StarSuite™ 7 Office Suite

A Sun™ ONE Software Offering

Basic プログラマ向けガイド

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A. 650-960-1300

Part No. 817-3925-10
2003, Revision A

著作権と商標について

目次

第 1 章

はじめに

このマニュアルでは、StarSuite 7 Basic でのプログラミングおよび、StarSuite に用意された StarSuite Basic を用いたアプリケーションの使用法について、基本的な説明をします。なお本マニュアルの読者には、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、StarSuite 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。

このマニュアルでは、Microsoft Visual Basic および旧バージョンの StarSuite Basic に精通した読者に対して、コードの移行に関する各種のヒントを紹介しています。該当箇所は、ページの片側にマークが表示されています。また、必要なヒントをすぐ探せるように、個 のヒントの掲載位置を一覧にまとめてあります。

StarSuite Basic とは

StarSuite Basic のプログラミング言語は、StarSuite 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、StarSuite 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 StarSuite Basic をマスターできるはずです。StarSuite Basic には、Visual Basic と共通する部分が多くあります。

StarSuite Basic のプログラミング言語を構成する要素は、大きく分けて以下の 4 つに分類できます。

- **StarSuite Basic プログラミング用の言語:** プログラミング言語を構成する基本的な要素のことで、たとえば変数宣言、ループ、関数などがこれに該当します。
- **実行時ライブラリ:** StarSuite 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。
- **StarSuite API (Application Programming Interface):** StarSuite ドキュメントへのアクセスに必要な機能を提供するもので、これらドキュメントの作成、保存、編集、印刷などを行う際に使用します。
- **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。

StarSuite Basic と VBA の互換性については、StarSuite 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 StarSuite 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、StarSuite に用意された多くの機能が使用不可能となる)。

StarSuite Basic の対象ユーザー

StarSuite Basic を用いたアプリケーションは、StarSuite に用意された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します。たとえば StarSuite 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。

StarSuite Basic は、すべての StarSuite の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。

StarSuite Basic の使用

StarSuite Basic は、すべての StarSuite 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以下の StarSuite 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):** マクロの構築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:** StarSuite Basic マクロの実行に必要です。
- **各種の StarSuite アプリケーションに対するインターフェース:** StarSuite ドキュメントに直接アクセスする際に必要となります。

本マニュアルの構成

最初の 3 つの章では、StarSuite Basic の操作に必要な基本知識について説明します。

- 第 2 章「StarSuite Basic のプログラミング言語」
- 第 3 章「StarSuite Basic の実行時ライブラリ」
- 第 4 章「StarSuite API について」

これらの章では StarSuite Basic の概要を説明しています。StarSuite 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 StarSuite API の個別的な説明です。必要に応じて目を通すようにしてください。

- 第 5 章「StarSuite ドキュメントの操作」
- 第 6 章「文書ドキュメント」
- 第 7 章「表計算ドキュメント」
- 第 8 章「図形描画とプレゼンテーション」
- 第 9 章「グラフ (ダイアグラム)」
- 第 10 章「データベースアクセス」

- 第 11 章「ダイアログ」
- 第 12 章「フォーム」

詳細情報

本マニュアルで説明する StarSuite API コンポーネントは、StarSuite Basic のプログラミングをする際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。このような API の詳細情報については、以下のインターネットサイトを参照してください。

```
http://api.openoffice.org/common/ref/com/sun/star/module-ix.html
```

StarSuite API については『』に本書よりも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。それでも StarSuite Basic にある程度習熟していれば、StarSuite Basic および StarSuite でのプログラミングで役立つ情報を『』から入手できます。『』は、以下のインターネットサイトからダウンロードできます。

```
http://api.openoffice.org/DevelopersGuide/DevelopersGuide.html
```

なお StarSuite Basic ではなく、Java や C++ により直接プログラミングを行うのであれば、本マニュアルの代わりに『』を参照してください。ただし Java および C++ によるプログラミングでは、StarSuite Basic によるプログラミングよりも、非常に複雑な過程を経る必要があります。

第 2 章

StarSuite Basic のプログラミング言語

StarSuite Basic は、いわゆる Basic プログラミング言語の一種です。また StarSuite 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、StarSuite 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、StarSuite Basic は簡単にマスターできるはずです。StarSuite 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 StarSuite 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。StarSuite 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、StarSuite Basic プログラミング言語の主要コンポーネント、および StarSuite 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。

StarSuite Basic プログラムの概要

StarSuite Basic はインタプリタ型の言語です。StarSuite のコンパイラでは、C++ や Turbo Pascal とは異なり、自動実行型の実行可能ファイル (EXE ファイル) を作成することはできません。StarSuite Basic のプログラムは、ボタンをクリックして実行します。これらのコードは、事前に文法的な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、コードを記入する表示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。以下のサンプルコードでは、1 行の数値演算行を 4 行に分割します。

```
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 StarSuite 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。たとえば以下のコードは、

```
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

StarSuite 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、トラブルシューティングや移植時の参考情報などを記入しておくことができます。

StarSuite Basic の場合、プログラムコード内へのコメント行の記述は、以下の 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 ' これは変数 A に関するコメントです。
```

- キーワード Rem に続く部分もコメントと見なされます。

```
Rem キーワード Rem に続けた部分もコメントとみなされます。
```

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。それ以降の行は、StarSuite Basic により通常の実行行として解釈されます。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       ' これは変数 B に関するコメントですが、複数行
            ' にわたる長めの目の文章となっています。その
            ' ような場合は、コメント指定記号を各行に付け
            ' る必要があります。
```

マーカー

StarSuite 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) を使用することができます。マーカーの名前は、以下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、アルファベット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えません。
- 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` という変数を示すマーカーは、`onetestVariable` や `ONETESTVARIABLE` と同じものと見なされます。

なお、このルールには例外があります。UNO-API 定数については、大文字と小文字が区別されます。UNO の詳細情報については、第 4 章を参照してください。

StarSuite Basic と VBA では、マーカーの記述規則が異なります。たとえば StarSuite Basic ではマーカーに特殊文字が使えませんが、これは多言語プロジェクトを扱う際に問題を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          ' 有効
Surname5         ' 有効 (数字の 5 が先頭に来ていないので有効)
First Name       ' 無効 (スペース記号は使えない)
DéjàVu           ' 無効 (欧文特殊文字は使えない)
5Surnames        ' 無効 (数字は先頭に使えない)
First,Name       ' 無効 (コンマやピリオド記号は使えない)
```

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、StarSuite 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、以下の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推奨されるものではなく、タイプミスなどによる混乱を招く原因となる場合もあります。たとえば間違った変数名を入力しても、エラーメッセージが出されるわけではなく、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進めてしまいます。このようなバグについては、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、StarSuite Basic には以下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が、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。

変数の明示的宣言を行う一番簡単な方法は、以下のようなコマンドを記述することです。

```
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、その変数型をバリアント型としています。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。以下は、各種のバリアント型変数の例です。

```
MyVar = "Hello World"          ' 文字列型データを割り当て
MyVar = 1                      ' 整数型データを割り当て
MyVar = 1.0                    ' 浮動小数点型データを割り当て
MyVar = True                   ' ブール型データを割り当て
```

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他の変数型のデータを割り当てることもできます。なお、たしかに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、StarSuite 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、以下のようにします。

```
Dim MyVar As Integer    ' 数値型の変数を宣言
```

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、以下のようにしても行えます。

```
Dim MyVar%                ' 数値型の変数を宣言
```

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、StarSuite Basic はバリアント型として変数宣言を行います。たとえば以下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、StarSuite 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。

文字列型

文字列型は、数値型とあわせて、StarSuite 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やプリンタへの文字出力も、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では、â、ä、î などのいわゆる欧文特殊文字や、キリル文字などは含まれていません。

ANSI 文字コードセット

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、様々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、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を規定しています。現在 Unicode の Version 2.0 は、StarSuite および StarSuite 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

StarSuite 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。StarSuite 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。その際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、以下の形式で行います。

```
Dim Variable As String
```

この変数宣言は、以下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際には、StarSuite 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."   ' 格納される文字列は a "-quotation mark
```

数値型

StarSuite Basic では、基本的な 5 種類の数値型が用意されています。

- 整数型
- ロング整数型
- 浮動小数点型
- 倍精度型
- 通貨型

整数変数

整数変数には -32768 から 32767 までの整数を収めることができます。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、以下の形式で行います。

```
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 までの整数を収めることができます。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、以下の形式で行います。

```
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、以下の形式で行います。

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

倍精度変数

倍精度変数には $1.79769313486232 \times 10^{308}$ から $1.401298 \times 4.94065645841247 \times 10^{-324}$ 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。1 つの倍精度変数が消費するメモリ量は、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、以下の形式で行います。

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

通貨変数

通貨変数では、他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、以下の形式で行います。

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

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。StarSuite Basic で扱う数値には、以下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 Single

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 Single

A = + 121
B = - 243
```

小数

小数点にはピリオド記号を使用します。これは、他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 Single

A = 1223.53       ' 値は整数に丸められます
B = - 23446.46    ' 値は整数に丸められます
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)。

整数変数に小数値を代入すると、StarSuite Basic は小数点以下を丸めて整数化します。

指数表示

StarSuite 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        ' 有効
A = + 1.43E2      ' 有効 (プラス記号と数値の間のスペース記号の有無)
A = - 1.43E2      ' 有効 (マイナス記号と数値の間のスペース記号の有無)
A = 1.43E-2       ' 有効 (負の指数)

A = 1.43E -2      ' 無効 (数値内にスペース記号は使用不可)
A = 1,43E-2       ' 無効 (小数点としてコンマ記号は使用不可)
A = 1.43E2.2      ' 無効 (指数部の数値は整数のみ使用可能)
```

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。たとえば最初の例では、次のように解釈されます。

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

これは 1.43 マイナス 2 と見なされるので、結果として -0.57 という値が返されます。この場合、本来意図していた数値表式に該当すべき値は $1.43 * 10^{-2}$ (つまり 0.0143) です。また最後の例では、次のように解釈されます。

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

StarSuite Basic は指数部の数値の小数点以下を無視するので、この数式は以下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には、1 バイトの値を 2 桁の数値で正確に表示できるというメリットがあります。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。StarSuite 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 Longer

A = &HFF    ' 16 進数の FF は 10 進数の 255 に該当
A = &H10    ' 16 進数の 10 は 10 進数の 16 に該当
```

8 進数

StarSuite Basic では、0 から 7 までの数字の組合せで数値を表現する 8 進数表記を扱うことも可能で、8 進数であることを指定するには &O を数値の先頭に付けます。

```
Dim A As Longer

A = &O77    ' 8 進数の 77 は 10 進数の 63 に該当
A = &O10    ' 8 進数の 10 は 10 進数の 8 に該当
```

True と False を取る変数

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、以下の形式で行います。

```
Dim Variable As Boolean
```

日付および時刻変数

日付変数

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。StarSuite 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、以下の形式で行います。

```
Dim Variable As Date
```

データフィールド

StarSuite Basic では、通常の変数 (スカラー値) の他に、データフィールド (配列) を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が、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。配列は以下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、たとえば以下の例では 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。以下の例は 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。以下の例では、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、以下の 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、以下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、StarSuite 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、以下のサンプルコードのようにして配列を宣言したとします。

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

この場合は `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、StarSuite 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。StarSuite 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、StarSuite 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。

多次元データフィールド

StarSuite 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。たとえば以下のサンプルコードでは、多次元データフィールドを宣言しています。

```
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。

StarSuite 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、配列を 1 つ作成して、ある文章内に存在する A という文字で始まるすべての単語をその中に格納するとしましょう。このようなケースでは、該当する単語がいくつあるのか最初の段階では分からないので、配列のサイズを随時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、StarSuite Basic では以下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、StarSuite 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。

以下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           ' 要素数 5 として配列を宣言

' ...

ReDim MyArray(10) As Integer        ' 要素数を 11 に変更

' ...

ReDim MyArray(20) As Integer        ' 要素数を 21 に変更
```

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                      ' 最初のサイズ
                                                ' を指定

' ...

ReDim Preserve MyArray(20)    As Integer        ' 代入値を保持しつつ
                                                ' データフィールドの
                                                ' サイズを拡張
```

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、StarSuite 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。

変数の有効範囲と寿命

StarSuite 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寿命があります。変数が確保され続ける寿命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、変数を静的変数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 StarSuite 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 StarSuite 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キーワード `Dim` 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
```

このサンプルコードで変数 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

プライベート変数は、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プライベート変数が使われている場合、StarSuite 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。以下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、変数名 C というプライベート変数を確保します。ここで関数 `Test` は、最初にモジュール A でプライベート変数を作成し、次にモジュール B でプライベート変数を作成します。

モジュール A:

```
Private C As Integer

Sub Test
      SetModuleA        ' モジュール A で変数 C の値を設定
      SetModuleB        ' モジュール B で変数 C の値を設定

      ShowVarA          ' モジュール A での変数 C の値を表示 (= 10)
      ShowVarB          ' モジュール B での変数 C の値を表示 (= 20)
End Sub

Sub SetmoduleeA
      A = 10
End Sub

Sub ShowVarA
      MsgBox C          ' モジュール A での変数 C の値を表示
End Sub
```

モジュール B:

```
Private C As Integer

Sub SetModuleB
      A = 20
End Sub

Sub ShowVarB
      MsgBox C          ' モジュール B での変数 C の値を表示
End Sub
```

定数

StarSuite 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
```

演算子

StarSuite 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。

`+` 数値データに対する加算 (足し算) を行うほか、文字列を結合させます。

`-` 数値データに対する減算 (引き算) を行います。

`*` 数値データに対する乗算 (掛け算) を行います。

`/` 数値データに対する除算 (割り算) を行います。

`\` 数値データに対する整数除算 (結果を整数で返す割り算) を行います。

`^` 数値データに対する指数演算 (累乗、べき乗) を行います。

`MOD` 整数除算を実行した際の余りの値を返します。

論理演算子

論理演算子とは、いわゆる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) 演算を行います。

OR　論理和 (Or) 演算を行います。

XOR　排他的論理和 (Xor) 演算を行います。

NOT　論理否定 (Not) 演算を行います。

EQV　論理等価 (Eqv) 演算を行います (両者の値が共に True または False かを判定)。

IMP　論理包含 (Imp) 演算を行います (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、両者が等しいかを比較。

<>　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。

VBA の Like 演算子は、StarSuite Basic には用意されていません。

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、これは以下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 より大きい場合は 2 となり、それ以外の場合は 0 となります。

If ステートメントをカスケード化して、以下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。以下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

      ' ... 変数 Var の値が 1 から 5 の間にある場合の処理

Case 6, 7, 8

      ' ... 変数 Var の値が 6、7、8 のいずれかの場合の処理

Case Var > 8 And Var < 11

      ' ... 変数 Var の値が 8 より大きく 11 より小さい場合の処理

Case Else

      ' ... その他の場合の処理

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。このタイプのループは、以下のサンプルコードのような形式で指定します。

```
Dim I

For I = 1 To 10

      ' ...  ループの内部

Next I
```

この場合、変数 `I` がループカウンタであり、初期値は 1 とされています。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。

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

```
Dim I

For I = 1 To 10 Step 0.5

      ' ... ループの内部

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

      ' ... ループの内部

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。以下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

      ' ... ループの内部

Next I
```

VBA にある For Each...Next ループは、StarSuite Basic では用意されていません。

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 つのバリエーションがあります (以下のサンプルコードで A > 10 の部分がループ条件)。

1. Do While...Loop

   ```
   Do While A > 10
         ' ... ループ処理の本体
   Loop
   ```

   この場合は、各ループの実行前に条件判定を行い、条件が満たされている場合のみループの内部を実行します。

2. Do Until...Loop

   ```
   Do Until A > 10
         ' ... ループ処理の本体
   Loop
   ```

   この場合は、ループの条件判定が満たされなくなるまで、繰り返しループを実行します。

3. Do...Loop While

   ```
   Do
         ' ... ループ処理の本体
   Loop While A > 10
   ```

   この場合は、ループの内部を最低 1 回は実行してから条件判定を行い、条件が満たされなくなった時点でループを終了します。

4. Do...Loop Until

   ```
   Do
         ' ... ループ処理の本体
   Loop Until A > 10
   ```

この場合も、ループの内部を最低 1 回は実行してから条件判定を行いますが、条件が満たされるまでループを続けます。

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

      ' ... ループ処理の本体
While A > 10
```

サンプルプログラム: 多重ループによるソーティング

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。以下のサンプルコードは、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 10
            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。

手続きと関数

手続きと関数こそは、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。

手続き

手続きは、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。構文は以下のようになります。

```
Sub Test

      ' ... 手続き本体のコード

End Sub
```

このサンプルコードでは Test という名前の手続きを定義していますが、この手続きによる処理は、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。手続きを呼び出すには、以下のようにプログラム内の該当行に手続き名を記述するだけです。

```
Test
```

関数

関数も、手続き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。手続きとの相違点として、関数は戻り値を返すという点があります。

```
Function Test

      ' ... 関数本体のコード

      Test = 123
End Function
```

関数の戻り値の指定は、通常の変数への値の代入と同様の形式で、関数名に対して値を代入することで実施します。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、以下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、その関数の戻り値としてはゼロ値が返されます (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
```

```
    ' ... 関数本体のコード

End Function
```

明示的に値を指定しなかった場合、戻り値はバリアント型として扱われます。

手続きと関数の強制終了

StarSuite Basic では、`Exit Sub` および `Exit Function` コマンドを使って、実行途中の手続きや関数を強制終了させることができ、エラーハンドルなどの処理をする際に有用です。これらのコマンドは、手続き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。

以下のサンプルコードでは、変数 `ErrorOccured` の値が `True` になった場合に、手続きを強制終了します。

```
Sub Test
      Dim ErrorOccured As Boolean

      ' ...

