。

RightMargin (Long)

100 分の 1 ミリ単位で指定した右マージン。

TopMargin (Long)

100 分の 1 ミリ単位で指定した上部マージン。

RepeatHeadline (Boolean)

テーブルヘッダをすべてのページに表示します。

Width (Long)

100 分の 1 ミリ単位で指定したテーブルの絶対幅。

テーブル行

テーブルは、行を含むリストで構成されます。次のサンプルコードでは、テーブル内の行を取得して書式を設定します。

```
Dim Doc As Object
Dim Table As Object
Dim Cursor As Object
Dim Rows As Object
Dim Row As Object
Dim I As Integer

Doc = StarDesktop.CurrentComponent
Cursor = Doc.Text.createTextCursor()

Table = Doc.createInstance("com.sun.star.text.TextTable")
Table.initialize(5, 4)

Doc.Text.insertTextContent(Cursor, Table, False)
Rows = Table.getRows
For I = 0 To Rows.getCount() - 1
   Row = Rows.getByIndex(I)
   Row.BackColor = &HFF00FF
Next
```

この例では、最初に Table.getRows を使用して、すべての行を含むリストを作成しています。リストをさらに処理するには、com.sun.star.table.XtableRows インターフェースに属する getCount および getByIndex メソッドを使用します。getByIndex メソッドは、

```
com.sun.star.text.TextTableRow
```

サービスをサポートする行オブジェクトを返します。

```
com.sun.star.table.XtableRows
```

インターフェースの主要なメソッドを次に示します。

getByIndex(Integer)

指定したインデックスの行オブジェクトを返します。

getCount()

行オブジェクトの数を返します。

insertByIndex(Index, Count)

テーブルの Index の位置に Count 行を挿入します。

removeByIndex(Index, Count)

テーブルの Index の位置から Count 行を削除します。

getByIndex および getCount メソッドはすべてのテーブルで使用できますが、insertByIndex および removeByIndex メソッドを使用できるのは、セルを結合していないテーブルだけです。

```
com.sun.star.text.TextTableRow
```

サービスは次のプロパティーを提供します。

BackColor (Long)

行の背景色。

Height (Long)

100 分の 1 ミリ単位で指定した行の高さ。

IsAutoHeight (Boolean)

テーブルの高さを内容に合わせて動的に調節する。

VertOrient (const)

テキスト枠の縦の方向 — テーブル内でのテキストの縦方向の詳細 (

```
com.sun.star.text.VertOrientation
```

で定義されている値)。

列

列には行と同じ方法でアクセスし、Column オブジェクトの getByIndex、getCount、insertByIndex、および removeByIndex メソッドを使用します。このオブジェクトは、getColumns を使用して取得します。ただしこれらのメソッドが利用できるのは、セルを結合していないテーブルだけです。また、OpenOffice.org Basic では、列単位でセルの書式を設定することはできません。このような処理を行うには、テーブルのセルを個別に書式設定する必要があります。

セル

OpenOffice.org ドキュメントの各セルには、固有の名前が付いています。OpenOffice.org のカーソルがセル内にある場合、そのセルの名前がステータスバーに表示されます。通常、左上隅のセルは A1 と表示され、右下隅のセルは Xn と表示されます。X は列を示す文字で、n は行を示す番号です。セルオブジェクトにアクセスするには、テーブルオブジェクトの getCellByName() メソッドを使用します。次のサンプルコードでは、ループを使用してテーブル内のすべてルにアクセスし、個々のセルごとに該当する行と列の番号を表示します。

```
Dim Doc As Object
Dim Table As Object
Dim Cursor As Object
Dim Rows As Object
Dim RowIndex As Integer
Dim Cols As Object
Dim ColIndex As Integer
Dim CellName As String
Dim Cell As Object

Doc = StarDesktop.CurrentComponent
Cursor = Doc.Text.createTextCursor()

Table = Doc.createInstance("com.sun.star.text.TextTable")
Table.initialize(5, 4)

Doc.Text.insertTextContent(Cursor, Table, False)

Rows = Table.getRows
Cols = Table.getColumns

For RowIndex = 1 To Rows.getCount()
   For ColIndex = 1 To Cols.getCount()
      CellName = Chr(64 + ColIndex) & RowIndex
      Cell = Table.getCellByName(CellName)
      Cell.String = "row: " & CStr(RowIndex) + ", column: " &
CStr(ColIndex)
   Next
Next
```

テーブルのセルは、通常のテキストに相当します。関連する TextCursor オブジェクトの作成用に、createTextCursor インターフェースがサポートされています。

```
CellCursor = Cell.createTextCursor()
```

したがって、個々の文字および段落に対して設定可能なすべての書式オプションが、自動的に使用できるようになります。

次のサンプルコードでは、文書ドキュメント上のすべてのテーブルを調べて、数値の入ったセルのみを右揃えにしますが、セルの書式設定の際に段落プロパティーを利用しています。

```
Dim Doc As Object
Dim TextTables As Object
Dim Table As Object
Dim CellNames
Dim Cell As Object
Dim CellCursor As Object
Dim I As Integer
Dim J As Integer

Doc = StarDesktop.CurrentComponent
TextTables = Doc.getTextTables()

For I = 0 to TextTables.count - 1
   Table = TextTables(I)
   CellNames = Table.getCellNames()

   For J = 0 to UBound(CellNames)
      Cell = Table.getCellByName(CellNames(J))
      If IsNumeric(Cell.String) Then
         CellCursor = Cell.createTextCursor()
         CellCursor.paraAdjust =
com.sun.star.style.ParagraphAdjust.RIGHT
      End If
   Next
Next
```

この例では、ループで処理したすべてのテキストのテーブルを含む TextTables リストを作成しています。続いて、OpenOffice.org はこれらの各テーブルに関連付けられているセル名のリストを作成します。次にこのリストを基にして、第 2 のループを開始します。このループでは、セル内のデータが数値であるかを判定し、その結果に応じて書式を設定します。そのために、まずテーブルセルの内容を参照する TextCursor オブジェクトを作成してから、テーブルセルの段落プロパティーを適用しています。

テキスト枠

テキスト枠も、テーブルやグラフと同様に TextContent オブジェクトとして扱われます。テキスト枠は本質的に通常のテキストと同質のものですが、ページ上の任意の位置に配置できることと、ドキュメント本文のテキストの流れから外れた存在である点が異なります。

これまでに見てきた TextContent オブジェクトと同様に、テキスト枠の場合も、オブジェクトの作成とドキュメント上への挿入は、それぞれ個別の操作として実行します。

```
Dim Doc As Object
Dim TextTables As Object
Dim Cursor As Object
Dim Frame As Object

Doc = StarDesktop.CurrentComponent
Cursor = Doc.Text.createTextCursor()
Frame = Doc.createInstance("com.sun.star.text.TextFrame")
Doc.Text.insertTextContent(Cursor, Frame, False)
```

テキスト枠の作成には、ドキュメントオブジェクトの createInstance メソッドを使用します。作成後のテキスト枠は、Text オブジェクトの insertTextContent メソッドを使用して、ドキュメントへ挿入します。その際には、

```
com.sun.star.text.TextFrame
```

というサービス名を指定する必要があります。

テキスト枠の挿入位置は Cursor オブジェクトにより指定されるため、挿入時はこのオブジェクトも準備しておく必要があります。

OpenOffice.org のテキスト枠は、MS Word の位置フレームに相当する機能です。ただし、VBA の処理では、Document.Frames.Add メソッドを使用しますが、StarSuite Basic では上述したように、ドキュメントオブジェクトの TextCursor および createInstance メソッドを使用して作成します。

テキスト枠オブジェクトには、テキスト枠の表示位置や動作を制御するために、各種の属性が用意されています。これらプロパティーの大部分は

```
com.sun.star.text.BaseFrameProperties
```

サービスで定義されているもので、これは各 TextFrame サービスでもサポートされています。以下に主要なプロパティーを示します。

BackColor (Long)

テキスト枠の背景色。

BottomMargin (Long)

100 分の 1 ミリ単位で指定した下部マージン。

LeftMargin (Long)

100 分の 1 ミリ単位で指定した左マージン。

RightMargin (Long)

100 分の 1 ミリ単位で指定した右マージン。

TopMargin (Long)

100 分の 1 ミリ単位で指定した上部マージン。

Height (Long)

100 分の 1 ミリ単位で指定したテキスト枠の高さ。

Width (Long)

100 分の 1 ミリ単位で指定したテキスト枠の幅。

HoriOrient (const)

テキスト枠の横長書式 (

```
com.sun.star.text.HoriOrientation
```

で定義されている値)。

VertOrient (const)

テキスト枠の縦方向 (

```
com.sun.star.text.VertOrientation
```

で定義されている値)。

次のサンプルコードでは、これらのプロパティーを使用してテキスト枠を作成します。

```
Dim Doc As Object
Dim TextTables As Object
Dim Cursor As Object
Dim Frame As Object

Doc = StarDesktop.CurrentComponent
Cursor = Doc.Text.createTextCursor()
Cursor.gotoNextWord(False)
Frame = Doc.createInstance("com.sun.star.text.TextFrame")

Frame.Width = 3000
Frame.Height = 1000
Frame.AnchorType = com.sun.star.text.TextContentAnchorType.AS_CHARACTER
Frame.TopMargin = 0
Frame.BottomMargin = 0
Frame.LeftMargin = 0
Frame.RightMargin = 0
Frame.BorderDistance = 0
Frame.HoriOrient = com.sun.star.text.HoriOrientation.NONE
Frame.VertOrient = com.sun.star.text.VertOrientation.LINE_TOP

Doc.Text.insertTextContent(Cursor, Frame, False)
```

このサンプルコードでは、テキスト枠の挿入位置を指定するために TextCursor を作成しています。実際の挿入位置は、1 つ目と 2 つ目の単語の間です。そして、Doc.createInstance を使用してテキスト枠を作成します。その後、テキスト枠の各属性に必要な値を指定しています。

AnchorType (TextContent サービス) プロパティーと VertOrient (BaseFrameProperties サービス) プロパティーとの関係には注意が必要です。AnchorType は AS_CHARACTER 値を受け取ります。これは、テキスト枠をテキストフロー中に直接挿入して、通常の文字として動作するよう指定するものです。このような指定により、たとえばテキストフローが途中で改行される場合、このテキスト枠も次の行に送られるようになります。一方の VertOrient プロパティーに指定した LINE_TOP という値は、テキスト枠とテキストの上端を同じ高さにするための設定です。

初期値の設定終了後、insertTextContent を使用してテキスト枠を文章ドキュメントに挿入します。

テキスト枠の内容を編集するには、TextCursor を使用する必要がありますが、その使用法はすでに説明したように、テキスト枠の場合も特に違いはありません。

```
Dim Doc As Object
Dim TextTables As Object
Dim Cursor As Object
Dim Frame As Object
Dim FrameCursor As Object

Doc = StarDesktop.CurrentComponent
Cursor = Doc.Text.createTextCursor()
Frame = Doc.createInstance("com.sun.star.text.TextFrame")

Frame.Width = 3000
Frame.Height = 1000

Doc.Text.insertTextContent(Cursor, Frame, False)

FrameCursor = Frame.createTextCursor()
FrameCursor.charWeight = com.sun.star.awt.FontWeight.BOLD
FrameCursor.paraAdjust = com.sun.star.style.ParagraphAdjust.CENTER
FrameCursor.String = "This is a small Test!"
```

上記のサンプルコードでは、テキスト枠を作成して現在のドキュメントに挿入し、このテキスト枠内部に TextCursor を移動しています。次にこのカーソルを使用して、テキスト枠内の表示フォントの太字への設定および、段落配置の中央揃への設定を行なっています。そして最後に「This is a small test!」という文字列を、テキスト枠内に表示させています。文字列。

テキストフィールド

テキストフィールドは、通常のテキストを拡張する追加ロジックを提供するので、TextContent オブジェクトです。テキストフィールドの文書ドキュメントへの挿入では、他の TextContent オブジェクトの場合と同様のメソッドを使用します。

```
Dim Doc As Object
Dim DateTimeField As Object
Dim Cursor As Object
Doc = StarDesktop.CurrentComponent
Cursor = Doc.Text.createTextCursor()

DateTimeField =
Doc.createInstance("com.sun.star.text.TextField.DateTime")
DateTimeField.IsFixed = False
DateTimeField.IsDate = True
Doc.Text.insertTextContent(Cursor, DateTimeField, False)
```

このサンプルコードでは、現在の日付を表示するテキストフィールドを、文書ドキュメントの先頭に挿入しています。`IsDate` プロパティーへの `True` の指定は、日付のみを表示し、時刻は表示しないことを意味します。また、`IsFixed` への `False` の指定は、ドキュメントを開く際に日付を自動更新することを意味します。

VBA ではフィールドの種類を指定するのに `Document.Fields.Add` メソッドのパラメータを使用しますが、OpenOffice.org Basic ではフィールドの種類を指定するサービス名がその役割を担っています。

OpenOffice.org の従来バージョンでは、テキストフィールドを操作するのに、`Selection` オブジェクトで使用できた各種のメソッドを利用する必要がありました (たとえば `InsertField`、`DeleteUserField`、`SetCurField`)。

OpenOffice.org 2.x では、テキストフィールドの処理もオブジェクト指向の概念に基づくよう改められました。つまりテキストフィールドを構築するに当たっては、最初に該当するタイプのテキストフィールドを作成してから、必要な属性値を指定して初期化する必要があります。このような処理を経た後、`insertTextContent` メソッドを使用してテキストフィールドをドキュメントに挿入します。このような処理の流れは、先に紹介したサンプルコードに示した通りです。主要なフィールドの種類およびそれらの属性については、次の節で説明しています。

テキストフィールド関係の操作では、ドキュメントへの挿入以外にも、ドキュメント上のフィールドを検索するという処理も行えます。次のサンプルコードでは、ループを使って文書ドキュメント上のすべてのテキストフィールドにアクセスし、フィールドの種類を確認します。

```
Dim Doc As Object
Dim TextFieldEnum As Object
Dim TextField As Object
Dim I As Integer

Doc = StarDesktop.CurrentComponent

TextFieldEnum = Doc.getTextFields.createEnumeration

While TextFieldEnum.hasMoreElements()

   TextField = TextFieldEnum.nextElement()

   If TextField.supportsService("com.sun.star.text.TextField.DateTime")
Then
      MsgBox "Date/time"
   ElseIf
TextField.supportsService("com.sun.star.text.TextField.Annotation") Then
      MsgBox "Annotation"
   Else
      MsgBox "unknown"
   End If

Wend
```

テキストフィールドの存在を設定する起点となるのは、ドキュメントオブジェクトの TextFields リストです。このサンプルコードでは、Enumeration オブジェクトを作成することでこのようなリストを取得し、すべてのテキストフィールドにアクセスするループに入ります。検出されたテキストフィールドに対しては、supportsService メソッドを用いて、そのサポートするサービスを確認します。フィールドの種類が日付/時刻 (date/time) ないしコメント (annotation) の場合は、該当するフィールドの種類をメッセージボックスに表示します。その他の種類のフィールドに対しては、「unknown」と表示します。

以下に、重要なテキストフィールドとそのプロパティーを示します。すべてのテキストフィールドに関するリストについては、『API reference』の com.sun.star.text.TextField モジュールを参照してください (OpenOffice.org Basic では、先に見たサンプルコードの表記と同様に、テキストフィールドのサービス名の大文字と小文字を区別する必要があります)。

ページ数、語数、文字数

テキストフィールドについては、次のサービスが利用できます。

.

```
com.sun.star.text.TextField.PageCount
```

.

```
com.sun.star.text.TextField.WordCount
```

.

```
com.sun.star.text.TextField.CharacterCount
```

それぞれページ数、語数、文字数を返します。これらは次のプロパティーをサポートしています。

NumberingType (const)

番号付け形式 (

```
com.sun.star.style.NumberingType
```

の定数に従ったガイドライン)。

現在のページ

現在のページ番号を表示するフィールドをドキュメントに挿入するには、テキストフィールドに

```
com.sun.star.text.TextField.PageNumber
```

を指定します。この場合は、次のプロパティーが利用できます。

NumberingType (const)

番号形式 (

```
com.sun.star.style.NumberingType
```

の定数に従ったガイドライン)。

Offset (short)

ページ番号に追加するオフセット値 (負の値も指定可能)。

次のサンプルコードでは、ページ番号表示用のフッタをドキュメントに挿入します。

```
Dim Doc As Object
Dim DateTimeField As Object
Dim PageStyles As Object
Dim StdPage As Object
Dim FooterCursor As Object
Dim PageNumber As Object

Doc = StarDesktop.CurrentComponent

PageNumber =
Doc.createInstance("com.sun.star.text.TextField.PageNumber")
PageNumber.NumberingType = com.sun.star.style.NumberingType.ARABIC

PageStyles = Doc.StyleFamilies.getByName("PageStyles")

StdPage = PageStyles("Default")
StdPage.FooterIsOn = True

FooterCursor = StdPage.FooterTextLeft.Text.createTextCursor()
StdPage.FooterTextLeft.Text.insertTextContent(FooterCursor, PageNumber,
False)
```

このサンプルコードでは、まず

```
com.sun.star.text.TextField.PageNumber
```

サービスをサポートするテキストフィールドを作成しています。OpenOffice.org ではヘッダおよびフッタ行をページテンプレートに定義しているため、PageStyles のリストから選択することによりオブジェクトを作成しています。

フッタ行を表示するには、FooterIsOn プロパティーに True を設定します。テキストフィールドをドキュメントに挿入する際には、左揃え表示を指定したテキストオブジェクトを使用しています。

コメント

コメント (注釈) フィールド (

```
com.sun.star.text.TextField.Annotation
```

) は、文章ドキュメント上で黄色のシンボルとして表示されます。このシンボルをクリックすると、テキストフィールドが開き、挿入箇所の文章に関するコメントを入力できるようになります。コメントフィールドでは、次の属性が利用できます。

Author (String)

作成者の名前。

Content (String)

コメントテキスト。

Date (Date)

コメントが入力された日付。

日付と時刻

日付/時刻フィールド (

```
com.sun.star.text.TextField.DateTime
```

) は、現在の日付および時刻の表示に使用します。この場合は、次のプロパティーが利用できます。

IsFixed (Boolean)

True を指定すると、表示時間を更新させず、False を指定するとドキュメントを開く際に自動更新する。

IsDate (Boolean)

True を指定するとフィールドに現在の日付を表示させ、それ以外の場合は現在の時刻を表示する。

DateTimeValue (struct)

フィールドの現在の内容 (

```
com.sun.star.util.DateTime
```

構造体)。

NumberFormat (const)

日付および時刻の書式。

章名および章番号

テキストフィールドに章名を表示するには、

```
com.sun.star.text.TextField.Chapter
```

型を使用します。表示形式は、次の 2 つの属性で指定します。

ChapterFormat (const)

章名と章番号のどちらを表示するかを指定する (

```
com.sun.star.text.ChapterFormat
```

に定められた値)。

Level (Integer)

章名ないし章番号として表示させる章レベルを指定する。指定値は 0 が最高レベルに該当します。

ブックマーク

ブックマーク (

```
com.sun.star.text.Bookmark
```

サービス) は TextContent オブジェクトです。このためテキストマークの作成と挿入も、すでに説明したものと同様の手順で行えます。

```
Dim Doc As Object
Dim Bookmark As Object
Dim Cursor As Object

Doc = StarDesktop.CurrentComponent

Cursor = Doc.Text.createTextCursor()

Bookmark = Doc.createInstance("com.sun.star.text.Bookmark")
Bookmark.Name = "My bookmarks"
Doc.Text.insertTextContent(Cursor, Bookmark, True)
```

このサンプルコードでは、ブックマークの挿入位置の指定用に Cursor というオブジェクトを作成してから、実際のブックマークオブジェクト (Bookmark) を作成しています。このブックマークは、名前を付けてから、insertTextContent を用いてカーソル位置に挿入します。

ブックマークへのアクセスには、Bookmarks と呼ばれるリストを使用します。また個々のテキストマークは、番号または名前により特定できます。

次のサンプルコードでは、文書ドキュメント上で特定のブックマークを選択して、該当位置へテキストを挿入します。

```
Dim Doc As Object
Dim Bookmark As Object
Dim Cursor As Object

Doc = StarDesktop.CurrentComponent

Bookmark = Doc.Bookmarks.getByName("My bookmarks")

Cursor = Doc.Text.createTextCursorByRange(Bookmark.Anchor)
Cursor.String = "Here is the bookmark"
```

ここでは getByName メソッドを用いて、ブックマークをその名前で特定しています。そして createTextCursorByRange を用いて Cursor というオブジェクトを作成して、このブックマークのアンカー位置を取得します。最後にこのカーソルの示す位置に、文字列を挿入しています。

表計算ドキュメント

OpenOffice.org Basic には、プログラム制御により表計算ドキュメントの作成および編集を行えるよう、各種のインターフェースが用意されています。この章では、表計算ドキュメントの操作に必要なサービス、メソッド、プロパティーについて説明します。

・表計算ドキュメントの構造

・表計算ドキュメントの効率的な編集方法

最初の節では、表計算ドキュメントの基本的な構造について触れ、個々のセルへのアクセスおよび編集方法を説明します。

次の節では、表計算ドキュメントのより効率的な編集方法を検討し、セル範囲という概念や、セルの内容の検索と置換を行うためのオプションを説明します。

Range オブジェクトを利用すると、様々なセル範囲にアクセスすることができますが、これらは新規 API の導入に伴って拡張された機能です。

表計算ドキュメントの構造

表計算ドキュメントのドキュメントオブジェクトは、

```
com.sun.star.sheet.SpreadsheetDocument
```

サービスをベースにしています。通常これらのドキュメントには、複数の表 (スプレッドシート) があります。マニュアルで使う用語として、表計算ドキュメント は 1 つのドキュメント全体を意味するものとし、スプレッドシート (略称: シート) は各ドキュメントを構成する個々の表 (テーブル) を意味するものとします。

VBA と OpenOffice.org Basic では、スプレッドシートとスプレッドシートの内容に対して、異なった用語が使用されています。VBA のドキュメントオブジェクトは Workbook、個々のページは Worksheets と呼ばれますが、OpenOffice.org Basic の場合は SpreadsheetDocument および Sheet と呼ばれています。

スプレッドシート

表計算ドキュメントの各シートにアクセスするには、Sheets リストを使用します。

次の 2 つのサンプルコードでは、それぞれ番号および名前を使って各シートへアクセスする方法を示します。

例 **1:** 番号によるアクセス **(**開始値は **0)**

```
Dim Doc As Object
Dim Sheet As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets (0)
```

例 **2:** 名前によるアクセス

```
Dim Doc As Object
Dim Sheet As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets.getByName(”Sheet 1”)
```

最初のサンプルコードでは、シートへのアクセスを番号指定で行なっています (開始値は 0)。次のサンプルコードでは、getByName メソッドにシート名を指定してアクセスしています。

getByName メソッドで取得される Sheet というオブジェクトは、

```
com.sun.star.sheet.Spreadsheet
```

サービスをサポートしています。このサービスは、シート編集用の各種インターフェースを提供するもので、次の属性を利用できます。

IsVisible (Boolean)

スプレッドシートを表示する指定。

PageStyle (String)

スプレッドシート用のページテンプレートの名前。

シートの作成、削除、名前の変更

spreadsheet ドキュメントの Sheets リストは、個々のシートの作成、削除、名前の変更にも使用します。次のサンプルコードでは、hasByName メソッドを用いて **MySheet** という名前のシートが存在するかをチェックします。そうした名前のシートが存在する場合、getByName メソッドを用いて該当オブジェクトへの参照を取得して、Sheet という変数に収めます。該当するシートが存在しない場合は、createInstance メソッドを用いてこれを新規作成して、insertByName メソッドにより表計算ドキュメントに挿入します。

```
Dim Doc As Object
Dim Sheet As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)

If Doc.Sheets.hasByName("MySheet") Then
   Sheet = Doc.Sheets.getByName("MySheet")
Else
   Sheet = Doc.createInstance("com.sun.star.sheet.Spreadsheet")
   Doc.Sheets.insertByName("MySheet", Sheet)
End If
```

「API について」で説明したように、getByName および insertByName メソッドは

```
com.sun.star.container.XnameContainer
```

インターフェースから提供されています。

行と列

個々のシートは、複数の行と列から構成されています。これらは、スプレッドシートオブジェクトの Rows および Columns プロパティーを通して使用でき、

```
com.sun.star.table.TableColumns
```

および

```
com.sun.star.table.TableRows
```

サービスをサポートします。

次のサンプルコードでは、FirstCol および FirstRow という 2 つのオブジェクト変数を作成し、それぞれに第 1 列および第 1 行の参照情報を格納させています。

```
Dim Doc As Object
Dim Sheet As Object
Dim FirstRow As Object
Dim FirstCol As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)

FirstCol = Sheet.Columns(0)
FirstRow = Sheet.Rows(0)
```

列オブジェクトは

```
com.sun.star.table.TableColumn
```

サービスをサポートしており、次のプロパティーを利用できます。

Width (long)

100 分の 1 ミリ単位で指定した列幅。

OptimalWidth (Boolean)

