4th Dimension®

ランゲージリファレンス

Mac™OS and Windows®版

4th Dimension
by
Laurent Ribardière
Adapted by Bernard Gallet

4th Dimension ランゲージリファレンス

注意

このソフトウェアの使用に際し、本製品に同梱のLicense Agreement（使用許諾契約書）に同意する必要があります。ソフトウェアを使用する前に、License Agreementを注意深くお読みください。

このマニュアルに記載されている事項は、将来予告なしに変更されることがあり、いかなる変更に関してもACI SAおよびACI USは一切の責任を負いかねます。このマニュアルで説明されるソフトウェアは、本製品に同梱のLicense Agreement（使用許諾契約書）のもとでのみ使用することができます。
ソフトウェアおよびマニュアルの一部または全部を、ライセンス保持者がこの契約条件を許諾した上での個人使用目的以外に、いかなる目的であれ、電子的、機械的、またどのような形であっても、無断で複製、配布することはできません。

目次

目次

目次

目次

目次

序章

4th Dimensionは、総数500以上にもおよぶ独自のプログラミング言語を持つ強力なリレーショナルデータベースです。この4th Dimensionプログラミング言語は、簡単な計算処理から複雑なカスタムユーザインタフェースの作成に至るまで、さまざまな業務で用いることができます。例えば、4th Dimensionプログラミング言語を使用すれば、次のようなことが可能になります。

■「ユーザ」モードで使用できる任意エディタのプログラミングによるアクセス

■ データベース情報からのレポートやグラフ、ラベルの作成と印刷

■ 他のデバイスとの通信

■ ドキュメント管理

■ 4th Dimensionと別のデータベース間におけるデータ読み込みと書き出し

■ 別の言語で作成したプログラムを4th Dimensionプログラミング言語の中に組み込むこと

4th Dimensionプログラミング言語は、柔軟性と能力を備え、あらゆるレベルのユーザや開発者がさまざまな情報管理業務を達成するための理想的なツールです。初心者ユーザでも計算処理を素早く行うことができます。またプログラミング経験がなくても、ある程度コンピュータの知識を持っているユーザであれば、自分のデータベースをカスタマイズすることができます。一方、熟練した開発者であれば、4th Dimensionの強力なプログラミング言語を使用して、ファイル転送や通信などの高度な機能をデータベースに組み込むことができます。他の言語でプログラミング経験がある開発者は、その独自のコマンドを4th Dimensionに追加することができます。

4th Dimensionプログラミング言語は、4th Dimensionのプラグインがアプリケーションに組み込まれると増加します。それぞれのプラグインは、専用のコマンドを持っています。

マニュアル全般について

4th Dimension（4D First）と4D Serverの両方の機能については、下記のマニュアルで説明しています。4D Serverの専門的な機能の説明は、4D Serverパッケージに含まれている『4D Serverリファレンス』の中でのみ説明されています。

『4th Dimensionランゲージリファレンス』は、4th Dimension言語を記述する際のリファレンスガイドです。4th Dimension言語を使ってデータベースをカスタマイズする方法を学習する時に、このマニュアルを利用してください。

『4th Dimension / 4D First デザインリファレンス』は、「デザイン」モード内で有効な操作を詳しく説明した「デザイン」モードのリファレンスガイドです。パッケージ内の他の解説書とともにご利用してください。

『4th Dimension / 4D First ユーザリファレンス』は、「ユーザ」モードに関する全情報を提供します。「ユーザ」モードとは、データを登録したり操作したりするデータベースやレイアウトを使用するモードです。

『4th Dimension / 4D First クイックスタート』は、実践演習をこなしながら、4th Dimensionのデータベースを作成および使用していきます。例題は、4th Dimensionと4D Firstの概念と機能を熟知できるように、簡単な体験学習方式になっています。

『4D Server リファレンス』は、4D Serverをインストールしたり、4D Serverを使ってマルチユーザデータベースを管理したりする際のリファレンスガイドです。このマニュアルは、4D Serverパッケージの中にのみ含まれています。

『4th Dimension インストールガイド』は、4th Dimensionをインストールするための手引書です。

『4th Dimension / 4D Firstユーティリティガイド』は、4D Tools、Customizer Plus、4D Transporterといった4th Dimensionと4D Firstに提供されるユーティリティの手引書です。

『4th Dimension / 4D First 用語集』は、バージョン6で変更された4th Dimension（4D First）用語を掲載しています。

その他に、バージョンアップ等で新しく追加／修正された情報はオンラインドキュメントで提供されています。このドキュメントは、4th Dimension（4D First）をインストールする際にハードディスク上にインストールされます。

このマニュアルについて

この『4th Dimension ランゲージリファレンス』マニュアルは、4th Dimensionのプログラミング言語を詳細に説明したものです。このマニュアルでは、テーブル、フィールド、フォームなどの用語をすでに習得していることを前提としています。このマニュアルを進む前に次のようなことを行ってください。

■『4th Dimension / 4D Firstクイックスタート』マニュアルを使って、例題のデータベースを実行してみる。

■ 自分自身でデータベースを作成してみる。必要に応じて、『4th Dimension / 4D Firstデザインリファレンス』マニュアルを参照する。

■「ユーザ」モードで作成したデータベースを使用してみる。「ユーザ」モードに関する詳細は『4th Dimension / 4D Firstユーザリファレンス』を参照する。

なお、このマニュアルは、4th Dimensionおよび4D ServerのWindowsとMacintoshユーザの両方を対象にしたクロスプラットフォームのマニュアルです。WindowsとMacintoshにおいて明らかに内容が異なる（画面、キーボード操作等）場合にのみ両方の説明を併記しています。それ以外はWindows版を中心に本文は記述されています。また、本文中で使用されているWindows版のスクリーンショットは「Windows 95」を基に作成しています。そのため、WindowsNT上で使用している場合、本文中の画面と多少異なる箇所がありますが、あらかじめご了承ください。

理解を深めるために

このマニュアルを含め、パッケージ中の全マニュアルではより内容を理解できるように一定のマークを使用しています。

次のようなマークが使用されています。

注：4th Dimensionをより豊富に使用出来るように、このような強調文で注釈やショートカットを提供します。

4D Server：マニュアルを通して、4th Dimension、4D Server／4D Clientは単に4th Dimensionと呼びます。2つの製品の操作の違いは、この4D Serverマークの中で説明されています。4D Serverマークは4D Server/4D Client の使用方法に関する情報を提供しています。この情報は、4D Server／4D Clientの操作が4th Dimensionと異なる部分のみ提供します。

このような注意書きは、重要な情報に対して注意を促しています。

警告：このような警告は、データが失われる可能性のある状況に対して注意を促しています。

また、このマニュアルはメソッドやコマンドを識別するために、次のような表記規則を使用します。

■ メソッドの例やコマンドは、次のように表記します。
例：**Piece of Code**

■ メソッドの例では、コマンドは太字（ボールド）で表記します。
例：**ADD RECORD**

■ 値を返さないコマンドは、すべて大文字で表記します。
例：**DEFAULT FILE**

■ 値を返すコマンド(関数)は、頭文字のみ大文字で表記します。
例：**Records in file**

■ プロジェクトメソッドは、斜体（イタリック）で表記します。
例：***My Proc***

■ 外部ルーチンは、太字斜体（ボールドイタリック）で表記します。
例：***My External***

■ コマンドの引数は、＜＞で囲んで表記します。
例：＜テーブル＞

この章では、4th Dimensionのプログラミング言語を紹介します。次のような事柄について説明します：

- プログラミング言語とは何か、また何ができるのか
- メソッドの使用方法
- 4th Dimensionを使ったアプリケーション開発

ここでは、一般的な概要についてのみ述べることにして、詳細については章の終わりで説明します。

プログラミング言語とは何か

4th Dimensionのプログラミング言語は、英会話に似ています。つまり、概念を表したり、情報を与えたり、指示するためのコミュニケーション手段の役割をします。会話体の言語と同じように4th Dimensionも独自の語彙、文法、構文を持っています。プログラミング言語は、4th Dimensionに対してデータベースとデータの処理方法や管理方法を指示するために使用します。

プログラミング言語についてすべてを知る必要はありません。これは、話すために単語をすべて知る必要がないことと同じです。実際に、少ない語彙しかなくても雄弁に話すことができるように、プログラミング言語をすべて知らなくても立派なデータベースを作成することができます。開発するために知る必要があるのは、プログラミング言語のごく一部だけです。あとは必要に応じて習得すれば、それで十分です。

プログラミング言語の使用目的

4th Dimensionにプログラミング言語はほとんど必要ありません。あらかじめ、「デザイン」モードと「ユーザ」モードにプログラミングを必要としない柔軟性のあるツールが用意されているので、さまざまなデータ管理処理を実現することができます。データ入力、検索、ソート、レポートなどの、基本的な処理を簡単に実行するのはもちろんのこと、データチェック、データ入力支援、グラフやラベル作成などの多くの機能をプログラミング言語を用いずに使用することができます。

それでは、どうしてプログラミング言語が必要なのでしょうか？それには、いくつかの理由があります。例えば、以下のような場合にプログラミング言語が必要になります：

■**「反復処理の自動化」：**データ修正、複雑なレポートの作成、一連の操作の自動実行

■**「ユーザインタフェースの管理」：**ウインドウ管理、メニュー管理、フォーム管理、インタフェースオブジェクト管理

■**「上級機能によるデータ管理の実行」：**トランザクション処理、複雑なデータチェック、マルチユーザ管理、セット操作、命名セレクションの操作

■**「コンピュータの管理」：**シリアルポート通信、文書管理、エラー管理

■**「データベースの作成」：**「カスタム」モードで使用するカスタマイズされたデータベースの作成

■**「内蔵4D Webサービスへの機能の追加」：**4Dがフォームから自動的に変換するページに加えて、ダイナミックHTMLページの作成

4th Dimensionのプログラミング言語は、データベース処理を制御するためのものです。「ユーザ」モードは強力な生成ツールを備えていますが、必要に応じて4th Dimensionのプログラミング言語を使用することにより、データベースをカスタマイズすることができます。

データ制御の実施

4th Dimensionは、強力かつ柔軟にデータを制御します。このプログラミング言語は初心者にも簡単に使用することができます。また、アプリケーション開発者の高度な要求にも十分に対応することができます。データベース内蔵の制御からカスタマイズされたデータベースへスムーズに移行することもできます。

4th Dimensionのコマンドは、「ユーザ」モードで同じ操作をほとんど行うことができます。例えば、**QUERY**コマンドを使用すると「クエリ」エディタが表示されますが、これは「ユーザ」モードの「クエリ」メニューから「検索」メニューコマンドを選択することと同じです。また、この **QUERY**コマンドは特定のデータを検索することもできます。例えば、**QUERY** ([従業員] ; [従業員]名前 = "山田")はデータベースの中から “ 山田 ” という名前のすべての人を検索します。

このプログラミング言語は非常に強力です。1つのコマンドで、従来のコンピュータ言語の何百あるいは何千ものステップに相当するものもあります。また、柔軟性に富み、強力であると同時にコマンドは平易な英語名を使用しているので非常に扱いやすくなっています。例えば、新しいレコードを追加するためには **ADD RECORD**コマンドを使用するだけでよいのです。

この言語は、ほとんどの標準的なデータ処理を簡単に実行できるように設計されています。レコードの追加やテーブルのソート、データの検索などの操作は単純で直接的なコマンドで指定できます。

従来のコンピュータ言語との相違点

従来のコンピュータ言語に馴れ親しんでいる方は、本章を参照してください。それ以外の方は、この章を読み飛ばしても構いません。

4th Dimensionのプログラミング言語は従来のコンピュータ言語とは異なります。この言語は、今日のコンピュータで使用できる最も先進的で柔軟性のある言語の1つです。4th Dimensionのプログラミング言語は、その言語自体で機能するように設計されています。

従来の言語を使用して開発を行う場合、まず広範な計画を立てる必要があります。そして計画の立案も開発の重要な工程の1つとなります。しかし、4th Dimensionはデータベースのあらゆる部分でプログラミング言語をいつでも使用することができます。例えば、フォームにオブジェクトメソッドを追加し、その後で1つまたは2つのメソッドを追加することができます。またデータがより高度になった場合でも、メニューから制御するプロジェクトメソッドを追加することもできます。つまり、4th Dimensionのプログラミング言語は、他の多くのデータベースのようにプログラミング言語が“すべてか、あるいは全くないか”ではなく、必要最低限のプログラミング言語を使用するだけでいいのです。

従来の言語の構文では、オブジェクトを定義または事前に宣言(定義)しなければなりませんでした。しかし、4th Dimensionではオブジェクトを作成し、それを使用するだけで構いません。4th Dimensionは自動的にオブジェクトを管理します。例えば、ボタンを使用するためには、ボタンをフォーム上に作成し、それを指定します。ユーザがボタンをクリックした時点で、そのことをメソッドに自動的に通知します。

従来の言語では、コマンドの使用を固定化したり限定するなどして融通性に欠ける点が多いのに対して、4th Dimensionのプログラミング言語は優れたユーザインタフェースを実現しています。

メソッドはプログラミング言語の入口

メソッドは、4th Dimensionに処理を実行させるための一連の命令文です。メソッド内の各行を、“ ステートメント ” と呼びます。各ステートメントは、プログラミング言語の一部分(コマンドなど)で構成されます。

ここでは、すでに『4th Dimension / 4D First クイックスタート』を通読し、オブジェクトメソッドやプロジェクトメソッドを作成し使用した経験があることを前提に説明を行います。

4th Dimensionで使用するメソッドには、オブジェクトメソッド、フォームメソッド、プロジェクトメソッドの3種類があります。

■**「オブジェクトメソッド」：**フォームオブジェクトを制御するために使用する短いメソッド

■**「フォームメソッド」：**フォームの表示を管理するメソッド

■**「テーブルメソッド/トリガ」：**データベースの規則を強制するためのメソッド

■**「プロジェクトメソッド」：**データベース全体で使用できるメソッド

■**「データベースメソッド」：**データベースのオープンやクローズのとき、またはWebブラウザがインターネットおよびイントラネット上でWebサーバとして発行されているデータベースに接続するときに、初期化や特別な動作を行うメソッド

次の節では、各メソッドの紹介とデータベースを自動化する方法について説明します。すべてのメソッドタイプの詳細は、このマニュアルの第3章で紹介します。

オブジェクトメソッド入門

動作を実行できるフォーム(すなわち、アクティブオブジェクト)は、それに関連するメソッドを持つことができます。オブジェクトメソッドは、データ入力時や印刷時にアクティブオブジェクトの監視や管理を行います。オブジェクトをコピーして貼り付けると、オブジェクトメソッドがそのアクティブオブジェクトとともにコピーされます。これにより、作成したオブジェクトの再利用可能なライブラリを作成することができます。オブジェクトメソッドはまさに必要な場合のみ制御を行います。

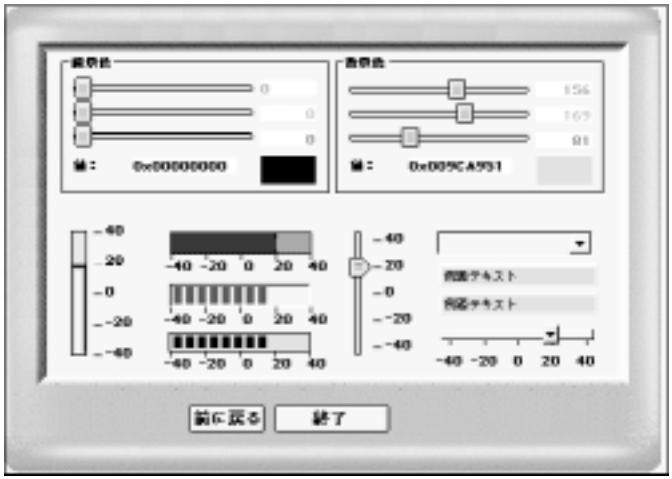
オブジェクトメソッドは、データベースへの入り口であるユーザインタフェースを管理するための主要ツールです。ユーザインタフェースは、コンピュータがユーザと通信するための手順と規則から成り立っています。その最終目的は、データベースのユーザインタフェースをできるだけ簡単で使いやすくすることです。ユーザインタフェースは、コンピュータとのやり取りを快適にし、ユーザがそれを楽しめるように、または気にならないようにする必要があります。

フォームには、次の2つの基本的なタイプのアクティブオブジェクトがあります：

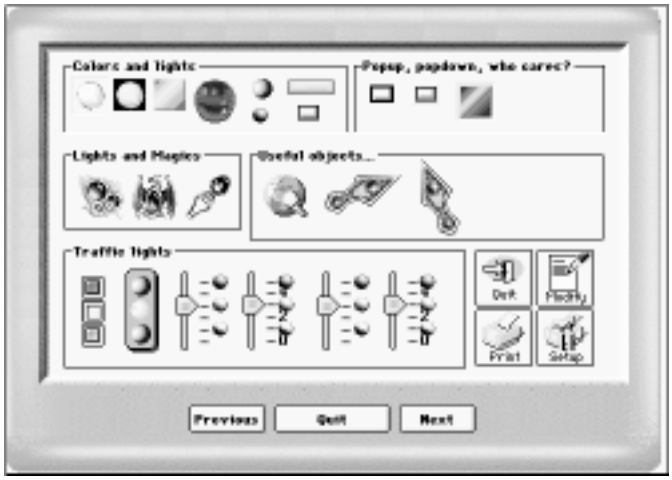
■ フィールドやサブフィールドなどのデータの入力、表示、格納のためのアクティブオブジェクト
■ 入力エリア、ボタン、スクロールエリア、階層化リスト、メータなどのコントロールのためのアクティブオブジェクト

4th Dimensionでは、以下のようなクラシックフォームを作成することができます。

また、以下のような複数のグラフィックコントロールを使用してフォームを作成することができます。



想像力が許す限りのグラフィックを使用したフォームを作成することもできます。



作成するスタイルがどのような形式であろうと、アクティブオブジェクトはすべてデータ入力エリアの範囲チェックや入力フィルタ、コントロール、メニュー、ボタン用の自動動作等の内蔵の支援機能を持っています。オブジェクトメソッドを追加する前には、必ずこれらの支援機能を使用します。内蔵の支援機能はメソッドと同様に、アクティブオブジェクトに関連付けられ、それがアクティブなのはアクティブオブジェクトが使用されるときだけです。通常、内蔵の支援機能とオブジェクトメソッドを組み合わせて使用して、ユーザインタフェースを制御します。

データ入力に使用するオブジェクトメソッドは、フィールドまたは変数に対して特定の処理を実行します。このオブジェクトメソッドは、データの属性チェック、フォーマット編集、計算処理、別テーブルのデータ取得などを行います。もちろん、これらの処理の一部は、4th Dimensionから提供されているデータ入力時のオブジェクトツールで実行することもできます。しかし、複雑な処理が必要な場合には、オブジェクトメソッドを使用します。このツールに関する詳細は、『4th Dimension / 4D First デザインリファレンス』を参照してください。

オブジェクトメソッドは、制御に使用するボタンなどのオブジェクトにも作成することができます。これらのオブジェクトは、データベースを使用する上でとても重要です。これには、レコード間の移動や別フォームの使用、データの追加、修正、削除などを行うボタンがあります。また、これらのオブジェクトは、データベースの使用を簡単にし、それを習得するために必要な時間を減少させます。ボタンにも、データ入力時と同様に4th Dimension内に組み込まれているツールの使用が可能です。オブジェクトメソッドを作成する前に、これらの組み込みツールを使用してください。オブジェクトメソッドを使用すれば、4th Dimension内に組み込まれていない動作を実現することができます。例えば、次の図は、クリックされた時点で「クエリ」エディタを表示するボタンのオブジェクトメソッドを示しています。

オブジェクトメソッドに精通してくると、オブジェクトメソッドを持つオブジェクトのライブラリを作成すると便利なことに気付きます。これらのオブジェクトやオブジェクトメソッドをフォーム、テーブル、データベース間でコピーしたり、貼り付けることができます。また、必要な場合に使用できるように、スクラップブックにそれらを保存することもできます。

フォームメソッドを使ってフォームを制御する

オブジェクトメソッドがフォームオブジェクトに付着するのと同様に、フォームメソッドはフォームに付着します。1つのフォームには、1つのフォームメソッドしか持つことができません。

フォームはデータを表示します。フォームを使用することで、魅力的で使いやすいデータ入力画面やレポートを作成することができます。フォームメソッドはデータの入出力におけるフォームをチェックし、制御します。

フォームメソッドは、オブジェクトメソッドよりも上位のフォームを管理します。オブジェクトメソッドは属しているオブジェクトが使用された場合にのみ実行されます。フォームメソッドは、フォーム上のオブジェクト全体を制御するために使用します。つまり、オブジェクトメソッドが実行されるたびに、フォームメソッドも実行されます。

フォームは非常に多くの用途に用いられるので、フォームには、フォームが使用されている状況をチェックするための“フォームイベント”が用意されています。このフォームイベントは、いくつかのフェーズに分かれています。それぞれのフェーズはフォームの状況が変わると発生します。フォームイベントに関する詳細は、第5章および第24章で説明します。

トリガを使ってデータベースの規則を制御する

トリガはテーブルに設定されるので、テーブルメソッドとも呼ばれます。トリガは、テーブルのレコードを操作(追加、削除、修正、ロード)するたび、4Dデータベースエンジンによって自動的に起動されます。トリガはデータベースのレコードに対する「違法な」操作を防止することができます。例えば、請求書発行システムでは、請求書発行先の顧客を指定しない請求書を追加できないようにすることができます。トリガはテーブル上の操作を制限する非常に強力なツールであると同時に、偶発的なデータの損失や不正な変更を防止します。最初は簡単なトリガを作成しておいて、後で複雑にしていくこともできます。

トリガについての詳細は、第57章の「トリガコマンド」を参照してください。

データベース全体でプロジェクトメソッドを使用する

特定のフォームに付随するフォームメソッドや、特定のオブジェクトに付随するオブジェクトメソッドと異なり、プロジェクトメソッドはデータベースのどこからでも使用することができます。また、プロジェクトメソッドは、繰り返して使用したり、別のメソッドの中から呼び出して使用することもできます。同じ処理を繰り返し行う必要がある場合でも、それぞれに対して同一のメソッドを書く必要はありません。

プロジェクトメソッドは、別のプロジェクトメソッドやオブジェクトメソッド、フォームメソッドから必要な場所で随時呼び出されます。呼び出されたプロジェクトメソッドは、呼び出した場所にメソッド全体を書き込んだ時と同じように動作します。別のメソッドから呼び出されるプロジェクトメソッドのことを“ サブルーチン ”と呼びます。
プロジェクトメソッドを使用するもう1つの方法として、メニューコマンドへの割り当てがあります。メニューコマンドに割り当てられたメソッドは、メニューコマンドを選択した時に実行されます。データベース内のメニューから呼び出されるメソッドを“ マスターメソッド ”と呼びます。このメソッドはデータベースの操作全体を制御します。

データベースメソッドによる作業セッションの取り扱い

フォームにイベントが発生したときに、オブジェクトメソッドとフォームメソッドが起動するのと同様に、作業セッションイベントの発生時に起動されるデータベースに関連付けるメソッドがあります。これがデータベースメソッドです。例えば、データベースをオープンするたびに、作業セッション全体で使用する変数をいくつか初期化したい場合があります。これには、データベースをオープンしたときに4Dが自動的に実行する「On Startup」データベースメソッドを使用します。

データベースメソッドについての詳細は、第6章の「データベースメソッド」の節を参照してください。

データベースを作成する

データベースを作成することは、プログラミング言語や4th Dimension内に用意されている内蔵ツールを使用してデータベースをカスタマイズするということです。

つまり、データベースを作成するだけで、プログラミング言語の第一歩を踏み出したことになります。データベースのテーブル、フィールド、フォーム、オブジェクト、メニューの各部分は、プログラミング言語と密接に関連しています。プログラミング言語は、データベースのこれらの部分について、すべて“熟知”しています。

プログラミング言語を初めて使用する場面は、おそらくデータ入力を制御するために、フォームオブジェクトやオブジェクトメソッドを作成する場合です。そして、フォームの表示を制御するために、フォームメソッドを作成します。大規模なデータベースの場合、データベースを完全にカスタマイズするには、プロジェクトメソッドとメニューバーを作成する必要があります。

4th Dimensionは、その他についても非常に柔軟性があります。データベースの作成に、決まった方法があるわけではありません。自分自身に合った方法で作成することができます。もちろん、次のようないくつかの一般的なパターンはあります。

■**「実行」：**「デザイン」モードで設計したものを実行する。

■**「テスト」：**「ユーザ」モードで設計したものをテストしたり、カスタマイズしたデータベースを使ってみる。

■**「使用」：**カスタマイズしたデータベースが完成したら、「カスタム」モードで使ってみる。

■**「訂正」：**エラーを見つけたら、「デザイン」モードに戻って、そのエラーを訂正する。

データベース作成に、特別な支援ツールが用意されています。これらは4th Dimension内部に組み込まれており、必要になるまでは隠れていますが、プログラミング言語の使用に熟知するにつれて、これらのツールが開発プロセスを容易にしてくれることがわかってきます。例えば、「メソッド」エディタは入力エラーを検知したり、シンタックスのネストレベル(入れ子の階層)を表示します。また、インタプリタ(4th Dimension内部に組み込まれたプログラミング言語の実行制御システム)はシンタックス中のエラーを取り出し、どこが誤りかを示します。デバッガは、メソッド中のエラーを見つけるために、実行中のメソッドをリスト表示します。

アプリケーションを構築する

これで、データ入力、検索、ソート、レポートなどのデータベースの一般的な使用方法について理解されたと思います。これらの処理は、4th Dimension内に組み込まれているメニューとエディタを使用して「ユーザ」モードで実行してきました。

データベースを使用するにつれて、繰り返して実行する一連の処理があることが明らかになります。例えば、個人連絡先のデータベースにおいて、データを変更するたびに、仕事上の知人の検索、名字でソート、特定のレポートを印刷すると仮定します。この場合、これらの処理はとても単純ですが、この処理を20回以上も行うとなると、とても大変です。また、数週間データベースを使用しなかった場合には、レポートを作成する手順を克明に憶えているとも限りません。しかし、メソッドを作成すれば、1つのメニューコマンドを選択するだけで、一連の処理を自動的に実行してくれるのでレポートを作成するための手順を忘れていても大丈夫です。

アプリケーションは、データベースを自動化します。アプリケーションは、データベースを使用するユーザのニーズに合ったカスタムメニューや、特殊な実行処理用のメニューを持つことができます。アプリケーションは、データベースの要素(ストラクチャ、フォーム、オブジェクトメソッド、メソッド、メニュー、パスワード等)から構成されます。

アプリケーションには、人の名前を入力して、レポートを印刷するだけの単純なものから、請求・在庫管理システムといった複雑なものまであります。アプリケーションの使用に関しては特に制限事項のようなものはありませんが、一般的にデータベースは、「ユーザ」モードで使用するものからカスタムメニューで完全に管理する大規模なものまでさまざまです。

アプリケーションの開発は、ユーザの思うままに複雑にも簡単にもすることができます。アプリケーション構築における操作手順に関する詳細は、第7章を参照してください。

プログラミング言語の構成要素　2

4th Dimensionのプログラミング言語は、次のような要素から構成されています。これらの要素は、処理を実行したりデータを管理する際の手助けをします。

- **データタイプ：**データベースでのデータの分類
- **変数：**データ用のメモリ内に入れられる一時的な記憶領域
- **演算子：**2つの値の間で計算をする記号
- **式：**値を求める構成要素の組み合せ
- **コマンド：**処理を実行するために、4th Dimensionに組み込まれている命令
- **メソッド：**プログラミング言語を使用して作成した命令の集まり

この章では、変数、演算子、式について説明します。

次の章でメソッドの各タイプについて説明します。

また、残りの章で、プログラミング言語の各コマンドについて説明します。

データタイプ

4th Dimensionのデータベースで使用するデータには、いくつかの種別があります。これらのデータ種別を“ データタイプ ”と呼びます。

4th Dimensionには、次のような8つのデータタイプがあります。

■ **文字列：**"こんにちは"などの一連の文字
文字フィールドとテキストフィールドは、ともに文字列タイプです。

■ **数値：**2や1,000.67などの数値
整数フィールド、倍長整数フィールド、実数フィールドはすべて数値タイプです。

■ **日付：**97.07.20などの日付
日付フィールドは、日付タイプです。

■ **時間：**1:00:00や4:35:30 P.M.などの時:分:秒
時間フィールドは、時間タイプです。

■ **ブール：**“ True(真) ”または“ False(偽) ”のどちらかの値を返すもの。
ブールフィールドは、ブールタイプです。

■ **ピクチャ：**PICTタイプで作成されたピクチャ
ピクチャフィールドは、ピクチャタイプです。

■ **BLOB (Binary Large OBject)：**サイズが2GB以内のバイト群
BLOBフィールドは、BLOBタイプです。

■ **ポインタ：**上級プログラミングで使用する特殊タイプのデータ
対応するフィールドはありません。

注：これらのデータタイプに格納されるデータは、『4th Dimension / 4D Firstデザイインリファレンス』の中で説明されている4th Dimensionフィールドの中に格納されるデータと一致します。例外は、255バイトまで含むことができる文字列変数です。

データタイプの中で文字列タイプと数値タイプが、複数のフィールドタイプを伴う点に注意してください。プログラミング言語でこれらのタイプのフィールドを操作する場合、プログラミング言語が自動的にサポートするデータタイプにデータを変換します。例えば、整数フィールドを使用すると、自動的に数値として扱います。つまり、プログラミング言語として使用する場合は、類似するフィールドタイプと厳密に区別する必要はありません。

しかし、プログラミング言語を使用する場合は、異なるデータタイプと混同しないようにすることは重要です。“ ABC ”を日付フィールドに格納しても意味がないように、日付として使用する変数に“ ABC ”を格納することも意味がありません。4th Dimensionは、実行したことをできるだけ有効にしようとします。例えば、日付に数値を加算した場合は、

日付に日数を加算したいものと認識します。しかし、日付に文字列を加算した場合には、4th Dimensionはその操作に意味を持たないことをユーザに警告します。

あるタイプとしてデータを格納したものを、別のタイプとしてそれを使用する場合があります。4th Dimensionのプログラミング言語には、あるタイプから別のタイプへ変換するためのコマンドが用意されています。

例えば、数値で始まり、“ abc ” などの文字で終了する部品番号を作成するような場合、次のように記述することができます。

部品番号:= String (数値) + "abc"

“ 数値 ” が17であれば、部品番号に “ 17abc ” という文字列が代入されます。

演算子

プログラミング言語を使用する際に、データのみを必要とする場合は非常に稀です。データを加工したり、それを何らかの目的のために使用することがほとんどです。例えば、演算子を使用して2つのデータから1つの新しいデータを生成する場合です。1+2は2つの数値を加算(演算子+)して、3という結果を返します。以下に、よく知られている4つの演算子を示します。

演算子	実行される演算	例
+	加算 (足し算)	1 + 2 の結果は3になります。
-	減算 (引き算)	3 - 2 の結果は1になります。
*	乗算 (掛け算)	2 * 3 の結果は6になります。
/	除算 (割り算)	6 / 2 の結果は3になります。

数値演算子は、使用可能な演算子のうちの1つのタイプにすぎません。

4th Dimensionは、数値、テキスト、日付、ピクチャなど異なるタイプのデータを扱うために、各データタイプに対する演算を行うための演算子を備えています。

同じ記号でも処理を行うデータタイプによって、異なる意味に使用される場合があります。例えば、プラス記号(+)は下記のように、各データタイプによって異なる演算を実行します。

データタイプ	実行される演算	例
数値	加算	1+2は数値を加算し、結果は3になります。
文字列	連結 (結合)	"みなさん　"+"こんにちは"は、文字列を連結し、結果は"みなさん　こんにちは"になります。
日付と数値	日付の加算	!97.07.01!+20は、1997年7月1日に20日を加算し、結果は!97.07.21!(1997年7月21日)になります。

式

式は、値を返します。実際に、プログラミング言語として使用している場合は、常に式を使用します。式は、“ フォーミュラ ” と呼ぶこともあります。

式は、コマンド、演算子、変数、フィールドなどプログラミングのほとんどの部分で使用します。式により、ステートメント(メソッドの1文...1行)を構成します。必要に応じて、メソッドの任意の場所に式を使用することができます。

式が “ 独立型 ” であることはほとんどありませんが、独立して使用できる場合がいくつかあります。「ユーザ」モードの「フォーミュラで検索」ダイアログボックスの場合、デバッガで式の値をチェックする場合、「フォーミュラで更新」ダイアログボックスの場合、およびカラムをフォーミュラとして使用する「クイックレポート」エディタの場合です。

数値4または文字列 “ こんにちは ” などの定数だけで構成された式の値は、常に一定で、決して変わることはありません。

演算子を使用すると、式でさまざまなことを行うことができます。前節で、演算子を使用する式をすでに紹介しました。例えば、4+2は加算演算子を使用した式で2つの数値を加算し、結果に6を返します。

データタイプによって、次の7つのタイプの式を使用することができます：

- 文字列式
- 数値式
- 日付式
- 時間式
- ブール式
- ピクチャ式
- ポインタ式

以下の表に、7つのタイプの式ごとに例を示します。演算子の詳細については、第11章を参照してください。

式	タイプ	説明
"みなさん　"	文字列	これは、文字列定数の"みなさん　"です。これが文字列定数であることを示すために2重引用符を使用している点に注意してください。
"みなさん　"+"こんにちは"	文字列	2つの文字列"みなさん　"と"こんにちは"を、文字列連結演算子(+)で連結して、文字列 "みなさん　こんにちは"を返します。
[従業員]名前+"さん"	文字列	名前フィールドの現在の値と"さん"の2つの文字列を連結します。名前フィールドの値が"スミス"の場合は、"スミスさん"を返します。
Uppercase ("smith")	文字列	この式は、文字列"smith"を大文字に変換するために、Uppercase関数を使用して"SMITH"を返します。
4	数値	これは、数値定数4です。
4*2	数値	掛け算演算子を使用して、2つの数値4と2が乗算します。結果は数値8になります。
私のボタン	数値	これは、ボタンの名前でボタンの現在の値を返します。クリックされた場合は1を、クリックされない場合には0を返します。
!97.07.21!	日付	これは、日付定数!97.07.21!(1997年7月21日)です。感嘆符を使用して、日付定数であることを示します。

式	タイプ	説明
Current date+30	日付	これはCurrent date関数を使用して、現在の日付を得るための日付式です。これは現在の日付に30日を加算した新しい日付を返します。
?8:05:30?	時間	これは、時間定数?8:05:30?(8時間5分30秒)です。
?2:03:04?+?1:02:03?	時間	これは、2つの時間を加算し、?3:05:07?を返します。
True	ブール	これは、ブール値 “ True(真) ” を返すコマンドです。
10#20	ブール	これは、2つの数値の間の論理比較です。この数値記号(#)は “ 等しくない ” ことを意味します。10と20は、等しくないので “ True(真) ” を返します。
"ABC"="XYZ"	ブール	これは、2つの文字列の間の論理比較です。結果は等しくないので、式は “ False(偽) ” を返します。
私の絵+50	ピクチャ	これは、私の絵というピクチャを右へ50ピクセル移動します。
->[従業員]名前	ポインタ	この式は、ポインタを[従業員]名前というフィールドに返します。
Table (1)	ポインタ	これは、最初のテーブルにポインタを返すコマンドです。

変数

4th Dimensionのデータは、フィールドまたは変数に格納されます。フィールドがディスクにデータを保存するのに対して、変数はメモリ上にだけデータを格納します。データベースを作成する際、フィールドに名前とデータタイプを指定するのと同様に、変数にも名前とデータタイプを指定します。

次の変数タイプは、各データタイプと一致します：

- **文字：**最大255バイトまでの固定文字列
- **テキスト：**最大32000バイトまでの文字
- **整数：**-32,768から32767までの整数
- **倍長整数：**-2^{31}から2^{31-1}までの整数
- **実数：**±1.7e±308の数字
- **日付：**100.1.1から32767.12.31
- **時間：**00:00:00から596000:00:00
- **ブール：**TrueまたはFalse
- **ピクチャ：**WindowsまたはMacintoshのピクチャ
- **BLOB (Binary Large OBject)：**サイズが2GB以内のバイト群
- **ポインタ：**ファイル、フィールド、変数、配列、配列要素のポインタ

変数は、画面上への表示やフォームからのデータ入力、レポートの印刷を行うことができます。また、フィールドと同様に、次のような4th Dimension内部に用意された組み込み制御ツールを使用することができます：

- データフォーマット
- データ範囲チェック
- 入力フィルタ
- 選択項目リスト、指定項目リスト、除外項目リスト
- 入力可属性または入力不可属性

また、変数は次のような特殊なこともできます：

- ボタン(ボタン、チェックボックス、ラジオボタン、3Dボタンなど)の制御
- スライダ(メータ、ルーラ、ダイアル)の制御
- スクロール可能エリア、ポップアップメニュー、ドロップダウンリストボックスの制御
- 階層リスト、階層ポップアップメニューの制御
- ボタングリッド、タブコントロール、ピクチャボタンなどの制御
- 保存の必要がない計算結果の表示

変数の作成

変数は、作成するだけで使用することができます。フィールドのように正式に定義(宣言)する必要はありません。例えば、現在の日付に30日を加えたものを変数に保存したい場合は、次のように定義します：

MyDate:=Current date + 30

この式は、現在の日付に30日を加えたものを変数“ MyDate ” に代入することを意味します。また、次のようにフィールド“ MyField ”に変数“ MyDate ”の値を代入することもできます：

MyField:=MyDate

また、変数はデータタイプを明確にするために使用される場合もあります。

注：このマニュアル上では、Current date関数などのようにコマンドや関数が太字（ボールド）で表記されています。これは、このマニュアルの表記規則に従っているためです。

変数にデータを代入する

変数にデータを格納したり、変数からデータをコピーすることができます。変数に対してデータを代入(格納)するには、代入演算子を使用します。代入演算子はフィールドに対してデータを代入する場合にも使用します。

代入演算子は、変数を作成したり、変数にデータを代入するために使用します。作成する変数名は代入演算子の左側に入力します。変数“ MyNumber ”を作成する例を次に示します。“ MyNumber ”に数値3を代入しています：

MyNumber:= 3

これで、数値3を値に持つ“ MyNumber ”という変数が作成されました。“ MyNumber ”が、すでに存在している場合には、数値3は単純に“ MyNumber ”に代入されます。

もちろん、変数からデータを取り出すことができなければ、便利とは言えません。再度代入演算子を使用してみましょう。“ Size ”というフィールドに“ MyNumber ”の値を代入する場合に、代入演算子の右側に“ MyNumber ”を記述します：

Size:=MyNumber

この“ Size ”の値は3になります。この例はとても単純ですが、データをある場所から別の場所へ転送する基本的な方法です。

代入演算子(:=)を比較演算子(=)と混同しないように注意してください。代入演算子と比較演算子はまったく異なる演算子です。比較演算子に関する詳細は、第11章を参照してください。

ローカル変数、プロセス変数、インタープロセス変数

4th Dimensionでは、ユーザはローカル変数、プロセス（グローバル）変数、およびインタープロセス変数の 3種類の変数を作成することができます。この3種類の変数の違いは、その使用できる「範囲」にあります。また、それらを使用することのできるオブジェクトによっても異なります。

ローカル変数

ローカル変数はその名のとおり、メソッド内でローカル（局所）に、つまり、作成されたメソッド内でのみアクセスすることができ、その他のメソッドからはアクセスできません。メソッド内でローカルであるというのは、「その範囲がローカルである」という意味です。ローカル変数は、その使用範囲をメソッド内に限定する場合に用います。

ローカル変数は、次のような目的のために使用されます：

- 他の変数名との重複を避ける。
- データを一時的に使用する。
- プロセス変数の数を減らす。

ローカル変数の名前は、必ずドル記号($)で始めます。

多くのメソッドや変数を持つデータベースで作業する場合、現在作業しているメソッド内でのみ使用可能な変数を必要とする場合がよくあります。この場合、同じ変数名が他で使用していないかどうかを気にすることなくローカル変数を作成することができます。

また、ユーザからのデータを必要とする場合があります。**Request**関数は、ユーザから1つのデータを取得します。**Request**関数は、ユーザにデータの入力を求めるダイアログボックスを表示し、ユーザがデータを入力した時点で、その入力情報を返します。このデータはメソッドに保存する必要はありません。これは、ローカル変数を使用するための良い例です。

例えば、次のメソッドは：

```
$応答:=Request ("従業員番号を入力してください。")
If (OK=1)
   QUERY ([従業員] ; [従業員]ID番号=$応答)
End if
```

ユーザに従業員のID番号を入力するように要求するものです。これはローカル変数 “ $応答 ” にID番号を代入し、それを基にユーザが入力した従業員ID番号を検索します。このメソッドが終了した時点で、“ $応答 ” というローカル変数は消去されます。この変数は1回しか必要としないのでローカル変数で十分です。

ローカル変数は、メソッド間のデータの受け渡しにも使用されます。

プロセス変数

プロセス変数は、1つのプロセス内でだけ使用可能です。この変数は、そのプロセス内で呼び出された他のメソッドで使用することができます。

プロセス変数は、名前の前に付ける接頭辞はありません。

インタープロセス変数

インタープロセス変数は、データベース全体で使用することができ、すべてのプロセスと共用します。

インタープロセス変数の名前は、必ずインタープロセス記号(<>)で始めます。例えば、変数「<>Color」はインタープロセス変数です。

インタープロセス変数は主に、プロセス間で情報を共有する場合に使用します。

フォーム上のオブジェクトと変数

「フォーム」エディタを使って、ボタン、スクロールエリア、サーモメータなどのオブジェクトを作成すると、それに付けた名前と同じ名前を持つ変数を自動的に生成します。例えば、“ MyButton ”という名前のボタンを作成すると、“ MyButton ”という名前の変数が自動的に生成されます。しかし、この変数名がこのボタンのラベル名ではなく、単にボタンの名前になっている点に注意してください。

この変数を使用してオブジェクトの制御、チェックを行うことができます。例えば、このボタンがクリックされた時点で、その変数に1が設定されます。それ以外の場合はすべて 0になります。サーモメータやダイアルに付随する変数は、現在の設定値を読み込んで、その設定値を変更することができます。例えば、サーモメータをドラッグした場合に、変数の値は新しい設定値に変更されます。同様にメソッドで変数の値を変更すると、サーモメータは新しい値を再表示します。

システム変数

4th Dimensionには、システム変数という特別な変数があります。この種類の変数を使用するとさまざまなデータをチェックすることができます。すべてのシステム変数が、データベースのどこでも使用できるプロセス変数です。

最も重要なシステム変数は、OKシステム変数です。名前が示すように、何かの事象がOKかどうかを判断します。例えば、レコードは保存されたか、操作は終了したか、ユーザがOKボタンをクリックしたかなどを判断します。システム変数OKは、処理が完了した場合には1が設定され、処理がうまく終了しない場合には 0が設定されます。

システム変数に関する詳細は、このマニュアルの付録Aを参照してください。

メソッドの使用方法　3

コマンドや演算子がプログラミング言語として機能するには、それらをメソッド内に配置する必要があります。この章では、すべての種類のメソッドの共通機能について説明します。メソッドには、トリガ（テーブルメソッド）、フォームメソッド、プロジェクトメソッド、データベースメソッドがあります。オブジェクトメソッドも特別なメソッドの1つです。

メソッドは、ステートメントで構成されます。ステートメントとは、メソッドの1行のことで1つの命令を実行します。ステートメントは単純な場合もあれば、複雑な場合もあります。ステートメントは常に1行ですが最大32,000バイトまで使用することができます。これは、ほとんどの処理で十分な長さです。例えば、次の行は[従業員]テーブルに新しいレコードを追加するステートメントです：

```
ADD RECORD ([従業員])
```

メソッドは、テストとループも含んでいます。これらは、制御フローをコントロールしています。制御フローに関する詳細は、後述の「制御フロー」の節を参照してください。

メソッドの種類

4th Dimensionのメソッドには、次の4種類があります。

■**「オブジェクトメソッド」：**オブジェクトメソッドとは、任意のオブジェクトプロパティです。これは、通常、有効なオブジェクトフォームに関連する短いメソッドのことです。一般に、オブジェクトメソッドはそのフォームが表示または印刷されているときにオブジェクトを「管理」します。ユーザがオブジェクトメソッドを呼び出すことはありません。オブジェクトメソッドが属しているオブジェクトをイベントが含んでいる場合に、4th Dimension がオブジェクトメソッドを自動的に呼び出します。

■**「フォームメソッド」：**フォームメソッドとは、任意のフォームプロパティです。ユーザはフォームメソッドを使用してデータとオブジェクトの管理を行うことができます。ただし、上述の目的には、オブジェクトメソッドを使用するほうが通常は簡単であり、より効果的です。ユーザがフォームメソッドを呼び出すことはありません。フォームメソッドが属しているオブジェクトをイベントが含んでいる場合に、4th Dimension がフォームメソッドを自動的に呼び出します。

オブジェクトメソッドとフォームメソッドの詳細に関しては、『4th Dimension / 4D First デザインリファレンス』および第4章を参照してください。

■**「トリガ（テーブルメソッド）」：**トリガとは、任意のテーブルプロパティです。ユーザがトリガ を呼び出すことはありません。ユーザがテーブル (追加、削除、修正、読み込み) のレコードを操作する度にトリガは 4D データベースエンジンによって自動的に呼び出されます。トリガはユーザのデータベースのレコードに対して「不正な」オペレーションが行われることを防ぎます。例えば、「送り状」システムでは、送り状を送付した顧客名を明記せずに第三者が送り状を追加することを防ぎます。トリガはとても強力なツールで、テーブル上のオペレーションの制限を行うことができます。同様に、思いがけないデータの損失や偽造も防ぐことができます。ユーザは簡単なトリガなら作成できます。そしてこれを徐々に精巧にしていくことができます。

トリガについての詳細は、第57章の「トリガコマンド」を参照してください。

■**「プロジェクトメソッド」：**オブジェクトメソッド、フォームメソッド、トリガはすべて特定のオブジェクト、フォーム、テーブルとそれぞれ関連があります。これとは異なり、プロジェクトメソッドはユーザのデータベースを通して使用可能となります。プロジェクトメソッドは再利用でき、他のメソッドによる使用が可能です。作業の反復が必要になった場合でも、ユーザは状況ごとに相応なメソッドを書く必要はありません。ユーザは必要に応じていつでもプロジェクトメソッドを他のプロジェクトメソッド、フォームオブジェクト、フォームメソッドから呼び出すことができます。ユーザがプロジェクトメソッドを呼び出すと、プロジェクトメソッドはユーザが呼び出した場所でユーザが作成したようにスムーズに機能します。他のメソッドから呼び出されたプロジェクトメソッドは「サブルーチン」として参照されることがよくあります。結果を返すプロジェクトメソッドは、関数とも呼ばれます。

プロジェクトメソッドには、メニューコマンドと結合させるというもう1つの使用方法があります。ユーザがプロジェクトメソッドをメニューコマンドと結合させると、そのメニューが選択されたときにそのメソッドが実行されるようになります。これでメニューコマンドをプロジェクトメソッドの呼び出しと同じように考えることができます。

プロジェクトメソッドに関する詳細は、第6章を参照してください。

■**「データベースメソッド」：**フォーム内でイベントが生じるときにオブジェクトとフォームのメソッドが呼び出されるのと同様に、作業セッションのイベントが生じると呼び出されるデータベースと連結するメソッドがあります。これが、データベースメソッドです。例えば、ユーザがデータベースを開く渡に、作業セッションの間使用する変数をいくつか初期化したいとします。このために、ユーザは「On Startup」データベースメソッドを使用します。これはユーザがデータベースを開くときに、4D によって自動的に実実行されます。

データベースメソッド関する詳細は、第6章を参照してください。

■**「外部ルーチン（プラグイン）」：**外部ルーチンは、4th Dimension以外で作成したメソッドです。外部ルーチンは一般にPascalやC言語で作成されます。このメソッドはコンパイルされた後、4th Dimensionのアプリケーションでの使用が可能になります。異なる機能を持った外部ルーチンを4th Dimensionに追加することができます。

外部ルーチン以外のメソッドは、4th Dimensionの「メソッド」エディタを使用して作成します。

4th Dimensionバージョン3との互換性

4th Dimensionのバージョン6 で新しく作成されたデータベースで作業している場合は、この互換性に関する注記は読み飛ばしてかまいません。

1. バージョン6 では、新しいオブジェクトとフォームイベント (On Double Clicked、On Getting Focusなど) が数多く提供されています。これらは、バージョン3の実行サイクルと置き替わります。バージョン3 のデータベースをバージョン6 に変換した場合、そのフォームも変換されます。これは、ユーザのフォームとオブジェクトが「期待通り」に動作するように管理するためです。バージョン3の4th Dimensionで作成したフォームとオブジェクト用に新しいイベントを利用したい場合、そのフォームとオブジェクト用に「フォームプロパティ」または「オブジェクトプロパティ」ウインドウ内で新しいイベントを有効にする必要があります。

2. テーブルメソッドとも呼ばれるトリガは、バージョン6 で新しく追加されたメソッドです。バージョン3 の4th Dimensionでは、テーブル用のフォームがデータ入力、表示、印刷に使用されるときだけ、4D が (ファイルプロシージャと呼ばれる) テーブルメソッドを実

行しました。このため、これらは滅多に使用されませんでした。トリガは、旧バージョンのファイルプロシージャのかなり低いレベルで実行されることに注意してください。(データ入力のような) ユーザアクションを通して、または (**SAVE RECORD** コマンドへの呼び出しのような) プログラムに従って記録を取ろうとも、4D がテーブルのトリガを呼び出します。トリガは、旧バージョンのファイルプロシージャとはまったく異なります。バージョン3 のデータベースをバージョン6用に変換した場合、そして新しいトリガの機能を利用したい場合、下図の「データベースプロパティ」ダイアログボックス内の「旧バージョンのファイルプロシージャ方式を使用する」チェックボックスを選択解除する必要があります。

3. データベースメソッドは、バージョン6 で新しく提供追加されたメソッドです。バージョン3の4th Dimension では、ユーザがデータベースを開いたときに4D が自動的に実行する唯一のメソッドがありました。このプロシージャを呼び出すには、「Startup」プロシージャを呼び出す必要がありました。バージョン3 のデータベースをバージョン6 に変換した場合、そして新しい「On Startup」データベースメソッドの機能を利用したい場合、下図の「データベースプロパティ」ダイアログボックス内の「旧バージョンのStartupプロシージャ方式を使用する」チェックボックスの選択解除をする必要があります。このチャックボックスは、「Startup」プロシージャまたは「On Startup」データベースメソッドのどちらか一方にしか効果はありません。この属性の選択解除を行わず、そして例えば「On Exit」データベースメソッドを作成した場合、4D は後者の「On Exit」データベースメソッドを呼び出します。

メソッドの例と用語

この節では、メソッドの用語、概念に共通する部分について説明するためにメソッドを詳しく調べてみることにします。この節に記述した内容の詳細は、このマニュアルの別の部分で説明されています。