      If ErrorOccured Then
            Exit Sub
      End If

      ' ...

End Sub
```

パラメータの渡し方

関数や手続きには、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、関数や手続きの名前の後に、かっこで囲んで指定する必要があります。これは以下のように指定します。

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

この手続きを呼び出す場合は、整数変数 A および文字列変数 B へ渡すパラメータ値の指定が必要です。

StarSuite Basic では、通常パラメータは参照渡しで与えます。この場合、呼び出した手続き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)
      ' この段階で変数 A の値は 20
End Sub
Sub ChangeValue(TheValue As Integer)
       TheValue = 20
End Sub
```

上記のサンプルコードでは、手続き Test の内部で定義した変数 A を、関数 ChangeValue にパラメータとして渡しています。次に TheValue に代入するためこのパラメータの値を 20 に変更していますが、この変更後の値は関数が終了してもそのまま維持されます。

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。パラメータを値渡しで与えるには、関数宣言用のヘッダ部で、該当する変数名の前にキーワード ByVal を指定する必要があります。

たとえば先のサンプルコードで、関数 ChangeValue に渡すパラメータを値渡しにするには、以下のように指定します。

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

このように変更すると、変数 A の値は、呼び出した関数内部での値の変更に影響されなくなります。この場合、関数 ChangeValue を呼び出した後も、変数 A の値は 10 のままです。

StarSuite Basic における手続きや関数へのパラメータの渡し方は、基本的に VBA と同じものです。標準でパラメータは、参照渡しとして与えられます。そのため、値渡しでパラメータを与えるには、キーワード ByVal を指定する必要があります。なお VBA の場合は、キーワード ByRef を指定することで、強制的に参照渡しでパラメータを与えることが可能です。StarSuite Basic にこのキーワードは用意されていませんが、これは StarSuite Basic の場合は標準で参照渡しとなるためです。

通常、StarSuite Basic の関数や手続きは、いわゆる `Public` 扱いになります。ただし、VBA にあるキーワード `Public` および `Private` は、StarSuite Basic ではサポートされていません。

オプションパラメータ

関数や手続きを呼び出す際には、必要なパラメータをすべて指定する必要があります。

ただし StarSuite Basic では、特定のパラメータをオプション (省略可能) となるよう指定が可能で、これらのパラメータについては、呼び出し時の指定が省略されても、値が空であるものとして、そのまま処理が進められます。このような指定は、以下のようにして行います。

```
Sub Test(A As Integer, Optional B As Integer)

End Sub
```

この場合パラメータ A の指定は必須ですが、パラメータ B の指定は省略できます。

`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

      ' パラメータ B が指定されているかをチェック
      If Not IsMissing (B) Then
            B_Local = B       ' パラメータ B が指定されている場合
      Else
            B_Local = 0       ' パラメータ B が指定されていない場合、標準値は 0
      End If

      ' ... 関数の本体

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。そしてパラメータに値が指定されていなかった場合、`B_Local` にはパラメータ値ではなく標準値 (この例では `0`) が代入されます。

VBA には、このようなオプションパラメータに適用する標準値を指定する機能がありますが、StarSuite Basic には用意されていません。

VBA にあるキーワード `ParamArray` は、StarSuite Basic ではサポートされていません。

再帰処理

現行の StarSuite Basic では、再帰処理が行えるようになりました。再帰処理とは、関数や手続き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。

以下のサンプルコードは再帰関数の使用例で、ここでは 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 )    ' 表示結果は 1,40500611775288E+51
   Msgbox CalculateFactorial( -42 )    ' 表示結果は「Invalid number for factorial!」
   Msgbox CalculateFactorial( 3.14 )   ' 表示結果は「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
       ' 再帰呼び出しの実行行
       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 で再帰呼び出しを行い、0! = 1 の計算になった段階を終了条件としています。

StarSuite Basic での再帰回数には 500 レベルという制限があるので注意が必要です。

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作業です。StarSuite Basic には、エラー処理用に各種の機能が用意されています。

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

      ' ... エラー発生時に実行行を引き継ぎ

Exit Sub

ErrorHandler:

      ' ... エラー処理用のコード

End Sub
```

ここで `On Error Goto ErrorHandler` 行により、StarSuite Basic に対して、エラー発生時に実行行をどのようにするかを指定しています。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 StarSuite 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:

      ' ... エラー処理用のコード

      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実行の行き先を指定することができます。

```
ErrorHandler:

      ' ... エラー処理用のコード
      Resume Proceed

Proceed:

      ' ... エラー処理後のプログラムの継続先
```

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、以下のように記述します。

```
Sub Test
      On Error Resume Next

      ' ... エラー発生時は次の行を継続実行

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。詳細情報については「効率的なエラー処理のヒント」を参照してください。

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。

このような情報を表示させるには、たとえば以下のようなコードを記述します。

```
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、StarSuite 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。
StarSuite Basic では、On Error および 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。

以下のサンプルコードは、エラー処理の手順の流れを示しています。

```
Sub Example

	' エラーハンドラをプログラムの先頭部に設定
	On Error Goto ErrorHandler

	' ... 本体のコードの先頭部

	' これ以降はエラーハンドラ機能を解除
	On Error Goto 0

	' 本体のコードの末尾
Exit Sub

' エラーハンドラの開始点
ErrorHandler:

	' あらかじめ想定されたエラーか判定
	If Err = ExpectedErrorNo Then
		' ... エラー処理用のコード
	Else
		' ... 想定外のエラーの場合は警告を表示
	End If

	On Error Goto 0			' これ以降はエラーハンドラ機能を解除
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。

第 3 章

StarSuite Basic の実行時ライブラリ

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

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 A に変数 B を代入すると、自動的に StarSuite 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。一方の変数 A 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 A に変数 B を代入する際には、変数 A 側の内部フォーマットに一致するよう、変数 B 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、以下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、特定の規則を設けてそれを実践するしかありません (このような理由からもバリアント型変数の使用は推奨できない)。

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、StarSuite 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、StarSuite 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 = CInt(B + C)          ' B と C の値を加えてから変換を実行
                           (結果は数値の 2)
A = CStr(B) + Cstr(C)    ' B と C の値を文字列に変換してから結合
                         ' (結果は文字列の "11")
```

このサンプルコードの最初の変換操作では、StarSuite 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 関数で数値、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、Cstr 関数など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)        '  この場合、ロケール設定にかかわらず正しく変換されます
```

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                    ' エラーメッセージを表示
```

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                    ' エラーメッセージを表示
```

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。

StarSuite Basic にはこのような処理を行うために、以下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、ユーザーからの入力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。

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、以下の関数 `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。

StarSuite Basic では、数値以外の文字列を数値として代入しようとしても、エラーメッセージは表示されませんが、実際には `0` という値が代入されます。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

StarSuite 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 (文字) との間の変換を行います。以下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")         ' アルファベットの A (Unicode 値 65)
Code = Asc("€")         ' ユーロ通貨記号 (Unicode 値 8364)
Code = Asc("л")         ' キリル文字の (Unicode 値 1083)
```

以下のサンプルコードは、これ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、以下のサンプルコードのように使用します。

```
MyString = Chr(9) + "That 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

StarSuite Basic には、文字列の一部の取得用に、以下の 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。

以下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 Strings
Dim MyLen As Integer

MyString = "This is a small test"

MyResult = Left(MyString,5)         ' 得られる文字列は "This "
MyResult = Right(MyString, 5)       ' 得られる文字列は " test"
MyResult = Mid(MyString, 8, 5)      ' 得られる文字列は " a sm"
MyLength = Len(MyString)            ' 得られる値は 20
```

検索と置換

StarSuite 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 (SearchString, MyString)
```

この関数は、文字列変数 SearchString 内に文字列変数 MyString 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、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、以下の形式で記述します。

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

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、以下のように最終パラメータとして 0 を追加します。

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

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、以下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、文字列パラメータ NewPart 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、StarSuite 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。この機能は、以下のサンプルコードのようにして使用します。

```
Dim MyString As String

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

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。

文字列の書式設定

Format 関数は、数値に書式設定を施した上で文字列に変換します。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 (#)、ピリオド記号 (.)、コンマ記号 (,) ドル記号 ($) の 5 種類です。

ゼロは、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。

ピリオド記号は、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。

以下のサンプルコードは、ゼロとピリオド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)      ' 結果は "-1579,80"
MyString = Format(1579.8, MyFormat)       ' 結果は "1579,80"
MyString = Format(0.4, MyFormat)          ' 結果は "0,40"
MyString = Format(0.434, MyFormat)        ' 結果は "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)      ' 結果は "-1579,80"
MyString = Format(1579.8, MyFormat)       ' 結果は "1579,80"
MyString = Format(0.4, MyFormat)          ' 結果は "0000,40"
MyString = Format(0.434, MyFormat)        ' 結果は "0000,40"
```

カンマ記号は、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。ナンバー記号は、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)      ' 結果は "-1,579,80"
MyString = Format(1579.8, MyFormat)       ' 結果は "1,579,80"
MyString = Format(0.4, MyFormat)          ' 結果は "0,40"
MyString = Format(0.434, MyFormat)        ' 結果は "0,43"
```

ドル記号は、通貨記号を表示させるための記号で、通貨記号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。

```
MyFormat = "#,##0.00 $"
```

```
MyString = Format(-1579.8, MyFormat)      ' 結果は "-1.579,80 €"
MyString = Format(1579.8, MyFormat)       ' 結果は "1.579,80 €"
MyString = Format(0.4, MyFormat)          ' 結果は "0,40 €"
MyString = Format(0.434, MyFormat)        ' 結果は "0,43 €"
```

VBA で用意されている日付および時刻の書式設定用プレースホルダは、StarSuite Basic ではサポートされていません。

日付および時刻

StarSuite 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 = "2002.1.1"
```

このような文字列形式の日付変数へ代入した場合、StarSuite 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。

StarSuite 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, 1)
```

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

以下の関数は、`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、指定した日付データに該当する日付や時刻の情報を取得します。以下のサンプルコードは、これらの使用例です。

```
Dim MyDate As Date

' ... 変数 MyDate を初期化

If Year(MyDate) = 2003 Then

      ' ... 該当年が 2003 年の場合
End If
```

上記の例では、変数 MyDate の日付データが 2003 年に該当するかをチェックしています。以下のサンプルコードも同様です。

```
Dim MyTime As Date

' ... 変数 MyTime を初期化

If Hour(MyTime) >= 12 And Hour(MyTime) < 14 Then

      ' ... 該当年が 2003 年の場合
End If
```

上記の例では、変数 MyTime の該当時刻が 12 時から 14 時の間にあるかをチェックしています。

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

' ... 変数 MyDate を初期化

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
```

注意: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

StarSuite Basic には、システム日付と時刻の取得用に、以下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の操作

ファイル操作は、アプリケーションの基本機能の 1 つです。StarSuite API には、StarSuite 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。詳細情報については、第 4 章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。StarSuite Basic の実行時ライブラリには、このような処理を行うための各種の関数が用意されています。

ファイルおよびディレクトリ関係の機能のうち DOS 固有のものは、StarSuite 7 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の使用が行えなくなっています (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、StarSuite 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。

ファイル操作

ディレクトリ内のファイル検索

StarSuite Basic でディレクトリシステム内でのファイルやサブディレクトリの検索を行うには、`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、最初にヒットしたディレクトリエントリが StarSuite Basic に返されます。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。

以下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 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 関数の戻り値に返させるためです。このようなパラメータ値としては、以下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。

以下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  = Dir
      Wend
      MsgBox AllDirs
End Sub
```

VBA の場合とは異なり、StarSuite 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余分な操作が必要となる)。

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、ボリューム名といった属性に基づいた検索オプションを利用できますが、StarSuite 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 関数のパス指定において、* および ？のプレースホルダは VBA および StarSuite Basic の双方で利用できます。ただし VBA とは異なり、StarSuite 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

StarSuite 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以下のコードを実行したとします。

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

これにより、C:\SubDir1\SubDir2 ディレクトリおよび C:\SubDir1\SubDir2\SubDir3 ディレクトリが作成されます。

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\")
```

指定ディレクトリ内にサブディレクトリやファイルが存在する場合、これらも削除されます。このため、RmDir 関数を使用する際には注意が必要です。

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 StarSuite Basic の MkDir 関数および RmDir 関数では、複数階層を対象とした操作が行えます。

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。StarSuite 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。

ファイルのコピー、名前変更、削除および存在確認

ファイルのコピーは、以下のようにして実行します。

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

この場合は Source という名前のファイルを、Destination という名前でコピーします。

ファイルの名前変更は、以下のようにして実行します。

```
Name OldName As NewName
```

この場合は、OldName というファイルの名前を NewName に変更します。このようにコンマを使わずキーワード As を使用するのが、Basic 言語の基本形です。

ファイルの削除は、以下のようにして実行します。

```
Kill(Filename)
```

この場合は、Filename という名前のファイルを削除します。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。

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
```

ファイル属性の取得と変更

ファイル関連の処理を行う際には、ファイル属性の変更、最終変更日の取得、ファイルサイズの確認などの操作が必要となる場合があります。

このような情報を取得するには、たとえば以下のようなコードを記述します。

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

この場合は、指定ファイルの属性を取得しています。戻り値はビットマスクの形式で返され、その中には以下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

以下のサンプルコードでは、この情報を利用しています。

```
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
```

ここでは、`test.txt` という名前のファイルのビットマスクを取得して、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。

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、StarSuite 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。この関数は、たとえば以下のように使用します。

```
SetAttr("test.txt", 1)
```

この場合は、指定ファイルに読み取り専用の属性を設定しています。逆に、ファイルの読み取り専用属性を解除するには、以下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")      ' ファイルを最後に変更した日付と時刻を取得。
```

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")           ' ファイルサイズをバイト値で取得。
```

テキストファイルの書き込みと読み取り

StarSuite Basic には、ファイルの書き込みおよび読み取り用の各種機能が用意されています。以下の説明は、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ない)。

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の取得には、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、以下のように Open 命令を記述します。

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

ここで 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 関数で取得しておい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."
```

ここでも 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。

以下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"                    ' ファイル名の指定
FileNo = Freefile                           ' フリーのファイルハンドルの取得
```

```
Open Filename For Output As #FileNo            ' ファイルのオープン (書き込みモード)
Print #FileNo, "This is a line of text"        ' 行単位で書き込み
Print #FileNo, "This is another line of text" ' 行単位で書き込み
Close #FileNo                                  ' ファイルのクローズ
```

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込みを行う点が異なります。

また、テキストファイルの最後まで読み込んだかは、以下のようにして確認します。

```
eof(FileNo)
```

これは、エンドオブファイル (eof) に到達したかを判定します。

以下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

' ファイル名の指定
Filename = "c:\data.txt"

' フリーのファイルハンドルの取得
FileNo = Freefile

' ファイルのオープン (読み取りモード)
Open Filename For Input As FileNo

' ファイルの末尾に到達したかをチェック
Do While not eof(FileNo)

      ' 行単位で読み込み
      Line Input #FileNo, CurrentLine
      If CurrentLine <>"" then
            Msg = Msg & CurrentLine & Chr(13)
      end if

Loop

' ファイルのクローズ
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

StarSuite 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

メッセージ表示 `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。以下のサンプルコードを実行すると、最も単純な形のダイアログが表示されます。

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

この場合の `MsgBox` は、メッセージテキストとボタン OK のみを表示します。

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、以下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 - キャンセルおよびやり直し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以下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、以下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 - クエスチョン記号のアイコンを表示します。
- 48 - エクスクラメーション記号の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。

このようなパラメータ値は、以下のように指定します。

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

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)、クエスチョン記号の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。

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、以下の戻り値を利用します。

- 1 - OK
- 2 - キャンセル
- 4 - やり直し
- 5 – 無視
- 6 - はい
- 7 - いいえ

たとえば先のサンプルコードの場合、戻り値の判定は以下のようにして行えます。

```
If MsgBox ("Do you want to continue?",  292) = 6 Then
      ' はいのボタンが押された場合
Else
      ' いいえのボタンが押された場合
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, "Window 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、以下の 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   ' 警告音を鳴らします
```

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)
```

ここで 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、以下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特定の時間中断させるには、`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、以下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。この関数は、以下のように使用します。

```
Dim TempDir

TempDir=Environ ("TEMP")
```

この場合は、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ています。

第 4 章

StarSuite API について

StarSuite API とは、StarSuite 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。StarSuite 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 StarSuite 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、StarSuite の機能を拡張するオプションも用意されています。

StarSuite API の使用は、必ずしも StarSuite 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、StarSuite Basic から StarSuite 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、StarSuite 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 StarSuite 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)