列を最適な幅に設定する指定。

IsVisible (Boolean)

列を表示する指定。

IsStartOfNewPage (Boolean)

印刷時に該当列の前で改ページをする指定。

列の幅は、OptimalWidth プロパティーに True を指定した場合のみ、自動的に最適化されます。個々のセル幅が変更されても、そのセルを含む列の幅は変更されません。実際の機能面から見た場合、OptimalWidth はプロパティーではなくメソッドとして分類されるべきものです。

行オブジェクトは

```
com.sun.star.table.RowColumn
```

サービスをベースとしており、次のプロパティーを利用できます。

Height (long)

100 分の 1 ミリ単位で指定した行の高さ。

OptimalHeight (Boolean)

行を最適な高さに設定する指定。

IsVisible (Boolean)

行を表示する指定。

IsStartOfNewPage (Boolean)

印刷時に該当行の前で改ページをする指定。

行の OptimalHeight プロパティーに True を指定した場合、その行に属するセルの高さが変更された際に、行全体の高さが自動的に最適化されます。こうした自動最適化は、Height プロパティーで行の高さを指定すると解除されます。

次のサンプルコードでは、シート内の最初の 5 行に対して高さの自動最適化を設定し、第 2 列を非表示にします。

```
Dim Doc As Object
Dim Sheet As Object
Dim Row As Object
Dim Col As Object
Dim I As Integer

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)

For I = 0 To 4
   Row = Sheet.Rows(I)
   Row.OptimalHeight = True
Next I

Col = Sheet.Columns(1)
Col.IsVisible = False
```

OpenOffice.org Basic では、Rows および Columns のリストには、インデックスを使用してアクセスできます。ただし VBA の場合とは異なり、列のインデックスの開始値は 1 ではなく 0 となります。

列および行の挿入と削除

各シート中の列や行へのアクセスおよび、これらの挿入と削除には、Rows および Columns オブジェクトを使用します。

```
Dim Doc As Object
Dim Sheet As Object
Dim NewColumn As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)

Sheet.Columns.insertByIndex(3, 1)
Sheet.Columns.removeByIndex(5, 1)
```

このサンプルコードでは、insertByIndex メソッドを用いて、シート上の第 4 列に新規列を挿入しています (開始値が 0 なのでインデックス値は 3)。挿入時の第 2 パラメータには、挿入する列数を指定します (この場合は 1)。

次に、removeByIndex メソッドを用いて、第 6 列を削除しています (インデックス値は 5)。この場合の第 2 パラメータには、削除する列数を指定します。

こうした列の処理法は、行に対する挿入や削除の場合も同様で、Columns オブジェクトの代わりに Rows オブジェクトを使用するだけです。

セルと範囲

セル個々の表計算ドキュメントを構成するのがセルという単位で、これらは 2 次元のリストで管理されます。つまり各セルは、左上隅を原点 (0,0) とする X と Y 座標で指定できます。

次のサンプルコードは、左上隅のセルにアクセスして、文字列を挿入します。

```
Dim Doc As Object
Dim Sheet As Object
Dim Cell As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)

Cell = Sheet.getCellByPosition(0, 0)
Cell.String = "Test"
```

`StarDesktop.CurrentComponent` は現在のコンポーネントを返します。BASIC IDE で作業している場合、マクロを実行すると、BASIC IDE が返されます。BASIC IDE には表計算ドキュメントコンポーネントがないため、RunTime エラーが生成されます。

サンプルコードを保存し、表計算ドキュメントからマクロを実行します。詳細情報については、「StarDesktop」を参照してください。

セルの参照は、こうした数字による座標指定だけではなく、名前による指定も可能で、たとえば表計算ドキュメントの左上隅のセル (0,0) は、`A1` セルとも呼ばれます。この場合のアルファベット `A` は列の位置を示し、数値 1 は行の位置を示します。このようなセル位置の参照方式を使い分ける際には、名前 (**name**) 方式の行指定は 1 から始まり、位置 (**position**) 方式の値は 0 から始まるので、両者を混同しないよう注意が必要です。

OpenOffice.org のセルには、テキスト、数値、数式のいずれかを入力でき、何も入力されていないものを空白セルと呼びます。セルの種類は、セルに入力する内容で規定されるのではなく、これらを入力する際に使用するオブジェクト属性により決まります。つまり数値を入力するには `Value` 属性、テキストを入力するには `String` 属性、数式を入力するには `Formula` 属性をそれぞれ使用します。

```
Dim Doc As Object
Dim Sheet As Object
Dim Cell As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)

Cell = Sheet.getCellByPosition(0, 0)
Cell.Value = 100

Cell = Sheet.getCellByPosition(0, 1)
Cell.String = "Test"

Cell = Sheet.getCellByPosition(0, 2)
Cell.Formula = "=A1"
```

このサンプルコードでは、A1 から A3 のセルに、それぞれ数値、テキスト、数式を入力しています。

Value、String、Formula 属性は、セルの値を指定する PutCell メソッドに取って代わる存在となっています。

仮に String 属性を用いて数値を入力した場合、OpenOffice.org はこれをテキストとして扱います。たとえば、このような方法でセルに入力された数値は、右揃えではなく左揃えに表示されます。また数式を用いてテキストや数値を表示させた場合にも、同様の注意が必要です。

```
Dim Doc As Object
Dim Sheet As Object
Dim Cell As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)

Cell = Sheet.getCellByPosition(0, 0)
Cell.Value = 100

Cell = Sheet.getCellByPosition(0, 1)
Cell.String = 1000

Cell = Sheet.getCellByPosition(0, 2)
Cell.Formula = "=A1+A2"

MsgBox Cell.Value
```

このサンプルコードを実行すると、A1 セルには 100、A2 セルには 1000 と表示されますが、A1+A2 という計算式の結果は 100 となります。こうなる原因は、A2 セルの値を数値ではなくテキストとして入力したためです。

セルの内容が数値であるか文字列であるかを確認するには、Type 属性を使用します。

```
Dim Doc As Object
Dim Sheet As Object
Dim Cell As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)
Cell = Sheet.getCellByPosition(1,1)

Cell.Value = 1000

Select Case Cell.Type
Case com.sun.star.table.CellContentType.EMPTY
   MsgBox "Content: Empty"
Case com.sun.star.table.CellContentType.VALUE
   MsgBox "Content: Value"
Case com.sun.star.table.CellContentType.TEXT
   MsgBox "Content: Text"
Case com.sun.star.table.CellContentType.FORMULA
   MsgBox "Content: Formula"
End Select
```

Cell.Type 属性を使うと、セルの内容の種類を示す

```
com.sun.star.table.CellContentType
```

の値を取得できます。返される値は、次のいずれかです。

EMPTY

値なし

VALUE

数値

TEXT

文字列

FORMULA

数式

セルの挿入、削除、コピー、移動

OpenOffice.org Calc には、セルの内容を直接編集する以外にも、セルの挿入、削除、コピー、移動を行うためのインターフェースが用意されています。このインターフェース (

```
com.sun.star.sheet.XRangeMovement
```

) は、表計算ドキュメントオブジェクトを通じて利用するもので、セルの内容を操作するために 4 種類のメソッドを提供しています。

insertCell メソッドは、セルを表計算ドキュメントに挿入する際に使用します。

```
Dim Doc As Object
Dim Sheet As Object
Dim CellRangeAddress As New com.sun.star.table.CellRangeAddress

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)

CellRangeAddress.Sheet = 0
CellRangeAddress.StartColumn = 1
CellRangeAddress.StartRow = 1
CellRangeAddress.EndColumn = 2
CellRangeAddress.EndRow = 2

Sheet.insertCells(CellRangeAddress,
com.sun.star.sheet.CellInsertMode.DOWN)
```

このサンプルコードは、表計算ドキュメントの最初のシート (インデックス値 0) の第 2 列と第 2 行の位置 (インデックス値はともに 1) に、2 行 2 列分のセル範囲を挿入します。また挿入位置にある既存のセルは、その内容ごと下方に移動しています。

挿入するセル範囲を指定するには、

```
com.sun.star.table.CellRangeAddress
```

構造体を使用します。この構造体には、次の値を設定できます。

Sheet (short)

シート番号 (開始値は 0)。

StartColumn (long)

セル範囲の先頭列 (開始値は 0)。

StartRow (long)

セル範囲の先頭行 (開始値は 0)。

EndColumn (long)

セル範囲の最終列 (開始値は 0)。

EndRow (long)

セル範囲の最終行 (開始値は 0)。

これらの設定の終了した CellRangeAddress 構造体は、insertCells メソッドの第 1 パラメータとして渡します。insertCells メソッドの第 2 パラメータには、セル範囲の挿入位置にある既存セルの処置を指定するため、

```
com.sun.star.sheet.CellInsertMode
```

の値を渡します。この CellInsertMode には、次の値が用意されています。

NONE

挿入前の値を、その位置にとどめます。

DOWN

挿入位置にあるセルを、下に移動します。

RIGHT

挿入位置にあるセルを、右に移動します。

ROWS

挿入位置にある行全体を、下に移動します。

COLUMNS

挿入位置にある列全体を、右に移動します。

removeRange メソッドは、insertCells メソッドと逆の機能を担っています。このメソッドは、CellRangeAddress 構造体で指定したセル範囲を、スプレッドシートから削除します。

```
Dim Doc As Object
Dim Sheet As Object
Dim CellRangeAddress As New com.sun.star.table.CellRangeAddress

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)

CellRangeAddress.Sheet = 0
CellRangeAddress.StartColumn = 1
CellRangeAddress.StartRow = 1
CellRangeAddress.EndColumn = 2
CellRangeAddress.EndRow = 2

Sheet.removeRange(CellRangeAddress,
com.sun.star.sheet.CellDeleteMode.UP)
```

このサンプルコードでは、B2:C3 のセル範囲をシートから削除して、該当範囲の下方にあるセルを上に 2 行分移動させています。削除に伴い周囲のセルをどう処理するかは、次の

```
com.sun.star.sheet.CellDeleteMode
```

の値により指定します。

NONE

削除前の値を、その位置にとどめます。

UP

該当範囲の下側にあるセルを上に移動します。

LEFT

該当範囲の右側にあるセルを左に移動します。

ROWS

該当範囲の下側にある行全体を、上に移動します。

COLUMNS

該当範囲の右側にある列全体を、左に移動します。

XRangeMovement インターフェースには、セル範囲の移動 (moveRange) およびコピー (copyRange) 処理用に、2 つの追加メソッドが用意されています。次のサンプルコードでは、B2:C3 のセル範囲を、A6 の位置へ移動します。

```
Dim Doc As Object
Dim Sheet As Object
Dim CellRangeAddress As New com.sun.star.table.CellRangeAddress
Dim CellAddress As New com.sun.star.table.CellAddress

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)

CellRangeAddress.Sheet = 0
CellRangeAddress.StartColumn = 1
CellRangeAddress.StartRow = 1
CellRangeAddress.EndColumn = 2
CellRangeAddress.EndRow = 2

CellAddress.Sheet = 0
CellAddress.Column = 0
CellAddress.Row = 5

Sheet.moveRange(CellAddress, CellRangeAddress)
```

moveRange メソッドを使用する場合は、CellRangeAdress 構造体の指定以外にも、

```
com.sun.star.table.CellAddress
```

構造体によるセル範囲の移動先も指定する必要があります。CellAddress には、次の値を設定できます。

Sheet (short)

表計算ドキュメント番号 (開始値は 0)。

Column (long)

移動先の列位置 (開始値は 0)。

Row (long)

移動先の行位置 (開始値は 0)。

moveRange メソッドを使用した場合、移動先のセルの内容は常に上書きされます。InsertCells メソッドの場合とは異なり、removeRange メソッドには、移動先のセルを自動的に移動させるパラメータは用意されていません。

`copyRange` メソッドの使用法は `moveRange` メソッドと同様ですが、`copyRange` の場合はセル範囲の移動ではなくコピーを行うという相違点があります。

機能面から見た場合、OpenOffice.org Basic の `insertCell`、`removeRange`、`copyRange` の各メソッドは、VBA の `Range.Insert`、`Range.Delete`、`Range.Copy` メソッドにそれぞれ相当します。ただし、これらの VBA のメソッドは該当する `Range` オブジェクトを対象とするのに対して、OpenOffice.org Basic のメソッドは `Sheet` オブジェクトを対象とします。

表計算ドキュメントの書式設定

表計算ドキュメントには、セルおよびページ単位での書式設定を行うための属性とメソッドが用意されています。

セル属性

セルの書式設定としては、表示フォントの種類やサイズなど、各種の項目があります。個々のセルは

```
com.sun.star.style.CharacterProperties
```

および

```
com.sun.star.style.ParagraphProperties
```

サービスをサポートしていますが、これらが使用する主要な属性については「文書ドキュメント」で説明しています。また特殊なセル書式については、

```
com.sun.star.table.CellProperties
```

サービスで処理します。このサービスで使用する主要な属性については、次の節で説明します。

これらの属性は、個々のセルおよびセル範囲に対して指定できます。

OpenOffice.org API の `CellProperties` オブジェクトは、VBA の `Interior` オブジェクトに相当しますが、これはセル固有の属性の指定にも使用します。

背景色および影

```
com.sun.star.table.CellProperties
```

サービスには、背景色および影の表示指定用に、次の属性が用意されています。

`CellBackColor (Long)`

テーブルのセルの背景色。

`IsCellBackgroundTransparent (Boolean)`

背景色を透明にする指定。

ShadowFormat (struct)

セルの影の指定 (

```
com.sun.star.table.ShadowFormat
```

に定められた構造体)。

```
com.sun.star.table.ShadowFormat
```

構造体とセルの影指定は、次のような構成になっています。

Location (enum)

影の位置 (

```
com.sun.star.table.ShadowLocation
```

構造体の値)。

ShadowWidth (Short)

100 分の 1 ミリ単位で指定した影の幅。

IsTransparent (Boolean)

影を透明にする指定。

Color (Long)

影の色。

次のサンプルコードでは、B2 セルへの数値 1000 の入力、`CellBackColor` 属性による背景色の赤への変更、左下側 1 mm の位置への明るい灰色の影の表示を行います。

```
Dim Doc As Object
Dim Sheet As Object
Dim Cell As Object
Dim ShadowFormat As New com.sun.star.table.ShadowFormat

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)
Cell = Sheet.getCellByPosition(1,1)

Cell.Value = 1000

Cell.CellBackColor = RGB(255, 0, 0)

ShadowFormat.Location = com.sun.star.table.ShadowLocation.BOTTOM_RIGHT
ShadowFormat.ShadowWidth = 100
ShadowFormat.Color = RGB(160, 160, 160)

Cell.ShadowFormat = ShadowFormat
```

テキストの配置

OpenOffice.org には、テーブルのセルのテキスト配置を指定する各種の機能が用意されています。

次の属性は、テキストの水平および垂直方向の配置を指定するものです。

HoriJustify (enum)

テキストの水平方向の配置 (

```
com.sun.star.table.CellHoriJustify
```

の値)。

VertJustify (enum)

テキストの垂直方向の配置 (

```
com.sun.star.table.CellVertJustify
```

の値)。

Orientation (enum)

テキストの向き (

```
com.sun.star.table.CellOrientation
```

に定められた値)。

IsTextWrapped (Boolean)

セル内の自動行ブレークを許可する指定。

RotateAngle (Long)

100 分の 1 度単位で指定したテキストの回転角。

次のサンプルコードでは、左上隅にあるセルの表示を「縦書き」(stack) にして、文字を 1 文字ずつ下方向に並べて表示します。なお、文字の回転は指定していません。

```
Dim Doc As Object
Dim Sheet As Object
Dim Cell As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)
Cell = Sheet.getCellByPosition(1,1)

Cell.Value = 1000

Cell.HoriJustify = com.sun.star.table.CellHoriJustify.LEFT
Cell.VertJustify = com.sun.star.table.CellVertJustify.TOP
Cell.Orientation = com.sun.star.table.CellOrientation.STACKED
```

数値、日付、テキストの書式

OpenOffice.org には、日付および時刻に対する各種の書式が用意されています。これらの書式にはそれぞれ固有の内部番号が割り当てられており、NumberFormat 属性による書式指定もこの番号を使用します。OpenOffice.org では、queryKey および addNew メソッドを利用して、既存の数値の書式にアクセスできるだけでなく、数値の書式をユーザー定義することもできます。これらのメソッドには、次のようなオブジェクト呼び出しでアクセスします。

```
NumberFormats = Doc.NumberFormats
```

書式の指定は、OpenOffice.org Basic の Format 関数とよく似た形式のフォーマット用文字列を使用します。ただし両者の間には大きな違いがあり、Format 関数のフォーマット用文字列は英語式の小数点と千単位の桁区切り記号を使用しますが、NumberFormats オブジェクトのフォーマット用文字列には、各ロケール別の記号を使う必要があります。

次のサンプルコードでは、B2 セルに対して、小数部を 3 桁表示とし、コンマを千単位の桁区切りの記号とする数の書式を指定しています。

```
Dim Doc As Object
Dim Sheet As Object
Dim Cell As Object
Dim NumberFormats As Object
Dim NumberFormatString As String
Dim NumberFormatId As Long
Dim LocalSettings As New com.sun.star.lang.Locale

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)
Cell = Sheet.getCellByPosition(1,1)

Cell.Value = 23400.3523565

LocalSettings.Language = "en"
LocalSettings.Country = "us"

NumberFormats = Doc.NumberFormats
NumberFormatString = "#,##0.000"

NumberFormatId = NumberFormats.queryKey(NumberFormatString,
LocalSettings, True)
If NumberFormatId = -1 Then
   NumberFormatId = NumberFormats.addNew(NumberFormatString,
LocalSettings)
End If

MsgBox NumberFormatId
Cell.NumberFormat = NumberFormatId
```

セルの別の書式設定用オプションについては、OpenOffice.org Calc の「セルの書式設定」ダイアログで概要を参照してください。

ページ属性

ページ属性では、ドキュメントの内容を書式設定オプションによって各ページに配置したり、全ページに共通して配置する項目を設定したりします。具体的には、次のようなオプションを指定します。

・用紙サイズ

・ページ余白

・ヘッダとフッタ

ページ書式は、他の書式指定と設定法が異なります。セル、段落、テキストの書式は直接指定できますが、ページ書式はページスタイルを用いた間接的な設定も行えます。たとえば、ヘッダやフッタは、ページスタイルとして登録できます。

以降の節では、表計算ドキュメントの主要なページ書式設定オプションについて説明します。ここで説明するページスタイルの多くは、文書ドキュメントと共通するものです。こうした共通で使用されるページ属性は、

```
com.sun.star.style.PageProperties
```

サービスで定義されています。これに対して、表計算ドキュメント固有のページ属性は、

```
com.sun.star.sheet.TablePageStyle
```

サービスで定義されています。

Microsoft Office ドキュメントのページ属性 (ページ余白、枠線など) は、`Worksheet` オブジェクト (Excel) または `Document` オブジェクト (Word) レベルの `PageSetup` オブジェクトで指定します。OpenOffice.org の場合、このような属性の指定はページスタイルを用いて行い、その後これらのページスタイルを該当するドキュメントにリンクさせる形になります。

ページ背景

```
com.sun.star.style.PageProperties
```

サービスには、ページ背景に関する次の属性が用意されています。

BackColor (long)

背景の色。

BackGraphicURL (String)

背景に表示する画像の URL。

BackGraphicFilter (String)

背景用の画像を解釈するフィルタ名。

BackGraphicLocation (Enum)

背景用の画像の位置 (列挙型で定められた値)。

BackTransparent (Boolean)

背景を透明にする指定。

ページ書式

ページ書式の指定には、次の

```
com.sun.star.style.PageProperties
```

サービスの属性を使用します。

IsLandscape (Boolean)

横長書式。

Width (long)

100 分の 1 ミリ単位で指定したページの幅。

Height (long)

100 分の 1 ミリ単位で指定したページの高さ。

PrinterPaperTray (String)

使用するプリンタの用紙トレイの名前。

次のサンプルコードでは、「標準」ページスタイルのページサイズを、A5 サイズ (高さ 14.8 cm、幅 21 cm) の横長書式に設定します。

```
Dim Doc As Object
Dim Sheet As Object
Dim StyleFamilies As Object
Dim PageStyles As Object
Dim DefPage As Object

Doc = StarDesktop.CurrentComponent
StyleFamilies = Doc.StyleFamilies
PageStyles = StyleFamilies.getByName("PageStyles")
DefPage = PageStyles.getByName("Default")

DefPage.IsLandscape = True
DefPage.Width = 21000
DefPage.Height = 14800
```

ページ余白、外枠、影

```
com.sun.star.style.PageProperties
```

サービスには、ページの余白、外枠、影の設定用に、次の属性が用意されています。

LeftMargin (long)

100 分の 1 ミリ単位で指定したページの左余白。

RightMargin (long)

100 分の 1 ミリ単位で指定したページの右余白。

TopMargin (long)

100 分の 1 ミリ単位で指定したページの上余白。

BottomMargin (long)

100 分の 1 ミリ単位で指定したページの下余白。

LeftBorder (struct)

左側のページの外枠線の指定 (

```
com.sun.star.table.BorderLine
```

構造体)。

RightBorder (struct)

右側のページの外枠線の指定 (

```
com.sun.star.table.BorderLine
```

構造体)。

TopBorder (struct)

上側のページの外枠線の指定 (

```
com.sun.star.table.BorderLine
```

構造体)。

BottomBorder (struct)

下側のページの外枠線の指定 (

```
com.sun.star.table.BorderLine
```

構造体)。

LeftBorderDistance (long)

100 分の 1 ミリ単位で指定した、左側のページの外枠線とページ本文との距離。

RightBorderDistance (long)

100 分の 1 ミリ単位で指定した、右側のページの外枠線とページ本文との距離。

TopBorderDistance (long)

100 分の 1 ミリ単位で指定した、上側のページの外枠線とページ本文との距離。

BottomBorderDistance (long)

100 分の 1 ミリ単位で指定した、下側のページの外枠線とページ本文との距離。

ShadowFormat (struct)

ページの内容領域の影の指定 (

```
com.sun.star.table.ShadowFormat
```

構造体)。

次のサンプルコードでは、「標準」ページスタイルの左および右側の余白を、1 センチメートルに指定しています。

```
Dim Doc As Object
Dim Sheet As Object
Dim StyleFamilies As Object
Dim PageStyles As Object
Dim DefPage As Object

Doc = StarDesktop.CurrentComponent
StyleFamilies = Doc.StyleFamilies
PageStyles = StyleFamilies.getByName("PageStyles")
DefPage = PageStyles.getByName("Default")

DefPage.LeftMargin = 1000
DefPage.RightMargin = 1000
```

ヘッダとフッタ

ドキュメントのヘッダおよびフッタもページ属性として扱われるもので、これらも

```
com.sun.star.style.PageProperties
```

サービスで定義されています。ヘッダの書式設定には、次の属性を使用します。

HeaderIsOn (Boolean)

ヘッダを有効化。

HeaderLeftMargin (long)

100 分の 1 ミリ単位で指定した、左ページ余白からヘッダまでの距離。

HeaderRightMargin (long)

100 分の 1 ミリ単位で指定した、右ページ余白からヘッダまでの距離。

HeaderBodyDistance (long)

100 分の 1 ミリ単位で指定した、ページ本文領域からヘッダまでの距離。

HeaderHeight (long)

100 分の 1 ミリ単位で指定したヘッダの高さ。

HeaderIsDynamicHeight (Boolean)

ヘッダの高さを表示内容に自動的に合わせる指定。

HeaderLeftBorder (struct)

ヘッダ周囲の左側の外枠線の詳細指定 (

```
com.sun.star.table.BorderLine
```

構造体)。

HeaderRightBorder (struct)

ヘッダ周囲の右側の外枠線の詳細指定 (

```
com.sun.star.table.BorderLine
```

構造体)。

HeaderTopBorder (struct)

ヘッダ周囲の上側の外枠線の詳細指定 (

```
com.sun.star.table.BorderLine
```

構造体)。

HeaderBottomBorder (struct)

ヘッダ周囲の下側の外枠線の詳細指定 (

```
com.sun.star.table.BorderLine
```

構造体)。

HeaderLeftBorderDistance (long)

100 分の 1 ミリ単位で指定した、左側の外枠線からヘッダ本文までの距離。

HeaderRightBorderDistance (long)

100 分の 1 ミリ単位で指定した、右側の外枠線からヘッダ本文までの距離。

HeaderTopBorderDistance (long)

100 分の 1 ミリ単位で指定した、上側の外枠線からヘッダ本文までの距離。

HeaderBottomBorderDistance (long)

100 分の 1 ミリ単位で指定した、下側の外枠線からヘッダ本文までの距離。

HeaderIsShared (Boolean)

左右のページで共通のヘッダを使う指定
(HeaderText、HeaderTextLeft、HeaderTextRight を参照)。

HeaderBackColor (long)

ヘッダの背景色。

HeaderBackGraphicURL (String)

背景に表示する画像の URL。

HeaderBackGraphicFilter (String)

ヘッダの背景画像を解釈するフィルタ名。

HeaderBackGraphicLocation (Enum)