メソッドは先頭の行から始まり、最後の行に到達するまで、各ステートメント(命令文)を実行します。基本的には、すべてのメソッドは同じです。次にプロジェクトメソッドの例を示します：

```
QUERY ([従業員])                          ` 「クエリ」エディタを表示する
If (Records in selection ([従業員]) =0)   ` 何も見つからない場合、
  ADD RECORD ([従業員])                   ` レコードを追加する。
End if                                    ` 終了
```

まず、プログラミング言語の用語と機能を説明することにします。上記の各行を“ステートメント”または“コード行”と呼びます。プログラミング言語を使用して作成したものを、単にコードとも呼びます。4th Dimensionは、コードで指定した処理を実行します。

それでは、最初の行を詳しく見てみましょう：

```
QUERY ([従業員])                          ` 「クエリ」エディタを表示する
```

この行の最初の要素である **QUERY**は、コマンドです。コマンドは、4th Dimensionのプログラミング言語の一部です。**QUERY**コマンドは「クエリ」エディタを表示します。「ユーザ」モードの「クエリ」メニューから「検索」を選択することと同じ機能です：

```
QUERY ([従業員])                          ` 「クエリ」エディタを表示する
```

括弧は、**QUERY**コマンドに対する引数(パラメータ)を指定します。引数は、コマンドが処理を実行させるために必要なものです。テーブル名は常に角括弧([...])の中で指定します。つまり、“従業員テーブルが **QUERY**コマンドの引数である”ということを意味します：

```
QUERY ([従業員])                          ` 「クエリ」エディタを表示する
```

行の終りにコメントがあります。コメントは逆アポストロフィ(`)によって示します。ユーザ(コードを解析する人)にこのコードが何を行っているのかを説明します。またコメント記号に続く内容は、コードを実行する時点で無視されます。コメントはそれ自体で1つの行になりますが、通常は上記の例で示すようにコードの右側に記述します。コメントの使用することにより、メソッドの内容を読みやすく、理解しやすいものにします。コメントは、80バイトまで入力することができます。

次の行は、レコードが見つかったかどうかを調べます：

```
If (Records in selection ([従業員])=0)          ` 何も見つからない場合...
```

Ifはフロー制御ステートメント(“ フロー制御文 ” または単に “ 文 ” ともいいます)です。これはメソッドの実行の流れを制御します。**If**文は、カッコ内の式を判定し、結果が “ True(真) ” の場合、実行が次の行に継続されます。本マニュアルでは、2つのブール値 “ True(真) ” と “ False(偽) ” を使用します。

Records in selectionは関数です。これは値を返すコマンドです。ここでは、**Records in selection**関数はカレントセレクションのレコード数を返します。

注：関数名の頭文字だけが大文字になっていることに注意してください。これは、4th Dimensionの関数に対する命名規則によるものです。

すでに、カレントセレクションとは何かについて説明しました。これは、その時点で作業対象となっているレコードの集まりのことです。レコードの数が 0件の場合(レコードが全く見つからない場合)に次の行を実行します：

```
ADD RECORD ([従業員])                    ` レコードを追加する。
```

ADD RECORDコマンドは入力フォームを表示し、新しいレコードを追加します。この行はインデントされています。4th Dimensionは、メソッドを保存した時点で、自動的にコードをフォーマットします。この行は、先ほどのフローステートメント(**If**)に従属することを示すためにインデントされています：

```
End if                                   ` 終了
```

End if文は**If**文の制御セクションを終了します。フロー制御ステートメントを使用した場合は、常に制御が終了する場所を示す対応ステートメントを持つ必要があります。

この節の概念をしっかりと把握し、理解するまで見直してください。

- オブジェクトメソッドとフォームメソッドに関する詳細は、第4章を参照してください。
- トリガに関する詳細は、第57章の「トリガコマンド」を参照してください。
- プロジェクトメソッドに関する詳細は、第6章を参照してください。
- データベースメソッドに関する詳細は、第6章を参照してください。

制御フロー

ユーザは、メソッドの単純性や複雑性に関係なく、プログラミング構造のシーケンス、分岐、ループの3種類のうち1つまたは複数をいつでも使用し、メソッド内でステートメントの実行や順序を制御します。

これらの構造のそれぞれを制御するステートメントがあります。この節では、これらのステートメントについての紹介にとどめます。詳細に関しては、この節を参照してください。

シーケンス構造

シーケンス構造は、単純な線形構造です。シーケンスは、4th Dimensionが最初から最後まで次々に実行する一連のステートメントです。例えば、次のようなものです：

```
OUTPUT FORM (([人事]) ; "リスト")
ALL RECORDS ([人事])
DISPLAY SELECTION ([人事])
```

1行から成るルーチンも、オブジェクトメソッドに対しては頻繁に使用されます。最も簡単なシーケンス構造の例を次に示します：

```
名字:= Uppercase (名字)
```

分岐構造

分岐構造は、条件を判定した結果に応じて、メソッドに、別のパスを与えることができます。この条件は“ True(真) ”または“ False(偽) ”に評価される式、つまりブール式です。分岐構造には、**If...Else...End if**構造があります。これは、2つのパスのうちのいずれかにプログラムの流れを導きます。他の分岐構造には、**Case of...Else...End case**構造があり、これは多くのパスの中の1つだけにプログラムの流れを導きます。

If...Else...End if構造

```
If(ブール式)
   ステートメント
Else
   ステートメント
End if
```

If...Else...End if構造は、プログラムが2つの動作のうちから、判定結果(ブール式)が“ True(真)か“ False(偽) ”かに応じて選択するものです。ブール式が“ True(真) ”の場合は、テストのすぐ後のステートメントを実行し、ブール式が“ False(偽) ”の場合には、**Else**文のすぐ後のステートメントを実行します。**Else**文はオプションです。**Else**を省略した場合の、実行は、**End if**の後の最初のステートメント(存在する場合だけ)を実行します。

例えば、

```
$名字:=Request ("名字を入力してください。")   ` ユーザに名字の入力を求める
If ($名字 # "")                                ` 何か名字が入力されたら...
   QUERY ([人事] ; [人事]名字 = $名字)
Else
   ALERT ("名字が入力されていません。")        ` ブール式“ False ”を実行する
End if
```

Case of...Else...End case構造

```
Case of
   \ (ケース)
         ステートメント
   \ (ケース)
         ステートメント
.
.
.
Else
   ステートメント
End Case
```

Case of...Else...End case構造は、メソッドを複数の動作から選択できるようにします。**Case of...Else...End case**構造は、**If...Else...End if**構造と異なり複数のブール式を評価し、“ True(真) ”であるものに対して処理を実行します。

それぞれのブール式の前には円記号(\)を付けます。この組み合わせ(円記号とブール式)を“ ケース ”と呼びます。例えば、次の行はケースです：

```
\ (番号=1)
```

最初に“ True(真) ”になったケースの後のステートメントだけを実行します。“ True(真) ”になるケースがない場合には、ステートメントを何も実行しません。また、最後のケースの後に**Else**文を含むことができます。すべてのケースが“ False(偽) ”の場合に、**Else**文の後のステートメントを実行します。

下記の例は数値変数を判定し、アラートボックスに対応する数字を表示します：

```
Case of
  \ (数値=1)                          ` 数値が1の場合の判定
    ALERT ("1")                       ` 1の場合のアラート表示
  \ (数値=2)                          ` 数値が2の場合の判定
    ALERT ("2")                       ` 2の場合のアラート表示
  \ (数値=3)                          ` 数値が3の場合の判定
    ALERT ("3")                       ` 3の場合のアラート表示
  Else                                ` 数値が1、2、3のどれでもない場合
    ALERT ("1、2、3のどれでもない。")  ` どれでもない場合のアラート表示
End case
```

比較するために、同じことを**If...Else...End if**構文で記述すると次のようになります：

```
If (数値=1)                                    ` 数値が1の場合の判定
  ALERT ("1")                                  ` 1の場合のアラート表示
Else
  If (数値=2)                                  ` 数値が2の場合の判定
    ALERT ("2")                                ` 2の場合のアラート表示
  Else
    If (数値=3)                                ` 数値が3の場合の判定
      ALERT ("3")                              ` 3の場合のアラート表示
    Else                                       ` 数値が1、2、3のどれでもない場合
      ALERT ("1、2、3のどれでもない。")        ` どれでもない場合のアラート表示
    End if
  End if
End if
```

Case of...Else...End case構造の場合には、最初に“ True(真) ”になったケースだけを実行します。2つ以上のケースが“ True(真) ”の場合は、最初に“ True(真) ”になったケースの後のステートメントだけを実行します。

ループ構造

メソッドの作成にあたって、何度も同じ処理を繰り返すことがあります。これを行うために、4th Dimensionは**For**、 **While**、 **Repeat**の3つのループ構造を備えています。ループを制御する方法には、条件が満たされるまでループする方法と指定した回数だけループする方法の2通りがあります。各ループ構造にはいずれの方法も用いることができますが、**While**ループと**Repeat**ループは条件が満たされるまで繰り返す場合に、**For**ループは指定した回数だけループする場合に有効です。

While...End whileループ

```
While (ブール式)
   ステートメント
End while
```

Whileループは、ブール式が " True(真) " である限り、ループ内のステートメントを実行し続けます。しかし、ブール式が " False(偽) " の場合には何もループを行いません。

一般に、**While**ループに入る手前で、ブール式で判定する値を初期化しておきます。通常はブール式が " True(真) " になるように設定してからループに入ります。

代表的な**While**ループの例として、レコードの追加があります：

```
CONFIRM ("新しいレコードを追加したいですか？") ` レコードを追加したいか？
While (OK=1)                                  ` ユーザが希望する限りループを続行する
  ADD RECORD ([従業員])                       ` 新規レコードを追加する
End while                                     ` End whileで終了する
```

この例では、ループに入る前に**CONFIRM**コマンドによりシステム変数OKをセットします。ユーザがダイアログボックスで「OK」ボタンをクリックすると、システム変数OKに1がセットされ、ループを開始します。取り消す場合はシステム変数OKに 0が設定され、ループをスキップします。ループに入ると、**ADD RECORD**コマンドはループを続けます。これは、ユーザがレコードを保存した時点で、システム変数OKに1が設定されるからです。ユーザが最後のレコードを取り消した(保存しない)時点で、システム変数OKに 0がセットされ、ループは終了します。

ループが永久に実行されないようにするために、ブール式に対してループ内で何らかの設定を行わなければなりません。次のループは、変数 " 永久 " が常に " True(真) " のため、ループを永久に続行します：

```
永久:= True
While (永久)
End while
```

注：メソッドが制御できない状態で実行されているメソッドの実行を停止するには、トレース機能を使用してください。

Repeat...Untilループ

```
Repeat
   ステートメント
Until (ブール式)
```

Repeatループは、ループの後でブール式を判定する以外は**While**ループとまったく同じです。つまり、**Repeat**ループは必ずループを1回は実行し、ブール式が最初に“ False(偽) ”となった時点で、ループを終了します。

その他の**While**ループとの相違点は、ブール式が“ True(真) ”になるまでループを続行することです。次の例を、**While**ループの例と比較してください。ブール式を、初期化する必要がない点に注目してください。システム変数OKを初期化する**CONFIRM**コマンドはありません。また、判定が逆である点にも注目してください。つまり、OK#1になることです（OKが1と等しくない）：

```
Repeat
  ADD RECORD ([従業員])
Until (OK#1)
```

For...End forループ

```
For (カウンタ ; 開始値 ; 終了値 ; {増分値})
   ステートメント
End for
```

Forループは、＜カウンタ＞によって制御されるループです。このカウンタは、**For**ループが値を初期化し、その後でループを実行するたびに加算される数値変数です。カウンタが指定した値を超えた時点で、ループを停止します。**For**ステートメントで指定する最初の引数は、カウンタ変数です。2番目と3番目の引数は、それぞれ開始値と終了値です。また、指定しない限り、増分値は1です。

▼次のループは、1で開始し100回ループします：

```
For (1 ; 1 ; 100)
End for
```

＜開始値＞が＜終了値＞より大きい（カウンタが減少していく）場合、＜増分値＞に負数を指定する必要があります。

▼ 次の例も100回ループします：

```
For (1i ; 100 ; 1 -1)
   ...
End for
```

Whileループと**Repeat**ループで、同じ処理を実行する方法を調べてみましょう。次の例は、同じ処理を実行する**While**ループです：

```
$i:=1                                  ` カウンタを初期化する
While ($i <=100)                       ` 100回までループを行う
  $i:=$i+1                             ` カウンタを1加算する
End while
```

▼ 次の例は、**Repeat**ループで同じ処理を実行します：

```
$i:=1                                  ` カウンタを初期化する
Repeat
  $i:=$i+1                             ` カウンタを1加算する
Until ($i >100)                        ` 100回を越えるまでループする
```

Forループの一般的な用途は、セレクションの各レコード間の移動を繰り返すことです。

▼ 次の例は、この処理を行うコードです：

```
For ($i ; 1 ; Records in selection ([従業員]))
  ` ここでレコードに対して何らかの処理を行う
  NEXT RECORD ([従業員])               ` 次のレコードに移動する
End for                                ` End forでForループを終了する
```

ループはカウンタ$iの初期値に1を設定し、$iがセレクションのレコードの数よりも大きくなるまでループします。1つのレコードしかない場合は、ループは1回しか実行しません。レコードが全くない場合には、ループは何も実行しません。

4th Dimensionが、ループの条件を各サイクルで内部的に判定し、カウンタを加算するため**For**ループは、**While**や **Repeat**ループよりも高速です。なるべく、**For**ループを使用してください。

注：この節のループの例で、カウンタに $iというローカル変数を使用していることに注目してください。これは、この変数をループ内でしか使用しないため、プロセス変数を使用する必要がないからです。

フォームメソッドとオブジェクトメソッド　4

フォームは、4th Dimensionの最も特徴的な部分です。4th Dimensionが提供するツールを使用することにより、自由にさまざまなフォームを作成することができます。入力フォームに優れたユーザインタフェースを利用することができるだけでなく、出力フォームにおいても読みやすいレポートを作成することができます。

4th Dimensionは、プログラミング言語を使用しないでフォームオブジェクトを管理できる組み込みツールを備えています。このツールには、データの属性チェック、入力フィルタ、フォーマット、データの範囲チェック、選択リスト、デフォルト値の設定、ボタンの属性設定などが用意されています。また、プログラミング言語は組み込みツールの機能を拡張し、複雑なフォームの制御や範囲チェックを実現することができます。だからと言って、馴れ親しんできた組み込みツールによる制御をあきらめる必要はありません。実際、これらのツールを最大限利用すべきです。プログラミング言語は、単にユーザがすでに会得したツールの機能を拡張するために使用します。

次の2種類のメソッドは、フォームを制御するために使用されます：

- オブジェクトメソッド
- フォームメソッド

オブジェクトメソッドは、フォーム制御のための最も一般的なメソッドです。フォームは、すべてのオブジェクトに対してオブジェクトメソッドを持つことができ、データの入出力時に使用します。

フォームメソッドは、特定の入力フォームまたは出力フォームで使用します。

フォームを制御する

フォームの制御は、データベースをカスタマイズする上で最も難しい部分の1つです。プロジェクトメソッドを実行している場合は、フォームを制御する必要はありません。メソッドは指示されたことだけを実行し、それに影響を与える外的要素はほとんどありません。しかし、フォームを使用すると、自分のメソッドが取り扱わなければならないユーザというイベントの新しいソースができます。

ユーザは、自分で実行することを決定します。例えば、ボタンのクリックや、フィールド間の移動、別のフォームページへの移動、メニューの選択など、ユーザは一般にメソッドを混乱させるような行動を取ります。

さらに、フォーム自体を使用することから生じる特別なイベントもあります。フォームが最初に表示された時点で、初期化が必要になるイベントです。レコードが受け入れられた時点で、特別な処理が必要な場合もあります。レポートを印刷する場合は、レポートのさまざまな部分に対していくつもの制御が必要となります。

これらのイベントは、どのように管理するのでしょうか？　メソッドが多くの起こり得るイベントをそれぞれをチェックし、応答しなければならないとしたら、メソッドで何かを行うことは、ほとんど不可能になります。これでは、イベントを判定するだけで終わってしまいます。しかし、4th Dimensionにはユーザに代わってイベントを管理し、それを的確にメソッドに知らせる手段が用意されています。

イベントの管理は、オブジェクトメソッドとフォームイベントの2つの手段を用いて行われます。オブジェクトメソッドはオブジェクト指向のイベントマネージャです。このイベントマネージャーは、フォームオブジェクトで発生したイベントに対して応答します。また、フォームイベントも重要なイベントマネージャです。これは、フォームに発生するすべてのイベントをチェックします。

オブジェクトメソッドを使用する

オブジェクトメソッドは、フォーム内のオブジェクトに付着するメソッドです。この役割は、オブジェクトに対して非常に特殊な存在であり、付着したオブジェクトだけを管理します。

オブジェクトメソッドは、次のような場合に実行されます：

- データ入力の実行
- 画面上でのレコード表示
- フォームを使ったレポートの印刷
- フォームを使ったデータの読み込みと書き出し

オブジェクトメソッドは必須のものではありませんが、各オブジェクトは1つのオブジェクトメソッドを持つことができます。

オブジェクトは、入力 (例えば、フィールドや変数など)やインタフェース (例えば、ボタン、ドロップダウンリスト、サーモメータなど)の一部として使用します。オブジェクトメソッドはオブジェクトが操作されるたびに実行されます。例えば、ボタンのオブジェクトメソッドは、ボタンがクリックされた時に実行されます。フィールドのオブジェクトメソッドは、そのフィールド上でデータが入力または修正された場合に実行されます。

また、レコードを画面上に表示したり、印刷したりする場合にもオブジェクトのオブジェクトメソッドを実行します。この場合のオブジェクトメソッドはオブジェクトのフォーマットや外観に影響を与えます。

オブジェクトメソッドは数行程度の短いものがほとんどです。4th Dimensionデータベースでは、オブジェクトメソッドが唯一のメソッドになっているのがほとんどです。オブジェクトメソッドがオブジェクトの中に作成されると、オブジェクトを切り取り、コピー、複製した場合でも、オブジェクトメソッドは属するオブジェクトに従います。つまり、オブジェクトのライブラリからスクラップブックに貼り付けて、別のフォームに貼り付けることができます。オブジェクトメソッドでプロジェクトメソッドを呼び出せる点に注目してください。プロジェクトメソッドでは、オブジェクトメソッドによってコピーされたり、貼り付けられたりすることはありません。

オブジェクトメソッドとデータ入力

フィールドや変数などのオブジェクトメソッドは、次のような操作に対して実行されます：

- 入力されたデータの属性チェック
- 変数に対する値の割り当て
- フィールドの連結や小文字から大文字への変換など、文字列の操作
- 合計、平均、カウントの計算など、算術および日付計算の実行
- リレートテーブルのデータ管理

データがデータ入力中のフィールドや変数によって変更されると、そのフィールドや変数に対するオブジェクトメソッドが実行されます。このオブジェクトメソッドは、フォームメソッドの前に実行されます。

オブジェクトメソッドとインタフェースオブジェクト

フォームには、ユーザとのインタフェースを司る広範囲のオブジェクトを含むことがあります。これらのオブジェクトは、ボタンの自動動作などの組み込みツールによって完全に制御することができます。オブジェクトの中には非常に柔軟性のあるものやプログラミング言語がチェックや制御に必要なものもあります。この節では、オブジェクトとプログラミング言語間の相互作用について説明します。

次の図は“変数”ページを示しています。オブジェクト(変数)は、ここで定義します(オブジェクトの作成とその使用に関する詳細は、『4th Dimension / 4D First デザインリファレンス』を参照してください)。オブジェクトの名前は、4th Dimensionが自動的に作成する変数の名前でもあります。この変数は、オブジェクトの存在をチェックする場合によく使用されます。

次の図は、28種類のオブジェクトタイプを定義するために使用するドロップダウンリストボックスを示しています。

オブジェクトメソッドは、オブジェクトの取り扱いを簡単にします。オブジェクトにオブジェクトメソッドを組み合わせて使用することによって、ユーザはメソッドと対話することができます。次節では、オブジェクトタイプ別のオブジェクトメソッドの使用方法について説明します。

ボタン

ボタンは、フォームの中で最もユーザインタフェースに優れたオブジェクトです。ボタンによるオブジェクトメソッドの扱いは非常に簡単です。つまり、ボタンがクリックされた時点で「On Clicked」イベントがチェックされていると、オブジェクトメソッドは実行されます。そうでない場合には、オブジェクトメソッドはフォームイベントに従って実行されます。

ボタンの名前は、変数の名前でもあります。変数は自動的に作成され、ボタンに付随します。ボタンがクリックされると、変数に1が代入されます。その他の場合には、変数の値は0となります。ボタンの自動制御はオブジェクトメソッドが実行された後で行われます。

下記は、6つのボタンタイプとそのボタンがクリックされた際の結果です。

注：ボタンのオブジェクトメソッドは、ボタン変数が設定された後で実行されることを覚えておいてください。

■**「ボタン」：**ボタン変数に1を代入します。

■**「ラジオボタン」：**ラジオボタンは、ボタン変数名の頭文字でグループ化され、同じ頭文字を持つボタンが1つのグループになります。ラジオボタンがクリックされると、ボタン変数に1を代入し、ボタンを反転表示します。他のすべてのラジオボタンはそれぞれの変数に 0を代入し、通常表示されます。

■**「チェックボックス」：**ボックスがチェックされると1を、チェックされないと 0が設定されます。

■**「透明ボタン」：**ボタン変数に1が代入されます。(このボタンは、クリックされても反転表示しません。)

■**「ハイライトボタン」：**ボタン変数に1が代入されます。(このボタンは、クリックされると反転表示します。)

■**「ラジオピクチャ」：**ラジオピクチャは、ラジオボタンと同じように機能するハイライトボタンです。ラジオピクチャはボタン変数名の頭文字でグループ化され、同じ頭文字を持つボタンが1つのグループになります。ラジオピクチャがクリックされるとボタン変数に1を代入し、ボタンは反転表示されたままとなります。他のすべてのラジオピクチャはそれぞれの変数に 0を代入し、通常表示します。

スクロールエリアとドロップダウンリストボックス

スクロールエリアとドロップダウンリストボックスは、プログラミングの観点から見ると同じように機能します。両方とも項目リスト(配列要素)を含み、これらの項目の中からユーザが要素を選択できるようにします。項目リストは配列です。配列の名前はオブジェクトの名前と同じです。

注：配列は、配列コマンドで作成されます。

ドロップダウンリストボックスとスクロールエリアは、次の4つのデータタイプを表示することができます：

- 文字列
- 数値
- 日付
- 時間

特にスクロールエリアでは、表示エリアとブール配列変更を自由にすることができます。

これらは本質的に柔軟ですので、他のオブジェクトの場合よりもプログラミング言語との相互作用が一層必要になります。例えば、次のようなことが必要です：

- スクロールエリアまたはドロップダウンリストボックスに項目を格納する。
- ユーザが項目を選択した場合に応答する。
- 必要に応じてリストを変更する。

ドロップダウンリストボックスとスクロールエリアを使用する例は、第9章を参照してください。

スクロールエリアまたはドロップダウンリストボックスにデータを格納する

配列は、任意の方法で格納できます。「デザイン」モードで作成したリストやレコードのセレクションからも格納できます。例えば、リストから配列へ格納するには、次のように行います：

LIST TO ARRAY ("Values" ; arValList)

配列の格納は常に可能です。通常、次の3つの場所で格納されます：

- 「On Startup」データベースメソッド
- フォームが表示される前のプロジェクトメソッド
- フォームイベントの Beforeフェーズ

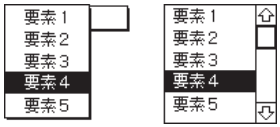
しかし、配列を格納するタイミングはエリアの使用方法によって異なりますので、配列要素が決して変わらない場合は「On Startup」データベースメソッドで、配列要素が各レコードごとに変わらない場合はプロジェクトメソッド、そして配列要素がレコードごとに変わる場合にはフォームメソッドまたはオブジェクトメソッドのBeforeフェーズで行うのが一般的です。

ユーザが項目を選択した場合の応答

ユーザがドロップダウンリストボックスまたはスクロールエリアの項目を選択すると、次のようなイベントが発生します：

1. **選択された項目番号(配列要素)を配列の名前に設定する。**
2. **Duringフェーズが実行される。**
3. **オブジェクトメソッドが実行される。**

配列の名前は、ユーザが選択した項目を表す正の数値に設定されます。次の図のドロップダウンリストボックスとスクロールエリアを参照してください。表示されている配列変数を“ 私の配列 ”とすると、ユーザが4番目の要素を選択した場合(図で示された通り)に、配列変数“ 私の配列 ”に4を設定します。



ユーザがスクロールエリアの空のエリアをクリックしたり、配列要素を全く選択しなかった場合は配列変数に 0を設定します。この場合には、適切な値を得るための特別な判断を加えたり、ドロップダウンリストボックスを変更する必要があります。

項目が常に選択されているようにする場合は、最後に選択した項目の数を保存し、それをその項目に戻します。

例えば、次の例はユーザが何も選択しなかった場合に、項目を前の値に戻します：

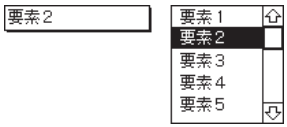
```
Case of
  \ (Before)
     項目:=1                                  ` 項目の初期化
     私の配列:= 1
  \ (私の配列=0)                              ` 何も選択されなかった場合...
     私の配列:=項目                           ` 前に選択された項目をセットする
  Else
     項目:=私の配列                           ` それ以外の場合は、選択された項目を保存する
End case
```

項目が選択されているかどうかを調べるために、オブジェクトのオブジェクトメソッドを使用します。オブジェクトメソッドが実行されると、(Duringフェーズ中に）オブジェクトが選択されたことを意味します。

スクロールエリアとドロップダウンリストボックスの項目変更

配列をどのような方法で変更しても、4th Dimensionは自動的にそれを認識し、その変更を反映させるためにオブジェクトを更新します。変更には、配列要素の変更、要素の削除と追加、配列全体の削除があります。

項目を“ 選択 ”するために、選択する項目に対して配列変数を設定します。この項目はスクロールエリアで反転表示されるか、あるいはドロップダウンリスト（ポップアップメニュー）の表示メニューアイテムとして表示されます。配列変数の設定には、代入演算子を使用します。これを使用しないと、配列の変更が認識されません。下図は2番目の要素が選択されたドロップダウンリストボックスとスクロールエリアを示しています。



このドロップダウンリストボックスとスクロールエリアが配列“ 私の配列 ”の場合、次のように“ 私の配列 ”に選択した項目の値を代入します：

```
私の配列:= 2
```

サーモメータ、ルーラ、ダイアル

サーモメータ、ルーラ、ダイアルはエリアの割合を数値として表示します。オブジェクトに属する変数は画面の状態を反映するために変更され、反対に画面は変数の値を反映するように変更されます。つまり、ユーザがオブジェクト上をドラッグした場合は変数の値を変更し、変数の値を変更した場合には新しい値を反映するために画面を変更します。

オブジェクトのオブジェクトメソッドは、一般にBeforeフェーズでオブジェクトを初期化します。Duringフェーズはユーザがオブジェクトを変更するたびに実行されます。

グラフエリア

グラフエリアは、**GRAPH**コマンドで完全に制御します。

外部ルーチンエリア

外部ルーチンエリアは、4th Dimensionのプログラミング言語以外の言語で作成されたメソッドで制御するエリアです。外部ルーチンエリアの処理に制限はありませんが、これを選択すると、4th Dimensionに制御を戻すまで、外部ルーチンがすべてを制御、管理します。4th Dimensionに対して制御を返す場合、Duringフェーズが1回発生します。

オブジェクトメソッドとレポート出力

フィールドや変数に付着するオブジェクトメソッドは、それらを含むフォームが印刷される時点で実行されます。また、オブジェクトのフォームエリアが印刷された場合にのみ実行されます。

例えば、下図で示したフォームには、B0ブレークエリアに“ vTotal ”という名前の変数があります。

このオブジェクトメソッドを持つ変数は、レベル0のブレークで印刷されます。このフォームにおける、“ vTotal ”のオブジェクトメソッドは次の通りです：

vTotal:= Subtotal (給与)

このオブジェクトメソッドは、変数“ vTotal ”に給与フィールドの小計を代入します。

フォームイベント　5

フォームメソッドとオブジェクトメソッドは、フォームイベントに従って実行されます。フォームイベントは、メソッドを実行するタイミングを決定します。

フォームメソッドは、フォームやプロセスで何か起こった場合に開始します。また、4th Dimensionのマルチプロセス環境では、あるプロセスに別のプロセスを呼び出させます。データ入力、表示、およびプロセス間通信で使用されるフォームの基本的なフォームイベントをいくつか次に示します：

- Beforeフェーズ
- Duringフェーズ
- Afterフェーズ
- Outside Callフェーズ
- Activatedフェーズ
- Deactivatedフェーズ

フォームイベントの各部は、“ フェーズ ”として参照されます。フォームのメソッドは、各フェーズで実行されます。Before、During、Afterの各フェーズは最も重要なものでり、頻繁に使用されます。Outside Callフェーズは、呼び出しのプロセスが別のプロセスから作成された場合に実行されます。ActivatedフェーズとDeactivatedフェーズは、ウインドウがアクティブかアクティブでない時に呼び出されます。この2つのフェーズは、オブジェクトメソッドの中では発生しません。

また、印刷時には、次のようなフォームイベントがあります：

- In Headerフェーズ
- Beforeフェーズ
- Duringフェーズ
- In Breakフェーズ
- In Footerフェーズ

注：この章では、バージョン6で新たに追加されたフォームイベントの説明は行っていません。バージョン6で新たに追加されたフォームイベントに関する詳細は、第24章を参照してください。

フォームイベントは、フォームに付着するすべてのメソッドに影響を与えます。

フォームイベントはフォーム処理の中核ですが、ほとんどの場合において、それを意識する必要はありません。必要な場合にのみ使用します。しかし、フォームイベントの各フェーズを使用することにより、さまざまなデータやオブジェクトの管理を適切なタイミングで制御することができます。

オブジェクトに対してオブジェクトメソッドを作成する場合、フォームイベントを考慮する必要はありません。オブジェクトメソッドは、それの属するオブジェクトが操作されるたびに実行されるのでフォームイベントを必要としません。これにより、オブジェクトメソッドは適切なタイミングで実行されます。しかし、特定のフェーズだけに処理を制限する場合には、オブジェクトメソッド内でフォームイベントのフェーズを判定する必要があります。

フォームイベントのチェック

プログラミング言語は、フォームイベントを判定するための関数を備えています。各関数は、フェーズが発生している場合に“ True ”を返し、他の場合は“ False ”を返します。

次に、各関数の名前とそれに対応するフェーズについて説明します。

■**「Before」：**Beforeフェーズは、フォームが表示、印刷される前に各レコードに対して発生します。

■**「During」：**Duringフェーズは、レコードの表示やオブジェクトの修正やクリック、あるいはメニューの選択の各時点で発生します。

■**「After」：**Afterフェーズは、データ入力時にレコードが受け入れられた後にだけ発生します。

■**「Before & During」：**これは特別なフェーズです。レコードが画面上に表示された時点で発生します。これは2つのフェーズが同時に“ True ”を返した場合のみです。

■**「Outside Call」：**Outside Callフェーズは、呼び出しプロセスが別のプロセスから作成された場合に発生します。

■**「Activated」：**Activatedフェーズは、ウインドウがアクティブになった場合に発生します。

■**「Deactivated」：**Deactivatedフェーズは、ウインドウがアクティブでない場合に発生します。

■**「In header」：**In headerフェーズは、フォームのヘッダエリアが印刷される前か、あるいは画面上に表示される前に発生します。1つのヘッダが各ページに印刷されます。ヘッダが複数の場合には、各ブレークレベルが印刷されるたびに発生します。最初のヘッダであるかどうかは、**Before selection**関数を使用して判定します。

■ **「In break」**：In Breakフェーズは、ブレーク処理中に発生します。ブレークエリアはそれから印刷されます。印刷されるブレークレベルは、**Level**関数を使用して判定することができます。

■ **「In footer」**：In Footerフェーズは、フッタエリアの印刷が開始される前にのみ発生します。1つのフッタが各ページに印刷されます。**End selection**関数では、最後のフッタであるかどうかを判定することができます。

メソッドは一般にフォームイベントのフェーズを判定するために、**Case**構文を使用します。次はデータ入力におけるフォームメソッドの一般的な構造を示しています：

```
Case of
  \ (Before)
    ` 変数の初期化
  \ (During)
    ` 入力データの表示
  \ (After)
    ` レコードの保存
End case
```

フォームイベントの一般的な規則

ここでは、フォームイベントに関連したいくつかの一般的な規則を示します：

■ メソッドのフォームイベントの各フェーズは、次の順序で実行されます：

1. **オブジェクトメソッドの実行**
2. **フォームメソッドの実行**

■ データ入力に使用するオブジェクトのオブジェクトメソッドは、データの入力順序順で実行されます。

■ データ入力で使用されないオブジェクトのオブジェクトメソッドは実行されません。オブジェクトメソッドはデータの入力順序とは無関係に実行されます。

■ レコードが画面上に表示または印刷される場合は、オブジェクトメソッドが“修正不可”でないオブジェクトのオブジェクトメソッドだけが実行されます。例えば、ヘッダの変数は、ヘッダが表示された場合にのみ、(In Headerフェーズ中で)そのオブジェクトメソッドが実行されます。

フォームイベント

次の節は、各フォームの使用ごとに、フォームイベントを説明しています。

データ入力の場合

ユーザがフォームにデータを入力する場合のフォームイベントには、次のフェーズがあります。

■**「Before」：**このフェーズは、フォームが表示される前に発生します。これは、新規レコードや修正レコードのチェックに使用します。Beforeフェーズは、変数の初期化やフィールドのデフォルト値の表示のために使用します。

■**「During」：**このフェーズは、データが修正されたり、アクティブオブジェクトが使用されると、データ入力時に発生します。

■**「After」：**このフェーズは、レコードが受け入れられた後にだけ発生します。

■**「Outside Call」：**このフェーズは、呼び出しが別のプロセスから作成された場合のデータ入力時に発生します。

■**「Deactivated」：**このフェーズは、フォームやオブジェクトメソッドを含んだウインドウが背後にある場合や最前面のウインドウがない場合のデータ入力時に発生します。Deactivatedフェーズは、警告メッセージやデバッガが表示されている場合にも発生します。

■**「Activated」：**このフェーズは、フォームやオブジェクトメソッドを含んだウインドウが最前面に配置されるたびにデータ入力時に発生します。

Duringフェーズは、ユーザがフォームに対して何かを行った場合に発生します。ユーザの動作から、Duringフェーズを発生させるものとしては、次のようなものがあります：

■ フィールドや変数のデータを変更してから、そのフィールドや変数を離れる

■ ボタンをクリックする

■ サーモメータ、ルーラー、ダイアルを操作する

■ メニューからメニューコマンドを選択する

■ ドロップダウンリストボックス（ポップアップメニュー）から項目を選択する

■ スクロールエリアから項目を選択する

■ ショートカットキーを押す

■ 外部ルーチンエリアをクリックする

■ 組み込みエリアを使用する

Duringフェーズは、データの属性チェック、入力データのフォーマット、リレートテーブルのデータ管理、制御選択のチェックや応答、フォームオブジェクトの特定の処理などに使用します。

Afterフェーズは、ユーザがレコードを受け入れた後で発生します。しかし、レコードをキャンセルした場合は発生しません。

テーブルのサブフォーム

サブフォーム内のレコード表示のフォームイベントには、次のフェーズがあります。

■**「Before」：**このフェーズは、新しいレコードがサブテーブルに追加された場合に発生します。

■**「During」：**このフェーズは、サブテーブルのレコードが最初に表示された場合と、その修正時に発生します。親レコードのDuringフェーズは、サブフォームのDuringフェーズの後に発生します。

■**「After」：**このフェーズは、サブテーブルの各レコードが変更され、受け入れられた場合に発生します。

■**「Deactivated」：**このフェーズは、フルページフォームを含んだウインドウが背後に配置されている場合に、リレートされたレコードのフルページフォーム内でのみ発生します。

■**「Activated」：**このフェーズは、フルページフォームが最前面に配置されて、リレートされたレコードのフルページフォーム内でのみ発生します。

■**「Outside Call」：**このフェーズは、プロセスの呼び出しが最前面のウインドウ内のフルページフォームを含んだプロセスで作成された場合に、リレートされたレコードのフルページフォーム内でのみ発生します。

フォームイベントの各フェーズに対して、オブジェクトメソッド、フォームメソッドが実行されます。

サブテーブルのサブフォーム

サブフォーム内のサブレコード表示のフォームイベントには、次のフェーズのがあります。

■**「Before」：**このフェーズは、親フォームのBeforeフェーズの実行前に新しいサブレコードが追加された場合に発生します。

■**「During」：**このフェーズは、各サブレコードが最初に表示された場合と、その修正時に発生します。親レコードのDuringフェーズは、サブフォームのDuringフェーズの後に発生します。

■**「After」：**このフェーズは、親レコードのAfterフェーズの前にそれぞれのサブレコードに対して発生します。

■**「Deactivated」：**このフェーズは、フルページフォームを含んだウインドウが背後に配置されている場合に、サブレコードのフルページフォーム内でのみ発生します。

■**「Activated」：**このフェーズは、フルページフォームが最前面になった場合に、サブレコードのフルページフォーム内でのみ発生します。

■**「Outside Call」：**このフェーズは、呼び出しが最前面のウインドウ内のフルページフォームを含んだプロセスで作成された場合に、サブレコードのフルページフォーム内でのみ発生します。

サブレコードのフルページフォームが受け入れられた時点では、Afterフェーズはありません。

フォームイベントの各フェーズに対して、オブジェクトメソッド、フォームメソッドが実行されます。

注：ご覧のように、リレートされたレコードとサブフォーム内のサブレコードのフォームイベントは同一のものではありません。

「ユーザ」モードのレコード表示

「ユーザ」モードのレコード表示のフォームイベントには、次の4つのフェーズがあります：

■**「In Header」：**このフェーズは、ヘッダエリアが表示される前に発生します。タイトルの表示やレコードのサマリー(小計)情報を生成する場合にも使用します。

■**「Before & During」：**このフェーズは、BeforeフェーズとDuringフェーズが同時に“ True(真) ”である場合に発生します。レコードが表示されるたびに1回発生します。

■**「During」：**このフェーズは、“ リスト更新 ”時にレコードのフィールドが修正された場合にのみ発生します。

■**「After」：**このフェーズは、“ リスト更新 ”時にレコードの修正を受け入れた場合にのみ発生します。

フォームイベントの各フェーズごとに、フォームメソッドが実行されます。

MODIFY SELECTIONコマンドとDISPLAY SELECTIONコマンド

MODIFY SELECTIONコマンドまたは**DISPLAY SELECTION**コマンドを使用してレコードを表示すると、フォームイベント中に7つのフェイズを使用することができます。レコードをダブルクリックすると、データ入力の規則に従って入力フォームメソッドを実行します。

■**「Before」：**このフェーズは、レコード表示前に1度だけ発生します。

■**「In Header」：**このフェーズは、ヘッダエリア表示前に1度だけ発生します。タイトルやサマリー(小計)情報を表示する場合に使用します。

■**「Before & During」：**このフェーズは、BeforeフェーズとDuringフェーズが同時に“ True ”である場合に発生します。各レコードが表示されるたびに発生します。

■**「During」：**このフェーズは、Beforeフェーズが“ False ”でDuringフェーズが“ True ”の場合に発生します。メニューの選択、ボタンのクリック、レコードのダブルクリックを行った場合に発生します。

■**「Deactivated」：**このフェーズは、レコードリストを含んだウインドウが背後に配置されるたびに発生します。

■**「Activated」：**このフェーズは、レコードリストを含んだウインドウが最前面に配置されるたびに発生します。

■**「Outside Call」：**このフェーズは、別のプロセスから出力フォームに含まれているプロセスの呼び出しがある場合に発生します。

フォームイベントの各フェーズごとに、フォームメソッドが実行されます。

フォームによる書き出し実行時

フォームによるレコードの書き出しの場合は、フォームイベント中のフェーズが 1つだけあります。

■**「Before」：**このフェーズは、レコードが書き出される前に1度だけ発生します。

フォームイベント中にオブジェクトメソッド、フォームメソッドを実行します。このフォームイベントを使用して、書き出す前のデータを修正することができます。例えば、複数のフィールドを連結したり、固定長データから埋め込み文字のスペースを取り除くといった使用方法があります。

フォームによる読み込み実行時

フォームによるレコードの読み込みの場合は、フォームイベント中のフェーズが 1つだけあります。

■**「After」：**このフェーズは、レコードが読み込みディスクに読み込まれる前に1度だけ発生します。

フォームイベント中にオブジェクトメソッド、フォームメソッドを実行します。このフォームイベントを使用して、読み込んだデータを修正することができます。例えば、固定長データから埋め込み文字のスペースを取り除くといった使用方法があります。

フォームレポート

レポートを印刷する場合には、次のようなフォームイベントのフェーズがあります。

■**「In Header」：**このフェーズは、ページごとにフォームヘッダエリアまたはブレークヘッダエリアが、印刷時に発生します。1つのページに1つのヘッダと複数のブレイクヘッダがあります。複数のブレークヘッダが存在する場合は、**Level**関数を使用して、どのブレークヘッダが印刷されるのかを判定することができます。また、**Before selection**関数を使用すると、レポートの最初のヘッダを判定することができます。

■**「Before」：**このフェーズは、ディテイルエリアにおいてレコードの印刷ごとに1度発生します。

■**「During」：**このフェーズは、ディテイルエリアのレコードの印刷ごとに1度発生します(Beforeフェーズに準じます)。

■**「In Break」：**このフェーズは、ブレークエリアでのブレークの印刷ごとに発生します。Level関数を使用してブレークレベルを判定することができます。

■**「In Footer」：**このフェーズは、ページごとのフッタエリアの印刷時に発生します。**End selection**関数を使用してレポートの最後のフッタを判定することができます。

フォームイベントの各フェーズごとに、フォームメソッドを実行します。例えば、ヘッダエリアに変数を設定すると、そのオブジェクトメソッドはヘッダが印刷される場合にのみ実行されます。

レポート処理を制御するようなオブジェクトメソッドをなるべく使用することをお勧めします。それぞれの印刷エリアにオブジェクトメソッドを持ったフォームオブジェクトを設定するだけで適切なタイミングでオブジェクトメソッドを実行します。

プロジェクトメソッド

プロジェクトメソッドには、適切な名前を付ける必要があります。フォームメソッドやオブジェクトメソッドはフォームやオブジェクトと密接に関連付けられていますが、プロジェクトメソッドはどこでも使用できます。4th Dimensionで使用できるプロジェクトメソッドの種類を次に示します：

- メニューメソッド
- サブルーチン
- プロセスメソッド
- イベント管理メソッド
- エラー管理メソッド

これらは、プロジェクトメソッドをその名称で識別するのでなく、その動作で識別します。

メニューメソッドは、カスタムメニューから呼び出されるプロジェクトメソッドです。これは、ユーザのアプリケーションの流れを管理します。メニューメソッドは、必要とされる場所での分岐、フォームの提供、レポートの生成、ユーザのデータベースの通常管理のコントロールを行います。

サブルーチンは、別のメソッドから呼び出されるプロジェクトメソッドです。関数は、関数を呼び出したメソッドに値を返すサブルーチンです。

プロセスメソッドは、プロセスの開始時に呼び出されるプロジェクトメソッドです。このプロセスは、プロジェクトメソッドが実行されている間だけ存続します。プロセスに関する詳細は、第10章を参照してください。「新規プロセス開始」チェックボックスが選択されているメニューコマンドに属しているメニューメソッドは、新しく開始されるプロセス用のプロセスメソッドでもあることに注意してください。

イベント管理メソッドは、イベントを管理するプロセスメソッドとして個々のプロセス内で実行します。通常、ユーザはユーザ用に操作しているイベントの大部分を 4D に任せています。例えば、データ入力中は4D がキーストロークを見つけてクリックし、それから正しいオブジェクトとフォームメソッドを呼び出します。このためユーザは、これらのメ

ソッド内で適切なイベントに応答できるのです。一方、ユーザが直接イベントを操作したい場合があります。例えば、ユーザが (**For...End For** ループ、レコードの操作などの) 処理時間の長い操作を実行する場合、ユーザは 「Ctrl - ピリオド」 (Windows の場合) や「コマンド - ピリオド」 (Macintosh の場合) をタイプして、その操作への割り込みができます。この場合、ユーザはイベント管理メソッドを使用する必要があります。詳細は、**ON EVENT CALL**コマンドの説明を参照してください。

エラー管理メソッドとは、プロジェクトメソッドを基にした割り込みのことです。エラーや例外が起こる度に、インストールされたプロセス内で実行します。詳細は、**ON ERR CALL**コマンドの説明を参照してください。

メニューメソッド

メニューメソッドは、カスタムメニューコマンドから起動されます。「メニュー」エディタを使用して、メソッドをメニューコマンドに割り当てます。メニューコマンドが選択されると、それに対応するメソッドが起動されます。このプロセスは、データベースをカスタマイズする上で大きな特長の1つです。特別な処理を実行するマスターメソッドを割り当てたメニューコマンドを作成することで、データベースをカスタマイズすることができます。『4th Dimension / 4D First デザインリファレンス』の「メニュー」エディタを参照してください。

カスタムメニューコマンドは、単一または複数の処理を実行することができます。例えば、データの入力処理を行うメニューコマンドは、次の2つの処理を行うメソッドを呼び出します。第1に、適切な入力フォームを設定します。第2に、ユーザがキャンセルするまでの **ADD RECORD**コマンドによるデータ入力を繰り返します。

連続処理の自動化は、プログラミング言語の強力な機能の1つです。カスタムメニューを使用すると、「ユーザ」モードにおいて手作業で行っていた処理を自動化することができます。また、カスタムメニューを使用することで、ユーザにより良い操作環境を提供することができます。

サブルーチン（メソッドから起動されるメソッド）

プロジェクトメソッドを作成すると、それは同じデータベースシステムの言語の一部になります。プロジェクトメソッドは、4th Dimensionに組み込まれたコマンドと同様に他のメソッドから呼び出すことができます。このように使用されるプロジェクトメソッドを“ サブルーチン ”と呼びます。

サブルーチンには、次の４つの利点があります：

■ コーディングの重複をなくすことができる。
■ メソッドの機能を明確にすることができる。
■ メソッドの編集を容易にすることができる。
■ メソッドをモジュール化することができる。

例えば、“ 顧客 ”というデータベースがあるとします。メソッドを作成していくうちに同じ処理を繰り返しコーディングしていることに気づきました。それは顧客テーブルを検索してレコードを修正するという一連の作業でした。そのコーディングは次のようになっています：

```
` 顧客テーブルの検索
QUERY BY EXAMPLE ([顧客])
` 入力フォームの設定
INPUT FORM ([顧客]) ; "入力フォーム")
` 顧客レコードの修正
MODIFY RECORD ([顧客])
```

サブルーチンを使用しなければ、10箇所で同じ処理が必要になると、同じコーディングを10回も行わなければなりません。しかし、サブルーチンを使用すれば、1回コーディングするだけで済みます。これが“ コーディングの重複をなくす ”というサブルーチンの第１の利点です。

このメソッドに、例えば“ M_顧客修正 ”という名前をつければ、その名前を他のメソッドで、記述するだけで呼び出すことができます。顧客テーブルの検索修正を行い、その後で印刷を行うメソッドのコーディング例を次に示します：

```
M_顧客修正
PRINT SELECTION ([顧客])
```

この例のようにサブルーチンを使用することにより、メソッドを単純化することができます。サブルーチンを呼び出す側は、サブルーチンがどのように処理するのかを知る必要はありません。サブルーチンが何を行うのかを理解していれば十分です。(この例ではサブルーチンの名前どおりに、顧客テーブルを修正するということを理解していれば十分です。） これが“ メソッドの機能を明確にする ”というサブルーチンの第2の利点です。このように、メソッドは 4th Dimension言語の機能を拡張します。

また、例題のデータベースで顧客テーブルの検索処理を変更する必要が発生した場合でも、“ M_顧客修正 ” のメソッドの修正だけで済みます。これが “ メソッドの編集を容易にする ” というサブルーチンの第３の利点です。

サブルーチンを使用して、“ メソッドのモジュール化 ” を行うことができます。つまり、プログラムを論理的な処理単位で、モジュール(サブルーチン)に分割します。次の当座預金データベースの例で考察してみましょう：

```
M_決済済み検索                  ` 決済済みの小切手を見つける
M_口座の照合                    ` 口座の照合
M_出納印刷                      ` 出納記録の印刷
```

データベースを知らない人でも、このプログラムが何をしているかはわかります。各サブルーチンの処理手順を知る必要はありません。各サブルーチンは長く、複雑な処理で構成されていることもありますが、そのサブルーチンが何を行うのかだけを知っていれば十分です。

プログラムを論理的な処理単位やモジュールにできるだけ分割することをお勧めします。メソッドが20行を越える場合には、そのメソッドを分割することを検討してください。これが “ メソッドのモジュール化 ” というサブルーチンの第４の利点です。

サブルーチンへ渡す引数（パラメータ）

サブルーチンへデータを渡すときに、引数（パラメータ）を使用すると簡単に行うことができます。引数は、サブルーチン内で行う処理に必要な一連のデータです。

引数は、4th Dimensionのコマンドへデータを渡す場合にも使用します。次の例は、文字列 “ こんにちは ” という引数を**ALERT**コマンドへ渡します：

```
ALERT ("こんにちは")
```

引数は、サブルーチンに対しても同じように渡すことができます。例えば、サブルーチンが、３つの引数を受け取る “ My Proc ” というメソッドを呼び出すには次のようにコーディングします：

```
My Proc (This ; That ; The Other)
```

引数の間は、セミコロン(;)で区切ります。

それぞれの引数の値は自動的に、サブルーチン内のローカル変数$1、$2、$3に順次格納されます。ローカル変数の番号は、使用する引数の数だけあります。ローカル変数は引数そのものを指しているのではなく、引数の値を保持しているだけです。ローカル変数は、サブルーチン内でのみ使用可能です。サブルーチンから戻ると、これらのローカル変数を使用することはできません。

ローカル変数の値を変更しても、サブルーチンを呼び出したメソッドの引数の値は、変更されません。例えば、次に示すコーディングは、“ 連結 ” というサブルーチンです。これは、2つの文字列を連結してアラートボックスにその結果を表示します：

```
$1:=$1+$2                    ` ２つの文字列を結合する
ALERT ($1)                   ` 結果を表示する
```

最初の処理で、変数$1の値が変わります。次のコーディングは、2つの引数をサブルーチン “ 連結 ” に渡します：

```
名前:="松木さん"
連結 (名前 ; "　こんにちは")
```

このコーディングを実行すると、ダイアログボックスに “ 松木さん　こんにちは ” という文が表示されます。ローカル変数$1は、文字列"松木さん"を受け取り、その後で文字列"松木さん　こんにちは"に置き換わります。しかし、変数 “ 名前 ” の内容は変わりません。