StarSuite 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、StarSuite 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。

StarSuite 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。

StarSuite 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 (詳細は第 2 章を参照)。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。また `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 StarSuite 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、StarSuite 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、StarSuite 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。

属性とメソッド

StarSuite Basic の各オブジェクトからは、各種の属性とメソッドを呼び出すことができます。

属性

ここで言う属性とはオブジェクトのもつ属性と同様のもので、たとえば `Document` オブジェクトには、`Filename` や `Title` などの属性が存在します。

属性は、値を代入することにより設定されます。

```
Document.Title = "Programmierhandbuch StarSuite 6.0"
Document.Filename = "progman.sxv"
```

各属性には、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。
上記のサンプルコードの `Filename` および `Title` 属性は、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

StarSuite Basic の各オブジェクトのもつ属性の大部分は、UNO サービスとして定義されています。StarSuite Basic の属性には、このような「リアル」な属性以外にも、UNO レベルの 2 つのメソッドから成る属性が存在します。その一方は、その属性値を取得する際に、もう一方は値を指定する際に使用します (`get` および `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` と `setPosition` というメソッドが用意されています。StarSuite Basic のプログラムでこのような値にアクセスするには、`Position` 属性を使用できます。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` と `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、これは以下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。以下のサンプルコードは、このようなメソッド指定の例です。

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

この場合は、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。
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

StarSuite 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、StarSuite Basic のモジュールには、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、StarSuite のサービスであることを示す `com.sun.star` 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。つまりこの場合の完全な名前は、以下のようになります。

```
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。

各インターフェースは、複数のメソッドを組み合わせて使用します。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、メソッドをリクエストする際には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 StarSuite Basic では、このような関係は適用されません。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。

複数のサービスで使用される主要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は、本章の最後で再度説明します。

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、本マニュアルの説明以外にも、`supportsService` メソッドおよびデバッグ用の各種メソッドを利用するか、『デベロッパ向けガイド』や『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。このメソッドは、以下のような形式で使用します。

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

この場合は、`TextElement` オブジェクトが `com.sun.star.text.Paragraph` サービスをサポートしているかを確認しています。

デバッグの属性

StarSuite 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、以下の属性です。

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_supportetInterfaces - 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。

以下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_Propierties
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、StarSuite API の『API reference』に収録されています。入手先は、www.openoffice.org の以下のアドレスです。

```
http://api.openoffice.org/common/ref/com/sun/star/module-ix.html
```

主なインターフェースの概要

StarSuite のインターフェースの中には、StarSuite 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、本マニュアルの後半で説明します。ここでは単に、StarSuite 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

StarSuite 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、以下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、以下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するアクセスを行い、`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。以下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。以下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、変更、削除を行うには、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、以下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 Objects
Dim ParagraphStyles As Objects
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「NewStyle」という名前でオブジェクト ParagraphStyles に挿入しています。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)
      ' 表 (シート) の編集
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は、insertByIndex および removeByIndex というメソッドが用意されています。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。

以下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 = ParagraphElements.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。

第 5 章

StarSuite ドキュメントの操作

StarSuite 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

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

- `com.sun.star.frame.Desktop` サービスは、StarSuite のコアサービスとよく似た機能を担っています。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 StarSuite 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。
- `com.sun.star.document.OfficeDocument` サービスは、個々のドキュメントオブジェクトに対する基本的な操作を提供します。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、StarSuite の起動時に自動的にオープンされます。その際 StarSuite は、`StarDesktop` というグローバル名でアクセスされるオブジェクトを作成します。

StarDesktop のインターフェースで、もっとも重要なものが 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。StarSuite 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 StarSuite のフレームオブジェクトに対して使われています。

基本的に StarDesktop オブジェクトは、StarSuite 5 でルートオブジェクトとして扱われていた Application オブジェクトの後継者として位置づけられています。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、StarSuite 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、MS Word では Application.ActiveDocument を使用し、Excel では Application.ActiveWorkbook を使用するのに対して、StarSuite の場合は、StarDesktop がこれらの役割を果たします。StarSuite 6 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 属性を使用します。

ただし Basic IDE 上でサンプルコードを走らせる場合は StarDesktop.CurrentComponent 属性は IDE 自身を表しますので、代わりに StarSuite Basic のステートメント ThisComponent を使用してください。

StarSuite ドキュメントに関する基本知識

StarSuite ドキュメントを操作する際に、StarSuite 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、StarSuite 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

StarSuite 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することを前提に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)。このため標準的なファイル名は、以下のプレフィックスが先頭に付きます。

```
file:///
```

そして、こ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。たとえば以下のパスは、`C` ドライブ直下の `doc` ディレクトリにある `test.sxw` ファイルを示しています。

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

ローカルのファイル名を URL に変換するには、StarSuite 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、StarSuite に用意されている `ConvertFromUrl` 関数を使用します。

```
MsgBox ConvertToUrl("C:\doc\test.sxw")
            ' 結果は file:///C:/doc/test.sxw

MsgBox ConvertFromUrl("file:///C:/doc/test.sxw")
            ' 結果は c:\doc\test.sxw (Windows の場合)
```

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 の文字の使用が許可されています。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内に半角スペース記号がある場合、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

StarSuite はバージョン 6.0 より、XML ベースのファイルフォーマットを採用しました。XML の採用により、他のプログラムでファイルを開いて編集することもできるようになっています。

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なものとなります。このため StarSuite 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。詳細情報は 97 ページの 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、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 StarSuite 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 StarSuite 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、StarSuite 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()

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

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

上記のサンプルコードでは、text.sxw 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。

StarSuite 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、StarSuite API の従来バージョンにあった Documents.Add と Documents.Open メソッドに取って代わる存在です。

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、loadComponentFromURL の呼び出し時に StarSuite は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、以下の定数値を指定できます。

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

以下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.sxw"
Doc = StarDesktop.loadComponentFromURL(Url, "MyFrame", _
            SearchFlags, Dummy)

MsgBox "Press OK to display the second document."

Url = "file:///C:/test2.sxw"
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.sxw` 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.sxw` 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` メソッドの第 4 パラメータは、`PropertyValue` というデータフィールドで、これはドキュメントのオープンと作成に関する各種のオプションを StarSuite に指定する際に使用します。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` には以下のオプションを指定できます。

- **`AsTemplate`** (ブール値) - 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`** (文字列) - ドキュメントで使用する文字列コードを指定します。

- **FilterName** (文字列) - loadComponentFromURL メソッドで使用する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\registry\data\org\openoffice\office\TypeDetection.xml に定義されています。
- **FilterOptions** (文字列) - フィルタの追加オプションを指定します。
- **JumpMark** (文字列) - ドキュメントのオープン後に、JumpMark の指定位置にジャンプさせます。
- **Password** (文字列) - パスワード保護したファイル用のパスワードを与えます。
- **ReadOnly** (ブール値) - 読み取り専用ドキュメントとして開くかを指定します。

以下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 StarSuite 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 (StarOffice 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 属性は、StarSuite でオープンするファイルに対して StarSuite Calc のテキストフィルタを使用する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、StarSuite 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、以下のような形式で 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、StarSuite 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` メソッドは、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、以下の 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: ドキュメントの保存処理を担当します。
- `com.sun.star.view.XPrintable` インターフェース: ドキュメント印刷用のメソッドを提供します。

StarSuite 7 への移行に際して、大部分のドキュメントオブジェクトは、従来の機能をそのまま引き継いでいます。たとえばドキュメントの保存や印刷に関するメソッドは、現行バージョンでもドキュメントオブジェクトが提供しています。ただし、メソッドの名前やパラメータには、変更されたものがあります。

ドキュメントのエクスポートと保存

StarSuite 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、以下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 を使用します。このメソッドは com.sun.star.frame.XStorable にも用意されており、以下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.sxw"

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。これに該当するのが、以下のメソッドです。

- **hasLocation()** - ドキュメントの URL 指定が完了しているかを確認します。
- **isReadonly()** - ドキュメントが読み取り専用であるかを確認します。
- **isModified()** - 前回保存時からドキュメントが変更されているかを確認します。

ドキュメント保存用のプログラムコードを構築する際には、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、StarSuite 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、以下のオプションです。

- **CharacterSet (文字列)** - ドキュメントで使用する文字列コードを指定します。
- **FilterName (文字列)** - loadComponentFromURL メソッドで使用する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\registry\data\org\openoffice\office\TypeDetection.xml に定義されています。
- **FilterOptions (文字列)** - フィルタの追加オプションを指定します。

- **Overwrite (ブール値)** - 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 (文字列)** - パスワード保護したファイル用のパスワードを指定します。
- **Unpacked (ブール値)** - ドキュメント (非圧縮) をサブディレクトリに保存するかを指定します。

以下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

' ... Doc を 初期化

Url = "file:///c:/test3.sxw"

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。
以下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 StarSuite 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 StarSuite 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 (整数)** - 印刷する部数を指定します。
- **FileName (文字列)** - 印刷するドキュメントを指定します。
- **Collate (ブール値)** - プリンタに対して印刷の丁合に関する指定を行います。
- **Sort (ブール値)** - 複数部数を印刷する際に (CopyCount > 1) ページをソートさせるかを指定します。
- **Pages (文字列)** - 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)。

以下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 属性を使用します。この属性には、PropertyValue データフィールドを使って、以下の内容を設定します。

- **Name (文字列)** - プリンタの名前を指定します。
- **PaperOrientation (列挙型)** - 用紙方向を指定します (縦にする場合は com.sun.star.view.PaperOrientation.PORTRAIT、横にする場合は com.sun.star.view.PaperOrientation.LANDSCAPE)。
- **PaperFormat (列挙型)** - 用紙フォーマットを指定します (たとえば A4 の場合は com.sun.star.view.PaperFormat.A4、US レターの場合は com.sun.star.view.PaperFormat.Letter)。
- **PaperSize (サイズ)** - 用紙サイズを 100 分の 1 ミリ単位で指定します。

以下のサンプルコードでは、Printer 属性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       ' 20 cm に相当
PaperSize.Height = 20000      ' 20 cm に相当

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 タイプ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。同様に、PrinterProperties という名前で PropertyValue タイプの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 属性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、様々な書式設定情報を 1 つにまとめて、名前を付けたものです。StarSuite 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、StarSuite 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。StarSuite 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。

StarSuite Writer は以下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
- ナンバリングテンプレート

StarSuite Calc は以下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

StarSuite Impress は以下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

StarSuite 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。

文字の属性、第 6 章「文書ドキュメント」
`com.sun.star.style.CharacterProperties` サービス

段落の属性、第 6 章「文書ドキュメント」
`com.sun.star.text.Paragraph` サービス

セルの属性、第 7 章「表計算ドキュメント」
`com.sun.star.table.CellProperties` サービス

ページの属性、第 7 章「表計算ドキュメント」
`com.sun.star.style.PageStyle` サービス

文字要素の属性、第 7 章「表計算ドキュメント」
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、第 7 章で説明するページ属性の大部分は、StarSuite Calc だけでなく StarSuite 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、第 6 章「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。

第 6 章

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はじめ、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。最初の節では、文書ドキュメントの構造について説明し、StarSuite ドキュメント上で繰り返し行うタイプの処理を StarSuite 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。StarSuite 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、第 5 章で説明してあります。

文書ドキュメントの構造

文書ドキュメントには、以下の 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。

StarSuite Writer の StarSuite 7 API は、前バージョンから大きく変更されています。旧バージョンの API は `Selection` オブジェクトによる操作を中心に構成されており、エンドユーザー用のユーザーインターフェースを主眼に置いて、マウス操作による強調処理を重点的に扱っていました。

StarSuite 7 API では、このような関係をユーザーインターフェースとプログラマインターフェースの間に置き換えています。これにより、アプリケーションの各部に対して並列アクセスをするプログラミングが可能となり、ドキュメント内に存在する複数のオブジェクトを同時に処理することができるようになりました。また従来の `Selection` オブジェクトは、現行バージョンでは使用できなくなっています。

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、第 4 章で説明する `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 (StarSuite 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。

以下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

' ドキュメントオブジェクトの作成
Doc = StarDesktop.CurrentComponent

' Enumeration オブジェクトの作成
Enum = Doc.Text.createEnumeration

' 全テキスト要素へのアクセス用ループ
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 StarSuite 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、その結果をメッセージボックスに表示させます。

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、you、too、for の各単語をそれぞれ U、2、4 の文字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、StarSuite Basic に用意されているものではありません。これは、第 3 章の「検索と置換」でサンプルコードとして説明してある関数です。

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)、StarSuite Basic の場合は、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、Words リストに直接該当する機能は、StarSuite 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 (詳細は「TextCursor」を参照)。

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、StarSuite 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、StarSuite 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。以

下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

' 全段落へのアクセス用ループ
While Enum1.hasMoreElements
      TextElement = Enum1.nextElement
      If TextElement.supportsService("com.sun.star.text.Paragraph") Then
            Enum2 = TextElement.createEnumeration

            ' 全サブ段落へのアクセス用ループ
            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。このようにしてアクセスした段落部位に対しては、先のサンプルコードと同様に、文字列の `String` 属性を使用してテキストの置き換えを行なっています。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。StarSuite 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 StarSuite 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)。StarSuite 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、以下の 2 つの節で説明しています。

StarSuite API の従来バージョンでは、テキストの書式設定は基本的に Selection オブジェクトとその下位オブジェクトを使って行なっていました (たとえば Selection.Font、Selection.Paragraph、Selection.Border など)。新しい API では、個 のオブジェクトに書式設定用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。このような操作に使用する文字および段落関係の属性については、以下の節で説明しています。

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。StarSuite 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、StarSuite 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 (文字列)** - 選択したフォントの名前。
- **CharColor (ロング整数)** - テキストの色。
- **CharHeight (浮動小数点)** - 文字の高さのポイント数 (pt)。
- **CharUnderline (定数グループ)** - 下線の種類 (com.sun.star.awt.FontUnderline に定められた定数)。
- **CharWeight (定数グループ)** - フォントの太さ (com.sun.star.awt.FontWeight に定められた定数)。
- **CharBackColor (ロング整数)** - 背景色。
- **CharKeepTogether (ブール値)** - 自動改行の抑制。
- **CharStyleName (文字列)** – 文字テンプレートの名前。

アジア言語の文字には com.sun.style.CharacterPropertiesAsian の属性を使用します。CharacterPropertyAsian に関しては StarSuite の『API reference』を参照してください。

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、StarSuite の『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 (列挙型)** - テキストの縦方向の配置 (com.sun.star.style.ParagraphAdjust に定められた定数)。
- **ParaLineSpacing (構造体)** - 行間 (com.sun.star.style.LineSpacing に定められた構造体)。
- **ParaBackColor (ロング整数)** - 背景色。
- **ParaLeftMargin (ロング整数)** - 100 分の 1 ミリ単位で指定した左余白。
- **ParaRightMargin (ロング整数)** - 100 分の 1 ミリ単位で指定した右余白。
- **ParaTopMargin (ロング整数)** - 100 分の 1 ミリ単位で指定した上余白。
- **ParaBottomMargin (ロング整数)** - 100 分の 1 ミリ単位で指定した下余白。
- **ParaTabStops (構造体配列)** - タブの種類と位置 (Typs com.sun.star.style.TabStop に定められた構造体配列)。
- **ParaStyleName (文字列)** – 段落テンプレートの名前。

例: HTML の簡易エクスポート

以下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

' 全段落へのアクセス用ループ
While Enum1.hasMoreElements
     TextElement = Enum1.nextElement
     If TextElement.supportsService("com.sun.star.text.Paragraph") Then
          Enum2 = TextElement.createEnumeration
          CurLine = "<P>"

          ' 全段落部位へのアクセス用ループ
          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

        ' 行の出力
        CurLine = CurLine & "</P>"
        Print #FileNo, CurLine

    End If
Wend

' HTML の出力
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

文字および段落属性の書式設定としては、常にダイレクトフォーマッティングがインダイレクトフォーマッティングより優先されます。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。StarSuite 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、StarSuite 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` に定義された以下の値が使われます。

- **`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`** - 確認不可能な属性
  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。

以下のサンプルコードでは、書式設定用属性が StarSuite 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

' 全段落へのアクセス用ループ
While Enum1.hasMoreElements
       TextElement = Enum1.nextElement
       If TextElement.supportsService("com.sun.star.text.Paragraph") Then
             Enum2 = TextElement.createEnumeration

             ' 全段落部位へのアクセス用ループ
             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存在します。StarSuite 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、`TextPortion` を使うよりもこのような操作に適しています。

TextCursor

StarSuite API の `TextCursor` は、StarSuite 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 StarSuite 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。

> 警告: これらの名称については、VBA の用語と異なっている場合があるので注意が必要です。VBA の `Range` オブジェクトの機能に該当するものは、StarSuite の `TextCursor` オブジェクトであって、同じ名前を持つ StarSuite 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、StarSuite 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)。StarSuite の `TextCursor` オブジェクトに該当する機能は、以下の節で説明しています。

テキスト内の移動

StarSuite 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 オブジェクトの作成には、以下のような形式で createTextCursor 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。たとえば以下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、以下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 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 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。

以下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。以下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。StarSuite 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、StarSuite 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 (改行)。
- **SOFT_HYPHEN** – ハイフネーションの候補位置。
- **HARD_HYPHEN** – ハイフネーションを強制する位置。
- **HARD_SPACE** – 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。以下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 StarSuite 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 StarSuite 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、以下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、以下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 StarSuite 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 (ブール値)** - 逆方向に検索する指定。
- **SearchCaseSensitive (ブール値)** - 検索をする際に大文字と小文字を区別する指定。
- **SearchRegularExpression (ブール値)** - 正規表現により検索する指定。
- **SearchStyles (ブール値)** - 指定した段落テンプレートを検索する指定。
- **SearchWords (ブール値)** - ワード (単語) として検索する指定。

StarSuite Basic では StarSuite 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 StarSuite 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 (ブール値)** - 類似検索を行う指定。
- **SearchSimilarityAdd (整数)** - 類似検索での追加可能な文字数の指定。
- **SearchSimilarityExchange (整数)** - 類似検索での置換可能な文字数の指定。
- **SearchSimilarityRemove (整数)** - 類似検索での削除可能な文字数の指定。
- **SearchSimilarityRelax (ブール値)** - 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、StarSuite 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
      ' 検索処理
      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

以下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」をはじめ「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
```