ヘッダの背景画像の位置 (

```
com.sun.star.style.GraphicLocation
```

の列挙型で定められた値)。

HeaderBackTransparent (Boolean)

ヘッダの背景を透明に表示する指定。

HeaderShadowFormat (struct)

ヘッダの影の詳細指定 (

```
com.sun.star.table.ShadowFormat
```

構造体)。

フッタの書式設定には、次の属性を使用します。

FooterIsOn (Boolean)

フッタを有効化。

FooterLeftMargin (long)

100 分の 1 ミリ単位で指定した、左ページ余白からフッタまでの距離。

FooterRightMargin (long)

100 分の 1 ミリ単位で指定した、右ページ余白からフッタまでの距離。

FooterBodyDistance (long)

100 分の 1 ミリ単位で指定した、ページ本文領域からフッタまでの距離。

FooterHeight (long)

100 分の 1 ミリ単位で指定したフッタの高さ。

FooterIsDynamicHeight (Boolean)

フッタの高さを表示内容に自動的に合わせる指定。

FooterLeftBorder (struct)

フッタ周囲の左側の外枠線の詳細指定 (

```
com.sun.star.table.BorderLine
```

構造体)。

FooterRightBorder (struct)

フッタ周囲の右側の外枠線の詳細指定 (

```
com.sun.star.table.BorderLine
```

構造体)。

FooterTopBorder (struct)

フッタ周囲の上側の外枠線の詳細指定 (

```
com.sun.star.table.BorderLine
```

構造体)。

FooterBottomBorder (struct)

フッタ周囲の下側の外枠線の詳細指定 (

```
com.sun.star.table.BorderLine
```

構造体)。

FooterLeftBorderDistance (long)

100 分の 1 ミリ単位で指定した、左側の外枠線からフッタ本文までの距離。

FooterRightBorderDistance (long)

100 分の 1 ミリ単位で指定した、右側の外枠線からフッタ本文までの距離。

FooterTopBorderDistance (long)

100 分の 1 ミリ単位で指定した、上側の外枠線からフッタ本文までの距離。

FooterBottomBorderDistance (long)

100 分の 1 ミリ単位で指定した、下側の外枠線からフッタ本文までの距離。

FooterIsShared (Boolean)

左右のページで共通のフッタを使う指定
(FooterText、FooterTextLeft、FooterTextRight を参照)。

FooterBackColor (long)

フッタの背景色。

FooterBackGraphicURL (String)

背景に表示する画像の URL。

FooterBackGraphicFilter (String)

フッタの背景画像を解釈するフィルタ名。

FooterBackGraphicLocation (Enum)

フッタの背景画像の位置 (

```
com.sun.star.style.GraphicLocation
```

の列挙型で定められた値)。

FooterBackTransparent (Boolean)

フッタの背景を透明に表示する指定。

FooterShadowFormat (struct)

フッタの影の詳細指定 (

```
com.sun.star.table.ShadowFormat
```

構造体)。

ヘッダおよびフッタの表示テキストの変更

表計算ドキュメントのヘッダやフッタの内容へアクセスするには、次の属性を使用します。

LeftPageHeaderContent (Object)

偶数ページのヘッダの表示内容 (

```
com.sun.star.sheet.HeaderFooterContent
```

サービス)。

RightPageHeaderContent (Object)

奇数ページのヘッダの表示内容 (

```
com.sun.star.sheet.HeaderFooterContent
```

サービス)。

LeftPageFooterContent (Object)

偶数ページのフッタの表示内容 (

```
com.sun.star.sheet.HeaderFooterContent
```

サービス)。

RightPageFooterContent (Object)

奇数ページのフッタの表示内容 (

```
com.sun.star.sheet.HeaderFooterContent
```

サービス)。

偶数ページと奇数ページで共通のヘッダやフッタを用いる場合は (`FooterIsShared` 属性に `False` を指定)、それぞれ奇数ページ用の属性を指定します。

これらのオブジェクトは、

```
com.sun.star.sheet.HeaderFooterContent
```

サービスをサポートしたオブジェクトを返します。このサービスでは、`LeftText`、`CenterText`、`RightText` という (疑似) 属性を用いて、OpenOffice.org Calc のヘッダやフッタに関する 3 種類のテキスト要素を利用できます。

次のサンプルコードを使うと、「デフォルト」テンプレートで、ヘッダの左側のテキストフィールドに「Just a Test.」と書き込まれます。

```
Dim Doc As Object
Dim Sheet As Object
Dim StyleFamilies As Object
Dim PageStyles As Object
Dim DefPage As Object
Dim HText As Object
Dim HContent As Object
Doc = StarDesktop.CurrentComponent
StyleFamilies = Doc.StyleFamilies
PageStyles = StyleFamilies.getByName("PageStyles")
DefPage = PageStyles.getByName("Default")

DefPage.HeaderIsOn = True
HContent = DefPage.RightPageHeaderContent
HText = HContent.LeftText
HText.String = "Just a Test."
DefPage.RightPageHeaderContent = HContent
```

サンプルコードの最後の行に注意してください。テキストを変更する場合、変更を有効にするには、TextContent オブジェクトをヘッダに再割り当てする必要があります。

文書ドキュメント (OpenOffice.org Writer) の場合、ヘッダやフッタは単一のテキストブロックから構成されているため、他の方法でヘッダやフッタの表示テキストを変更できます。次の属性は、

```
com.sun.star.style.PageProperties
```

サービスに定義されています。

HeaderText (Object)

ヘッダの内容を含むテキストオブジェクト (

```
com.sun.star.text.XText
```

サービス)。

HeaderTextLeft (Object)

左側のページのヘッダの内容を含むテキストオブジェクト (

```
com.sun.star.text.XText
```

サービス)。

HeaderTextRight (Object)

右側のページのヘッダの内容を含むテキストオブジェクト (

```
com.sun.star.text.XText
```

サービス)。

FooterText (Object)

フッタの内容を含むテキストオブジェクト (

```
com.sun.star.text.XText
```

サービス)。

FooterTextLeft (Object)

左側のページのフッタの内容を含むテキストオブジェクト (

```
com.sun.star.text.XText
```

サービス)。

FooterTextRight (Object)

右側のページのフッタの内容を含むテキストオブジェクト (

```
com.sun.star.text.XText
```

サービス)。

次のサンプルコードでは、文書ドキュメントの「標準」ページスタイルにヘッダを作成し、そのヘッダにテキスト「Just a Test.」を追加しています。

```
Dim Doc As Object
Dim Sheet As Object
Dim StyleFamilies As Object
Dim PageStyles As Object
Dim DefPage As Object
Dim HText As Object

Doc = StarDesktop.CurrentComponent
StyleFamilies = Doc.StyleFamilies
PageStyles = StyleFamilies.getByName("PageStyles")
DefPage = PageStyles.getByName("Default")

DefPage.HeaderIsOn = True
HText = DefPage.HeaderText

HText.String = "Just a Test."
```

この場合は、HeaderFooterContent オブジェクトではなく、ページスタイルの HeaderText 属性を使用してヘッダに直接アクセスしています。

中央揃え（表計算ドキュメントのみ）

```
com.sun.star.sheet.TablePageStyle
```

サービスは、OpenOffice.org Calc のページスタイルでのみ使用するもので、印刷するセル範囲をページの中央に配置させることができます。このサービスでは、次の属性を利用できます。

CenterHorizontally (Boolean)

表の内容を水平方向に中央揃え。

CenterVertically (Boolean)

表の内容を垂直方向に中央揃え。

印刷対象の指定（表計算ドキュメントのみ）

スプレッドシートの書式設定では、ページ上の各種要素を印刷させるかどうかを指定できます。

```
com.sun.star.sheet.TablePageStyle
```

サービスには、このような処理を行うために次の属性が用意されています。

PrintAnnotations (Boolean)

セルのコメントを印刷する指定。

PrintGrid (Boolean)

セルのグリッド線を印刷する指定。

PrintHeaders (Boolean)

行と列の見出しを印刷する指定。

PrintCharts (Boolean)

シートに含まれるグラフを印刷する指定。

PrintObjects (Boolean)

埋め込みオブジェクトを印刷する指定。

PrintDrawing (Boolean)

図形描画オブジェクトを印刷する指定。

PrintDownFirst (Boolean)

シートの印刷範囲が複数ページにまたがる場合に、上から下の方向に印刷してから右側のページという順番で印刷する指定。

PrintFormulas (Boolean)

計算された値ではなく数式を印刷する指定。

PrintZeroValues (Boolean)

ゼロの値を印刷する指定。

表計算ドキュメントの効率的な編集方法

これまでの節では、表計算ドキュメントの基本構造について説明しましたが、この節では個々のセルやセル範囲への効率的なアクセスについて説明します。

セル範囲

OpenOffice.org には、個々のセルを示すオブジェクト (

```
com.sun.star.table.Cell
```

サービス) の他に、セル範囲を示すオブジェクトも用意されています。これは CellRange オブジェクトと呼ばれるもので、スプレッドシートオブジェクトから getCellRangeByName を呼び出して作成します。

```
Dim Doc As Object
Dim Sheet As Object
Dim CellRange As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets.getByName("Sheet 1")
CellRange = Sheet.getCellRangeByName("A1:C15")
```

表計算ドキュメント上のセル範囲を指定するには、コロン記号 (:) を使用します。たとえば A1:C15 という指定は、列 A から 列 C の第 1 から第 15 行目のセル範囲を示します。

特定のセル範囲内にある個々のセル位置を指定するには、getCellByPosition メソッドを使用しますが、その際には左上隅のセルの座標は (0, 0) として扱われます。次のサンプルコードでは、このメソッドを利用して、C3 セルを示すオブジェクトを作成しています。

```
Dim Doc As Object
Dim Sheet As Object
Dim CellRange As Object
Dim Cell As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets.getByName("Sheet 1")
CellRange = Sheet.getCellRangeByName("B2:D4")
Cell = CellRange.GetCellByPosition(1, 1)
```

セル範囲の書式設定

個々のセルの場合と同様、セル範囲に対しても

```
com.sun.star.table.CellProperties
```

サービスを用いて書式設定を行うことができます。このサービスの詳細情報およびサンプルコードについては「表計算ドキュメントの書式設定」の節を参照してください。

セル範囲を利用した計算

セル範囲に対しては、computeFunction メソッドを用いて各種の算術計算を実行できます。この computeFunction のパラメータには、実行する計算処理を示す定数を渡します。こうした定数は、

```
com.sun.star.sheet.GeneralFunction
```

に定められています。指定可能な定数値は次のものです。

SUM
すべての数値の合計

COUNT
すべてのデータ数 (数値以外のデータも含む)

COUNTNUMS
数値データの数

AVERAGE
すべての数値の平均値

MAX
数値の最大値

MIN
数値の最小値

PRODUCT
すべての数値の積

STDEV
標準偏差

VAR
分散

STDEVP
母集団全体の標準偏差

VARP
母集団全体の分散

次のサンプルコードでは、セル範囲 A1:C3 の平均値を計算して、メッセージボックスに表示します。

```
Dim Doc As Object
Dim Sheet As Object
Dim CellRange As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets.getByName("Sheet 1")
CellRange = Sheet.getCellRangeByName("A1:C3")

MsgBox
CellRange.computeFunction(com.sun.star.sheet.GeneralFunction.AVERAGE)
```

セルの内容の削除

セルやセル範囲の内容を削除する場合、clearContents メソッドを利用すると、セル範囲中の特定のタイプの内容だけを消去することができます。

次のサンプルコードでは、セル範囲 B2:C3 から、すべての文字列および直接指定した書式設定情報を削除しています。

```
Dim Doc As Object
Dim Sheet As Object
Dim CellRange As Object
Dim Flags As Long

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)
CellRange = Sheet.getCellRangeByName("B2:C3")

Flags = com.sun.star.sheet.CellFlags.STRING + _
      com.sun.star.sheet.CellFlags.HARDATTR

CellRange.clearContents(Flags)
```

clearContents のフラグは、

```
com.sun.star.sheet.CellFlags
```

の定数のリストにより指定します。このような定数値としては、次の値を使用できます。

VALUE

日付や時刻として書式設定されていない数値

DATETIME

日付や時刻として書式設定されている数値

STRING

文字列

ANNOTATION

セルに付けられたコメント

FORMULA

数式

HARDATTR

セルに直接指定した書式

STYLES

間接的に設定した書式

OBJECTS

セルに配置された図形描画オブジェクト

EDITATTR

セル中の一部のテキストに対してのみ施された書式

clearContents による処理では、これらの定数を加算することにより同時指定することが可能で、該当する種類のデータをまとめて削除することもできます。

セルの内容の検索と置換

文書ドキュメントと同様に、表計算ドキュメントにも検索と置換の機能が用意されています。

表計算ドキュメントの場合、検索と置換のオプション指定用オブジェクトは、ドキュメントオブジェクトから直接作成するのではなく、Sheets のリストを使用する必要があります。次のサンプルコードは、検索と置換の実行例です。

```
Dim Doc As Object
Dim Sheet As Object
Dim ReplaceDescriptor As Object
Dim I As Integer

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)

ReplaceDescriptor = Sheet.createReplaceDescriptor()
ReplaceDescriptor.SearchString = "is"
ReplaceDescriptor.ReplaceString = "was"
For I = 0 to Doc.Sheets.Count - 1
   Sheet = Doc.Sheets(I)
   Sheet.ReplaceAll(ReplaceDescriptor)
Next I
```

このサンプルコードでは、最初のシートに対して ReplaceDescriptor を作成してからループに入り、すべてのシートを対象とした検索と置換の処理を行なっています。

図形描画とプレゼンテーション

本章では、マクロ制御による図形描画とプレゼンテーションドキュメントの作成と編集方法について説明します。

最初の節では、図形描画ドキュメントの構造について、これらを構成する基本要素なども含めて説明します。次の節では、オブジェクトのグループ化、回転、サイズ変更など、より複雑な操作方法を説明します。3 番目の節では、プレゼンテーションの処理を説明します。

図形描画ドキュメントの作成、オープン、保存については、「ドキュメントの使い方」を参照してください。

図形描画ドキュメントの構造

OpenOffice.org の図形描画ドキュメントには、特にページ数の制限はありません。これらの各ページは、それぞれ個別に編集できます。またページごとに配置できる図形描画要素の数についても、特に制限はありません。

ページ

図形描画ドキュメント内のページへのアクセスには、DrawPages リストを利用します。また個々のページに対しては、番号または名前により指定できます。ドキュメント内に **Slide 1** という名前のページだけしかない場合、次の 2 つのサンプルコードは同じ動作を示します。

例 **1:**

```
Dim Doc As Object
Dim Page As Object

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)
```

例 **2:**

```
Dim Doc As Object
Dim Page As Object

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages.getByName("Slide 1")
```

上記の例 1 では、ページへのアクセスを番号指定で行なっています (開始値は 0)。上記の例 2 では、getByName メソッドを用いた名前によるアクセスを行なっています。

```
Dim sUrl As String, sFilter As String
Dim sOptions As String
Dim oSheets As Object, oSheet As Object

   oSheets = oDocument.Sheets

   If oSheets.hasByName("Link") Then
      oSheet = oSheets.getByName("Link")
   Else
      oSheet =
oDocument.createInstance("com.sun.star.sheet.Spreadsheet")
      oSheets.insertByName("Link", oSheet)
      oSheet.IsVisible = False
   End If
```

上記の呼び出しを実行すると、com.sun.star.drawing.DrawPage サービスをサポートしたページオブジェクトが返されます。このサービスでは、次の属性を使用できます。

BorderLeft (Long)

100 分の 1 ミリ単位で指定した左側の枠。

BorderRight (Long)

100 分の 1 ミリ単位で指定した右側の枠。

BorderTop (Long)

100 分の 1 ミリ単位で指定した上側の枠。

BorderBottom (Long)

100 分の 1 ミリ単位で指定した下側の枠。

Width (Long)

100 分の 1 ミリ単位で指定したページ幅。

Height (Long)

100 分の 1 ミリ単位で指定したページの高さ。

Number (Short)

ページ数 (開始値は 1) 読み取り専用。

Orientation (Enum)

ページの向き (com.sun.star.view.PaperOrientation に定められた列挙型)。

これらの設定値の変更は、該当ドキュメント内のすべてのページに適用されます。

次のサンプルコードは、新規に開いた図形描画ドキュメントに対して、ページサイズを 20 × 20 センチメートル、ページ余白を 0.5 センチメートルに設定します。

```
Dim Doc As Object
Dim Page As Object

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

Page.BorderLeft = 500
Page.BorderRight = 500
Page.BorderTop = 500
Page.BorderBottom = 500

Page.Width = 20000
Page.Height = 20000
```

図形描画オブジェクトの基本属性

図形描画オブジェクトには、図形 (四角形や円など)、線、テキストなどのオブジェクトが該当します。これらのオブジェクトは com.sun.star.drawing.Shape サービスをサポートしており、多数の機能が共通しています。たとえば図形描画オブジェクトの Size や Position 属性は、このサービスに定義されています。

なお書式設定や塗りつぶしなど、位置やサイズ以外の属性の変更については、OpenOffice.org Basic に用意されているその他の各種サービスを利用します。どのような書式設定オプションが利用できるかは、図形描画オブジェクトの種類によって異なります。

次のサンプルコードでは、四角形を作成して、図形描画ドキュメント上に挿入します。

```
Dim Doc As Object
Dim Page As Object
Dim RectangleShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

RectangleShape =
Doc.createInstance("com.sun.star.drawing.RectangleShape")
RectangleShape.Size = Size
RectangleShape.Position = Point

Page.add(RectangleShape)
```

このサンプルコードでは、オープンするドキュメントの指定に StarDesktop.CurrentComponent を使用しています。このようにして指定されたドキュメントオブジェクトに対しては、drawPages(0) を実行することで、その最初のページを取得できます。

次に Point および Size という構造体を使って、図形描画オブジェクトの表示位置 (左上隅) とサイズを指定しています。これらの値は、100 分の 1 ミリ単位で指定します。

その次にくる Doc.createInstance のコード行では、

```
com.sun.star.drawing.RectangleShape
```

サービスを指定することで、四角形の図形描画オブジェクトを作成しています。最後に実行する Page.add のコード行では、作成した図形描画オブジェクトをページ上に挿入しています。

塗りつぶし属性

この節では 4 つのサービスに付いて説明し、四角形オブジェクトに対して各種の書式設定を行うサンプルコードを紹介します。塗りつぶし関係の属性は、

```
com.sun.star.drawing.FillProperties
```

サービスで扱われます。

OpenOffice.org で行う塗りつぶしに関しては、大きく分けて 4 種類の書式設定が存在します。最も単純なものは、単一色による塗りつぶしです。その他に、複数の色を組み合わせたグラデーションおよびハッチングのオプションもあります。そして第 4 のタイプとして、既存の画像を塗りつぶし領域にはめ込むというオプションも使用できます。

図形描画オブジェクトの塗りつぶしモードは、FillStyle 属性で指定します。指定可能な値は、

```
com.sun.star.drawing.FillStyle
```

に定義されています。

単一色による塗りつぶし

単一色による塗りつぶしを行う場合、主として次の属性を使用します。

FillColor (Long)

領域の塗りつぶし。

この塗りつぶしモードを使用するには、FillStyle 属性を SOLID に設定しておく必要があります。

次のサンプルコードでは、四角形オブジェクトを作成して、赤の単一色 (RGB 値: 255、0、0) で塗りつぶします。

```
Dim Doc As Object
Dim Page As Object
Dim RectangleShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

RectangleShape =
Doc.createInstance("com.sun.star.drawing.RectangleShape")
RectangleShape.Size = Size
RectangleShape.Position = Point

RectangleShape.FillStyle = com.sun.star.drawing.FillStyle.SOLID
RectangleShape.FillColor = RGB(255,0,0)

Page.add(RectangleShape)
```

色のグラデーション

FillStyle 属性に GRADIENT を設定すると、OpenOffice.org ドキュメント上の塗りつぶし領域に対して、色のグラデーションを施すことができます。

事前定義されたグラデーションを適用する場合は、FillTransparenceGradientName 属性に該当するグラデーション名を指定します。グラデーションを定義する場合は、

```
com.sun.star.awt.Gradient
```

構造体の形で必要な設定を指定して、FillGradient 属性に渡します。この属性には、次のオプションを使用できます。

Style (Enum)

線形や放射線状などのグラデーションの種類の指定 (

```
com.sun.star.awt.GradientStyle
```

に定められたデフォルト値)。

StartColor (Long)

グラデーションの開始色。

EndColor (Long)

グラデーションの開始色。

Angle (Short)

10 分の 1 度単位で指定したグラデーションの角度。

XOffset (Short)

100 分の 1 ミリ単位で指定した、グラデーション開始点の X 座標。

YOffset (Short)

100 分の 1 ミリ単位で指定した、グラデーション開始点の Y 座標。

StartIntensity (Short)

パーセント単位で指定した `StartColor` の強度 (OpenOffice.org Basic では 100 パーセント以上の値も指定可能)。

EndIntensity (Short)

パーセント単位で指定した `EndColor` の強度 (OpenOffice.org Basic では 100 パーセント以上の値も指定可能)。

StepCount (Short)

OpenOffice.org に計算させるグラデーションのステップ数。

次のサンプルコードでは、

```
com.sun.star.awt.Gradient
```

構造体を用いてグラデーションを施す方法を示します。

```
Dim Doc As Object
Dim Page As Object
Dim RectangleShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size
Dim Gradient As New com.sun.star.awt.Gradient

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

RectangleShape =
Doc.createInstance("com.sun.star.drawing.RectangleShape")
RectangleShape.Size = Size
RectangleShape.Position = Point
Gradient.Style = com.sun.star.awt.GradientStyle.LINEAR
Gradient.StartColor = RGB(255,0,0)
Gradient.EndColor = RGB(0,255,0)
Gradient.StartIntensity = 150
Gradient.EndIntensity = 150
Gradient.Angle = 450
Gradient.StepCount = 100

RectangleShape.FillStyle = com.sun.star.drawing.FillStyle.GRADIENT
RectangleShape.FillGradient = Gradient

Page.add(RectangleShape)
```

このサンプルコードでは、線形のグラデーションを施しています (Style = LINEAR)。グラデーションは開始色 (StartColor) を赤、終了色 (EndColor) を緑として、角度 (Angle) を 45 度にしています。開始色と終了色の強度 (StartIntensity および EndIntensity) はともに 150 パーセントとしてあるので、StartColor と EndColor 属性の指定色よりも明るく表示されます。グラデーションする色のステップ数は 100 段階としてあります (StepCount)。

ハッチング

塗りつぶしにハッチングを施す場合は、FillStyle 属性を HATCH に設定しておく必要があります。ハッチング用のプログラムコードは、グラデーションの場合とよく似た手順を取ります。ここでは、ハッチングの模様を指定するのに

```
com.sun.star.drawing.Hatch
```

構造体を使用します。こうしたハッチング用の構造体には、次の属性を指定できます。

Style (Enum)

横線、横縦線、横縦斜線などのハッチングの種類の指定
(com.sun.star.awt.HatchStyle に定められたデフォルト値)。

Color (Long)

線の色。

Distance (Long)

100 分の 1 ミリ単位で指定した、線の間隔。

Angle (Short)

10 分の 1 度単位で指定したハッチングの角度。

次のサンプルコードは、ハッチング指定用の構造体の使用法を示します。

```
Dim Doc As Object
Dim Page As Object
Dim RectangleShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size
Dim Hatch As New com.sun.star.drawing.Hatch

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

RectangleShape =
Doc.createInstance("com.sun.star.drawing.RectangleShape")
RectangleShape.Size = Size
RectangleShape.Position = Point

RectangleShape.FillStyle = com.sun.star.drawing.FillStyle.HATCH

Hatch.Style = com.sun.star.drawing.HatchStyle.SINGLE
Hatch.Color = RGB(64,64,64)
Hatch.Distance = 20
Hatch.Angle = 450

RectangleShape.FillHatch = Hatch

Page.add(RectangleShape)
```

このサンプルコードでは、線の種類を横線 (HatchStyle = SINGLE) にして、ハッチングの角度 (Angle) を 45 度にしています。また線の表示色 (Color) は灰色、線の間隔 (Distance) は 0.2 ミリメートルとしています。

ビットマップ

ビットマップによるプロジェクションを行うには、FillStyle 属性に BITMAP を指定する必要があります。必要なビットマップが OpenOffice.org 上にすでに配置されているのであれば、FillBitMapName 属性にその名前を指定し、FillBitmapMode 属性に表示スタイル (シンプル、繰り返し、拡大) を指定します (

```
com.sun.star.drawing.BitmapMode
```

に定められたデフォルト値)。

外部のビットマップファイルを使用する場合は、その URL を FillBitmapURL 属性に指定します。

次のサンプルコードでは、四角形を描画して、OpenOffice.org に用意されている Sky というビットマップで、その中を埋めています。

```
Dim Doc As Object
Dim Page As Object
Dim RectangleShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

RectangleShape =
Doc.createInstance("com.sun.star.drawing.RectangleShape")
RectangleShape.Size = Size
RectangleShape.Position = Point

RectangleShape.FillStyle = com.sun.star.drawing.FillStyle.BITMAP
RectangleShape.FillBitmapName = "Sky"
RectangleShape.FillBitmapMode = com.sun.star.drawing.BitmapMode.REPEAT

Page.add(RectangleShape)
```