注：“ {} ” を使用してサブルーチンへ引数を渡すこともできます。例えば、${i}は、iの値が1の場合の$1と同じ意味です。

関数としてのサブルーチン

サブルーチンから、データを返すこともできます。値を返すサブルーチンを “ 関数 ” と呼びます。(値を返すコマンドも “ 関数 ” と呼びます。)

次の行は、文字列のデータ長を返す**Length**関数を用いたステートメントです。このステートメントは、**Length**関数が “ 長さ ” という変数に値を返します：

```
長さ:= Length ("How did I get here?")
```

どのようなサブルーチンでも値を返すことができます。返す値を、ローカル変数$0に格納します。例えば、“ Uppercase4 ” という次の関数は、始めの4文字を大文字に変換した文字列を返します：

```
$0:=Uppercase (Substring ($1 ; 1 ; 4)) + Substring ($1 ; 5)
```

次は、Uppercase4関数を使用した例です：

```
NewPhrase:=Uppercase4 ("This is good.")
```

この例では、変数「NewPhrase」に “ THIS is good. ” が格納されます。

関数の結果である $0 は、サブルーチン内のローカル変数です。サブルーチン内では、ユーザは他のローカル変数を使用するときと同様に$0 を使用できます。(サブルーチンが終了したときのままの) $0 の値を呼ばれたメソッドに返すのは 4D です。

再帰プロジェクトメソッド

プロジェクトメソッドは、自分自身を呼び出すことができます。例えば、

■ メソッドAは、Aを呼び出すメソッドBを呼び出します。このため A が再び Bを呼び出します。

■ メソッドA は自身を呼び出すことができます。

これは、再帰性と呼ばれています。4D 言語は再帰性を完全にサポートしています。

ここに例題があります。2つのフィールドで構成された [友人関係]テーブルを持っているとします：

- [友人関係] 名前
- [友人関係] 子供' 名前

この例題では、フィールド内の値は重複しない（つまり、同じ名前の人がいない）ものとします。名前を与え、次に以下の文を組み立てます。「私の友人であり、正雄の子供である弘子の子供である光男の子供である祐也の子供である栄治は生活のためにこれを行っています。」

1. ユーザは、次のように文を組み立てることができます。

```
$vsName:=Request("名前を入力してください";"正雄")
If (OK=1)
    QUERY([友人関係];[友人関係]名前=$vsName)
    If (Records in selection([友人関係])>0)
        $vtTheWholeStory:="私の友人であり、"+$vsName
        Repeat
            QUERY([友人関係];[友人関係]子供'名前=$vsName)
            $vlQueryResult:=Records in selection([友人関係])
            If ($vlQueryResult > 0)
                $vtTheWholeStory:=[友人関係]名前+"の子供である "+$vtTheWholeStory
                $vsName:=[友人関係]名前
            End if
        Until ($vlQueryResult=0)
        $vtTheWholeStory:=$vtTheWholeStory+"は生活のためにこれを行っています。"
        ALERT($vtTheWholeStory)
    End if
End if
```

2. また、ユーザは次のように文を組み立てることもできます。

```
$vsName:=Request("名前を入力してください";"正雄")
If (OK=1)
    QUERY([友人関係];[友人関係]名前=$vsName)
    If (Records in selection([友人関係])>0)
       ALERT("私の友人であり、"+世代 ($vsName)+"は生活のためにこれを行っています。")
    End if
End if
```

次は 、「世代」再帰性関数のコードです。

```
` 「世代」プロジェクトメソッド
` 世代 (文字列) → テキスト
` 世代 (名前) → 文の一部
$0:=$1
QUERY([友人関係];[友人関係]子供'名前=$1)
If (Records in selection([友人関係])>0)
   $0:=$0+"の子供である "+世代 ([友人関係]名前)
End if
```

自分自身を呼び出す「世代」メソッドであることに注意してください。

最初に挙げた方法は反復性のアルゴリズムです。2番目に挙げた方法は再帰性のアルゴリズムです。

前述の例題のような場合にコードを履行する場合、反復性や再帰性を使用してメソッドを書くことが常にできるということに注意してください。

一般的に、再帰性はより明瞭で、読みやすく、維持しやすいコードを提供します。ただし、この使用は強制ではありません。

4D 内での再帰性の代表的な使用方法は次の通りです。

■ 例題と同じく、お互いに関連するテーブル内でのレコードの取り扱い。

■ **FOLDER LIST**コマンドと**DOCUMENT LIST**コマンドを使用してのユーザディスク上にあるドキュメントとフォルダのブラウズ処理。フォルダにはフォルダとドキュメントが含まれており、サブフォルダはフォルダとドキュメントとその他を含むことができます。

重要：再帰性の呼び出しは、常にある地点で終了する必要があります。例えば Genealogy メソッドが自身の呼び出しを止めるのは、クエリーがレコードを返さないときです。この条件のテストをしないと、メソッドは際限なく自身を呼び出します。最終的に、4D は “ スタックがいっぱいです。” エラーを返します。その理由は 4D がすでに呼び出しを「蓄積する」容量を持たないためです (メソッド内で使用される引数とローカル変数についても同様です) 。

データベースメソッド

データベースメソッドは、通常のセッションイベントが発生すると、4th Dimensionが自動的に実行するメソッドです。

データベースメソッドを作成するかまたは開いて編集する方法は、次のとおりです。

1. **「エクスプローラ」ウィンドウを開く。**
2. **「メソッド」タブを選択する。**
3. **「データベースメソッド」テーマを拡げる。**
4. **メソッド名をダブルクリックします。**

または、次のように実行します。

1. **メソッドを選択する。**
2. **「編集」ボタンをクリックするか、EnterキーまたはReturnキーを押す。**

データベースメソッドは、他のメソッドと同じ方法で編集します。

データベースメソッドは、他のメソッドから呼び出すことはできません。データベースメソッドは、作業セッション中のある時点で、4th Dimensionから自動的に起動されます。次の表は、データベースメソッドの実行の概要を示しています。

データベースメソッド	4th Dimension	4D Server	4D Client
On Startup	○、1回	×	○、1回
On Exit	○、1回	×	○、1回
On Web Connection	○、複数	○、複数	×
On Server Startup	×	○、1回	×
On Server Shutdown	×	○、1回	×
On Server Open Connection	×	○、複数	×
On Server Close Connection	×	○、複数	×

「On Startup」データベースメソッド

「On Startup」データベースメソッドは、データベースを開くと呼び出されます。

この状態は、次のような4D環境で発生します。

- 4th Dimension
- 4D Client (クライアント側で、接続が4D Serverにより受け付けられた後)
- 4D Runtime
- 4D Compilerコンパイラおよび4D Engineでコンパイルされたりマージされた4Dアプリケーション

注：「On Startup」データベースメソッドは、4D Serverによって起動されることはありません。

「On Startup」データベースメソッドは、4Dによって自動的に起動されます。プロジェクトメソッドとは異なり、このデータベースメソッドをユーザが呼び出すことはできません。「On Startup」データベースメソッド内から作業を呼び出して実行するには、後に紹介しているプロジェクトメソッドの場合と同様に、サブルーチンを使用します。

「On Startup」データベースメソッドは、次のような処理に最適です。

- 作業セッション全体で使用するインタープロセス変数を初期化する。
- データベースを開いた時にプロセスを自動的に開始する。
- 以前の作業セッション中にこの目的で保存された初期設定やシステム定義をロードする。
- 条件が一致しない(システムリソースがないなどの)場合には、明示的に**QUIT 4D**コマンドを呼び出して、データベースを開かないようにする。
- データベースを開く度に自動的に実行したい他の動作を実行する。

バージョン3の4th Dimensionとの互換性

データベースメソッドは、バージョン6で導入された新しいメソッドです。バージョン3の4th Dimensionでは、データベースを開いた時に4Dが自動的に実行するメソッドは「Startup」プロシージャだけでした。バージョン3のデータベースをバージョン6に変換し、新しい「On Startup」データベースメソッドの機能を活用したい場合には、下図の「データベースプロパティ」ダイアログボックスにある「旧バージョンのStartupプロシージャ方式を使用する」チェックボックスを選択解除する必要があります。このプロパティは「Startup」メソッドおよび「On Startup」データベースメソッドの代替方法の切り替えだけに影響を与えます。 このプロパティを非選択にせずに、例えば「On Exit」データベースメソッドを追加すると、この後者のデータベースメソッドが4Dによって起動されます。

「On Exit」データベースメソッド

「On Exit」データベースメソッドは、データベースを終了すると呼び出されます。

この状態は、次のような4D環境で発生します。

- 4th Dimension
- 4D Client (クライアント側で、接続が4D Serverにより受け付けられた後)
- 4D Runtime
- 4D Compilerコンパイラおよび4D Engineでコンパイルされたりマージされた4Dアプリケーション

注：「On Exit」データベースメソッドは、4D Serverによっては起動されません。

「On Exit」データベースメソッドは、4Dによって自動的に起動されます。プロジェクトメソッドとは異なり、このデータベースメソッドをユーザが呼び出すことはできません。「On Exit」データベースメソッド内から作業を呼び出して実行するには、後に紹介しているプロジェクトメソッドの場合と同様に、サブルーチンを使用します。

データベースは、次のうちいずれかの状態になると終了します。

- ユーザが「ユーザ」モードまたは「デザイン」モードの「ファイル」メニューから「終了」メニューコマンドを選択した場合
- **QUIT 4D**コマンドへの呼び出しが実行された場合
- 4Dのプラグインソフトから**QUIT 4D**入力ポイントへの呼び出しが実行された場合

データベースの終了がどのような方法で実行されたかに関わらず、4Dは次のような処理を実行します。

■「On Exit」データベースメソッドがない場合には、4Dは実行プロセスそれぞれを区別せずに1つずつアボートします。ユーザがデータ入力を実行している場合には、レコードはキャンセルされ、保存されません。

■「On Exit」データベースメソッドがある場合には、4Dは新しく作成されたローカルプロセスの中でこのメソッドの実行を開始します。従って、このデータベースメソッドを使用して、インタープロセス間通信を通じて、(データ入力を)終了したり、処理の実行を中止したりしなければならないことを、他のプロセスに通知することができます。4Dは、いずれ終了するということに注意が必要です。「On Exit」データベースメソッドは、必要なクリーンアップや終了の処理を実行することができますが、中断処理を拒否することができず、ある時点で終了することになります。

「On Exit」データベースメソッドは、次のような処理を実行するには最適です。

■ データベースを開いた時に自動的に開始されたプロセスを停止する。

■「On Startup」データベースメソッドの次のセッションの開始時に再使用される初期設定やシステム定義を(ローカルに、ディスク上に)保存する。

■ データベースを終了する度に実行したい処理が他にあれば、それを実行する。

▼ 次の例では、作業セッション中に発生する重要なイベントを把握し、その内容説明を“ジャーナル”という名前のテキストドキュメントに書き込むために使用します：

■ 「On Startup」データベースメソッドは、すべての使用プロセスにデータベースが実行されているかどうかについて通知するインタープロセス変数「<>vbQuit4D」を初期化します。このデータベースメソッドは、ジャーナルファイルがまだない場合には、これを作成します。

```
` 「On Startup」データベースメソッド
C_TEXT(<>vtIPMessage)
C_BOOLEAN(<>vbQuit4D)
<>vbQuit4D:=False
If (Test path name("ジャーナル")  #  Is a document)
   $vhDocRef:=Create document("ジャーナル")
   If (OK=1)
      CLOSE DOCUMENT($vhDocRef)
   End if
End if
WRITE JOURNAL ("セッションのオープン")
```

▼「WRITE JOURNAL」プロジェクトメソッドは、他のメソッドのサブルーチンとして使用され、受け取った情報をジャーナルファイルに書き込みます。

```
` 「WRITE JOURNAL」プロジェクトメソッド
` WRITE JOURNAL (テキスト)
` WRITE JOURNAL (イベントの内容)
C_TEXT($1)
C_TIME($vhDocRef)
While (Semaphore("$Journal"))
   DELAY PROCESS(Current process ; 1)
End while
$vhDocRef:=Append document("ジャーナル")
If (OK=1)
   PROCESS PROPERTIES(Current process ; $vsProcessName ; $vlState ;
         vlElapsedTime ; $vbVisible)
   SEND PACKET($vhDocRef;String(Current date)+Char(9)+String(Current time)+Char(9)
            +String(Current process)+Char(9)+$vsProcessName+Char(9)+$1+Char(13))
   CLOSE DOCUMENT($vhDocRef)
End if
CLEAR SEMAPHORE("$Journal")
```

文書（ドキュメント）は、毎回開いて閉じられることに注意が必要です。さらに、文書へのアクセス保護としてセマフォを利用していることにも注意が必要です。2つのプロセスがジャーナルファイルに同時にアクセスすることを防ぐためです。

▼「M_ADD_RECORDS」プロジェクトメソッドは、「カスタム」メニューから「レコード追加」メニュー項目が選択されると実行されます。

```
` 「M_ADD_RECORDS」プロジェクトメソッド
MENU BAR(1)
Repeat
   ADD RECORD([テーブル1] ; *)
   If (OK=1)
      WRITE JOURNAL ("テーブル1：" + "追加レコード#"+String(Record number([テーブル1])))
   End if
Until ((OK=0)  |  <>vbQuit4D)
```

このメソッドは、ユーザが最後のデータ入力をキャンセルするか、またはデータベースを終了するまでループします。

▼ [テーブル1]の入力フォームには、「On Outside Call」イベントの処理手順も含まれています。従って、プロセスがデータ入力中であっても、ユーザは現在のデータ入力を保存するかまたは保存しないで、スムーズに終了できます：

```
` [テーブル1];"入力"フォームのフォームメソッド
Case of
    \ (Form event=On Outside Call)
        If (<>vtlPMessage="終了")
            CONFIRM("このレコードで行った変更を保存したいですか？")
            If (OK=1)
                ACCEPT
            Else
                CANCEL
            End if
        End if
End case
```

▼「M_QUIT」プロジェクトメソッドは、「ファイル」メニューから「終了」が選択されると実行されます：

```
`「M_QUIT」プロジェクトメソッド
$vlProcessID:=New process("DO_QUIT" ; 32*1024 ; "$DO_QUIT")
```

このメソッドには、ある仕掛けがあります。**QUIT 4D**コマンドが呼び出されると、このコマンドは即座に有効になります。従って、呼び出しが実行されているプロセスは、データベースが実際に終了されるまでの間は「停止モード」になります。このプロセスは、データ入力が発生するプロセスのうちのいずれかである可能性があるため、**QUIT 4D**コマンドへの呼び出しは、この目的でだけ開始されるローカルプロセス内で実行されます。「DO_QUIT」メソッドは、次のようになります：

```
`「DO_QUIT」プロジェクトメソッド
CONFIRM("終了してもいいですか？")
If (OK=1)
   WRITE JOURNAL ("データベースの終了")
   QUIT 4D
      `QUIT 4Dは即座に有効になるため、それ以降のコード行はすべて実行されません。
      ` ...
End if
```

▼ 最後に、すべての開いているユーザプロセスに対して「ただちに終了するように」と通知する「On Exit」データベースメソッドは、次のようになります。このメソッドは、<>vbQuit4Dを**True**に設定し、データ入力を実行しているユーザプロセスに、プロセス間メッセージを送信します：

```
` 「On Exit」データベースメソッド
<>vbQuit4D:=True
Repeat
   $vbDone:=True
   For ($vlProcess ; 1 ; Count tasks)
      PROCESS PROPERTIES($vlProcess ; $vsProcessName ; $vlState
         ; $vlElapsedTime ; $vbVisible)
      If ((($vsProcessName="ML_@")  |  ($vsProcessName="M_@")))  &  ($vlState>=0))
         $vbDone:=False
         <>vtIPMessage:="QUIT"
         BRING TO FRONT($vlProcess)
         CALL PROCESS($vlProcess)
         $vhStart:=Current time
         Repeat
            DELAY PROCESS(Current process ; 60)
         Until((Process state($vlProcess)<0)  |  (Current time-$vhStart)>=?00:01:00?))
      End if
   End for
Until ($vbDone)
WRITE JOURNAL ("セッションのクローズ")
```

注："ML_..."または"M_..."で始まる名前を持つプロセスは、「新規プロセス開始」プロパティを選択したメニューコマンドによって開始されます。この例では、そのようなプロセスは、「レコード追加」メニューコマンドが選択された時に開始されたプロセスです。

「(Current time-$vhStart)>=?00:01:00?」というテストを実行すると、データベースメソッドは「他のプロセスを待っている」状態を終了します。他のプロセスがただちに反応しない場合には、ループを繰り返します。

▼ 次に示しているのは、データベースによって作成されたジャーナルファイルの代表的な例です：

```
97/8/21   15:47:25   1   ユーザ/カスタムメニュープロセス  セッションのオープン
97/8/21   15:55:43   5   ML_1                          テーブル1：レコード追加#23
97/8/21   15:55:46   5   ML_1                          テーブル1：レコード追加#24
97/8/21   15:55:54   6   $DO_QUIT                      データベースの終了
97/8/21   15:55:58   7   $xx                           セッションのクローズ
```

注：$xxという名前は、「On Exit」データベースメソッドを実行するために4Dが開始したローカルプロセスの名前です。

「On Server Startup」データベースメソッド

「On Server Startup」データベースメソッドは、バージョン6の4D Serverがリリースされた際に説明します。

「On Server Shutdown」データベースメソッド

「On Server Shutdown」データベースメソッドは、バージョン6の4D Serverがリリースされた際に説明します。

「On Server Open Connection」データベースメソッド

「On Server Open Connection」データベースメソッドは、バージョン6の4D Serverがリリースされた際に説明します。

「On Server Close Connection」データベースメソッド

「On Server Close Connection」データベースメソッドは、バージョン6の4D Serverがリリースされた際に説明します。

データベースアプリケーション　7

アプリケーションとは、特定の要求を満たすために作成する一種のデータベースです。アプリケーションは、その使用を容易にするためのユーザインターフェースを備えています。4th Dimensionの機能は、そのアプリケーションの機能に依ります。しかし、4th Dimensionで作成したアプリケーションは、従来のプログラミング言語で作成されたものに比べて無駄なく簡単に操作することができます。4th Dimensionは、次のようなアプリケーションを作成することができます：

- 請求書発行システム
- 在庫管理システム
- 会計システム
- 給与システム
- 人事システム
- 顧客管理システム
- インターネットまたはイントラネット経由の共有データベース

1つのアプリケーションがこれらのシステムすべてを含むことも可能です。これらのようなアプリケーションは、データベースを使用するのが普通です。加えて、4th Dimensionが持つツールにより、以下のような先進的なアプリケーションを作成することができます。

- 文書管理システム
- グラフィックイメージ管理システム
- カタログ発行アプリケーション
- シリアルデバイス制御と監視システム
- 電子メイルシステム
- マルチユーザスケジューリングシステム
- メニューリスト、ビデオコレクション、ミュージックコレクションのようなリスト

アプリケーションは、一般に「ユーザ」モードで使用するデータベースとして作成されます。そのデータベースからカスタマイズされたアプリケーションへ“拡張”させることができます。アプリケーションが通常のデータベースと異なる点は、データベースを管理するのに必要なプロセスはユーザから見えないようになっていることです。ユーザは、メニューを使用して処理を実行します。

「ユーザ」モードのデータベースで何らかの処理を行う場合には、その手順を記憶させる必要があります。アプリケーションは「カスタム」モードで起動します。「カスタム」モードでは、「ユーザ」モードで自動化される機能を管理する必要があります。これには、次のようなものがあります。

■**「テーブル操作」：**「テーブル選択」ダイアログボックスと“ テーブルリスト ”ウインドウは使用することができません。テーブル間の操作を制御するには、メニューコマンドやオブジェクトメソッドを用います。

■**「メニュー」：**「カスタム」モードには、デフォルトでは「終了」メニューコマンドのある「ファイル」メニューと「編集」メニューの2つしか用意されていません。アプリケーションに多くのメニューを用いる場合は、メニューを作成し、管理しなければなりません。

■**「エディタ」：**すべてのエディタが、「カスタム」モードでは自動的に使用することはできません。アプリケーションでメニューコマンドやエディタを用いる場合は、これらを管理しなければなりません。

次の節では、プログラミング言語によるデータベース使用の自動化について説明します。

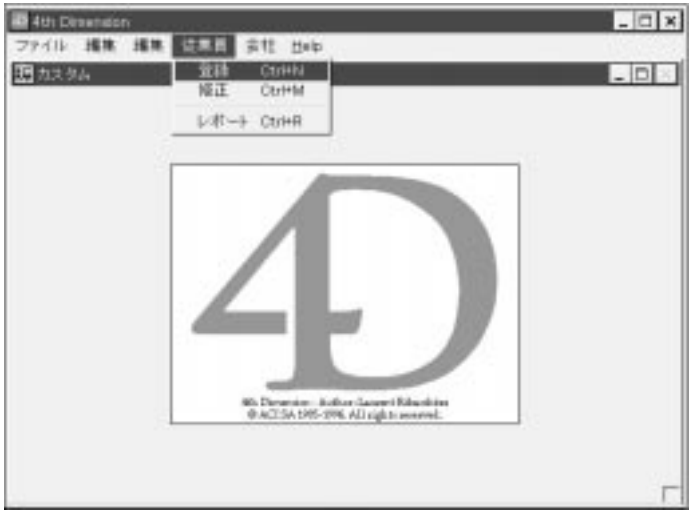
カスタムメニューの例

カスタムメニューは、データベースの代表的なインタフェースです。これは、データベースの使用を容易にします。カスタムメニューは、「メニュー」エディタを使用して、各メニューコマンドにメソッド名を連結するだけで簡単に作成することができます。

次の節では、ユーザがメニューを選択したときに起こる事象について説明します。事例はとても簡単ですが、データベースの使用や学習を行うには十分なものになっています。ユーザは、4th Dimensionの“一般的な”ツールや「ユーザ」モードのメニューコマンドを見るのではなく、ユーザ自身が必要とする事象のみを見ます。

ユーザの認識

ユーザは新しい従業員を追加するために、“登録”というカスタムのメニューコマンドを選択します。



注：ここで使用している図の中には、バージョン3の4th Dimensionをもとに作成されたものが一部含まれています。

従業員テーブルの入力フォームが表示されます。

ユーザは、従業員の名字を入力し、次のフィールドへ移動します。

ユーザは、従業員の名前を入力し、次のフィールドへ移動します。

ユーザは、名前が大文字に変換されたことを確認します。

ユーザは、レコードの入力を終了すると、「登録」ボタンをクリックします。

空の入力フォームが表示されるので、ユーザは「キャンセル」ボタンをクリックして「データ入力ループ」を終了します。再びメニューバーが表示されます。

メニュー表示の裏側

メニューは、「デザイン」モードの「メニュー」エディタを使って作成します。

メニューコマンドの“ 登録 ”には、“ 従業員登録 ”という名前のプロジェクトメソッドが付随しています。このメソッドは、「デザイン」モードの「メソッド」エディタで作成されたものです。

メソッド“ 従業員登録 ”は、次のように実行します：

```
Repeat
   ADD RECORD ([従業員])
Until (OK=0)
```

ADD RECORDコマンドが指定された「**Repeat...Until**」ループは、「ユーザ」モードにおける「新規レコード...」メニューコマンドと同様の動作を行います。これにより、入力フォームがユーザに表示され、ユーザが新規レコードを追加できるようになります。ユーザがレコードを保存すると、空の入力フォームが表示されます。この**ADD RECORD**ループは、ユーザが「キャンセル」ボタンをクリックするまで実行を続けます。

レコードを入力すると、次のことが起こります。

■ 「名前」フィールドにはオブジェクトメソッドはないので、何も実行されません。

■ 「名字」フィールドにはオブジェクトメソッドがあるので、「フォーム」エディタと「メソッド」エディタを使用して、このオブジェクトメソッドが「デザイン」モードに作成されています。メソッドは以下を実行します。

名前:=Uppercase (名前)

これは、名前のフィールドを大文字に変換します。

レコードが入力され、ユーザが次のフォームで「キャンセル」ボタンをクリックすると、OK変数が0に設定されて**ADD RECORD**ループが終了します。

それ以上実行するステートメントがないので、“ 従業員登録 ” メソッドは実行を終了し、制御がメニューバーに戻ります。

「ユーザ」モードとアプリケーションの比較

「ユーザ」モードで処理を実行する方法と、プログラミング言語で同じ処理を実行する方法とを比較してみましょう。実行する処理は、両方ともレコードの検索、ソート、およびレポートの印刷です。この章の例では、「ユーザ」モードと「カスタム」モードの両方で4th Dimensionの組み込みエディタを使用します。この場合、プログラミング言語は処理を部分的に自動化します。

次の節の “「ユーザ」モードにおけるデータベースの使用 ” では、「ユーザ」モードで実行される処理を示します。また “ 組み込みエディタを使用したアプリケーション ” の節では、アプリケーションで実行される同じ処理を表示します。

どちらも同じ処理を実行しますが、2番目の節の行程ではプログラミング言語の使用により自動化されている点に注目してください。

「ユーザ」モードにおけるデータベースの使用

ユーザは、「クエリ」メニューから「検索」を選択します。

「クエリ」エディタが表示されます。

ユーザが検索条件を入力して「OK」ボタンをクリックすると、検索が実行されます。

次にユーザは、「クエリ」メニューから「並び替え...」を選択します。

「並び替え」エディタが表示されます。

ユーザは、ソート条件を入力し「並び替え」ボタンをクリックします。

最後にユーザは、「ファイル」メニューから「プリント...」を選択します。

「プリントフォーム選択」ダイアログボックスが表示されます。

この場合、ユーザはどのフォームを選択したかを覚えておく必要があります。フォームを選択し、「OK」ボタンをクリックします。

「用紙設定」ダイアログボックスが表示されます。

ユーザが「OK」ボタンをクリックすると、レポートが印刷されます。

組み込みエディタを使ったアプリケーションの使用

ユーザは、カスタムメニューから該当するメニューコマンド「レポート」を選択します。

アプリケーションの使用は、この時点で既にユーザにとって容易になっています。ユーザは、最初の手順が検索であることを認識している必要はありません。

メソッド“ 従業員印刷 ” は、メニューコマンドに付随しています。これは次のようなメソッドです：

QUERY ([従業員])
ORDER BY ([従業員])
OUTPUT FORM ([従業員] ; "印刷")
PRINT SELECTION ([従業員])

最初の行が実行されます。

QUERY ([従業員])

「クエリ」エディタが表示されます。

ユーザが検索条件を入力し「クエリ」ボタンをクリックすると、検索が実行されます。

次にメソッド“ 従業員印刷 ”の2行目が実行されます：

QUERY BY ([従業員])

ユーザは、ソートが次のステップであることを知っておく必要がないことに注目してください。

「並び替え」エディタが表示されます。

ユーザがソート条件を入力し「並び替え」ボタンをクリックすると、ソートが実行されます。

次にメソッド“ 従業員印刷 ”の3行目が実行されます。

OUTPUT FORM ([従業員] ; "印刷")

ここでもユーザは、次に行うべきことを記憶しておく必要がありません。処理手順は、既に定義されています。

最後にメソッド“ 従業員印刷 ”の最終行が実行されます。

PRINT SELECTION ([従業員])

「用紙設定」ダイアログボックスが表示されます。

ユーザが「OK」ボタンをクリックすると、レポートが印刷されます。

高度に自動化されたアプリケーション

前述の比較で使用したものと同じコマンドで、より高度に自動化されたアプリケーションを作成することができます。

次の比較では、最初に前例のアプリケーションで組み込みエディタを使用したものを示します。その次に、プログラミング言語で完全に処理を自動化した例を示します。この場合、ユーザが唯一知っておかなければならないことはレポートを作成するために「レポート」メニューを選択しなければならないということだけです。必要な動作をより完全に記述するために同じコマンドがどのように使用されているか注目してください。

完全に自動化されたアプリケーション

ユーザは、メソッドを開始するカスタムのメニューコマンドを選択します。

メソッド“ 従業員印刷 ”は、メニューコマンドに付随しています。このメソッドは、次のようになっています：

```
QUERY ([従業員] ; [従業員]会社 = "刀根エンタープライズ")
ORDER BY ([従業員] ; [従業員]名字 ; > ; [従業員]名前 ; >)
OUTPUT FORM ([従業員] ; "印刷")
PRINT SELECTION ([従業員] ; *)
```

最初の行が実行されます：

```
QUERY ([従業員] ; [従業員]会社 = "刀根エンタープライズ")
```

「クエリ」エディタは表示されません。その代わり、**QUERY**コマンドに指定した検索条件が実行されます。ユーザは、何もする必要がありません。

メソッド“ 従業員印刷 ”の2行目が実行されます：

```
ORDER BY ([従業員] ; [従業員]名字 ; > ; [従業員]名前 ; >)
```

この場合も「並べ替え」ダイアログボックスは表示されずにソートがすぐに実行されます。ここでもユーザは何もする必要がありません。

メソッド“ 従業員印刷 ”の最終行が実行されます：

```
OUTPUT FORM ([従業員] ; "印刷")
PRINT SELECTION ([従業員] ; *)
```

PRINT SELECTIONコマンドは、「用紙設定」ダイアログボックスを表示しません。出力フォームの作成時点に指定した値を用いて印刷されますので、引数のアスタリスク(*)を指定してレポートを印刷します。

「ユーザ」モードメニューと同等のコマンド

これまでの例でもわかるように、プログラミング言語には、「ユーザ」モードのメニューコマンドと同じ動作を実行するコマンドが数多く存在します。これらのコマンドは、データベースをカスタマイズをする上で便利な機能を提供します。

各メニューコマンドは、動作を実行させたり、エディタやダイアログボックスを表示します。コマンドを使用して、エディタと動作を“結合”したカスタムシーケンスで、自動的に反復処理を行うことができます。また、これらのコマンドは、エディタなどからユーザによる割り込みを受けなくても、引数で指定されたとおりに検索などの動作を自動的に実行することができます。

次の表は、「ユーザ」モードのメニューコマンドと、それに相当するコマンドと使用例を示したものです。

メニューコマンド	コマンド	使用例
フォーミュラで更新	APPLY TO SELECTION	APPLY TO SELECTION ([従業員];フォーマット名)
テーブル / フォーム選択	DEFAULT TABLE	DEFAULT TABLE ([従業員])
	INPUT FORM	INPUT FORM ([従業員] ;入力)
	OUTPUT FORM	OUTPUT FORM ([従業員] ; 出力)
ASCIIテーブル編集	USE ASCII MAP	USE ASCII MAP ("テーブル名")
データ書き出し	EXPORT TEXT	EXPORT TEXT ([従業員];"")
チャート	GRAPH TABLE	GRAPH TABLE ([従業員])
データ読み込み	IMPORT TEXT	IMPORT TEXT ([従業員];"")
ラベル	PRINT LABEL	PRINT LABEL ([従業員];"")
レコード修正	MODIFY RECORD	MODIFY RECORD ([従業員])
新規レコード	ADD RECORD	ADD RECORD ([従業員])
プリント	PRINT SELECTION	PRINT SELECTION ([従業員])
クイックレポート	REPORT	REPORT ([従業員];"")
フォーミュラで検索	QUERY BY FORMULA	QUERY BY FORMULA ([従業員]年齢=20)
フォームで検索	QUERY BY EXAMPLE	QUERY BY EXAMPLE([従業員];"検索")
検索	QUERY	QUERY ([従業員])
すべてを表示	ALL RECORDS	ALL RECORDS ([従業員])
	MODIFY SELECTION	MODIFY SELECTION ([従業員])
並び替え	ORDER BY	ORDER BY ([従業員])

アプリケーション開発用ツール

ACI社からアプリケーションの開発に使用できるいくつかのツールが提供されています。これらのツールを使用して、別のデータベースからファイルやフィールド等のオブジェクトを移動したり、データベースのコンパイル、データベースのシンタックスチェックを行うことができます。次のようなツールがあります。

■ **「4D Insider」：**このツールは、別のデータベースからテーブル、フィールド、メソッド、メニューバー、リスト、外部モジュール等を移動することができます。また、4th Dimensionデータベースのクロスリファレンスを作成することもできます。このツールはプロシージャや変数、コマンド、外部ルーチン、ストラクチャ、リスト、フォーム等を表示したり、印刷する場合に使用します。

■ **「4D Compiler」：**このツールは、アセンブラ命令でメソッドやオブジェクトメソッドを解析します。このツールはデータベースの処理速度を高速にし、コードの整合性をチェックします。また、論理的やシンタックス上の矛盾を発見したり、データベースを保護することができます。

これらのユーティリティに関する詳細は、それぞれのツールに付属するマニュアルを参照してください。

4D プラグイン

4D アプリケーションの能力は、4D開発環境に専門のプラグインを追加することにより拡張することができます。

ACIは、以下の4Dプラグインを提供しています。

- **4D Calc：**スプレッドシート
- **4D Draw：**グラフィカル描画プログラム

ACIは、また以下の接続プラグインも提供しています。

- **4D ODBC：**ODBC経由の接続
- **4D for Oracle：**ORACLEデータベースとの接続
- **4D SQL Server：**SYBASE SQL ServerおよびMicrosoft SQL Serverとの接続
- **4D Open：**分散4D情報システムを構築するための4D間接続

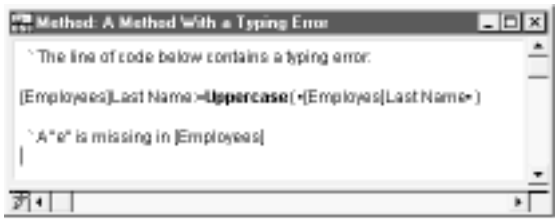
なぜデバッガを使用するか?

メソッドを開発しテストする際には、エラーを発見し修正することが重要です。

言語を使用する場合にありがちなエラーには、入力エラー、シンタックス（構文）エラー、環境エラー、設計やロジックのエラー、実行時エラーなど、いくつかの種類があります。

入力エラー

入力エラーは、「メソッド」エディタによって検出され、黒丸(•)によって示されます。次のウィンドウは、入力エラーを表しています：



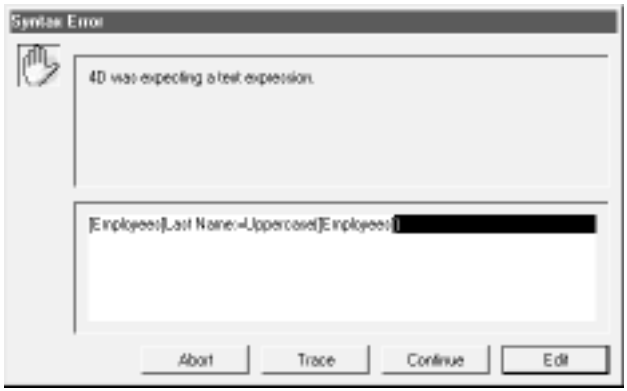
注：コメントは、このマニュアルで説明するために手動で入力しています。4Dがエラーの発生位置で挿入するのは(•)だけです。

このようになった場合には、入力エラーを修正し(数値キーパッド上で)、Enterキーを押して、修正内容を確認します。

「メソッド」エディタに関する詳細は、『4th Dimension / 4D First デザインリファレンス』を参照してください。

シンタックス（構文）エラー

シンタックス（構文）エラーは、メソッドを実行する時に検出されます。シンタックスエラーが発生すると、次ページの図のように「シンタックスエラー」ウィンドウが表示されます。

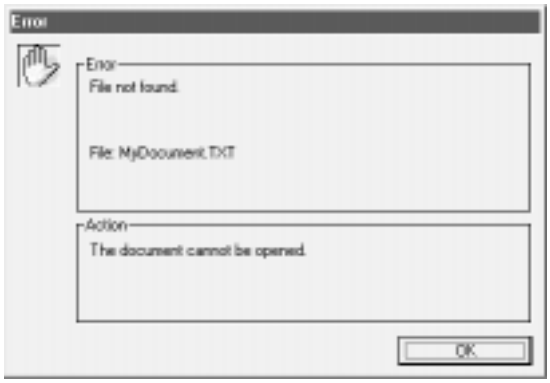


このウィンドウでのエラーは、テキスト式が対象となっている**Uppercase**関数にテーブル名が渡されていることです。このウィンドウとボタンの詳細については、後述の「シンタックスエラーウィンドウ」の節を参照してください。

環境エラー

まれに、配列やBLOBを作成するために十分なメモリがない場合があります。

ディスク上のドキュメントにアクセスした時にドキュメントが存在しないか、またはすでに他のアプリケーションによって開かれている可能性があります。このような場合には次のように「エラー」ウィンドウが表示され、エラーの内容と実行できなかった操作が表示されます。



このようなエラーは、作成したコードやコードの作成方法が直接の原因となって発生するわけではありません。ただ単に「思わしくない状況が発生した」ことが原因となって発生します。ほとんどの場合、これらのエラーは、**ON ERR CALL**コマンドを使用してインストールした「エラー検出」メソッドによって、簡単に対応できます。詳細については、**ON ERR CALL**コマンドの説明個所を参照してください。

設計エラーとロジックエラー

これらは通常、発見することが最も難しいタイプのエラーです。検出するには、デバッガを使用します。

これまでに説明しているエラーは、入力エラーを除いて、「設計エラーとロジックエラー」という範疇に該当します。例は次のとおりです。

■ まだ初期化されていない変数を用いようとしたため、シンタックスエラーが発生する場合があります。

■ 引数の中で正しい値を取得していないサブルーチンによって名前が受け取られたドキュメントを開こうとしたため、環境エラーが発生する場合があります。この場合には、実際に中断が発生しているコードの一部分が、問題の原因とは異なっている場合があることに注意が必要です。

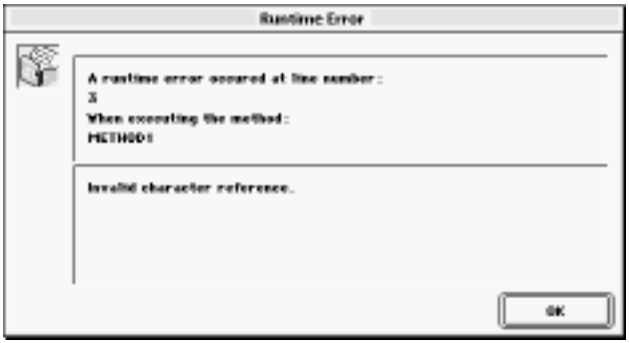
設計やロジックのエラーには、次のような場合もあります。

■ **SAVE RECORD**コマンドを呼び出す時に、対象となるレコードがロックされているかどうかを最初にテストしなかったために、レコードが正しく更新されていません。

■ オプション引数を追加した状態がテストされていないため、メソッドが想定どおりに動作しません。

実行時エラー

コンパイルモードでは、インタプリタモードでは決して見られない次のようなエラーが発生する場合があります。



これは、「**Position**関数が文字列の長さを超えるような文字にアクセスしようとしている」という意味です。

問題の原因を迅速に発見するには、メソッドの名前と行番号を記録し、ストラクチャファイルのインタプリタバージョンを再び開いて、メソッドの指定された行を確認します。

エラーが発生した場合の対応

エラーは、日常的に発生します。相当行数(数百行程度)のコードをエラーが発生しないように作成することは、非常にまれです。むしろ、エラーに対応したり、修正することのほうが普通の状態です。

4th Dimensionはマルチタスク対応アプリケーションなので、ウィンドウを切り替えるだけで、すばやくメソッドを編集し実行することができます。その度メソッド全体を再実行する必要がないので、失敗やエラーを非常に迅速に修正できます。

エラー検出の際によくある失敗は、「シンタックスエラー」ウィンドウの「アボート」ボタンをクリックし、「メソッド」エディタに戻り、コードを表示して原因を確認しようとすることです。これは絶対に止めてください。常にデバッガを使用すれば、相当の時間と労力を節約することができます。

- 予想していないシンタックスエラーが発生した場合には、デバッガを使用します。
- 環境エラーが発生した場合には、デバッガを使用します。
- その他どのようなタイプのエラーが発生した場合でも、デバッガを使用します。

ほとんどの場合、デバッガは、エラーが発生した理由を知るために必要な情報を表示します。この情報があれば、エラーの修正方法はわかります。

Tips：デバッガの使用法を学習し、実際に試しておけば、エラーの原因を究明しなければならなくなった時に何日分、何週間分もの時間と労力をかけずにすむことなります。

デバッガを使用するもう一つの理由は、コードの作成です。いつも以上に複雑なアルゴリズムを作成してしまう場合があります。達成感こそありますが、コーディングが正しいかどうか、テストする前でもまったく確かではありません。見当もつかないままコーディングするのではなく、コーディングの前に**TRACE**コマンドを使用します。その後、コードを少しずつ実行して動作を制御し、予想どおりエラーが発生したかどうかを確認します。完全主義的な考え方では、このような方法は望ましくないかもしれませんが、実利主義的な方法が報われる場合もあります。いずれにしても、デバッガを使用してください。

一般的な結論

エラーが出たら、デバッガを使用する。

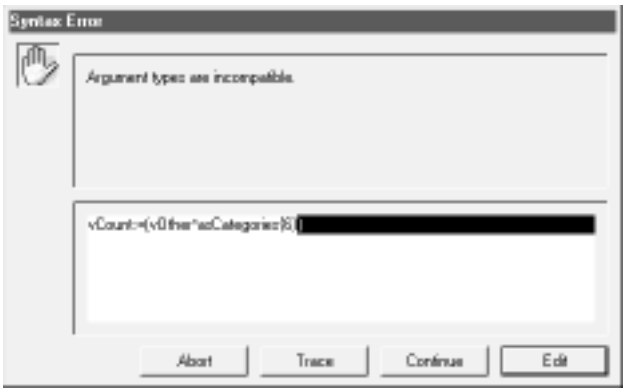
「シンタックスエラー」ウィンドウ

「シンタックスエラー」ウィンドウは、メソッドの実行が一時停止されると表示されます。メソッドは、次の2つのうちいずれかの理由で一時停止されます。

■ これ以上のメソッドの実行を防ぐため、4th Dimensionが実行を一時停止する場合があります。

■ メソッドが実行されている時に、「Alt -クリック(Windows)」または「option -クリック(Macintosh)」を実行してユーザ割り込みを発生させると、メソッドは一時停止します。

次の図は、「シンタックスエラー」ウィンドウです。



「シンタックスエラー」ウィンドウ上部のテキストエリアには、エラーの内容が表示されています。下部のテキストエリアには、エラーが発生した時に実行されていた行が表示され、エラーが発生した部分が強調表示されています。

ウィンドウの一番下の部分には、「アボート」、「トレース」、「続ける」、「編集」の4つのボタンがあります。

■ **「アボート」：**メソッドは一時停止され、メソッドの実行を開始する前の状態に戻ります。イベントに対してフォームメソッドまたはオブジェクトメソッドが実行されている場合には、いずれの場合にも停止され、フォームに戻ります。メソッドが「カスタム」モードから実行されている場合には、「カスタム」モードに戻ります。

■ **「トレース」：**「トレース／デバッガ」モードになったため、「デバッグ」ウィンドウが表示されています。現在の行が一部実行されている場合には、「トレース」ボタンを数回クリックする必要があるかもしれません。行の実行が終了すれば、「デバッグ」ウィンドウが表示されます。

■ **「続ける」：**実行は継続されます。エラーが発生した行は、その位置によっては一部実行される場合があります。その場合には、慎重に実行を継続する必要があります。エラーが原因で、メソッドが正常に実行できない場合があるためです。通常、この方法で

継続しないようにお勧めします。**SET WINDOW TITLE**コマンドなどのように、コードの残りの部分の実行やテストの妨げにならない単純な呼び出しでエラーが発生している場合には、「続ける」ボタンをクリックしても構いません。このようにすれば、より重要なコーディングに集中し、些細なエラーは後で修正することができます。

■**「編集」：**すべてのメソッドの実行は一時停止されます。4th Dimensionは「デザイン」モードに切り替わります。エラーが発生したメソッドが「メソッド」エディタで開かれ、エラーを修正することができます。このオプションは、エラーの内容がすぐわかり、これ以上調査せずに修正できる場合に使用します。

デバッガ

デバッガという用語は、バグに由来しています。メソッド内のバグとは、除去すべき間違いのことです。エラーが発生した場合、またはメソッドの実行を監視する必要がある場合には、デバッガを使用します。デバッガでは、メソッドをステップごとにゆっくり確認してメソッドの情報を検証できるため、バグを発見するために役立ちます。このようにメソッドをステップごとに確認する処理は、トレースと呼ばれます。

次のような方法を使用して、「デバッグ」ウィンドウでメソッドを表示し、トレースすることができます。

■「シンタックスエラー」ウィンドウで「トレース」ボタンをクリックした場合。

■ **TRACE**コマンドを使用した場合。

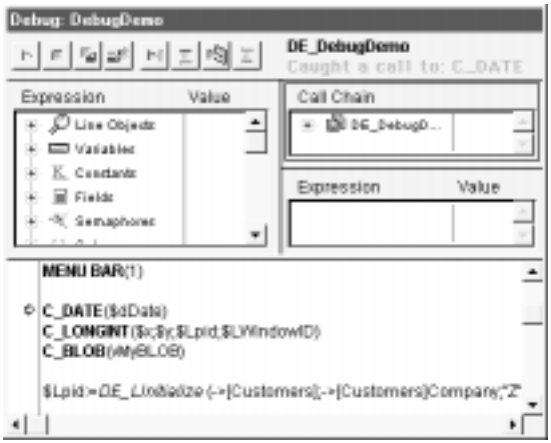
■ メソッドが実行されている時に「Alt -クリック(Windows)」または「option -クリック(Macintosh)」を実行した場合。

■「プロセスリスト」ウィンドウで選択されたプロセスに対して、「デザイン」モードのの「プロセス」メニューから「トレース」を選択した場合(「表示されないプロセスまたはコードを実行していないプロセスのトレース」の節を参照してください)。

■「ブレークリスト」ウィンドウで「ブレークポイント」を作成または編集した場合。

注：データベースにパスワードのシステムがある場合には、その構造へのアクセス特権を持つグループに所属する設計者やユーザだけが、メソッドをトレースできます。

次に表示されているのは、「デバッグ」ウィンドウです。

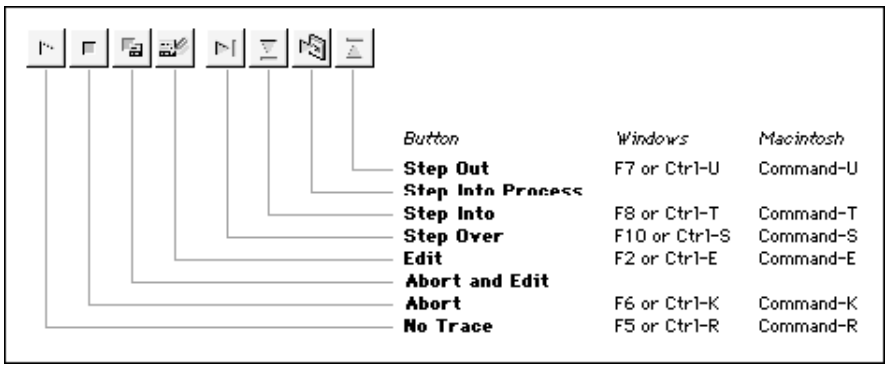


必要に応じて「デバッグ」ウィンドウを移動したり、その内部にあるウィンドウのサイズを変更することができます。

4th Dimensionは、マルチタスクの環境です。複数のユーザプロセスを実行した場合には、それぞれのプロセスを個別にトレースできます。プロセスそれぞれについて1つの「デバッグ」ウィンドウを開くことができます。

「実行コントロール」ツールバーボタン

「デバッグ」ウィンドウの上部にある「実行コントロール」ツールバーには、8個のボタンがあります。



Button	Windows	Macintosh
Step Out	F7 or Ctrl-U	Command-U
Step Into Process		
Step Into	F8 or Ctrl-T	Command-T
Step Over	F10 or Ctrl-S	Command-S
Edit	F2 or Ctrl-E	Command-E
Abort and Edit		
Abort	F6 or Ctrl-K	Command-K
No Trace	F5 or Ctrl-R	Command-R

「トレースなし（No Trace）」ボタン

トレースが一時停止され、通常のメソッドの実行が再開されます。

注：ALT+F5 (Windows)や「Option-Command-R」(Macintosh)を押すと、実行が再開されます。この操作により、現在のプロセスでの次のTRACEコマンド呼び出しも無効になります。

「アボート（Abort)」ボタン
メソッドは一時停止され、メソッドの実行を開始する前の状態に戻ります。イベントに対して実行しているフォームメソッドまたはオブジェクトメソッドをトレースしている場合には、いずれの場合にも停止され、フォームに戻ります。「カスタム」モードから実行しているメソッドをトレースしていた場合には、「カスタム」モードに戻ります。

「アボート&編集（Abort and Edit)」ボタン
「アボート」ボタンががクリックされたかのように、メソッドは一時停止されます。さらに、必要に応じて、4th Dimensionは「デザイン」プロセスを開いて前面に表示し、「アボート&編集」ボタンがクリックされた時点で実行していたメソッドの「メソッドエディタ」ウィンドウを開きます。

Tips：メソッドのテストを実行するにはコードにどのような変更が必要か、またいつその変更が必要かがわかっている場合には、このボタンを使用します。変更が完了したら、メソッドを再実行します。

「編集（Edit)」ボタン
「編集」ボタンをクリックすると、「アボート&編集」ボタンをクリックした場合と同じ動作が実行されますが、現在の実行をアボートしません。メソッドの実行はその時点で一時停止されます。必要に応じて、4th Dimensionは「デザイン」プロセスを開いて前面に表示し、「編集」ボタンがクリックされた時点で実行されていたメソッドを「メソッドエディタ」ウィンドウで開きます。

重要：このメソッドを修正することはできますが、「デバッグ」ウィンドウで現在トレースされているメソッドの場合には、そのような修正内容は表示されず、実行もされません。メソッドがアボートされるか、または正常に終了した後で、このメソッドが次に実行される時に修正内容が表示されます。つまり、メソッドへの変更を有効にするには、メソッドを再ロードしなければなりません。

Tips：このボタンは、コードのどの部分で変更が必要かを知る場合や、必要な変更が実行やトレースの対象となるコードの残りの部分と整合がとれていない場合に使用します。

Tips：オブジェクトメソッドは、各イベントごとに再ロードされます。オブジェクトメソッドを(ボタンのクリックに対して)トレースしている場合には、フォームを終了する必要はありません。オブジェクトメソッドを編集し、変更内容を保存し、フォームに戻って再実行することができます。フォームメソッドのトレースや変更の際にフォームメソッドを再ロードするには、フォームを終了し、再び開かなければなりません。フォームを大規模にデバッグする場合のコツは、(デバッグの対象となっている)コードを、フォームメソッドからのサブルーチンとして使用しているプロジェクトメソッドに入力することです。このようにすれば、このサブルーチンがフォームメソッドから呼び出されるつど再ロードされるため、フォームをトレースし、エディットし、再テストしている間もフォームを使用することができます。