StarSuite 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、StarSuite 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 StarSuite 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 および 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、StarSuite で行う通常の置換操作も、StarSuite 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。先に説明した `SearchDescriptor` 用の属性は、すべてこの `ReplaceDescriptor` でも利用できます。たとえば置換処理の場合も、大文字と小文字の区別や、類似検索の有無を指定することができます。

以下のサンプルコードは、`ReplaceDescriptors` を使用した StarSuite ドキュメント上での置換処理を示します。

```
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

StarSuite 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。

StarSuite で使用可能な正規表現の詳細情報については、StarSuite 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。

以下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)
```

文書ドキュメント: 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。

このような基本機能を扱うため、これらの StarSuite オブジェクトは、`com.sun.star.text.TextContent` という基本サービスをサポートしています。このサービスでは、以下の属性が利用できます。

- **`AnchorType` (列挙型)** - `TextContent` オブジェクトのアンカーの種類を特定します (`com.sun.star.text.TextContentAnchorType` に定められた列挙型)。
- **`AnchorTypes` (列挙型のシーケンス)** - 特殊な `TextContent` オブジェクトをサポートするすべての `AnchorTypes` の一覧。
- **`TextWrap` (列挙型)** - `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` に定められた列挙型)。

`TextContent` のオブジェクト群は、オブジェクトの作成、挿入、削除に関するメソッドなどをいくつか共有しています。

- `TextContent` オブジェクトを新規作成するには、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、以下の節で各種の例を説明します。

テーブル

以下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)
```

テーブルの作成では、テーブルオブジェクトの作成後、`initialize` による行数および列数の初期化を行い、`insertTextContent` を用いて文書ドキュメントに挿入します。

上記のサンプルコードにあるように、`insertTextContent` メソッドには `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ための `Cursor` オブジェクト。
- `Content` オブジェクトを選択範囲と置き換えるか (`True`)、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。

文書ドキュメントへのテーブルの作成と挿入は、VBA でも StarSuite Basic でもオブジェクトを利用して処理しますが、StarSuite 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するのに対し、VBA では `Range` オブジェクトがこれに対応します。また VBA でのテーブル作成と設置は `Document.Tables.Add` メソッドが処理しますが、StarSuite Basic では上記のサンプルコードのように、テーブルの作成は `createInstance` で行い、初期化後のテーブルを `insertTextContent` によりドキュメントに挿入します。

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。このような処理では、文書ドキュメントの `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)

      ' テーブルの編集処理
Next I
```

StarSuite 7 では、ドキュメントオブジェクトの `TextTables` リストを利用してテキストテーブルにアクセスできます。これは、従来 `Selection` オブジェクトで取得していたテーブルリストに置き換わるものです。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、以下の節で説明しています。

テーブルの編集

1 つのテーブルは、1 行または複数の行から構成されています。そして各行は、いくつかのセルに分割されています。厳密に表現すると、StarSuite のテーブルに列というものは存在しません。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 StarSuite には、テーブルの操作性を高めるため、列を対象とした操作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、テーブル本体の属性について説明します。これらは、`com.sun.star.text.TextTable` サービスに定義されています。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` (ロング整数)** - テーブルの背景色。
- **`BottomMargin` (ロング整数)** - 100 分の 1 ミリ単位で指定した下余白。
- **`LeftMargin` (ロング整数)** - 100 分の 1 ミリ単位で指定した左余白。
- **`RightMargin` (ロング整数)** - 100 分の 1 ミリ単位で指定した右余白。
- **`TopMargin` (ロング整数)** - 100 分の 1 ミリ単位で指定した上余白。
- **`RepeatHeadline` (ブール値)** - テーブルヘッダを各ページごとに表示させるかの指定。
- **`Width` (ロング整数)** - 100 分の 1 ミリ単位で指定したテーブル本体の幅。

テーブル行

個々のテーブルについては、その構成行をリストの形で一括取得できます。以下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
```

ここでは、`Table.getRows` を使用して、テーブル構成行のリストを取得しています。このリストの内容にアクセスするには、`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` メソッドを実行すると行オブジェクトが返されますが、これは `com.sun.star.text.TextTableRow` サービスをサポートしています。

以下に、`com.sun.star.table.XtableRows` インターフェースの主要なメソッドを示します。

- **`getByIndex(整数)`** - 指定インデックスに該当する行オブジェクトを返します。
- **`getCount()`** - 行オブジェクトの個数を返します。
- **`insertByIndex(Index, Count)`** - `Index` 位置のテーブルに Count 行を挿入します。
- **`removeByIndex(Index, Count)`** - `Index` 位置のテーブルから 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` が使用できるのは、セルを結合していないテーブルだけです。

`com.sun.star.text.TextTableRow` サービスでは、以下の属性が利用できます。

- **`BackColor (ロング整数)`** - 行の背景色。
- **`Height (ロング整数)`** - 100 分の 1 ミリ単位で指定した行の高さ。
- **`IsAutoHeight (ブール値)`** - 内容に合わせてテーブルの高さを動的に調整する指定。
- **`VertOrient (定数)`** - テキスト枠およびテーブル内のテキストの縦方向 (`com.sun.star.text.VertOrientation` に定められた値)。

テーブル列

列へのアクセスは、行の場合と同様に `getByIndex`、`getCount`、`insertByIndex`、`removeByIndex` メソッドを用いますが、この場合は `Column` オブジェクトを `getColumns` で取得してから使用します。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 StarSuite 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。

テーブルセル

StarSuite ドキュメントのテーブルのセルには、それぞれ固有の名前が存在します。StarSuite ドキュメント上のテーブルのセルにカーソルを移動すると、ステータスバーに該当する名前が表示されます。たとえば左上隅のセルは通常 `A1` と表示され、`n` 行 `X` 列のテーブルであれば右下隅のセルは `Xn` となります。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。以下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のセルにアクセスし、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。`TextCursor` オブジェクトの作成は、`createTextCursor` インターフェースを使用します。

```
CellCursor = Cell.createTextCursor()
```

個々の文字および段落に対して設定可能なすべてのオプションは、この場合も使用できます。

以下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属性を利用しています。

```
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
```

このサンプルコードでは、TextTables というオブジェクトを作成して、その中にすべてのテキストテーブルのリストを取得することで、全テーブルを対象としたループに入っています。そして個々のテーブルごとにセル名のリストを取得します。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。その際の処理としては、まずテーブルセルの内容の参照用に TextCursor オブジェクトを作成してから、テーブルセルの段落属性を使用して、必要な書式を設定しています。

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。

StarSuite のテキスト枠は、MS Word のテキストボックス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 createInstance メソッドおよび TextCursor 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挙動を制御するために、各種の属性が用意されています。これら属性の大部分は 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。以下に主要な属性を示します。

- **BackColor (ロング整数)** - テキスト枠の背景色。
- **BottomMargin (ロング整数)** - 100 分の 1 ミリ単位で指定した下余白。
- **LeftMargin (ロング整数)** - 100 分の 1 ミリ単位で指定した左余白。
- **RightMargin (ロング整数)** - 100 分の 1 ミリ単位で指定した右余白。
- **TopMargin (ロング整数)** - 100 分の 1 ミリ単位で指定した上余白。
- **Height (ロング整数)** - 100 分の 1 ミリ単位で指定したテキスト枠の高さ。
- **Width (ロング整数)** - 100 分の 1 ミリ単位で指定したテキスト枠の幅。
- **HoriOrient (定数)** - テキスト枠の横方向の向き (com.sun.star.text.HoriOrientation に定められた値)。
- **VertOrient (定数)** - テキスト枠の縦方向の向き (com.sun.star.text.VertOrientation に定められた値)。

以下のサンプルコードでは、これらの属性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) 属性と `VertOrient` (`BaseFrameProperties` サービス) 属性との関係には注意が必要です。ここでは、`AnchorType` に `AS_CHARACTER` という値を指定しています。これは、テキスト枠をテキストフロー内に直接挿入して、通常の文字として振る舞う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` 属性に指定した `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書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。

テキストフィールド

テキストフィールドは、通常のテキストの機能を拡張したものであるため、これらも TextContent オブジェクトとして扱われます。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 属性への 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、StarSuite 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。

StarSuite 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`)。

StarSuite 7 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` を使っ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、以下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。以下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
```

すべてのテキストフィールドにアクセスするには、ドキュメントオブジェクト上に存在するフィールドのリストを取得する必要があります。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) またはコメント (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属性を示します。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 (StarSuite 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る)。

ページ数、語数、文字数

テキストフィールド

- 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。これらは以下の属性をサポートしています。

- **NumberingType (定数)** - 数の書式 (com.sun.star.style.NumberingType に定められた値)

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、以下の属性が利用できます。

- **NumberingType (定数)** - 数の書式 (com.sun.star.style.NumberingType に定められた値)。
- **Offset (整数)** - ページ番号に追加するオフセット値 (負の値も指定可能)。

以下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。StarSuite 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。

`FooterIsOn` 属性に `True` を指定しているのは、フッタ行を表示させるための処理です。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、文書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、以下の属性が利用できます。

- **Author (文字列)** - 作成者の名前。
- **Content (文字列)** - コメントの文章。
- **Date (日付)** - コメントを記入した日付。

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、以下の属性が利用できます。

- **IsFixed (ブール値)** - True を指定すると、表示時間を更新せず、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 (ブール値)** - True を指定するとフィールドに現在の日付を表示し、それ以外の場合は現在の時刻を表示する。
- **DateTimeValue (構造体)** - フィールドの現在の内容 (com.sun.star.util.DateTime 構造体)。
- **NumberFormat (定数)** - 日付および時刻の表示書式。

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 を使用します。表示形式は、以下の 2 つの属性で指定します。

- **ChapterFormat (定数)** - 章名と章番号のどちらを表示するかを指定します (com.sun.star.text.ChapterFormat に定められた値)。
- **Level (整数)** - 章名または章番号として表示させる章レベルを指定します。指定値は 0 が最高レベルに該当します。

テキストマーク

テキストマーク (com.sun.star.text.Bookmark) も TextContent オブジェクトとして扱われます。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、テキスト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、実際のテキストマークオブジェクト (Bookmark) を作成しています。このテキストマークは、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。

テキスト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。

以下のサンプルコードでは、文書ドキュメント上で特定のテキストマークを選択して、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、テキスト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、このテキスト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。

第 7 章

表計算ドキュメント

StarSuite 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。本章では、表計算ドキュメントの操作に必要なサービス、メソッド、属性について説明します。

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 (スプレッドシート) 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。この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 StarSuite Basic では、表計算ドキュメント関係の用語に違いがあります。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、StarSuite Basic の場合は表計算ドキュメントまたはシートと呼んでいます。

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。

以下の 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、以下の属性を利用できます。

- **`IsVisible` (ブール値)** - スプレッドシートの表示状態を指定します。
- **`PageStyle` (文字列)** - スプレッドシートのページテンプレート名を指定します。

シートの作成、削除、名前の変更

表計算ドキュメントの 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。以下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。その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、createInstance 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
```

第 4 章で説明したように、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。

表の行と列

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 com.sun.star.table.TableColumns または com.sun.star.table.TableRows サービスをサポートしており、アクセスする際には表計算オブジェクトの Rows および Columns 属性を使用します。

以下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て、それぞれに第 1 列および第 1 行の参照情報を格納します。

```
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.TableRow サービスをサポートしており、以下の属性を利用できます。

- **Width (ロング整数)** - 100 分の 1 ミリ単位で指定した列幅。
- **OptimalWidth (ブール値)** - 列の幅を最適化する指定。
- **IsVisible (ブール値)** - 列の表示状態の指定。
- **IsStartOfNewPage (ブール値)** - 印刷時に該当列の前で改ページをする指定。

列の幅は、OptimalWidth 属性に 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 は属性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、以下の属性を利用できます。

- **Height (ロング整数)** - 100 分の 1 ミリ単位で指定した行の高さ。
- **OptimalHeight (ブール値)** - 行の高さを最適化する指定。
- **IsVisible (ブール値)** - 行の表示状態の指定。
- **IsStartOfNewPage (ブール値)** - 印刷時に該当行の前で改ページをする指定。

OptimalHeight 属性に 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。このような自動最適化は、Height 属性で行の高さを指定すると解除されます。

以下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
```

StarSuite Basic では、Rows および Columns のリストに対して、インデックス指定によるアクセスが行えます。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

各シート内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。

このような列の処理方法は、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

個々のスプレッドシー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。

以下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"
```

セルの参照は、このような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、たとえばスプレッドシートの左上隅のセル (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。

StarSuite 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、StarSuite 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、以下のいずれかです。

- **EMPTY** - 空白
- **VALUE** - 数値
- **TEXT** - 文字列
- **FORMULA** - 数式

セルの挿入、削除、コピー、移動

StarSuite 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、スプレッドシー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、セルをスプレッドシー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、以下の値を設定できます。

- **`Sheet` (整数)** - シート番号 (開始値は 0)。
- **`StartColumn` (ロング整数)** - セル範囲の先頭列 (開始値は 0)。
- **`StartRow` (ロング整数)** - セル範囲の先頭行 (開始値は 0)。
- **`EndColumn` (ロング整数)** - セル範囲の末尾列 (開始値は 0)。
- **`EndRow` (ロング整数)** - セル範囲の末尾行 (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、以下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、以下の 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。以下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、以下の値を設定できます。

- **Sheet (整数)** - スプレッドシート番号 (開始値は 0)。
- **Column (ロング整数)** - 移動先の列位置 (開始値は 0)。
- **Row (ロング整数)** - 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、moveRange 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、StarSuite 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、StarSuite Basic のメソッドは Sheet オブジェクトを対象とします。

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、各種の項目が存在します。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第 6 章「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、以下の節で説明します。

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。

StarSuite 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、以下の属性が用意されています

- **CellBackColor (ロング整数)** - セルの背景色を指定します。
- **IsCellBackgroundTransparent (ブール値)** - 背景色を透明にするよう指定します。
- **ShadowFormat (構造体)** - セルに付ける影を指定します (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、以下のような関係になっています。

- **Location (列挙型)** - 影の位置 (com.sun.star.table.ShadowLocation 構造体の値)。
- **ShadowWidth (整数)** - 100 分の 1 ミリ単位で指定した影の幅。
- **IsTransparent (ブール値)** - 影を透明にする指定。
- **Color (ロング整数)** - 影の色。

以下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

StarSuite には、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。

以下の属性は、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 (列挙型)** - テキストの水平方向の配置指定 (com.sun.star.table.CellHoriJustify に定められた値)。

- **VertJustify (列挙型)** - テキストの垂直方向の配置指定 (com.sun.star.table.CellVertJustify に定められた値)。
- **Orientation (列挙型)** - テキストの表示方向の指定 (com.sun.star.table.CellOrientation に定められた値)。
- **IsTextWrapped (ブール値)** - セル内でテキストを自動改行させるかの指定。
- **RotateAngle (ロング整数)** - 100 分の 1 度単位で指定したテキストの回転角。

以下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 文字ずつ上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、テキストの表示書式

StarSuite には、日付および時刻に対する各種の表示書式が用意されています。これらの表示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。StarSuite では、queryKey および addNew メソッドを利用して、既存の数の書式だけでなく、数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、以下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、StarSuite 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。

以下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
```

セルの書式設定用オプションについては、StarSuite Calc 上の通常の操作で書式設定を行う際に用いる、**セルの書式設定**用のダイアログも参照してください。

ページ属性

ページ属性では、ドキュメントの各ページの表示内容に対する書式設定および、全ページに共通して配置する項目の設定などを行います。具体的には、以下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。このような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。
StarSuite 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以下の属性が用意されています。

- **`BackColor` (ロング整数)** - 背景色。
- **`BackGraphicURL` (文字列)** - 背景に表示させる画像の URL。
- **`BackGraphicFilter` (文字列)** - 背景用の画像に対するフィルタ名。
- **`BackGraphicLocation` (列挙型)** - 背景用の画像の位置 (`com.sun.star.style.GraphicLocation` に定められた値)。
- **`BackTransparent` (ブール値)** - 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、`com.sun.star.style.PageProperties` サービスを使用します。

- **`IsLandscape` (ブール値)** - 用紙方向を横にする指定。
- **`Width` (ロング整数)** - 100 分の 1 ミリ単位で指定したページの幅。
- **`Height` (ロング整数)** - 100 分の 1 ミリ単位で指定したページの高さ。
- **`PrinterPaperTray` (文字列)** - 使用する用紙トレイの名前。

以下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) の横方向に設定します。