透過性あり

塗りつぶしをする際には、その透過性を指定できます。最も簡単な指定法は、FillTransparence 属性を利用することです。

次のサンプルコードでは、赤色の四角形を描画し、その透過性を 50 パーセントに設定しています。

```
Dim Doc As Object
Dim Page As Object
Dim RectangleShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

RectangleShape =
Doc.createInstance("com.sun.star.drawing.RectangleShape")
RectangleShape.Size = Size
RectangleShape.Position = Point

RectangleShape.FillStyle = com.sun.star.drawing.FillStyle.SOLID
RectangleShape.FillTransparence = 50
RectangleShape.FillColor = RGB(255,0,0)

Page.add(RectangleShape)
```

FillTransparence 属性の値を 100 とすると、塗りつぶしは完全な透明になります。

```
com.sun.star.drawing.FillProperties
```

サービスには、FillTransparence 属性以外にも FillTransparenceGradient という属性が用意されています。この属性は、塗りつぶし領域のグラデーション指定に使用します。

線の属性

外枠の線を表示可能なすべてのオブジェクトは、

```
com.sun.star.drawing.LineStyle
```

サービスをサポートしています。このサービスに用意されている代表的な属性としては、次のものがあります。

LineStyle (Enum)

線の種類 (

```
com.sun.star.drawing.LineStyle
```

に定められたデフォルト値)。

LineColor (Long)

線の色。

LineTransparence (Short)

線の透過性。

LineWidth (Long)

100 分の 1 ミリ単位で指定した線の太さ。

LineJoint (Enum)

接続点への遷移 (

```
com.sun.star.drawing.LineJoint
```

に定められたデフォルト値)。

次のサンプルコードは外枠付きの四角形を描画するもので、線の種類を実線 (LineStyle = SOLID)、線の幅 (LineWidth) を 5 ミリメートルとし、線の透過性を 50 パーセントとしています。外枠線の頂点の形状は、直角にするように指定しています (LineJoint = MITER)。

```
Dim Doc As Object
Dim Page As Object
Dim RectangleShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

RectangleShape =
Doc.createInstance("com.sun.star.drawing.RectangleShape")
RectangleShape.Size = Size
RectangleShape.Position = Point

RectangleShape.LineColor = RGB(128,128,128)
RectangleShape.LineTransparence = 50
RectangleShape.LineWidth = 500
RectangleShape.LineJoint = com.sun.star.drawing.LineJoint.MITER

RectangleShape.LineStyle = com.sun.star.drawing.LineStyle.SOLID

Page.add(RectangleShape)
```

```
com.sun.star.drawing.LineStyle
```

サービスには、上記に一覧した属性以外にも、点線や破線を描画するためのオプションが用意されています。詳細情報については、OpenOffice.org の『API reference』を参照してください。

テキスト属性（図形描画オブジェクト）

```
com.sun.star.style.CharacterProperties
```

サービスおよび

```
com.sun.star.style.ParagraphProperties
```

サービスでは、図形描画オブジェクト内でテキストを書式設定できます。これらのサービスは、個々の文字や段落に関する指定を行うものですが、詳細については「文書ドキュメント」で説明しています。

次のサンプルコードでは、描画した四角形にテキストを挿入し、

```
com.sun.star.style.CharacterProperties
```

サービスを用いたフォントの書式指定を行なっています。

```
Dim Doc As Object
Dim Page As Object
Dim RectangleShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size
Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000
Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

RectangleShape =
Doc.createInstance("com.sun.star.drawing.RectangleShape")
RectangleShape.Size = Size
RectangleShape.Position = Point

Page.add(RectangleShape)

RectangleShape.String = "This is a test"
RectangleShape.CharWeight = com.sun.star.awt.FontWeight.BOLD
RectangleShape.CharFontName = "Arial"
```

上記のサンプルコードでは、四角形オブジェクトの `String` 属性を用いて表示テキストを指定してから、このテキストの書式設定を

```
com.sun.star.style.CharacterProperties
```

サービスの `CharWeight` および `CharFontName` 属性を利用して行なっています。

オブジェクト中へテキストを挿入するには、図形描画ページ上に該当するオブジェクトを事前に追加しておく必要があります。図形描画オブジェクト内でのテキストの表示位置と書式に関しては、

```
com.sun.star.drawing.Text
```

サービスも利用できます。このサービスに用意されている代表的な属性としては、次のものがあります。

TextAutoGrowHeight (Boolean)

図形描画要素の高さを、その中の表示テキストに合わせる指定。

TextAutoGrowWidth (Boolean)

図形描画要素の幅を、その中の表示テキストに合わせる指定。

TextHorizontalAdjust (Enum)

図形描画要素内のテキストの水平位置の指定 (

```
com.sun.star.drawing.TextHorizontalAdjust
```

に定められたデフォルト値)。

TextVerticalAdjust (Enum)

図形描画要素内のテキストの垂直位置の指定 (

```
com.sun.star.drawing.TextVerticalAdjust
```

に定められたデフォルト値)。

TextLeftDistance (Long)

100 分の 1 ミリ単位で指定した、図形描画要素の左端からテキストまでの間隔。

TextRightDistance (Long)

100 分の 1 ミリ単位で指定した、図形描画要素の右端からテキストまでの間隔。

TextUpperDistance (Long)

100 分の 1 ミリ単位で指定した、図形描画要素の上端からテキストまでの間隔。

TextLowerDistance (Long)

100 分の 1 ミリ単位で指定した、図形描画要素の下端からテキストまでの間隔。

次のサンプルコードは、これらの属性の使用例です。

```
Dim Doc As Object
Dim Page As Object
Dim RectangleShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

RectangleShape =
Doc.createInstance("com.sun.star.drawing.RectangleShape")
RectangleShape.Size = Size
RectangleShape.Position = Point

Page.add(RectangleShape)

RectangleShape.String = "This is a test"  ' May only take place after
Page.add!

RectangleShape.TextVerticalAdjust =
com.sun.star.drawing.TextVerticalAdjust.TOP
RectangleShape.TextHorizontalAdjust =
com.sun.star.drawing.TextHorizontalAdjust.LEFT

RectangleShape.TextLeftDistance = 300
RectangleShape.TextRightDistance = 300
RectangleShape.TextUpperDistance = 300
RectangleShape.TextLowerDistance = 300
```

このサンプルコードでは、ページ上に図形描画要素を挿入してからテキストを挿入し、このテキストの配置位置を TextVerticalAdjust および TextHorizontalAdjust 属性を用いて、図形描画オブジェクトの左上隅に設定しています。また図形描画要素の外形線からテキストまでの間隔は、3 ミリメートルとしています。

影の属性

```
com.sun.star.drawing.ShadowProperties
```

サービスを使用して、ほとんどの図形描画オブジェクトに影を追加できます。このサービスでは、次の属性を利用できます。

Shadow (Boolean)

影を有効化する指定。

ShadowColor (Long)

影の色。

ShadowTransparence (Short)

影を透明にする指定。

ShadowXDistance (Long)

100 分の 1 ミリ単位で指定した、影と図形描画オブジェクトの垂直方向の間隔。

ShadowYDistance (Long)

100 分の 1 ミリ単位で指定した、影と図形描画オブジェクトの水平方向の間隔。

次のサンプルコードでは、四角形のオブジェクトを作成し、水平および垂直方向のオフセットを 2 ミリメートルとした影を表示します。影の色は灰色、透過性は 50 パーセントとしています。

```
Dim Doc As Object
Dim Page As Object
Dim RectangleShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

RectangleShape =
Doc.createInstance("com.sun.star.drawing.RectangleShape")
RectangleShape.Size = Size
RectangleShape.Position = Point

RectangleShape.Shadow = True
RectangleShape.ShadowColor = RGB(192,192,192)
RectangleShape.ShadowTransparence = 50
RectangleShape.ShadowXDistance = 200
RectangleShape.ShadowYDistance = 200

Page.add(RectangleShape)
```

各種の図形描画オブジェクトの概要

四角形オブジェクト

四角形オブジェクト (

```
com.sun.star.drawing.RectangleShape
```

) は、書式設定オブジェクトの次のサービスをサポートしています。

Fill properties

```
com.sun.star.drawing.FillProperties
```

Line properties

```
com.sun.star.drawing.LineProperties
```

Text properties

```
com.sun.star.drawing.Text
```

(

```
com.sun.star.style.CharacterProperties
```

および

```
com.sun.star.style.ParagraphProperties
```

も含む)

Shadow properties

```
com.sun.star.drawing.ShadowProperties
```

CornerRadius (Long)

100 分の 1 ミリ単位で指定した、角の丸みの半径。

円および楕円オブジェクト

```
com.sun.star.drawing.EllipseShape
```

サービスは、円および楕円の表示関係の処理を行うもので、次のサービスをサポートしています。

Fill properties

```
com.sun.star.drawing.FillProperties
```

Line properties

```
com.sun.star.drawing.LineProperties
```

Text properties

```
com.sun.star.drawing.Text
```

(

```
com.sun.star.style.CharacterProperties
```

および

```
com.sun.star.style.ParagraphProperties
```

も含む)

Shadow properties

```
com.sun.star.drawing.ShadowProperties
```

円と楕円オブジェクトには、これらの他に次の 3 つの属性が用意されています。

CircleKind (Enum)

円または楕円の種類の指定 (

```
com.sun.star.drawing.CircleKind
```

に定められたデフォルト値)。

CircleStartAngle (Long)

10 分の 1 度単位で指定した、切片の開始角 (円および楕円形の切片の描画のみで有効)。

CircleEndAngle (Long)

10 分の 1 度単位で指定した、切片の終了角 (円および楕円形の切片の描画のみで有効)。

CircleKind 属性は、円全体を描画するか、あるいは切片や円弧として描画するかを指定します。指定可能な定数値は次のものです。

```
com.sun.star.drawing.CircleKind.FULL
```

円または楕円。

```
com.sun.star.drawing.CircleKind.CUT
```

円の切片 (円弧の両端を直線で結んだ図形)。

```
com.sun.star.drawing.CircleKind.SECTION
```

円の断面。

```
com.sun.star.drawing.CircleKind.ARC
```

角度 (円の線は含まない)。

次のサンプルコードは、中心角 70 度の扇型を描画します (開始角を 20 度、終了角 90 度に設定)。

```
Dim Doc As Object
Dim Page As Object
Dim EllipseShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

EllipseShape = Doc.createInstance("com.sun.star.drawing.EllipseShape")
EllipseShape.Size = Size
EllipseShape.Position = Point

EllipseShape.CircleStartAngle = 2000
EllipseShape.CircleEndAngle = 9000
EllipseShape.CircleKind =  com.sun.star.drawing.CircleKind.SECTION

Page.add(EllipseShape)
```

線

OpenOffice.org では、線のオブジェクトに対する

```
com.sun.star.drawing.LineShape
```

サービスが用意されています。線オブジェクトは、表面関係の指定を除いた、基本的な表示指定サービスをすべてサポートしています。LineShape サービスで利用する属性は次のものです。

Line properties

```
com.sun.star.drawing.LineProperties
```

Text properties

```
com.sun.star.drawing.Text
```

(

```
com.sun.star.style.CharacterProperties
```

および

```
com.sun.star.style.ParagraphProperties
```

も含む)

Shadow properties

```
com.sun.star.drawing.ShadowProperties
```

次のサンプルコードでは、線オブジェクトを作成し、上記の属性を用いて各種の表示を設定します。なお線の始点は Location 属性で指定しますが、終点は Size 属性の座標による間接的に指定します。

```
Dim Doc As Object
Dim Page As Object
Dim LineShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

LineShape = Doc.createInstance("com.sun.star.drawing.LineShape")
LineShape.Size = Size
LineShape.Position = Point

Page.add(LineShape)
```

多角形オブジェクト

OpenOffice.org は、

```
com.sun.star.drawing.PolyPolygonShape
```

サービスを使用した複雑な多角形オブジェクトもサポートしています。ただし正確には、ここでの PolyPolygon は単純な多角形ではなく、複合多角形を意味します。そのため、1 つのオブジェクト全体は、個々の頂点を指定する各種のリストから構成されます。

多角形オブジェクトに対しても、四角形同様の表示設定属性が各種用意されています。

Fill properties

```
com.sun.star.drawing.FillProperties
```

Line properties

```
com.sun.star.drawing.LineProperties
```

Text properties

```
com.sun.star.drawing.Text
```

(

```
com.sun.star.style.CharacterProperties
```

および

```
com.sun.star.style.ParagraphProperties
```

も含む)

Shadow properties

```
com.sun.star.drawing.ShadowProperties
```

また PolyPolygonShape サービスには、多角形の座標指定用に、次の属性も用意されています。

・PolyPolygon (Array) – 多角形の座標を指定するフィールド (

```
com.sun.star.awt.Point
```

タイプのデータを格納する配列)

次のサンプルコードは、PolyPolygonShape サービスを用いて三角形を描画する方法の一例です。

```
Dim Doc As Object
Dim Page As Object
Dim PolyPolygonShape As Object
Dim PolyPolygon As Variant
Dim Coordinates(2) As New com.sun.star.awt.Point

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

PolyPolygonShape =
Doc.createInstance("com.sun.star.drawing.PolyPolygonShape")
Page.add(PolyPolygonShape)   ' Page.add must take place before the
coordinates are set

Coordinates(0).x = 1000
Coordinates(1).x = 7500
Coordinates(2).x = 10000
Coordinates(0).y = 1000
Coordinates(1).y = 7500
Coordinates(2).y = 5000

PolyPolygonShape.PolyPolygon = Array(Coordinates())
```

ここでの多角形は、個々の頂点を絶対座標により指定するため、多角形のサイズや表示位置などの指定は不要です。その代わりここでは、頂点の座標データを収めた配列を用意して、これを第 2 の配列に変換 (Array(Coordinates() による処理) してから、多角形の頂点指定用属性に代入しています。なお多角形への頂点指定を行う前には、多角形オブジェクトをドキュメント上に挿入しておく必要があります。

頂点指定用の配列を利用することで、複数の多角形を組み合わせた図形を構築できます。たとえば、四角形の中に別の四角形を挿入することで、穴の空いた四角形を描画できます。

```
Dim Doc As Object
Dim Page As Object
Dim PolyPolygonShape As Object
Dim PolyPolygon As Variant
Dim Square1(3) As New com.sun.star.awt.Point
Dim Square2(3) As New com.sun.star.awt.Point
Dim Square3(3) As New com.sun.star.awt.Point

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

PolyPolygonShape =
Doc.createInstance("com.sun.star.drawing.PolyPolygonShape")

Page.add(PolyPolygonShape)   ' Page.add must take place before the
coordinates are set

Square1(0).x = 5000
Square1(1).x = 10000
Square1(2).x = 10000
Square1(3).x = 5000
Square1(0).y = 5000
Square1(1).y = 5000
Square1(2).y = 10000
Square1(3).y = 10000

Square2(0).x = 6500
Square2(1).x = 8500
Square2(2).x = 8500
Square2(3).x = 6500
Square2(0).y = 6500
Square2(1).y = 6500
Square2(2).y = 8500
Square2(3).y = 8500

Square3(0).x = 6500
Square3(1).x = 8500
Square3(2).x = 8500
Square3(3).x = 6500
Square3(0).y = 9000
Square3(1).y = 9000
Square3(2).y = 9500
Square3(3).y = 9500

PolyPolygonShape.PolyPolygon = Array(Square1(), Square2(), Square3())
```

このような操作を行う場合、どの領域が残り、どの領域が消去されるかが問題となりますが、OpenOffice.org では「外部のオブジェクトの縁が多角形の外周部となる」という規則に従って処理されます。同様に、内側の縁は多角形の内周部となるので、こうした部分に穴が空くことになります。更に内側にくる縁がある場合は、そこから塗りつぶされた領域の描画が再開されます。

図オブジェクト

最後に説明する図形描画要素は図オブジェクトです。これは

```
com.sun.star.drawing.GraphicObjectShape
```

サービスを利用します。このサービスは、OpenOffice.org のすべての画像に対して利用可能で、各種の属性を通じて画像の表示に関する設定を行います。

図オブジェクトは、次の 2 つの表示設定用属性をサポートしています。

Text properties

```
com.sun.star.drawing.Text
```

(

```
com.sun.star.style.CharacterProperties
```

および

```
com.sun.star.style.ParagraphProperties
```

も含む)

Shadow properties

```
com.sun.star.drawing.ShadowProperties
```

図オブジェクトのサポートするその他の属性には、次のものがあります。

GraphicURL (String)

図の URL。

AdjustLuminance (Short)

パーセント単位で指定した、色の輝度 (負の値も指定可能)。

AdjustContrast (Short)

パーセント単位で指定した、色のコントラスト (負の値も指定可能)。

AdjustRed (Short)

パーセント単位で指定した、色の赤成分 (負の値も指定可能)。

AdjustGreen (Short)

パーセント単位で指定した、色の緑成分 (負の値も指定可能)。

AdjustBlue (Short)

パーセント単位で指定した、色の青成分 (負の値も指定可能)。

Gamma (Short)

図のガンマ値。

Transparency (Short)

パーセント単位で指定した図の透過性。

GraphicColorMode (enum)

標準、グレースケール、白黒などのカラーモード (

```
com.sun.star.drawing.ColorMode
```

に定められたデフォルト値)。

次のサンプルコードは、ページへの図オブジェクトの挿入法の例です。Dim Doc As Object

```
Dim Page As Object
Dim GraphicObjectShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size

Point.x = 1000          ' specifications, insignificant because latter
                        coordinates are binding
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

GraphicObjectShape =
Doc.createInstance("com.sun.star.drawing.GraphicObjectShape")

GraphicObjectShape.Size = Size
GraphicObjectShape.Position = Point

GraphicObjectShape.GraphicURL = "file:///c:/test.jpg"
GraphicObjectShape.AdjustBlue = -50
GraphicObjectShape.AdjustGreen = 5
GraphicObjectShape.AdjustBlue = 10
GraphicObjectShape.AdjustContrast = 20
GraphicObjectShape.AdjustLuminance = 50
GraphicObjectShape.Transparency = 40
GraphicObjectShape.GraphicColorMode =
com.sun.star.drawing.ColorMode.STANDARD

Page.add(GraphicObjectShape)
```

上記のサンプルコードでは、test.jpg というファイル名の画像を挿入して、その表示設定を Adjust 関係の属性により行なっています。またここでは、画像の透過性を 40 パーセントとし、カラーモードは標準のままとしています (GraphicColorMode = STANDARD)。

図形描画オブジェクトの編集

オブジェクトのグループ化

複数の図形描画オブジェクトをグループ化して、1 つのオブジェクトとして扱えると便利な場合があります。

次のサンプルコードは、2 つのオブジェクトをグループ化します。

```
Dim Doc As Object
Dim Page As Object
Dim Square As Object
Dim Circle As Object
Dim Shapes As Object
Dim Group As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size
Dim NewPos As New com.sun.star.awt.Point
Dim Height As Long
Dim Width As Long

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)
Point.x = 3000
Point.y = 3000
Size.Width = 3000
Size.Height = 3000
' create square drawing element
Square = Doc.createInstance("com.sun.star.drawing.RectangleShape")
Square.Size = Size
Square.Position = Point
Square.FillColor = RGB(255,128,128)
Page.add(Square)

' create circle drawing element
Circle = Doc.createInstance("com.sun.star.drawing.EllipseShape")
Circle.Size = Size
Circle.Position = Point
Circle.FillColor = RGB(255,128,128)
Circle.FillColor = RGB(0,255,0)
Page.add(Circle)

' combine square and circle drawing elements
Shapes = createUnoService("com.sun.star.drawing.ShapeCollection")
Shapes.add(Square)

Shapes.add(Circle)
Group = Page.group(Shapes)
' centre combined drawing elements
Height = Page.Height
Width = Page.Width
NewPos.X = Width / 2
NewPos.Y = Height / 2
Height = Group.Size.Height
Width = Group.Size.Width
NewPos.X = NewPos.X - Width / 2
NewPos.Y = NewPos.Y - Height / 2
Group.Position = NewPos
```

上記のサンプルコードでは、まず四角形および円形のオブジェクトを作成し、ページへ挿入しています。そして次に、

```
com.sun.star.drawing.ShapeCollection
```

というサービスをサポートしたオブジェクトを新規に作成し、先に作成しておいた四角形と円形のオブジェクトを Add メソッドを用いてこの新規オブジェクトに追加しています。そして Group というメソッドを使用し、この ShapeCollection のサポートオブジェクトをページに挿入することにより、実際の Group オブジェクトを作成しています。このグループ化オブジェクトは、1 つの独立した Shape 図形描画オブジェクトと同様な操作が行えます。

グループ化する各オブジェクトに対する表示設定は、グループ化を行う前に実行しておく必要があります。いったんグループ化すると、このようなオブジェクトに変更を加えることはできません。

図形描画オブジェクトの回転と傾斜

これまでの節で説明した図形描画オブジェクトに対しては、

```
com.sun.star.drawing.RotationDescriptor
```

サービスを用いて回転させたり傾斜させたりすることができます。

このサービスには、次の属性が用意されています。

RotateAngle (Long)

100 分の 1 度単位で指定した回転角度。

ShearAngle (Long)

100 分の 1 度単位で指定したせん断角。

次のサンプルコードでは、四角形を作成して、RotateAngle 属性により 30 度回転させます。

```
Dim Doc As Object
Dim Page As Object
Dim RectangleShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

RectangleShape =
Doc.createInstance("com.sun.star.drawing.RectangleShape")
RectangleShape.Size = Size
RectangleShape.Position = Point

RectangleShape.RotateAngle = 3000

Page.add(RectangleShape)
```

次のサンプルコードでも上記と同様の四角形を作成していますが、ここでは ShearAngle 属性による 30 度の傾斜を与えています。

```
Dim Doc As Object
Dim Page As Object
Dim RectangleShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)
RectangleShape =
Doc.createInstance("com.sun.star.drawing.RectangleShape")
RectangleShape.Size = Size
RectangleShape.Position = Point

RectangleShape.ShearAngle = 3000

Page.add(RectangleShape)
```

検索と置換

図形描画ドキュメントにも、文書ドキュメント同様の検索および置換用の機能が用意されています。実際このような検索と置換は、「文書ドキュメント」で説明した、文書ドキュメントの該当する機能とよく似ています。ただし図形描画ドキュメントの場合、検索と置換オプションの指定用オブジェクトは、ドキュメントオブジェクトを通じて直接作成するのではなく、対応する文字レベルを用いて作成します。次のサンプルコードは、このような検索と置換の手順を説明するために用意した例です。

```
Dim Doc As Object
Dim Page As Object
Dim ReplaceDescriptor As Object
Dim I As Integer

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

ReplaceDescriptor = Page.createReplaceDescriptor()
ReplaceDescriptor.SearchString = "is"
ReplaceDescriptor.ReplaceString = "was"

For I = 0 to Doc.drawPages.Count - 1
   Page = Doc.drawPages(I)
   Page.ReplaceAll(ReplaceDescriptor)
Next I
```

上記のサンプルコードでは、図形描画ドキュメント上の最初の DrawPage を対象として ReplaceDescriptor を作成してからループに入り、該当ドキュメントのすべてのページを対象とする処理を行なっています。

プレゼンテーション

OpenOffice.org のプレゼンテーションは、図形描画ドキュメントが基になります。プレゼンテーションを構成する各スライドは、ページとも呼びます。ドキュメントオブジェクトの DrawPages リストを使用して標準の図形描画にアクセスするのと同じ方法で、スライドにアクセスできます。プレゼンテーションドキュメントの処理に利用する

```
com.sun.star.presentation.PresentationDocument
```

サービスは、完全な

```
com.sun.star.drawing.DrawingDocument
```

サービスも提供します。

プレゼンテーションの操作

プレゼンテーションドキュメントには、Presentation プロパティーによる図形描画機能の他に、プレゼンテーションオブジェクトがあり、プレゼンテーションで利用する主要なプロパティーやコントロールへのアクセスには、このオブジェクトを利用します。たとえば、プレゼンテーションを開始する際の start メソッドは、このオブジェクトを利用して実行します。

```
Dim Doc As Object
Dim Presentation As Object

Doc = StarDesktop.CurrentComponent
Presentation = Doc.Presentation
Presentation.start()
```

上記のサンプルコードでは、現在のプレゼンテーションドキュメントを参照する Doc オブジェクトを作成し、このオブジェクトを通じて、該当するプレゼンテーションオブジェクトを取得しています。このオブジェクトの start() メソッドを使用して、サンプルを開始し、画面プレゼンテーションを実行しています。

プレゼンテーションオブジェクトには、次のメソッドが用意されています。

start

プレゼンテーションを開始します。

end

プレゼンテーションを終了します。

rehearseTimings

先頭からプレゼンテーションを開始し、ランタイムを設定します。

また次のプロパティーも使用できます。

AllowAnimations (Boolean)

プレゼンテーションでアニメーションを事項します。

CustomShow (String)

プレゼンテーションで名前を参照できるよう、プレゼンテーションの名前を指定できるようにします。

FirstPage (String)