「ステップ（同一メソッドのみ）（Step Over)」ボタン
現在のメソッド行(プログラムカウンタと呼ばれる黄色い矢印で示されている行)が実行され、デバッガは次の行に移動します。「ステップ（同一メソッドのみ)」ボタンはサブルーチンや関数に割り込むことはなく、現在トレースの対象となっているメソッドのレベルにとどまります。サブルーチンや関数呼び出しもトレースしたい場合には、「ステップ(呼び出しメソッドもトレース)」ボタンを使用します。

「ステップ（呼び出しメソッドもトレース）（Step Into}」ボタン
別のメソッド(サブルーチンまたは関数)を呼び出す行が実行される時にこのボタンを使用すると、呼び出されているメソッドが「デバッグ} ウィンドウに表示され、このメソッドをステップごとに実行できます。

「デバッグ} ウィンドウの「メソッド連鎖」エリアでは、新しいメソッドが現在(トップ)のメソッドになります。別のメソッドを呼び出していない行が実行される時には、このボタンは「ステップ（同一メソッドのみ)」ボタンと同等の動作します。

「ステップ（新規プロセスもトレース）（Step info process)」ボタン
新しいプロセスを作成する行が実行される時にこのボタンを使用すると、新しい「デバッグ」ウィンドウが開かれ、新しく作成されたプロセスのプロセスメソッドをトレースすることができます。新しいプロセスを作成しない行が実行される時には、このボタンは「ステップ（同一メソッドのみ)」ボタンと同等の動作します。

「ステップ（呼び出し元へ戻る）{Step Out)」ボタン
サブルーチンや関数をトレースする場合にこのボタンをクリックすると、現在トレースされているメソッド全体を実行し、呼び出し元メソッドに戻ることができます。「デバッグ」ウィンドウは呼び出しチェーンにある以前のメソッドに戻ります。現在のメソッドが呼び出しチェーン内にある最後のメソッドである場合には、「デバッグ」ウィンドウが閉じます。

「実行コントロール」ツールバーについて

「実行コントロール」ツールバーの右側には、デバッガによる次のような情報が表示されます。

- 現在トレースしているメソッドの名前(黒で表示されます)
- 「デバッグ} ウィンドウが表示される原因となった問題(赤で表示されます)

ここで示されているウィンドウの例を使用すると、次のような情報が表示されます。

- 現在トレースされているメソッドは「DebugDemo」メソッドです。
- 「デバッグ」ウィンドウが表示されているのは、**C_DATE** コマンドへの呼び出しが検出され、このコマンドは検出の対象となったコマンドのうちの1つであるためです。

デバッガとメッセージが表示される理由の一部は、次のとおりです(赤で表示されます)。

■ **TRACEコマンド：TRACE**コマンドへの呼び出しが実行されたため。

■ **ブレークポイントに到達：**一時的ブレークポイントまたは永続的ブレークポイントが発見されたため。

■ **ユーザ割り込み：「**Alt -クリック(Windows)」または「option -クリック(Macintosh)」を使用するか、あるいは「デザイン」モードの「プロセス」メニューから「トレース」メニューコマンドを使用したため。

■ **次のコールを捕まえました：** 検出の対象となった4Dコマンドへの呼び出しが実行位置にあるため。

■ **新規プロセスへステップしています：**「ステップ（新規プロセスもトレース)」ボタンを使用したので、新しく作成されたプロセス用の「デバッグ｝ウィンドウがこのメッセージを表示したため。

「デバッグ」ウィンドウの各エリア

「デバッグ」ウィンドウは、すでに説明している「実行コントロール」ツールバーとサイズ変更可能な次の4つのエリアから成ります。

■ **「デフォルト表現式／値」エリア**

■ **「メソッド連鎖」エリア**

■ **「カスタム表現式／値」エリア**

■ **「ソースコード」エリア**

最初の3つのエリアでは、操作が簡単な階層型リストを使用して、関連するデバッグ情報を表示します。4番目の「ソースコード」エリアは、トレースされているメソッドのソースコードを表示します。それぞれのエリアには、デバッグ作業を補助する独自の機能があります。マウスを使用すれば、「デバッグ」ウィンドウの他に、各エリアも垂直および水平方向にサイズを変更することができます。さらに、最初の3つのエリアには、表示する2つのカラムの間に点線による区切り線が含まれています。マウスを使用すれば、この点線を移動して、必要に応じて水平方向にカラムのサイズを変更することができます。

「デフォルト表現式／値」エリア

「デフォルト表現式／値」エリアは、「デバッグ」ウィンドウの左上隅の「実行コントロール」ツールバーの下に表示されます。

「デフォルト表現式／値」エリアには、システム、4D環境、および実行環境について役立つ一般情報が表示されます。

「表現式」欄には、オブジェクトや表現式の名前が表示されます。「値」欄には、オブジェクトや表現式に対応する現在の値が表示されます。

エリア右側にある値をクリックすると、そのオブジェクトの値を変更できる場合には、オブジェクトの値を修正できます。

複数レベルを対象とする階層リストは、メインレベルでテーマごとにまとめられています。テーマは、次のとおりです：

- ラインオブジェクト
- 変数
- 定数
- フィールド
- セマフォ
- セット
- プロセス
- 命名セレクション
- 情報

テーマによっては、各項目に1つまたは複数のサブレベルがある場合もあります。テーマ名の隣にあるリストノード（アイコン）をクリックすると、テーマが拡大したり縮小します。テーマが拡大されている時には、そのテーマにある項目は見えています。テーマに複数レベルの情報がある場合には、各項目の隣にあるリストノードをクリックすると、そのテーマで提供されているすべての情報を検索することができます。

どの時点でも、テーマ、テーマサブリスト(あれば)、テーマ項目を「カスタム表現式／値」エリアにドラッグアンドドロップすることができます。

情報：現在のデフォルトテーブル(あれば)などのような一般的な情報を表示します。このテーマからの表現式を修正することはできません。

命名セレクション：現在の(現在トレースの対象となっている)プロセスで定義されているプロセス命名セレクションを一覧表示します。インタープロセス命名セレクションも一覧表示します。各命名セレクションでは、「値」欄にレコード数およびテーブル名が表示されます。命名セレクションを使用していない場合には、このリストは空白の場合があります。このテーマからの表現式を修正することはできません。

プロセス：作業セッションを開始してから起動されたプロセスを一覧表示します。「値」欄に、それぞれのプロセスの現在の状態(実行中、一時停止など)が表示されます。このテーマからの表現式を修正することはできません。

セット：現在の(現在トレースの対象となっている)プロセスで定義されているセットを一覧表示します。インタープロセスセットも一覧表示します。各セットでは、「値」欄にレコードの数とテーブル名が表示されます。セットを使用していない場合には、このリストは空白の場合があります。このテーマからの表現式を修正することはできません。

セマフォ：現在設定されているローカルセマフォおよびグローバルセマフォを一覧表示します。各セマフォでは、「値」欄にセマフォを設定するプロセスの名前が表示されます。セマフォを使用していない場合には、このリストは空白の場合があります。

フィールド：データベース内にあるテーブルやフィールドを一覧表示しますが、サブフィールドは表示しません。各テーブル項目では、「値」欄に現在のプロセスのカレントセレクションのサイズが表示されます。各フィールド項目では、カレントレコードがある場合には、「値」欄にカレントレコードのフィールドの値(ピクチャ、サブテーブル、BLOBは除く)が表示されます。このテーマでは、フィールドの値を修正することはできますが(ただし、やり直しはできません)、テーブル情報を修正することはできません。

定数：「エクスプローラ」ウィンドウの「定数」ページのように、4th Dimensionが提供する定義済み定数を表示します。このテーマからの表現式を修正することはできません。

変数：このテーマは、次のサブテーマから構成されます。

■ **インタープロセス変数：**この時点で使用されているインタープロセス変数のリストを表示します。インタープロセス変数を使用していない場合には、このリストは空白の場合があります。インタープロセス変数の値は、変更することができます。

■ **プロセス変数：**現在のプロセスで使用されている変数のリストを表示します。このリストが空白であることはほとんどありません。プロセス変数の値は、修正することができます。

■ **ローカル変数：**現在トレースの対象となっているメソッド(ソースコードペインに表示されているメソッド)で使用されているローカル変数のリストを表示します。ローカル変数を使用していない場合や、ローカル変数がまだ作成されていない場合には、このリストは空白の場合があります。ローカル変数の値は、修正することができます。

■ **パラメータ（引数）：**メソッドで受信したパラメータのリストを表示します。現在トレースの対象となっているメソッド(ソースコードペインに表示されているメソッド)にパラメータが渡されていない場合には、このリストは空白の場合があります。パラメータの値は、修正することができます。

■ **セルフポインタ：**オブジェクトメソッドをトレースしている場合には、現在のオブジェクトへのポインタを表示します。この値を修正することはできません。

注：文字列変数、テキスト変数、数値変数、日付変数、および時間変数は、修正することができます。つまり、キーボードを使用して値を入力できる変数は、修正することができます。

他の変数と同様に、配列は、インタープロセス配列サブテーマ、プロセス配列サブテーマ、およびローカル配列サブテーマで、それぞれの範囲によって表示されます。デバッガは各配列に階層レベルをつけて表示します。このため、配列要素の値がある場合にはこれを取得したり変更することができます。デバッガは要素ゼロを含む最初の100個の要素を表示します。「値」欄には、配列名ごとのサイズが表示されます。配列を作成した後、最初のサブ項目には、現在選択されている要素番号、次に要素ゼロ、その次に他の要素(100個まで)が表示されます。文字列配列、テキスト配列、数値配列、および日付配列は、修正することができます。選択された要素番号、要素ゼロ、および他の要素(100個まで)も、修正することができます。配列のサイズを修正することはできません。

注：個別の配列要素も含め、項目はいつでも「デフォルト表現式／値」エリアから「カスタム表現式／値」エリアへドラッグアンドドロップすることができます。

ラインオブジェクト

このテーマには、次のようなオブジェクトや式の値が表示されます。

■ 実行されるコードの行(プログラムカウンタにより、「ソースイベント」エリア内で黄色の矢印でマークされている行)で使用されているか、または

■ コードの以前の行で使用されている

コードの以前の行とは実行された直後の行であるため、「ラインオブジェクト」テーマでは、その行が実行される前または後の現在の行のオブジェクトや式が表示されます。例えば、次のメソッドを実行した場合を想定します：

```
TRACE
a:=1
b:=a+1
c:=a+b
  `…
```

1.「ソースコード」エリアのある「デバッグ｝ウィンドウに、プログラムカウンタを「a:=1」にセットして入力します。この時点では「ラインオブジェクト」テーマには、次のように表示されています：

a：　未定義

変数aが表示されているのは、実行の対象となっている(ただし、まだ初期化されていない)行で使用されているためです。

2. 1行先に移動してみると、プログラムカウンタは行「b:=a+1」に設定されています。この時点では「ラインオブジェクト｝テーマには、次のように表示されています。

a：　1
b：　未定義

変数aが表示されているのは、ちょうど実行され、数値1を割り当てられたばかりの行で使用されているためです。また、変数aが表示されているのは、変数bに割り当てられる式として実行される対象となっている行でも使用されているためでもあります。変数bが表示されているのは、実行の対象となっている(ただし、まだ初期化されていない)行で使用されているためです。

3. 再び1行先に移動してみると、プログラムカウンタは行「c:=a+b」に設定されています。この時点では「ラインオブジェクト｝テーマには、次のように表示されています。

c：　未定義
a：　1
b：　2

変数cが表示されているのは、実行の対象となっている(ただし、まだ初期化されていない)行で使用されているためです。変数aとbが表示されているのは、1つ前の行で使用され、その行で実行されているからです。

「ラインオブジェクト」テーマは、とても便利なツールです。それは、ある行が実行される度に「カスタム表現式／値」エリアに式を入力することなく、「ラインオブジェクト」テーマによって表示される値を見ることができます。

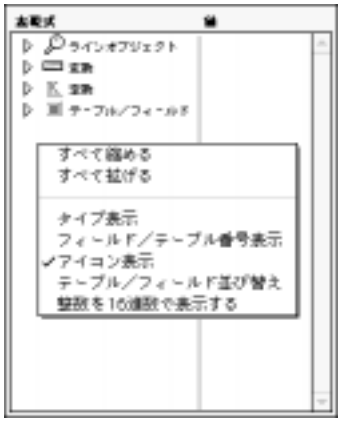
「スピード」メニュー

「デフォルト表現式／値」エリアの「スピード」メニューで追加オプションを提供します：。

■ Windowsでは、マウスの右ボタンを使用して「デフォルト表現式／値」エリア内の任意の位置をクリックする。

■ Macintoshでは、「デフォルト表現式／値」エリアの任意の位置で「Control -クリック」を実行する。

「デフォルト表現式／値」エリアの「スピード」メニューが、次のように表示されます。



■**「すべて縮める」：**「デフォルト表現式／値」エリアの階層リストのすべてのレベルを縮小します。

■**「すべて拡げる」：**「デフォルト表現式／値」エリアの階層リストのすべてのレベルを拡張します。

■**「タイプ表示」：**それぞれのオブジェクトのオブジェクトタイプを(適切な場合に)表示します。

■**「フィールド／テーブル番号表示」：**「フィールド」のそれぞれのテーブルまたはフィールドの番号を表示します。テーブル番号やフィールド番号を操作している場合、または、**Table**関数や**Field**関数を使用してポインタを操作している場合にはこのオプションは非常に便利です。

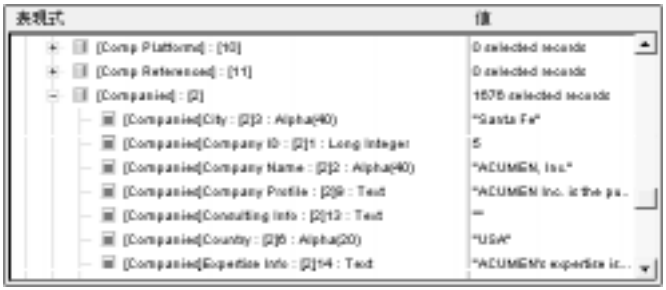
■**「アイコン表示」：**それぞれのオブジェクトのオブジェクトタイプを示すアイコンを表示します。表示速度を速くするために、あるいは「アイコン表示} オプションを使用することにして、このオプションをオフにすることもできます。

■**「テーブル／フィールド並び替え」：**テーブルおよびフィールドをそれぞれ対応するリストの中で強制的にアルファベット順に表示します。

■**「整数を16進数で表示する」：**数字は通常10進法で表示されます。このオプションを使用すると、数字が16進法表記で表示されます。

注：数値を16進法で入力するには、0x(ゼロの後にx)とタイプし、その後に16進数を続けます。

次の図は「デフォルト表現式／値」エリアのオブジェクトを選択した状態を示しています。



「メソッド連鎖」エリア

1つのメソッドによって他のメソッドも呼び出される場合があります。さらにそれらがその他のメソッドを呼び出す場合もあります。このため、デバッグ処理中には、メソッドの連鎖つまり呼び出しチェーンを表示しておくと非常に便利です。「メソッド連鎖」エリアはこのの便利な機能を提供していますが、「デバッグ」ウィンドウの上部右側にあります。このエリアは、階層リストを使用して表示されます。次の図は、「メソッド連鎖」エリアの例を示しています。

■ それぞれのメインレベルの項目は、メソッドの名前です。最も上にある項目は、現在トレースされているメソッド、次のメインレベルの項目は呼び出し元のメソッド(現在トレースしているメソッドを呼び出したメソッド)、その次の項目は呼び出し元のメソッドの呼び出しメソッド、などのように続きます。この例では、メソッド「M_BitTestDemo」がトレースされています。このメソッドは、メソッド「DE_LInitialize」によって呼び出され、これは「DE_DebugDemo」によって呼び出されています。

■「メソッド連鎖」エリアにあるメソッドの名前をダブルクリックすると、呼び出し元のメソッドに戻り、そのソースコードが「ソースコード」エリアに表示されます。このようにすると、呼び出し元のメソッドが呼び出されたメソッドへの呼び出しをどのように実行したか、すばやく確認することができます。このようにして、呼び出しチェーンをどの段階でも検証することができます。

■ メソッド名の隣にあるノードアイコンをクリックすると、メソッドのパラメータ($1, $2...)およびオプションの関数結果($0)のリストが拡張されたり縮小されます。値はエリアの右側に表示されます。

右側の矢印にある値をクリックすると、パラメータや関数の結果の値を変更することができます。この図では、次のとおりです。

1.「M_BitTestDemo」メソッドは、まだどのパラメータも受け取っていません。

2.「M_BitTestDemo」メソッドの $0は現在未定義です。メソッドが値を $0に割り当てていないためです(メソッドがこの割り当てをまだ実行していないか、メソッドがサブルーチンであり、関数ではないことが原因です)。

3.「DE_LInitialize」メソッドは、「DE_DebugDemo」メソッドから3つのパラメータを受け取りました。$1はテーブル[Customers]へのポインタ、$2はフィールド[Customers]Companyへのポインタ、$3は値が"Z"の英数字パラメータです。

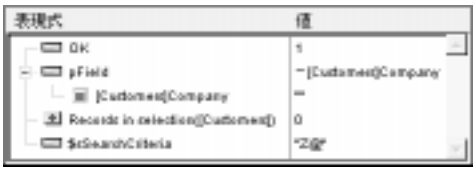
■ メソッドにパラメータリストを割り当てれば、パラメータや関数の結果を「カスタム表現式／値」エリアにドラッグアンドドロップすることができます。

「カスタム表現式／値」エリア

「メソッド連鎖」エリアのすぐ下にあるのは、「カスタム表現式／値」エリアです。このエリアは、式を評価するために使用します。フィールド、変数、ポインタ、演算、内部関数、カスタム定義関数など、値を戻すものなら何でも、どのようなタイプの式でも評価できます。

テキスト形式で表示できる式であれば、どのような式でも評価することができます。ピクチャやBLOBのフィールドや変数は、対象になりません。一方、デバッガは、割り当てられた階層リストを使用して、配列やポインタを表示できるようにします。**BLOB to text**などのようなBLOBコマンドを使用して、BLOBの内容を表示することができます。

次の例では、2つの変数、1つのフィールドポインタ変数、内部関数の結果、演算の項目のうちいくつかが示されています。



新しい式の挿入

次のように「カスタム表現式／値」エリアに式を追加して、評価することができます。

■「デフォルト表現式／値」エリアからオブジェクトまたは式をドラッグアンドドロップします。

■「メソッド連鎖」エリアからオブジェクトまたは式をドラッグアンドドロップします。

■「ソースコード」エリアで、評価できる式をクリックします。

空の式を作成するには、「デフォルト表現式／値」エリアの空白スペースの任意の位置をダブルクリックします。このようにすれば、新しい式 “ `新規表現式 ” を追加し、編集モードに移行してこれを編集できます。結果を戻す4th Dimensionのフォーミュラを入力できます。

フォーミュラを入力した後、EnterまたはReturnキーを押して(またはエリアの任意の位置をクリックして)、式を評価します。

式を変更するには、その式をクリックして選択し、再びクリックして(またはEnter数値キーパッドを押して)編集モードに移行します。

式を必要としなくなった場合には、その式をクリックして選択し、Backspaceキーまたは Deleteキーを押します。

「カスタム表現式／値」エリアの「スピード」メニュー

式を入力し編集する場合には、「カスタム表現式／値」エリアの「スピード」メニューを使用すると、4th Dimensionの「フォーミュラ」エディタにアクセスできて便利です。実際には、この「スピード」メニューには、他にもオプションがあります。

このメニューを表示するには、次のように実行します。

■ Windowsでは、マウスの右ボタンを使用して、「カスタム表現式／値」エリアの任意の位置をクリックする。

■ Macintoshでは、「カスタム表現式／値」エリアの任意の位置で「Control - クリック」を実行する。

■**「新規表現式...」**：新しい式を入力し、(次の図のような)4th Dimensionの「フォーミュラ」エディタを表示して、新しい式を編集できるようにします。

「フォーミュラ」エディタに関する詳細は、『4th Dimension / 4D First ユーザリファレンス』を参照してください。

■**「コマンド挿入」**：この階層メニュー項目は、コマンドを新しい式として「フォーミュラ」エディタを使用せずに挿入するためのショートカットです。

■**「すべて削除」**：現在あるすべての式を削除します。

■**「すべて縮める／すべて拡げる」**：階層型リストを使用して評価が実行されたすべての式(ポインタ、配列など)を縮小したり拡大します。

■**「タイプ表示」**：(適切な場合に)各オブジェクトのオブジェクトタイプを表示します。

■**「フィールド、テーブル番号表示」**：「フィールド」にあるそれぞれのテーブルやフィールドの番号を表示します。**Table**関数や**Field**関数を使用してポインタを操作している場合には、このオプションは非常に便利です。

■**「アイコン表示」**：それぞれのオブジェクトのオブジェクトタイプを示すアイコンを表示します。表示速度を速くするために、あるいは「アイコン表示」オプションを使用することにして、このオプションをオフにすることもできます。

■**「テーブル／フィールド並べ替え」**：テーブルおよびフィールドをそれぞれ対応するリストの中で強制的にアルファベット順に表示します。

■**「整数を16進数で表示する」**：数字は通常10進法で表示されます。このオプションを使用すると、数字が16進法表記で表示されます。

注：数値を16進法で入力するには、0x(ゼロの後にx)とタイプし、その後に16進数を続けます。

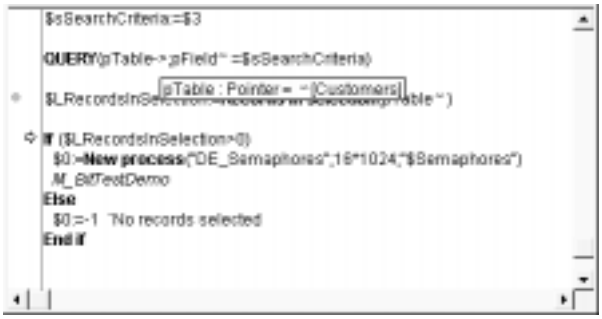
「ソースコード」エリア

「ソースコード」エリアには、トレースされているメソッドのソース･コードが表示されます。

■ メソッドが長すぎてテキストエリアに収まらない場合には、スクロールするとメソッドの他の部分も表示できます。

■ 評価できる式(フィールド、変数、ポインタ、配列など)にマウスポインタを移動すると、「ツールチップ」には、オブジェクトまたは表現式の現在の値とその宣言型が表示されます。

次の図は「ソースコード」エリアを示しています。



「ツールチップ」が表示されているのは、マウスポインタが「ツールチップ」によると[Customers]テーブルへのポインタである変数「pTable」の上に移動されたためです。

Tips：「ソースコード」エリアで(評価できる)式をクリックすると、式またはオブジェクトが「カスタム表現式／値／エリアにコピーされます。

プログラムカウンタ

「ソースコード」エリアの左マージンにある黄色の矢印(上図を参照してください)は、実行される次の行を表しています。この矢印は、“ プログラムカウンタ ” と呼ばれます。プログラムカウンタは、常に実行寸前の行を表示します。

デバッグのために、メソッドのプログラムカウンタの位置を呼び出しチェーン(実際に実行されるメソッド)の先頭に変更することができます。そのためには、黄色の矢印を目的の行まで上下にドラッグアンドドロップします。

警告：この機能は、十分注意して使用してください。

プログラムカウンタを順方向に移動する時には、スキップした行をデバッガがすばやく実行しているわけではありません。

同様に、プログラムカウンタを逆方向に移動する時には、すでに実行された行の結果をデバッガが逆方向に実行しているわけではありません。

プログラムカウンタを移動するということは、デバッガにその位置からのトレースや実行を追跡するように指示するということに過ぎません。すべての現在の設定内容、フィールド、変数などには、プログラムカウンタの移動による影響はありません。

プログラムカウンタを移動する例は、次のとおりです。例えば、次のようなコードをデバッグすると仮定してみましょう。

```
`...
If (条件)
   DO SOMETHING
Else
   DO SOMETHING ELSE
End if
```

プログラムカウンタは、行「**If (条件)**」に設定されています。1ステップ先に進むと、プログラムカウンタが行「**DO SOMETHING ELSE**」に移動していることがわかります。別の分岐にある行を実行しようとしていたため、これは思いがけないことです。この場合には、“ 条件 ” という式が次のステップに影響を与えるような演算を実行していなければ、プログラムカウンタを行「**DO SOMETHING**」に戻します。これで、コードの当初目的としていた部分を続けてトレースすることができます。

ブレークポイントの設定

デバッグ処理では、コードの一部のトレースをスキップする必要がある場合がありますデバッガでは、特定の位置までコードを実行する方法をいくつか提供しています。

■ ステップごとの処理中に、「ステップ（呼び出しメソッドもトレース」ボタンではなく「ステップ（同一メソッドのみ）」ボタンをクリックすることができます。プログラムカウンタ行で呼び出されている可能性のあるサブルーチンや関数の実行を避けたい場合には、この方法が役立ちます。

■ 間違ってサブルーチンの処理を始めてしまった場合には、「ステップ（呼び出し元へ戻る）」ボタンをクリックすると、そのサブルーチンを実行した後直接、呼び出し元のメソッドに戻ることができます。

■ **TRACE**コマンド呼び出しをある位置で指定した場合には、「トレースなし」ボタンをクリックすると、その**TRACE**コマンド呼び出しまでの実行を再開始することができます。

次に、プログラムカウンタを行「**ALL RECORDS([This Table])**」に設定し、次のようなコードを実行していると仮定してみましょう。

```
`...
ALL RECORDS ([This Table])
$vrResult:=0
For ($vlRecord ; 1 ; Records in selection ([This Table])
   $vrResult:= this Function ([This Table])
   NEXT RECORD ([This Table])
End for
If ($vrResult >= $vrLimitValue)
```

ここでの目的は、**For**ループが終了した後で「$vrResult」の値を評価することです。コードでこの位置まで実行されるにはかなりの実行時間がかかるため、現在の実行をアボートしたくない状況である場合には、**TRACE**コマンド呼び出しを行「**If($vrResult..**」.の前に挿入するようにメソッドを編集する必要があります。

ループのステップ処理を実行することも1つの方法ですが、**「This Table」**テーブルに何百件ものレコードが入っている場合には、この処理に一日費やすことになります。このような状況では、デバッガのブレークポイントを使用できます。ブレークポイントは、「ソースコード」エリアの左マージンをクリックすると挿入できます。

例えば、次の例では、行「**If($vrResult...**」のレベルで「ソースコード」エリアの左マージンをクリックします。

このようにすると、その行にブレークポイントが挿入されます。ブレークポイントは赤色の点で表されます。次に、「トレースなし」ボタンをクリックします。

このようにすると、ブレークポイントで示された行まで、通常の実行が再開始されます。ブレークポイントで示された行それ自体は実行されず、トレースモードに戻ります。この例では、ループ全体は連続して正常に実行されてきています。そのため、ブレークポイントになった時には、マウスボタンを「$vrResult」の上に移動して、その値をループの終了地点で評価する必要があります。

プログラムカウンタの位置を越えてブレークポイントを設定し、「トレースなし」ボタンをクリックすると、トレースされた部分のメソッドをスキップすることができます。

赤色のブレークポイントは、永続的ブレークポイントです。このブレークポイントは、一度作成すると残ります。データベースを終了し、後で再び開くと、ブレークポイントはそのまま残っています。

永続的ブレークポイントを消去する方法には、次の2つがあります。

■ 永続的ブレークポイントを使用した後、赤色の点をクリックすると、ブレークポイントは消えます。

■ 永続的ブレークポイントをまだ使用する場合には、保存する必要がある場合もあります。永続的ブレークポイントを編集すると、一時的に使用不可能にすることができます。編集方法については、「ブレークポイント」の節で説明しています。

ブレークポイント

「ソースコード」エリアの節で説明しているように、ブレークポイントは、コードのブレークしたい行と同じレベルで、「ソースコード」エリアの左マージンをクリックして設定します。次の図では、ブレークポイントは「**If($vrResult>=$vrLimitValue)**」に設定されています。

赤色の点を再びクリックすると、ブレークポイントは削除されます。

ブレークポイントの編集

「ソースコード」エリアの左マージンで「Alt - クリック(Windows)」または「option - クリック」(Macintosh)」を実行すると、コードの行単位で「ブレークポイントプロパティ」ウィンドウにアクセスできます。

■ 既存のブレークポイントをクリックすると、そのブレークポイントについてのウィンドウが表示されます。

■ ブレークポイントが設定されていない行をクリックすると、デバッガはブレークポイントを作成し、この新しく作成されたブレークポイントに関するウィンドウを表示します。

「ブレークポイントプロパティ」ウィンドウは、次の図のとおりです。

プロパティは、次のとおりです。

「場所」：メソッドの名前とブレークポイントが設定されている行番号を示します。この情報を変更することはできません。

{タイプ」：デフォルトでは、デバッガは、永続的ブレークポイントを作成することができます。永続的ブレークポイントは、「デバッグ」ウィンドウの「ソースコード」エリアで赤色の点で示されます。一時的ブレークポイントを作成するには、「一時的」オプションを選択します。一時的ブレークポイントは、メソッド中で1度だけブレークしたい場合に役立ちます。一時的ブレークポイントは、「デバッグ」ウィンドウの「ソースコード」エリアで緑色の点で示されます。

注：「ALT+Shift (Windows)」キーまたは「Option+Shift (Macintosh)」キーを押している間に左マージンをクリックして、「ソースコード」エリアに一時的ブレークポイントを直接設定することもできます。

「次に当てはまる場合にブレークする」：TrueまたはFalseを戻す4th Dimensionフォーミュラを入力することによって、条件付きブレークポイントを作成することができます。例えば、、「**Records in selection([aTable])=0**」の条件を満たす場合に限って行でブレークさせる場合には、このフォーミュラを入力すると、デバッガがこのブレークポイントを検出した時に、テーブル[aTable]のレコードが選択されていない場合に限ってブレークが発生します。フォーミュラのシンタックスについて確認できていない場合には、「シンタックス検査」ボタンをクリックします。

「ブレーク前のスキップ回数」：ループ構造(While、Repeat、またはFor)またはループから呼び出されているサブルーチンや関数にあるコードの行にブレークポイントを設定することができます。例えば、、現在調査している問題は、少なくともループを200回繰り返すまでは発生しないことがわかっていると想定します。このような場合には200と入力すると、ブレークポイントは201回目の繰り返しからアクティブになります。

「ブレークポイントを表示しない」：永続的ブレークポイントが現在は必要でないものの、後で必要になるかもしれない場合には、ブレークポイントを編集して一時的に使用不可能にしておくことができます。使用不可能なブレークポイントは、「デバッグ」ウィンドウの「ソースコード」エリアおよび「ブレークリスト」ウィンドウにおいて、点(•)ではなくダッシュ記号(-)で表示されます。

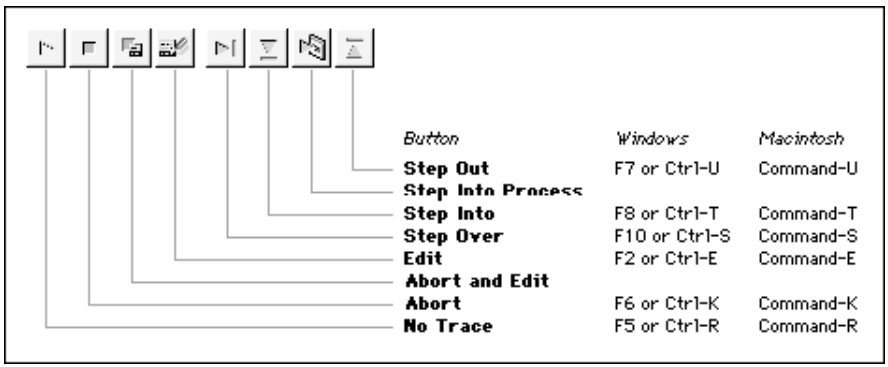
「デバッグ」ウィンドウ内でブレークポイントを作成したり編集することができます。さらに「デザイン」モードの「ブレークリスト」ウィンドウを使用して、既存のブレークポイントを編集することもできます。詳細については、「ブレークリストウィンドウ」の節を参照してください。

デバッガのショートカット

この節では、「デバッグ」ウィンドウで提供されているすべてのショートカットをリストしています。

「実行コントロール」ツールバーでのボタンショートカット

■ 次の図では、「デバッグ」ウィンドウの左上隅にある8個のボタンのショートカットを示しています。



■「ALT+F5 (Windows)」および「Option - Command - R (Macintosh)」は、実行を再開始します。さらに、現在のプロセスの次の**TRACE**コマンド呼び出しをすべて使用不可能にします。

「デフォルト表現式／値」エリアでのキーボードショートカット

■「デフォルト表現式／値」エリアで「マウスの右ボタンをクリック(Windows)」するか、または「Control - クリック(Macintosh)」を実行すると、「デフォルト表現式／値」エリアの「スピード」メニューがプルダウンされます。

■ 「デフォルト表現式／値」エリア内の項目をダブルクリックすると、その項目が「カスタム表現式／値」エリアにコピーされます。

「メソッド連鎖」エリアでのキーボードショートカット

■ 「メソッド連鎖」エリアでメソッドの名前をダブルクリックすると、メソッドが、「ソースコード」エリアの呼び出しチェーンにある呼び出しに対応する行に表示されます。

「カスタム表現式／値」エリアでのキーボードショートカット

■ 「カスタム表現式／値」エリアで「マウスの右ボタンをクリック(Windows)」するか、または「Control - クリック(Macintosh)」を実行すると、「カスタム表現式／値」エリアの「スピード」メニューがプルダウンされます。

■ 「カスタム表現式／値」エリア内でダブルクリックすると、新しい表現式が作成されます。

「ソースコード」エリアでのキーボードショートカット

■ 左マージンをクリックすると、ブレークポイントが設定されたり(永続的ブレークポイントの場合)、ブレークポイントが削除されます。

■「ALT - Shift -クリック(Windows)」または「option - shift クリック(Macintosh)」により、テンポラリブレークポイントが設定されます。

■「Alt -クリック(Windows)」または「option -クリック(Macintosh)」により、新しいブレークポイントや既存のブレークポイントの「ブレーク編集」ウィンドウが表示されます。

■「ソースコード」エリアで評価可能な式をクリックすると、式またはオブジェクトが「カスタム表現式／値」エリアにコピーされます。

全エリア共通のキーボードショートカット

■ どのペインでも項目が選択されていない場合にEnterキーを押すと、1行ずつ進みます。

■ 項目の値が選択されている場合には、矢印キーを使用してリスト内を移動します。

■ 項目が編集されている場合には、矢印キーを使用してカーソルを移動します。「Ctrl -A/X/C/V (Windows)」、または「Command -A/X/C/V (Macintosh)」を「編集」メニューの「すべてを選択／切り取り／コピー／貼り付け」メニューコマンドへのショートカットとして使用します。

表示されないプロセスやコードを実行していないプロセスのトレース（4th Dimensionのみ）

デバッガの「ステップ（新規プロセスもトレース）」ボタンを使用すると、**New Process** 関数を使用してプロセスを開始した時点から、そのプロセスをトレースすることができます。

また、プロセスが開始されてからかなり経過した後でプロセスをトレースしたくなる場合もあります。

プロセスで、表示されているウィンドウが少なくとも1個あり、そのウィンドウが最も手前に表示されている場合に、そのウィンドウ内で「Alt -クリック(Windows)」または「option -クリック(Macintosh)」を押すと、そのプロセスのトレースモードが開始されます。

「Alt - クリック」または「option - クリック」は、プロセスが次のような状態の時には、使用することが困難な場合もあります。

■ プロセスがコードを実行しているが、そのプロセスのウィンドウが他のウィンドウの後ろにあるため、それらを移動してプロセスにアクセスしたい場合

■ プロセスがコードを実行しているが、ユーザインタフェースがない(ウィンドウがない)場合

■ プロセスがデータ入力の状態にあり、イベントを待っている場合(*)

■ プロセスが現在一時停止されている場合(*)

■ プロセスが現在遅延されている場合(*)

(*)は、プロセスが実行されているもののコードを実行していないことを示しています。

4Dにはプロセスのトレースを開始する別の便利な方法もあります。この技法では、プロセスが表示されている必要もなく、プロセスがコードを実行している必要もありません。

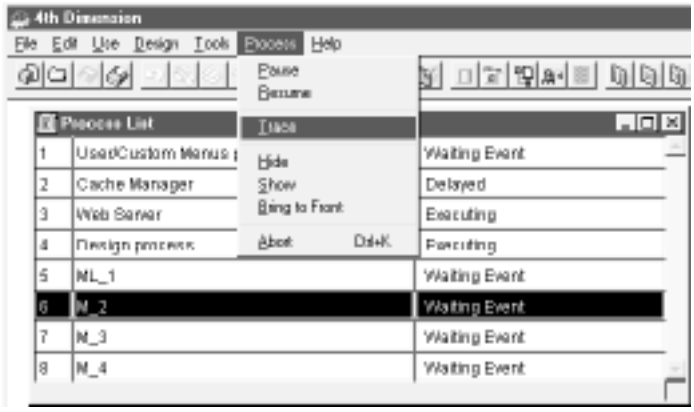
1. まだ「デザイン」モードになっていない場合には、「デザイン」モードに切り替えます。

2.「ツール」メニューから「プロセスリスト」を選択します。

3.「プロセスリスト」ウィンドウが表示されます。「プロセスリスト」ウィンドウには、現在実行されているプロセスが(実際にコードを実行しているかどうかに関わらず)表示されます。

4. トレースしたいプロセスをクリックして選択します。

5.「プロセス」メニューから「トレース」を選択します。



ここで注目するべき点は、4Dはトレースの要求を「記憶する」ということです。

■ プロセスが現在コードを実行している場合には、そのプロセスのデバッガがただちに表示されます。

■ プロセスが現在コードを実行していない(たとえば、プロセスが、データ入力モードでイベントを待っている)場合には、デバッガは、プロセスがコードの実行を再開始した直後に表示されます。

Tips：ボタンをクリックした時に、そのボタンのオブジェクトメソッドをトレースするとよいでしょう。「Alt -クリック(Windows)」や「option -クリック(Macintosh)」は、クリックの「速度」によって有効な場合も無効な場合もあります。そのような場合には、「プロセス」メニューから「トレース」メニューコマンドを使用します。オブジェクトメソッドが開始されると、ただちにデバッガが表示されます。それ以外の場合には、メソッドそれ自体にTRACEコマンド呼び出しを設定することもできます。

この章では、配列について説明します。このマニュアルの前の章で説明したものよりも高度な技法について説明します。

配列

配列は、関連するデータのグループをより効率良く記憶、アクセス、管理するために使用します。配列は番号付きの変数の集まりのようなものです。その各項目を配列の“要素”と呼びます。

配列は、変数と同じようにデータタイプを持っています。すべての要素は、同じデータタイプになります。また、一般にサイズは固定ですが、サイズを変えるためのコマンドも用意されています。

配列は、メモリに常駐しますので、コピー、ソート、検索などの処理を高速に行うことができます。ただし、大きな配列を作成する場合には、メモリの容量を上回らないように、随時使用しなければなりません。特に、配列にレコードを代入するコマンドを使用する場合には注意が必要です。レコード数が多いと、大きな配列が作成されて、メモリを浪費してしまうからです。

配列を使用する

後の章で説明する配列コマンドを使って配列を作成します。次のステートメントは、配列“名字”に名字を5つ代入します。

ARRAY TEXT (名字 ; 5)

中括弧({})を使用して配列の要素を参照します。中カッコ内には数値を記述します。次のステートメントは、5つの名字を配列“名字”に格納します。

名字{1}:="鳥居"
名字{2}:="石井"
名字{3}:="高橋"
名字{4}:="吉野"
名字{5}:="森"

SELECTION TO ARRAYコマンドと **LIST TO ARRAY**コマンドは、より効率良くデータを配列に格納するコマンドです。

配列要素の参照は、変数と同じように扱います。例えば、次のステートメントは、変数“ 私の名前 ”に文字列“ 高橋 ”を代入します。

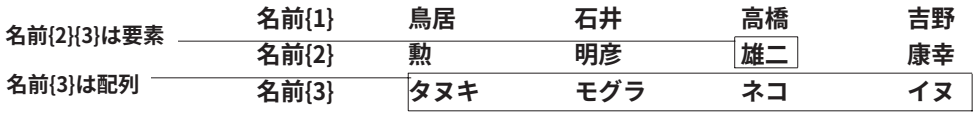
```
私の名前:=名字{3}
```

2次元配列を使用する

2次元配列も配列コマンドで作成します。次のステートメントは、3行4列の配列を作成します。4th Dimensionは、2次元配列の行を分割した配列として取り扱います。

```
ARRAY TEXT (名前 ; 3 ; 4)
```

次の図はデータを格納した配列を示しています。



2つの中括弧を組み合せて2次元配列の要素を参照します。例えば、3行目の2番目の要素を参照するには、次のように記述します。

```
名前{3}{2}
```

上のデータを使用すると、このステートメントは文字列“ モグラ ”を返します。

“ 名前 ”が各要素の行であると理解することが重要です。例えば、“ 名前{3} ”は3行目を参照します。この場合は、ニックネームを示します。

2次元配列の1“ 次元 ”は他のすべての配列と同様に処理できます。例えば、3番目の配列“ 名前 ”を別の配列“ arLast ”にコピーするには、次のコマンドを使用します。

```
COPY ARRAY (名前{3} ; arLast)
```

ローカル配列、プロセス配列、インタープロセス配列

配列には、ローカル配列、プロセス配列、およびインタープロセス配列の 3種類あります。

ローカル配列は任意のメソッド内で使用できます。この配列はメソッドが終了すると消去されます。ローカル変数と同じように、なるべくこの種の配列を使用してください。ローカル配列は、配列名の先頭にドル記号($)を付けて宣言します。同じ名前のローカル

配列が異なるメソッドにおいて、異なるタイプになっていても構いません。ローカル配列は実行サイクルに対しローカルなため、スクロールエリアやドロップダウンリストボックス（Macintosh版では、ポップアップメニュー）としてフォーム内にローカル配列を表示することはできません。

プロセス配列は任意のプロセス内で使用します。この配列はプロセスが終了すると消去されます。複数のプロセスが同じ配列を定義し、プロセスごとに異なった値を入れることができます。ただし、これらの配列はプロセス間で同じ型を持っていなければなりません。

インタープロセス配列はすべてのプロセスで使用可能です。これはプロセス間でデータを共用する場合と情報を交換する場合にのみ使用します。インタープロセス配列は配列名の先頭にインタープロセス記号 “ <> ” を付けます。

次の例は、整数型のプロセス配列を作成します。

ARRAY INTEGER (<>整数配列 ; 20)

配列を表示する

配列は、ドロップダウンリストボックス（ポップアップメニュー）やスクロールエリアを使用しているフォームに表示することができます。どちらの場合においても、ドロップダウンリストボックスやスクロールエリアに表示される配列の名前は、フォーム上のドロップダウンリストボックスとスクロールエリアの名前と同一になっていることを理解することが重要です。さらに、ドロップダウンリストボックスやスクロールエリアの名前は、ドロップダウンリストボックスやスクロールエリア内で選択された配列要素の番号を含む変数であることを理解することも重要です。

注：フォームに2次元配列の1列やローカル配列を表示することはできません。

表示される配列は、次の4つの中のいずれかを使用して作成します：

■ **LIST TO ARRAY**コマンド：このコマンドは、「リスト」エディタで定義されたリストから配列を作成します。

■ **SELECTION TO ARRAY**コマンド：このコマンドは、レコードのセレクションから1つまたは複数の配列を作成します。

■ **DISTINCT VALUES**コマンド：このコマンドは、テーブルから個々の値の配列を作成します。

■ 配列の各要素は、1つのメソッド内で個々に割り当てることができます。

注：実数、整数、倍長整数、日付、ブールタイプの配列は、『4th Dimension / 4D First デザインリファレンス』の中で記述されている表示フォーマットを使って、フォーマットすることができます。

スクロールエリアの例

次の例は、スクロールエリア内での配列の挿入および表示方法を示しています。配列“配列給与”は、**DISTINCT VALUES**コマンドを使用して作成されます。これには、会社の従業員の給与のすべてが含まれています。また、“配列給与”と定義されたスクロールエリアはフォーム上に描画されています。

ユーザがスクロールエリアから配列要素を選択すると、“給与”フィールドに「ユーザ」モードまたは「カスタム」モードで選択された値が割り当てられます。

上記の例を作成するには、まず、ある会社に異なる従業員の名前と給与を含んだデータベースがあることが前提となります。

1. 配列を初期化する。
プロジェクトメソッド内で、**DISTINCT VALUES**コマンドを使用して“配列給与”を初期化します。また、前述の4つの方法のいずれかで配列を初期化することができます。

```
DISTINCT VALUES ([従業員]給与 ; 配列給与)
```

2. フォーム上にスクロールエリア“配列給与”を作成する。
フォーム上にスクロールエリアを作成し、その名前を“配列給与”にする。スクロールエリアの名前は配列の名前とまったく同一です。

3. ユーザがスクロールエリアから選択した配列要素を確認するために、スクロールエリアにオブジェクトメソッドを作成する。
オブジェクトメソッドはスクロールエリアを初期化し、そしてスクロールエリア内で選択された値をフィールドや別の変数に割り当てます。スクロールエリアは変数であり、ユーザによって選択された値を自動的にレコードに保存しないことを理解することが重要です。

```
If (Before | (配列給与=0))
   配列給与:=1
End if
[従業員]給与:=配列給与{配列給与}
```

Beforeフェーズにおいて、現在の要素に配列の最初の要素が代入されます。ユーザがスクロールエリアから配列要素を選択すると、[従業員]給与フィールドに選択された値が代入されます。これはユーザによって選択された値を保存します。

ドロップダウンリストボックス（ポップアップメニュー）の例

次の例は、ドロップダウンリストボックス内での配列の挿入および表示方法を示しています。例では、ユーザはドロップダウンリストボックスから国を選択しています。**LIST TO ARRAY**コマンドは配列 “ 配列国名 ” の初期化に使用されます。

1. 配列を初期化する。
プロジェクトメソッド内で、**LIST TO ARRAY**コマンドを使用して “ 配列国名 ” を初期化します。

```
LIST TO ARRAY ("国名" ; 配列国名)
```

2. フォーム上にドロップダウンリストボックス “ 配列国名 ” を作成する。
フォーム上にポップアップメニューを作成し、その名前を “ 配列国名 ” にする。ドロップダウンリストボックスの名前は配列の名前とまったく同一です。

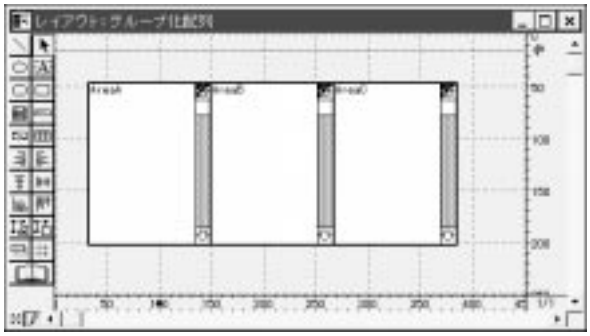
3. ユーザがドロップダウンリストボックスから選択した配列要素を確認するために、ドロップダウンリストボックス**にオブジェクトメソッドを作成する。**
オブジェクトメソッドはドロップダウンリストボックスを初期化し、そしてドロップダウンリストボックス内で選択された値をフィールドや別の変数に割り当てます。ドロップダウンリストボックスは変数であり、ユーザによって選択された値を自動的にレコードに保存しないことを理解することが重要です。

```
If (Before | (配列国名=0))
   配列国名:=1
End if
[従業員]国:=配列国名{配列国名}
```

Beforeフェーズにおいて、現在の要素に配列の最初の要素が代入されます。ユーザがドロップダウンリストボックスから配列要素を選択すると、[従業員]国フィールドに選択された値が代入されます。これはユーザによって選択された値を保存します。

グループ化したスクロールエリアを使用する

フォーム上のスクロールエリアをグループ化することができます。スクロールエリアをグループ化すると、それらはまるで1つのスクロールエリアであるかのように動作します。また、スクロールエリアごとに、フォントやスタイルを設定することができます。データ入力時に、最前面にあるスクロールエリアのみがスクロールバーを表示します。下図は「デザイン」モード内のグループ化された3つのスクロールエリアのフォームです。



下記に、グループ化したスクロールエリアを作成する際の注意点を記します：

■ すべての配列が同じサイズ（同じ要素数）になっているか確認する。

■ 各スクロールエリアに対して、すべて同じ文字サイズ(ポイント数)を使用する。

■ 各スクロールエリアを同じ高さにする。

■ 各スクロールエリアの1番上と1番下を揃える。

■ 隣接するスクロールエリアが互いに重ならないようにする。

■ 1番右側のスクロールエリアを前面に移動する。実行時には、最前面のスクロールエリアのスクロールバーだけを操作可能にする。

■ 各スクロールエリアに対する単独の操作を終了してから、すべてのエリアを選択し、「設定」メニューから「グループ」を選択する。

■ グループの周りにグループよりも1ピクセル大きい境界線を引く。(グループを選択し、“ Ctrl（Macintosh版では、コマンド）” キーを押したまま “ 1 ” を押す。） 右端の境界線を左に1ピクセル移動（ショートカットでは、“ Ctrl (コマンド) - 左矢印キー）すると、見やすくなる。

次のステートメントは、グループ化されたスクロールエリアで示した3つの配列にデータを格納します。これは、[従業員]テーブルのフィールドのデータと[部門]テーブルのフィールドのデータを使用します。[部門]テーブルは、[従業員]テーブルからリレートされているため使用することができます。

SELECTION TO ARRAY ([従業員]名字 ; エリアA ; [従業員]役職 ; エリアB;
[部門]名称;エリアC)

次の図は、先ほどの実行結果の画面を示しています。

1つのスクロールバーしか表示されない点に注目してください。これは、最前面にあるスクロールエリアのスクロールバーだけが表示されるからです。3つの配列がまるで1つであるかのようにスクロールします。

ユーザが任意行をクリックすると、3つのエリアが同時に反転表示されます。また、各スクロールエリアに付随する変数に、ユーザがクリックした行の番号が代入され、クリックされたエリアのオブジェクトメソッドだけが実行されます。例えば、ユーザが“小田”という名前をクリックすると、配列に付随する変数“エリアA”、“エリアB”、“エリアC”のすべてに3が代入されます。しかし、“エリアA”に対するオブジェクトメソッドだけが実行されます。

1つの配列に付随する変数に値が代入されると、他の配列に付随する変数も自動的に同じ値がセットされ、スクロールエリアの該当する行が反転表示されます。

配列は、次のステートメントでソートすることができます：

SORT ARRAY (エリアA ; エリアB ; エリアC ; >)

次の図は、ソートの結果を示しています。

配列が**SORT　ARRAY**コマンドの最初の引数を基準にソートされる点に注目してください。他の2つの配列はその行との同調を維持するために設定されます。

ポインタ

ポインタの使用は、データを参照するための上級技法です。第1章から第12章までで示した概念を完全に理解した上でポインタを使用してください。

プログラミング言語を使用する場合、名前を用いてテーブル、フィールド、変数、配列などをアクセスします。通常は、名前によるデータの参照が最も簡単な方法ですが、名前を使用しないでデータを参照したり、アクセスした方が便利な場合もあります。この場合、ポインタを使用すると実現することができます。