```
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、以下の属性が用意されています。

- **LeftMargin (ロング整数)** - 100 分の 1 ミリ単位で指定したページの左余白。
- **RightMargin (ロング整数)** - 100 分の 1 ミリ単位で指定したページの右余白。
- **TopMargin (ロング整数)** - 100 分の 1 ミリ単位で指定したページの上余白。
- **BottomMargin (ロング整数)** - 100 分の 1 ミリ単位で指定したページの下余白。
- **LeftBorder (構造体)** - 左側のページ外枠線の指定 (com.sun.star.table.BorderLine に定められた構造体)。
- **RightBorder (構造体)** - 右側のページ外枠線の指定 (com.sun.star.table.BorderLine に定められた構造体)。
- **TopBorder (構造体)** - 上側のページ外枠線の指定 (com.sun.star.table.BorderLine に定められた構造体)。
- **BottomBorder (構造体)** - 下側のページ外枠線の指定 (com.sun.star.table.BorderLine に定められた構造体)。
- **LeftBorderDistance (ロング整数)** - 100 分の 1 ミリ単位で指定した、左側の外枠線とページ本文との距離。
- **RightBorderDistance (ロング整数)** - 100 分の 1 ミリ単位で指定した、右側の外枠線とページ本文との距離。
- **TopBorderDistance (ロング整数)** - 100 分の 1 ミリ単位で指定した、上側の外枠線とページ本文との距離。
- **BottomBorderDistance (ロング整数)** - 100 分の 1 ミリ単位で指定した、下側の外枠線とページ本文との距離。
- **ShadowFormat (構造体)** - ページ本文領域に対する影付き表示の指定 (com.sun.star.table.ShadowFormat に定められた構造体)。

以下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

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、以下の属性を使用します。

- **`HeaderIsOn` (ブール値)** - ヘッダを表示する指定。
- **`HeaderLeftMargin` (ロング整数)** - 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` (ロング整数)** - 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` (ロング整数)** - 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` (ロング整数)** - 100 分の 1 ミリ単位で指定した、ヘッダの高さ。
- **`HeaderIsDynamicHeight` (ブール値)** - ヘッダの高さを表示内容に自動的に合わせる指定。
- **`HeaderLeftBorder` (構造体)** - ヘッダの左側の外枠線に関する指定 (`com.sun.star.table.BorderLine` に定められた構造体)。
- **`HeaderRightBorder` (構造体)** - ヘッダの右側の外枠線に関する指定 (`com.sun.star.table.BorderLine` に定められた構造体)。
- **`HeaderTopBorder` (構造体)** - ヘッダの上側の外枠線に関する指定 (`com.sun.star.table.BorderLine` に定められた構造体)。
- **`HeaderBottomBorder` (構造体)** - ヘッダの下側の外枠線に関する指定 (`com.sun.star.table.BorderLine` に定められた構造体)。
- **`HeaderLeftBorderDistance` (ロング整数)** - 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` (ロング整数)** - 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` (ロング整数)** - 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` (ロング整数)** - 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` (ブール値)** - 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` (ロング整数)** - ヘッダの背景色。
- **`HeaderBackGraphicURL` (文字列)** - 背景に表示する画像の URL。
- **`HeaderBackGraphicFilter` (文字列)** - ヘッダの背景画像に対するフィルタ名。
- **`HeaderBackGraphicLocation` (列挙型)** - ヘッダの背景画像の位置 (`com.sun.star.style.GraphicLocation` に定められた値)。

- **HeaderBackTransparent (ブール値)** - ヘッダの背景を透明にする指定。
- **HeaderShadowFormat (構造体)** - ヘッダに付ける影の指定 (com.sun.star.table.ShadowFormat に定められた構造体)。

フッタの書式設定には、以下の属性を使用します。

- **FooterIsOn (ブール値)** - フッタを表示する指定。
- **FooterLeftMargin (ロング整数)** - 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 (ロング整数)** - 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 (ロング整数)** - 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 (ロング整数)** - 100 分の 1 ミリ単位で指定した、フッタの高さ。
- **FooterIsDynamicHeight (ブール値)** - フッタの高さを表示内容に自動的に合わせる指定。
- **FooterLeftBorder (構造体)** - フッタの左側の外枠線に関する指定 (com.sun.star.table.BorderLine に定められた構造体)。
- **FooterRightBorder (構造体)** - フッタの右側の外枠線に関する指定 (com.sun.star.table.BorderLine に定められた構造体)。
- **FooterTopBorder (構造体)** - フッタの上側の外枠線に関する指定 (com.sun.star.table.BorderLine に定められた構造体)。
- **FooterBottomBorder (構造体)** - フッタの下側の外枠線に関する指定 (com.sun.star.table.BorderLine に定められた構造体)。
- **FooterLeftBorderDistance (ロング整数)** - 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 (ロング整数)** - 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 (ロング整数)** - 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 (ロング整数)** - 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 (ブール値)** - 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 (ロング整数)** - フッタの背景色。
- **FooterBackGraphicURL (文字列)** - 背景に表示する画像の URL。
- **FooterBackGraphicFilter (文字列)** - フッタの背景画像に対するフィルタ名。
- **FooterBackGraphicLocation (列挙型)** - フッタの背景画像の位置 (com.sun.star.style.GraphicLocation に定められた値)。

- **FooterBackTransparent (ブール値)** - フッタの背景を透明にする指定。
- **FooterShadowFormat (構造体)** - フッタに付ける影の指定 (com.sun.star.table.ShadowFormat に定められた構造体)

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、以下の属性を使用します。

- **`LeftPageHeaderContent (オブジェクト)`** - 偶数ページのヘッダの内容 (`com.sun.star.sheet.HeaderFooterContent` サービス)。
- **`RightPageHeaderContent (オブジェクト)`** - 奇数ページのヘッダの内容 (`com.sun.star.sheet.HeaderFooterContent` サービス)。
- **`LeftPageFooterContent (オブジェクト)`** - 偶数ページのフッタの内容 (`com.sun.star.sheet.HeaderFooterContent` サービス)。
- **`RightPageFooterContent (オブジェクト)`** - 奇数ページのフッタの内容 (`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。このサービスは、`LeftText`、`CenterText`、`RightText` という (疑似) 属性を用いて、StarSuite Calc のヘッダやフッタに関する 3 種類のテキスト情報を取得します。

以下のサンプルコードでは、「標準」テンプレートのヘッダの左側のテキストフィールドに「Just a Test.」という文字列を記入します。

```
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
```

このサンプルコードの最終行に注意してください。変更したテキストを有効にするには、テキストの変更後に `TextContent` オブジェクトを再度割り当てる必要があります。

文書ドキュメント (StarSuite 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。以下の属性は、`com.sun.star.style.PageProperties` サービスに定義されています。

- **HeaderText (オブジェクト)** - ヘッダの内容を示すテキストオブジェクト (`com.sun.star.text.XText` サービス)。
- **HeaderTextLeft (オブジェクト)** - 左ページのヘッダの内容を示すテキストオブジェクト (`com.sun.star.text.XText` サービス)。
- **HeaderTextRight (オブジェクト)** - 右ページのヘッダの内容を示すテキストオブジェクト (`com.sun.star.text.XText` サービス)。
- **FooterText (オブジェクト)** - フッタの内容を示すテキストオブジェクト (`com.sun.star.text.XText` サービス)。
- **FooterTextLeft (オブジェクト)** - 左ページのフッタの内容を示すテキストオブジェクト (`com.sun.star.text.XText` サービス)。
- **FooterTextRight (オブジェクト)** - 右ページのフッタの内容を示すテキストオブジェクト (`com.sun.star.text.XText` サービス)。

以下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、その表示テキストを「Just a Test.」としています。

```
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、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、StarSuite 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、以下の属性を利用できます。

- **CenterHorizontally (ブール値)** - 表セルの水平方向の配置を、中央揃えにする指定。
- **CenterVertically (ブール値)** - 表セルの垂直方向の配置を、中央揃えにする指定。

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以下の属性が用意されています。

- **`PrintAnnotations` (ブール値)** - セルのコメントを印刷する指定。
- **`PrintGrid` (ブール値)** - セルの罫線を印刷する指定。
- **`PrintHeaders` (ブール値)** - 行および列のヘッダを印刷する指定。
- **`PrintCharts` (ブール値)** - シート上のグラフを印刷する指定。
- **`PrintObjects` (ブール値)** - 埋め込みオブジェクトを印刷する指定。
- **`PrintDrawing` (ブール値)** - 図形描画オブジェクトを印刷する指定。
- **`PrintDownFirst` (ブール値)** - 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` (ブール値)** - セルの計算結果ではなく、数式を印刷する指定。
- **`PrintZeroValues` (ブール値)** - ゼロ値を印刷する指定。

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

StarSuite 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が (0, 0) として扱われます。以下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「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。このような定数は、`com.sun.star.sheet.GeneralFunction` に定められています。指定可能な定数値は以下のものです。

- **`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

以下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、セル範囲内の特定のタイプの内容だけを消去することができます。

以下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)
```

ここで `clearContents` で削除するデータの種類は、`com.sun.star.sheet.CellFlags` に定められている定数値を、フラグとして指定します。このような定数値としては、以下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** - 計算式
- **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** - セル内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に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。以下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。

第 8 章

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。

図形描画ドキュメントの作成、オープン、保存については、第 5 章「StarSuite ドキュメントの操作」を参照してください。

図形描画ドキュメントの構造

StarSuite 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。

ただしレイヤーという概念があるため、これらの関係は多少複雑です。標準の図形描画ドキュメントには、レイアウト、コントロール、寸法線というレイヤーが用意されており、すべての図形描画要素はレイアウトレイヤーに配置します。これらの他にも、ユーザーが新規レイヤーを追加することも可能です。図形描画用のレイヤーの詳細については、StarSuite の『デベロッパ向けガイド』を参照してください。

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、以下の 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 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、以下の属性を使用できます。

- **`BorderLeft` (ロング整数)** - 100 分の 1 ミリ単位で指定した、左側の外枠線。
- **`BorderRight` (ロング整数)** - 100 分の 1 ミリ単位で指定した、右側の外枠線。
- **`BorderTop` (ロング整数)** - 100 分の 1 ミリ単位で指定した、上側の外枠線。
- **`BorderBottom` (ロング整数)** - 100 分の 1 ミリ単位で指定した、下側の外枠線。
- **`Width` (ロング整数)** - 100 分の 1 ミリ単位で指定した、ページ幅。
- **`Height` (ロング整数)** - 100 分の 1 ミリ単位で指定した、ページの高さ。
- **`Number` (整数)** - ページ数 (開始値は 1)。読み取り専用。
- **`Orientation` (列挙型)** - ページの方向 (`com.sun.star.view.PaperOrientation` に定められた値)。

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。以下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、ページ余白を 0.5 センチに設定します。

```
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、StarSuite 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。

以下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 つのサービスについて説明し、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。

StarSuite 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、主として以下の属性を使用します。

- **`FillColor` (ロング整数)** - 領域の塗りつぶし色。

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。

以下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、StarSuite 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、以下のオプションを使用できます。

- **`Style` (列挙型)** - 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` に定められた値)。
- **`StartColor` (ロング整数)** - グラデーションの開始色。
- **`EndColor` (ロング整数)** - グラデーションの終了色。
- **`Angle` (整数)** - 10 分の 1 度単位で指定したグラデーションの角度。
- **`XOffset` (整数)** - 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` (整数)** - 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` (整数)** - パーセント単位で指定した `StartColor` の強度 (StarSuite Basic では 100 パーセント以上の値も指定可能)。
- **`EndIntensity` (整数)** - パーセント単位で指定した `EndColor` の強度 (StarSuite Basic では 100 パーセント以上の値も指定可能)。
- **`StepCount` (整数)** - StarSuite に計算させるグラデーションのステップ数。

以下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、左上から右下の方向に施しています。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。このようなハッチング用の構造体には、以下の属性を指定できます。

- **Style (列挙型)** - 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 に定められた値)。
- **Color (ロング整数)** - 線の表示色。
- **Distance (ロング整数)** - 100 分の 1 ミリ単位で指定した、線の間隔。
- **Angle (整数)** - 10 分の 1 度単位で指定したハッチングの角度。

以下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塗りつぶしを行うには、`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 StarSuite 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。

以下のサンプルコードでは、四角形を描画して、StarSuite 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

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。

以下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、以下のものがあります。

- **`LineStyle` (列挙型)** - 線の種類 (`com.sun.star.drawing.LineStyle` に定められた値)。
- **`LineColor` (ロング整数)** - 線の色。

- **LineTransparence (整数)** - 線の透過性。
- **LineWidth (ロング整数)** - 100 分の 1 ミリ単位で指定した線の幅。
- **LineJoint (列挙型)** - 線の接続する点の形状 (com.sun.star.drawing.LineJoint に定められた値)

以下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、StarSuite 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)

図形描画オブジェクト内のテキストの書式設定を行うには、com.sun.star.style.CharacterProperties および com.sun.star.style.ParagraphProperties サービスを使用します。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、詳細については第 6 章「文書ドキュメント」で説明してあります。

以下のサンプルコードでは、描画した四角形に挿入するテキストに対し、com.sun.star.style.CharacterProperties サービスを用いて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。

オブジェクト内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、以下のものがあります。

- **TextAutoGrowHeight (ブール値)** - 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 (ブール値)** - 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 (列挙型)** - 図形描画要素内のテキストの水平方向の表示位置 (`com.sun.star.drawing.TextHorizontalAdjust` に定められた値)
- **TextVerticalAdjust (列挙型)** - 図形描画要素内のテキストの垂直方向の表示位置 (`com.sun.star.drawing.TextVerticalAdjust` に定められた値)
- **TextLeftDistance (ロング整数)** - 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 (ロング整数)** - 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 (ロング整数)** - 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 (ロング整数)** - 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。

以下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" ' 事前に Page.add の実行が必要

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

図形描画オブジェクトの多くには、`com.sun.star.drawing.ShadowProperties` サービスを用いて、影を表示することができます。このサービスでは、以下の属性を利用できます。

- **Shadow (ブール値)** - 影を表示させる指定。
- **ShadowColor (ロング整数)** - 影の色の指定。
- **ShadowTransparence (整数)** - 影の透過性の指定。
- **ShadowXDistance (ロング整数)** - 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 (ロング整数)** - 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。

以下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、以下の表示設定用サービスをサポートしています。

- **塗りつぶし関係の属性** - `com.sun.star.drawing.FillProperties`
- **線関係の属性** - `com.sun.star.drawing.LineProperties`
- **テキスト関係の属性** - `com.sun.star.drawing.Text` (`com.sun.star.style.CharacterProperties` および `com.sun.star.style.ParagraphProperties` と併用)

- **影関係の属性** - com.sun.star.drawing.ShadowProperties
- **CornerRadius (ロング整数)** - 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、以下のサービスをサポートしています。

- **塗りつぶし関係の属性** - com.sun.star.drawing.FillProperties
- **線関係の属性** - com.sun.star.drawing.LineProperties
- **テキスト関係の属性** - com.sun.star.drawing.Text (com.sun.star.style.CharacterProperties および com.sun.star.style.ParagraphProperties と併用)
- **影関係の属性** - com.sun.star.drawing.ShadowProperties

円と楕円オブジェクトには、これらの他に以下の 3 つの属性が用意されています。

- **CircleKind (列挙型)** - 円および楕円の種類 (com.sun.star.drawing.CircleKind に定められた値)。
- **CircleStartAngle (ロング整数)** - 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 (ロング整数)** - 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。指定可能な値は以下のものです。

- **com.sun.star.drawing.CircleKind.FULL** - 円または楕円の全体
- **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** - 扇型
- **com.sun.star.drawing.CircleKind.ARC** - 円弧 (円の外周線の部分のみ)

以下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)
```

線オブジェクト

StarSuite には線オブジェクトの表示設定用に、`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以下のものです。

- **線関係の属性** - `com.sun.star.drawing.LineProperties`
- **テキスト関係の属性** - `com.sun.star.drawing.Text` (`com.sun.star.style.CharacterProperties` および `com.sun.star.style.ParagraphProperties` と併用)
- **影関係の属性** - `com.sun.star.drawing.ShadowProperties`

以下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` 属性の座標により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

StarSuite には、複雑な多角形を描画するための com.sun.star.drawing.PolyPolygonShape サービスも用意されています。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。

- **塗りつぶし関係の属性** - com.sun.star.drawing.FillProperties
- **線関係の属性** - com.sun.star.drawing.LineProperties
- テキスト関係の属性 - com.sun.star.drawing.Text (com.sun.star.style.CharacterProperties および com.sun.star.style.ParagraphProperties と併用)
- **影関係の属性** - com.sun.star.drawing.ShadowProperties