プレゼンテーションを開始するスライドの名前。

IsAlwaysOnTop (Boolean)

常にプレゼンテーションウィンドウを画面の一番手前のウィンドウとして表示します。

IsAutomatic (Boolean)

プレゼンテーションを自動的に実行します。

IsEndless (Boolean)

終了時に、プレゼンテーションを最初から再実行します。

IsFullScreen (Boolean)

プレゼンテーションを全画面モードで自動的に開始します。

IsMouseVisible (Boolean)

プレゼンテーション中にマウスを表示します。

Pause (long)

プレゼンテーション終了時に表示するブランク画面の時間。

StartWithNavigator (Boolean)

プレゼンテーション開始時にナビゲータウィンドウを表示します。

UsePn (Boolean)

プレゼンテーション中にポインタを表示します。

グラフ（図）

OpenOffice.org にはデータをグラフ化して表示する機能が用意されており、棒グラフ、円グラフ、折れ線グラフなどの表が作成可能です。データの表示は 2D グラフか 3D グラフを選択でき、これらグラフを構成する各要素に関しても、図形描画要素の場合と同様の方法で表示設定が行えます。

OpenOffice.org でのグラフは、独立したドキュメントとしてではなく、既存のドキュメント内に埋め込まれたオブジェクトとして扱われます。

グラフには独自のデータを含めたり、コンテナドキュメントからのデータを表示することができます。たとえば、表計算ドキュメント内のグラフでは、セル範囲から取得したデータを表示でき、文書ドキュメント内ではライターテーブルから取得したデータを表示できます。

表計算ドキュメント内のグラフの使用

表計算ドキュメント内のグラフでは、表計算ドキュメント内で割り当てられたセル範囲からデータを表示できます。表計算ドキュメント内でデータに加えられた変更は、割り当てられたグラフにも反映されます。次のサンプルコードは、表計算ドキュメント内のセル範囲に割り当てられたグラフの作成方法の例です。

```
Dim Doc As Object
Dim Charts As Object
Dim Chart as Object
Dim Rect As New com.sun.star.awt.Rectangle
Dim RangeAddress(0) As New com.sun.star.table.CellRangeAddress

Doc = StarDesktop.CurrentComponent
Charts = Doc.Sheets(0).Charts

Rect.X = 8000
Rect.Y = 1000
Rect.Width = 10000
Rect.Height = 7000
RangeAddress(0).Sheet = 0
RangeAddress(0).StartColumn = 0
RangeAddress(0).StartRow = 0
RangeAddress(0).EndColumn = 2
RangeAddress(0).EndRow = 12

Charts.addNewByName("MyChart", Rect, RangeAddress(), True, True)
```

上記のサンプルコードは一見するとかなり複雑に感じられますが、主要な処理は 3 行に集約されます。その 1 つ目は、現在の表計算ドキュメントを参照するために、Doc というドキュメント変数を用意する行です (Doc= StarDesktop.CurrentComponent)。2 つ目は、表計算ドキュメントの最初の表 (スプレッドシート) 上を対象に、存在するすべてのグラフを登録したリストを作成する行です (Charts= Doc.Sheets(0).Charts)。そして、最後の行に addNewByName メソッドを使用して、このリストに新しいグラフを追加します。新規に作成したグラフは、この処理を経て初めて画面上に表示されます。変数 RangeAddress により、割り当てられたセル範囲が指定され、グラフ内にそのデータが表示されます。変数 Rect により、表計算ドキュメントの最初のシート内のグラフの位置とサイズが指定されます。

上記のサンプルコードのままでは、作成されるグラフは常に縦棒グラフとなります。作成したグラフの種類を棒グラフ以外に変更するには、次のようなコードを追加して、表示するグラフの種類を明示的に指定する必要があります。

```
Chart = Charts.getByName("MyChart").embeddedObject
Chart.Diagram = Chart.createInstance("com.sun.star.chart.LineDiagram")
```

上記のコードの最初の行は、対象とするグラフオブジェクトを特定するための処理です。その次の行は、作成済みのグラフの種類を変更する処理で、この場合は折れ線グラフを指定しています。

Microsoft Excel の場合、ドキュメント中のワークシート (スプレッドシート) として挿入したグラフと、ワークシート上に埋め込んだグラフとは、明確に区別されています。そのため、これらのグラフへのアクセスに関しても、異なる 2 通りの方法が用意されています。これに対して OpenOffice.org Basic の場合、OpenOffice.org Calc のグラフは、常に表 (スプレッドシート) への埋め込みグラフとして作成されます。このためグラフへのアクセスも、常に `Sheet` オブジェクトの `Charts` リストを用いて行います。

グラフの構造

グラフの構造および、サポートするサービスやインターフェースは、個々のグラフの種類ごとに異なります。たとえば、メソッドと Z 軸の属性は、3D グラフでは使用できますが、軸を操作するインターフェースがないため、2D グラフや円グラフでは使用できません。

タイトル、サブタイトルおよび凡例

タイトル、サブタイトルおよび凡例は、基本要素としてグラフ作成時に使用できます。そして Chart オブジェクトには、このようなオブジェクトの操作用に、次の属性が用意されています。

HasMainTitle (Boolean)

タイトルを有効化する指定。

Title (Object)

グラフのタイトルに関する詳細な情報のオブジェクト (

```
com.sun.star.chart.ChartTitle
```

サービスをサポート)。

HasSubTitle(Boolean)

サブタイトルを有効化する指定。

Subtitle (Object)

グラフのサブタイトルに関する詳細な情報のオブジェクト (

```
com.sun.star.chart.ChartTitle
```

サービスをサポート)。

HasLegend (Boolean)

凡例を有効化する指定。

Legend (Object)

凡例に関する詳細な情報のオブジェクト (

```
com.sun.star.chart.ChartLegend
```

サービスをサポート)。

com.sun.star.chart.ChartTitle サービスおよび
com.sun.star.chart.ChartLegend サービスの両方が
com.sun.star.drawing.Shape サービスをサポートしています。このため、Position 属性および Size 属性を使用して、要素の位置とサイズを指定できます。凡例とタイトルのサイズは、現在の表示内容と文字の高さなどに基づいて自動的に計算されるため、サイズのプロパティーは読み取りアクセスのみが許可されています。

要素の詳細な書式設定には、塗りつぶしと線の属性 (

```
com.sun.star.drawing.FillProperties
```

サービスおよび

```
com.sun.star.drawing.LineProperties
```

サービス) と文字の属性 (

```
com.sun.star.style.CharacterProperties
```

サービス) が提供されています。

```
com.sun.star.chart.ChartTitle
```

には、一覧表示される書式設定の属性だけでなく、他 2 つの属性も含まれています。

String (String)
タイトルまたはサブタイトルとして表示するテキスト。

TextRotation (Long)
100 分の 1 度単位で指定したテキストの回転角。

凡例 (

```
com.sun.star.chart.ChartLegend
```

) には、次の追加属性が含まれています。

Alignment (Enum)
凡例を表示する位置 (

```
com.sun.star.chart.ChartLegendPosition
```

タイプの値)。

次のサンプルコードは、「Main Title String」というタイトル、「Subtitle String」というサブタイトル、および凡例を持つグラフを作成します。凡例については、背景色を灰色、表示位置をグラフの下部、テキストサイズを 7 ポイントとしています。

```
Dim Doc As Object
Dim Charts As Object
Dim Chart as Object
Dim Rect As New com.sun.star.awt.Rectangle
Dim RangeAddress(0) As New com.sun.star.table.CellRangeAddress

Rect.X = 8000
Rect.Y = 1000
Rect.Width = 10000
Rect.Height = 7000
RangeAddress(0).Sheet = 0
RangeAddress(0).StartColumn = 0
RangeAddress(0).StartRow = 0
RangeAddress(0).EndColumn = 2
RangeAddress(0).EndRow = 12

Doc = StarDesktop.CurrentComponent

Charts = Doc.Sheets(0).Charts
Charts.addNewByName("MyChart", Rect, RangeAddress(), True, True)
Chart = Charts.getByName("MyChart").EmbeddedObject
Chart.HasMainTitle = True
Chart.Title.String = "Main Title String"
Chart.HasSubTitle = True
Chart.Subtitle.String = "Subtitle String"
Chart.HasLegend = True
Chart.Legend.Alignment = com.sun.star.chart.ChartLegendPosition.BOTTOM
Chart.Legend.FillStyle = com.sun.star.drawing.FillStyle.SOLID
Chart.Legend.FillColor = RGB(210, 210, 210)
Chart.Legend.CharHeight = 7
```

背景

各グラフには、背景表示用の領域 (グラフエリア) があります。Chart オブジェクトは、背景を書式設定する Area 属性を提供します。

Area (Object)

グラフの背景表示用の領域 (

```
com.sun.star.chart.ChartArea
```

サービスをサポート)。

ここで言うグラフの背景 (グラフエリア) とは、タイトル、サブタイトル、凡例などの表示位置も含めた、グラフ全体をカバーする領域を指します。関連する

```
com.sun.star.chart.ChartArea
```

サービスは、線と塗りつぶしの属性をサポートしています。

ダイアグラム

Chart オブジェクトは、最終的にデータを表示する軸と目盛線のある座標系を形成する Diagram 属性を提供します。

Diagram (Object)

データがプロットされる座標系を形成するオブジェクト。

```
com.sun.star.chart.Diagram
```

サービスと以下をサポートします。

1.

```
com.sun.star.chart.StackableDiagram
```

2.

```
com.sun.star.chart.ChartAxisXSupplier
```

3.

```
com.sun.star.chart.ChartAxisYSupplier
```

4.

```
com.sun.star.chart.ChartAxisZSupplier
```

5.

```
com.sun.star.chart.ChartTwoAxisXSupplier
```

6.

```
com.sun.star.chart.ChartTwoAxisYSupplier
```

グラフの種類によって、さまざまなサービスがサポートされています (「グラフの種類」参照)。

壁面と床面

グラフの壁面とは、座標系の背景で、ここにデータがプロットされます。3D グラフの壁面は、通常、プロットされたデータの後に 1 つ、左または右の境界設定として 1 つあります。これは、グラフの回転によって異なります。3D グラフには、床面も表示されます。

Diagram オブジェクトによって、壁面と床面の属性が提供されています。

Wall (Object)

座標系の背景の壁面 (

```
com.sun.star.chart.ChartArea
```

サービスをサポート)。

Floor (Object)

座標系の床面パネル (3D グラフのみ。

```
com.sun.star.chart.ChartArea
```

サービスをサポート)。

指定したオブジェクトで com.sun.star.chart.ChartArea サービスをサポートし、通常の塗りつぶしと線の属性を提供します (

```
com.sun.star.drawing.FillProperties
```

サービスおよび

```
com.sun.star.drawing.LineProperties
```

サービスをサポート。「図形描画とプレゼンテーション」参照)。

次のサンプルコードでは、グラフの背景として、OpenOffice.org に標準で用意されている画像 (名称 Sky) を表示させています。壁面の表示色は、青に設定されています。

```
Dim Doc As Object
Dim Charts As Object
Dim Chart as Object
Dim Rect As New com.sun.star.awt.Rectangle
Dim RangeAddress(0) As New com.sun.star.table.CellRangeAddress

Rect.X = 8000
Rect.Y = 1000
Rect.Width = 10000
Rect.Height = 7000
RangeAddress(0).Sheet = 0
RangeAddress(0).StartColumn = 0
RangeAddress(0).StartRow = 0
RangeAddress(0).EndColumn = 2
RangeAddress(0).EndRow = 12

Doc = StarDesktop.CurrentComponent

Charts = Doc.Sheets(0).Charts
Charts.addNewByName("MyChart", Rect, RangeAddress(), True, True)
Chart = Charts.getByName("MyChart").EmbeddedObject
Chart.Area.FillStyle = com.sun.star.drawing.FillStyle.BITMAP
Chart.Area.FillBitmapName = "Sky"
Chart.Area.FillBitmapMode = com.sun.star.drawing.BitmapMode.REPEAT

Chart.Diagram.Wall.FillStyle = com.sun.star.drawing.FillStyle.SOLID
Chart.Diagram.Wall.FillColor = RGB(00,132,209)
```

グラフ軸

OpenOffice.org では、5 種類の軸が認識されグラフに使用できます。通常表示されるのは、X 軸と Y 軸です。3D グラフの中には、Z 軸が表示されるものもあります。また個々のデータ列間が相互に異なる値をもつような場合、OpenOffice.org では X 軸と Y 軸の第 2 数値軸を表示させることができます。

Diagram オブジェクトには、軸へのアクセス用に、次の属性が用意されています。

HasXAxis (Boolean)

X 軸を有効化する指定。

XAxis (Object)

X 軸に関する詳細な情報のオブジェクト (

```
com.sun.star.chart.ChartAxis
```

サービスをサポート)。

HasXAxisDescription (Boolean)

X 軸のラベルと区切りを有効化する指定。

HasYAxis (Boolean)

Y 軸を有効化する指定。

YAxis (Object)

Y 軸に関する詳細な情報のオブジェクト (

```
com.sun.star.chart.ChartAxis
```

サービスをサポート)。

HasYAxisDescription (Boolean)

Y 軸のラベルと区切りを有効化する指定。

HasZAxis (Boolean)

Z 軸を有効化する指定。

ZAxis (Object)

Z 軸に関する詳細な情報のオブジェクト (

```
com.sun.star.chart.ChartAxis
```

サービスをサポート)。

HasZAxisDescription (Boolean)

Z 軸のラベルと区切りを有効化する指定。

HasSecondaryXAxis (Boolean)

第 2X 軸を有効化する指定。

SecondaryXAxis (Object)

第 2X 軸に関する詳細な情報のオブジェクト (

```
com.sun.star.chart.ChartAxis
```

サービスをサポート)。

HasSecondaryXAxisDescription (Boolean)

第 2X 軸のラベルと区切りを有効化する指定。

HasSecondaryYAxis (Boolean)

第 2Y 軸を有効化する指定。

SecondaryYAxis (Object)

第 2Y 軸に関する詳細な情報のオブジェクト (

```
com.sun.star.chart.ChartAxis
```

サービスをサポート)。

HasSecondaryYAxisDescription (Boolean)

第 2X 軸のラベルと区切りを有効化する指定。

軸の属性

OpenOffice.org グラフの軸オブジェクトは、

```
com.sun.star.chart.ChartAxis
```

サービスをサポートしています。ここでは文字 (

```
com.sun.star.style.CharacterProperties
```

サービスについては「文書ドキュメント」を参照) および線 (

```
com.sun.star.drawing.LineStyle
```

サービスについては「図形描画とプレゼンテーション」を参照) に関係する属性に加えて、次の属性が用意されています。

スケーリングの属性

Max (Double)

軸の最大値。

Min (Double)

軸の最小値。

Origin (Double)

交差する軸の交点。

StepMain (Double)

軸の区切りの間の距離。

StepHelp (Double)

補助区切りの間の距離 (OpenOffice.org 3.0 以降非推奨。StepHelpCount 属性で代用)。

StepHelpCount (Long)

軸の区切り内の補助区切りの数。たとえば、StepHelpCount を 5 にすると、軸の区切りが 5 つに分かれて、補助区切りが 4 つになります。(OpenOffice.org 3.0 以降で使用可能)。

AutoMax (Boolean)

true の設定で軸の最大値を自動計算。

AutoMin (Boolean)

true の設定で軸の最小値を自動計算。

AutoOrigin (Boolean)

true の設定で原点を自動設定。

AutoStepMain (Boolean)

true の設定で StepMain を自動設定。

AutoStepHelp (Boolean)

true の設定で StepHelpCount を自動設定。

Logarithmic (Boolean)

線形ではなく対数を使用して軸を拡大/縮小。

ReverseDirection (Boolean)

軸の方向を数学的にするか逆方向にするかを指定。(OpenOffice.org 2.4 以降で使用可能)。

ラベルの属性

DisplayLabels (Boolean)

区切りのテキストラベルを有効化する指定。

TextRotation (Long)

100 分の 1 度単位で指定したテキストラベルの回転角。

ArrangeOrder (enum)

ジグザグになる可能性のあるラベルを 2 本の線上に交互に配置する指定 (

```
com.sun.star.chart.ChartAxisArrangeOrderType
```

で定められた値)。

TextBreak (Boolean)

軸ラベル内の行ブレークを許可する指定。

TextCanOverlap (Boolean)

軸ラベルの重ね合わせを許可する指定。

NumberFormat (Long)

軸ラベルで使用する数の書式。

LinkNumberFormatToSource (Boolean)

コンテナドキュメントで与えられた数の書式を使用するか属性 NumberFormat を使用するかの指定。(OpenOffice.org 2.3 以降)。

区切りの属性:

Marks (Const)

軸の区切りの位置の指定 (

```
com.sun.star.chart.ChartAxisMarks
```

に定められた値)。

HelpMarks (Const)

補助区切りの位置の指定 (

```
com.sun.star.chart.ChartAxisMarks
```

に定められた値)。

棒グラフのみ:

Overlap (Long)

パーセント単位で指定した、データ系列間の棒の重なり合い具合 (100 % で棒同士は完全に重なり合い、-100 % で棒の幅分だけの間隔を確保)。

GapWidth (long)

パーセント単位で指定した、各データグループ間の棒の間隔 (100 % で棒の幅分だけの間隔を確保)。

目盛線

軸の主目盛線と補助目盛を、軸の区切りと補助区切りに合わせて表示できます。Diagram オブジェクトには、目盛線へのアクセス用に、次の属性が用意されています。

HasXAxisGrid (Boolean)

X 軸の主目盛線を有効化する指定。

XMainGrid (Object)

X 軸の主目盛線に関する詳細な情報のオブジェクト (

```
com.sun.star.chart.ChartGrid
```

サービスをサポート)。

HasXAxisHelpGrid (Boolean)

X 軸の補助目盛線を有効化する指定。

XHelpGrid (Object)

X 軸の補助目盛線に関する詳細な情報のオブジェクト (

```
com.sun.star.chart.ChartGrid
```

サービスをサポート)。

y および z についても同様:

HasYAxisGrid (Boolean)

Y 軸の主目盛線を有効化する指定。

YMainGrid (Object)

Y 軸の主目盛線に関する詳細な情報のオブジェクト (

```
com.sun.star.chart.ChartGrid
```

サービスをサポート)。

HasYAxisHelpGrid (Boolean)

Y 軸の補助目盛線を有効化する指定。

YHelpGrid (Object)

Y 軸の補助目盛線に関する詳細な情報のオブジェクト (

```
com.sun.star.chart.ChartGrid
```

サービスをサポート)。

HasZAxisGrid (Boolean)

Z 軸の主目盛線を有効化する指定。

ZMainGrid (Object)

Z 軸の主目盛線に関する詳細な情報のオブジェクト (

```
com.sun.star.chart.ChartGrid
```

サービスをサポート)。

HasZAxisHelpGrid (Boolean)

Z 軸の補助目盛線を有効化する指定。

ZHelpGrid (Object)

Z 軸の補助目盛線に関する詳細な情報のオブジェクト (

```
com.sun.star.chart.ChartGrid
```

サービスをサポート)。

目盛線のオブジェクトは

```
com.sun.star.chart.ChartGrid
```

サービスに基づいています。このサービスは、

```
com.sun.star.drawing.LineStyle
```

サポートサービスの線の属性をサポートしています (「図形描画とプレゼンテーション」参照)。

軸のタイトル

すべての軸と追加のタイトルを表示できます。Diagram オブジェクトには、軸のタイトルへのアクセス用に、次の属性が用意されています。

HasXAxisTitle (Boolean)

X 軸のタイトルを有効化する指定。

XAxisTitle (Object)

X 軸のタイトルに関する詳細な情報のオブジェクト (

```
com.sun.star.chart.ChartTitle
```

サービスをサポート)。

y および z についても同様:

HasYAxisTitle (Boolean)

Y 軸のタイトルを有効化する指定。

YAxisTitle (Object)

Y 軸のタイトルに関する詳細な情報のオブジェクト (

```
com.sun.star.chart.ChartTitle
```

サービスをサポート)。

HasZAxisTitle (Boolean)

Z 軸のタイトルを有効化する指定。

ZAxisTitle (Object)

Z 軸のタイトル (

```
com.sun.star.chart.ChartTitle
```

サービスをサポート)

および第 2 軸に関する詳細な情報のオブジェクト (OpenOffice.org 3.0 以降で使用可能)。

HasSecondaryXAxisTitle (Boolean)

第 2X 軸のタイトルを有効化する指定。

SecondXAxisTitle (Object)

第 2X 軸のタイトルに関する詳細な情報のオブジェクト (

```
com.sun.star.chart.ChartTitle
```

サービスをサポート)。

HasSecondaryYAxisTitle (Boolean)

第 2Y 軸のタイトルを有効化する指定。

SecondYAxisTitle (Object)

第 2Y 軸のタイトルに関する詳細な情報のオブジェクト (

```
com.sun.star.chart.ChartTitle
```

サービスをサポート)。

軸タイトルの書式設定に用いるオブジェクトは、グラフタイトルと同様に、

```
com.sun.star.chart.ChartTitle
```

サービスをベースとしています。

例

次のサンプルコードを実行すると、折れ線グラフが作成されます。ここでグラフの壁面は、表示色を白に設定しています。X 軸および Y 軸に関しては、目盛線を灰色で表示しています。なお Y 軸の最小値を 0、最大値を 100 とするよう明示的に指定してあるため、表示するデータが変更されても、このグラフの表示範囲は固定されたままになります。X 軸は、逆方向で右から左を指しています。X 軸のタイトルが追加されています。

```
Dim Doc As Object
Dim Charts As Object
Dim Chart as Object
Dim Rect As New com.sun.star.awt.Rectangle
Dim RangeAddress(0) As New com.sun.star.table.CellRangeAddress

Doc = StarDesktop.CurrentComponent
Charts = Doc.Sheets(0).Charts

Rect.X = 8000
Rect.Y = 1000
Rect.Width = 10000
Rect.Height = 7000
RangeAddress(0).Sheet = 0
RangeAddress(0).StartColumn = 0
RangeAddress(0).StartRow = 0
RangeAddress(0).EndColumn = 2
RangeAddress(0).EndRow = 12

Charts.addNewByName("MyChart", Rect, RangeAddress(), True, True)
Chart = Charts.getByName("MyChart").embeddedObject
Chart.Diagram = Chart.createInstance("com.sun.star.chart.LineDiagram")
Chart.Diagram.Wall.FillColor = RGB(255, 255, 255)
Chart.Diagram.HasXAxisGrid = True
Chart.Diagram.XMainGrid.LineColor = RGB(192, 192, 192)
Chart.Diagram.HasYAxisGrid = True
Chart.Diagram.YMainGrid.LineColor = RGB(192, 192, 192)
Chart.Diagram.YAxis.Min = 0
Chart.Diagram.YAxis.Max = 100

Chart.Diagram.XAxis.ReverseDirection = true 'needs OpenOffice.org 2.4 or
newer
Chart.Diagram.HasXAxisTitle = true
Chart.Diagram.XAxisTitle.String = "Reversed X Axis Example"
```

3D グラフ

OpenOffice.org のグラフの多くは、3 次元表示が可能です。`Diagram` オブジェクトでは、3D グラフに次の属性を利用できます。

`Dim3D (Boolean)`

3D 表示を有効化する指定。

`Deep (Boolean)`

系列は、z 方向で互いの後側に配置されます。

`RightAngledAxes (Boolean)`

X 軸と Y 軸がプロジェクション内で正しい角度を形成する場合に、3D 表示モードを有効化する指定。(OpenOffice.org 2.3 以降で使用可能)。

D3DScenePerspective (Enum)

3D オブジェクトを遠近法のプロジェクション、または並列のプロジェクションのいずれかで描画するかの定義。(

```
com.sun.star.drawing.ProjectionMode
```

で定められた値)。

Perspective (Long)

3D グラフの透視図 ([0,100]) (OpenOffice.org 2.4.1 以降で使用可能)。

RotationHorizontal (Long)

度数で指定する 3D グラフの水平回転 ([-180,180]) (OpenOffice.org 2.4.1 以降で使用可能)。

RotationVertical (Long)

度数で指定する 3D グラフの垂直回転 ([-180,180]) (OpenOffice.org 2.4.1 以降で使用可能)。

次のサンプルコードを実行すると、3D の面グラフが作成されます。

```
Dim Doc As Object
Dim Charts As Object
Dim Chart as Object
Dim Rect As New com.sun.star.awt.Rectangle
Dim RangeAddress(0) As New com.sun.star.table.CellRangeAddress

Doc = StarDesktop.CurrentComponent
Charts = Doc.Sheets(0).Charts

Rect.X = 8000
Rect.Y = 1000
Rect.Width = 10000
Rect.Height = 7000
RangeAddress(0).Sheet = 0
RangeAddress(0).StartColumn = 0
RangeAddress(0).StartRow = 0
RangeAddress(0).EndColumn = 2
RangeAddress(0).EndRow = 12

Charts.addNewByName("MyChart", Rect, RangeAddress(), True, True)
Chart = Charts.getByName("MyChart").embeddedObject
Chart.Diagram = Chart.createInstance("com.sun.star.chart.AreaDiagram")
Chart.Diagram.Dim3D = true
Chart.Diagram.Deep = true
Chart.Diagram.RightAngledAxes = true 'needs OpenOffice.org 2.3 or newer
Chart.Diagram.D3DScenePerspective =
com.sun.star.drawing.ProjectionMode.PERSPECTIVE
Chart.Diagram.Perspective = 100 'needs OpenOffice.org 2.4.1 or newer
Chart.Diagram.RotationHorizontal = 60 'needs OpenOffice.org 2.4.1 or
newer
Chart.Diagram.RotationVertical = 30 'needs OpenOffice.org 2.4.1 or newer
```