ポインタの背景にある概念は、日常生活でもよく用いられています。対象物を正確に知らないで何かを示すことがあります。例えば、友人に対して“ 登録番号123ABDの車に乗ろう ”と言わないで“ 君の車に乗ろう ”という場合があります。つまり、“ 登録番号123ABDの車 ”を“ 君の車 ”で示したわけです。この場合、“ 登録番号123ABDの車 ”はオブジェクトの名前で、“ 君の車 ”はオブジェクトを参照するためのポインタと考えることができます。

あるものの対象物を明示しないで参照することができると非常に便利です。実際に友人が新しい車に買い替えても、“ 君の車 ”は正しくなっています。ポインタも同じように機能します。例えば、ある場合は数値フィールド“ Age ”を参照したポインタで、数値変数“ Old Age ”を参照することもできます。この場合のポインタは、計算に使用する数値データを参照しています。

テーブル、フィールド、変数、配列、配列要素を参照するためにポインタを使用することができます。次の表に、各タイプの例を示します。

データタイプ	参照	引数として使用	代入
テーブル	私のﾃｰﾌﾞﾙ:=->[ﾃｰﾌﾞﾙ]	DEFAULT TABLE (私のﾃｰﾌﾞﾙ>>)	不使用
フィールド	私のﾌｨｰﾙﾄﾞ:=->[ﾃｰﾌﾞﾙ]ﾌｨｰﾙﾄﾞ	ALERT(私のﾌｨｰﾙﾄﾞ->)	私のﾌｨｰﾙﾄﾞ->:= "ジョン"
変数	私の変数:=->変数	ALERT(私の変数->)	私の変数->:= "ジョン"
配列	私の配列:=->配列	COPY ARRAY(私の配列-> ; B)	COPY ARRAY(B;私の配列->)
配列要素	私の要素:=->配列{1}	ALERT(私の要素->)	私の要素->:= "ジョン"

ポインタの使用例

例題を用いてポインタの使用方法について説明します。次の例は、ポインタを通して変数にアクセスする方法を示しています。まず、変数の作成から始めます：

私の変数:="こんにちは"

“ 私の変数 ” は、文字列 “ こんにちは ” を含む変数です。“ 私の変数 ” に対するポインタを作成します：

私のﾎﾟｲﾝﾀ:=->私の変数

ポインタ記号(->)は、“ ･･･に対するポインタを求める ” ことを意味します。ここでは、“ 私の変数 ” を参照するポインタを呼び出します。このポインタは、代入演算子(:=)で “ 私のﾎﾟｲﾝﾀ ” に対して割り当てられます。

“ 私のﾎﾟｲﾝﾀ ” は、“ 私の変数 ” に対するポインタを含む変数です。“ 私のﾎﾟｲﾝﾀ ” は、“ こんにちは ” という “ 私の変数 ” の値を含みませんが、“ 私の変数 ” に含まれる値を指すことはできます。次の式は “ 私の変数 ” の値を返します：

私のﾎﾟｲﾝﾀ->

前述の式は、“ こんにちは ” という文字列を返します。ポインタ記号(->)をポインタの後につけると、そのポインタの指すオブジェクトの値を示します。

オブジェクトの名前を直接使用して参照する代わりに、ポインタ記号(->)を後につけたポインタを使用してオブジェクトを参照することの意味を理解することが重要です。つまり、変数 “ 私の変数 ” を使用することと、式 “ 私のﾎﾟｲﾝﾀ-> ” を使用することは、全く同じ意味です。

例えば、次のステートメントはアラートボックスに文字列 “ こんにちは ” を表示します：

ALERT (私のﾎﾟｲﾝﾀ->)

“ 私のﾎﾟｲﾝﾀ ” を使用して “ 私の変数 ” の値を変更することもできます。下記のステートメントは、変数 “ 私の変数 ” に文字列"さようなら"を代入します：

私のﾎﾟｲﾝﾀ->:="さようなら"

この2つの “ 私のﾎﾟｲﾝﾀ-> ” を使用したステートメントのとおり、“ 私の変数 ” を使用した場合と全く同じ動作を実行します。次の2つのステートメントも、全く同じ動作を実行します。両方とも、変数 “ 私の変数 ” の現在の値をアラートボックスに表示します：

ALERT (私のﾎﾟｲﾝﾀ->)
ALERT (私の変数)

次の2つのステートメントも、全く同じ動作を実行します。両方とも“私の変数”に、文字列“さようなら”を代入します。

```
私のﾎﾟｲﾝﾀ->:="さようなら"
私の変数:="さようなら"
```

ボタンに対するポインタの使用

この節では、ボタンを参照するためのポインタの使用方法について説明します。ボタン(プログラミング言語の観点から)は、変数に属します。この節では、ボタンを参照するためのポインタの例を使用していますが、ここで示す概念は、すべての種類のポインタに適用されます。

フォーム上に、それぞれの条件によって、使用可または使用不可にする必要のあるボタンが数多くあるとしましょう。ボタンを使用可にしたり、使用不可にする必要があるたびに、次のような判定を実行します：

```
If (条件)                                ` 条件が“True”の場合...
 ENABLE BUTTON (私のボタン)              ` ボタンを使用可にする
Else                                     ` それ以外の場合...
 DISABLE BUTTON (私のボタン)             ` ボタンを使用不可にする
End if
```

フォーム中のその他のボタンに対しても、同様の判定を実行する必要があります。さらに効率良く判定処理を実行するためには、各ボタンの参照にポインタを用いて判定を実行するサブルーチンを採用します。

サブルーチンを使用する場合は、ポインタを使用しなければなりません。これは、これ以外の方法では、ボタンの変数を参照することができないからです。例えば、次のサブルーチン“Set Button”はポインタを使用してボタンを参照しています：

```
   `$1－ブール：“True”は、ボタンを使用可に、“False”は、ボタンを
`使用不可にする
   `$2－ボタンのポインタ
If ($1)                                  ` 条件が“True”の場合...
 ENABLE BUTTON ($2->)                    ` ボタンを使用可にする
Else                                     ` それ以外の場合...
 DISABLE BUTTON ($2->)                   ` ボタンを使用不可にする
End if
```

サブルーチン“Set Button”は次のように呼び出されます：

```
Set Button (判定1 ; ->ボタン1)
Set Button (判定2 ; ->ボタン2)
```

テーブルに対するポインタの使用

プログラミング言語でテーブルをアクセスする場合には、ポインタを使用して、テーブルを参照することができます。テーブルへのポインタは、主にテーブルを操作するコマンドの最初の引数として指定されます。

次のようなステートメントで、テーブルのポインタを作成します：

```
テーブルp:=->[テーブル1]
```

あるいは、次のように**Table**コマンドを使用してテーブルのポインタを呼び出すこともできます：

```
テーブルp:= Table (1)
```

次のようにコマンドに対して、参照するテーブルを指定することもできます：

```
DEFAULT Table (テーブルp->)
```

フィールドに対するポインタの使用

プログラミング言語でフィールドをアクセスする場合には、ポインタを使用して、フィールドを参照することができます。

次のようなステートメントで、フィールドのポインタを作成します：

```
フィールドp:=->[テーブル1]フィールド2
```

あるいは、次のように**Field**コマンドを使用してフィールドのポインタを呼び出すこともできます：

```
フィールドp:=Field (1 ; 2)
```

次のようにコマンドに対して、参照するフィールドを指定することもできます：

```
FONT (フィールドp-> ; "Osaka")
```

配列要素に対するポインタの使用

配列要素に対するポインタを作成することができます。下記のステートメントは配列を作成し、変数 “ 要素p ” に最初の配列要素を指し示すポインタを割り当てます。

```
ARRAY REAL (配列 ; 10)                ` 配列の作成
要素p:=->配列{1}                      ` 配列要素に対するポインタの作成
```

次のように、ポインタを使用して要素に値を代入することができます：

```
要素p->:=1
```

配列に対するポインタの使用

配列に対するポインタを作成することができます。次のステートメントは、配列を作成し、変数“配列p”に最初の配列を指し示すポインタを割り当てます：

```
ARRAY REAL (配列 ; 10)                    ` 配列の作成
配列p:=->配列                             ` 配列へのポインタの作成
```

これは、ポインタが配列の要素を指しているのではなく、配列を指しています。ポインタを、次のように使用することもできます：

```
SORT ARRAY ( 配列p-> ; >)                 ` 配列をソートする
```

例えば、ポインタを使用して配列の4番目の要素を参照する場合は、次のように記述します：

```
配列p->{4}:=84
```

この方法はポインタ配列を使用することとは異なります。ポインタ配列については、次の節を参照してください。

ポインタ配列の使用

関連するオブジェクトのグループを参照する場合にポインタ配列を持つと便利な場合があります。

そのようなオブジェクトのグループの例として、フォーム上の変数のグリッドがあります。グリッドのそれぞれの変数には、“変1”、“変2”、...、“変10”という連番が付いています。間接的にこれらの変数を数字で参照することができます。ポインタの配列を作成し、各変数を指すためにポインタを初期化すれば、変数を簡単に参照することができます。例えば、配列を作成し、各要素を初期化するために、次のようなステートメントを使用します：

```
ARRAY POINTER (変 ; 10)              `10個のポインタを保存するため配列を作成する
For ($i ; 1 ; 10)                    ` 各変数に対して1回ずつループする
  変{$i}:=Get pointer ("変" +String ($i))` 配列要素を初期化する
End for
```

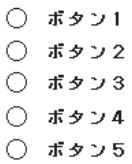
Get pointer関数は、指定されたオブジェクトへのポインタを返します。

ポインタに対応する変数を参照するためには、配列要素を使用します。例えば、変数に10個の日付を代入する場合に(変数が日付タイプであると想定して)、次のようなステートメントを使用します：

```
For ($i ; 1 ; 10)                    ` 各変数に対して1回ずつループする
  変{$i}->:=Current date + $i         ` 日付を代入する
End for
```

ポインタを使用したボタンの設定

フォーム中に関連する一連のラジオボタンがある場合に、それらを素早く設定しなければならないことがあります。名前を使用して各ラジオボタンを直接参照することは、効率が良くありません。次の図に、ボタン1、ボタン2、. . . 、ボタン5という名前の5つのラジオボタンを示します。



一連のラジオボタンでは、1つのラジオボタンのみがオンになります。オンになっているラジオボタンの番号は、数値フィールドに記憶されます。例えば、“ 設定 ” フィールドの値が3の場合は、ボタン3に1を設定します。フォームメソッドで、次のようなステートメントを使用して、ボタンをセットします：

```
If (Before)
 Case of                    ` “ 設定 ” フィールドを判定する
 \ (設定=1)                 ` “ 設定 ” フィールドが “ True ” の場合...
  ボタン1:=1                ` ラジオボタンをオンにする
 \ (設定=2)
  ボタン2:=1
 \ (設定=3)
  ボタン3:=1
 \ (設定=4)
  ボタン4:=1
 \ (設定=5)
  ボタン5:=1
 End case
End if
```

各ラジオボタンごとに判定しなければなりません。フォーム内に多くのラジオボタンがあると、非常に長いメソッドになる場合もあります。この問題を解決するために、ポインタを使用します。

Get pointer関数を使用して、ラジオボタン(あるいは任意のボタン)のポインタを作成することができます。値を設定する必要のあるラジオボタンを参照するためにポインタを使用します。次に改善されたメソッドの例を示します：

```
If (Before)
  $P:=Get pointer ("ボタン" + String (設定))
  $P->:=1                                          ` ラジオボタンをオンにする
End if
```

ラジオボタンの番号が、“ 設定 ” フィールドに記憶される必要があります。これは、各ラジオボタンに対してオブジェクトメソッドを1行加えるだけで実現されます。“ ボタン3 ” に対するオブジェクトメソッドを次に示します：

```
設定:=3
```

メソッドに対するポインタの受け渡し

ポインタを引数としてメソッドに渡することができます。メソッド内で、ポインタによって参照されるオブジェクトを修正することができます。

例えば、次のメソッド “ Take Two ” は、引数に2つのポインタ持ちます。最初の引数は、大文字に変換するオブジェクトを参照します。2番目の引数は小文字に変換するオブジェクトを参照します。次に、そのプメソッドを示します：

```
` $1－文字列のポインタで、大文字に変更 する
` $2－文字列のポインタで、小文字に変更 する
$1->:=Uppercase ($1->)
$2->:=Lowercase ($2->)
```

次のステートメントは、フィールドを大文字に、変数を小文字に変更するためにメソッド “ Take Two ” を使用します：

```
Take Two (->[私のテーブル]私のフィールド ; ->私の変数)
```

このフィールド “ [私のテーブル]私のフィールド ” が、“ jones ” であれば、“ JONES ” に変更されます。一方、変数 “ 私の変数 ” が “ HELLO ” であれば、“ hello ” に変更されます。

メソッド “ Take Two ” で使用するポインタと、参照されるオブジェクトのデータタイプが一致していることが重要です。この例では、ポインタに対して必ず文字列のオブジェクトを割り当てなければなりません。

ポインタに対するポインタ

ポインタを使用して他のポインタを参照することができます。次の例を見てみましょう：

```
私の変数:="こんにちは"
ポイント1:=->私の変数
ポイント2:=->ポイント1
(ポイント2->)->:="さようなら"
ALERT ((ポイント2->)->)
```

このステートメントは、アラートボックスに文字列 “ さようなら ” を表示します。この例は、ポインタの使用に内在する複雑性を示しています。つまり、ポインタがどこの何を指すのかを知る必要があります。

各行について説明していきます：

■ **私の変数:="こんにちは"**

これは、単に文字列 “ こんにちは ” を変数 “ 私の変数 ” に代入します：

■ **ポイント1:=->私の変数**

“ ポイント1 ” を “ 私の変数 ” のポインタにします：

■ **ポイント2:=->ポイント1**

“ ポイント2 ” は “ ポイント1 ” のポインタで、“ 私の変数 ” を指し示します：

■ **(ポイント2->)->:="さようなら"**

この場合の “ ポイント2-> ” は、“ ポイント1 ” の値を参照し、“ 私の変数 ” の位置を指します。“ ポイント2->-> ” は、“ ポイント1 ” の値の指す位置を参照します。つまり、“ (ポイント2 ->)-> ” は、“ 私の変数 ” の値を参照することになります。従って、“ 私の変数 ” に文字列 “ さようなら ” を代入します：

■ **ALERT ((ポイント2->)->)**

上記のステートメントは、“ 私の変数 ” の値をアクセスします。“ (ポイント2->)-> ” は、“ 私の変数 ” の値を呼び出します。このステートメントでは、“ 私の変数 ” の位置を指すものとシンタックスが異なる点に注目してください。

次のステートメントは、文字列 “ こんにちは ” を “ 私の変数 ” に代入します：

(ポイント2->)->:="こんにちは"

次のステートメントは、“ 私の変数 ” から文字列 “ こんにちは ” を取り出し、“ 新変数 ” にそれを代入します：

新変数:=(ポイント2->)->

プロセス　10

4th Dimensionにおけるマルチタスクにより、個々のデータベース処理を同時に行います。これらの個々のデータベース処理を“ プロセス ”と呼びます。

マルチプロセスは、あたかも同じコンピュータに複数のユーザがいて独自の処理を行っているようなものです。これは、メソッドが個々のデータベース作業に対応して実行できることを意味します。

この章では、次の項目について説明します：

- プロセスの作成と消去
- プロセスの要素
- ユーザプロセス
- 4th Dimensionが作成するプロセス
- ローカルプロセスとグローバルプロセス
- プロセス間のレコードロック

プロセスの作成と消去

新規プロセスを作成する方法には、次の3つの方法があります：

■「メソッド実行」ダイアログボックスで「新規プロセス」チェックボックスをチェックした後、「ユーザ」モードでメソッドを実行する方法。「メソッド実行」ダイアログボックスで選択したメソッドが“ プロセスメソッド ”です。

■ メニューコマンドを選択してプロセスを開始する方法。「デザイン」モードの「メニューエディタ」において、任意のメニューコマンドを選択して「新規プロセス開始」チェックボックスをクリックする。メニューアイテムに対応するメソッドが“ プロセスメソッド ”です。

■ **New process**関数を使用する方法。**New process**関数の引数として渡されたメソッドが“ プロセスメソッド ”です。

プロセスは次の方法で消去します。最初の2つは自動的に行われます：

■ プロセスメソッドの実行が完了した場合。

■ ユーザがデータベースから抜けた場合。

■ メソッドからプロセスを中止するか、またはデバッガから**ABORT**コマンドを使用した場合。

■「デザイン」モードの「プロセス」メニューから「アボート」メニューコマンドを選択した場合。

プロセスは別のプロセスを作成することができます。プロセスは階層構造にはなっていません。どのプロセスから作成されようとすべてのプロセスは同等です。いったん、“ 親 ”プロセスが“ 子 ”プロセスを作成すると、子プロセスは親プロセスの存在に関係なく、子プロセスは存在し続けます。

プロセスの要素

各プロセスには個々の要素があります。プロセスには次の3種類の要素があります：

■**「インタフェース要素」：**この要素は、プロセスを表示するのに必要な要素で構成されます。

■**「データ要素」：**この要素は、データベース内のデータに関連する情報で構成されます。

■**「ランゲージ(言語)要素」：**この要素は、メソッドの中で使用する任意の要素またはユーザがアプリケーションを開発する際に重要な要素で構成されます。

インタフェース要素

インタフェース要素には次のものがあります：

■**「メニューバー」：**各プロセスは、独自のカレントメニューバーを持っています。最前面のプロセスのメニューバーがデータベースのカレントメニューバーになります。

■**「1つまたは複数のウインドウ」：**各プロセスは、1つまたは複数のウインドウを同時に開くことができます。ウインドウを全く持たないプロセスもあります。

■**「1つのアクティブ（最前面）ウインドウ」：**プロセスは、複数のウインドウを同時に開くことができますが、アクティブウインドウは各プロセスに1つしかありません。複数のアクティブウインドウを持つには、アクティブウインドウの数だけのプロセスを起動しなければなりません。

データ要素

データ要素は、データベースが使用するデータです。次のようなデータ要素があります：

■**「テーブルごとのカレントセレクション」：**各プロセスは、個々にカレントセレクションを持っています。同じテーブルに対して別々のプロセスにそれぞれのカレントセレクションを持つことができます。

■**「テーブルごとのカレントレコード」：**各テーブルは、プロセスごとに異なるカレントレコードを持つことができます。

注：この記述は、プロセスがグローバルの場合に有効です。デフォルトでは、すべてのプロセスはグローバルです。グローバルプロセスとローカルプロセスの違いに関する詳細は、後述の「グローバルプロセスとローカルプロセス」の節を参照してください。

ランゲージ要素

プロセスのランゲージ要素は、4th Dimensionのプログラミングに関連した要素で構成されます。

■**「変数」：**各変数は、独自のプロセス変数を持っています。詳細は、第2章のプロセス変数の節を参照してください。

■**「デフォルトテーブル」：**プロセスごとに固有のデフォルトテーブルを持っています。ただし、**DEFAULT TABLE**コマンドは単なるプログラミング上の規則です。

■**「入力フォームと出力フォーム」：**各プロセスの各テーブルに対して、デフォルトの入力フォームと出力フォームをメソッドから設定することができます。

■**「UserSet、LockedSetおよびプロセスセット」：**各プロセスは、独自のプロセスセットを持っています。UserSetとLockedSetはプロセスセットです。プロセスセットはプロセスメソッドが終了すると直ちに消去されます。

■**「プロセスごとのOn Error Call」：**各プロセスは、独自のエラーメソッドを持っています。

■**「デバッグウインドウ」：**各プロセスは、独自のデバッグウインドウを持つことができます。これに関する詳細は、第8章の「デバッグ」の節を参照してください。

ユーザプロセス

ユーザプロセスは、あるタスクを実行するためにユーザによって作成されるプロセスです。

4th Dimensionによって作成されるプロセス（カーネルプロセス）

次のプロセスは、4th Dimensionによって作成、管理されます：

■**「ユーザ／カスタムメニュープロセス」：**ユーザ／カスタムメニュープロセスは、「カスタム」モードと「ユーザ」モードで構成されます。「カスタム」モードのデフォルトの初期画面(スプラッシュ画面)はユーザ／カスタムメニュープロセスの一部でもあります。このプロセスは、4th Dimensionの起動直後に作成されます。

■**「デザインプロセス」：**デザインプロセスは、別プロセスとして実行される「デザイン」モードで構成されます。デザインプロセスは、「デザイン」モードで「ファイル」メニューの「閉じる→ストラクチャ」メニューコマンドを使用し閉じることができます。コンパイルしたばかりのデータベースにはストラクチャプロセスはありません。デザインプロセスは、ユーザが最初に「デザイン」モードに入った場合にのみ作成されます。アプリケーションが「ユーザ」モードまたは「カスタム」モードで開く場合には、このプロセスは作成されません。

■**「Webサーバプロセス」：**Webサーバプロセスは、Web上にデータベースが公開された時に起動します。これに関する詳細は、「Webサービス」の節を参照してください。

■**「キャッシュマネージャプロセス」：**キャッシュマネージャプロセスは、データベースのディスク入出力を管理します。このプロセスは、4th Dimensionの起動直後に作成されます。

■**「インデックス作成プロセス」：**インデックス作成プロセスは、データベース内のインデックスフィールドを別プロセスとして管理します。このプロセスは、フィールドのインデックスを作成するときに起動されます。

■**「On Serial Port Managerプロセス」：**On Serial Port Managerプロセスは、プロセスメソッドとしてシリアルメソッドを持つプロセスです。このプロセスは、**ON SERIAL PORT CALL**コマンドでシリアル処理メソッドがインストールされる際に作成されます。

■**「On Event Managerプロセス」：**このプロセスは、**ON EVENT CALL**コマンドでシリアル処理メソッドがインストールされる際に作成されます。このプロセスは、イベントが発生すると**ON EVENT CALL**コマンドがインストールしたイベントメソッドを実行します。このイベントメソッドがこのプロセスの“プロセスメソッド”です。これはメソッドが実行されていなくても実行し続けます。イベント処理は、「デザイン」モードでも行われます。

グローバルプロセスとローカルプロセス

プロセスは、ローカルとグローバルの両方を作成することができます。デフォルトでは、すべてのプロセスはグローバルです。

ほとんどの場合、グローバルプロセスを使用します。グローバルプロセスはデータアクセスやデータ処理等の操作を実行することができます。

ローカルプロセスは、データアクセスを必要としない操作の場合にしか使用してはいけません。例えば、フローティングウインドウやイベント処理メソッドの実行等のインタフェース要素を制御するためにローカルプロセスを使用します。

警告：ローカルプロセスでデータアクセスを行うと、データ変造の危険性があります。

ローカルプロセス名は、先頭にドル記号($)を付ける必要があります。次のコードは、**New process**関数を使ってローカルプロセスを作成しています。

```
pID:=New Process ("私のメソッド" ; 1 ; 32000)
```

プロセス間のレコードのロック

別のプロセスがレコードを変更するために正常にロードされているときは、そのレコードはロックされます。ロックされているレコードは、別のプロセスでロードすることはできますが、変更することはできません。レコードは、それを変更しているプロセス内でのみロックを解除することができます。レコードをロック解除の状態でロードするには、テーブルが“ リードライト(書き込み可能) ”モードになっていなければなりません。

ここでは、4th Dimensionのプログラミング言語を構成する次のような構成要素を定義します：

- 識別子
- データタイプ
- 定数
- 演算子

識別子

本節は、プログラミング言語のさまざまなオブジェクトを指定するための規則を説明します。すべてのオブジェクトの名前は、次の規則に従います：

- 名前は、文字(全角文字または半角アルファベット)で始めます。
- 名前には、文字、数字、スペース、アンダーラインが使用できます。
- ピリオド(.)、スラッシュ(/)、コロン(:)は使用することはできません。
- +、-、*、/、(、)などの演算子に用いられる文字は使用できません。
- 名前の最後につけたスペースは無視されます。

テーブル

角括弧内に名前を入れると、テーブルを表します。テーブル名は、31バイト以内で指定します。

テーブル名	ステートメント内のテーブル名
[注文]	DEFAULT TABLE ([注文])
[顧客]	INPUT FORM ([顧客] ; "入力")
[レター]	ADD RECORD ([レター])

フィールド

フィールドが属するテーブルを最初に指定して、フィールドを表します。フィールドの名前はテーブル名のすぐ後に続けます。フィールド名は、31バイト以内で指定します。

フィールド名の先頭文字にアンダーライン(_)を使用してはいけません。このアンダーライン文字は4Dモジュール用の予約語として使用されます。4th Dimensionは「メソッド」エディタ内のフィールドの先頭でこの文字を見つけると、アンダーラインを取り除きます。

フィールド名	ステートメント内のフィールド名
[注文]総合計	[注文]総合計:=Sum ([明細]合計)
[顧客]名前	SEARCH ([顧客]; [顧客]名前="山田")
[レター]テキスト	*Capitalize* ([レター]テキスト)

トリガ、フォームメソッド、オブジェクトメソッド内で指定する必要がない場合でも、フィールドの前にテーブル名を指定することは、良いプログラミング技法です。

サブテーブル

サブテーブルは、サブテーブルの属するテーブルを最初に指定します。このテーブルは、そのサブテーブルの親テーブルです。サブテーブルの名前は、テーブル名のすぐ後に続けます。サブテーブル名は、31バイト以内で指定します。

サブテーブル名	ステートメント内のサブテーブル名
[従業員]子供	ALL SUBRECORDS ([従業員]子供)
[顧客]電話番号	ADD SUBRECORD ([顧客]電話番号 ; "追加")
[レター]キーワード	NEXT SUBRECORD ([レター]キーワード)

サブテーブルは、フィールドタイプとして扱われます。従って、フォームで使用する場合は、フィールドと同じ規則に従います。トリガ、フォームメソッド、親テーブルのオブジェクトメソッドでサブテーブルを指定する場合に、親テーブル名を指定する必要はありません。しかし、サブテーブル名の前にテーブル名を指定することは良いプログラミング技法です。

サブフィールド

フィールドと同じようにサブフィールドを表します。サブフィールドは、最初にサブフィールドが属するサブテーブルを指定して表します。サブフィールドの名前は、アポストロフィ(')でサブテーブルの名前とを区切ります。サブシールド名は、31バイト以内で指定します。

サブフィールド名	ステートメント内のサブフィールド名
[従業員]子供'名前	[従業員]子供'名前:=Uppercase ([従業員]子供'名前)
[顧客]電話番号'番号	[顧客]電話番号'番号:="03-1234-5678"
[レター]キーワード'ワード	*Capitalize* ([レター]キーワード'ワード)

サブテーブルメソッド、フォームメソッド、サブテーブルのオブジェクトメソッドでサブフィールドを指定する場合に、サブテーブル名を指定する必要はありません。しかし、サブフィールド名の前にテーブル名やサブテーブル名を指定しておくことは、良いプログラミング技法です。

インタープロセス変数

名前の先頭にインタープロセス(<>)記号を付けることによって、インタープロセス変数を表します。インタープロセス変数名は、インタープロセス (<>)記号を除いて31バイト以内で指定します。

インタープロセス変数名	ステートメント内のインタープロセス変数名
<>プロセスID	<>プロセスID:=Current process
<>キー	<>キー:=Keycode
<>カラー	If (<>カラー # 0)

プロセス変数

名前を使用して、プロセス変数を表します。プロセス変数名は、31バイト以内で指定します。

プロセス変数名	ステートメント内のプロセス変数名
総合計	総合計:=Sum ([会計]金額)
ボタン１	If (ボタン１=1)
私の変数	私の変数:="定数文字列"

ローカル変数

ドル記号($)を名前の先頭につけてローカル変数を表します。ローカル変数名は、ドル ($) 記号を除いて31バイト以内で指定します。

ローカル変数名	ステートメント内のローカル変数名
$i	For ($i ; 1 ; 100)
$TempVar	If ($TempVar="No")
$My String	$My String:="こんにちは"

配列

名前を使用して、配列を表します。配列には次の3種類があります：

- インタープロセス配列
- プロセス配列
- ローカル配列

インタープロセス配列

インタープロセス配列の名前は、名前の先頭にインタープロセス(<>)記号が付きます。インタープロセス配列名は、インタープロセス (<>)記号を除いて31バイト以内で指定します。

インタープロセス配列名	ステートメント内のインタープロセス配列名
<>ID配列	ARRAY TEXT (<>ID配列 ; Records in File ([ID]))
<>キーワード	SORT ARRAY (<>キーワード ; >)
<>大きい配列	ARRAY INTEGER (<>大きい配列 ; 10000)

中括弧({})を使用して、インタープロセス配列の要素を参照します。参照される配列要素は数式で表されます。次に例を示します。

インタープロセス配列要素	ステートメント内のインタープロセス配列要素
<>ID配列	<>ID配列{$i}:=[従業員]ID番号
<>キーワード{3}	If (<>キーワード {3}="終わり")
<>大きい配列{$ID}	私のファイル:=<>大きい配列{$ID}

中括弧({})を使用して、2次元配列の要素を参照します。参照される要素は2組の数式で表されます。

インタープロセス配列要素	ステートメント内のインタープロセス配列要素
<>ID配列{$i}{$j}	<>ID配列{$i}{$j}:=[従業員]名前
<>キーワード{3}{2}	If (<>キーワード{3}{2}="終わり")
<>ファイル{$名前1}{$名前2}	私のフィールド:=<>ファイル{$名前1}{$名前2}

プロセス配列

プロセス配列名は、31バイト以内で指定します。

プロセス配列名	ステートメント内のプロセス配列名
アイテム	ARRAY TEXT (アイテム ; 20)
キーワード	SORT ARRAY (キーワード ; >)
ファイル	COPY ARRAY (ファイル ; スクロール)

中括弧({})を使用して、プロセス配列の要素を参照します。参照された配列要素は数式で表されます。次に例を示します。

プロセス配列要素	ステートメント内のプロセス配列要素
アイテム{$i}	アイテム{$i}:=[従業員]名前
キーワード{3}	If (キーワード{3}="終わり")
ファイル{$ID}	私のフィールド:=ファイル{$ID}

中括弧({})を使用して、2次元配列の要素を参照します。参照される要素は2組の数式で表されます。

プロセス配列要素	ステートメント内のプロセス配列要素
ID配列{$i}{$j}	ID配列{$i}{$j}:=[従業員]名前
キーワード{3}{2}	If (キーワード{3}{2}="終わり")
ファイル{$名前1}{$名前2}	私のフィールド:=ファイル{$名前1}{$名前2}

ローカル配列

配列名がドル記号($)で始まるものは、ローカル配列です。ローカル配列名は、ドル ($)記号を除いて31バイト以内で指定します。

ローカル配列名	ステートメント内のローカル配列名
$アイテム	ARRAY TEXT ($アイテム ; 20)
$キーワード	SORT ARRAY ($キーワード ; >)
$ファイル	COPY ARRAY ($ファイル ; スクロール)

中括弧({...})を使用して、ローカル配列の要素を参照します。参照される要素は数式で表されます。次に例を示します。

ローカル配列要素	ステートメント内のローカル配列要素
$アイテム{$i}	$アイテム{$i}:=[従業員]名前
$キーワード{3}	If ($キーワード{3}="終わり")
$ファイル{$名前}	私のフィールド:=$ファイル{$名前}

中括弧({...})を使用して、2次元配列の要素を参照します。参照される要素は2組の数式で表されます。

ローカル配列要素	ステートメント内のローカル配列要素
$アイテム{$i}{$j}	$アイテム{$i}{$j}:=[従業員]名前
$キーワード{3}{2}	If ($キーワード{3}{2}="終わり")
$ファイル{$名前1}{$名前2}	私のフィールド:=$ファイル{$名前1}{$名前2}

フォーム

フォームの名前は、文字列式を使用して表します。フォーム名は、31バイト以内で指定します。

フォーム名	ステートメント内のフォーム名
"入力"	INPUT FORM ([従業員] ; "入力")
"出力"	OUTPUT FORM ([従業員] ; "出力")
"名前" + String ($i)	DIALOG ([ステージ] ; "名前" + String ($i))

メソッドと関数

名前を使用して、メソッドや関数を表します。メソッド名は、31バイト以内で指定します。

メソッド名	ステートメント内のメソッド名
新規顧客	If (*新規顧客*)
重複データ削除	*重複データ削除*
大文字変換	APPLY TO SELECTION (*大文字変換*)

メソッドや関数に、引数を渡すことができます。引数は、メソッドまたは関数の後の括弧内に記述します。各引数は、セミコロン(;)で次のものと区切ります。

メソッド名	引数を受け取るメソッド名
Drop Spaces	[従業員]名前:=*Drop Spaces* ([従業員]名前)
Creator	*Creator* (1 ; 5 ; Nice)
Dump	Clone:=*Dump* ("is" ; "the" ; "it")

引数は、呼び出されたメソッドまたは関数内で、連番付きのローカル変数$1、 $2、. . . 、$nとして使用できます。

関数は値を返します。関数内のローカル変数 “ $0 ” に、返す値が代入されます。

プラグインルーチン（外部ルーチン、関数、エリア）

名前を使用して、プラグインルーチンを表します。プラグインルーチン名は、31バイト以内で指定します。

外部ルーチン名	ステートメント内の外部ルーチン名
SP New offscreen area	新規エリア:=*SP New offscreen area*
DR PUBLISH	*DR PUBLISH* (エリア ; 0 ; $Temp)

セット

セットには、2種類あります：

- インタープロセスセット
- プロセスセット

インタープロセスセット

インタープロセスセットの名前は、名前の先頭にインタープロセス(<>)記号が付きます。インタプロセスセット名は、インタープロセス (<>)記号を除いて80バイト以内で指定します。

インタープロセスセット名	ステートメント内のインタープロセスセット名
"ラレコード削除"	USE SET ("ラレコード削除")
"ラ顧客注文"	CREATE SET ([顧客] ; "ラ顧客注文")
"ラ選択" + String ($i)	Records in set ("ラ選択" + String ($i))

プロセスセット

セットの名前を表す文字列式を使用してプロセスセットを表します。インタプロセスセット名は、80バイト以内で指定します。

プロセスセット名	ステートメント内のプロセスセット名
"レコード削除"	USE SET ("レコード削除")
"顧客注文"	CREATE SET ("顧客注文")
"選択" + String ($i)	Records in set ("選択" + String ($i))

クライアントセット

クライアントセット名は、名前の先頭にドル($)記号を指定します。クライアントセット名は、ドル記号を除いて31バイト以内で指定します。

注：バージョン1.5までの4D Client／Serverでは、セットはセットが作成されたクライアントマシン上で保守されていました。バージョン6からは、セットはサーバマシン上で保守されるようになりました。効率や特殊目的のために、クライアントマシン内でローカルにセットで作業をする必要がある場合もあります。これを行うときにはクライアントセットを使用します。

クライアントセット名	ステートメント内のクライアントセット名
"$レコード削除"	USE SET ("$レコード削除")
"$顧客注文"	CREATE SET ("$顧客注文")
"$選択" + String ($i)	Records in set ("$選択" + String ($i))

プロセス

プロセスには、2種類あります：

- グローバルプロセス
- ローカルプロセス

グローバルプロセス

プロセスの名前を表す文字列式を使用してグローバルプロセスを表します。グローバルプロセス名は、31バイト以内で指定します。

グローバルプロセス名	ステートメント内の外部グローバルプロセス名
"イベントプロセス"	MyProc:=New Process ("EventProc" ; 32768 ; "イベントプロセス")

ローカルプロセス

ドル記号($)を名前の前につけてローカルプロセスを表します。ローカルプロセス名は、ドル ($)記号を除いて31バイト以内で指定します。

ローカルプロセス名	ステートメント内のローカルプロセス名
"$イベントマネージャ"	"$イベントマネージャ"
"$チェック"	MyLocal:=New Process (My Proc ; 16*1024 ;"$チェック")
"$シリアルマネージャ"	"$シリアルマネージャ"

命名セレクション

文字列式を使用して、命名セレクションを表します。命名セレクション名は、31バイト以内で指定します。

命名セレクション	ステートメント内の命名セレクション
郵便番号	USE NAMED SELECTION ([顧客] ; "郵便番号")
顧客注文	顧客注文
選択 + String ($i)	選択 + String ($i)

インタープロセス命名セレクション

インタープロセス命名セレクションの名前は、名前の先頭にインタープロセス(<>)記号が付きます。インタプロセス命名セレクション名は、インタープロセス (<>)記号を除いて31バイト以内で指定します。

インタープロセス命名セレクション名	ステートメント内のインタープロセス命名セレクション名
"<>郵便番号"	USE NAMED SELECTION ([顧客] ; "<>郵便番号")
"<>顧客注文"	"<>顧客注文"
"<>選択" + String ($i)	"<>選択" + String ($i))

名前が重複する場合

特定のオブジェクトが別タイプのオブジェクトと同じ名前を持つ場合(例えば、フィールドが“ Person ”という名前で、変数も“ Person ”という名前の場合)に、4th Dimensionは、4th Dimension内部のオブジェクトを識別するための優先順位システムによって認識します。従って、重複しない名前の使用に関してはユーザ自身に委ねられます。

4th Dimensionは、メソッドで使用される名前を次の順位で識別します：

1. **フィールド**
2. **コマンド**
3. **メソッド**
4. **プラグインルーチン**
5. **定義済定数**
6. **変数**

例えば、4th Dimensionには、“ **Date** ”という組み込み関数があります。メソッドに“ Date ”という名前を付けると、4th Dimensionは組み込み関数の“ **Date** ”として認識し、メソッドとしては認識しません。つまり、メソッドからの呼び出しができなくなります。しかし、もし、フィールドに“ Date ”と定義すると、4th Dimensionは**Date**関数の代わりにフィールドとして使用します。

データタイプ

4th Dimensionのフィールド、変数、式は以下のデータタイプになります。

データタイプ	フィールド	変数	式
文字列（備考1参照）	○	○	○
数値（備考2参照）	○	○	○
日付	○	○	○
時間	○	○	○
ブール	○	○	○
ピクチャ	○	○	○
ポインタ	×	○	○
BLOB（備考3参照）	○	○	×
配列（備考4参照）	×	○	×
サブテーブル	○	×	×
未定義	×	○	○

備考

1. 文字列には、英数字フィールド、固定長変数、テキストフィールド、テキスト変数があります。

2. 数値には、実数、整数、倍長整数の各フィールドと変数があります。

3. BLOBはBinary Large OBjectの省略形です。BLOBの詳細については、BLOBコマンドの節を参照してください。

4. 配列にはすべてのタイプの配列が含まれます。詳細については配列セクションを参照してください。

文字列

文字列とは、以下を示す総称です。

- 英数字フィールド
- 固定長変数
- テキストフィールドまたはテキスト変数
- 任意の文字列またはテキスト式

文字列は、文字で構成されます。文字は256種類のASCIIコードのいずれかです。ASCIIコードと、4DがASCIIコードをクロスプラットフォーム環境で処理する手順については、ASCIIコードセクションを参照してください。

- 英数字フィールドは0個から80個までの文字を含むことができます(文字数の制限はフィールド定義で決まります)。
- 固定長変数は0個から255個までの文字を含むことができます(その制限は変数宣言によって決まります)。
- テキストフィールド、テキスト変数、テキスト式は0個から32,000個までの文字を含むことができます。

テキストフィールドには文字列を代入することができ、また、反対に文字列にテキストフィールドを代入することができます。4Dが変換を行い、必要に応じて切り捨てを行います。文字列とテキストは1つの式の中で混在させて使用できます。

注：このマニュアルでは、コマンド説明の文字列とテキストの引数（パラメータ）は特に明記されていない限り、両方とも「文字列」と表記されています。

数値

数値とは、以下を示す総称です。

- 実数のフィールド、変数、または式
- 整数のフィールド、変数、または式
- 倍長整数のフィールド、変数、または式

実数データタイプの範囲は、±1.7e±ｱ308(15桁)です。

整数データタイプ(2バイト整数)の範囲は、-32,768から32,767まで(2^15から(2^15)-1まで)です。

倍長整数データタイプ(4バイト整数)の範囲は、-2^31から(2^31)-1までです。

数値データタイプは、別のデータタイプに代入することができます。このとき、4th Dimensionが必要に応じて変換、切り捨て、丸め処理を行います。ただし、値が範囲外の場合には、変換は正しい値を返さないことに注意してください。数値データタイプは式の中に混在させて使用することができます。

注：このマニュアルでは、実際のデータタイプに関わらず、コマンド説明の実数、整数、倍数整数の引数（パラメータ）は特に明記されていない限り、「数値」と表記されています。

日付

■ 日付として認識できる範囲は、1904年から2039年までです。

■ 日付の順序は、「年．月．日」の順になります。

■ 年を2桁で指定すると1900年代であると解釈されます(このデフォルトが**SET DEFAULT CENTURY**コマンドで変更されていない場合に限る)。

注：このマニュアルでは、コマンド説明の日付引数（パラメータ）は特に明記されていない限り、「日付」と表記されています。

時間

■ 時間のフィールド、変数、式の範囲は00:00:00から596,000:00:00までです。

■ 時間の順序は、「時：分：秒」の順になります。

■ 時間は24時間制です。

■ 時間の値は数値として扱うことができます。時間から返される数値は時間が表す秒数です。詳細については時間演算子セクションを参照してください。

注：このマニュアルでは、コマンド説明の時間引数（パラメータ）は特に明記されていない限り、「時間」と表記されています。

ブール

ブールのフィールド、変数、式は、TRUE(真)またはFALSE(偽)のいずれかになります。

注：このマニュアルでは、コマンド説明のブール引数（パラメータ）は特に明記されていない限り、「ブール」と表記されています。

ピクチャ

ピクチャのフィールド、変数、式は、任意のWindowsまたはMacintoshのピクチャになります。一般に、クリップボード上に置いたり、4Dコマンドやプラグインコマンドを使用してディスクから読み出すことのできる任意のピクチャがこれに当てはまります。

注：このマニュアルでは、コマンド説明のピクチャ引数（パラメータ）は特に明記されていない限り、「ピクチャ」と表記されています。

ポインタ

ポインタの変数または式は、別の変数(配列、配列要素を含む)、テーブルまたはフィールドへの参照です。ポインタタイプのフィールドは、存在しません。

ポインタの詳細については、第9章を参照してください。

注：このマニュアルでは、コマンド説明のポインタ引数（パラメータ）は特に明記されていない限り、「ポインタ」と表記されています。

BLOB

BLOBのフィールドまたは変数は、個別にまたはBLOBコマンドを使用してアドレスを指定できるバイト群(長さは0バイトから2GBまで)です。BLOBタイプの式は、存在しません。

注：このマニュアルでは、コマンド説明のBLOB引数（パラメータ）は特に明記されていない限り、「BLOB」と表記されています。

配列

配列は、実際にはデータタイプではありません。さまざまな種類の配列(整数配列、テキスト配列など)はこの配列という名前の下にグループ化されています。配列は、変数です。配列タイプのフィールドは存在せず、配列タイプの式も存在しません。配列の詳細については、第9章を参照してください。

注：このマニュアルでは、コマンド説明の配列引数（パラメータ）は特に明記されていない限り、「配列」と表記されています。

サブテーブル

サブテーブルは実際にはデータタイプではありません。サブテーブルのタイプになれるのはフィールドだけです。サブテーブルタイプの変数や式は存在しません。サブテーブルの詳細については、『4th Dimension / 4D First デザインリファレンス』マニュアルとサブレコードのテーマのコマンドを参照してください。

未定義

未定義は、実際にはデータタイプではありません。未定義は、まだ定義されていない変数を指しています。

関数(結果を返すプロジェクトメソッド)は、メソッド内にある場合には未定義値を返すことができ、関数の結果($0)は未定義式(少なくとも1つの未定義変数で計算された式)に代入されます。フィールドは、未定義にはできません。

データタイプの変換

4th Dimension言語には演算子とコマンドがあり、変換の実行が意味のあるところでデータタイプを変換できます。4th Dimension言語は、データタイプの検証を強化しています。例えば、"abc" + 0.5 + !97/08/21! - ?00:30:45?のように表記することはできません。これは、シンタックス（構文）エラーになります。

次の表は、基本的なデータタイプ、変換できるデータタイプ、それを行う際に使用するコマンドを示しています。

データタイプ	文字列変換	数値変換	日付変換	時間変換
文字列		Num	Date	Time
数値	String			
日付	String			
時間	String			
ブール		Num		

(*) 時間値は数値として扱うことができます。

注：この表に示すデータ変換の他に、演算子と他のコマンドを組み合せることで、より洗練されたデータ変換を行うことができます。

定数

定数は、固定値を持つ式です。定数には、名前で選択できる「定義済定数」と実際の値を入力する「リテラル定数」の2種類があります。

定義済定数

4th Dimensionのバージョン6からあらかじめ定義されている「定義済定数」が導入されました。これらの定数は、「エクスプローラ」ウインドウに次のようにリストされています。

定義済定数は、テーマ別にリストされています。「メソッドエディタ」ウィンドウで定義済定数を使用するには、次の手順で行います：

■「エクスプローラ」ウィンドウから「メソッドエディタ」ウィンドウに定数をドラッグ&ドロップします。

■「メソッドエディタ」ウィンドウに定数の名前を直接入力します。

定義済定数名は、最大31文字まで指定できます。

Tips：定義済定数の名前を直接入力する場合に@記号(アットマーク)を使用すると、定数名すべてを入力しなくても済みます。例えば、「No such da@」と入力してからReturnキーまたはEnterキーを押すと、4th Dimensionはコードの行の正当性を検査するときにその行に「No such data in clipboard」という定数を入れます。

定義済定数は、「メソッドエディタ」ウィンドウや「デバッグ」ウィンドウ内では、次のように下線付きで表示されます。

注：日本版では、下線（アンダーライン）は表示されません。

例えば、上記のウィンドウでは「Is Alpha Field」が定義済定数です。

リテラル定数

リテラル定数は、次の4つのデータタイプにできます。

- 文字列
- 数値
- 日付
- 時間

文字列定数

文字列定数は、次のように2重引用符("...")で囲んで表します。

"レコード追加"
"レコードが見つかりません"
"送り状"

空の文字列は、2つの引用符の間に何も入れない状態("")で指定します。

数値定数

数値定数は、実数として記述します。

下記に数値定数の例をいくつか次に示します：

27
123.76
0.0076

負の数値は、次のようにマイナス記号(-)で指定されます：

-27

-123.76

-0.0076

日付定数

日付定数は、エクスクラメーションマーク(! ... !)で囲んで表します。

日付は、“ 年.月.日 ” の順で表し、それぞれをピリオド(.)で区切ります。

下記に、日付定数の例を示します：

!76.1.1!

!04.4.4!

!97.07.25!

空の日付は、!00.00.00!のように指定します。

2桁の年度は、1900年代であると認識します。

時間定数

時間定数は、2つのクエスチョンマーク(? ... ?)で囲んで表します。

時間は、“ 時:分:秒 ” の順で表し、それぞれをコロン(:)で区切ります。

時間は、24時間制で指定します。

分または秒を省略した場合は、それはゼロと想定されます。例えば、?1? は、?1:00?と等しく、また、?01:00:00? とも等しくなります。

時間定数の例を次に示します：

?00:00:00? ` **午前0時**
?09:30:00? ` **午前9時30分**
?13:01:59? ` **午後1時1分59秒**

空の時間は、?00.00.00?のように指定します。

演算子

演算子とは、式の間で行われる操作を指定するために使用する記号です。

演算子の機能は、以下のとおりです：

- 数値、日付、時間の計算を行います。
- 文字列操作、論理式における論理演算、図形上での特殊演算を行います。
- 単純な式を組み合わせて新たな式を生成します。

優先順位

式を評価する順番を“ 優先順位 ”と呼びます。4th Dimensionの優先順位は左から右です：

3+4*5

上記の式は、35を返します。最初に式 3+4 の結果 7を求め、それに5を乗じますので、結果は35になります。

括弧は、左から右の優先順位を無視するために使用します：

3 + (4*5)

この式は、23を返します。最初に式 (4*5) の結果 20を求め、それに 3を加えて、結果は 23 になります。

括弧は、他の括弧の組の内側にネストすることができます。各左括弧と右括弧との数に注意してください。

式の評価が適切に行われるように注意を払わなければなりません。括弧がなかったり、誤って使用した場合は、予測できない結果になります。

また、4D Compilerでアプリケーションをコンパイルする場合は、左括弧と右括弧は同じ数でなければなりません。コンパイラは、組になっていない括弧をシンタックスエラーとして検出します。

代入演算子

代入演算子(:=)は、右側の式の値を演算子の左側の変数やフィールドにコピーします。

例えば、次のステートメントは、3("Aci"の文字数)を変数“ 私の変数 ”に代入します。“ 私の変数 ”のデータタイプは数値です。

私の変数:=Length ("Aci")

文字列演算子

次の表に、文字列演算子を示します。文字列演算子を使用する式は、文字列を返します。

演算子	シンタックス	結果	例
連結　+	文字列 + 文字列	文字列	"abc" + "def" → "abcdef"
反復　*	文字列 * 文字列	文字列	"ab" * 3 → "ababab"

数値演算子

次の表に、数値演算子を示します。数値演算子を使用する式は数値を返します。数値は全ての演算子で返されます。

モジューロ演算子(%)は最初の数値を2番目の数値で除算し、その余りの整数を返します。

10%2は、0を返します。10は2で割り切れるからです。
10%3は、1を返します。余りが1だからです。
10.5%2は、0を返します。剰余が整数でないからです。

演算子	シンタックス	結果	例
加算　+	数値 + 数値	数値	2 + 3 → 5
減算　-	数値 - 数値	数値	3 - 2 → 1
乗算　*	数値 * 数値	数値	5 * 2 → 10
除算　/	数値 / 数値	数値	5 / 2 → 2.5
整数値を返す除算　//	数値 // 数値	数値	5 // 2 → 2
モジューロ　%	数値 % 数値	数値	5 % 2 → 1
指数　^	数値 ^ 数値	数値	2 ^ 3 → 8

日付演算子

次の表に、日付演算子を示します。日付演算子を使用する式は、日付または数値を演算の種類に応じて返します。日付演算は、閏年も考慮に入れて正確な結果を返します。

演算子	シンタックス	結果	例
日付の差　-	日付 - 日付	数値	!96.01.20! - !96.01.01! → 19
日付の加算　+	日付 + 数値	日付	!96.01.20! + 9 → !96.01.29!
日付の引算　-	日付 - 数値	日付	!96.01.20! - 9 → !96.01.11!