また PolyPolygonShape サービスには、多角形の座標指定用に、以下の属性も用意されています。

- **PolyPolygon (配列)** - 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)

以下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)      ' 座標の指定前に Page.add の実行が必要

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)      ' 座標の指定前に Page.add の実行が必要

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、StarSuite 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、このような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、StarSuite 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、以下の 2 つの表示設定用属性をサポートしています。

- **テキスト関係の属性** - com.sun.star.drawing.Text (com.sun.star.style.CharacterProperties および com.sun.star.style.ParagraphProperties と併用)
- **影関係の属性** - 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、以下のものがあります。

- **GraphicURL (文字列)** - 画像の URL
- **AdjustLuminance (整数)** - 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 (整数)** - 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 (整数)** - 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 (整数)** - 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 (整数)** - 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 (整数)** - 画像のガンマ値
- **Transparency (整数)** – パーセント単位で指定した、画像の透過性
- **GraphicColorMode (列挙型)** - 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 に定められた値)

以下のサンプルコードは、ページへの図オブジェクトの挿入方法の例です。

```
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                          ' これは単なる表示位置の指定
                                        ' 画像の表示サイズは以下の座標で指定
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。

以下のサンプルコードでは、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
' 四角形の図形描画オブジェクトを作成
Square = Doc.createInstance("com.sun.star.drawing.RectangleShape")
Square.Size = Size
Square.Position = Point
Square.FillColor = RGB(255,128,128)
Page.add(Square)
' 円の図形描画オブジェクトを作成
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)
' 円と四角形のオブジェクトのグループ化
Shapes = createUnoService("com.sun.star.drawing.ShapeCollection")
Shapes.add(Square)
Shapes.add(Circle)
Group = Page.group(Shapes)
' グループ化した図形描画オブジェクトを作成
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、以下の属性が用意されています。

- **RotateAngle (ロング整数)** - 100 分の 1 度単位で指定した回転角
- **ShearAngle (ロング整数)** - 100 分の 1 度単位で指定した傾斜角

以下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)
```

以下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、第 6 章「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。以下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

StarSuite のプレゼンテーションドキュメントは、図形描画ドキュメントをベースに構成されています。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。図形描画ドキュメント内のページへのアクセスには、`DrawPages` リストが利用できましたが、このようなプレゼンテーション内のページへのアクセスも同様の手順で行えます。プレゼンテーションドキュメントの処理に利用するのは `com.sun.star.presentation.PresentationDocument` サービスと呼ばれるものですが、これは `com.sun.star.drawing.DrawingDocument` サービスのすべても提供しています。

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` 属性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というものが存在し、プレゼンテーションで利用する主要な属性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スタートメソッドは、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。そしてこのオブジェクトに対して `start()` メソッドを実行することで、プレゼンテーションの連続実行を開始させています。

プレゼンテーションオブジェクトには、以下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`** - 先頭ページからプレゼンテーションを開始し、ページ切り替えのタイミング設定を行います。

また以下の属性を指定することもできます。

- **`AllowAnimations` (ブール値)** - プレゼンテーション内のアニメーションを実行する指定。
- **`CustomShow` (文字列)** - 目的別スライドショーで実行するプレゼンテーション名の指定。
- **`FirstPage` (文字列)** - プレゼンテーションを開始するページ名の指定。
- **`IsAlwaysOnTop` (ブール値)** - 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する指定。
- **`IsAutomatic` (ブール値)** - プレゼンテーションを自動的に実行する指定。
- **`IsEndless` (ブール値)** - プレゼンテーション終了時に、最初から再実行する指定。
- **`IsFullScreen` (ブール値)** - フルスクリーンでプレゼンテーションを自動的に実行する指定。

- **IsMouseVisible (ブール値)** - プレゼンテーション実行中にマウスポインタを表示する指定。
- **Pause (ロング整数)** - プレゼンテーション終了時に使用するブラック画面の表示時間の指定。
- **StartWithNavigator (ブール値)** - 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する指定。
- **UsePn (ブール値)** - プレゼンテーション実行中に、書き込み用のペンポインタを表示する指定。

第 9 章

グラフ (ダイアグラム)

StarSuite 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、各種の表示が可能です。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。

表計算ドキュメント上に用意されたデータに関しては、個々の値と動的にリンクしたグラフを作成できます。つまりこの場合、データの値に変更が加えられると、その結果がただちにグラフに反映されます。本章では、StarSuite に用意されたグラフモジュール用のプログラミングインターフェースの概要を説明し、表計算ドキュメント上でのグラフ操作に焦点を当てます。

表計算ドキュメントでのグラフ操作

StarSuite 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。

文書ドキュメントや図形描画ドキュメントの場合、グラフは他のコンテンツから独立した存在として扱われますが、表計算ドキュメントのグラフは、ドキュメント上のデータとリンクさせてグラフを作成できます。以下のサンプルコードは、このような表計算ドキュメントとグラフの関係を説明しています。

```
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

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)。3 つ目は addNewByName メソッドを用いて、この既存グラフのリストに新規グラフを登録する行です。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。

最終行の addNewByName メソッドのパラメータでは、Rect および RangeAddress という変数を用いて、グラフ描画に必要な情報を指定しています。この Rect には、表計算ドキュメント上でのグラフの表示位置を設定します。同じく RangeAddress には、グラフのデータ範囲を設定します。

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、以下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。

Excel の場合、ドキュメント内のワークシート (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 StarSuite Basic の場合、StarSuite 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、その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 Z 軸に関係するメソッドや属性は、3D グラフでのみ使用可能で、2D グラフでは利用できません。同じく円グラフには、グラフ軸に関するインターフェースは用意されていません。

グラフの構成要素

タイトル、サブタイトル、凡例

グラフを構成する基本的な要素としては、タイトル、サブタイトル、凡例が挙げられます。個々のグラフには、これらの要素に対応するオブジェクトが存在します。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、以下の属性が用意されています。

- **`HasMainTitle` (ブール値)** - タイトルを表示する指定。
- **`Title` (オブジェクト)** - グラフのタイトルに関する情報を収めたオブジェクト (`com.sun.star.chart.ChartTitle` サービスをサポート)。
- **`HasSubTitle`(ブール値)** - サブタイトルを表示する指定。
- **`Subtitle` (オブジェクト)** - グラフのサブタイトルに関する情報を収めたオブジェクト (`com.sun.star.chart.ChartTitle` サービスをサポート)。
- **`HasLegend` (ブール値)** - 凡例を表示する指定。
- **`Legend` (オブジェクト)** - グラフの凡例表示に関する情報を収めたオブジェクト (`com.sun.star.chart.ChartLegendPosition` サービスをサポート)。

ここで取り上げたグラフ要素には、図形描画要素と共通する部分が多数あります。これは、`com.sun.star.chart.ChartTitle` サービスおよび `com.sun.star.chart.ChartLegendPosition` サービスが、図形描画要素の根幹を成す `com.sun.star.drawing.Shape` サービスをサポートしているためです。

そのため、これらの要素についても、`Size` および `Position` 属性を用いて位置やサイズを確認することができます。

また要素の表示設定についても、塗りつぶし属性や線属性 (com.sun.star.drawing.FillProperties および com.sun.star.drawing.LineStyle サービス) および文字属性 (com.sun.star.style.CharacterProperties サービス) を使用できます。

com.sun.star.chart.ChartTitle サービスには、先に説明した属性の他に、以下の 2 つの属性も用意されています。

- **TextRotation (ロング整数)** - 100 分の 1 度単位で指定したテキストの回転角。
- **String (文字列)** - タイトルまたはサブタイトルとして表示するテキスト。

グラフの凡例 (com.sun.star.chart.ChartLegend サービス) に関しては、以下の属性も用意されています。

- **Alignment (列挙型)** - 凡例の表示位置の指定 (com.sun.star.chart.ChartLegendPosition に定められた値)。

以下のサンプルコードでは、グラフを作成して、そのタイトルを「Test」、サブタイトルを「Test 2」とし、凡例表示の設定を行なっています。凡例表示については、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 = "Test"

Chart.HasSubTitle = True
Chart.Subtitle.String = "Test 2"

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。この領域にも対応するオブジェクトが存在し、このようなグラフオブジェクトへのアクセスには以下の属性を使用します。

- **Area (オブジェクト)** - グラフ背景の表示エリア (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。この部分を扱う com.sun.star.chart.ChartArea サービスは、線属性と塗りつぶし属性をサポートしていますが、その他の属性は特に使用しません。

グラフの壁面と床面

グラフの背景がグラフの全域を指すのに対して、グラフの壁面は、データの表示領域のみを対象とします。

通常 3D グラフに関しては、グラフの壁面が 2 つあります。その 1 つはデータの表示領域の背面に表示され、もう 1 つは Y 軸側に表示されます。また 3D グラフには、床面も表示されます。

- **Floor (オブジェクト)** - グラフの床面 (3D グラフのみ。com.sun.star.chart.ChartArea サービスをサポート)。
- **Wall (オブジェクト)** - グラフの壁面 (3D グラフのみ。com.sun.star.chart.ChartArea サービスをサポート)。

これらのオブジェクトは com.sun.star.chart.ChartArea サービスをサポートしているので、線属性および塗りつぶし属性を利用できます (com.sun.star.drawing.FillProperties および com.sun.star.drawing.LineStyle に関しては第 8 章を参照してください)。

壁面と床面は Chart オブジェクトの一部であり、これらのアクセスにも Chart オブジェクトを使用します。

```
Chart.Area.FillBitmapName = "Sky"
```

以下のサンプルコードでは、グラフの背景として、StarSuite 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。

```
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
```

グラフ軸

StarSuite では、5 種類のグラフ軸が利用できます。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、StarSuite 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。

X 、Y 、Z の主軸

X 、Y 、Z の各軸については、タイトル、ラベル、目盛線、補助目盛線を表示できます。これらの要素については、その表示と非表示を個別に設定できます。グラフオブジェクトには、これらの設定を行うため以下の属性が用意されています (ここでは X 軸に関する説明となっていますが、その他の軸についても同様)。

- **HasXAxis (ブール値)** - X 軸を表示する指定。
- **XAxis (オブジェクト)** - X 軸に関する情報を収めたオブジェクト (com.sun.star.chart.ChartAxis サービスをサポート)。
- **HasXAxisDescription (ブール値)** - X 軸のラベルを表示する指定。
- **HasXAxisGrid (ブール値)** - X 軸の主目盛線を表示する指定。
- **XMainGrid (オブジェクト)** - X 軸の主目盛線に関する情報を収めたオブジェクト (com.sun.star.chart.ChartGrid サービスをサポート)。
- **HasXAxisHelpGrid (ブール値)** - X 軸の補助目盛線を表示する指定。
- **XHelpGrid (オブジェクト)** - X 軸の補助目盛線に関する情報を収めたオブジェクト (com.sun.star.chart.ChartGrid サービスをサポート)。
- **HasXAxisTitle (ブール値)** - X 軸のタイトルを表示する指定。
- **XAxisTitle (オブジェクト)** - X 軸のタイトルに関する情報を収めたオブジェクト (com.sun.star.chart.ChartTitle サービスをサポート)。

X 、Y の第 2 数値軸

X 軸および Y 軸の第 2 数値軸 (第 2 軸)に関しては、以下の属性が用意されています (ここでは X の第 2 数値軸について説明)。

- **HasSecondaryXAxis (ブール値)** - X 軸の第 2 数値軸を表示する指定。
- **SecondaryXAxis (オブジェクト)** - X 軸の第 2 数値軸に関する情報を収めたオブジェクト (com.sun.star.chart.ChartAxis サービスをサポート)。
- **HasSecondaryXAxisDescription (ブール値)** - X 軸の第 2 数値軸のラベルを表示する指定。

グラフ軸の属性

StarSuite のグラフで使用する軸オブジェクトは、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第 6 章を参照) および線 (com.sun.star.drawing.LineStyle サービスについては第 8 章を参照) に関係する属性に加えて、以下の属性が用意されています。

- **Max (倍精度)** - 軸の最大値。
- **Min (倍精度)** - 軸の最小値。
- **Origin (倍精度)** - 軸の原点。

- **StepMain (倍精度)** - 軸の主目盛線の間隔。
- **StepHelp (倍精度)** - 軸の補助目盛線の間隔。
- **AutoMax (ブール値)** - 軸の最大値を自動判定する指定。
- **AutoMin (ブール値)** - 軸の最小値を自動判定する指定。
- **AutoOrigin (ブール値)** - 軸の原点を自動判定する指定。
- **AutoStepMain (ブール値)** - 軸の主目盛線の間隔を自動判定する指定。
- **AutoStepHelp (ブール値)** - 軸の補助目盛線の間隔を自動判定する指定。
- **Logarithmic (ブール値)** - 軸を対数目盛にする指定 (通常は線形表示)。
- **DisplayLabels (ブール値)** - 軸ラベルを表示する指定。
- **TextRotation (ロング整数)** - 100 分の 1 度単位で指定した軸ラベルの回転角。
- **Marks (定数)** - 軸の区切りを、グラフエリアの内側または外側に表示する指定 (`com.sun.star.chart.ChartAxisMarks` に定められた値)。
- **HelpMarks (定数)** - 軸の補助区切りを、グラフエリアの内側または外側に表示する指定 (`com.sun.star.chart.ChartAxisMarks` に定められた値)。
- **Overlap (ロング整数)** - 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 (ロング整数)** - 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)。
- **ArrangeOrder (列挙型)** - 軸ラベルの配置を 1 列に並べるか、上下交互 2 列に並べるかの指定 (`com.sun.star.chart.ChartAxisArrangeOrderType` に定められた値)。
- **TextBreak (ブール値)** - 折り返し表示を許可する指定。
- **TextCanOverlap (ブール値)** - ラベルの重ね合わせを許可する指定。
- **NumberFormat (ロング整数)** - 数の書式 (第 7 章の「数値、日付、テキストの表示書式」を参照)。

軸目盛線の属性

軸目盛線のオブジェクトは、`com.sun.star.chart.ChartGrid` サービスをベースとしており、その表示設定には `com.sun.star.drawing.LineStyle` サービスの属性を使用します (第 8 章を参照してください)。

軸タイトルの属性

軸タイトルの表示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

以下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。