積み上げグラフ

積み上げグラフとは、複数のデータ系列の値を積み上げる形で表示し、その総和を示すためのグラフです。このグラフを利用すると、各データ系列ごとの値だけでなく、これらの総計も同時に確認できます。

OpenOffice.org に用意されているグラフの多くは、積み上げグラフによる表示に対応しています。これに該当するグラフ種は、すべて

```
com.sun.star.chart.StackableDiagram
```

サービスをサポートしており、次の属性を使用できます。

Stacked (Boolean)

積み上げ表示モードを有効化する指定。

Percent (Boolean)

絶対値ではなく、百分率分布を表示する指定。

グラフの種類

折れ線グラフ

折れ線グラフ (

```
com.sun.star.chart.LineDiagram
```

) では、X 軸 2 つ、Y 軸 2 つ、Z 軸 1 つがサポートされます。またこのグラフでは 2D 表示だけでなく、3D 表示 (

```
com.sun.star.chart.Dim3Ddiagram
```

サービス) も使用できます。線は積み上げることができます (

```
com.sun.star.chart.StackableDiagram
```

)。

折れ線グラフには、次の属性を指定できます。

SymbolType (const)

データポイント表示記号 (

```
com.sun.star.chart.ChartSymbolType
```

に一致する定数)。

SymbolSize (Long)

100 分の 1 ミリ単位で指定した、データポイント表示記号のサイズ。

SymbolBitmapURL (String)

データポイントの表示記号とする画像のファイル名。

Lines (Boolean)

線を通じてデータポイントにリンクします。

SplineType (Long)

グラフの線に対して平滑線 (スプライン) 処理を行う指定 (0: 平滑線なし、1: 平滑線つなぎ、2: B 平滑線)。

SplineOrder (Long)

スプラインの多項式加重 (B スプラインのみ)

SplineResolution (Long)

スプライン計算に対する指示点の数

面グラフ

面グラフ (

```
com.sun.star.chart.AreaDiagram
```

サービス) では、X 軸 2 つ、Y 軸 2 つ、Z 軸 1 つがサポートされます。またこのグラフでは 2D 表示だけでなく、3D 表示 (

```
com.sun.star.chart.Dim3Ddiagram
```

サービス) も使用できます。面は積み上げることができます (

```
com.sun.star.chart.StackableDiagram
```

)。

棒グラフ

棒グラフ (

```
com.sun.star.chart.BarDiagram
```

) では、X 軸 2 つ、Y 軸 2 つ、Z 軸 1 つがサポートされます。またこのグラフでは 2D 表示だけでなく、3D 表示 (

```
com.sun.star.chart.Dim3Ddiagram
```

サービス) も使用できます。棒は積み上げることができます (

```
com.sun.star.chart.StackableDiagram
```

)。

棒グラフには、次の属性を指定できます。

Vertical (Boolean)

標準の横棒表示に対して、縦棒表示とする指定。

Deep (Boolean)

3D 表示の際に、棒を交互に並べるのではなく、前後方向に並べる指定。

StackedBarsConnected (Boolean)

積み上げグラフ表示の際に、棒を線でつなぐ指定 (水平表示の場合のみ指定可能)。

NumberOfLines (Long)

積み上げグラフ表示の際に、棒ではなく折れ線表示させる数。

GroupBarsPerAxis (Boolean)

別々の軸の棒を前後または左右に表示する指定 (OpenOffice.org 2.4 以降で使用可能)

円グラフ

円グラフ (

```
com.sun.star.chart.PieDiagram
```

) は、軸を表示することも、積み上げグラフとすることもできません。またこのグラフでは 2D 表示だけでなく、3D 表示 (

```
com.sun.star.chart.Dim3DDiagram
```

サービス) も使用できます。

Diagram オブジェクトからは、円グラフとドーナツグラフに対する次の属性を利用できます。

StartingAngle (Long)

最初の扇形の度単位の角度 (OpenOffice.org 3.0 以降で使用可能)

データベース

OpenOffice.org には、Star Database Connectivity (SDBC) という名称の統合データベースインターフェースが用意されています (すべてのシステムから独立して存在)。こうしたインターフェースが開発された主な目的は、可能な限り広範なデータソースへのアクセスを可能にするためです。

このような目的を達成する一環として、データソースのアクセスにはドライバを使用しています。データソースからのデータ取得をこれらドライバに担当させることで、データソースと SDBC ユーザーとの関係を分離させることができます。こうしたドライバの中には、ファイルベースのデータベースにアクセスして、直接データを引き出すものがあります。その他のドライバは、JDBC や ODBC などの標準インターフェースを利用します。また MAPI アドレスブック、LDAP ディレクトリ、OpenOffice.org 表計算ドキュメントをデータソースとしてアクセスする特殊なドライバもあります。

こうしたドライバは UNO コンポーネントをベースに使用しているため、ドライバを新規開発して新たなデータソースを使用することもできます。必要な詳細情報については、OpenOffice.org の『デベロッパ向けガイド』を参照してください。

SDBC の概念は、VBA の ADO および DAO ライブラリに相当するものです。これを利用することで、データベースのバックエンドに影響されることなく、ハイレベルなデータベースアクセスが行えます。

SQL: クエリー言語

SDBC 用のクエリー言語としては、SQL 言語が利用されています。一口に SQL といっても言語ごとに細かな相違点が存在するため、SDBC コンポーネントには OpenOffice.org 独自の SQL パーサー (構文解析プログラム) が装備されています。これはクエリーウィンドウを用いて入力される SQL コマンドをチェックし、大文字と小文字の入力ミスなどに対する簡単なチェックを行います。

SQL をサポートしていないデータソースに対するアクセスをドライバが許可した場合、転送されてきた SQL コマンドに対して、アクセスに必要な変換が独自に行われます。

データベースアクセスの種類

OpenOffice.org のデータベースインターフェースは、OpenOffice.org Writer と OpenOffice.org Calc の各アプリケーションおよび、データベースフォームから利用できます。

たとえば OpenOffice.org Writer の場合、SDBC データソースを利用してレターを作成し、印刷できます。またドラッグ&ドロップ機能を使用して、データベースウィンドウから文書ドキュメントにデータを移動するという処理も可能です。

データベーステーブルから OpenOffice.org の表計算ドキュメントにデータを移動した場合に作成される表は、オリジナルのデータに変更があった場合に、マウスクリックにより更新されます。また逆に、表計算ドキュメント側のデータをデータベーステーブルに移動することにより、データベースへのインポートを行うことも可能です。

OpenOffice.org では、データベース内部のデータを、フォームの形式で利用することもできます。この処理を行うには、まず OpenOffice.org Writer または OpenOffice.org Calc 上でフォームを作成して、その上にデータベースとリンクしたフィールドを配置します。

これまでに説明した処理は、すべて OpenOffice.org のユーザーインターフェースを利用しています。これらの機能のみを使用するだけであれば、プログラミングに関する特別な知識は不要です。

ただし本節では、このようなユーザーインターフェースに関する説明は最小限にとどめ、SDBC 用のプログラミングインターフェースに焦点を当て、データベースの自動検索処理をはじめとした、各種の操作法を説明します。

また、これらの説明を完全に理解するに当たっては、データベースの機能および SQL のクエリー言語に関する基本的な知識が必要です。

データソース

データベースの OpenOffice.org への組み込みは、いわゆるデータソースと呼ばれるものを作成することにより実施できます。ユーザーインターフェースのデータソース作成用オプションは、メニュー ツールに用意されています。データソースを作成し、OpenOffice.org Basic を使用して操作することもできます。

データソースへのアクセスを行う場合、まず最初に createUnoService 関数によるデータベースコンテキストオブジェクトの作成を行う必要があります。これはデータベース処理のルートオブジェクトとして機能するもので、その操作には

```
com.sun.star.sdb.DatabaseContext
```

サービスを利用します。

次のサンプルコードでは、データベースコンテキストの作成方法、および使用可能なすべてのデータソースの取得方法を示します。ここで取得した名前は、逐次メッセージボックスに表示します。

```
Dim DatabaseContext As Object
Dim Names
Dim I As Integer

DatabaseContext = createUnoService("com.sun.star.sdb.DatabaseContext")

Names = DatabaseContext.getElementNames()

For I = 0 To UBound(Names())
  MsgBox Names(I)
Next I
```

個々のデータソースは

```
com.sun.star.sdb.DataSource
```

サービスをベースとしており、データベースコンテキストに getByName メソッドを適用することで各データソースを個別に指定できます。

```
Dim DatabaseContext As Object
Dim DataSource As Object

DatabaseContext = createUnoService("com.sun.star.sdb.DatabaseContext")
DataSource = DatabaseContext.getByName("Customers")
```

上記のサンプルコードでは、**Customers** というデータソース名を指定して、その DataSource オブジェクトを作成しています。

データソースには各種の属性が用意されており、これらを通じてデータの出自やアクセス方式などの一般的な情報を取得することができます。以下にこれらの属性を 示します。

Name (String)

データソースの名前。

URL (String)

URL (文字列) データソースの URL (フォーマットは **jdbc:** サブプロトコル **:** サブネーム または **sdbc:** サブプロトコル **:** サブネーム)。

Settings (Array)

接続パラメータを持つ `PropertyValue` ペアを含む配列 (通常はユーザー名とパスワードが最低必要)。

User (String)

ユーザー名。

Password (String)

ユーザーパスワード (保存されません)。

IsPasswordRequired (Boolean)

ユーザーに対してパスワードを要求する指定。

IsReadOnly (Boolean)

データベースへの読み取り専用アクセスを許可。

NumberFormatsSupplier (Object)

データベースに使用可能な数値書式を含むオブジェクト (

```
com.sun.star.util.XNumberFormatsSupplier
```

インターフェースをサポート)。

TableFilter (Array)

表示させるテーブル名のリスト。

TableTypeFilter (Array)

表示させるテーブルの種類のリスト。使用可能な値: `TABLE`、`VIEW` および `SYSTEM TABLE`

SuppressVersionColumns (Boolean)

バージョン管理用の列を非表示とする指定。

OpenOffice.org のデータソースと ODBC のデータソースは、一対一に対応するわけではありません。ODBC のデータソースがデータの出自のみを対象としているのに対して、OpenOffice.org のデータソースでは、OpenOffice.org のデータベースウィンドウでのデータ表示といった、より広範な情報も格納しています。

クエリー

データソースに対しては、事前定義されたクエリーを利用できます。OpenOffice.org は、SQL のクエリーコマンドを記録して、随時利用できるようにしています。クエリーとは、データベースの利用を簡単化する目的で開発されたもので、SQL に関する専門的な知識をもたないユーザーでも、マウスによるクリック操作のみで SQL コマンドの実行に必要な各種オプションを指定できます。

クエリーを使用する場合、

```
com.sun.star.sdb.QueryDefinition
```

サービスをサポートしたオブジェクトを直接操作する必要はありません。クエリーへのアクセスは、該当するデータソースに対して `QueryDefinitions` メソッドを適用することにより実行できます。

次のサンプルコードでは、データソースに記録されているクエリー名の一覧を取得して、逐次メッセージボックスに表示します。

```
Dim DatabaseContext As Object
Dim DataSource As Object
Dim QueryDefinitions As Object
Dim QueryDefinition As Object
Dim I As Integer

DatabaseContext = createUnoService("com.sun.star.sdb.DatabaseContext")
DataSource = DatabaseContext.getByName("Customers")
QueryDefinitions = DataSource.getQueryDefinitions()

For I = 0 To QueryDefinitions.Count() - 1
  QueryDefinition = QueryDefinitions(I)
  MsgBox QueryDefinition.Name
Next I
```

```
com.sun.star.sdb.QueryDefinition
```

サービスには、上記のサンプルコードで使用した Name 属性をはじめとする各種の属性が用意されています。以下にその属性を示します。

Name (String)

クエリー名。

Command (String)

SQL コマンド (通常 `SELECT` コマンド)。

次のサンプルコードは、プログラム制御によるクエリーオブジェクトの作成およびデータソースへの登録を行う場合の例です。

```
Dim DatabaseContext As Object
Dim DataSource As Object
Dim QueryDefinitions As Object
Dim QueryDefinition As Object
Dim I As Integer

DatabaseContext = createUnoService("com.sun.star.sdb.DatabaseContext")
DataSource = DatabaseContext.getByName("Customers")
QueryDefinitions = DataSource.getQueryDefinitions()
QueryDefinition = createUnoService("com.sun.star.sdb.QueryDefinition")
QueryDefinition.Command = "SELECT * FROM Customer"
QueryDefinitions.insertByName("NewQuery", QueryDefinition)
```

この場合の処理の流れは、まず最初に createUnoService を用いてクエリーオブジェクトを作成し、次にその初期化を行い、最後に insertByName メソッドにより QueryDefinitions オブジェクトへ追加します。

データベースアクセス

データベースへのアクセスを行うには、該当するデータベース接続を確立しておく必要があります。これは、データベースとの直接的な交信を行うための転送チャネルが確保された状態を意味します。前節で説明したデータソースとは異なり、こうしたデータベース接続は、プログラムを起動するごとに確立し直す必要があります。

OpenOffice.org でのデータベース接続の確立は、各種の方法で実施できます。次のサンプルコードは、既存のデータソースへの接続方法の例です。

```
Dim DatabaseContext As Object
Dim DataSource As Object
Dim Connection As Object
Dim InteractionHandler as Object

DatabaseContext = createUnoService("com.sun.star.sdb.DatabaseContext")
DataSource = DatabaseContext.getByName("Customers")

If Not DataSource.IsPasswordRequired Then
  Connection = DataSource.GetConnection("","")
Else
  InteractionHandler =
createUnoService("com.sun.star.sdb.InteractionHandler")
  Connection = DataSource.ConnectWithCompletion(InteractionHandler)
End If
```

上記のサンプルコードでは、まず最初にデータベースがパスワード保護されているかをチェックしています。パスワード保護されていない場合は、`GetConnection` を使用して必要なデータベース接続を確立します。コマンド行内の 2 つの空白文字列は、ここでのユーザー名とパスワードに該当します。

データベースがパスワード保護されている場合、このサンプルコードでは、`InteractionHandler` を作成し、`ConnectWithCompletion` メソッドを使用してデータベース接続を確立しています。この InteractionHandler は、OpenOffice.org 側からユーザーに対して、ログイン情報の入力を求める際に使用します。

テーブルからのデータの取得

通常 OpenOffice.org は、`ResultSet` オブジェクトを通じて、テーブルへアクセスします。この `ResultSet` とは一種のマーカーで、`SELECT` コマンドにより得られた多量のデータに対して、現在のデータセットを示すために使用します。

次のサンプルコードは、`ResultSet` を用いてデータベーステーブルに対するクエリー処理を行う方法を示します。

```
Dim DatabaseContext As Object
Dim DataSource As Object
Dim Connection As Object
Dim InteractionHandler as Object
Dim Statement As Object
Dim ResultSet As Object

DatabaseContext = createUnoService("com.sun.star.sdb.DatabaseContext")
DataSource = DatabaseContext.getByName("Customers")

If Not DataSource.IsPasswordRequired Then
  Connection = DataSource.GetConnection("","")
Else
  InteractionHandler =
createUnoService("com.sun.star.sdb.InteractionHandler")
  Connection = DataSource.ConnectWithCompletion(InteractionHandler)
End If

Statement = Connection.createStatement()
ResultSet = Statement.executeQuery("SELECT ""CustomerNumber"" FROM
""Customer""")

If Not IsNull(ResultSet) Then
  While ResultSet.next
    MsgBox ResultSet.getString(1)
  Wend
End If
```

このサンプルコードでは、データベース接続の確立後、まず最初に `Connection.createObject` を使用して `Statement` オブジェクトを作成しています。次にこの `Statement` オブジェクトに対して `executeQuery` を実施して、その結果を `ResultSet` に格納しています。こうして得られた `ResultSet` については、その内容が空でないかをチェックした後ループに入り、個々のレコード情報を取り出します。レコード情報 (上記のサンプルコードでは `CustomerNumber` フィールドの値) の取得は、`ResultSet` に対して `getString` メソッドを適用することで行っていますが、その際のパラメータ値の 1 は、第 1 列の値を取り出すことを指定しています。

SDBC を利用した場合の `ResultSet` オブジェクトは、データベースへのインタラクティブなアクセスを行うものであり、DAO ないし ADO を利用した場合の `Recordset` オブジェクトに相当します。

OpenOffice.org でのデータベースアクセスは、`ResultSet` オブジェクトを通じて行います。この中には、テーブル内の登録データまたは、SQL の SELECT コマンドの実行結果が格納されます。従来の `ResultSet` オブジェクトは、`Application` オブジェクトのデータナビゲーション用メソッド (`DataNextRecord` など) を提供するものでした。

種類別データの取得

前節のサンプルコードでも触れたように、OpenOffice.org にはテーブルからのデータ取得用に `getString` メソッドが用意されています。このメソッドの実行結果は、文字列の形で返されます。次の `get` メソッドが利用できます。

`getByte()`

サポートする SQL のデータ型は、数値、文字、文字列。

`getShort()`

サポートする SQL のデータ型は、数値、文字、文字列。

`getInt()`

サポートする SQL のデータ型は、数値、文字、文字列。

`getLong()`

サポートする SQL のデータ型は、数値、文字、文字列。

`getFloat()`

サポートする SQL のデータ型は、数値、文字、文字列。

`getDouble()`

サポートする SQL のデータ型は、数値、文字、文字列。

`getBoolean()`

サポートする SQL のデータ型は、数値、文字、文字列。

`getString()`

サポートする SQL のデータ型は、すべてのデータ型。

`getBytes()`

サポートする SQL のデータ型は、バイナリデータ。

`getDate()`

サポートする SQL のデータ型は、数値、文字列、日付と時刻のタイムスタンプ。

`getTime()`

サポートする SQL のデータ型は、数値、文字列、日付と時刻のタイムスタンプ。

`getTimestamp()`

サポートする SQL のデータ型は、数値、文字列、日付と時刻のタイムスタンプ。

`getCharacterStream()`

サポートする SQL のデータ型は、数値、文字列、バイナリデータ。

`getUnicodeStream()`

サポートする SQL のデータ型は、数値、文字列、バイナリデータ。

`getBinaryStream()`

バイナリデータ。

`getObject()`

サポートする SQL のデータ型は、すべてのデータ型。

いずれのメソッドを用いる場合でも、データを取得する列の番号をパラメータとして指定します。

ResultSet のバリエーション

データベースへのアクセス処理には、かなりの時間を要する場合があります。このため OpenOffice.org には ResultSets の処理を最適化する各種の方法が用意されており、アクセス速度を制御できるようになっています。ResultSet の機能が豊富になるほど、その実装は複雑化する傾向にあり、その分だけ処理速度が低下します。

簡単な ResultSet で、最低限の関数が利用できます。この場合、テーブル内でのデータ検索は順方向にのみ進行し、必要なデータも順次取得するだけです。そのため、データ更新などのより複雑なナビゲーションが必要となっても、このままでは対処できません。

ResultSet の作成に用いた Statement オブジェクトには、ResultSet の機能に関係するいくつかのオプションが用意されています。

ResultSetConcurrency (const)

データが変更可能かどうかの指定 (

```
com.sun.star.sdbc.ResultSetConcurrency
```

に一致する指定)。

ResultSetType (const)

ResultSets のタイプに関する指定 (

```
com.sun.star.sdbc.ResultSetType
```

に一致する指定)。

```
com.sun.star.sdbc.ResultSetConcurrency
```

では、次の値が定義されます。

UPDATABLE

ResultSet でのデータ更新を許可する。

READ_ONLY

ResultSet でのデータ更新を許可しない。

```
com.sun.star.sdbc.ResultSetConcurrency
```

に定められた定数では、次の指定を行えます。

FORWARD_ONLY

ResultSet に対し順方向のナビゲーションのみを許可する。

SCROLL_INSENSITIVE

ResultSet にすべてのナビゲーションを許可するが、オリジナルデータへの変更は記録しない。

SCROLL_SENSITIVE

ResultSet にすべてのナビゲーションを許可し、オリジナルデータへの変更による ResultSet への影響を認める。

READ_ONLYと SCROLL_INSENSITIVE 属性が指定された ResultSet は、ADO および DAO の Snapshot タイプに相当します。

ResultSet's UPDATEABLE と SCROLL_SENSITIVE 属性が指定された ResultSet は、ADO および DAO の Dynaset タイプ Recordset に相当します。

ResultSets のナビゲーション用メソッド

SCROLL_INSENSITIVE または SCROLL_SENSITIVE が指定された ResultSet は、すべてのデータナビゲーション用メソッドを利用できます。以下に主要なメソッドを示します。

next()

次のデータレコードへ移動。

previous()

前のデータレコードへ移動。

first()

最初のデータレコードへ移動。

last()

最後のデータレコードへ移動。

beforeFirst()

最初のデータレコードの前へ移動。

afterLast()

最後のデータレコードの次へ移動。

いずれのメソッドを実行した場合も、その戻り値として、ナビゲーションが正常に終了したかを示すブール値が返されます。

現在のカーソル位置を確認するには、次のテスト用メソッドを使用して、戻り値として返されるブール値から判定します。

isBeforeFirst()

ResultSet が最初のデータレコードの前にあるかを判定。

isAfterLast()

ResultSet が最後のデータレコードの次にあるかを判定。

isFirst()

ResultSet が最初のデータレコードにあるかを判定。

isLast()

ResultSet が最後のデータレコードにあるかを判定。

データレコードの変更

ResultSet の作成時に ResultSetConcurrency = UPDATEABLE と指定した場合、データレコードを変更することができます。このような状況は、基本的に SQL コマンドがデータベースへのデータの書き換えを許可している場合のみ有効です。ただし、列のリンクや累積計算を行う複雑な SQL コマンドなどに対しては、この状況は当てはまりません。

ResultSet オブジェクトには、データ変更用に Update メソッドが用意されていますが、その構成はデータ取得用の get メソッドと基本的に同じです。たとえば文字列の書き換えを許可するには updateString メソッドを使用します。

必要な変更を行なったデータは、updateRow() メソッドを用いてデータベースに再転送する必要があります。このメソッド呼び出しは、次のナビゲーション用コマンドの実行前に実行しておく必要があり、実行しないと、変更した値は失われます。

データ変更時に何らかのエラーが発生した場合、cancelRowUpdates() メソッドを使用することで、これらの処理を取り消すことができます。ただしこのような処理の取り消しが行えるのは、updateRow() によってデータベース上のデータを書き直す前の段階だけです。

ダイアログ

OpenOffice.org のドキュメントでは、ユーザー定義によるカスタムダイアログやフォームを表示することができます。これらは OpenOffice.org Basic マクロとのリンクが可能という点で、OpenOffice.org Basic の利用範囲を大いに広げています。ダイアログの用途としては、データベースの登録情報の表示をはじめ、オートパイロットによる新規ドキュメント作成時の手順表示などにも利用できます。

ダイアログの操作

OpenOffice.org Basic のダイアログは、1 個のダイアログウィンドウ上にテキストフィールド、リストボックス、ラジオボタンなどの各種コントロールが配置されます。

ダイアログの作成法

OpenOffice.org ダイアログエディタを使用して、ダイアログを作成および構成できます。

コントロール要素をデザインパレット (右) からダイアログ領域にドラッグして、位置とサイズを定義できます。

ここでのサンプルダイアログでは、ラベルとリストボックスを配置しています。

ダイアログを表示するには、次のサンプルコードのような手順で処理します。

```
Dim Dlg As Object

DialogLibraries.LoadLibrary("Standard")
Dlg = CreateUnoDialog(DialogLibraries.Standard.DlgDef)
Dlg.Execute()
Dlg.dispose()
```

上記のサンプルコードでは、`CreateUnoDialog` により `Dlg` という名前のオブジェクトを作成していますが、該当するダイアログに対してはこのオブジェクトを介して参照することになります。また実際にダイアログを作成するには、必要なライブラリ (上記のサンプルコードでは `Standard` ライブラリ) を事前に読み込む必要があります。`LoadLibrary` メソッドがこのタスクを実行します。

ダイアログオブジェクト (上記のサンプルコードでは `Dlg`) に対して必要な初期化を行った後、`Execute` メソッドを実行することで、該当するダイアログが表示されます。ここでサンプルとしたタイプのダイアログは、表示中に他のプログラムによる処理が何もできなくなるため、モーダルダイアログと呼ばれます。ダイアログが表示されている間のプログラムは、`Execute` メソッドを実行し続けている状態になります。