時間演算子

次の表に、時間演算子を示します。時間演算子を使用する式は、時間または数字を演算の種類に応じて返します。

演算子	シンタックス	結果	例
加算 +	時間 + 時間	時間	?02.03.04? + ?01.02.03? → ?03.05.07?
減算 -	時間 - 時間	時間	?02.03.04? - ?01.02.03? → ?01.01.01?
加算 +	時間 + 数値	数値	?02.03.04? + 65 → 7449
減算 -	時間 - 数値	数値	?02.03.04? - 65 → 7319
乗算 *	時間 * 数値	数値	?02.03.04? * 2 → 14768
除算 /	時間 / 数値	数値	?02.03.04? / 2 → 3692
整数値を返す除算 //	時間 // 数値	数値	?02.03.04? // 2 → 3692
モジューロ %	時間 % 数値	数値	?02.03.04? % 2 → 0

Tips：

(1) 時間式を数字と組み合わせた式から時間式を取得するには、**Time**コマンドと**Time string**コマンドを使用します。

次の行は、$vlSecondsに夜中の12時から今から1時間後までの秒数を割り当てます。

$vlSeconds:=**Current Time**+3600

次の行は、$vHSoonに今から1時間後の時間を割り当てます。

$vhSoon:=**Time** (**Time string**(**Current time**+3600))

2行目はより簡単な方法で書けます。

次の行は、$vHSoonに今から1時間後の時間を割り当てます。

$vhSoon:=**Current time**+?00:01:00?

ただし、アプリケーションを開発する際、秒数で表現され時間値に追加される遅延が数値としてしか利用できない状況に遭遇することがあります。

この場合、次のヒントを参考にしてください。

(2) 状況によっては、時間表現を数値表現に変換する必要があります。例えば、**Open Document**関数を使用して文書をオープンすると、文書参照(DocRef)が返されますが、これは元々時間式です。後で、文書参照として数値を受け取る4D Extensionルーチンに DocRefを渡すとします。この場合、ゼロに加えることにより、時間値からその値を変更することなく数値を取得できます。

```
` 文書を選択して、オープンする
$vhDocRef:=Open document("")
If (OK=1)
`数値表現としてのDocRef時刻表現を4D Extensionルーチンに渡します。
   DO SOMETHING SPECIAL (0+$vhDocRef)
End if
```

比較演算子

この節の表は文字列、数値、日付、時間、ポインタに適用する比較演算子を示しています。比較演算子を使用する式は、“ True(真) ” または “ False(偽) ” のブール値を返します。

文字列比較に関する注意点をいくつか挙げます。

■ 文字ごとに比較します。

■ 文字の大小文字を無視します。従って、"a"="A"は “ True(真) ” を返します。大文字と小文字を意識して比較する場合には、ASCIIコードで比較してください。例えば、次の式は “ False(偽) ” です：

```
Ascii ("A")=Ascii ("a")
```

■ 文字列を比較する場合、発音区別記号は使用するコンピュータのシステム文字比較テーブルを使用して比較されます。例えば、以下の式はTRUEを返します：

```
"A" = "å"
```

■ ワイルドカード記号(@)は、すべての文字列の比較に使用することができます。これは、任意の数の文字を比較します。従って、次の式は “ True(真) ” になります：

```
"abcefghij"="abc@"
```

ワイルドカード記号は、比較演算子の右側の式で使用しなければなりません。従って、次の式は “ False(偽) ” です：

```
"abc@"="abcefghij"
```

ワイルドカードは、"1つ以上の文字または無"という意味です。次の式は “ True(真) ” になります：

"abcdefghij" = "abcdefghij@"
"abcdefghij" = "@abcdefghij"
"abcdefghij" = "abcd@efghij"
"abcdefghij" = "@abcdefghij@"
"abcdefghij" = "@abcde@fghij@"

一方、どのような場合でも、ワイルドカードを2つ連続して使用した文字列比較は常にFalse(偽)を返します。次の式はFalse(偽)になります：

"abcdefghij" = "abc@@"

Tips：

文字列をデータ入力から取得する場合、文字列で@マークを使用することができますが、他の文字と同様にこのワイルドカードを扱うことはできません。次の例を考えてみましょう。

```
$vsValue:=Request("検索したい値を入力してください。")
If (OK=1)
   QUERY ([顧客];[顧客]名前=$vsValue+"@")
End if
```

値は、**Request**コマンドを使用して要求します。次に、この値は “ 先頭一致 ” クエリによって使用されます。@マークを2つ続けると、前述のように比較結果がFALSEになるので、値の後に@を付けることができるのは、最後の文字が@でない場合のみです。これは、次のように変更した例で行うことができます。

```
$vsValue:=Request("検索したい値を入力してください。")
If (OK=1)
   If (Ascii($vsValue[[Length($vsValue)]]) # 64)
      $vsValue:=$vsValue+"@"
   End if
   QUERY ([顧客];[顧客]名前=$vsValue)
End if
```

次の式が($vsValueが空でなければ)常にTRUEを返すので、ここでは**Ascii**関数を使用する必要があります。

```
$vsValue[[Length($vsValue)]])="@"
```

さらにこの例について述べると、「リクエスト」ダイアログボックスに複数の@マークを使った“ @@D@OE@@@ ”のような文字列を入れることもできます。次のコードは、文字列に存在する@文字をすべて削除します。

```
` 「No at signs」プロジェクトメソッド
` No at signs (文字列) → 文字列
` No at signs (任意の文字列 ) → @なしの文字
$0:=""
For ($vlChar ; 1 ; Length($1))
   If (Ascii($1[[$vlChar]]) # 64)
      $0:=$0+$1[[$vlChar]]
   End if
End for
```

言い換えると、この短いプロジェクトメソッドは**Replace string**関数と同じ操作を行います。ただし、次の**Replace string**式は常に空の文字列を返すので、このプロジェクトメソッドが必要になります。

```
Replace string($vsValue ; "@" ; "") ` すべての文字が削除される
```

最終的に、この例は以下のようになります。

```
$vsValue:=Request("検索したい値を入力してください。")
If (OK=1)
   QUERY ([顧客];[顧客]名前=Not at signs ($vsValue)+"@")
End if
```

このコードでは、「リクエスト」ダイアログボックスにどのような文字を入力しようと、クエリは、常に“ 先頭一致 ”クエリになります。

文字列比較演算子

演算子	シンタックス	結果	例
等号(=)	文字列 = 文字列	ブール	"abc"="abc" → “ True(真) ” "abc"="abd" → “ False(偽) ”
不等号(#)	文字列 # 文字列	ブール	"abc"#"abd" → “ True(真) ” "abc"#"abc" → “ False(偽) ”
より大きい(>)	文字列 > 文字列	ブール	"abd">"abc" → “ True(真) ” "abc">"abc" → “ False(偽) ”
より小さい(<)	文字列 < 文字列	ブール	"abc"<"abd" → “ True(真) ” "abc"<"abc" → “ False(偽) ”
より大きいまたは等しい(>=)	文字列 >= 文字列	ブール	"abd">="abc" → “ True(真) ” "abc">="abd" → “ False(偽) ”
より小さいまたは等しい(<=)	文字列 <= 文字列	ブール	"abc"<="abd" → “ True(真) ” "abd"<="abc" → “ False(偽) ”

数値比較演算子

演算子	シンタックス	結果	例
等号(=)	数値 = 数値	ブール	10 = 10 → “ True(真) ” 10 = 11 → “ False(偽) ”
不等号(#)	数値 # 数値	ブール	10 # 11 → “ True(真) ” 10 # 10 → “ False(偽) ”
より大きい(>)	数値 > 数値	ブール	11 > 10 → “ True(真) ” 10 > 11 → “ False(偽) ”
より小さい(<)	数値 < 数値	ブール	10 < 11 → “ True(真) ” 11 < 10 → “ False(偽) ”
より大きいまたは等しい(>=)	数値 >= 数値	ブール	11 >= 10 → “ True(真) ” 10 >= 11 → “ False(偽) ”
より小さいまたは等しい(<=)	数値 <= 数値	ブール	10 <= 11 → “ True(真) ” 11 <= 10 → “ False(偽) ”

日付比較演算子

演算子	シンタックス	結果	例
等号(=)	日付 = 日付	ブール	!96.01.01! = !96.01.01! → “ True(真) ” !96.01.20! = !96.01.01! → “ False(偽) ”
不等号(#)	日付 # 日付	ブール	!96.01.20! # !96.01.01! → “ True(真) ” !96.01.01! # !96.01.01! → “ False(偽) ”
より大きい(>)	日付 > 日付	ブール	!96.01.20! > !96.01.01! → “ True(真) ” !96.01.01! > !96.01.01! → “ False(偽) ”
より小さい(<)	日付 < 日付	ブール	!96.01.01! < !96.01.20! → “ True(真) ” !96.01.01! < !96.01.01! → “ False(偽) ”
より大きいまたは等しい(>=)	日付 >= 日付	ブール	!96.01.20! >= !96.01.01! → “ True(真) ” !96.01.01! >= !96.01.20! → “ False(偽) ”
より小さいまたは等しい(<=)	日付 <= 日付	ブール	!96.01.20! <= !96.01.20! → “ True(真) ” !96.01.20! <= !96.01.01! → “ False(偽) ”

時間比較演算子

演算子	シンタックス	結果	例
等号(=)	時間 = 時間	ブール	?01.02.03? = ?01.02.03? → “ True(真) ” ?01.02.03? = ?01.02.04? → “ False(偽) ”
不等号(#)	時間 # 時間	ブール	?01.02.03? # ?01.02.04? → “ True(真) ” ?01.02.04? # ?01.02.04? → “ False(偽) ”
より大きい(>)	時間 > 時間	ブール	?01.02.04? > ?01.02.03? → “ True(真) ” ?01.02.03? > ?01.02.03? → “ False(偽) ”
より小さい(<)	時間 < 時間	ブール	?01.02.03? < ?01.02.04? → “ True(真) ” ?01.02.03? < ?01.02.03? → “ False(偽) ”
より大きいまたは等しい(>=)	時間 >= 時間	ブール	?01.02.03? >= ?01.02.03? → “ True(真) ” ?01.02.03?>= ?01.02.04? → “ False(偽) ”
より小さいまたは等しい(<=)	時間 <= 時間	ブール	?01.02.03? <= ?01.02.03? → “ True(真) ” ?01.02.04? <= ?01.02.03? → “ False(偽) ”

ポインタ比較演算子

演算子	シンタックス	結果	例
等号(=)	ポインタ = ポインタ	ブール	(->Object) = (->Object) → “ True(真) ” (->Object1) = (->Object2) → “ False(偽) ”
不等号(#)	ポインタ # ポインタ	ブール	(->Object1) # (->Object2) → “ True(真) ” (->Object) # (->Object) → “ False(偽) ”

論理演算子

4th Dimensionは、論理積(&)と論理和(|)をサポートしています。演算子は、両方ともブールとして機能します。“ & ” は両方の式が “ 真 ” である場合に “ 真 ” を返します。“ | ” は少なくとも一方の式が “ 真 ” のときに “ 真 ” を返します。

演算子	シンタックス	結果	例
論理積 (AND)	ブール & ブール	ブール	("A"="A") & (15 # 3) → “ True(真) ” ("A"="B") & (15 # 3) → “ False(偽) ” ("A"="B") & (15 = 3) → “ False(偽) ”
論理和 (OR)	ブール | ブール	ブール	("A"="A") | (15 # 3) → “ True(真) ” ("A"="B") | (15 # 3) → “ True(真) ” ("A"="B") | (15 = 3) → “ False(偽) ”

また4th Dimensionは、ブール関数に**True**、**False**、**Not**もあります。この関数の詳細は、第16章を参照してください。

次の表に、論理演算子(&)の真偽表を示します。真偽表は、演算子に対する2つの値を与え、それぞれのケースに対する結果を示します。

式１	式２	式１ & 式２
True(真)	True(真)	True(真)
True(真)	False(偽)	False(偽)
False(偽)	True(真)	False(偽)
False(偽)	False(偽)	False(偽)

次の図に、論理演算子(|)の真偽表を示します。

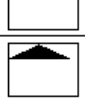
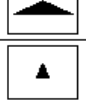
式１	式２	式１ | 式２
True(真)	True(真)	True(真)
True(真)	False(偽)	True(真)
False(偽)	True(真)	True(真)
False(偽)	False(偽)	False(偽)

ピクチャ演算子

次の表は、4th Dimensionのピクチャ演算子を要約したものです。

演算子	シンタックス	動作
水平連結(+)	ピクチャ1 + ピクチャ2	ピクチャ2をピクチャ1の右側に移動する
垂直連結(/)	ピクチャ1 / ピクチャ2	ピクチャ2をピクチャ1の下に移動する
XOR重ね(&)	ピクチャ1 & ピクチャ2	ピクチャ1とピクチャ2の排他的ORを実行する
OR重ね(|)	ピクチャ1 | ピクチャ2	ピクチャ2の上にピクチャ1を重ねる
水平移動(+)	ピクチャ + 数値	ピクチャを指定ピクセル分、横に移動する
垂直移動(/)	ピクチャ / 数値	ピクチャを指定ピクセル分、縦に移動する
サイズ変更(*)	ピクチャ * 数値	割合によってピクチャのサイズを変更する
水平スケーリング(*+)	ピクチャ *+ 数値	割合によって水平にピクチャサイズを変更する
垂直スケーリング(*/)	ピクチャ */ 数値	割合によって垂直にピクチャサイズを変更する

次の表はピクチャ演算子の例を表しています。結果はトランケート(中央合わせ)とバックグラウンドの両方で示されています。

演算子	例	ピクチャ	バックグラウンド	トランケート
水平移動	ピクチャ1 + ピクチャ2			
垂直連結	ピクチャ1 | ピクチャ2			
XOR重ね	ピクチャ1 & ピクチャ2			
OR重ね	ピクチャ1 | ピクチャ2			
右へ水平移動	ピクチャ1 + 5			
左へ水平移動	ピクチャ1 + (-5)			
下へ垂直移動	ピクチャ1 / 5			
上へ垂直移動	ピクチャ1 / (-5)			
サイズを拡大	ピクチャ * 2			
サイズを縮小	ピクチャ * 0.5			
水平スケール拡大	ピクチャ *+ 2			
水平スケール縮小	ピクチャ *+ 0.5			
垂直スケール拡大	ピクチャ */ 2			
垂直スケール縮小	ピクチャ */ 0.5			

比較演算子は、ピクチャでは使用することはできません。ピクチャフィールドやピクチャ変数は、0を掛けることにより空にすることができます。

例えば、このように記述します：

vピクチャ:=vピクチャ*0

ビットワイズ演算子

ビットワイズ演算子は、倍長整数式また倍長整数値の演算を行います。

注：ビットワイズ演算子に整数値または実数値を渡す場合、4th Dimensionは、値をロング整数値として評価してから、ビットワイズ演算子を使用した式を計算します。

ビットワイズ演算子を使用する場合、倍長整数値を32ビットの配列と考える必要があります。これらのビットには、右から左に0～31の番号が付けられます。

それぞれのビットは0か1なので、倍長整数値は32の論理値を格納できる値と考えることもできます。1に等しいビットはTrue、0に等しいビットはFalseを意味します。

ビットワイズ演算子を使用する式は倍長整数値を返しますが、「Bitテスト」演算子は例外的に式がブール値を返します。次の表にビットワイズ演算子とそのシンタックスを示します：

演算	演算子	シンタックス	結果
Bitwise AND	&	倍長整数 & 倍長整数	倍長整数
Bitwise OR (包合)	|	倍長整数 | 倍長整数	倍長整数
Bitwise OR (排他)	^|	倍長整数 ^| 倍長整数	倍長整数
Left Bit Shift	<<	倍長整数 << 倍長整数	倍長整数 (備考1)
Right Bit Shift	>>	倍長整数 >> 倍長整数	倍長整数 (備考1)
Bitセットt	?+	倍長整数 ?+ 倍長整数	倍長整数 (備考2)
Bit消去	?-	倍長整数 ?- 倍長整数	倍長整数 (備考2))
Bitテスト	??	倍長整数 ?? 倍長整数	ブール (備考2))

備考：
(1) 「Left Bit Shift」および「Right Bit Shift」演算では、2番目のオペランドは、結果値において1番目のオペランドのビットがシフトされるビット数を示します。従って、この2番目のオペランドは、0～32の間でなければなりません。0ビットシフトするとその値がそのまま返されます。また、31ビット以上シフトするとすべてのビットがなくなるので、0x00000000が返されます。もう1つの値を2番目のオペランドとして渡すと、結果は符号無しになります。

(2) 「Bitセット」、「Bit消去」、「Bitテスト」演算では、2番目のオペランドは、作用の対象となるビット番号を示します。従って、この2番目のオペランドは0～31の間です。そうでない場合、「Bitセット」と「Bit消去」に対しては1番目のオペランドがそのまま返され、「Bitテスト」に対してはFalseが返されます。

次の表は、ビットワイズ演算子とその効果を示します：

演算子	説明
Bitwise AND	それぞれの結果ビットは2つのオペランドのビットの論理ANDです。 下記は、論理ANDの真偽表です。 1 & 1 → 1 0 & 1 → 0 1 & 0 → 0 0 & 0 → 0 言い換えると、両方のオペランドのビットが1の場合、結果ビットが1になり、その他の場合は結果ビットが0になります。
Bitwise OR (包含)	それぞれの結果ビットは2つのオペランドのビットの論理ORです。 下記は、論理ORの真偽表です。 1 | 1 → 1 0 | 1 → 1 1 | 0 → 1 0 | 0 → 0 言い換えると、いずれかのオペランドのビットが1の場合、 結果ビットが1になり、その他の場合は結果ビットが0になります。
Bitwise OR (排他)	それぞれの結果ビットは2つのオペランドのビットの論理XORです。 下記は、論理XORの真偽表です。 1 ^| 1 → 0 0 ^| 1 → 1 1 ^| 0 → 1 0 ^| 0 → 0 言い換えると、オペランドのビットのいずれか一方だけが1の場合 結果ビットが1になり、その他の場合は結果ビットが0になります。
Left Bit Shift	結果値が最初のオペランド値に設定され、次に結果ビットが2番目のオペランドで示されたビット数だけ左にシフトします。 一番左のビットがなくなり、一番右のビットが0に設定されます。 **注：正の数だけを考えると、Nビット左にシフトすることは、2^Nを掛けることと同じです。**
Right Bit Shift	結果値が最初のオペランド値に設定され、次に結果ビットが2番目のオペランドで示されたビット数だけ右にシフトします。 一番右のビットがなくなり、一番左のビットが0に設定されます。 **注：正の数だけを考えると、Nビット左にシフトすることは、2^Nで割ることと同じです。**

Bitセット 結果値が最初のオペランド値に設定され、次に結果ビットのうち2番目のオペランドで示されたビットが1に設定されます。他のビットはそのままです。

Bit消去 結果値が最初のオペランド値に設定され、次に結果ビットのうち2番目のオペランドで示されたビットが0に設定されます。他のビットはそのままです。

Bitテスト 最初のオペランドのうち、2番目のビットで示されたビットが1の場合、Trueが返されます。最初のオペランドのうち、2番目のビットで示されたビットが0の場合、Falseが返されます。

例：

(1) 次の表にそれぞれのビット演算子の例を示します。

演算子	例	結果
Bitwise AND	0x0000FFFF & 0xFF00FF00	0x0000FF00
Bitwise OR (包合)	0x0000FFFF | 0xFF00FF00	0xFF00FFFF
Bitwise OR (排他)	0x0000FFFF & 0xFF00FF00	0xFF0000FF
Left Bit Shift	0x0000FFFF << 8	0x000FFFF0
Right Bit Shift	0x0000FFFF >> 8	0x00000FFF
Bitセット	0x00000000 ?+16	0x00010000
Bit消去	0x00010000 ?-16	0x00000000
Bitテスト	0x00010000 ?? 16	True

(2) 4th Dimensionのバージョン6では、多くの定義済み定数を提供します。これらの定数の中には、そのリテラルが"bit"または"mask"で終わるものがあります。例えば、「Resources properties」テーマで提供される定数の場合を以下に示します。

定数	タイプ	値
System heap resource mask	倍長整数	64
System heap resource bit	倍長整数	6
Purgeable resource mask	倍長整数	32
Purgeable resource bit	倍長整数	5
Locked resource mask	倍長整数	16
Locked resource bit	倍長整数	4
Protected resource mask	倍長整数	8
Protected resource bit	倍長整数	3
Preloaded resource mask	倍長整数	4
Preloaded resource bit	倍長整数	2
Changed resource mask	倍長整数	2
Changed resource bit	倍長整数	1

これらの定数により、**Get resource properties**関数が返した値をテストしたり、**SET RESOURCE PROPERTIES**コマンドに渡す値を作成したりすることができます。リテラルが"bit"で終わる定数は、テスト、消去またはセットするビットの位置を指定します。リテラルが"mask"で終わる定数は、テスト、消去またはセットするビットが1の場合のみ、倍長整数値を指定します。

例えば、(プロパティが変数$vlResAttrで取得された)リソースがパージ可能かどうかをテストするには、以下のように記述します。

If ($vlResAttr ?? Purgeable resource bit) `リソースはパージ可能か?

または

If (($vlResAttr & Purgeable resource mask) # 0) リソースはパージ可能か?

逆に、これらの定数を使用して同じビットを設定することもできます。以下のように記述します。

$vlResAttr:=$vlResAttr ?+ Purgeable resource bit

または

$vlResAttr:=$vlResAttr | Purgeable resource mask

(3) この例では、2つの整数値を倍長整数値に格納します。以下のように記述します。

$vlLong:=($viIntA<<16) | $viIntB ` 2つの整数を倍長整数に格納します。
$vlIntA:=$vlLong>>16 ` ハイワードに格納されている整数を抽出し直します。
$viIntB:=$vlLong & 0xFFFF ` ローワードに格納されている整数を抽出し直します。

Tips：倍長整数または整数の値を数値とビットワイズ演算子を組み合わせた式で操作する場合は注意が必要です。ハイビット(倍長整数の場合はビット31、整数の場合はビット15)は、その値が消去された場合には正の数で、値が設定された場合には負の数です。数値演算子はこのビットを使用して、値の符号を検出します。ビットワイズ演算子はこのビットの意味を無視します。

名前規則の要約

次の表は、4th Dimensionの名前規則の要約を示しています。

タイプ	長さ	例
テーブル	３１	[テーブル1]
フィールド	３１	[テーブル1]フィールド１
サブテーブル	３１	[テーブル1]サブファイル
サブフィールド	３１	[テーブル1]サブファイル'サブフィールド
インタープロセス変数	<> + ３１	<>インタープロセス変数
プロセス変数	３１	変数
ローカル変数	$ + ３１	$ローカル変数
フォーム	３１	"フォーム"
インタープロセス配列	<> + ３１	<>インタープロセス配列
プロセス配列	３１	プロセス配列
ローカル配列	$ + ３１	$ローカル配列
メソッド	３１	*メソッド*
プラグインルーチン	３１	*CT GET DOCUMENT SIZE*
インタープロセスセット	<> + ８０	"<>インタープロセスセット"
プロセスセット	８０	"プロセスセット"
クライアントセット	$ + ８０	"$クライアントセット"
ローカルプロセス	$ + ３１	$ローカルプロセス
グローバルプロセス	３１	グローバルプロセス
命名セレクション	８０	"命名セレクション"
インタープロセス命名セレクション	<> + ８０	"<>インター命名セレクション"

この章では、コマンドの説明に使用する表記方法について説明します。また、コマンドに引数を渡す際の規則についても説明します。

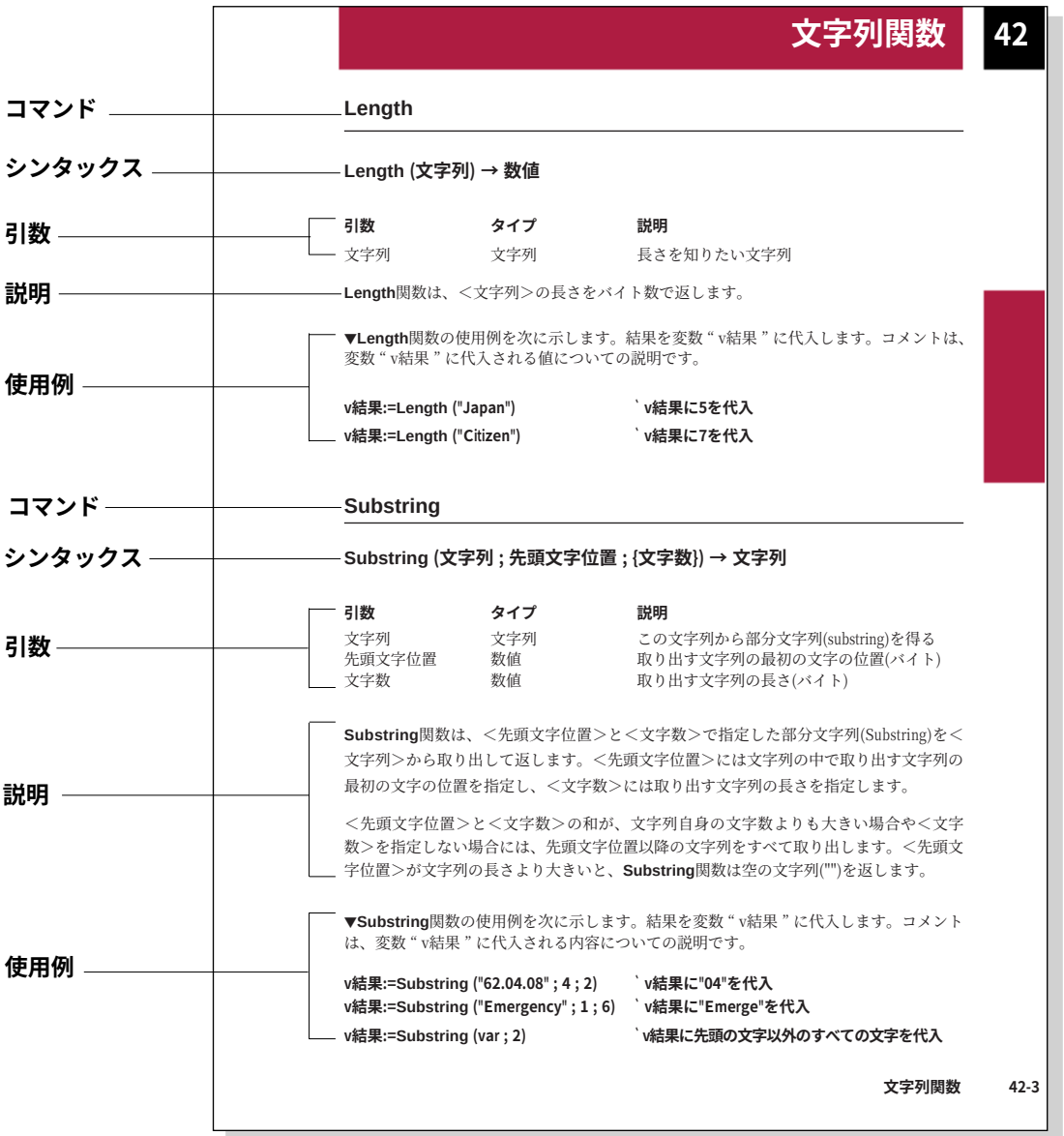
コマンド定義

コマンドの説明は、次の5つの部分で構成されています：

- 見出し
- シンタックス
- 引数
- 機能説明
- 使用例

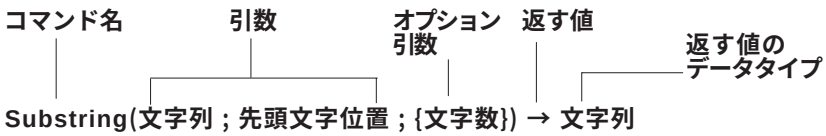
また、コマンドの説明には、警告、注釈、および4D Serverでのコマンドの動作に関する説明も含まれています。

次の図は、コマンド定義を示しています：



シンタックス（構文）

シンタックスは、コマンドの文法です。各コマンドの最初には、コマンドの文法を示します。コマンド名の後には引数が続きます。コマンドが関数である場合、引数の後には矢印が続き、コマンドが返す値のデータタイプを示します。次の図に、シンタックスダイアグラムの例を示します。



引数（パラメータ）

引数（パラメータ）は、コマンドに渡すデータで、“ <> ” で囲みます。

引数表は、表形式で各引数のデータタイプと短い説明文を併記します。次の図に、引数の記述方法を示すシンタックスダイアグラムと引数表を示します。

Substring (文字列 ; 先頭文字位置 ; {文字数}) → 文字列

引数	タイプ	説明
文字列	文字列	この文字列から部分文字列(substring)を得る
先頭文字位置	数値	取り出す文字列の最初の文字の位置(バイト)
文字数	数値	取り出す文字列の長さ(バイト)

引数がオプションの場合は、引数の名前を、シンタックスダイアグラム中で中括弧({...}) で囲みます。（先ほどのシンタックスダイアグラムの “ 文字数 ” 引数を参照してください。）

引数を繰り返し使用する場合(常にオプション)は、引数の後に省略記号(...)つけて示します。省略記号の後に、引数名と引数が繰り返すことのできる回数を示す数字をつけます。

引数が無限に繰り返すことができる場合、数字はNになります。例えば、次のようになります。

SAVE VARIABLES (文書ファイル ; 変数1{;...; 変数N})

引数（パラメータ）の指定

引数（パラメータ）をコマンドに対して指定する際に、守らなければならない規則がいくつかあります。その規則を次に示します：

■ 引数は、括弧で囲みます。例えば、**ADD RECORD**コマンドに[私のテーブル]を引数として指定する場合は、次のように記述します：

ADD RECORD ([私のテーブル])

■ コマンドが複数の引数を持つ場合は、引数間をセミコロン(;)で区切ります。引数を繰り返す場合も同様です。例えば、**INPUT FORM**コマンドに[[私のテーブル]と"入力フォーム"を引数として指定する場合は、次のように記述します：

INPUT FORM([私のテーブル] ; "入力フォーム")

■ オプションの引数を省略する場合は、付属のセミコロンも省略します。例えば、**Request** 関数に対する2番目の引数はオプションです。2番目の引数を指定しない場合は、次のように記述します：

Result := Request (文字列1)

2番目の引数を指定する場合は2つの引数間をセミコロン(;)で区切り、次のように記述します：

Result := Request (文字列1 ; 文字列2)

■ 引数がない場合は、括弧も省略します。引数を持たないコマンドも引数を省略したコマンドも同様です。例えば、**ADD RECORD**コマンドの引数に[テーブル]を指定する場合は、次のように記述します：

ADD RECORD ([テーブル])

テーブルの引数を省略した場合には、括弧も省略し、次のように記述します：

ADD RECORD

4D環境コマンド　13

この章では、「ルーチン」エディタの「4D Environment」テーマ内にある4D環境コマンドについて説明します。

QUIT 4D
FLUSH BUFFERS
SELECT LOG FILE
ADD DATA SEGMENT
PLATFORM PROPERTIES
ACI folder
Structure file
Data file
Application file
Compiled application
Application version
Application type
Version Type
DATA SEGMENT LIST

QUIT 4D

QUIT 4D

引数　　タイプ　　説明

このコマンドには、引数はありません。

QUIT 4Dコマンドは、4th Dimensionを終了してデスクトップに戻ります。**QUIT 4D**コマンドを呼び出すと、カレントプロセスはそれ自身の実行を中止して、4th Dimensionは次の動作を行います：

■「On Exit」データベースメソッドがある場合には、4Dは新しく作成されたローカルプロセス内でこのメソッドの実行を開始します。従って、このデータベースメソッドを使用して、インタープロセス間通信を通じて、(データ入力を)終了したり、処理の実行を中止したりしなければならないことを、他のプロセスに通知することができます。4Dは、いずれ終了するということに注意が必要です。「On Exit」データベースメソッドは、必要なクリーンアップや終了の処理を実行することができますが、中断処理を拒否することができず、ある時点で終了することになります。

■「On Exit」データベースメソッドがない場合には、4Dは実行プロセスそれぞれを区別せずに1つずつアボートします。ユーザがデータ入力を実行している場合には、レコードはキャンセルされ、保存されません。

現在開いているウインドウ上で行われたデータ入力の変更内容をユーザに保存させたい場合は、インタープロセス間通信を使って、データベースが終了されようとしているプロセスをその他のすべてのユーザに合図することができます。これを行うには、次の2つの方法があります：

■ **QUIT 4D**コマンドを呼び出す前にカレントプロセスの中から上記の操作を実行する

■「On Exit」データベースメソッドの中から上記の操作を取り扱う

3番目の方法もあります。**QUIT 4D**コマンドを呼び出す前に、ウインドウが妥当性検査を必要とするかどうかをチェックします。もし、このケースの場合は、ユーザにこのウインドウを有効にするか、または取り消すかを尋ね、それから再度、「終了」を選択するか尋ねます。しかし、ユーザインタフェースの観点から見ると、最初の2つの方法を使用するのをお勧めします。

▼次の例は、「ファイル」メニューに「終了」メニューコマンドを持ったプロジェクトメソッドを示しています：

```
`「M_終了」プロジェクトメソッド
CONFIRM ("終了してもよろしいですか？")
If (OK=1)
   QUIT 4D
End if
```

FLUSH BUFFERS

FLUSH BUFFERS

引数	タイプ	説明

このコマンドには、引数はありません。

FLUSH BUFFERSコマンドを実行すると、データバッファの内容を即座にディスクに保存します。データベースへのすべての変更をディスクに保存します。

通常、4th Dimensionがデータの変更内容を保存するので、このコマンドを呼び出す必要はありません。「デザイン」モードの「データベースプロパティ」ダイアログボックス内にある「データ保存：」プロパティで、通常使用の場合にデータバッファの内容を保存する間隔を指定できます。

SELECT LOG FILE

SELECT LOG FILE ({ログファイル名})
SELECT LOG FILE ({"*"})

引数	タイプ	説明
ログファイル名	文字列	ログファイルの名前

このコマンドは、引数＜ログファイル名＞で指定されたログファイルを開きます。

＜ログファイル名＞が空の文字列の場合、**SELECT LOG FILE**コマンドは「ファイルオープン」ダイアログボックスが表示され、ログファイルを開くことができます。ユーザが「開く」ボタンをクリックし、ログファイルが正常に開かれると、システム変数OKに1が設定されます。それ以外は0が設定されます。また、ユーザが「キャンセル」ボタンをクリックした場合は、システム変数OKに0が設定されます。

このコマンドは、「ユーザ」モードの「ファイル」メニューから「ログファイル...」アイテムを選択するのと同じです。これはデータベースに4D Backupモジュールがインストールされている場合にのみ使用することができます。

＜ログファイル名＞が“*”の場合、**SELECT LOG FILE**コマンドはカレントログファイルを閉じます。ログファイルが閉じられると、システム変数OKに0が設定されます。

フルバックアップがまだ行われていない場合やデータファイルにすでにレコードが含まれている場合に**SELECT LOG FILE**コマンドを使って、ログファイルを作成したり、開いたりすると、エラーコード -4447が返されます。

注：4D Server では、このコマンドは動作しません。このコマンドに関する詳細は、4D Backupモジュールに付属のマニュアルを参照してください。

ADD DATA SEGMENT

このコマンドは、「データセグメント管理」ダイアログボックスを表示します。

「終了」ボタンをクリックしてダイアログボックスを有効にすると、システム変数 OKに1が設定されます。「キャンセル」ボタンをクリックすると、システム変数 OKに 0が設定されます。

注：4D Server では、このコマンドは動作しません。

既存のデータセグメントがすべて満杯になると、4th Dimension または 4D Server にエラーコード -9999 が発生します。そして、ディスクの空きがないことを示すエラーメッセージがユーザに表示されます。

4th Dimension を使っている場合は、エラーをプロシージャで処理できるように **ON ERROR CALL**コマンドを使用して、このエラーメッセージをトラップすることができます。その後で、**ADD DATA SEGMENT**コマンドを使うようにすれば、使用可能な別の空きボリューム上に新しいデータセグメントを追加することができます。

4D Serverを使っている場合、データベース管理者がサーバマシンから新しいデータセグメントを追加する必要があることを示す警告が表示されます。

PLATFORM PROPERTIES

PLATFORM PROPERTIES (プラットフォーム ; システム ; マシン)

引数	タイプ		説明
プラットフォーム	数値	←	1=68KベースのMacintosh 2=Power Macintosh 3=Windows
システム	数値	←	動作している種類に依存
マシン	数値	→	動作している種類に依存

PLATFORM PROPERTIESコマンドは、実行しているプラットフォーム、システムオペレーション（OS）のバージョン、使用しているマシンにインストールされたプロセッサの種類に関する情報を返します。

PLATFORM PROPERTIESコマンドは、引数＜プラットフォーム＞、＜システム＞、＜マシン＞に4D環境の情報を返します。

＜プラットフォーム＞は、ユーザが実行している4th Dimensionのバージョンが68KのMacintoshなのか、それともPower Macintosh、またはWindowsなのかを示します。この引数は、次のようにあらかじめ定義されている定数の1つを返します。

定数	タイプ	値
Macintosh 68K	倍長整数	1
Power Macintosh	倍長整数	2
Windows	倍長整数	3

＜システム＞と＜マシン＞の中に返される情報は、動作している4th Dimensionの種類によって異なります。

Macintosh （68KとPowerPC）

MacOS バージョンの 4th Dimension を実行している場合、残りの 2 つの引数には次の情報が返されます。

■ 引数＜システム＞には 32ビット（倍長整数）値が返され、上位 16ビットは未使用で、下位 16ビットは次のような構造になっています。

上位バイトには、主要なバージョン番号が入っています。

下位バイトは、 2 つの部分（各 4ビット）に分かれています。上位部分はアップデートの主要なバージョン番号で、下位部分はアップデート番号の枝番です。例えば：
System 7.5.1 は $0751 のようにコード化されるので、10進数では 1873 になります。

注：4D で、これらの値は %（モジューロ）と //（整数値を返す除算）を使った数値計算で求められます。また、バージョン6から追加された「ビットワイズ」演算子を使ってもできます。

■ 引数＜マシン＞には、Macintoshのモデルを示す一意 (重複しない) ID番号が返されます。

注：このID番号は、Appleコンピュータ社から発行されているデベロッパまたはテクニカルドキュメントに記載されています。新しいモデルの Macintosh がリリースされた場合は、新たに値を追加する必要があります。

Windowsバージョン

Windowsバージョンの 4th Dimension を実行している場合、引数＜システム＞と＜マシン＞には、次で説明する情報が返されます。

■ 引数＜システム＞からは 32ビット（倍長整数）値が返され、そのビットとバイトは次のような構造になっています：

上位レベルのビットが 1 になっている場合、Windows NT の特色を持つシステムで実行していることを示します。ビットが 0 になっている場合は、Windows 3.1 または Windows 95 のもとで実行しているということを意味します。

注：上位レベルのビットは倍長整数の符号に使われています。従って 4Dでは、値が正か負かを調べるだけです。正ならWindows NTが動作してることになります。

下位バイトには、Windows の主要なバージョン番号が入っています。3 が返された場合、Windows バージョン 3.x もしくは Windows NTが動作中であることを示します。4が返された場合、Windows 95 または新しいシェルで Windows NT が動作中であることを示します。次の下位バイトには Windows のバージョン番号の枝番が入っています。

注：4D で、これらの値は %（モジューロ）と //（整数値を返す除算）を利用した数値計算で求められます。また、バージョン6から追加された「ビットワイズ」演算子を使ってもできます。

■ 引数＜マシン＞は、次のようにあらかじめ定義されている定数の1つを返します。

定数	タイプ	値
IINTEL 386	倍長整数	386
INTEL 486	倍長整数	486
Pentium	倍長整数	586
PowerPc 601	倍長整数	601
PowerPc 603	倍長整数	603
PowerPc 604	倍長整数	604

▼ 次のプロジェクトメソッドは、使用しているOSを示した「アラート」ボックスを表示します：

```
` 「SHOW OS VERSION」プロジェクトメソッド
PLATFORM PROPERTIES($vlPlatform ; $vlSystem ; $vlMachine)
If (($vlPlatform<1)  |  (3<$vlPlatform))
   $vsPlatformOS:=""
Else
   If ($vlPlatform=3)
      $vsPlatformOS:=""
      If ($vlSystem<0)
         $winMajVers:=((2^31)+$vlSystem)%256
         $winMinVers:=(((2^31)+$vlSystem)\256)%256
         If ($winMajVers>=4)
            $vsPlatformOS:="Windows™ 95"
         Else
            $vsPlatformOS:="Windows™ (with Win32s)"
         End if
      Else
         $winMajVers:=$vlSystem%256
         $winMinVers:=($vlSystem//256)%256
         $vsPlatformOS:="Windows™ NT"
      End if
      $vsPlatformOS:=$vsPlatformOS+" バージョン"+String($winMajVers)+"."
      +String($winMinVers)
   Else
      $vsPlatformOS:="MacOS™バージョン "+String($vlSystem//256)+"."
` +String(($vlSystem//16)%16)+(("."+String($vlSystem%16))*Num(($vlSystem%16)  #  0))
   End if
End if
ALERT($vsPlatformOS)
```

Windows上では、次の図のような「アラート」ボックスを表示します。

Macintosh上では、次の図のような「アラート」ボックスを表示します。

ACI folder

ACI folder → 文字列

引数	タイプ		説明
このコマンドには、引数はありません。			
関数の返す値	文字列	←	ACIフォルダのパス名

ACI folder関数は、稼働しているシステムフォルダまたはディレクトリの中に配置されたACIフォルダのパス名を返します。

Windows上：

“ ACI ”と名付けられた「ACI」ディレクトリ（フォルダ）は、稼働している「WINDOWS」ディレクトリ（通常、**C:\ WINDOWS**）の中に配置されます。そういうわけで、**ACI folder**関数は通常、**「C:\ WINDOWS \ACI」**パス名を返します。しかし、PCコンピュータはマルチブートコンフィギュレーション（システム定義）を使って、セットアップを行うことができ、配置場所と稼働中のWINDOWSディレクトリの名前をインストール作業中にカスタマイズすることができます。そのため、もし、ACIフォルダの中にユーザ自身のファイル（書類）を保存したい場合、**ACI folder**関数はこのフォルダへの実際のパス名を取得することができます。

プラットフォーム独立性と国際性：そのフォルダへの実際のパス名を取得するために**ACI folder**関数を使用することにより、あるローカライズされたシステムで動作している任意のプラットフォーム上での機能を保証します。

4D環境は、次の情報を格納するために「ACI」フォルダを使用します。

■ ユーザレジストレーションファイル

■ 4D環境アプリケーション、ツール、ユーティリティプログラムで使用される初期設定（Preference）ファイル

■ 4D Client／4D Serverまたはインターネット／イントラネットのネットワークコンポーネント（ただし、Windows上では...\ACI\NETWORKディレクトリの中）

■ 4D Serverからダウンロードされたリソースを格納するために4D Clientによって作成された「.rex」ファイルと「.res」ファイル

■ 4D Serverからダウンロードされたリソースを格納するために4D Clientによって作成されたローカルデータベース

警告：ACIフォルダの中にどんなファイルまたは書類を格納するのは自由ですが、4D環境自身によって作成されたファイルの移動および変更は避けた方が賢明です。

▼ 次の例は、シングルユーザデータベースの起動中に、ACIフォルダの中に配置されたファイル内にユーザ自身が設定した内容をロードしたいとします。これを行うには、「On Startup」データベースメソッド内に次のようなコードを記述します：

```
MAP FILE TYPES("PREF" ; "PRF" ; "初期設定ファイル")
` 「PREF」MacOSファイルタイプを「PRF」Windowsファイル拡張子にマップする
" $vsPrefDocName:=ACI folder+"MyPrefs" `初期設定ファイルにパス名を作成する
` そのファイルが存在するかチェックする
If (Test pathname($vsPrefDocName+(".PRF"*Num( On Windows)))#Is a document)
   $vtPrefDocRef:=Create document($vsPrefDocName ; "PREF") ` ファイルを作成する
Else
   $vtPrefDocRef:=Open document($vsPrefDocName ; "PREF") `ファイルを開く
End if
If (OK=1)
   `文書内容を処理する
   CLOSE DOCUMENT($vtPrefDocRef)
Else
   `エラーの取り扱い
End if
```

参照：System folder、Temporary folder、Test path name

Structure file

Structure file → 文字列

引数	タイプ		説明
このコマンドには、引数はありません。			
関数の返す値	文字列	←	データベースストラクチャファイルのロング名

Structure file関数は、現在使用しているデータベースのストラクチャファイルのロング名を返します。

Windows上：

例えば、ボリュームG上の「DOCS\MyCD」の中に配置されたデータベースを使って作業している場合、この関数は、「**G:\DOCS\MyCD\myCD.4DB**」を返します。

Macintosh上：

例えば、ハードディスク「Macintosh HD」上の「MyCD」フォルダの中に配置されたデータベースを使って作業している場合、この関数は、「**Macintosh HD:MyCD f:My CD**」を返します。

警告：4D Clientを実行している最中にこの関数を呼び出すと、ロング名ではなくストラクチャファイルの名前のみが返されます。

▼ 次の例は、現在使用中のスタイルシートファイルの名前と配置場所を表示します：

```
If (Application type#4D Client)
   $vsStructureFilename:= Long name to file name (Structure file)
   $vsStructurePathname:= Long name to path name (Structure file)
   ALERT("現在下記のデータベースを使用しています"+Char(34)+$vsStructureFilename
         +Char(34)+" located at "+Char(34)+$vsStructurePathname+Char(34)+".")
Else
   ALERT("次のデータベースに接続されます："+Char(34)+Structure file+Char(34))
End if
```

注：プロジェクトメソッド「Long name to file name」と「Long name to path name」は、第52章の「システム文書コマンド」の節で説明されています。

参照：Application file、Data file、DATA SEGMENT LIST

Data file

Data file ({セグメント]) → 文字列

引数	タイプ		説明
セグメント	数値	→	セグメント番号
関数の返す値	文字列	←	データベースのデータファイルのロング名

Data file関数は、現在使用しているデータベースのデータファイルのロング名、または現在使用しているデータベースの1つのデータセグメントを返します。

オプション引数＜セグメント＞を指定しない場合、**Data file**関数はそのデータファイルまたは1番目のセグメント（データベースが分割されている場合）のロング名を返します。＜セグメント＞を指定した場合、**Data file**関数は対応しているデータセグメントのロング名を返します。データセグメントの数よりも大きいセグメント番号を指定すると、**Data file**関数は空の文字列を返します。

Windows上：

例えば、ボリュームG上の「DOCS\MyCD」の中に配置されたデータベースを使って作業している場合、この関数は、「**G:\DOCS\MyCD\myCD.4DD**」を返します。

Macintosh上：

例えば、ハードディスク「Macintosh HD」上の「MyCD」フォルダの中に配置されたデータベースを使って作業している場合、この関数は、「**Macintosh HD:MyCD f:My CD.data**」を返します。

警告：4D Clientを実行している最中にこの関数を呼び出すと、ロング名ではなくストラクチャファイルの名前のみが返されます。また、たとえデータベースが分割されていようとも、Data file関数は他のデータセグメントに対して空の文字列を返します。もし、4D Client上にデータセグメントの一覧を表示したい場合、ストアドプロシージャを使ってデータセグメントリストを作成し、サーバマシン上の任意の変数内にそのリストを格納します。そして、GET PROCESS VARIABLEコマンドを使って、この変数の内容を取得します。

▼ 次の例は、データベースのデータベースセグメントを調べます：

```
If (Application type#4D Client)
   $vlDataSegNum:=0
   Repeat
      $vlDataSegNum:=$vlDataSegNum+1
      $vsDataSegName:=Data file($vlDataSegNum)
      If ($vsDataSegName#"")
         ALERT ("データセグメント"+String($vlDataSegNum)+":"+Char(34)+
         $vsDataSegName+Char(34)+".")
      End if
   Until ($vsDataSegName="")
   ALERT(String($vlDataSegNum-1)+"個のデータセグメントがあります。")
End if
```

参照：Application file、Data file、DATA SEGMENT LIST

Application file

Application file → 文字列

引数	タイプ		説明
このコマンドには、引数はありません。			
関数の返す値	文字列	←	4D実行可能ファイルまたはアプリケーションのロング名

Application file関数は、現在使用しているデ4D実行可能ファイルまたはアプリケーションのロング名を返します。

Windows上：

例えば、ボリュームE上の「**4DWIN600\PROGRAM**」の中にに配置された4th Dimensionを使用している場合、この関数は、「**E:\4DWIN600\PROGRAM\4D,EXE**」を返します。

Macintosh上：

例えば、ハードディスク「Macintosh HD」上の「4th Dimension® 6.0 *f*」フォルダの中にある4th Dimensionを使用している場合、この関数は、「Macintosh HD:4th Dimension® 6.0 *f* :4th Dimension® 6.0」を返します。

▼ 次の例は、Windowsの起動時に、任意のDLLライブラリが4D実行可能ファイルと同じ階層に配置されているかどうかをチェックします。「On Startup」データベースメソッドの中に次のコードを記述します：

```
If ( On Windows & (Application type#4D Server))
   If (Test path name ( Long name to path name (Application file)
      +"XRAYCAPT.DLL")#Is a document)
      `「XRAYCAPT.DLL」ライブラリが見つからないことを説明したダイアログボックス
        を表示する。そのため、放射線照射機能を利用できない
   End if
End if
```

注：プロジェクトメソッド「On Windows」と「Long name to path name」は、第52章の「システム文書コマンド」の節で説明されています。

参照：Data file、DATA SEGMENT LIST、Structure file

Compiled application

Compiled application → 文字列

引数	タイプ		説明
このコマンドには、引数はありません。			
関数の返す値	文字列	←	True=コンパイル、False=インタプリタ

Compiled application関数は、データベースをコンパイルモード（True）またはインタプリタモード（False）のどちらで実行しているかを検査します。

▼ 次の例は、インタプリタモードで実行している場合にのみ意味を持ちます。If文で囲まれた中にデバッグ用のコードを記述します：

```
If (Not(Compiled application))
   ` ここにデバッグ用のコードを記述する
End if
   ` ...
```

参照：IDLE、Undefined

Application version

Application version ({*}) → 文字列

引数	タイプ		説明
セグメント	数値	→	指定した場合、ロングバージョン番号 指定しなかった場合、ショートバージョン番号
関数の返す値	文字列	←	バージョン番号のエンコードされた文字列

Application version関数は、現在使用している4D環境のバージョン番号を表すエンコードされた文字列値を返します。

■ 引数オプション＜*＞を指定した場合、次のようにフォーマットされた4バイト文字列を返します：

キャラクラ	意味
1-2	バージョン番号
3	更新番号
4	訂正版番号

例えば、文字列 “ 0600 ” は、バージョン6.0.0を表します。

■ 引数オプション＜*＞を指定しない場合、次のようにフォーマットされた8バイト文字列を返します：

キャラクラ	意味
1	"F"は最終版を意味する "B"はベータ版を意味する それ以外の文字は、ACIの内部バージョンを意味する
2-3-4	ACIの内部コンパイル番号
5-6	バージョン番号
7	更新番号
8	訂正版番号

例えば、文字列 “ B0120602 ” は、バージョン6.0.2のベータ12を表します：

▼ 次の例は、4D環境のバージョン番号を表示します。

```
$vs4Dversion:=Application version
ALERT("現在使用している4Dのバージョン： "+String(Num(Substring($vs4Dversion
        ; 1 ; 2)))+"."+$vs4Dversion[[3]]+"."+$vs4Dversion[[4]])
```