```
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
```

3D グラフ

StarSuite のグラフの多くは、3 次元表示が可能です。このような 3D 表示が可能なグラフは、`com.sun.star.chart.Dim3DDiagram` サービスをサポートしています。このサービスには、以下の属性のみが用意されています。

- **`Dim3D` (ブール値)** - 3D 表示をする指定。

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。

StarSuite 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、以下の属性を使用できます。

- **`Stacked` (ブール値)** - 積み上げグラフによる表示の指定。
- **`Percent` (ブール値)** - 通常の数値表示ではなく、個々の構成比をパーセント表示する指定。

グラフの種類

折れ線グラフ

折れ線グラフ (`com.sun.star.chart.LineDiagram` サービス) では、最大で X 軸を 1 本、Y 軸を 2 本、Z 軸を 1 本表示できます。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。また、積み上げグラフの表示 (`com.sun.star.chart.StackableDiagram`) も可能です。

折れ線グラフには、以下の属性を指定できます。

- **SymbolType (定数)** - データポイントの表示記号の指定 (com.sun.star.chart.ChartSymbolType に定められた定数値)。
- **SymbolSize (ロング整数)** - 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 (文字列)** - データポイントの表示記号とする画像のファイル名。
- **Lines (ブール値)** - データポイント間を線でつなぐ指定。
- **SplineType (ロング整数)** - 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 (ロング整数)** - スプライン処理の次数 (B 平滑線のみ)。
- **SplineResolution (ロング整数)** - スプライン処理で使用する中間点の数。

エリアグラフ

エリアグラフ (com.sun.star.chart.AreaDiagram サービス) では、最大で X 軸を 1 本、Y 軸を 2 本、Z 軸を 1 本表示できます。 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。また、積み上げグラフの表示 (com.sun.star.chart.StackableDiagram) も可能です。

棒グラフ

棒グラフ (com.sun.star.chart.BarDiagram サービス) では、最大で X 軸を 1 本、Y 軸を 2 本、Z 軸を 1 本表示できます。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。また、積み上げグラフの表示 (com.sun.star.chart.StackableDiagram) も可能です。

棒グラフには、以下の属性を指定できます。

- **Vertical (ブール値)** - 標準の横棒表示に対して、縦棒表示とする指定。
- **Deep (ブール値)** - 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 (ブール値)** - 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 (ロング整数)** - 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。

円グラフ

円グラフ (com.sun.star.chart.PieDiagram サービス) 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。ただしこのグラフでも、2D 表示および、3D 表示 (com.sun.star.chart.Dim3Ddiagram サービス) を使用できます。

データベースアクセス

StarSuite 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)。このような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。このようなドライバの中には、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、StarSuite 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。

このようなドライバは 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、StarSuite 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。

StarSuite のデータベースインターフェースは、StarSuite 7 の開発を通じて発展してきました。ただし従来は、主として `Application` オブジェクトの各種メソッドを用いてデータベースアクセスを行なっていたため、StarSuite 7 のインターフェースも、いくつかのオブジェクトに細分されています。データベース関連の機能でルートオブジェクトとして用いられるものは、`DatabaseContext` です。

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 StarSuite 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。

SDBC の使用する SQL は、SQL-ANSI 標準をベースとして実装されています。そのため Microsoft が独自に拡張した `INNER JOIN` コンストラクトなどはサポートされていません。このようなコマンドに関しては、該当する標準コマンドに置き換える必要があります (たとえば `INNER JOIN` は `WHERE` 句を用いた構文で記述可能)。

データベースアクセスの種類

StarSuite のデータベースインターフェースは、StarSuite Writer と StarSuite 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 StarSuite 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。データベースウィンドウから文書ドキュメントへのデータの移動には、ドラッグ & ドロップも使用できます。

データベーステーブルから StarSuite 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。

StarSuite 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 StarSuite Writer または StarSuite 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 StarSuite 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。

ただし本章では、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、各種の操作方法を説明します。

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 StarSuite 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行できます。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。また、StarSuite Basic 上からもデータソースの作成および操作が行えます。

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。

以下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 (文字列)** - データソースの名前。
- **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)。
- **Info (配列)** - PropertyValue の値に接続パラメータを収めた配列 (通常はユーザー名とパスワードが最低必要)。
- **User (文字列)** - ユーザーの名前。
- **Password (文字列)** - ユーザーのパスワード (保存はされない)。
- **IsPasswordRequired (ブール値)** - ユーザーに対してパスワードを要求する指定。
- **IsReadOnly (ブール値)** - データベースアクセスを読み取り専用とする指定。
- **NumberFormatsSupplier (オブジェクト)** - データベースで使用する数の書式を収めたオブジェクト (com.sun.star.util.XNumberFormatsSupplier インターフェースをサポート。詳細情報は第 7 章「数値、日付、テキストの表示書式」を参照)。
- **TableFilter (配列)** - 表示するテーブル名のリスト。
- **TableTypeFilter (配列)** - 表示するテーブルタイプのリスト。指定可能な値は TABLE、VIEW、SYSTEM TABLE。
- **SuppressVersionColumns (ブール値)** - バージョン管理用の列を非表示とする指定。

StarSuite 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、StarSuite のデータソースでは、StarSuite 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。StarSuite 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。

以下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 (文字列)** - クエリーの名前。
- **Command (文字列)** - SQL コマンド (SELECT などのコマンド)。
- **UpdateTableName (文字列)** - 複数テーブルを用いるクエリーに対する、データ更新が可能なテーブル名の指定。
- **UpdateCatalogName (文字列)** - 更新テーブルのカタログ名。
- **UpdateSchemaName (文字列)** - 更新テーブルのダイアグラム名。

以下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。

データベースフォームとのリンク

StarSuite には、データソース操作を簡易化するため、データソースをデータベースフォームにリンクするオプションが用意されています。このようなリンクの構築には `getBookmarks()` メソッドを使用します。実行後の戻り値としては、データソースとのリンクがすべて格納されたコンテナ (`com.sun.star.sdb.DefinitionContainer`) が返されます。ブックマークに対しては、名前またはインデックスによりアクセスできます。

以下のサンプルコードでは、MyBookmark というブックマークの URL を取得します。

```
Dim DatabaseContext As Object
Dim DataSource As Object
Dim Bookmarks As Object
Dim URL As String
Dim I As Integer

DatabaseContext = createUnoService("com.sun.star.sdb.DatabaseContext")
DataSource = DatabaseContext.getByName("Customers")
Bookmarks = DataSource.Bookmarks()

URL = Bookmarks.getByName("MyBookmark")
MsgBox URL
```

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、このような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。

StarSuite 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。以下のサンプルコードでは、既存のデータソースを利用します。

```
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、StarSuite 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 StarSuite 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。

以下のサンプルコードでは、`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.createStatement` 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` を実行して、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なっていますが、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 または ADO を利用した場合の `Recordset` オブジェクトに相当します。

StarSuite 7 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、StarSuite 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。同様のメソッドとしては、以下のものが利用できます。

- **`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 StarSuite 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。

たとえば「テーブルからのデータの取得」の節のサンプルコードで見たような単純な処理の場合、`ResultSet` は最小限の機能のみを使用しています。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 (定数)** - データの変更を許可する指定 (`com.sun.star.sdbc.ResultSetConcurrency` に定められた定数値)。
- **ResultSetType (定数)** - `ResultSets` のタイプに関する指定 (`com.sun.star.sdbc.ResultSetType` に定められた定数値)。

以下に `com.sun.star.sdbc.ResultSetConcurrency` に定められた定数を示します。

- **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、以下の指定を行えます。

- **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。

`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、以下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、pdateRow() 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。

第 11 章

ダイアログ

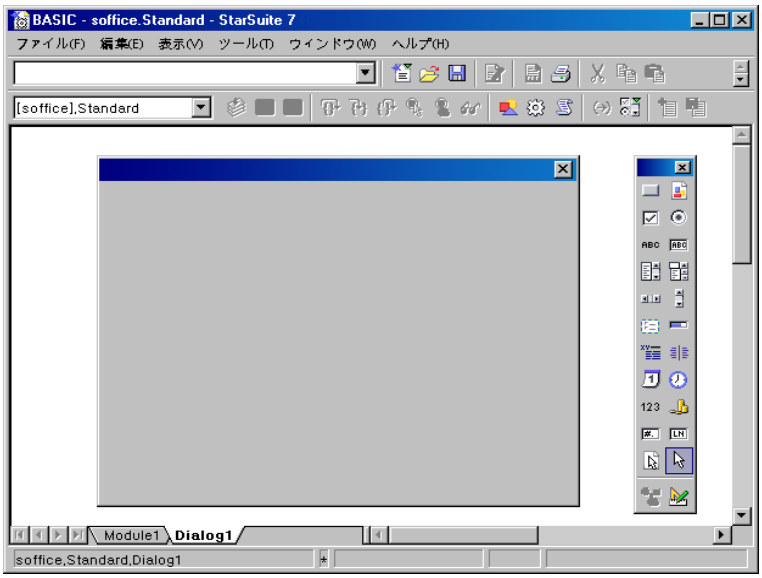
StarSuite のドキュメントでは、ユーザー定義によるカスタムダイアログやフォームを表示することができます。これらは StarSuite Basic マクロとのリンクが可能という点で、StarSuite Basic の利用範囲を大いに広げています。ダイアログの用途としては、データベースの登録情報の表示をはじめ、オートパイロットによる新規ドキュメント作成時の手順表示などにも利用できます。

ダイアログの操作

StarSuite 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。

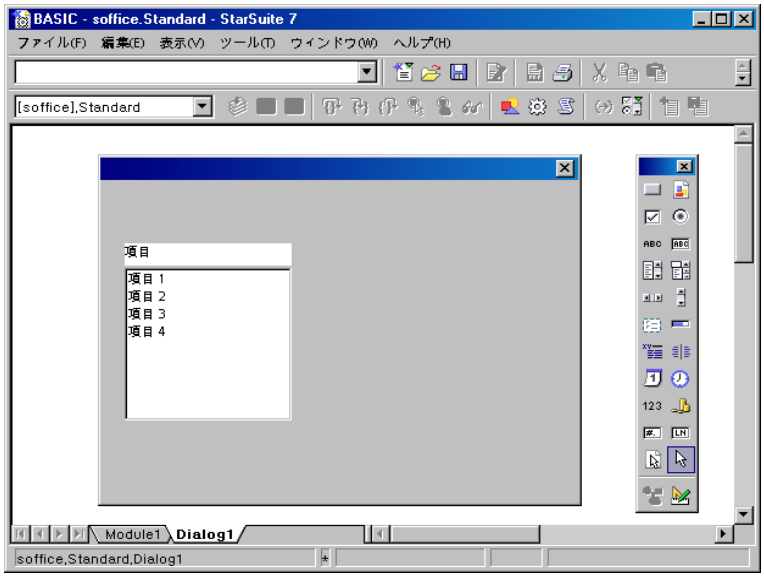
ダイアログの作成法

ダイアログの作成は StarSuite ダイアログエディタを使って行いますが、その操作方法は StarSuite Draw の場合と同様です。



コントロール要素の基本的な配置法は、使用するコントロール要素をデザインパレット (画面右) で選択して、ダイアログウィンドウ上の表示領域で必要なサイズにドラッグするだけです。

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、以下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。読み込んでいない場合は、`LoadLibrary` メソッドで読み込みます。

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なった後、`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。サンプルダイアログのタイプは、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。これらのボタンの詳細情報については、本章のダイアログコントロールの詳細の節で説明しています。

OK の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

必要であれば、タイトルバーの閉じるボタンをクリックすることでも、ダイアログウィンドウをクローズできます。そのような操作を行なった場合、ダイアログの `Execute` メソッドによる戻り値には 0 が返されますが、これはキャンセルのボタンを押した場合と同様です。

プログラムによる明示的なクローズ処理

プログラム内に以下のような `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なっています。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。

StarSuite Basic では、コントロール要素の名前に使われる大文字と小文字が区別されるので注意が必要です。

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

StarSuite 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。

StarSuite 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して、プログラム制御によりアクセスする際に利用します。

```
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、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。

属性

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、以下のモデル属性を用いて参照することができます。

- **Model.Name (文字列)** - コントロール要素の名前。

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、以下のモデル属性を用いて参照することができます。

- **Model.Title (文字列)** - ダイアログのタイトル (ダイアログでのみ利用可能)。

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、以下のモデル属性を用いて参照することができます。

- **Model.Height (ロング整数)** - コントロール要素の高さ (ma 単位)。
- **Model.Width (ロング整数)** - コントロール要素の幅 (ma 単位)。
- **Model.PositionX (ロング整数)** - 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 (ロング整数)** - 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)。

StarSuite 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。StarSuite 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。このような操作に関係するコントロール要素モデルでは、以下の属性が利用できます。

- **Model.Enabled (ブール値)** - コントロール要素をアクティブにする指定。
- **Model.Tabstop (ブール値)** - 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 (ロング整数)** - Tab キーによるフォーカス移動の順序の指定。

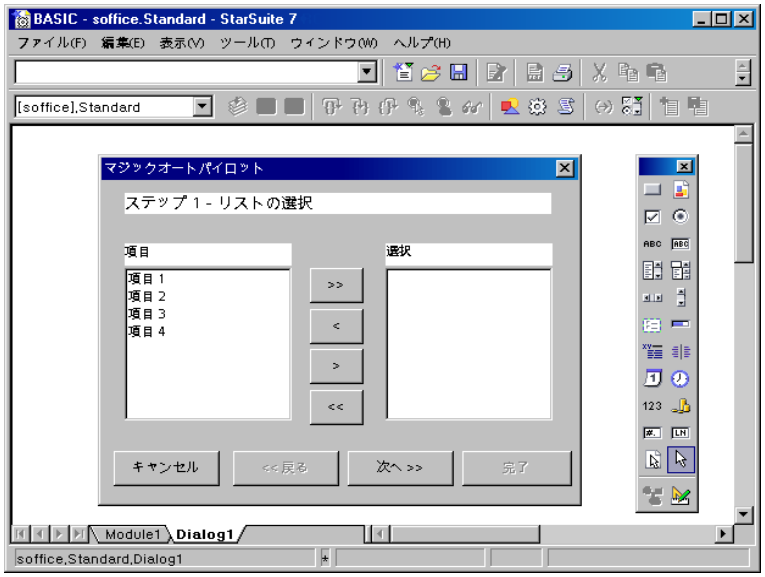
最後に紹介する getFocus メソッドは、ダイアログ上のコントロール要素にフォーカス移動を行うよう設定するものです。

- **getFocus** - コントロール要素にフォーカスを移動する設定 (ダイアログのみ)。

マルチページダイアログ

StarSuite 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` 属性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` 属性が存在し、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` 属性値に 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。



たとえば上記のダイアログの場合、分割線および「キャンセル」、「戻る」、「次へ」、「完了」の各ボタンは、すべてのタブページ上に表示させる性質のコントロールなので、これらの `Step` 値は `0` に設定しておきます。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`)。

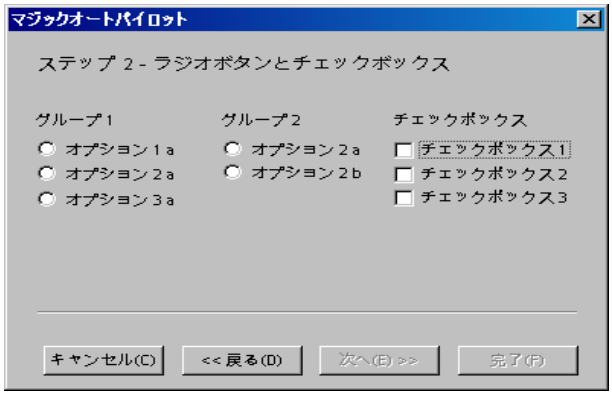
以下のサンプルコードでは、イベントハンドラに応じて、「次へ」と「戻る」というボタンの `Step` 値を増減させて、ボタンのステップ値の表示用ステータスを変更しています。

```
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、以下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:



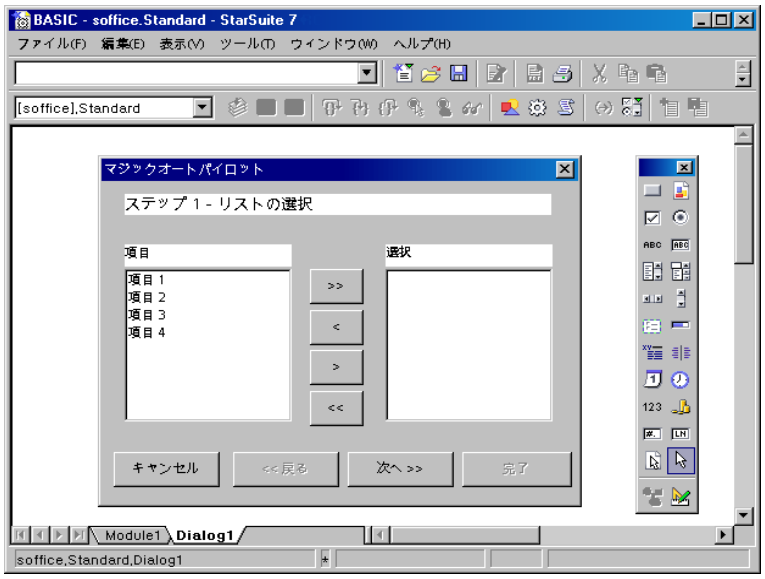
イベント

StarSuite 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。イベントハンドラとは、特定のアクションの実行に応じて、事前に定めておいた手続きを起動するための機構で、このようなトリガーとなるアクションは、他のイベント中で発生した場合も有効です。イベントハンドラを利用することで、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といった、各種の処理を制御できます。

StarSuite 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:** キーボードのキー操作に関係するイベント
- **フォーカス移動:** コントロール要素のアクティブ化または非アクティブ化に応じて StarSuite が発生するイベント
- **コントロール要素の固有イベント:** 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、StarSuite の開発環境で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 (フォームを使用する場合) を用意しておく必要があります。



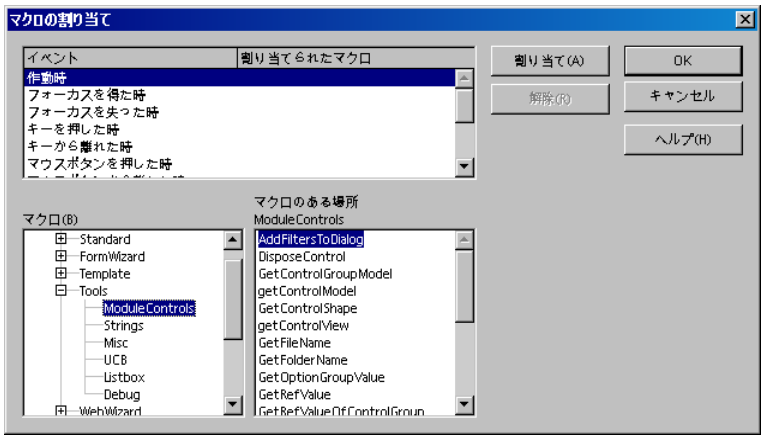
上記の図は StarSuite Basic の開発環境を示したもので、このダイアログウィンドウには 2 つのリストボックスが配置されています。この場合、2 つのリストボックスの間に配置されたボタンを使って、リスト内の各項目を相互に移動できるものとします。

このような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 StarSuite 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危険性があります。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 つ別途作成しておくべきです。

以下のサンプルコードでは、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.SelectedItemPos, 1)
      Else
            Beep
      End If
End Sub
```

この手続きを StarSuite 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、StarSuite 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。そのような場合、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利用できます。

StarSuite Basic では以下のような方法で、イベントから手続きに対してオブジェクトパラメータを渡すことができます。

```
Sub ProcessEvent(Event As Object)

End Sub
```

Event オブジェクトの構成および付随する属性は、手続き呼び出しのトリガーとなるイベントの種類に依存します。以降の節では、イベントタイプの詳細について説明します。

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、割り当てられたコントロール要素およびモデルへアクセスします。コントロール要素へのアクセスは、以下の形式で行います。