プログラム最終行の `dispose` メソッドは、ダイアログの使用するリソースをプログラム終了時に解放させるためのものです。

ダイアログのクローズ処理

OK またはキャンセルのボタンによるクローズ

ダイアログに **OK** またはキャンセルのボタンが表示されている場合、いずれかのボタンがクリックされた段階で、ダイアログは自動的に閉じられます。これらのボタンの操作の詳細は、「ダイアログコントロールの詳細」で説明されています。

OK ボタンをクリックしてダイアログを閉じた場合は Execute- メソッドの戻り値として 1 が返され、それ以外の場合は 0 が返されます。

```
Dim Dlg As Object

DialogLibraries.LoadLibrary("Standard")
Dlg = CreateUnoDialog(DialogLibraries.Standard.MyDialog)
Select Case Dlg.Execute()
Case 1
  MsgBox "Ok pressed"
Case 0
  MsgBox "Cancel pressed"
End Select
```

タイトルバーの閉じるボタンによるクローズ処理

タイトルバーの閉じるボタンをクリックして、ダイアログウィンドウをクローズできます。ダイアログの Execute メソッドは、0 の値を返します。これは、「キャンセル」をクリックした場合と同じです。

プログラムによる明示的なクローズ処理

プログラム内に次のような endExecute メソッドを記述することでも、ダイアログウィンドウをクローズできます。

```
Dlg.endExecute()
```

コントロール要素へのアクセス

各ダイアログには、必要な数のコントロール要素を配置することができます。コントロール要素へアクセスするには、getControl メソッドを実行して、戻り値として返される該当コントロール要素の名前を利用します。

```
Dim Ctl As Object

Ctl = Dlg.getControl("MyButton")
Ctl.Label = "New Label"
```

上記のサンプルコードでは、MyButton というコントロール要素に対する参照用オブジェクトを用意して、この Ctl というオブジェクト変数を介することで、コントロール要素の初期化を行っています。そしてここでは最後に、コントロール要素の Label 属性に New Label という値を設定しています。

OpenOffice.org Basic の識別子と異なり、コントロール要素の名前では、大文字と小文字が区別されます。

コントロール要素およびダイアログでの モデル の使用法

OpenOffice.org API を利用する際には、実際に表示されるプログラム要素 (**View**: ビュー) と、その背後に存在するデータやドキュメント (**Model**: モデル) とを使い分けなければならない場合があります。ダイアログとコントロール要素のオブジェクトの下層には、コントロール要素のメソッドや属性以外に、Model というオブジェクトが存在します。このオブジェクトを利用することで、ダイアログやコントロール要素の内容に直接アクセスすることができます。

OpenOffice.org に用意された API の中でも、ダイアログに関しては、データとその表示内容の境界が非常にあいまいです。API の要素には、View と Model のどちらからでもアクセスできます。

Model 属性は、ダイアログおよびコントロール要素オブジェクトのモデルに対する、プログラム制御によるアクセスに利用します。

```
Dim cmdNext As Object

cmdNext = Dlg.getControl("cmdNext")
cmdNext.Model.Enabled = False
```

上記のサンプルコードでは、Dlg というダイアログにある cmdNtext というボタンを、cmdNtext からのモデルオブジェクトを用いて非アクティブにしています。

プロパティー

名前とタイトル

各コントロール要素には名前が付けられており、これらの名前に対しては、次のモデルプロパティーを用いて参照することができます。

Model.Name (String)

コントロール要素名

ダイアログのタイトルバーに表示されるタイトルに対しては、次のモデルプロパティーを用いて参照することができます。

Model.Title (String)

ダイアログタイトル (ダイアログに対してのみ適用)

位置とサイズ

コントロール要素の表示位置とサイズに対しては、次のモデルプロパティーを用いて参照することができます。

Model.Height (long)

コントロール要素の高さ (ma 単位)

Model.Width (long)

コントロール要素の幅 (ma 単位)

Model.PositionX (long)

ダイアログ内側の左端から測った、コントロール要素の X 座標 (ma 単位)

Model.PositionY (long)

ダイアログ内側の上端から測った、コントロール要素の Y 座標 (ma 単位)

OpenOffice.org では、プラットフォームへの非依存性を確保する観点から、ダイアログ内での位置とサイズを示す際に、**Map AppFont (ma)** という内部単位を使用しています。ma 単位では、オペレーティングシステムに設定されたシステムフォントの平均サイズを基準に、その高さの 8 分の 1 および幅の 4 分の 1 を、各方向の 1 単位と定めています。OpenOffice.org はこのような ma 単位を利用することで、システム設定の異なる環境下においても、ダイアログの表示が同じになるようにしています。

実行時に表示されるコントロール要素の位置とサイズを変更する場合は、ダイアログのサイズを確認してから、コントロール要素の表示指定を調整します。

Map AppFont (ma) 単位は、プラットフォームへの非依存性を確保する観点から、従来の Twip (トゥウィップ) 単位に替わって導入されたものです。

フォーカスおよびタブ順

ダイアログ上に配置されたコントロール要素に対しては、Tab キーによるフォーカス移動が行えます。こうした操作に関係するコントロール要素モデルでは、次の属性が利用できます。

Model.Enabled (Boolean)

コントロール要素をアクティブにする指定

Model.Tabstop (Boolean)

コントロール要素を Tab キーによるフォーカス移動の対象にする指定

Model.TabIndex (Long)

アクティブ化の順序におけるコントロール要素の位置

最後に紹介する `getFocus` メソッドは、ダイアログ上のコントロール要素にフォーカス移動を行うよう設定するものです。

getFocus

コントロール要素がフォーカスを受け取ります (ダイアログの場合のみ)

マルチページダイアログ

OpenOffice.org のダイアログには、複数のページを配置できます。ダイアログにある `Step` プロパティーはダイアログの現在のタブページを指定するものですが、コントロール要素にも同様の `Step` プロパティーが存在し、こちらは該当コントロールを表示するタブページを指定します。

`Step` に指定する値のうち、0 は特別な意味を持ちます。ダイアログ側で `Step` プロパティー値に 0 を指定すると、コントロール要素側の指定値とは無関係に、すべてのコントロール要素が表示されるようになります。これとは逆に、Step 属性値を 0 に設定したコントロール要素は、ダイアログのすべてのタブページ上に表示されるようになります。

ダイアログのページ 1 のデザイン

前の例では、分割線だけでなく「`Cancel`」、「`Prev`」、「`Next`」、「`Done`」の各ボタンについても `Step` の値を 0 に設定し、これらの要素をすべてのページに表示できます。ただし必要であれば、各コントロール要素を特定のタブページだけに表示させることもできます (たとえばページ 1)。

次のサンプルコードでは、イベントハンドラに応じて、「`Next`」および「`Prev`」ボタンの `Step` 値を増減させて、ボタンの表示用ステータスを変更しています。

```
Sub cmdNext_Initiated

   Dim cmdNext As Object
   Dim cmdPrev As Object

   cmdPrev = Dlg.getControl("cmdPrev")
   cmdNext = Dlg.getControl("cmdNext")
   cmdPrev.Model.Enabled = Not cmdPrev.Model.Enabled
   cmdNext.Model.Enabled = False
   Dlg.Model.Step = Dlg.Model.Step + 1

End Sub

Sub cmdPrev_Initiated

   Dim cmdNext As Object
   Dim cmdPrev As Object

   cmdPrev = Dlg.getControl("cmdPrev")
   cmdNext = Dlg.getControl("cmdNext")
   cmdPrev.Model.Enabled = False
   cmdNext.Model.Enabled = True
   Dlg.Model.Step = Dlg.Model.Step - 1

End Sub
```

このサンプルコードを使用する場合は、ダイアログの参照に用いる Dlg という変数を、広域変数として用意しておく必要があります。そしてこのサンプルコードの実行により、ダイアログ上のボタンは、次の図のように表示ステータスが切り換えられます。

ページ 1

ページ 2

イベント

OpenOffice.org のダイアログとフォームは、イベント指向型のプログラミングモデルを基に構築されているため、各コントロール要素に対してはイベントハンドラを指定できます。イベントハンドラは、特定のアクションが発生すると、定義済みの手順を実行します。イベントハンドラを利用して、ドキュメントの編集やデータベースのオープンをはじめ、他のコントロール要素へのアクセスもできます。

OpenOffice.org のコントロール要素は、様々な状況で発生する各種のイベントに対応しています。これらのイベントは、4 つのグループに分かれます。

・マウス制御: マウスの動作に関係するイベント (たとえば単なるマウスポインタの移動や、画面上の特定部分のクリックなど)

・キーボード制御: キーボードのキーストロークで発生するイベント

・フォーカス変更: コントロール要素が有効化または無効化されている場合に、OpenOffice.org が実行するイベント。

・コントロール要素固有のイベント: 特定のコントロール要素のみに発生するイベント。

イベントを利用した処理を行う場合は、OpenOffice.org の開発環境で関連するダイアログを構築して、必要なコントロール要素やドキュメント (イベントをフォームに適用する場合) を用意しておく必要があります。

上の図は OpenOffice.org Basic の開発環境を示したもので、このダイアログウィンドウには 2 つのリストボックスが配置されています。2 つのリストボックスの間に配置されたボタンを使って、リスト内の各項目を相互に移動できます。

このレイアウトを画面に表示させる場合は、イベントハンドラからの呼び出し用の手続きを OpenOffice.org Basic のコード内に作成するようにしてください。このような手続きは、任意のモジュール内に記述できますが、使用するモジュールは 2 つまでにすることが推奨されます。またプログラムコードの可読性を高める観点からも、これらの手続き名には、その機能を簡潔に示す名前をつけておくべきです。マクロから汎用プログラムの手続きに直接ジャンプさせると、プログラムコードが不明瞭になる可能性があります。バグ修正も含めたプログラムコード全体の管理性を高めておくには、たとえそれがターゲットとなる手続きを呼び出すだけの処理であっても、イベントハンドル用のエントリポイントとして利用する手続きを 1 つ別途作成するようにしてください。

次のサンプルコードは、ダイアログ上に配置された左側のリストボックス内の項目の 1 つを、右側のリストボックスへ移動します。

```
Sub cmdSelect_Initiated

   Dim objList As Object

   lstEntries = Dlg.getControl("lstEntries")
   lstSelection = Dlg.getControl("lstSelection")

   If lstEntries.SelectedItem > 0 Then
     lstSelection.AddItem(lstEntries.SelectedItem, 0)
     lstEntries.removeItems(lstEntries.SelectItemPos, 1)
   Else
     Beep
   End If

End Sub
```

この手続きを OpenOffice.org Basic で記述したら、ダイアログエディタの属性ウィンドウを表示して、該当するイベントへの割り当てを行います。

このダイアログには、OpenOffice.org Basic に記述したすべての手続きが表示されます。該当するイベントに手続きを割り当てるには、手続きを選択して、割り当てボタンをクリックします。

パラメータ

特定のイベントの発生だけでは、どのような処理を行うかの判定ができない場合もあります。そのようなケースでは、何らかの追加情報が必要となります。たとえばマウスクリックに関する処理では、どの位置でマウスボタンが押されたかの情報が必要になることもあります。

OpenOffice.org Basic では、次に示すように、オブジェクトパラメータを使用して、イベントについての詳細情報を手続きに渡すことができます。

```
Sub ProcessEvent(Event As Object)

End Sub
```

Event オブジェクトの構造と属性は、手続き呼び出しを発生させるイベントタイプにより異なります。

いずれのイベントタイプにせよ、すべてのオブジェクトは、関連するコントロール要素およびモデルへのアクセスを提供します。コントロール要素には、Event.Source および Event.Source.Model を使用したそのモデルでアクセスできます。

これらの属性は、イベントハンドラ内で特定のイベントをトリガーする際に使用します。

マウスイベント

OpenOffice.org Basic では次のマウスイベントを利用できます。

Mouse moved

ユーザーによるマウスポインタの移動。

Mouse moved while key pressed

ユーザーによるキーを押しながらのマウスによるドラッグ。

Mouse button pressed

ユーザーによるマウスボタンの押し下げ。

Mouse button released

ユーザーによるマウスボタンの解放。

Mouse outside

ユーザーによるマウスポインタの現在のウィンドウ外への移動。

これらのイベントを扱うためのイベントオブジェクトは

```
com.sun.star.awt.MouseEvent
```

構造体として定義されており、次の情報を取り扱うことができます。

Buttons (short)

押されたボタン (

```
com.sun.star.awt.MouseButton
```

に定められた 1 つ以上の定数)。

X (long)

ピクセル単位で指定した、コントロール要素の左上隅を原点とするマウスポインタの X 座標。

Y (long)

ピクセル単位で指定した、コントロール要素の左上隅を原点とするマウスポインタの Y 座標。

ClickCount (long)

マウスイベントに関係したクリック数 (ダブルクリックは、OpenOffice.org が十分高速に反応できる場合、1 つのイベントを開始するだけなので、`ClickCount` によるカウントも 1 となる)。

```
com.sun.star.awt.MouseButton
```

に定義されているマウスボタン関連の定数は、次のものです。

LEFT

マウスの左ボタン。

RIGHT

マウスの右ボタン。

MIDDLE

中央マウスボタン。

次のサンプルコードでは、マウスボタンのクリック位置と押されたボタンを表示します。

```
Sub MouseUp(Event As Object)

  Dim Msg As String

  Msg = "Keys: "
  If Event.Buttons AND com.sun.star.awt.MouseButton.LEFT Then
    Msg = Msg & "LEFT "
  End If

  If Event.Buttons AND com.sun.star.awt.MouseButton.RIGHT Then
    Msg = Msg & "RIGHT "
  End If

  If Event.Buttons AND com.sun.star.awt.MouseButton.MIDDLE Then
    Msg = Msg & "MIDDLE "
  End If

  Msg = Msg & Chr(13) & "Position: "
  Msg = Msg & Event.X & "/" & Event.Y
  MsgBox Msg

End Sub
```

VBA に用意されている `Click` および `Doubleclick` イベントは OpenOffice.org Basic では利用できません。OpenOffice.org Basic では `MouseUp` イベントの代わりに `click` イベントを使用し、`Doubleclick` イベントについてはアプリケーションロジックの変更で対処します。

キーボードイベント

OpenOffice.org Basic では次のキーボードイベントを利用できます。

Key pressed

ユーザーによるキーの押し下げ。

Key released

ユーザーによるキーの解放。

どちらのイベントも、論理的 なキーアクションに対するもので、物理的 なキーアクションに直接対応するものではありません。つまり、1 つの文字の入力に複数キーのコンビネーションが必要な場合 (たとえば欧文のアクセント記号など)、これに対して OpenOffice.org Basic が発生させるイベントは 1 つだけです。

また Shift キー や Alt キーなどの修飾キーを単独で押しても、それだけでは独立したイベントは発生しません。

OpenOffice.org Basic は、押されたキーに関する情報を、イベントオブジェクトによりイベントハンドル用手続きに渡します。この場合は、次の属性を利用できます。

KeyCode (short)

押されたキーのコード (

```
com.sun.star.awt.Key
```

に定められたデフォルト値)。

KeyChar (String)

入力された文字 (修飾キーによる効果も含めた結果)。

次のサンプルコードでは、KeyCode 属性を利用して、Enter キーや Tab キーなどの制御用キーのうち何が押されたかを判定します。これらの制御用キーが押されていた場合は、該当するキーの名前を表示し、それ以外の場合は、入力された文字を表示します。

```
Sub KeyPressed(Event As Object)

   Dim Msg As String

   Select Case Event.KeyCode
   Case com.sun.star.awt.Key.RETURN
     Msg = "Return pressed"
   Case com.sun.star.awt.Key.TAB
     Msg = "Tab pressed"
   Case com.sun.star.awt.Key.DELETE
     Msg = "Delete pressed"
   Case com.sun.star.awt.Key.ESCAPE
     Msg = "Escape pressed"
   Case com.sun.star.awt.Key.DOWN
     Msg = "Down pressed"
   Case com.sun.star.awt.Key.UP
     Msg = "Up pressed"
   Case com.sun.star.awt.Key.LEFT
     Msg = "Left pressed"
   Case com.sun.star.awt.Key.RIGHT
     Msg = "Right pressed"
   Case Else
     Msg = "Character " & Event.KeyChar & " entered"
   End Select
   MsgBox Msg

End Sub
```

キーボード処理に使用するその他の定数については、『API Reference』の

```
com.sun.star.awt.Key
```

グループの定数値を参照してください。

フォーカスイベント

フォーカスイベントは、コントロール要素へのフォーカス移動を判定するためのものです。たとえばユーザーが特定のコントロール要素での処理を終えたかを判定してから、ダイアログ上にある他のコントロール要素を更新するような場合、このイベントが利用できます。以下に、利用できるフォーカスイベントを示します。

When receiving focus

要素がフォーカスを受け取ったとき。

When losing focus

要素がフォーカスを失ったとき。

フォーカスイベント関係の Event オブジェクトは、次のように構成されています。

FocusFlags (short)

フォーカスが変わります (

```
com.sun.star.awt.FocusChangeReason
```

に定められたデフォルト値)。

NextFocus (Object)

フォーカスを受け取るオブジェクト (When losing focus イベントのみ)。

Temporary (Boolean)

一時的に失われたフォーカス。

コントロール要素の固有イベント

これまで説明したイベントは、すべてのコントロール要素でサポートされていますが、その他にも特定のコントロール要素についてのみ定義された固有なイベントが存在します。これらのうち特に重要なものは次のイベントです。

When Item Changed

コントロール要素の値の変更。

Item Status Changed

コントロール要素の変更のステータス。

Text modified

コントロール要素の変更のテキスト。

When initiating

コントロール要素を動作させるアクションの実行 (ボタンの押し下げなど)。

イベントを処理する場合、作動時のイベント When initiating などは、単にコントロール要素をクリックするだけで発生することがあるため、その扱いには注意が必要です (たとえばラジオボタンのクリック)。このような場合、コントロール要素のステータスが実際に変更されたかについてはチェックされません。このような「ブラインドイベント」による混乱を避けるには、変更前のコントロール要素の値を広域変数に保存しておき、イベント実行時にそうした値に変化があったかを確認するという手法が使えます。

Item Status Changed イベントには、次の属性を使用できます。

Selected (long)

現在選択されているエントリ。

Highlighted (long)

強調表示されているエントリ。

ItemId (long)

エントリの ID。

ダイアログコントロールの詳細

OpenOffice.org Basic には各種のコントロール要素が用意されていますが、これらは次の 4 つのグループに分類できます。

入力フィールド	ボタン	選択リスト	その他
・テキストフィールド (テキストボックス)	・標準ボタン (コマンドボタン)	・リストボックス	・スクロールバー (水平および垂直スクロールバー)
・日付フィールド	・チェックボックス	・コンボボックス	・グループ枠
・時刻フィールド	・ラジオボタン		・進行グラフ
・番号フィールド			・分割線 (横線および縦線)
・通貨フィールド			・図オブジェクト
・その他の書式設定可能なフィールド			・ファイルの選択

ボタン

ボタンは、ユーザーによるクリックに応じて、特定のアクションを実行させる際に使用します。

最も単純な使用法は、ユーザーのクリックで発生する「When Initiating」イベントをトリガーとして、ボタンのアクションを実行させるという使い方です。またボタンに他のアクションを割り当てて PushButtonType 属性を利用し、別のダイアログを開くという処理も可能です。この属性値を 0 としたボタンをクリックしても、ダイアログはそのまま残されます。この属性値を 1 としたボタンをクリックした場合、ダイアログは閉じられ、ダイアログを表示していた Execute メソッドは戻り値として 1 を返します (ダイアログの処理は正常終了)。PushButtonType の属性値を 2 としたボタンをクリックした場合、ダイアログは閉じられ、ダイアログを表示していた Execute メソッドは戻り値として 0 を返します。

以下に、ボタンモデルで利用可能なすべての属性を示します。

Model.BackgroundColor (long)

背景の色。

Model.DefaultButton (Boolean)

フォーカスのない状態で Enter キーが押された場合に反応させる標準ボタンとする指定

Model.FontDescriptor (struct)

表示フォントの詳細指定用の構造体 (

```
com.sun.star.awt.FontDescriptor
```

に定められた構造体)

Model.Label (String)

ボタンに表示するラベル (タイトル)

Model.Printable (Boolean)

コントロール要素を印刷する指定。

Model.TextColor (Long)

コントロール要素のテキストの色。

Model.HelpText (String)

コントロール要素にマウスポインタを重ねた際に表示させるヘルプテキスト

Model.HelpURL (String)

コントロール要素で使用するオンラインヘルプの URL。

PushButtonType (short)

ボタンアクションの指定 (0: なし、1: OK、2: キャンセル)

ラジオボタン

通常これらは複数のボタンをグループ化して、そのうち 1 つのオプションを選択することにより使用します。その際にオプションの 1 つが選択されると、残りのオプションは非選択状態になります。このように処理することで、選択状態にあるオプションは常に 1 つだけになります。

ラジオボタンのコントロール要素には、次の 2 つの属性があります。

State (Boolean)

ボタンの有効化。

Label (String)

ボタンに表示するラベル (タイトル)

ラジオボタンのモデルからは、次の属性も使用できます。

Model.FontDescriptor (struct)

表示フォントの詳細指定用の構造体 (

```
com.sun.star.awt.FontDescriptor
```

に定められた構造体)。

Model.Label (String)

コントロール要素に表示するラベル (タイトル)

Model.Printable (Boolean)

コントロール要素が印刷できると指定。

Model.State (Short)

オプションをアクティブとするか (属性値を 1 とした場合)、非アクティブとするかの指定 (その他の値の場合)。

Model.TextColor (Long)

コントロール要素のテキストの色。

Model.HelpText (String)

コントロール要素にマウスポインタを重ねた際に表示するヘルプテキスト。

Model.HelpURL (String)

コントロール要素で使用するオンラインヘルプの URL。

複数のラジオボタンをグループ化する場合は、これらをアクティブ化する順序の設定値 (`Model.TabIndex` 属性の指定値で、はダイアログエディタ上では順序の指定に該当) に連続した値を指定しておく必要があります。アクティブ化する順序の途中に他のコントロール要素が入っていると、OpenOffice.org は新規のコントロール要素グループが始まるものと判断するため、本来のグループ内部での切り替えが想定通りに行えなくなります。

VBA とは異なり OpenOffice.org Basic では、グループ枠の中にラジオボタンを挿入するということはできません。OpenOffice.org Basic に用意されているグループ枠というコントロール要素は、グループ化するコントロール要素を目で見て区別できるように、これらを囲む枠線を引くだけのものです。

チェックボックス

チェックボックスは基本的に Yes または No の形式の情報を入力するために使用しますが、モード設定によっては、このような 2 つのステータス間だけでなく、3 つのステータス間で選択することもできます。通常使用するのは Yes か No かの選択肢ですが、どちらともつかない中間状態が選択肢としてあり得る場合は、それを示すステータスも表示できます。

チェックボックスには、次の属性を指定できます。

State (Short)

チェックボックスの状態 (0: No、1: Yes、2: 中間状態)

Label (String)

コントロール要素のラベル。

enableTriState (Boolean)

選択状態と非選択状態の他に、中間状態を表示する指定

チェックボックスのモデルオブジェクトでは、次の属性を使用できます。

Model.FontDescriptor (struct)

表示フォントの詳細指定用の構造体 (

```
com.sun.star.awt.FontDescriptor
```

に定められた構造体)。

Model.Label (String)

コントロール要素のラベル。

Model.Printable (Boolean)

コントロール要素を印刷する指定。

Model.State (Short)

チェックボックスの状態 (0: No、1: Yes、2: 中間状態)

Model.Tabstop (Boolean)

コントロール要素を Tab キーによるフォーカス移動の対象にする指定。

Model.TextColor (Long)

コントロール要素のテキストの色。

Model.HelpText (String)

コントロール要素にマウスポインタを重ねた際に表示させるヘルプテキスト。

Model.HelpURL (String)

コントロール要素で使用するオンラインヘルプの URL。

テキストボックス（テキストフィールド）

テキストボックスは、ユーザーによる数値およびテキストの入力に使用できます。

```
com.sun.star.awt.UnoControlEdit.
```

サービスにより、テキストボックスの機能が主として提供されています。

テキストボックスの表示は 1 行に制限することもできれば、複数行表示を許可することも可能で、またユーザーからの入力内容を編集することも、読み取り専用とすることもできます。またテキストボックスは、通常の通貨フィールドや番号フィールドまたはパターンフィールドでは処理しきれない場合の代用フィールドとしても利用できます。そもそも、これらのコントロール要素はどれも Uno サービスの `UnoControlEdit` をベースとしているので、基本的に共通した手法でプログラム制御をすることができます。

テキストボックスには、次の属性が用意されています。

Text (String)

現在のテキスト

SelectedText (String)

現在強調表示されているテキスト。

Selection (Struct)

読み取りの強調表示設定 (

```
com.sun.star.awt.Selection
```

に定められた構造体で、Min および Max 属性により強調表示の開始と終了箇所を指定)

MaxTextLen (short)

フィールド内に入力可能な最大文字数。

Editable (Boolean)

テキスト入力が許可されるか (True)、拒否されるか (False) の設定 (この属性は IsEditable を介した間接的な利用のみが可能)

IsEditable (Boolean)