▼ 次の例は、最終版の4Dを使用しているかどうかを検査します：

```
If(Subtring(Application version(*) ; 1 ; 1) # "F")
   ALERT("最終版の4Dでこのデータベースを使用しているかどうか確認してください。")
   QUIT 4D
End if
```

参照：Application type、Version type

Application type

Application type → 倍長整数

引数	タイプ	説明

このコマンドには、引数はありません。

関数の返す値	倍長整数	←	アプリケーションタイプを示す数値

Application type関数は、現在実行している4D環境のアプリケーションタイプを示す数値を返します。4th Dimensionは、次のようにあらかじめ定義された定数を持っています。

定数	タイプ	値
4th Dimension	倍長整数	0
4D Engine	倍長整数	1
4D Runtime	倍長整数	2
4D Runtime Classic	倍長整数	3
4D Client	倍長整数	4
4D Server	倍長整数	5
4D First	倍長整数	6

▼ 次の例は、4D Serverが実行されているかどうかを検査します。次ぎのコードを「On Server Startup」データベースメソッドの中に記述します：

```
If (Application type=4D Server)
   `4D Server上におけるそれ相応の動作を実行する
End if
```

参照：Application version、Version type

Version type

Version type → 倍長整数

引数	タイプ		説明
このコマンドには、引数はありません。			
関数の返す値	倍長整数	←	0=フルバージョン（製品版） 1=デモ版（制限付き）

Version type関数は、現在実行している4D環境のバージョンタイプを示す数値を返します。4th Dimensionは、次のようにあらかじめ定義された定数を持っています。

定数	タイプ	値
Full Version	倍長整数	0
Demo Version	倍長整数	1

▼次の例は、あなたの4Dアプリケーションが4D環境のデモ版が使用される際に利用できないいくつかの機能を含んでいることを示しています：

```
If (Version type=Full Version)
   `何らかの処理を実行する
Else
   ALERT("デモ版では、いくつかの機能を使用することができません。")
End if
```

参照：Application type、Application version

DATA SEGMENT LIST

DATA SEGMENT LIST ({セグメント})

引数	タイプ		説明
セグメント	文字列配列	→	データベースのデータセグメントのロング名

DATA SEGMENT LISTコマンドは、引数<セグメント>配列に現在使用しているデータベースのデータセグメントのロング名を代入します。

警告：このコマンドは、4D Client上では何も行いません。4D Client上にデータセグメントの一覧を表示したい場合、ストアドプロシージャを使ってデータセグメントリストを作成し、サーバマシン上の任意の変数内にそのリストを格納します。そして、GET PROCESS VARIABLEコマンドを使って、この変数の内容を取得します。

▼ 次の例は、[ダイアログ]テーブルの“ データセグメント情報 ”フォームにおいて、データセグメントの名前を持ったドロップダウンリストを表示したいとします。これを行うには、“ データセグメント情報 ”フォームのフォームメソッドに次のコードを記述します：

```
` [ダイアログ];"データセグメント情報" フォームのフォームメソッド
Case of
   \ (Form event=On Load )
      ` ...
     ARRAY STRING(255 ; asDataSegName ; 0)
     DATA SEGMENT LIST(asDataSegName)
      ` ...
End case
```

▼ 次のメソッドは、データベースが分割されているかどうかをユーザに知らせます：

```
` データベースが分割されているかどうか → ブール
C_BOOLEAN ($0)
DATA SEGMENT LIST($asDataSegName)
$0:=(Size of array($asDataSegName)>1)
```

▼ 次の例は、**ADD DATA SEGMENT**コマンドを呼び出した後、新しくセグメントを追加したいかどうかを検査します：

```
DATA SEGMENT LIST(($asBefore)
ADD DATA SEGMENT
DATA SEGMENT LIST($asAfter)
If(Size of array($asBefore) # Size of array($asAfter))
   `データセグメントが多くある場合
Else
   ` セグメントの数が同じ場合
End if
```

参照：Application file、Data file、Structure file

配列コマンド　14

配列とは、同じタイプの変数を番号付きで並べたものです。各変数は、配列の要素といいます。配列のサイズとは、それが持つ要素の数を指します。配列は作成時にサイズが与えられ、要素の追加、挿入、削除によって、またはそれを作成するのに使用したのと同じコマンドを使用して配列のサイズを変更すれば、何度でもサイズを変更することができます。

配列は、配列宣言コマンドのいずれかを使用して作成します。詳細については、次節の「配列を作成する」を参照してください。

要素には、1からNの番号が付けられ、Nは配列のサイズとなります。配列には、常に要素ゼロがあります。この要素は他の要素と同様にアクセスできますが、配列をフォームに表示しても表示されません。要素ゼロは、配列がフォームオブジェクトをサポートする場合には表示されませんが、プログラミング言語で使用する際には制限がありません。要素ゼロについての詳細は、後述の「配列の要素ゼロを使用する」の節を参照してください。

配列は、4Dの変数です。他の変数と同様、配列にもスコープがあり、4D言語の規則に従いますが、他と異なるところがいくつかあります。詳細については、後述の「配列と4D言語」および「配列とポインタ」の節を参照してください。配列は言語のオブジェクトですので、画面に絶対表示されない配列を作成、使用することができます。

配列はまた、ユーザインタフェースオブジェクトでもあります。配列とフォームオブジェクトの間の相互作用についての詳細は、「配列とフォームオブジェクト」および「グループ化されたスクロールエリア」の節を参照してください。配列は短時間に妥当な量のデータを保持するように設計されています。配列はメモリ内に保持されるため、取り扱いやすく、高速に操作できます。詳細については、「配列とメモリ」の節を参照してください。

配列を作成する

配列は、この章で説明する配列宣言コマンドのいずれかを使用して作成します。次の表に、配列宣言コマンドを記載します：

コマンド	配列の作成またはサイズ変更
ARRAY INTEGER	2バイト整数値
ARRAY LONGINT	4バイト整数値
ARRAY REAL	実数値
ARRAY TEXT	テキスト値（1要素当たり0から32000バイト）（参照：備考）
ARRAY STRING	文字列値（1要素当たり0から32000バイト）（参照：備考）
ARRAY DATE	日付値
ARRAY BOOLEAN	ブール値
ARRAY PICTURE	ピクチャ値
ARRAY POINTER	ポインタ値

配列宣言コマンドは、1次元または2次元の配列の作成やサイズ変更が可能です。

2次元配列についての詳細は、「2次元配列」の節を参照してください。

備考：テキスト配列と文字列配列の違いは、要素の性質にあります。どちらのタイプの配列も、要素がテキスト値(文字)を持つことができます。しかし、以下の点が異なります。

■ テキスト配列では、各要素は可変長で、その文字はメモリの異なる部分に格納されます。

■ 文字列配列では、すべての要素が固定長です(配列の作成時に長さが渡されます)。要素はすべて、その内容に関わらずメモリの同じ部分に順番に格納されます。

この構造上の相違のため、文字列配列はテキスト配列より高速に動作します。ただし、文字列配列は最高255バイトしか保持できないので注意してください。

次のコードは、10個の要素からなる整数配列を作成(宣言)します。

```
ARRAY INTEGER(aiAnArray ; 10)
```

次のコードは、同じ配列の要素を20個にします。

```
ARRAY INTEGER(aiAnArray ; 20)
```

次のコードは、同じ配列の要素をゼロにします。

```
ARRAY INTEGER(aiAnArray ; 0)
```

配列中の要素は中カッコ({ })を使用して参照します。中カッコの中には数字を入れて特定の要素を指定します。この数字を要素番号といいます。次の行は、5つの名前をatNamesという配列に入れ、それらを警告ウィンドウに表示します。

```
ARRAY TEXT (atNames ; 5)
atNames{1} := "吉川"
atNames{2} := "伊藤"
atNames{3} := "安藤"
atNames{4} := "田中"
atNames{5} := "中村"
For ($vlElem ; 1 ; 5)
   ALERT ("配列要素 #"+String($vlElem)+"は"+atNames{$vlElem} + "です。")
End for
```

atNames{$vlElem}というシンタックスに注意してください。atNames{3}のような数値そのものを指定するのではなく、数値変数を使用して配列のどの要素を指定するかを示すことができます。ループ構造による反復を使用すると(**For...End for,**、**Repeat...Until(...)**または**While (...) End while**)、短いコードで配列のすべてのまたは一部のコードを指定することができます。

配列および4D言語のその他のエリア

その他に配列で使用できる4Dコマンドがあります。詳細については、次のコマンドの説明を参照してください。

■ 配列とレコードの選択された一部を操作するには、**SUBSELECTION TO ARRAY**、**SELECTION TO ARRAY**、**ARRAY TO SELECTION**、**DISTINCT VALUES**コマンドを使用します。

■ テーブル、サブテーブルおよび配列に格納された値を元にグラフや図を作成することができます。詳細については、**GRAPH**コマンドを参照してください。

■ バージョン6には階層リストを扱う新規コマンドの数多く用意されていますが、それに加えて旧バージョンの**LIST TO ARRAY**コマンドと**ARRAY TO LIST**コマンドが互換性を保つために残されています。

■ バージョン6の新規コマンドは、配列を1度の呼び出しで行います。このようなコマンドには、**FONT LIST**、**WINDOW LIST**、**VOLUME LIST**、**FOLDER LIST**、**DOCUMENT LIST**コマンドがあります。

配列とフォームオブジェクト

配列は、プログラム言語のオブジェクトです。決して画面に表示されない配列を作成、使用することができます。ただし、配列は同時にユーザインタフェースでもあります。配列では、次のタイプのフォームオブジェクトがサポートされます。

- ポップアップメニュー
- ドロップダウンリスト
- コンボリスト
- スクロールエリア
- タブコントロール

これらのオブジェクトは、「デザイン」モードのフォームエディタで(「オブジェクトプロパティ」ウィンドウの「デフォルト値」ボタンを使用して)あらかじめ定義することができますが、配列コマンドを使用してプログラムによって定義することもできます。いずれの場合も、フォームオブジェクトは手動または4Dが作成した配列によってサポートされます。

これらのオブジェクトを使用するとき、その配列で選択された要素を確認することにより、オブジェクト内のどの項目が選択されているかを検出することができます。逆に、選択した要素を配列に設定して、オブジェクトの中の特定の項目を選択することができます。

配列がフォームオブジェクトのサポートに使用される場合、この配列は、ランゲージオブジェクトとユーザインタフェースの2つの性質を併せ持ちます。例えば、フォームを設計する場合、スクロールエリアを作成します。すなわち、「オブジェクトプロパティ」ウィンドウの「変数」ページで変数オブジェクトを指定します。

名前(この例ではatName)は、配列の作成と処理に使用する配列の名前です。

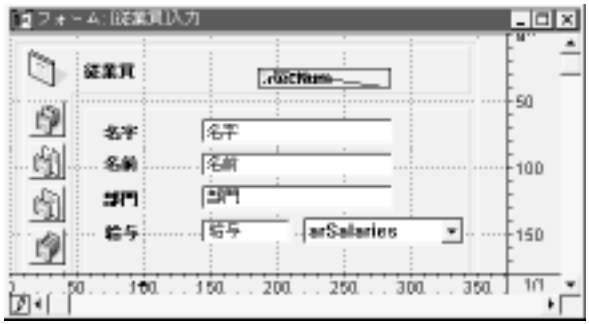
注：2次元配列やポインタ配列は表示できません。

例：ドロップダウンリストを作成する

次の例では、配列を埋めてそれをドロップダウンリストに表示する方法を示します。配列arSalariesは、**ARRAY REAL**コマンドを使用して作成します。これには、社内で社員に支払われる基準給与がすべて入っています。ユーザがドロップダウンリストから要素を選択すると、[従業員]給与フィールドに「ユーザ」モードまたは「カスタムメニュー」モードで選択された値が割り当てられます。

フォーム上で「arSalaries」ドロップダウンリストを作成する

ドロップダウンリストを作成し、それを“ arSalaries ”と名付けます。ドロップダウンリストの名前は、配列の名前と同じでなければなりません。



配列を初期化する

「On Load」イベントをオブジェクトに対して使用し、配列arSalariesを初期化します。これには、「オブジェクトプロパティ」ウィンドウでそのイベントを有効にします。

「オブジェクトメソッド」ボタンをクリックして、次のようにメソッドを作成します。

次の行は、

```
ARRAY REAL(arSalaries ; 10)
For($vlElem ; 1 ; 10)
   arSalaries{$vlElem}:=2000+($vlElem*500)
End for
```

税引き前の給与に応じて、2500、3000 ...7000の数値配列を作成します。

次の行は、

```
arSalaries:=Find in array(arSalaries ; [従業員]給与)
If (arSalaries=-1)
arSalaries:=0
End if
```

新規レコードの作成と既存レコードの修正の両方を処理します。

■ 新規レコードを作成する場合、[従業員]給与フィールドは最初はゼロです。この場合、**Find in array**関数は配列の中で値を検索せずに-1を返します。**If (arSalaries=-1)**というテストにより、arSalariesをゼロにリセットして、ドロップダウンリストで要素がまったく選択されていないことを示します。

■ 既存レコードを修正する場合、**Find in array**関数により配列内の値が取得されてドロップダウンリストの選択した要素をフィールドの現在の値に設定します。特定の従業員の値がリストにない場合、**If (arSalaries=-1)**というテストにより、リストの任意の要素の選択が解除されます。

注：配列選択要素についての詳細は、次の節を参照してください。

選択した値の[従業員]給与フィールドへのレポート

「arSalaries」ドロップダウンリストから選択した値をレポートするには、オブジェクトに対する「On Clicked」または「On Data Change」イベントを処理する必要があります。選択した要素の要素番号は、配列arSalaries自体の値です。従って、式arSalaries{arSalaries}はドロップダウンリストで選択した値を返します。

「arSalaries」ドロップダウンリストのオブジェクトメソッドは、以下のようになります。

```
Case of
    \ (Form event=On Load)
        ARRAY REAL(arSalaries ; 10)
        For($vlElem ; 1 ; 10)
            arSalaries{$vlElem}:=2000+($vlElem*500)
        End for
        arSalaries:=Find in array(arSalaries ; [従業員]給与)
        If (arSalaries=-1)
            arSalaries:=0
        End if
    \ (Form even=On Data Change)
        [従業員]給与:=arSalaries{arSalaries}
End case
```

「ユーザ」モードまたは「カスタムメニュー」モードでは、このドロップダウンリストは次のように表示されます。

次の節では、配列をフォームオブジェクトとして使用する際に配列に対して行う一般的な基本操作を説明します。

配列のサイズを取得する

配列の現在のサイズは、**Size of array**コマンドを使用して取得することができます。前述の例を使用すると、次のコードは、5を表示します。

```
ALERT ("atNames配列のサイズは"+String(Size of array(atNames)) +"です。")
```

配列の要素を並べ替える

配列の要素は、**SORT ARRAY**コマンドを使用して並べ替えることができます。前述の例を使用し、配列がスクロールエリアに表示されているとします。

a. まず、エリアは左のリストのように表示されています。

b. 次のコードを実行すると、エリアは中央のリストのように表示されます。

```
SORT ARRAY(atNames ; >)
```

c. コードの以下の行を実行すると、エリアが右のリストのように表示されます。

```
SORT ARRAY(atNames ; <)
```

要素の追加と削除

要素の追加、挿入、削除は、**INSERT ELEMENT**コマンドおよび**DELETE ELEMENT**コマンドを使用して行います。

配列内のクリックの処理 : 選択した要素のテスト

前述の例を使用し、配列がスクロールエリアに表示されているとすると、このエリアでのクリックは、以下のように処理できます。

```
` 「atNames」スクロールエリアのオブジェクトメソッド
Case of
   \ (Form event=On Load)
      ` 配列を初期化する (前述の例を参照)
      ARRAY TEXT (atNames ; 5)
      ` ...
   \ (Form event=On Unload)
      ` 配列が必要なくなった場合、
      CLEAR VARIABLE(atNames)
   \ (Form event=On Clicked)
      If (atNames#0)
         vtInfo:="atNames{atNames} + "上をクリックしました。"
      End if
   \ (Form event=On Double Clicked)
      If (atNames#0)
         ALERT (atNames{atNames}+ "上をダブルクリックしました。")
      End if
End case
```

注 : 各イベントは、「オブジェクトプロパティ」ウィンドウでアクティブにする必要があります。

シンタックス「**atNames{$vlElem}**」は、配列の特定の要素を使用して作業が行え、シンタックス「**atNames**」は、配列内で選択した要素の要素番号を返します。従って、シンタックス「**atNames{atNames}**」は配列atNames内で選択した要素の値を意味します。

要素が選択されていない場合、atNamesはゼロに等しいので、「**If (atNames#0)**」というテストは、要素が実際に選択されているかどうかを検出します。

選択した要素を設定する

同様に、配列に値を割り当てることにより、選択した要素をプログラムによって変更することができます。

```
` 最初の要素を選択する (配列が空の場合)
atNames:=1

` 最後の要素を選択する (配列が空の場合)
atNames:=Size of array(atNames)

` 選択した要素を選択解除する
atNames:=0

If ((0<atNames)&(atNames<Size of array(atNames))
   ` 可能なら、選択した要素の次の要素を選択する
   atNames:=atNames+1
End if

If (1<atNames)
   ` 可能なら、選択した要素の1つ前の要素を選択する
   atNames:=atNames-1
End if
```

配列内の値を検索する

Find in array関数は、配列内の特定の値を検索します。前述の例を使用すると、次のコードは、「リクエスト」ダイアログボックスに“ 田中 ”と入力すると、その値が“ 田中 ”である要素を選択します。

```
$vsName:=Request ("名字を入力してください：")
If (OK=1)
    $vlElem:=Find in array (atNames ; $vsName)
    If ($vlElem>0)
        atNames:=$vlElem
    Else
        ALERT ($vsName+"という名字の人はいません。")
    End if
End if
```

ポップアップメニュー、ドロップダウンリスト、スクロールエリア、タブコントロールは、通常同じ方法で扱われます。もちろん、別のコードを作成して、要素の値を変更したり、要素を追加、削除したりするたびに画面のオブジェクトを再描画する必要はありません。

注：アイコンと有効、無効なタブによってタブコントロールを作成、使用するには、タブコントロールの支援オブジェクトとして階層リストを使用する必要があります。詳細については、New listコマンドの例を参照してください。

コンボボックスの扱い

ポップアップメニュー、ドロップダウンリスト、スクロールエリア、タブコントロールは前節で説明したアルゴリズムによって制御できますが、コンボボックスの扱いは、これらとは異なります。

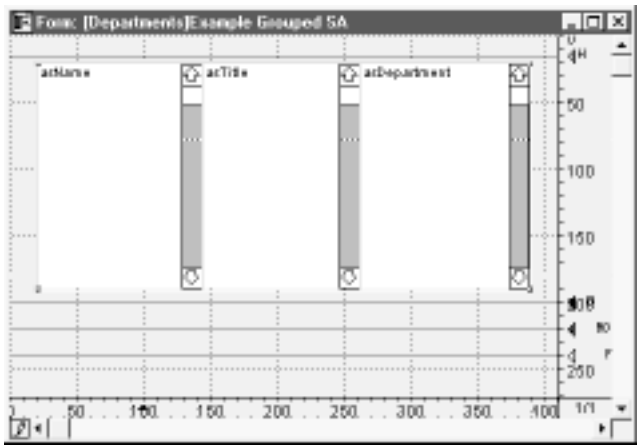
コンボボックスは、実際は、値のリスト(配列の要素)を添付されたテキスト入力可能なエリアです。ユーザはこのリストから値を取り出し、テキストを編集することができます。したがって、コンボボックスでは、選択した要素という概念は適用されません。

コンボボックスには、選択された要素というものはありません。ユーザがエリアに添付された値のいずれかを選択するたびに、その値が配列の要素ゼロに入ります。次にユーザがテキストを編集すると、ユーザが変更した値が要素ゼロに入ります。

```
` 「asColors」コンボボックスのオブジェクトメソッド
Case of
   \ (Form event=On Load)
      ARRAY STRING(31 ; asColors ; 3)
      asColors{1}:="青"
      asColors{2}:="白"
      asColors{3}:="赤"
   \ (Form event=On Clicked)
      If (asColors{0}#"")
         `オブジェクトは自動的にそれの値を変更します
         `コンボボックスでの「On Clicked」イベントの使用は、追加アクションを
         `取得しなければならない時にのみ必要となります
      End if
   \ (Form event=On Data Change)
      ` Find in array関数は、要素ゼロを無視し、-1または0以上の値を返します
      If (Find in array(asColors ; asColors{0})<0)
         ` 入力した値がオブジェクトに付着した値の1つではない場合
         ` 次回、そのリストに値を追加する
         $vlElem:=Size of array(asColors)+1
         INSERT ELEMENT(asColors;$vlElem)
         asColors{$vlElem}:=asColors{0}
      Else
         ` 入力した値がオブジェクトに付着した値の中にある場合
      End if
End case
```

グループ化されたスクロールエリア

スクロールエリアは、グループ化してフォームに表示することができます。複数のスクロールエリアをグループ化すると、それらは1つのスクロールエリアとして動作します。それぞれのスクロールエリアには独自のフォントとフォントサイズが指定できますが、各カラムのフォントの高さ(フォントとフォントサイズに依存します)は同一にすることをお勧めします。データ入力中に表示する場合、一番前にあるスクロールエリアがスクロールバーを表示します。次の図は、「デザイン」モードにおいて1つにグループ化されたスクロールエリアです。



以下にグループ化されたスクロールエリアを作成する際のヒントを示します。

■ すべての配列が同じサイズ（同じ要素数）になっているか確認する。

■ 各スクロールエリアに対して、すべて同じ文字サイズ(ポイント数)を使用する。

■ 各スクロールエリアを同じ高さにする。

■ 各スクロールエリアの1番上と1番下を揃える。

■ 隣接するスクロールエリアが互いに重ならないようにする。

■ 1番右側のスクロールエリアを前面に移動する。実行時には、最前面のスクロールエリアのスクロールバーだけを操作可能にする。

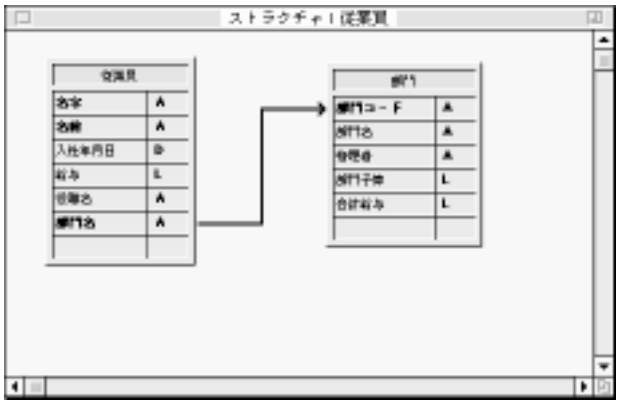
■ 各スクロールエリアに対する単独の操作を終了してから、すべてのエリアを選択し、「設定」メニューから「グループ」を選択する。

■ グループの周りにグループよりも1ピクセル大きい境界線を引く。(グループを選択し、“ Ctrl（Macintosh版では、コマンド）” キーを押したまま “ 1 ” を押す。） 右端の境界線を左に1ピクセル移動（ショートカットでは、“ Ctrl (コマンド) - 左矢印キー）すると、見やすくなる。

次のステートメントは、グループ化されたスクロールエリアで示した3つの配列にデータを格納します。

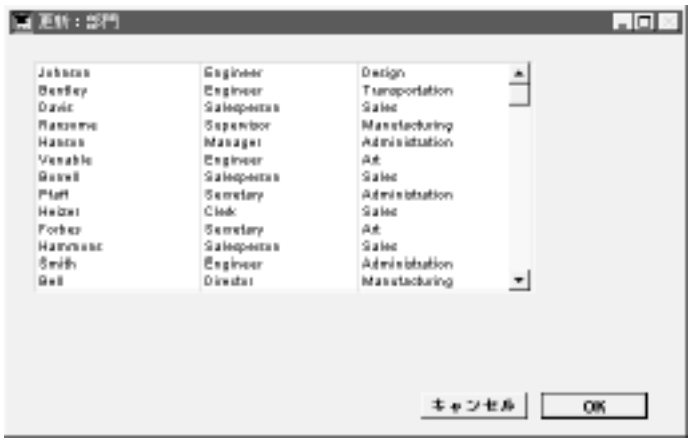
```
ALL RECORDS([従業員])
SELECTION TO ARRAY([従業員]名字 ; asName ; [従業員]役職名 ; asTitle
;[部門]部門名 ; asDepartment)
DIALOG([部門];"グループ化")
```

このメソッドは、[従業員]テーブルと[部門]テーブルのフィールドにあるデータを使用します。これらのテーブルを次に示します。



注：[部門]テーブルは、[従業員]から[部門]への自動リレーションが存在する場合に使用できます。

結果は、次のとおりです。

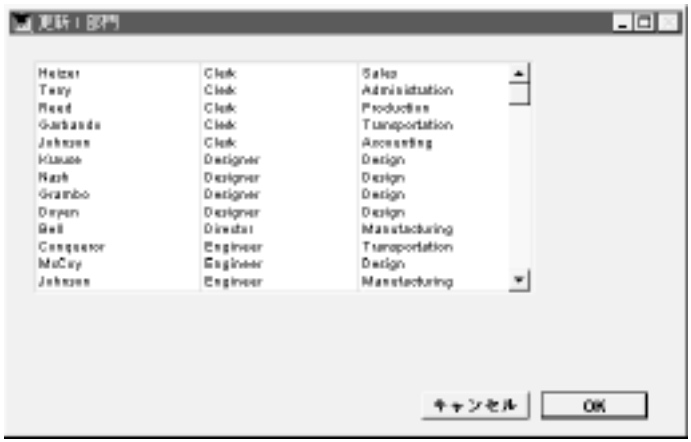


スクロールバーが1つしか表示されていないことに注意してください。スクロールバーは、常に一番前のスクロールエリアに表示されます。このスクロールバーは、3つの配列をあたかも1つの配列のように制御します。ユーザが行をクリックすると、3つのエリア全部が同時に反転表示されます。それぞれのスクロールエリアに関連付けられた変数は、ユーザがクリックした行の番号に設定され、クリックされたエリアのオブジェクトメソッドだけが実行されます。例えば、ユーザが「Bentley」をクリックすると、asName、asTitle、、asDepartmentが全部 2に設定されますが、実行されるのは asNameのオブジェクトメソッドだけです。いずれかの配列で選択した要素をグループ化されたスクロールエリアに設定すると、グループの他の配列が次のイベントについて選択した同じ要素に設定され、スクロールエリアのそれぞれの行が反転表示されます。

配列は、**SORT ARRAY**コマンドを使用してソートすることができます。次に例を示します。

SORT ARRAY(asTitle ; asName ;asDepartment ; >)

次にソート結果を示します。



配列が**SORT ARRAY**コマンドの最初の引数に基づいてソートされたことに注意してください。他の2つの配列は、行が同期するように指定されています。**SORT ARRAY**コマンドは、常に最初の配列の値に基づいて配列をソートし、他の配列をそれに同期させます。

注：SORT ARRAYコマンドは、配列に対するマルチレベルのソートはできません。上記のようなテーブルを表示して、マルチレベルのソートを行うには(すなわち部門でソートしてから、役職名でソートし、次に部門名でソートする)、テーブルを表示するサブフォームを使用してから、ORDERED BYコマンドを使用します。

配列と4D言語

配列は、4D変数です。他の変数と同様、配列にも適用範囲があり4D言語の規則に従いますが、いくつか他の変数とは異なることがあります。

ローカル配列、プロセス配列、インタープロセス配列

ローカル配列、プロセス配列、インタープロセス配列を作成して使用することができます。次に例を示します。

```
ARRAY INTEGER ($aiCodes;100) `2バイト整数値のローカル配列を100個作成します。
ARRAY INTEGER (aiCodes;100) `2バイト整数値のプロセス配列を100個作成します。
ARRAY INTEGER (<>aiCodes;100) `2バイト整数値のインタープロセス配列を100個作成
```

これらの配列の適用範囲（スコープ）は、他のローカル変数、プロセス変数、インタープロセス変数と同一です。

ローカル配列

ローカル配列は、配列の名前をドル記号($)で始めることによって宣言されます。

ローカル配列の適用範囲は、それが作成されたメソッドです。メソッドが終了すると配列も消去されます。2つの異なるメソッドにある同じ名前のローカル配列は、実際には適用範囲の異なる2つの別個の変数なので、それぞれに異なるタイプを指定することができます。

フォームメソッド、オブジェクトメソッド、またはこれらのメソッドによってサブルーチンとして呼び出されるプロジェクトメソッドの中でローカル配列を作成する場合、配列はフォームメソッドまたはオブジェクトメソッドが起動されるたびに作成され消去されます。言い換えれば、配列はフォームイベントごとに作成され消去されます。従って、目的が表示であれ印刷であれ、ローカル配列をフォームに使用することはできません。

ローカル変数については、可能であればローカル配列を使用することをお勧めします。そうすることにより、多くの場合、アプリケーションを実行するために必要なメモリの量を小さくできます。

プロセス配列

プロセス配列は、配列名を文字で始めることによって宣言されます。

プロセス配列の適用範囲（スコープ）は、それが作成されたプロセスです。配列は、プロセスの終了時またはアボート時に消去されます。プロセス配列には、プロセスごとに1つのインスタンスが自動的に作成されます。従って、配列はプロセスのどこでも同じタイプです。ただし、その内容はプロセスによって異なります。

インタープロセス配列

インタープロセス配列は、配列名を<>(WindowsおよびMacintosh)またはUSキーボードで「Option」-「Shift」-「V」で表示されるダイヤモンド記号(Macintoshのみ)で始めることによって宣言されます。

インタープロセス配列の適用範囲（スコープ）は、作業セッションのすべてのプロセスで構成されます。これらの使用は、プロセス間でのデータの共有や情報の転送のみに限る必要があります。

Tips：インタープロセス配列が複数のプロセスによってアクセスされ、それが競合を生じる恐れがあると予測できる場合、その配列へのアクセスをセマフォによって保護します。詳細については、Semaphore関数を参照してください。

注：プロセス配列とインタープロセス配列をフォームの中で使用して、スクロールエリアやドロップダウンリストなどのフォームオブジェクトなどを作成することができます。

配列の引数（パラメータ）としての受け渡し

配列を引数（パラメータ）として4Dコマンドや4Dプラグインのルーチンに渡すことができます。しかし、配列をパラメータとしてメソッドに渡すことはできません。代わりに、配列へのポインタを引数としてメソッドに渡すことはできます。詳細については、次節の「配列とポインタ」を参照してください。

配列の他の配列への割り当て

テキストや文字列変数とは異なり、配列を他の配列に割り当てることはできません。配列を他の配列にコピーする(割り当てる)には、**COPY ARRAY**コマンドを使用します。

配列とポインタ

配列を引数（パラメータ）として4Dコマンドや4Dプラグインのルーチンに渡すことができます。しかし、配列を引数としてメソッドに渡すことはできません。代わりに、配列へのポインタを引数としてメソッドに渡すことはできます。

注：プロセス配列やインタープロセス配列を引数として渡すことはできますが、ローカル配列は渡せません。

次に例を示します。

▼ 次のような例を考えます。

```
If ((0<atNames)&(atNames<Size of array(atNames))
``可能であれば、選択した要素の次の要素を選択する
   atNames:=atNames+1
End if
```

異なるフォームの50の異なる配列に同じ処理を行う場合、50回同じことを書かずに次のプロジェクトメソッドを使用することができます。

```
` 「SELECT NEXT ELEMENT」プロジェクトメソッド
` SELECT NEXT ELEMENT (ポインタ)
` SELECT NEXT ELEMENT (-> 配列)
C_POINTER ($1)
If ((0<$1->)&($1-><Size of array($1->))
   $1->:=$1->+1 `可能であれば、選択した要素の次の要素を選択します。
End if
```

次に、以下のように記述します。

```
SELECT NEXT ELEMENT (->atNames)
  ...
SELECT NEXT ELEMENT (->asZipCodes)
  ...
SELECT NEXT ELEMENT (->alRecordIDs)
` ...
```

▼ 次のプロジェクトメソッドは、数値配列(整数、倍長整数または実数)のすべての要素の合計を返します。

```
` 「Array sum」プロジェクトメソッド
` Array sum (ポインタ)
` Array sum (->配列)
C_REAL ($0)
$0:=0
For ($vlElem ; 1 ; Size of array($1->))
   $0:=$0+$1->{$vlElem}
End for
```

次に、以下のように記述します。

```
$vlSum:= Array sum (->arSalaries)
` ...
$vlSum:= Array sum (->aiDefectCounts)
` ..
$vlSum:= Array sum (->alPopulations)
```

▼ 次のプロジェクトメソッドは、文字列またはテキスト配列のすべての要素を大文字に変換します。

```
` 「CAPITALIZE ARRAY」プロジェクトメソッド
` CAPITALIZE ARRAY (ポインタ)
` CAPITALIZE ARRAY (->配列)
For ($vlElem ; 1 ; Size of array($1->))
I   f ($1->{$vlElem}#"")
      $1->{$vlElem}:=Uppercase ($1->{$vlElem}[[1]])+ Lowercase(Substring( $1->{$vlElem} ; 2))
   End if
End if
```

次に、以下のように記述します。

```
CAPITALIZE ARRAY (->atSubjects )
` ...
CAPITALIZE ARRAY (->asLastNames )
```

配列、ポインタ、およびループ構造(**For... End for**など)を組み合わせることにより、配列を扱うための短くて便利なプロジェクトメソッドを作成することができます。

配列の要素ゼロを使用する

配列には、常に要素ゼロがあります。配列がフォームオブジェクトをサポートする場合は要素ゼロは表示されませんが、ランゲージでの使用については制限はありません。

要素ゼロの用途の一例として、配列とフォームオブジェクトで説明するコンボボックスの例があります。

その他の例を2つ挙げます。

1. その前に選択した要素以外の要素をクリックした場合のみ動作を行わせる場合、選択した要素のそれぞれを追跡する必要があります。これを行う方法の1つは、選択する要素の要素番号を保持するプロセス変数を使用することです。もう1つの方法は、次のように配列の要素ゼロを使用する方法です。

```
` 「atNames」スクロールエリアのオブジェクトメソッド
Case of
   \ (Form event=On Load)
      `配列を初期化する
      ARRAY TEXT (atNames ; 5)
      ` ...
      `要素ゼロをその文字列フォームの中で選択した現在の要素の番号で初期化する
      `これで選択した要素なしで開始します。
      atNames{0}:="0"
   \ (Form event=On Unload)
      ` この配列はもう必要ありません
      CLEAR VARIABLE(atNames)
   \ (Form event=On Clicked)
      If (atNames#0)
         If (atNames#Num(atNames{0}))
            vtInfo:=atNames{atNames}+"上がクリックされました。
                        これは、以前選択されていませんでした。"
            atNames{0}:=String(atNames)
         End if
      End if
   \ (Form event=On Double Clicked)
      If (atNames#0)
         ALERT (atNames{atNames} +"上がダブルクリックされました。")
      End if
End case
```

2. ドキュメントまたはシリアルポートから一連の文字群を送受信する場合、4Dはプラットフォームと異なるASCIIマップを使用するプラットフォームおよびシステム間でASCIIコードをフィルタリングする方法として、**USE ASCII MAP**、**Mac to ISO**、**ISO to Mac**、**Mac to Win**、**Win to Mac**コマンドを提供します。

場合によっては、ASCIIコードを変換する方法を完全に制御したいことがあります。これには、255要素からなる整数配列を使用して、N番目の要素をその送信元のASCIIコードがNである文字の変換後のASCIIコードに設定します。例えば、ASCIIコード#187を#156に変換する場合、データベース中で使用されるインタープロセス配列を初期化するメソッドの中に、「**<>aiCustomOutMap{187}:=156**」と「**<>aiCustomInMap{156}:=187**」と記述します。すると、次のカスタムプロジェクトメソッドを使用して、一連の文字群を送信することができます。

```
` X SEND PACKET (テキスト { ; 時間} )
For ($vlCha r ; 1 ; Length($1))
   $1[[vlChar]]:=Char(<>aiCustomOutMap{Ascii($1[[vlChar]])})
End for
If (Count parameters>=2)
   SEND PACKET ($2 ; $1)
Else
   SEND PACKET ($1)
End if

` X Receive packet (テキスト { ; 時間} )→テキスト
If (Count parameters>=2)
   RECEIVE PACKET ($2 ; $1)
Else
   RECEIVE PACKET ($1)
End if
$0:=$1
For ($vlChar ; 1 ;Length($1))
$0[[vlChar]]:=Charr(<>aiCustomInMap{Ascii($0[[vlChar]])})
End for
```

この応用例では、NULL文字(ASCIIコードのゼロ)が入っている一連の文字群を送受信する場合、配列のゼロ要素<>aiCustomOutMapおよび<>aiCustomInMapは 255要素の配列の他の要素としての役割を果たします。

2次元配列

配列宣言コマンドは、すべて1次元配列または2次元配列の作成またはサイズ変更を行えます。次に例を示します。

```
ARRAY TEXT (atTopics ; 100 ; 50) `100行50列からなるテキスト配列を作成します。
```

2次元配列は、本質的にはランゲージオブジェクトなので、表示も印刷もできません。

前述の例で、

- atTopicsは、2次元配列です。
- atTopics{8}{5 }は、8行目の5番目(5列目)の要素です。
- atTopics{20}は、20行目で配列自体は1次元配列です。
- **Size of array**(atTopics)は、行数の100を返します。
- **Size of array**(atTopics{17})は、17行目の列数である50を返します。

次の例では、データベースの各テーブルの各フィールドへのポインタが2次元配列に格納されます。

```
` まず、空の行をテーブルと同じ数だけ作成する
ARRAY POINTER (<>apFields ; Count tables ; 0)
`各テーブルに対して、
For ($vlTable ; 1 ; Size of array(<>apFields))
   `行の列数がテーブルのフィールドと同じ数になるようにサイズ変更する
   INSERT ELEMENT (<>apFields{$vlTable} ; 1 ; Count tables($vlTable))
   `要素の値を設定する
   For ($vlField ; 1 ; Size of array(<>apFields{$vlTable}))
      <>apFields{$vlTable}{$vlField}:=Field($vlTable;$vlField)
   End for
End for
```

この2次元配列が初期化されている場合、次の方法で特定のテーブルのフィールドへのポインタを取得することができます。

```
`現在画面表示されているテーブルのフィールドへのポインタを取得する
COPY ARRAY (<>apFields{Table(Current form table)};$apTheFieldsIamWorkingOn)
`ブールと日付フィールドを初期化する
For ($vlElem ; 1 ; Size of array($apTheFieldsIamWorkingOn))
   Case of
      : (Type($apTheFieldsIamWorkingOn{$vlElem}->)=Is Date)
         $apTheFieldsIamWorkingOn{$vlElem}->:=Current date
      : (Type($apTheFieldsIamWorkingOn{$vlElem}->)=Is Boolean)
         $apTheFieldsIamWorkingOn{$vlElem}->:=True
   End case
End for
```

注：この例でわかるように、2次元配列の行は同じサイズでも異なるサイズでも構いません。

配列とメモリ

テーブルやレコードを使用してディスク上に格納したデータと異なり、配列は常に全部がメモリに保存されます。

例えば、米国内の郵便番号がすべて[郵便番号]テーブルに入力されている場合、約100,000個のレコードになります。加えて、そのテーブルが郵便番号自体、対応する市、郡、州という複数のフィールドを持ちます。カリフォルニアからのみ郵便番号を選択する場合、4Dデータベースエンジンが[郵便番号]テーブルの中の対応するレコードを選択して、それらが必要な場合のみ、そのレコードをロードします(例えば表示や印刷時)。言い換えれば、4Dのデータベースエンジンによってディスクからメモリに部分的にロードされた(各フィールドで同じタイプの)番号付きの一連の値で作業します。

同じことを配列で行うのは、次の理由で禁止されています。

■ 4つの情報タイプ(郵便番号、市、郡、州)を維持するためには、4つの大きな配列をメモリ内で維持する必要があります。

■ 配列は、常に全体がメモリ内に維持されるので、絶えず使用するわけでない場合も、作業セッションの間中すべての郵便番号をメモリに置いておく必要があります。

■ 同じく配列全体が常にメモリ内に維持されることから、データが作業セッション中に使用も変更もされていない場合でも、データベースが開始されて終了するたびに、4つの配列をロードしてからディスクに保存する必要があります。

結論：配列は、ほどよい量のデータを短時間維持するためのものです。他方、配列はメモリ内に置かれるので、扱いやすく高速操作が可能です。

しかし、状況によっては何百、何千という要素を持った配列で作業する必要があります。次の表に、各配列がメモリ上に占めるバイト数を求めるための計算式を示します。

配列タイプ	メモリ上に占めるバイト数を求めるための計算式
ブール	(((1 + 配列の要素の数) // 8) +1) *8
日付	(1 + 配列の要素の数) *6
文字列	2+ (1 + 配列の要素の数) * 指定した文字列の長さ(+1 余りの場合, +2 余りなしの場合)
整数	(1 + 配列の要素の数) * 2
倍長整数	(1 + 配列の要素の数) * 4
ピクチャ	4 + (1 + 配列の要素の数) * 4 + 各ピクチャサイズの合計
実数	(1 + 配列の要素の数) * 10
テキスト	4 + ((1 + 配列の要素の数) * 6) + 各テキストサイズの合計
2次元配列	4 + (1 + 配列の要素の数) * 12 + 各配列サイズの合計

注：選択した要素、要素番号、配列自体の情報を保存するために、別に数バイトを要します。非常に大きな配列で作業する場合、メモリが一杯という状況を扱う最善の方法は、配列の作成をON ERR CALLプロジェクトメソッドで囲むことです。以下に例を示します。

```
`大きな配列を作成する必要がある晩に一晩かけてバッチ操作を実行します。
`夜中にエラーが発生する危険を犯さないように、操作の始めに配列を作成して、
`この時点でエラーをテストします。
gError:=0 `エラーがないと仮定する
ON ERR CALL ("エラー処理") `エラーを検出するためのメソッドをインストールする
ARRAY STRING (63 ; asThisArray ; 50000) ` 約3125K
ARRAY REAL (arThisAnotherArray ; 50000) ` 488K
ON ERR CALL ("") `これ以上エラーを検出する必要はない
If (gError=0)
   `配列を作成することができました。操作を続行します。
Else
   ALERT ("この処理は多くのメモリを必要とします。")
End if
`どのような場合でも、これ以上配列は必要ありません。
CLEAR VARIABLE (asThisArray)
CLEAR VARIABLE (arThisAnotherArray)
```

「エラー処理」プロジェクトメソッドを次に示します。

```
`「エラー処理」プロジェクトメソッド
gError:=Error `エラーコードだけが返される
```

配列コマンド

この節では、この章では、「ルーチン」エディタの「Arrays」テーマ内にある配列コマンドについて説明します。本節のコマンドを使用すると、配列の作成、配列のソート、配列要素の検索、テーブルに対するデータの出し入れ、およびその他の処理を行うことができます。配列は一般にスクロールエリアやポップアップメニューに表示します。また、配列はメモリ上でデータを扱うためにも使用します。次のコマンドは配列処理に適しています：

ARRAY BOOLEAN	**ARRAY STRING**	**Find in array**
ARRAY DATE	**ARRAY TEXT**	**LIST TO ARRAY**
ARRAY INTEGER	**ARRAY TO LIST**	**INSERT ELEMENT**
ARRAY LONGINT	**ARRAY TO SELECTION**	**Size of array**
ARRAY PICTURE	**COPY ARRAY**	**SELECTION TO ARRAY**
ARRAY POINTER	**DELETE ELEMENT**	**SUB SELECTION TO ARRAY**
ARRAY REAL	**DISTINCT VALUES**	**SORT ARRAY**

ARRAY BOOLEAN
ARRAY DATE
ARRAY INTEGER
ARRAY LONGINT
ARRAY PICTURE
ARRAY POINTER
ARRAY REAL
ARRAY TEXT
ARRAY STRING

ARRAY BOOLEAN (配列名 ; サイズ1 ; {サイズ2})
ARRAY DATE (配列名 ; サイズ1 ; {サイズ2})
ARRAY INTEGER (配列名 ; サイズ1 ; {サイズ2})
ARRAY LONGINT (配列名 ; サイズ1 ; {サイズ2})
ARRAY PICTURE (配列名 ; サイズ1 ; {サイズ2})
ARRAY POINTER (配列名 ; サイズ1 ; {サイズ2})
ARRAY REAL (配列名 ; サイズ1 ; {サイズ2})
ARRAY TEXT (配列名 ; サイズ1 ; {サイズ2})

引数	タイプ	説明
配列名	配列	新しい配列の名前
サイズ1	数値	配列の要素の数 サイズ2を指定した場合は、配列の数
サイズ2	数値	２次元配列の要素の数

ARRAY STRING (長さ ; 配列名 ; サイズ1 ; {サイズ2})

引数	タイプ	説明
長さ	数値	文字列の長さ
配列名	配列	配列の名前
サイズ1	数値	配列の要素の数 サイズ2を指定した場合は、配列の数
サイズ2	数値	２次元配列の要素の数

これらの配列コマンドは、すべてメモリ上に配列を作成します。

引数＜配列名＞は、作成する配列の名前です。

引数＜サイズ1＞は、配列の要素の数です。

引数＜サイズ2＞はオプションです。この引数は、２次元配列を作成する場合に使用します。その場合に、＜サイズ1＞に配列の数を、＜サイズ2＞には各配列の要素の数を指定します。

2次元配列の各配列は要素と同じように扱います。従って、この節の他のコマンドを使用して、2次元配列中の配列に要素を挿入したり削除することができます。

引数＜長さ＞は、文字列の配列にのみ使用します。これは、配列中の各要素に含むことのできる最大文字数を指定します。文字列の配列は、テキストの配列よりも高速に処理を実行します。

0番目の要素(配列名{0})は、常に１つの配列として作成されます。配列タイプに応じた空の値が代入されます。配列の要素数は、**Size of array**関数を使用して求めます。次の行は配列の定義はそのまま残しますが、(0番目の要素を除く)すべての配列要素を削除します：

ARRAY TEXT (Mine ; 0)

配列を作成した時点の配列の各要素は、空の値になります。数値の配列は0、文字列とテキストは""、日付の配列は!00.00.00!、ブールの配列は“ False ”です。

配列の要素は、中括弧({})を使用して参照します。例えば、配列“ 私の配列 ”の2番目の要素を参照する場合は、“ 私の配列{2} ”と記述します。

２次元配列の要素は、2組の中括弧を使用して参照します。例えば、“ 私の配列{3}{5} ”と記述すると、3番目の配列の5番目の要素を参照することができます。

この節のコマンドを使用して配列の大きさを変更することができます。これらのコマンドを既存の配列に使用して、要素を追加したり、削除することができます。例えば、4つの要素を持つテキストの配列“ Mine ”に対して次のステートメントを実行すると、2つの要素が削除されます：

ARRAY TEXT (Mine ; 2)

そして、次のステートメントを実行すると、既存の要素に影響を与えないで4つの要素が追加されます：

ARRAY TEXT (Mine ; 6)

▼ 次の例は、文字列の配列を作成し、サブテーブルの情報を配列に格納します：

```
ALL SUBRECORDS ([従業員]子供)
ARRAY STRING (15 ; a名前 ; Records in subselection ([従業員]子供))
For ($i;1;Size of array (a名前))
  a名前{$i}:=[従業員]子供'名前
  NEXT SUBRECORDS ([従業員]子供)
End for
```

SORT ARRAY

SORT ARRAY (配列1{ ;...; 配列N} ; {ソート種別})

引数	タイプ	説明
配列	配列	ソートする配列
ソート種別	>または<	> 昇順 、 < 降順

SORT ARRAYコマンドは、１つまたは複数の配列を昇順や降順にソートします。ピクチャ以外の、すべての配列タイプに使用することができます。また、配列を2次元配列の最初の次元にすることはできません。

引数＜ソート種別＞は、配列を昇順にソートするのか、降順にソートするのかを指定します。＜ソート種別＞に(>)を指定すると昇順にソートします。 ＜ソート種別＞に(<)を指定すると降順にソートします。＜ソート種別＞を省略した場合は昇順にソートします。

複数の配列を指定した場合は、最初の配列の順番でソートします。各配列ごとに個々にソートするわけではありません。この特性は、互いに関連する配列をソートする上で有用です。

▼ 次の例は、配列を2つ作成し、元のレコードに影響を与えないで名前を昇順でソートします：

```
ALL RECORDS ([従業員])
SELECTION TO ARRAY ([従業員]名前 ; arNames ; [従業員]会社 ; arComp)
SORT ARRAY (arNames ; arComp ; >)
```

COPY ARRAY

COPY ARRAY (コピー元 ; コピー先)

引数	タイプ	説明
コピー元	配列	コピー元の配列
コピー先	配列	コピー先の配列

COPY ARRAYコマンドは、既存の配列を複製します。＜コピー先＞が存在しない場合は、＜コピー元＞の配列と全く同じ大きさとタイプを持つ配列を作成します。＜コピー先＞が存在する場合には、新しく作成した配列に置き換えます。

＜コピー元＞と＜コピー先＞の配列はローカル配列、プロセス配列にすることができます。配列の範囲は、配列を複製する際には問題ではありません。

▼ 次の例は、配列 “ arCompany ” と同じ大きさと同じ内容を持つ配列 “ arData ” を作成します：

```
` 従業員テーブルのすべてのレコードを選択
ALL RECORDS ([従業員])
` 会社フィールドのデータを配列「arCompany」に格納
SELECTION TO ARRAY ([従業員]会社 ; arCompany)
` 配列 “ arCompany ” を配列 “ arData ” にコピー
COPY ARRAY (arCompany ; arData)
```

INSERT ELEMENT

INSERT ELEMENT (配列 ; 挿入位置 ; {要素数})

引数	タイプ	説明
配列	配列	配列の名前
挿入位置	数値	要素を挿入する位置
要素数	数値	挿入する要素の数

INSERT ELEMENTコマンドは、＜配列＞に 1つまたは複数の要素を挿入します。引数＜挿入位置＞の示す位置に新しい要素を挿入します。新しく挿入された要素には、配列タイプに応じた空の値が代入されます。＜挿入位置＞より後ろの要素は、＜要素数＞で指定した数だけ後ろに移動します。

＜挿入位置＞が配列より大きい(配列の要素の数より多い)場合は、配列の最後に挿入します。

引数＜要素数＞は、挿入する要素の数を指定します。省略した場合は、1と解釈します。配列の大きさは、＜要素数＞によって動的に増加します。

▼次の例は、配列“ リスト ”の10番目の要素の位置に新しい 5つの要素を挿入します：

INSERT ELEMENT (リスト ; 10 ; 5)

DELETE ELEMENT

DELETE ELEMENT (配列 ; 削除位置 ; {要素数})

引数	タイプ	説明
配列	配列	要素を削除する配列
削除位置	数値	要素の削除を開始する位置
要素数	数値	削除する要素の数

DELETE ELEMENTコマンドは、1つまたは複数の要素を＜配列＞から削除します。引数＜削除位置＞の示す位置から要素の削除を開始します。

＜削除位置＞が配列より大きい(配列の要素の数より多い)場合は、配列の最後を削除します。引数＜要素数＞は、削除する要素の数を指定します。省略した場合は、1と解釈します。配列の大きさは、＜要素数＞によって動的に減少します。