```
Event.Source
```

同様に、モデルへのアクセスは以下の形式で行います。

```
Event.Source.Model
```

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

StarSuite Basic では以下のマウスイベントを利用できます。

- **マウス操作時** - ユーザーによるマウスポインタの移動
- **キーを押しマウス操作する時** - ユーザーによるキーを押し下げながらのマウスポインタの移動
- **マウスボタンを押した時** - ユーザーによるマウスボタンの押し下げ
- **マウスボタンから離れた時** - ユーザーによるマウスボタンの解放
- **マウスを外に出した時** - 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、以下の情報を取り扱うことができます。

- **`Buttons` (整数)** - 押し下げられたボタン (`com.sun.star.awt.MouseButton` に定められた 1 または複数の定数)。
- **`X` (ロング整数)** - 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` (ロング整数)** - 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` (ロング整数)** - 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 1 つのイベントとして処理されるため、StarSuite が十分高速に反応できる場合、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、以下のものです。

- **LEFT** - マウスの左ボタン
- **RIGHT** - マウスの右ボタン
- **MIDDLE** - マウスの中央ボタン

以下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 StarSuite Basic では利用できません。StarSuite Basic では `Click` イベントの代わりに `MouseUp` 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

StarSuite Basic では以下のキーボードイベントを利用できます。

- **キーを押した時** - ユーザーによるキーの押し下げ
- **キーから離れた時** - 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 StarSuite Basic が発生するイベントは 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下げても、それだけでは独立したイベントは発生しません。

StarSuite Basic は、押し下げら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、以下の属性を利用できます。

- **`KeyCode` (整数)** - 押し下げられたキーを示すコード番号 (`com.sun.star.awt.Key` に定められた値)
- **`KeyChar` (文字列)** - 入力された文字 (修飾キーによる効果も含めた結果)

以下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。

- フォーカスを得た時 - コントロール要素へのフォーカス移動
- フォーカスを失った時 - コントロール要素からのフォーカスの喪失

フォーカスイベント関係の `Event` オブジェクトは、以下のように構成されています。

- **`FocusFlags` (整数)** - フォーカス移動の原因 (`com.sun.star.awt.FocusChangeReason` に定められた値)。
- **`NextFocus` (オブジェクト)** - フォーカスの移動先のオブジェクト (フォーカスを失った時のイベントのみ)。
- **`Temporary` (ブール値)** – フォーカスが一時的に喪失されたかの判定。

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以下のイベントです。

- **When `Item` Changed** - コントロール要素の値の変更
- **ステータスを変更した時** - コントロール要素のステータスの変更

- **テキストを変更した時** - コントロール要素のテキストの変更
- **作動時** - 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。

ステータスを変更した時のイベントに関係する属性には以下のものがあります。

- **Selected (ロング整数)** - 現在の選択項目
- **Highlighted (ロング整数)** - 現在の強調表示中の項目
- **ItemId (ロング整数)** - 項目の ID

ダイアログコントロールの詳細

StarSuite Basic には各種のコントロール要素が用意されていますが、これらは以下の 4 つのグループに分類できます。

入力フィールド:

- テキストフィールド (テキストボックス)
- 日付フィールド
- 時刻フィールド
- 番号フィールド
- 通貨フィールド
- その他の書式設定可能なフィールド

ボタン:

- 標準ボタン (コマンドボタン)
- チェックボックス
- ラジオボタン

選択用リスト:

- リストボックス
- コンボボックス

その他のコントロール要素:

- スクロールバー (水平および垂直スクロールバー)
- グループ枠
- 進行グラフ
- 分割線 (横線および縦線)

- イメージコントロール
- ファイルの選択

次に、これらのコントロール要素のうち、特に重要なものについて説明します。

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「作動時」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)。この属性値を 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 (ロング整数)** - 背景の色
- **Model.DefaultButton (ブール値)** - 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 (構造体)** - 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 (文字列)** - ボタンに表示するラベル (タイトル)
- **Model.Printable (ブール値)** - コントロール要素を印刷可能とする指定
- **Model.TextColor (ロング整数)** - コントロール要素のテキストの色
- **Model.HelpText (文字列)** - 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 (文字列)** - コントロール要素で使用するオンラインヘルプの URL
- **PushButtonType (整数)** - ボタンアクションの指定 (0: なし、1: OK、2: キャンセル)

オプションボタン

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。

オプションボタンのコントロール要素には、以下の 2 つの属性があります。

- **State (ブール値)** - ボタンをアクティブにする指定
- **Label (文字列)** - ボタンに表示するラベル (タイトル)

オプションボタンのモデルからは、以下の属性も使用できます。

- **Model.FontDescriptor (構造体)** - 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 (文字列)** - コントロール要素に表示するラベル (タイトル)
- **Model.Printable (ブール値)** - コントロール要素を印刷可能とする指定
- **Model.State (整数)** - 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指定 (その他の値の場合)
- **Model.TextColor (ロング整数)** - コントロール要素のテキストの色
- **Model.HelpText (文字列)** - 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 (文字列)** - コントロール要素で使用するオンラインヘルプの URL

複数のオプション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、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、StarSuite 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 StarSuite Basic では、グループ枠の中にオプションボタンを挿入するということはできません。StarSuite 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、以下の属性を指定できます。

- **State (整数)** - チェックボックスの状態 (0: No、1: Yes、2: 中間状態)
- **Label (文字列)** - コントロール要素に表示するラベル (タイトル)
- **enableTriState (ブール値)** - 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、以下の属性を使用できます。

- **Model.FontDescriptor (構造体)** - 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 (文字列)** - コントロール要素のラベル (タイトル)
- **Model.Printable (ブール値)** - コントロール要素を印刷可能とする指定
- **Model.State (整数)** - チェックボックスの状態 (0: No、1: Yes、2: 中間状態)
- **Model.Tabstop (ブール値)** - 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 (ロング整数)** - コントロール要素のテキストの色
- **Model.HelpText (文字列)** - 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 (文字列)** - 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。テキストボックスの機能は、主として `com.sun.star.awt.UnoControlEdit` サービスにより提供されています。

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を制御できます。

テキストボックスには、以下の属性が用意されています。

- **Text (文字列)** - 現在の表示テキスト
- **SelectedText (文字列)** - 現在の強調表示中のテキスト
- **Selection (構造体)** - 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 (整数)** - フィールド内に入力可能な最大文字数
- **Editable (ブール値)** - 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 (ブール値)** - 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、以下の属性を利用できます。

- **Model.Align (整数)** - テキストの配置 (0: 左揃え、1: 中央揃え、2: 右揃え)
- **Model.BackgroundColor (ロング整数)** - コントロール要素の背景色
- **Model.Border (整数)** - 外枠の種類 (0: なし、1: 3D 表示、2: 平坦な線)
- **Model.EchoChar (文字列)** - パスワードフィールドとしての利用時のマスク記号
- **Model.FontDescriptor (構造体)** - 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 (ブール値)** - コントロール要素のテキスト内で改行する指定
- **Model.HScroll (ブール値)** -
- **Model.MaxTextLen (整数)** - 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 (ブール値)** - 複数行表示を許可する指定
- **Model.Printable (ブール値)** - コントロール要素を印刷可能とする指定
- **Model.ReadOnly (ブール値)** - コントロール要素を読み取り専用とする指定
- **Model.Tabstop (ブール値)** - 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 (文字列)** - コントロール要素の表示テキスト
- **Model.TextColor (ロング整数)** - コントロール要素のテキストの色
- **Model.VScroll (ブール値)** - 垂直スクロールバーを表示する指定
- **Model.HelpText (文字列)** - 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 (文字列)** - 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、以下の属性をサポートしています。

- **ItemCount (整数)** - 項目の数 (読み取り専用)
- **SelectedItem (文字列)** - 強調表示中の項目のテキスト(読み取り専用)
- **SelectedItems (文字列配列)** - 強調表示中の項目を格納したデータフィールド (読み取り専用)
- **SelectedItemPos (整数)** - 強調表示中の項目の数 (読み取り専用)
- **SelectedItemsPos (整数配列)** - 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 (ブール値)** - 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 (ブール値)** - リスト内で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、以下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、以下の属性が用意されています。

- **Model.BackgroundColor (ロング整数)** - コントロール要素の背景色
- **Model.Border (整数)** - 外枠の種類 (0: なし、1: 3D 表示、2: 平坦な線)
- **Model.FontDescriptor (構造体)** - 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 (整数)** - コントロール要素の行数
- **Model.MultiSelection (ブール値)** - 項目の複数選択を許可する指定
- **Model.SelectedItems (文字列配列)** - 強調表示中の項目のリスト
- **Model.StringItemList (文字列配列)** - すべての項目のリスト
- **Model.Printable (ブール値)** - コントロール要素を印刷可能とする指定
- **Model.ReadOnly (ブール値)** - コントロール要素を読み取り専用とする指定
- **Model.Tabstop (ブール値)** - 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 (ロング整数)** - コントロール要素のテキストの色

- **Model.HelpText (文字列)** - 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 (文字列)** - 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、StarSuite 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) を割り当てる場合、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。

第 12 章

フォーム

StarSuite に用意されているフォーム機能には、前章で説明したダイアログと共通する部分が多くあります。その一方で、両者には以下のような相違点もあります。

- 各ダイアログは単独で表示され、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、StarSuite 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。フォームの作成は、**フォーム機能のツールバー**を利用して、ドキュメント上に直接配置します。
- ダイアログの機能は、すべての StarSuite 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、文書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、第 11 章「ダイアログ」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

StarSuite のフォームは、文書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。

StarSuite 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 (属性) 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

StarSuite 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、以下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。このような機能は、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、以下のような 3 段階構成になっています。

- まず、各コントロール要素の Model (モデル) オブジェクトです。フォーム用のコントロール要素を StarSuite Basic でプログラミング制御する際には、このオブジェクトが中心となります。
- 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ます (主に位置とサイズ)。

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へアクセスする際には、対応するモデルを特定しておく必要があります。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、その前に取り上げ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` というオドキュメント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、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、シェイプ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オブジェクトには、Size および Position という属性が用意されています。

- **Size (構造体)** - コントロール要素のサイズ (com.sun.star.awt.Size に定められた構造体)。
- **Position (構造体)** - コントロール要素の位置 (com.sun.star.awt.Point に定められた構造体)。

以下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シェイプ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ています。

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。具体的な扱い方については、第 11 章の「データベースフォーム」の節で説明しています。

ボタン

フォーム用ボタンのモデルオブジェクトには、以下の属性が用意されています。

- **BackgroundColor (ロング整数)** - 背景色。
- **DefaultButton (ブール値)** - 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 (ブール値)** - コントロール要素をアクティブにする指定。
- **Tabstop (ブール値)** - 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 (ロング整数)** - Tab キーによるフォーカス移動の順序の指定。
- **FontName (文字列)** - フォントの名前。
- **FontHeight (整数)** - ポイント単位 (pt) で指定した文字の高さ。
- **Tag (文字列)** - 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 (文字列)** - ボタンを URL にリンクさせる場合のターゲット URL。
- **TargetFrame (文字列)** - TargetURL のターゲット URL の内容を開くウィンドウ (または フレーム) の名前 (URL リンク型のボタンをクリックした場合)。
- **Label (文字列)** - ボタンのラベル (タイトル)。
- **TextColor (ロング整数)** - コントロール要素のテキストの色。
- **HelpText (文字列)** - 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 (文字列)** - コントロール要素で使用するオンラインヘルプの URL。
- **ButtonType (列挙型)** - ボタンに実行させるアクションの種類 (`com.sun.star.form.FormButtonType` に定められた値)。

`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、この属性指定用に以下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モデルオブジェクトには、以下の属性が用意されています。

- **Enabled (ブール値)** - コントロール要素をアクティブにする指定。
- **Tabstop (ブール値)** - 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 (ロング整数)** - Tab キーによるフォーカス移動の順序の指定。
- **FontName (文字列)** - フォントの名前。
- **FontHeight (整数)** - ポイント単位 (pt) で指定した文字の高さ。
- **Tag (文字列)** - 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 (文字列)** - ボタンのラベル (タイトル)。
- **Printable (ブール値)** - コントロール要素を印刷可能とする指定。
- **State (整数)** - 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 (文字列)** - 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 (ロング整数)** - コントロール要素のテキストの色。
- **HelpText (文字列)** - 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 (文字列)** - コントロール要素で使用するオンラインヘルプの URL。

オプション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オプションボタンには、すべて同じ名前をつけます。その際に StarSuite 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、StarSuite 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。以下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方法を示します。

```
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方法を説明した、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

フォーム用のチェックボックスのモデルオブジェクトには、以下の属性が用意されています。

- **Enabled (ブール値)** - コントロール要素をアクティブにする指定。
- **Tabstop (ブール値)** - 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 (ロング整数)** - Tab キーによるフォーカス移動の順序の指定。
- **FontName (文字列)** - フォントの名前。
- **FontHeight (整数)** - ポイント単位 (pt) で指定した文字の高さ。
- **Tag (文字列)** - 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 (文字列)** - ボタンのラベル (タイトル)。
- **Printable (ブール値)** - コントロール要素を印刷可能とする指定
- **State (整数)** - 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 (文字列)** - 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 (ロング整数)** - コントロール要素のテキストの色。

- **HelpText (文字列)** - 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 (文字列)** - 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)

フォーム用のテキストボックス (テキストフィールド) のモデルオブジェクトには、以下の属性が用意されています。

- **Align (整数)** - テキストの配置 (0: 左揃え、1: 中央揃え、2: 右揃え)。
- **BackgroundColor (ロング整数)** - コントロール要素の背景色。
- **Border (整数)** - 外枠の種類 (0: なし、1: 3D 表示、2: 平坦な線)。
- **EchoChar (文字列)** - パスワードフィールドとしての利用時のマスク記号。
- **FontName (文字列)** - フォントの名前。
- **FontHeight (整数)** - ポイント単位 (pt) で指定した文字の高さ。
- **HardLineBreaks (ブール値)** - コントロール要素のテキスト内で改行する指定。
- **HScroll (ブール値)** - 水平スクロールバーを表示する指定。
- **MaxTextLen (整数)** - 表示テキストの最大数で、0 の指定は制限無しに対応。
- **MultiLine (ブール値)** - 複数行表示を許可する指定。
- **Printable (ブール値)** - コントロール要素を印刷可能とする指定。
- **ReadOnly (ブール値)** - コントロール要素を読み取り専用とする指定。
- **Enabled (ブール値)** - コントロール要素をアクティブにする指定。
- **Tabstop (ブール値)** - 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 (ロング整数)** - Tab キーによるフォーカス移動の順序の指定。
- **FontName (文字列)** - フォントの名前。
- **FontHeight (整数)** - ポイント単位 (pt) で指定した文字の高さ。
- **Text (文字列)** - コントロール要素の表示テキスト。
- **TextColor (ロング整数)** - コントロール要素のテキストの色。
- **VScroll (ブール値)** - 垂直スクロールバーを表示する指定。
- **HelpText (文字列)** – 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 (文字列)** - 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

フォーム用のリストボックスのモデルオブジェクトには、以下の属性が用意されています。

- **BackgroundColor (ロング整数)** - コントロール要素の背景色。
- **Border (整数)** - 外枠の種類 (0: なし、1: 3D 表示、2: 平坦な線)。
- **FontDescriptor (構造体)** - 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 (整数)** - コントロール要素の行数。
- **MultiSelection (ブール値)** - 項目の複数選択を許可する指定。
- **SelectedItems (文字列配列)** - 強調表示中の項目のリスト。
- **StringItemList (文字列配列)** - すべての項目のリスト。
- **ValueItemList (文字列配列)** - 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 (ブール値)** - コントロール要素を印刷可能とする指定
- **ReadOnly (ブール値)** - コントロール要素を読み取り専用とする指定。
- **Enabled (ブール値)** - コントロール要素をアクティブにする指定。
- **Tabstop (ブール値)** - 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 (ロング整数)** - Tab キーによるフォーカス移動の順序の指定。
- **FontName (文字列)** - フォントの名前。
- **FontHeight (整数)** - ポイント単位 (pt) で指定した文字の高さ。
- **Tag (文字列)** - 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 (ロング整数)** - コントロール要素のテキストの色。
- **HelpText (文字列)** - 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 (文字列)** - 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以下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` 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

StarSuite 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。

ユーザーが選択したテーブルやクエリーに対しては、ブラウズや検索をはじめ、データレコードの変更や新規挿入などの処理が行えます。データベースからのデータの取得や、変更したデータのデータベースへの書き込み処理は、StarSuite が自動的に実行します。

データベースフォームの構造は、基本的に StarSuite の通常のフォームと同じものです。ただし、通常の属性の他に以下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 (第 10 章「データベースアクセス」の説明を参照。StarSuite 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 (文字列)** - 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 (定数)** - 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 には以下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、以下の属性で指定します。

- **DataField (文字列)** – リンクす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用には、テーブルコントロールというコントロール要素も用意されています。このコントロールを用いると、テーブルやクエリーのデータを直接表示させることができます。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。ただし、この処理に関係する API は非常に複雑であるため、ここでは説明しません。

付録

VBA からのプログラムコードの移行について

StarOffice 5.x からのプログラムコードの移行について

索引

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

い

え

お

か

き

く

け

こ

さ

し

す