コントロール要素の内容の変更を許可するか、読み取り専用とするかの指定。

モデルオブジェクトからは、次の属性を利用できます。

Model.Align (short)

テキストの配置 (0: 左揃え、1: 中央揃え、2: 右揃え)。

Model.BackgroundColor (long)

コントロール要素の背景色。

Model.Border (short)

外枠の種類 (0: なし、1: 3D 表示、2: 平坦な線)。

Model.EchoChar (String)

パスワードフィールドのエコー文字。

Model.FontDescriptor (struct)

表示フォントの詳細指定用の構造体 (

```
com.sun.star.awt.FontDescriptor
```

に定められた構造体)。

Model.HardLineBreaks (Boolean)

コントロール要素のテキスト内で改行する指定。

Model.HScroll (Boolean)

テキストに横スクロールバーを使用する指定。

Model.MaxTextLen (Short)

表示テキストの最大数で、0 の指定は制限無しに対応。

Model.MultiLine (Boolean)

数行に渡る項目を許可。

Model.Printable (Boolean)

コントロール要素を印刷する指定。

Model.ReadOnly (Boolean)

コントロール要素を読み取り専用とする指定。

Model.Tabstop (Boolean)

コントロール要素を Tab キーによるフォーカス移動の対象にする指定。

Model.Text (String)

コントロール要素に関連するテキスト。

Model.TextColor (Long)

コントロール要素のテキストの色。

Model.VScroll (Boolean)

テキストに縦スクロールバーを使用する指定。

Model.HelpText (String)

コントロール要素にマウスポインタを重ねた際に表示するヘルプテキスト。

Model.HelpURL (String)

コントロール要素で使用するオンラインヘルプの URL。

リストボックス

リストボックス (

```
com.sun.star.awt.UnoControlListBox
```

サービス) は、次の属性をサポートしています。

ItemCount (Short)

要素数 (読み取り専用)。

SelectedItem (String)

強調表示された項目のテキスト (読み取り専用)。

SelectedItems (Array Of Strings)

強調表示された項目のデータフィールド (読み取り専用)。

SelectItemPos (Short)

強調表示中の項目の数 (読み取り専用)。

SelectItemsPos (Array of Short)

強調表示中の項目の数を格納したデータフィールド (複数選択可能なリストボックスのみ。読み取り専用)。

MultipleMode (Boolean)

複数選択が許可されるか (True)、拒否されるか (False) の設定 (この属性は IsMultipleMode を介した間接的な利用のみが可能)

IsMultipleMode (Boolean)

リスト内の複数選択を許可する指定 (読み取り専用)。

リストボックスには、次のメソッドが用意されています。

addItem (Item, Pos)

Item として渡された文字列を、Pos で指定するリスト位置に挿入します。

addItems (ItemArray, Pos)

文字列データフィールド ItemArray の形で渡された複数の項目を、Pos で指定するリスト位置に挿入します。

removeItems (Pos, Count)

Pos で指定するリスト位置から、Count 個の項目を削除します。

selectItem (Item, SelectMode)

文字列 Item に指定された項目の強調表示を、ブール値 SelectMode の指定に応じて切り換えます。

makeVisible (Pos)

Pos の指定位置にある項目を表示するよう、リストフィールドをスクロールします。

リストボックスのモデルオブジェクトには、次の属性が用意されています。

Model.BackgroundColor (long)

コントロール要素の背景色。

Model.Border (short)

外枠の種類 (0: なし、1: 3D 表示、2: 平坦な線)。

Model.FontDescriptor (struct)

表示フォントの詳細指定用の構造体 (

```
com.sun.star.awt.FontDescriptor
```

に定められた構造体)。

Model.LineCount (Short)

コントロール要素の行数。

Model.MultiSelection (Boolean)

項目の複数選択を許可する指定。

Model.SelectedItems (Array of Strings)

強調表示された項目のリスト。

Model.StringItemList (Array of Strings)

すべての項目のリスト。

Model.Printable (Boolean)

コントロール要素を印刷する指定。

Model.ReadOnly (Boolean)

コントロール要素を読み取り専用とする指定。

Model.Tabstop (Boolean)

コントロール要素を Tab キーによるフォーカス移動の対象にする指定。

Model.TextColor (Long)

コントロール要素のテキストの色。

Model.HelpText (String)

コントロール要素にマウスポインタを重ねた際に表示するヘルプテキスト。

Model.HelpURL (String)

コントロール要素で使用するオンラインヘルプの URL。

VBA に用意されているリスト項目への数値付加オプション (`ItemData`) は、OpenOffice.org Basic では利用できません。リストボックスの項目に数値 (データベース ID など) を割り当てるのであれば、追加のデータフィールドを用意して、両者のデータを格納するようにします。

フォーム

多くの点で、OpenOffice.org フォームの構造は、ダイアログに一致しています。その一方で、両者には次のような相違点もあります。

・各ダイアログは独立したウィンドウとして表示され、ダイアログを終了するまで、ドキュメント上に表示されているダイアログ以外の操作を行うことはできません。これに対してフォームは、図形描画要素などと同様に、ドキュメント上に直接表示されます。

・ダイアログの作成には、OpenOffice.org Basic 開発環境に用意されているダイアログエディタを利用します。フォームは、フォームコントロールとフォーム設計ツールバーを使用して、ドキュメント内で直接作成します。

・ダイアログの機能は、すべての OpenOffice.org ドキュメントで利用できるのに対して、フォームの機能を利用できるのは、文章ドキュメントと表計算ドキュメントだけです。

・フォームのコントロール要素は、外部データベーステーブルとリンクできますダイアログには、このような機能は用意されていません。

・ダイアログとフォームとでは、使用可能なコントロール要素がいくつかの点で異なります。

フォームに用意されているイベントハンドル用メソッドの使用法については、「ダイアログ」の章を参照してください。ここで説明してある内容は、フォームの場合でも同様です。

フォームの使用

OpenOffice.org のフォームは、文章ドキュメントまたは表計算ドキュメントの上に、テキストボックス、リストボックス、ラジオボタンをはじめとする各種のコントロール要素を直接配置することにより構成されます。フォームの編集には、フォームの機能ツールバー を利用します。

OpenOffice.org フォームには、デザインモードとディスプレイモードの 2 種類のモードが存在します。デザインモードでは、コントロール要素の表示位置を調整したり、属性ウィンドウにより属性 (プロパティー) を変更したりすることができます。

モードの切り替えは、フォームの機能ツールバー から行えます。

オブジェクトフォームの指定

OpenOffice.org のフォームで用いるコントロール要素は、図形描画オブジェクトと同じレベルに配置されます。実際のオブジェクトフォームには、図形描画レベルの Forms リストを利用してアクセスできます。文書ドキュメント上のオブジェクトへは、次のサンプルコードのようにしてアクセスします。

```
Dim Doc As Object
Dim DrawPage As Object
Dim Form As Object

Doc = StarDesktop.CurrentComponent
DrawPage = Doc.DrawPage
Form = DrawPage.Forms.GetByIndex(0)
```

ここで GetByIndex メソッドの戻り値としては、インデックス値 0 のフォームが返されます。

表計算ドキュメントの場合は、図形描画レベルが表計算ドキュメントの直下ではなく個々のシート (表) に置かれているため、Sheets リストを経由してアクセスする必要があります。

```
Dim Doc As Object
Dim Sheet As Object
Dim DrawPage As Object
Dim Form As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets.GetByIndex(0)
DrawPage = Sheet.DrawPage
Form = DrawPage.Forms.GetByIndex(0)
```

GetByIndex というメソッド名からも分かるように、1 つのドキュメントで複数のフォームを利用することができます。こうした機能は、複数のデータベースの内容を 1 つのドキュメント上に表示させたり、1 対 n のリレーション関係にあるデータベース情報を 1 つのフォームで表示させる場合に便利です。またこの種の処理に関しては、サブフォームの作成機能も用意されています。

フォーム用コントロール要素の構成

フォーム用のコントロール要素は、次のような 3 段階構成になっています。

・コントロール要素のモデルは、フォーム用コントロール要素を操作する場合、OpenOffice.org Basic のプログラマにとって重要なオブジェクトです。

・これに対して各コントロール要素の **View** (ビュー) オブジェクトは、実際に表示する情報を扱います。

・そして、ドキュメントに配置するフォーム用のコントロール要素は、一種の特殊な図形描画要素として扱われるため、図形描画要素に固有のコントロール要素属性を、**Shape** (シェイプ) オブジェクトというものを用いて処理しています (主として位置とサイズ)。

フォーム用コントロール要素のモデルへのアクセス

フォーム用コントロール要素のモデルへのアクセスには、フォームオブジェクトの GetByName メソッドを利用します。

```
Dim Doc As Object
Dim Form As Object
Dim Ctl As Object

Doc = StarDesktop.CurrentComponent
Form = Doc.DrawPage.Forms.GetByIndex(0)
Ctl = Form.getByName("MyListBox")
```

上記のサンプルコードでは、現在開いている文書ドキュメントの最初のフォームにある MyListBox というコントロール要素のモデルへアクセスするものとしています。

コントロール要素の配置されたフォームがどれであるか不明な場合は、すべてのフォームを対象として該当するコントロール要素を検索することもできます。

```
Dim Doc As Object
Dim Forms As Object
Dim Form As Object
Dim Ctl As Object
Dim I as Integer

Doc = StarDesktop.CurrentComponent
Forms = Doc.Drawpage.Forms

For I = 0 To Forms.Count - 1
  Form = Forms.GetbyIndex(I)
  If Form.HasByName("MyListBox") Then
    Ctl = Form.GetbyName("MyListBox")
    Exit Function
  End If
Next I
```

上記のサンプルコードでは、HasByName メソッドを用いて、MyListBox というコントロール要素モデルがあるかを、文書ドキュメント上のすべてのフォームを対象にチェックしています。そして該当するモデルを検出した段階で、変数 Ctl にその参照情報を格納して、検索処理を終了しています。

フォーム用コントロール要素のビューへのアクセス

フォーム用コントロール要素のビューにアクセスするには、関連付けられたモデルが必要です。そしてこのモデルをドキュメントコントローラに指定することにより、コントロール要素のビューを取得します。

```
Dim Doc As Object
Dim DocCrl As Object
Dim Forms As Object
Dim Form As Object
Dim Ctl As Object
Dim CtlView As Object
Dim I as Integer

Doc = StarDesktop.CurrentComponent
DocCrl = Doc.getCurrentControler()
Forms = Doc.Drawpage.Forms

For I = 0 To Forms.Count - 1
  Form = Forms.GetbyIndex(I)
  If Form.HasByName("MyListBox") Then
    Ctl = Form.GetbyName("MyListBox")
    CtlView = DocCrl.GetControl(Ctl)
    Exit Function
  End If
Next I
```

上記のサンプルコードは、1 つ前に見たコントロール要素のモデル検索用サンプルコードと非常によく似ています。ただしこのサンプルコードでは Doc というドキュメントオブジェクトに加えて、現在のドキュメントウィンドウの参照用に DocCrl というドキュメントコントローラオブジェクトを用意しています。そして、このコントローラオブジェクトに対して、先に用意したコントロール要素のモデルを渡すことにより、GetControl メソッドを使用して、フォーム用コントロール要素のビュー (ここでは変数 CtlView) を特定しています。

フォーム用コントロール要素のシェイプオブジェクトへのアクセス

コントロール要素のシェイプオブジェクトへアクセスする場合も、ドキュメントの図形描画レベルを使用します。特定のコントロール要素を特定するには、図形描画レベルにあるすべての図形描画要素を検索する必要があります。

```
Dim Doc As Object
Dim Shape as Object
Dim I as integer

Doc = StarDesktop.CurrentComponent

For i = 0 to Doc.DrawPage.Count - 1
  Shape = Doc.DrawPage(i)
  If HasUnoInterfaces(Shape, "com.sun.star.drawing.XControlShape") Then
    If Shape.Control.Name = "MyListBox" Then
      Exit Function
    End If
  End If
Next
```

上記のサンプルコードでは、すべての図形描画要素をチェックして、フォーム用コントロール要素に必要な

```
com.sun.star.drawing.XControlShape
```

インターフェースをサポートしているものがあるかを確認しています。該当するものがある場合は、Control.Name 属性を用いて、MyListBox という名前のコントロール要素があるかをチェックします。そしてこの条件も満たされたならば、検索処理を終了します。

コントロール要素のサイズと位置

先に述べたように、コントロール要素のサイズと位置の処理には、shape オブジェクトを利用します。このような処理を行うため、コントロール要素のシェイプには、他のすべての shape オブジェクトと同じように、Size および Position という属性が用意されています。

Size (struct)

コントロール要素のサイズ (

```
com.sun.star.awt.Size
```

データ構造)。

Position (struct)

コントロール要素の位置 (

```
com.sun.star.awt.Point
```

データ構造)。

次のサンプルコードでは、シェイプオブジェクトを用いた、コントロール要素のサイズと位置の指定方法を示します。

```
Dim Shape As Object

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Shape.Size = Size
Shape.Position = Point
```

このサンプルコードは、コントロール要素の shape オブジェクトは既知であることを前提としています。既知でない場合は、先のコードを利用して必要な判定処理を行う必要があります。

フォーム用コントロール要素の詳細

フォームの作成に用いるコントロール要素は、ダイアログ用のコントロール要素と多くの共通点があります。ここでは、テキストボックスをはじめ、リストボックスとコンボボックスおよび各種のボタンについて説明します。

以下に、フォーム用コントロール要素の属性のうち、重要度の高いものをまとめます。これらの属性は、対応するモデルオブジェクトにも関係します。

フォームの場合、通常のコントロール要素に加えて、テーブルコントロール要素が使用でき、これを配置することでデータベーステーブル内のデータを直接表示することができます。これは「データベースフォーム」の章で説明されています。

ボタン

フォーム用ボタンのモデルオブジェクトには、次の属性が用意されています。

BackgroundColor (long)

背景色

DefaultButton (Boolean)

標準ボタンとする指定。True を指定した場合、フォーカスのない状態で Enter キーを押した場合に反応します。

Enabled (Boolean)

コントロール要素を有効化する指定。

Tabstop (Boolean)

コントロール要素を Tab キーによるフォーカス移動の対象にする指定。

TabIndex (Long)

Tab キーによるフォーカス移動の順序の指定。

FontName (String)

フォントの種類の名前。

FontHeight (Single)

文字の高さ (ポイント)。

Tag (String)

プログラム制御によるアクセス用にボタンに格納しておく追加情報用の文字列。

TargetURL (String)

ボタンを URL にリンクさせる場合のターゲット URL。

TargetFrame (String)

TargetURL の内容を開くウィンドウ (またはフレーム) の名前 (URL 型のボタンをクリックした場合)。

Label (String)

ボタンのラベル。

TextColor (Long)

コントロール要素のテキストの色。

HelpText (String)

コントロール要素にマウスポインタを重ねた際に表示するヘルプテキスト。

HelpURL (String)

コントロール要素で使用するオンラインヘルプの URL。

ButtonType (Enum)

ボタンアクションの指定 (

```
com.sun.star.form.FormButtonType
```

のデフォルト値)。

ButtonType 属性の指定値は、ボタンをクリックした際に実行するアクションを規定します。

```
com.sun.star.form.FormButtonType
```

には、この属性指定用に次の定数値が定められています。

PUSH

標準のプッシュ式ボタン。

SUBMIT

フォーム入力の終了用 (主として HTML フォームで使用)。

RESET

すべてのフォーム入力値の初期状態へのリセット用。

URL

TargetURL に指定した URL の呼び出し用 (表示先は TargetFrame の指定ウィンドウ)。

ダイアログの場合の **OK** およびキャンセルのボタンは、フォームでは用意されていません。

ラジオボタン

オプションボタンの次の属性は、モデルオブジェクトを通じて使用します。

Enabled (Boolean)

コントロール要素を有効化する指定。

Tabstop (Boolean)

コントロール要素を Tab キーによるフォーカス移動の対象にする指定。

TabIndex (Long)

Tab キーによるフォーカス移動の順序の指定。

FontName (String)

フォントの種類の名前。

FontHeight (Single)

文字の高さ (ポイント)。

Tag (String)

プログラム制御によるアクセス用にボタンに格納しておく追加情報用の文字列。

Label (String)

ボタンの表書き。

Printable (Boolean)

コントロール要素を印刷する指定。

State (Short)

オプションをアクティブとするか (属性値を 1 とした場合)、非アクティブとするかの指定 (その他の値の場合)。

RefValue (String)

追加情報用の文字列 (データレコード ID の管理用などに使用)。

TextColor (Long)

コントロール要素のテキストの色。

HelpText (String)

コントロール要素にマウスポインタを重ねた際に表示するヘルプテキスト。

HelpURL (String)

コントロール要素で使用するオンラインヘルプの URL。

ラジオボタンをグループ化する方法は、ダイアログとフォームとで異なります。ダイアログの場合は、タブストップの順番が連続したものは自動的にグループ化されますが、フォームの場合は、コントロール要素の名前を基準にしてグループ化が行われます。つまりグループ化するラジオボタンには、すべて同じ名前をつけます。OpenOffice.org その際には、グループ内の全コントロール要素を 1 つの配列として管理するため、OpenOffice.org Basic によるボタン制御はこれまでと同様の方式で実行できます。

次のサンプルコードでは、グループ化したコントロール要素へのアクセス法を示します。

```
Dim Doc As Object
Dim Forms As Object
Dim Form As Object
Dim Ctl As Object
Dim I as Integer

Doc = StarDesktop.CurrentComponent
Forms = Doc.Drawpage.Forms

For I = 0 To Forms.Count - 1
  Form = Forms.GetbyIndex(I)
  If Form.HasByName("MyOptions") Then
    Ctl = Form. GetGroupbyName("MyOptions")
    Exit Function
  End If
Next I
```

上記のサンプルコードの処理の流れは、グループ化しない単独のコントロール要素へのアクセス法を説明した、先のサンプルコードのものと基本的に同じです。ここでは、現在の文書ドキュメントにあるすべてのフォームを取得してから、ループと HasByName メソッドを用いて、MyOptions という名前のコントロール要素が配置されたフォームがあるかをチェックしています。そして該当するフォームを検出した時点で、目的とするモデル配列への参照情報を GetGroupByName メソッドにより取得します (グループ化していないモデルの場合に使用したメソッドは GetByName)。

チェックボックス

チェックボックスのモデルオブジェクトでは、次の属性を使用できます。

Enabled (Boolean)

コントロール要素を有効化する指定。

Tabstop (Boolean)

コントロール要素を Tab キーによるフォーカス移動の対象にする指定。

TabIndex (Long)

Tab キーによるフォーカス移動の順序の指定。

FontName (String)

フォントの種類の名前。

FontHeight (Single)

文字の高さ (ポイント)。

Tag (String)

プログラム制御によるアクセス用にボタンに格納しておく追加情報用の文字列。

Label (String)

ボタンのラベル。

Printable (Boolean)

コントロール要素を印刷する指定。

State (Short)

オプションをアクティブとするか (属性値を 1 とした場合)、非アクティブとするかの指定 (その他の値の場合)。

RefValue (String)

追加情報用の文字列 (データレコード ID の管理用などに使用)。

TextColor (Long)

コントロール要素のテキストの色。

HelpText (String)

コントロール要素にマウスポインタを重ねた際に表示するヘルプテキスト。

HelpURL (String)

コントロール要素で使用するオンラインヘルプの URL。

テキストボックス（テキストフィールド）

テキストフィールドフォームのモデルオブジェクトでは、次の属性を使用できます。

Align (short)

テキストの配置 (0: 左揃え、1: 中央揃え、2: 右揃え)。

BackgroundColor (long)

コントロール要素の背景色。

Border (short)

外枠の種類 (0: なし、1: 3D 表示、2: 平坦な線)。

EchoChar (String)

パスワードフィールドのエコー文字。

FontName (String)

フォントの種類の名前。

FontHeight (Single)

文字の高さ (ポイント)。

HardLineBreaks (Boolean)

コントロール要素のテキスト内で改行する指定。

HScroll (Boolean)

テキストに横スクロールバーを使用する指定。

MaxTextLen (Short)

表示テキストの最大数で、0 の指定は制限無しに対応。

MultiLine (Boolean)

複数行項目を許可する指定。

Printable (Boolean)

コントロール要素を印刷する指定。

ReadOnly (Boolean)

コントロール要素を読み取り専用とする指定。

Enabled (Boolean)

コントロール要素を有効化する指定。

Tabstop (Boolean)

コントロール要素を Tab キーによるフォーカス移動の対象にする指定。

TabIndex (Long)

Tab キーによるフォーカス移動の順序の指定。

FontName (String)

フォントの種類の名前。

FontHeight (Single)

文字の高さ (ポイント)。

Text (String)

コントロール要素のテキスト。

TextColor (Long)

コントロール要素のテキストの色。

VScroll (Boolean)

テキストに縦スクロールバーを使用する指定。

HelpText (String)

コントロール要素にマウスポインタを重ねた際に表示するヘルプテキスト。

HelpURL (String)

コントロール要素で使用するオンラインヘルプの URL。

リストボックス

ボックスのモデルオブジェクトには、次の属性が用意されています。

BackgroundColor (long)

コントロール要素の背景色。

Border (short)

外枠の種類 (0: なし、1: 3D 表示、2: 平坦な線)。

FontDescriptor (struct)

表示フォントの詳細指定用の構造体 (

```
com.sun.star.awt.FontDescriptor
```

に定められた構造体)。

LineCount (Short)

コントロール要素の行数。

MultiSelection (Boolean)

項目の複数選択を許可する指定。

SelectedItems (Array of Strings)

強調表示された項目のリスト。

StringItemList (Array of Strings)

すべての項目のリスト。

ValueItemList (Array of Variant)

個々の項目に付加する追加情報のリスト (データレコード ID の管理用などに使用)。

Printable (Boolean)

コントロール要素を印刷する指定。

ReadOnly (Boolean)

コントロール要素を読み取り専用とする指定。

Enabled (Boolean)

コントロール要素を有効化する指定。

Tabstop (Boolean)

コントロール要素を Tab キーによるフォーカス移動の対象にする指定。

TabIndex (Long)

Tab キーによるフォーカス移動の順序の指定。

FontName (String)

フォントの種類の名前。

FontHeight (Single)

文字の高さ (ポイント)。

Tag (String)

プログラム制御によるアクセス用にボタンに格納しておく追加情報用の文字列。

TextColor (Long)

コントロール要素のテキストの色。

HelpText (String)

コントロール要素にマウスポインタを重ねた際に表示するヘルプテキスト。

HelpURL (String)

コントロール要素で使用するオンラインヘルプの URL。

フォーム用のリストボックスに用意された `ValueItemList` 属性は、VBA の `ItemData` 属性に相当するもので、リストの各項目に追加情報を付加する際に利用できます。

リストボックスの場合、ビューオブジェクトに対して次の属性が用意されています。

addItem (Item, Pos)

`Item` として渡された文字列を、`Pos` で指定するリスト位置に挿入します。

addItems (ItemArray, Pos)

文字列データフィールド ItemArray の形で渡された複数の項目を、Pos で指定するリスト位置に挿入します。

removeItems (Pos, Count)

Pos で指定するリスト位置から、Count 個の項目を削除します。

selectItem (Item, SelectMode)

文字列 Item に指定された項目の強調表示を、ブール値 SelectMode の指定に応じて切り換えます。

makeVisible (Pos)

Pos の指定位置にある項目を表示するよう、リストフィールドをスクロールします。

データベースフォーム

OpenOffice.org のフォームは、データベースと直接リンクさせることができます。このようなフォームでは、特別なプログラムを用意することなく、データベースのフロントエンド機能をそのまま利用できます。

ページを移動したり、選択したテーブルやクエリーを検索したり、データレコードを変更して新しいデータレコードを挿入することができます。OpenOffice.org は、関連データはデータベースから取得されており、変更はデータベースに書き戻されていることを自動的に確認します。

データベースフォームの構造は、基本的に OpenOffice.org の通常のフォームと同じものです。ただし、通常の属性の他に次のようなデータベース固有の属性があるため、これらに対して必要な設定をする必要があります。

DataSourceName (String)

データソースの名前 (「データベースアクセス」の章説明を参照。OpenOffice.org のデータソースは広域アクセスできるよう準備しておく必要あり)。

Command (String)

テーブル、クエリー、SQL 選択コマンドなどのリンク対象の名前。

CommandType (Const)

Command 属性の対象がテーブル、クエリー、SQL コマンドのいずれであるかの指定 (

```
com.sun.star.sdb.CommandType
```

に定められた列挙型)。

```
com.sun.star.sdb.CommandType
```

には次の値を指定できます。

TABLE

テーブル

QUERY

クエリー

COMMAND

SQL コマンド

個々のコントロール要素へのデータベースフィールドの割り当ては、次の属性で指定します。

DataField (String)

リンクされているデータベースフィールドの名前。

テーブル

データベースの操作には、テーブルコントロール要素という別のコントロール要素が用意されています。完全なデータベーステーブルまたはクエリーの内容を示します。たとえばテーブルコントロールからデータベースへのリンクの構築は、オートパイロットによるフォーム作成機能を利用すると簡単に実行でき、その際にはデータベースフィールドとテーブル列のリンクに関する必要な設定を行うこともできます。