▼次の例は、配列“リスト”の5番目の要素から3つの要素を削除します：

```
DELETE ELEMENT (リスト ; 5 ; 3)
```

Find in array

Find in array (配列 ; 値 ; {開始}) → 数値

引数	タイプ	説明
配列	配列	検索する配列
値	文字列または数値、日付またはブール	検索する値
開始	数値	検索を開始する要素の番号

Find in array関数は、＜配列＞から引数＜値＞と同じものを検索し、最初に発見された要素の番号を返します。

Find in array関数は、タイプがテキスト、文字列、数値、日付、ブールの配列に使用します。配列タイプと値のタイプは、必ず同じにしてください。

引数＜値＞と同じものを発見できない場合には、**Find in array**関数は -1を返します。

引数＜開始＞を指定するとその番号の要素から検索を初めます。引数＜開始＞を指定していない場合には、コマンドは第1要素から検索します。

▼ 次の例は、セレクションのレコードから配列 “ arComp ” を作成し、フィールド “ [従業員]会社 ” が “ 福岡 ” のレコードを探すために、**GOTO SELECTED RECORD**コマンドの引数として**Find in array**関数を使用します：

```
` 従業員テーブルの全レコードを選択
ALL RECORDS ([従業員])
` 会社ﾌｨｰﾙﾄﾞのデータを配列「arComp」に格納
SELECTION TO ARRAY ([従業員]会社 ; arComp)
GOTO SELECTED RECORD ([従業員] ; Find in array (arComp ; "福岡"))
```

Size of array

Size of array (配列) → 数値

引数	タイプ	説明
配列	配列	大きさを求める配列

Size of array関数は、＜配列＞の大きさ(要素の数)を返します。＜配列＞が2次元配列の場合には、配列の数を返します。

▼ 次の例は、配列 “ 私の配列 ” の大きさを変数 “ vサイズ ” に代入します：

```
vサイズ:=Size of array (私の配列)          ` 私の配列の大きさを求める
```

LIST TO ARRAY

LIST TO ARRAY (リスト ; 配列 ; {リンク配列})

引数	タイプ	説明
リスト	文字列	コピー元のリスト
配列	配列	コピー先の配列
リンク配列	配列	配列リンク先リストのコピー先の配列

LIST TO ARRAYコマンドは、＜リスト＞から＜配列＞を作成します。このコマンドは、データを＜リスト＞から＜配列＞にコピーします。配列が既存の場合は、配列を上書きします。配列のタイプを前もって文字型として定義しない限り、テキストの配列を作成します。

＜リンク配列＞には、リストからリンクされたリストの名前を代入します。リストの項目がリンクしたリストを持っている場合、リンク先のリストの名前をリンク配列の同じ番号の要素に代入します。リンク先のリストが存在しない場合は、空の文字列を代入します。＜リンク配列＞は、リストと同じ大きさです。リンク配列に含まれる名前は、リンク先のリストを参照するために使用します。

▼ 次の例は、リスト “ 地方 ” の項目を配列 “ a地方 ” にコピーします。リスト “ 地方 ” のリンク先のリストの名前は、配列 “ リンク ” にコピーします：

```
LIST TO ARRAY ("地方" ; a地方 ; リンク)
```

ARRAY TO LIST

ARRAY TO LIST (配列 ; リスト ; {リンク配列})

引数	タイプ	説明
配列	配列	コピー元の配列
リスト	文字列	コピー先のリスト
リンク配列	配列	配列リンク先のリストの名前

ARRAY TO LISTコマンドは、＜配列＞を＜リスト＞にコピーします。＜リスト＞が存在しない場合は、リストを作成します。

＜リンク配列＞の各要素は、リンク先のリストの名前として使用します。＜リンク配列＞の要素が空の文字列でない場合は、その要素に含まれる名前は、リストの項目とリンク先のリストとの対応をとるために使用します。その項目が有効なリストの名前でない場合には、リンクされません。

▼ 次の例は、配列 “ a地方 ” をリスト “ 地方 ” にコピーします。配列 “ リンク ” は、リストのリンク対応に使用します：

ARRAY TO LIST (a地方 ; "地方" ; リンク)

SELECTION TO ARRAY

SELECTION TO ARRAY (フィールド1 ; 配列1{ ;...; フィールドN ; 配列N})

引数	タイプ	説明
フィールド	フィールド	データを使用するフィールド
配列	配列	フィールドのデータを格納する配列

SELECTION TO ARRAY (テーブル ; 配列1{ ; ... ; フィールドN ; 配列N})

引数	タイプ	説明
テーブル	テーブル	レコード番号を使用するテーブル
配列	配列	レコード番号を格納する配列

第1のシンタックスでは、**SELECTION TO ARRAY**コマンドは、1つまたは複数の配列を作成し、フィールドのデータやカレントセレクションのフィールドをその配列にコピーします。最初に指定したフィールドの属するテーブルのカレントセレクションを指定します。自動リレートが設定されていれば、別のテーブルのフィールドも使用することができます。配列の大きさ(要素の数)は、セレクションのレコード数と一致します。

配列は、対応する各フィールドのタイプと同じになります。ただし、既存の文字列の配列に、テキストフィールドのデータをコピーした場合は、文字列の配列のままです。また、時間フィールドを配列にコピーした場合は、倍長整数型の配列になります。

第２のシンタックスでは、**SELECTION TO ARRAY**コマンドは、＜配列＞に＜テーブル＞のレコード番号の配列を作成します。デフォルトのデータタイプは倍長整数型の配列です。

SELECTION TO ARRAYコマンドは、セレクションの大きさによって非常に大きな配列を作成する場合があります。配列はメモリに常駐しますので、十分なメモリがあるかどうか必ず確認してください。

SELECTION TO ARRAYコマンドを呼び出した後、カレントセレクションとカレントレコードは同じままですが、カレントレコードはロードされません。そのため、もし、カレントレコードのフィールド値を使いたい場合は、**SELECTION TO ARRAY**コマンドの後に**LOAD RECORD**コマンドを使用してください。

▼ 次の例は、自動リレートで関連している[従業員]テーブルと[会社]テーブルのデータから“ arFirst ”と“ arAddress ”の2つの配列を作成します。この場合に、配列の大きさは[従業員]テーブルのセレクションのレコード数と一致します：

SELECTION TO ARRAY ([従業員]名前 ; arFirst ; [会社]住所 ; arAddress)

▼ 次の例は、配列 “ AFileNum ” にレコード番号を返し、配列 “ ANames ” に[顧客]名前フィールドを返します：

SELECTION TO ARRAY ([顧客] ; AFileNum ; [顧客]名前 ; ANames)

ARRAY TO SELECTION

ARRAY TO SELECTION (配列1 ; フィールド1{ ;...; 配列N ; フィールドN})

引数	タイプ	説明
配列	配列	セレクションをコピーした配列
フィールド	フィールド	配列のデータを格納するフィールド

ARRAY TO SELECTIONコマンドは、1つまたは複数の配列をセレクションのレコードにコピーします。すべてのフィールドは、同一テーブルのものでなければなりません。

セレクションが既存の場合は、配列要素を要素の順番とレコードの順番に従ってレコードに書き込みます。要素の方がレコードよりも多い場合は、新しいレコードを作成します。レコードは、既存でも新規でも、自動的に保存されます。

大きさが異なる配列を複数指定した場合に、最初に指定した配列の大きさが採用されます。もし、他の配列が大きいと、追加された要素は無視され、小さい場合は、追加された要素はヌル値になります。

このコマンドは、**SELECTION TO ARRAY**コマンドの機能を補足します。**ARRAY TO SELECTION**コマンドは、自動リレートが設定されていても、リレートテーブルのフィールドを使用することはできません。

ARRAY TO SELECTIONコマンドはカレントセレクションとカレントレコードを修正します。

警告：ARRAY TO SELECTIONコマンドは、既存のレコードの情報を上書きします。十分に注意して使用してください。

ARRAY TO SELECTIONコマンド実行中にレコードがロックされると、そのレコードは修正されません。ロックされたレコードは、「LockedSet」というシステムセットの中に置かれます。**ARRAY TO SELECTION**コマンドを実行した後に、任意のレコードがロックされていないかを確認するために「LockedSet」を調べることができます。

▼ 次の例は、“ arFirst ”、“ arComp ” の2つの配列のデータを[従業員]テーブルにコピーします。配列 “ arFirst ” は “ 従業員]名前 ” フィールドに、配列 “ arComp ” は “ [従業員]会社 ” フィールドに、それぞれ書き込みます：

ARRAY TO SELECTION (arFirst ; [従業員]名前 ; arComp ; [従業員]会社)

DISTINCT VALUES

DISTINCT VALUES (フィールド ; 配列)

引数	タイプ	説明
フィールド	フィールド	データとして使用するフィールド
配列	配列	インデックスフィールドデータを受け取る配列

DISTINCT VALUESコマンドは、フィールドやサブフィールドが属すテーブルのカレントセレクションに対して引数＜フィールド＞から生成される重複しない値を使って＜配列＞を作成します。

バージョン3の4th Dimensionでは、このコマンドは文字フィールドしか受け渡すことができませんでしたが、バージョン6から任意のインデックスフィールドおよびサブフィールドを受け渡すことができるようになりました。ただし、インデックス属性を持つブールフィールドを使用した場合は、**DISTINCT VALUES**コマンドは何も行いません。

あるテーブルのフィールドを受け渡す場合、**DISTINCT VALUES**コマンドは現在選択されたレコード群の中にある重複しない値をブラウズ（走査）および保持します。しかし、サブフィールドを受け渡すと、**DISTINCT VALUES**コマンドは現在選択されているレコードの中にあるサブレコードをすべてブラウズします。

このコマンドを呼び出す前に配列を作成すると、**DISTINCT VALUES**コマンドはあなたが受け渡したフィールドまたはサブフィールドと互換性のある配列タイプと予想します。別の言い方をすれば、インタプリタモードでは、**DISTINCT VALUES**コマンドはそれ相応の配列タイプを作成します。しかし、任意のフィールドまたはサブフィールドが時間タイプの場合、**DISTINCT VALUES**コマンドは任意の倍長整数配列と予想または作成します。

DISTINCT VALUESコマンドを呼び出した後、配列の大きさはセレクション内の個々の値の数と等しくなります。このコマンドはカレントセレクションやカレントレコードを変更しません。**DISTINCT VALUES**コマンドはインデックス属性を持ったフィールドを使用するので、＜配列＞の要素は昇順にソートされた順番で返されます。これがあなたが必要とする順番の場合は、**DISTINCT VALUES**コマンドを使用した後に**SORT ARRAY**コマンドを呼び出す必要はありません。

DISTINCT VALUESコマンドは、セレクションの大きさとレコード内の個々の値の数によって大きな配列を作成します。配列はメモリに常駐しますので、十分なメモリがあるかどうか必ず確認してください。

4D Server：このコマンドは、4D Server用に最適化されます。配列は、サーバマシン上で作成され、値が計算され、その配列はクライアントにそっくりそのまま送信されます。

▼ 次の例は、セレクション内の全都市のリストを作成し、店舗のある都市の数をユーザに知らせます：

```
ALL RECORDS ([売上])                    ` レコードのセレクションを作成
DISTINCT VALUES ([売上]都市 ; 配列都市)
ALERT ("店舗が " + String (Size of array (配列都市))+" 市にあります。")
```

SUBSELECTION TO ARRAY

SUBSELECTION TO ARRAY (開始 ; 終了 ; フィールドまたはテーブル ; 配列 ; {フィールド2またはテーブル2 ; 配列2...;フィールドNまたはテーブルN ; 配列N})

引数	タイプ	説明
開始	数値	データ抽出を開始するために選択されたレコード番号
終了	数値	データ抽出を終了するために選択されたレコード番号
フィールド / テーブル	フィールドまたはテーブル	データ抽出用に使用するフィールドまたはレコード番号抽出用に使用するテーブル
配列	配列	フィールドデータまたはレコード番号を抽出するための配列

SUBSELECTION TO ARRAYコマンドは1つまたは複数の配列を作成し、その配列にフィールドデータまたはカレントセレクションのレコード番号をコピーします。

配列にカレントセレクションを用いる**SELECTION TO ARRAY**コマンドと違って、**SUBSELECTION TO ARRAY**コマンドは、単に引数「開始」と「終了」によって指定された選択レコードの範囲を用いるだけです。

ユーザは、次のフォーミュラ（**1 <=「開始」<=「終了」<= Records in selection([...])**）を満たしている「開始」と「終了」の選択レコード番号を**SUBSELECTION TO ARRAY**コマンドに受け渡します。

もし、次のフォーミュラ（**1 <=「開始」=「終了」< Records in selection([...])**）を受け渡すと、選択されたレコードが（「開始」=「終了」）であるレコードからフィールドをロードしてくるか、またはレコード番号を取得します。

もし、間違った選択レコード番号を受け渡すと、このコマンドは次のようなことを行います：

■（**「終了」> Records in selection([...])**）の場合は、「開始」によって指定された選択レコードから最終の選択レコードに値を返します。

■（**「開始」>「終了」**）の場合は、選択されたレコードが「開始」であるレコードから値を返します。

■ 両方の引数がセレクションのサイズと一致しない場合は、空の配列が返されます。

SELECTION TO ARRAYコマンド同様、**SUBSELECTION TO ARRAY**コマンドも第1引数で指定したテーブルのセレクションを用います。

SUBSELECTION TO ARRAYコマンドは**SELECTION TO ARRAY**コマンド同様、次のようなことを行うことができます：

■ 1つまたは複数のフィールドから値をロードする

■ シンタックス（[テーブル] ; 配列 ; ...）を使って、レコード番号をロードする

■ n対 1自動リレートまたはマニュアル（手動）のn対 1 リレートを自動リレートにするために**AUTOMATIC RELATION**コマンドをコールして提供されるリレートフィールドから値をロードする（どちらの場合も、値はn対 1 リレートのいくつかのレベルを通してテーブルからロードされます）

各配列のタイプは、下記の2つの例外を除いてフィールドタイプに依存します：

■ テキストフィールドは、文字列配列の中にコピーされます。この場合、その配列は文字列配列のままです。

■ 時間フィールドは、倍長整数配列の中にコピーされます。

注：サブテーブルおよびサブフィールドを指定することはできません。

レコード番号をロードすると、それらは倍長整数配列の中にコピーされます。

警告：SUBSELECTION TO ARRAYコマンドは、ロードしてくるデータのタイプやサイズだけでなく「開始」と「終了」で指定した範囲により大きい配列を作成します。配列はメモリ上に常駐するので、返された配列のサイズを調べることによってそのコマンド実行後の結果を調べたり、またはON ERR CALLコマンドを基にしたプロシージャを使ってそのコマンドの呼びだしをカバーすることは良いアイデアです。

コマンドが正常に実行されると、返される配列のサイズは（**「終了」-「開始」+ 1**）になります。ただし、引数「終了」がセレクションのレコード数より大きい場合は、その返される配列は（**Records in selection([...]) -「開始」+ 1**）個の要素を含みます。

▼ 次の例は、[送り状]テーブルのカレントセレクションの先頭から50レコードを選択します。そして、[送り状]送り状番号フィールドおよびリレートフィールドの[顧客]顧客番号から値をロードしてきます：

```
SUBSELECTION TO ARRAY ( 1 ; 50 ; [送り状]送り状番号 ; arInvoID ; [顧客]顧客番号 ; arCustID)
```

▼ 次の例は、[送り状]テーブルのカレントセレクションの最終から50レコードを選択します。そして、[送り状]レコードおよび[顧客]リレートレコードのレコード番号をロードしてきます：

```
vSelSize := Records in selection ([送り状])
SUBSELECTION TO ARRAY (vSelSize - 49 ; vSelSize ; [送り状] ; arInvoID ; arInvRecNo
; [顧客] ; arCustRecNo)
```

▼ 次の例は、配列の中にダウンロードできない大きいセレクション内の1000レコードのかたまりを使って処理を行い続けます：

```
vMaxPage := 1000
vSelSize := Records in selection ([電話帳])
For ($vPage ; 1 ; 1 + (vSelSize - 1) // vMaxPage))
   `値またはレコード番号をロードする
   SUBSELECTION TO ARRAY ( 1 + (vMaxPage * ($vPage - 1)) ; vMaxPage * $vPage ;
   ... ; ... ; ... ; ... ; ... ; ... )
   `何らかの配列処理を行う
End for
```

参照：AUTOMATIC RELATION、ON ERR CALL、SELECTION TO ARRAY、Size of array

BLOBの定義

4th Dimension バージョン 6 では、BLOB (Binary Large OBjects) データタイプが新しく追加されました。

BLOB フィールドおよび BLOB 変数は、次のように定義します。

■ BLOB フィールドを作成するには、「フィールドプロパティ」ウィンドウ内の「フィールドタイプ」ドロップダウンリストで BLOB を選択します。

■ BLOB 変数を作成するには、コンパイラ宣言コマンド **C_BLOB**コマンドを使用します。タイプが BLOB のローカル変数、プロセス変数、インタープロセス変数を作成することができます。

注： BLOB の配列はありません。

4th Dimension の中では、1つの BLOB は連続した可変長バイトであり、1つのまとまったオブジェクトまたは各バイトが個々にアドレス指定できるオブジェクトとして取り扱うことができます。1つの BLOBは空 (長さが NUL) でもよく、また最大 2,147,483,647 バイト (2 GB) まで含むことができます。

BLOB とメモリ

1 つの BLOB は、そっくりそのままメモリの中にロードされます。1 つの BLOB 変数は、メモリ内にだけ保持され、存在します。1 つの BLOB フィールドは、そのフィールドが属すレコードの他の部分と同様に、ディスクからメモリにロードされます。

大量のデータを保持できる他のフィールドタイプ (ピクチャフィールドやサブテーブルフィールド) と同様に、レコードを更新しても BLOB フィールドはメモリに複製されません。その結果、**Old**関数および **Modified**関数によって返される結果は、BLOBフィールドに適用しても意味を持ちません。

BLOBの表示

1つの BLOBには、どのような種類のデータでも保持できるため、画面上でのデフォルトの表現はありません。フォーム内で BLOBフィールドまたは変数を表示すると、どのような内容であっても常に空白になります。

BLOBフィールド

BLOBフィールドを使用すると、最大で 2GB（ギガバイト）のどのような種類のデータでも保存することができます。BLOBフィールドを検索することはできないため、BLOBフィールドに保存された値のレコードを検索するには、数式を使用しなければなりません。検索処理を使用してすばやく検索する対象となるようなデータを保存する方法としては、BLOBを使用してはいけません。例えば、BLOBフィールドには、キーワードを格納してはいけません。その代わりに、キーワードサブフィールドをインデックス付けできるサブテーブルを使用します。

引数 (パラメータ) の受け渡し、ポインタおよび関数の結果

4th Dimensionの BLOBは、4Dコマンドまたは BLOBパラメータを想定している4D Extentionsルーチンの引数 (パラメータ)として受け渡すことができます。一方では、BLOBをユーザメソッドのパラメータとして受け渡すことはできません。BLOBは、関数の結果として戻すことができません。

BLOBを自分の独自のメソッドに渡すには、その BLOB へのポインタを定義し、そのポインタをパラメータとして受け渡します。

例：

```
`タイプが BLOBの変数を宣言する
C_BLOB (anyBlobVar)
` BLOB は、パラメータとして任意の4D コマンドに受け渡される
SET BLOB SIZE (anyBlobVar ; 1024*1024)
` BLOB は、パラメータとして外部ルーチンに受け渡される
$errCode:= Do Something With This BLOB (anyBlobVar)
` BLOB へのポインタは、ユーザメソッドへパラメータとして渡される
COMPUTE BLOB (->anyBlobVar )
` タイプがポインタの変数を宣言する
C_POINTER (aPointer)
` BLOB へのポインタを定義する
aPointer :=->anyBlobVar
` BLOB へのポインタは、ユーザメソッドへパラメータとして渡される
COMPUTE BLOB (aPointer)
```

4D Extensions開発者のための注意点：BLOBパラメータは “ &O ” (数字の"0"ではなく、アルファベットの"O")として宣言されます。

BLOBの割り当て

BLOBを相互に割り当てることができます。

例：
```
`タイプがBLOBの2つの変数を宣言する
C_BLOB (vBlobA ; vBlobB)
`最初のBLOBのサイズを10Kに設定する
SET BLOB SIZE (vBlobA ; 10*1024)
` 最初のBLOBを2番目のBLOBに割り当てる
vBlobB:=vBlobA
```

ただし、BLOBに演算子を適用することはできません。BLOBタイプの式はありません。

BLOBの内容のアドレス指定

中括弧{...} を使用すれば、BLOBの各バイトは、個別にアドレス指定できます。1つのBLOB内では、各バイトには0～N-1の番号が割り当てられています。このとき、NはBLOBのサイズです。例は、次のとおりです。

```
` BLOBのタイプの変数を宣言する
C_BLOB (vBlob)
` BLOBのサイズを256バイトに設定する
SET BLOB SIZE (vBlob ; 256)
` このループは、BLOBの256バイトをゼロに初期化する
For ( vByte ; 0 ; BLOB size (vBlob)-1)
   vBlob{vByte}:=0
End for
```

BLOBの各バイトはすべて、個別にアドレス指定できるため、BLOBフィールドまたはBLOB変数に格納したいものは実際には何でも格納できます。

BLOBの4th Dimensionコマンド

4th Dimensionには、動作しているBLOBのための次のようなコマンドがあります。

■ **SET BLOB SIZE**コマンドは、BLOBフィールドやBLOB変数のサイズを変更します。

■ **BLOB size**関数は、BLOBのサイズを戻します。

■ **DOCUMENT TO BLOB**コマンドおよび**BLOB TO DOCUMENT**コマンドを使用すると、ドキュメント全体をBLOBからロードしたり、BLOBに書き込むことができます(オプションで、Macintosh上のデータフォークおよびリソースフォークとの間でもこの操作が可能です)。

■ **VARIABLE TO BLOB**コマンドおよび**BLOB TO VARIABLE**コマンドを使用すると、**LIST TO BLOB**コマンドおよび**BLOB to list**関数と同様に、4D変数をBLOBで格納したり、検索することができます。

■ **COMPRESS BLOB**コマンド、**EXPAND BLOB**コマンド、および**BLOB PROPERTIES**コマンドを使用すると、圧縮されたBLOBを操作することができます。

■ **BLOB to integer**関数、**BLOB to longint**関数、**BLOB to real**関数、**BLOB to text**関数、**INTEGER TO BLOB**コマンド、**LONGINT TO BLOB**コマンド、**REAL TO BLOB**コマンド、および**TEXT TO BLOB**コマンドを使用すると、ディスク、リソース、OSなどから入力される構造化されたデータを操作することができます。

■ **DELETE FROM BLOB**コマンド、**INSERT IN BLOB**コマンド、および**COPY BLOB**コマンドを使用すると、BLOB内にある大きいサイズのデータのまとまりをすばやく処理することができます。

これらのコマンドについては、この章で説明しています。

追記:

■ **C_BLOB**コマンドは、タイプがBLOBの変数を宣言します。詳細は、第19章の「コンパイラコマンド」の章を参照してください。

■ **GET CLIPBOARD**コマンドおよび**APPEND CLIPBOARD**コマンドを使用すると、クリップボードに格納されているどのデータタイプでも操作できます。詳細は、第17章の「クリップボードコマンド」の章を参照してください。

■ **GET RESOURCE**コマンドおよび**SET RESOURCE**コマンドを使用すると、ディスク上に格納されているリソースでも操作できます。詳細は、第46章の「リソースコマンド」の章を参照してください。

BLOBコマンド

15

この章では、「ルーチン」エディタの「BLOB」テーマ内にあるBLOBコマンドについて説明します。

DOCUMENT TO BLOB
BLB TO DOCUMENT
VARIABLE TO BLOB
BLOB TO VARIABLE
COMPRESS BLOB
EXPAND BLOB
BLOB PROPERTIES
INTEGER TO BLOB
BLOB to integer
LONGINT TO BLOB
BLOB to longint
REAL TO BLOB
BLOB to real
TEXT TO BLOB
BLOB to text
LIST TO BLOB
BLOB to list
COPY BLOB
INSERT IN BLOB
DELEE FROM BLOB
BLOB size
SET BLOB SIZE

SET BLOB SIZE

SET BLOB SIZE (**blob ; サイズ {;フィルタ}**)

引数	タイプ		説明
blob	BLOB	→	BLOBフィールドまたはBLOB変数
サイズ	数値	←	BLOBの新しいサイズ
フィルタ	数値	→	フィルタ文字のASCIIコード

SET BLOB SIZEコマンドは、引数＜サイズ＞に渡された値にしたがって、BLOB＜blob＞のサイズを変更します。

デフォルトでは、BLOBに新しく割り当てられたバイトがある場合には、それらは“ 0x00 ”に初期化されます。これらのバイトを他の値に初期化したい場合には、その値(0～255)をオプション引数の＜フィルタ＞に渡します。

例：

1. 大規模なプロセスBLOBまたはインタープロセスBLOBを終了した後では、占有していたメモリを解放することをお勧めします。そのためには、次のように記述します：

```
SET BLOB SIZE(aProcessBLOB ; 0)
SET BLOB SIZE(<>anInterprocessBLOB ; 0)
```

2. 次の例では、0xFFが埋め込まれた16KBのBLOBが1つできます：

```
C_BLOB(vxData)
SET BLOB SIZE(vxData ; 16*1024 ; 0xFF)
```

参照：BLOB size

エラー処理

メモリ不足のためにBLOBのサイズを変更できない場合には、エラーコード -108が発生します。**ON ERR CALL**割り込みメソッドを使用すれば、このエラーを取り出すことができます。

BLOB size

BLOB size (blob) → 数値

引数	タイプ		説明
blob	BLOB	→	BLOBフィールドまたはBLOB変数
関数の返す値	数値	←	BLOBのサイズ（バイト単位）

BLOB size関数は、バイトで表現された＜blob＞のサイズを返します。

▼ 次の例は、“ myBlob ” BLOBに100バイトが追加されます：

```
SET BLOB SIZE (BLOB size(myBlob)+100)
```

参照：SET BLOB SIZE

COMPRESS BLOB

COMPRESS BLOB (blob {;圧縮})

引数	タイプ		説明
blob	BLOB	→	BLOBフィールドまたはBLOB変数
圧縮	数値	←	省略しなかった場合、 1=できるだけ小さく圧縮する 2=できるだけ速く圧縮する

COMPRESS BLOBコマンドは、内部の4th Dimension圧縮アルゴリズムを使用して、引数＜blob＞というBLOBを圧縮します。

オプション引数の＜圧縮＞を使用すると、BLOBを圧縮する方法を設定できます：

■ 1を渡すと、圧縮および解凍の操作の速度と引き換えに、BLOBができるだけ小さく圧縮されます。

■ 2を渡すと、圧縮率と引き換えに(圧縮されたBLOBのサイズは大きくなります)、BLOBができるだけ速く圧縮されます(展開の速度もできるだけ速くなります)。

■ 他の値を渡したり、引数を省略すると、圧縮モード1を使用してBLOBができるだけ小さく圧縮します。

4th Dimensionには、次のようにあらかじめ定義された定数があります。

定数	タイプ	値
Compact compression mode	倍長整数	1
Fast compression mode	倍長整数	2

呼び出し後、BLOBが正常に圧縮された場合には、OK変数は1に設定されます。BLOBを圧縮するために必要なメモリがない、などの理由で圧縮が実行できなかった場合には、OK変数は 0に設定されます。

BLOBは、圧縮された後には**EXPAND BLOB**コマンドを使用して解凍することができます。

BLOBが正常に圧縮されたかどうかを確認するには、**BLOB PROPERTIES**コマンドを使用します。

警告：圧縮されてもBLOBは依然BLOBであるため、その内容を更新できなくなるわけではありません。ただし、BLOBの内容を更新する場合には、EXPAND BLOBコマンドを使用すると、BLOBプロパティを正常に解凍することはできません。

▼ 次の例では、“ vxMyBlob ” BLOBが圧縮されたかどうかテストし、圧縮されていない場合には、これを圧縮します：

```
BLOB PROPERTIES (vxMyBlob ; $vlCompressed ; $vlExpandedSize ; $vlCurrentSize)
If ($vlCompressed=Is not compressed)
   COMPRESS BLOB (vxMyBlob)
End if
```

ただし、すでに圧縮されているBLOBに対してCOMPRESS BLOBを実行すると、このコマンドはこれを認識し、何も実行しません。

▼ 次の例では、ドキュメントを選択して、これを圧縮します：

```
If (OK=1)
    CLOSE DOCUMENT ($vhDocRef)
    DOCUMENT TO BLOB (Document ; vxBlob)
    If (OK=1)
        COMPRESS BLOB (vxBlob)
        If (OK=1)
            BLOB TO DOCUMENT (vxBlob ; Document)
        End if
    End if
End if
```

参照：BLOB PROPERTIES、EXPAND BLOB

システム変数またはシステムセット

BLOBが正常に圧縮された場合には、OK変数は1に設定されます。そうでない場合には、0に設定されます。

EXPAND BLOB

EXPAND BLOB (**blob**)

引数	タイプ		説明
blob	BLOB	→	解凍するBLOB

EXPAND BLOBコマンドは、**COMPRESS BLOB**コマンドを使用してすでに圧縮されている引数<blob>を解凍します。

呼び出し後、BLOBが圧縮された(または、BLOBがそもそも圧縮されていなかった)場合には、OK変数は1に設定されます。BLOBを圧縮するために必要なメモリがない、などの理由で圧縮が実行できなかった場合には、OK変数は0に設定されます。

BLOBが圧縮されているかどうかを確認するには、**BLOB PROPERTIES**コマンドを使用します。

▼ 次の例では、“ vxMyBlob ” BLOBが圧縮されたかどうかテストし、圧縮されている場合には、これを展開します：

```
BLOB PROPERTIES (vxMyBlob ; $vlCompressed ; $vlExpandedSize ; $vlCurrentSize)
If ($vlCompressed # Is not compressed)
   EXPAND BLOB (vxMyBlob)
End if
```

ただし、圧縮されていないBLOBに対して**EXPAND BLOB**コマンドを実行すると、このコマンドはそれを検知し、何も行いません。

▼ 次の例では、文書を選択し、その文書が圧縮されている場合には解凍することができます：

```
$vhDocRef := Open document ("")
If (OK=1)
    CLOSE DOCUMENT ($vhDocRef)
    DOCUMENT TO BLOB (Document ; vxBlob)
    If (OK=1)
        BLOB PROPERTIES (vxBlob;$vlCompressed;$vlExpandedSize;$vlCurrentSize)
        If ($vlCompressed#Is not compressed)
            EXPAND BLOB (vxBlob)
            If (OK=1)
                BLOB TO DOCUMENT (vxBlob ; Document)
            End if
        End if
    End if
End if
```

参照：BLOB PROPERTIES, COMPRESS BLOB

システム変数またはシステムセット

BLOBが正常に圧縮された場合には、OK変数は1に設定されます。そうでない場合には、0に設定されます。

BLOB PROPERTIES

BLOB PROPERTIES (blob ; 圧縮 {;サイズ {現在サイズ}})

引数	タイプ		説明
blob	BLOB	→	BLOBフィールドまたはBLOB変数
圧縮	数値	←	0=BLOBは圧縮されていない 1=できるだけ小さく圧縮されたBLOB 2=できるだけ速く圧縮されたBLOB
サイズ	数値	←	圧縮されてない場合のBLOBサイズ
現在サイズ	数値	←	BLOBの現在サイズ（バイト単位）

BLOB PROPERTIESコマンドは、引数＜blob＞に関する情報を返します。

■ 引数＜圧縮＞はBLOBが圧縮されたかどうかを示し、次のようなあらかじめ定義された定数の1つの値を返します。

定数	タイプ	値
Is not compressed	倍長整数	0
Compact compression mode	倍長整数	1
Fast compression mode	倍長整数	2

■ BLOBの圧縮状態がどのようなものであっても、引数＜サイズ＞は、圧縮されていない時のBLOBのサイズを戻します。

■ 引数＜現在サイズ＞は、BLOBの現在のサイズを戻します。BLOBが圧縮されている場合には、通常、＜サイズ＞より小さい＜現在サイズ＞を取得します。BLOBが圧縮されていない場合には、常に、＜サイズ＞に等しい＜現在サイズ＞を取得します。

例：

▼ **COMPRESS BLOB**コマンドおよび**EXPAND BLOB**コマンドの例を参照してください。

▼ BLOBが圧縮された後、次のプロジェクトメソッドはこの圧縮でできた空間の割合を取得します：

```
` 「Space saved by compression」プロジェクトメソッド
` Space saved by compression (ポインタ {; ポインタ } ) → 倍長整数
` Space saved by compression (-> BLOB {; ->savedBytes } ) → 割合
C_POINTER ($1 ;$2)
C_LONGINT ($0 ; $vlCompressed ; $vlExpandedSize ; $vlCurrentSize)
BLOB PROPERTIES ($1-> ; $vlCompressed ; $vlExpandedSize ; $vlCurrentSize)
If ($vlExpandedSize=0)
   $0:=0
   If (Count parameters>=2)
      $2->:=0
   End if
Else
   $0:=100-(($vlCurrentSize/$vlExpandedSize)*100)
   If (Count parameters>=2)
      $2->:=$vlExpandedSize-$vlCurrentSize
   End if
End if
```

このメソッドがアプリケーションに追加された後は、これを次のように使用することができます。

```
COMPRESS BLOB (vxBlob)
$vlPercent:= Space saved by compression (->vxBlob ; ->vlBlobSize)
ALERT ("圧縮サイズと割合："+String  (vlBlobSize)+"バイト"+String ($vlPercent;"#0%")+"%")
```

参照：COMPRESS BLOB、EXPAND BLOB

DOCUMENT TO BLOB

DOCUMENT TO BLOB (**文書 ; blob {;*}**)

引数	タイプ		説明
文書	文字列	→	文書の名前
blob	BLOB	→	文書を受信するBLOBフィールドまたはBLOB変数
		←	文書の内容
*	*	←	Macintosh上でのみ： *が指定された場合は、リソースフォーク、 それ以外は、データリソースがロードされる

DOCUMENT TO BLOBコマンドは、文書の内容全体を引数＜blob＞にロードします。まだ開かれていない既存の文書の名前を渡す必要があり、それ以外の場合には、エラーが発生します。BLOBへロードする文書をユーザが選択できるようにするには、**Open document**関数およびプロセス変数Documentを使用します(下の例を参照してください)。

Macintoshでの注意点：

Macintoshの文書は、データフォークおよびリソースフォークの2つのフォークから構成されています。デフォルトでは、**DOCUMENT TO BLOB**コマンドは文書のデータフォークをロードします。文書のリソースフォークをロードするには、オプション引数＜＊＞を渡します。Windowsでは、オプション引数＜＊＞は無視されます。4D環境では、MacOSのリソースフォークと同等のものをWindowsで実現することに注意が必要です。例えば、4Dデータベースのデータフォークは、「.4DB」ファイル拡張子のあるファイルに格納され、リソースフォークは、同じ名前でファイル拡張子が「.RSR」のファイルに格納されます。Windowsでは、データフォークおよびリソースフォークをBLOBに格納して4Dアプリケーションを記述する場合には、操作したい方のフォークに対応するファイルにアクセスするだけですみます。

▼ 次の例は、文書をすばやく格納したり検索できるような情報システムを記述する場合を想定します。データ入力フォームで、文書をBLOBフィールドにロードできるようなボタンを作成します。このボタンのメソッドは、次のようにできます。

```
$vhDocRef:=Open document("") `目的のドキュメントを選択する
If (OK=1) `文書が選択されている場合、
   CLOSE DOCUMENT($vhDocRef) `開いたままにしておく必要はない
   DOCUMENT TO BLOB (文書 ; [テーブル1]BLOBフィールド) `文書をロードする
   If (OK=0)
      `エラー処理を行う
   End if
End if
```

参照：BLOB TO DOCUMENT、Open document.

システム変数

文書が正常にロードされた場合には、OK変数は1に設定されます。そうでない場合には、0に設定され、エラーが発生します。

エラー処理

■ 存在しない文書や、すでに他のプロセスやアプリケーションで開かれている文書を(BLOBに)ロードしようとすると、それぞれに対応するファイルマネージャエラーが発生します。

■ 文書がロックされていたり、ロックされているボリュームにあったり、文書をロードする際に問題が発生すると、I/Oエラーが発生する場合があります。

■ メモリ不足のために文書をロードできない場合には、エラーコード -108が発生します。

いずれの場合でも、**ON ERR CALL**割り込みメソッドを使用すれば、このエラーをトラップできます。

BLOB TO DOCUMENT

BLOB TO DOCUMENT (**文書 ; blob {;*}**)

引数	タイプ		説明
文書	文字列	→	文書の名前
blob	BLOB	→	文書の新しい内容
*	*	←	Macintosh上でのみ： *が指定された場合は、リソースフォーク、 それ以外は、データリソースがロードされる

BLOB TO DOCUMENTコマンドは、引数＜blob＞に格納されているデータを使用して文書の内容全体を再記述します。まだ開かれていない既存の文書の名前を渡す必要があり、それ以外の場合には、エラーが発生します。文書をユーザが選択できるようにするには、**Open document**関数または**Create document**関数、およびプロセス変数documentを使用します(下の例を参照してください)。

Macintoshでの注意点：

Macintoshの文書は、データフォークおよびリソースフォークの2つのフォークから構成されています。デフォルトでは、**DOCUMENT TO BLOB**コマンドは文書のデータフォークをロードします。文書のリソースフォークをロードするには、オプション引数＜＊＞を渡します。Windowsでは、オプション引数＜＊＞は無視されます。4D環境では、MacOSのリソースフォークと同等のものをWindowsで実現することに注意が必要です。例えば、4Dデータベースのデータフォークは、「.4DB」ファイル拡張子のあるファイルに格納され、

リソースフォークは、同じ名前でファイル拡張子が「.RSR」のファイルに格納されます。Windowsでは、データフォークおよびリソースフォークをBLOBに格納して4Dアプリケーションを記述する場合には、操作したい方のフォークに対応するファイルにアクセスするだけですみます。

▼ 次の例は、文書をすばやく格納したり検索できるような情報システムを記述する場合を想定します。データ入力フォームで、BLOBフィールドにロードされているデータが含まれている文書を保存できるようなボタンを作成します。このボタンのメソッドは、次のようにできます：

```
$vhDocRef:=Create document("") `目的のドキュメントを保存する
If (OK=1) `文書が作成された場合、
   CLOSE DOCUMENT($vhDocRef) `開いたままにしておく必要はない
   DOCUMENT TO BLOB (文書 ; [テーブル1]BLOBフィールド) `文書の内容を記述する
   If (OK=0)
      `エラー処理を行う
   End if
End if
```

参照：Create document、**DOCUMENT TO BLOB**、Open document

システム変数

文書が正常にロードされた場合には、OK変数は1に設定されます。そうでない場合には、0に設定され、エラーが発生します。

エラー処理

■ 存在しない文書や、すでに他のプロセスやアプリケーションで開かれている文書を(BLOBに)ロードしようとすると、それぞれに対応するファイルマネージャエラーが発生します。

■ 文書がロックされていたり、ロックされているボリュームにあったり、文書をロードする際に問題が発生すると、I/Oエラーが発生する場合があります。

■ メモリ不足のために文書をロードできない場合には、エラーコード -108が発生します。

いずれの場合でも、**ON ERR CALL**割り込みメソッドを使用すれば、このエラーをトラップできます。

VARIABLE TO BLOB

VARIABLE TO BLOB (**変数 ; blob {;オフセット | *}**)

引数	タイプ		説明
変数	変数	→	BLOBの中に格納する変数
blob	BLOB	→	変数を受信するBLOB
オフセット	*	変数	*	→	値を追加するためのBLOB内でのオフセット(バイト単位)または *
		←	* でなければ、書き込み後の新しいオフセット

VARIABLE TO BLOBコマンドは、引数＜変数＞を＜blob＞に格納します。

オプション引数＜* ＞を指定した場合には、変数はBLOBに追加され、これに合わせてBLOBのサイズも拡張されます。オプション引数＜ * ＞を使用すれば、BLOBがメモリ容量内であれば変数やリスト(他のBLOBコマンドを参照してください)をいくつでも順番にBLOBの中に格納することができます。

オプション引数＜ * ＞や＜オフセット＞変数を指定しない場合には、変数はBLOBの先頭に格納され、それ以前にそこにあった内容を上書きします。これに合わせてBLOBのサイズも調整されます。

引数＜オフセット＞変数を渡した場合には、変数はBLOB内のオフセット(ゼロから始まります)に書き込まれます。変数を書き込む位置に関わらず、BLOBのサイズは渡した位置(および必要な場合には変数のサイズ)にしたがって増加します。現在書き込んでいるバイト以外の新しく割り当てられるバイトは、ゼロに初期化されます。

呼び出し後、＜オフセット＞変数は返され、書き込まれたバイト数分だけ増分されます。従って、その同じ変数を別のBLOB書き込みコマンドにも使用して別の変数やリストを書き込むことができます。

VARIABLE TO BLOBコマンドは、次のものを除いて、どのようなタイプの変数でも(他の BLOBも)受け付けます：

- ポインタ
- ポインタ配列
- 2次元配列

ただし、階層リスト(ListRef)への参照である倍長整数の変数を格納した場合には、**VARIABLE TO BLOB**コマンドは階層リストではなく倍長整数の変数を格納します。BLOB内に階層リストを格納したり、BLOBから階層リストを取り出すには、**LIST TO BLOB**コマンドおよび**BLOB to list**関数を使用します。

警告：BLOBを使用して変数を格納すると、変数は4D内部形式を使用してBLOBに格納されるため、格納されたBLOBの内容を読み出すにはBLOB TO VARIABLEコマンドを使用しなければなりません。

呼び出し後、変数が正常に格納された場合には、OK変数は1に設定されます。変数を格納するために必要なメモリがない、などの理由で処理が実行できなかった場合には、OK変数は0に設定されます。

プラットフォームからの独立性に関する注意点：

VARIABLE TO BLOBコマンドおよび**BLOB TO VARIABLE**コマンドは、4D内部形式を使用してBLOB内に格納された変数を処理します。このメリットとして、これら2つのコマンドを使用していればプラットフォーム間でのバイトのスワップに配慮する必要がありません。つまり、これらのコマンドのうちいずれかを使用してWindows上で作成されたBLOBはMacintosh上でも使用でき、その逆も可能です。

例：

1. 次の2つのプロジェクトメソッドを使用すると、ディスク上の文書へすばやく配列を格納したり、文書からすばやく配列を取得できます：

```
` 「配列保存」プロジェクトメソッド
` 配列保存 ( 文字列 ; ポインタ )
` 配列保存 ( 文書 ; ->配列 )
C_STRING (255 ; $1)
C_POINTER ($2)
C_BLOB ($vxArrayData)
VARIABLE TO BLOB ($2-> ; $vxArrayData) ` BLOBの中に配列を格納する
COMPRESS BLOB ($vxArrayData) ` BLOBを圧縮する
BLOB TO DOCUMENT ($1 ; $vxArrayData) ` ディスク上にBLOBを保存する

` 「配列読み込み」プロジェクトメソッド
` LOAD ARRAY ( 文字列 ; ポインタ )
` LOAD ARRAY ( 文書 ; ->配列 )
C_STRING (255;$1)
C_POINTER ($2)
C_BLOB ($vxArrayData)
DOCUMENT TO BLOB ($1 ; $vxArrayData) ` ディスクからBLOBを読み込む
EXPAND BLOB ($vxArrayData) `BLOBを解凍する
BLOB TO VARIABLE ($vxArrayData ; $2->) `BLOBから配列を取り出す
```

上記のメソッドをアプリケーションに追加すれば、次のように記述することができます。

```
ARRAY STRING (...;asAnyArray;...)
   ` ...
配列保存 ( $vsDocName ; ->asAnyArray)
   ` ...
配列読み込み ( $vsDocName ; ->asAnyArray)
```

2. 次の2つのプロジェクトメソッドを使用すると、BLOBへすばやく変数のセットを格納したり、BLOBからすばやく変数のセットを取得することができます：

```
` 「STORE VARIABLES INTO BLOB」プロジェクトメソッド
` STORE VARIABLES INTO BLOB ( ポインタ { ; ポインタ ... { ; ポインタ } } )
` STORE VARIABLES INTO BLOB ( BLOB { ; 変数1 ... { ; 変数2 } } )
C_POINTER ($1)
C_LONGINT ($vlParam)
SET BLOB SIZE ($1-> ; 0)
For ($vlParam ; 2 ; Count parameters)
   VARIABLE TO BLOB (${$vlParam}-> ; $1-> ; *)
End for

` 「RETRIEVE VARIABLES FROM BLOB」プロジェクトメソッド
` RETRIEVE VARIABLES FROM BLOB ( ポインタ { ; ポインタ ... { ; ポインタ } } )
` RETRIEVE VARIABLES FROM BLOB ( BLOB { ; 変数1 ... { ; 変数2 } } )
C_POINTER ($1)
C_LONGINT ($vlParam ; $vlOffset)
$vlOffset:=0
For ($vlParam ; 2 ; Count parameters)
   BLOB TO VARIABLE ($1-> ; ${$vlParam}-> ; $vlOffset)
End for
```

これらのメソッドをアプリケーションに追加すれば、次のように記述することができます。

```
STORE VARIABLES INTO BLOB (->vxBLOB ; ->vgPicture ; ->asAnArray ; ->alAnotherArray)
` ...
RETRIEVE VARIABLES FROM BLOB (->vxBLOB ; ->vgPicture ; ->asAnArray ; ->alAnotherArray)
```

参照：BLOB to list、BLOB TO VARIABLE、LIST TO BLOB

システム変数またはシステムセット

変数が正常に格納された場合には、OK変数は1に設定されます。それ以外の場合には、OK変数は0に設定されます。

BLOB TO VARIABLE

BLOB TO VARIABLE (blob ; 変数 {;オフセット})

引数	タイプ		説明
blob	BLOB	→	4D変数を含んだBLOB
変数	変数	←	BLOBの内容を上書きする変数
オフセット	数値	→	BLOB内の変数の位置
		←	BLOB内の次の変数の位置

BLOB TO VARIABLEコマンドは、引数＜オフセット＞で指定されたバイトオフセット(ゼロから開始します)にある＜blob＞に格納されているデータを使用して＜変数＞を上書きします。

BLOBデータは宛先変数と整合性を保っていなければなりません。通常、**VARIABLE TO BLOB**コマンドを使用して入力したBLOBを使用します。

オプション引数の＜オフセット＞を指定しない場合には、変数データはBLOBの最初から読み込まれます。複数の変数が格納されているBLOBを操作する場合には、＜オフセット＞の他に、数値変数も渡さなければなりません。呼び出しの前に、この数値変数を適切なオフセットに設定します。呼び出しの後で、この同じ数値変数はBLOB内に格納されている次の変数のオフセットを返します。

呼び出し後、変数が正常に再記述された場合には、OK変数は1に設定されます。変数を再記述するために必要なメモリがない、などの理由で処理が実行できなかった場合には、OK変数は0に設定されます。

プラットフォームからの独立性に関する注意点：

BLOB TO VARIABLEコマンドおよび **VARIABLE TO BLOB**コマンドは、4D内部形式を使用してBLOB内に格納された変数を処理します。このメリットとして、これら2つのコマンドを使用していればプラットフォーム間でのバイトのスワップに配慮する必要がありません。つまり、これらのコマンドのうちいずれかを使用してWindows上で作成されたBLOBはMacintosh上でも使用でき、その逆も可能です。

▼ **VARIABLE TO BLOB**コマンドの例を参照してください。

参照：VARIABLE TO BLOB

システム変数またはシステムセット

変数が正常に格納された場合には、OK変数は1に設定されます。それ以外の場合には、OK変数は0に設定されます。

LIST TO BLOB

LIST TO BLOB (**変数 ; blob {;オフセット | *}**)

引数	タイプ		説明
変数	変数	→	BLOBの中に格納する階層リスト
blob	BLOB	→	階層リストを受信するBLOB
オフセット	*	変数	*	→	値を追加するためのBLOB内でのオフセット(バイト単位)または *
		←	* でなければ、書き込み後の新しいオフセット

LIST TO BLOBコマンドは、BLOB内に階層リストを格納します。

オプション引数＜ * ＞を指定した場合には、階層リストはBLOBに追加され、これに合わせてBLOBのサイズも拡張されます。オプション引数＜ * ＞を使用すれば、BLOBがメモリ容量内であれば変数やリスト(他のBLOBコマンドを参照してください)をいくつでも順番にBLOBの中に格納することができます。

オプション引数＜ * ＞や＜オフセット＞変数を指定しない場合には、階層リストはBLOBの最初に格納され、それ以前にそこにあった内容を上書きします。これに合わせてBLOBのサイズも調整されます。

＜オフセット＞変数を渡した場合には、階層リストはBLOB内のオフセット(ゼロから始まります)に書き込まれます。階層リストを書き込む位置に関わらず、BLOBのサイズは渡した位置(および必要な場合には階層リストのサイズ)にしたがって増加します。現在書き込んでいるバイト以外の新しく割り当てられるバイトは、ゼロに初期化されます。

呼び出し後、＜オフセット＞変数は返され、書き込まれたバイト数分だけ増分されます。したがって、その同じ変数を別のBLOB書き込みコマンドにも使用して別の変数やリストを書き込むことができます。

警告：BLOBを使用して階層リストを格納すると、階層リストは4D内部形式を使用してBLOBに格納されるため、格納されたBLOBの内容を読み出すにはBLOB to list関数を使用しなければなりません。

呼び出し後、階層リストが正常に格納された場合には、OK変数は1に設定されます。階層リストを格納するために必要なメモリがない、などの理由で処理が実行できなかった場合には、OK変数は0に設定されます。

プラットフォームからの独立性に関する注意点：

LIST TO BLOBコマンドおよび **BLOB to list**関数は、4D内部形式を使用してBLOB内に格納された変数を処理します。このメリットとして、これら2つのコマンドを使用していれ

ばプラットフォーム間でのバイトのスワップに配慮する必要がありません。つまり、これらのコマンドのうちいずれかを使用してWindows上で作成されたBLOBはMacintosh上でも使用でき、その逆も可能です。

▼ **BLOB to list**関数の例を参照してください。

参照：BLOB to list、BLOB TO VARIABLE、VARIABLE TO BLOB

BLOB to list

BLOB to list (blob {;オフセット}) → リスト参照番号

引数	タイプ		説明
blob	BLOB	→	階層リストを含んだBLOB
オフセット	数値	→	BLOB内でのオフセット(バイト単位)
		←	読み込み後の新規オフセット
関数の返す値	リスト参照番号	←	新しく作成されたリストの参照番号

BLOB to list関数は、オプション引数＜オフセット＞で指定されたバイトオフセット(ゼロから開始します)にある＜blob＞に格納されているデータを使用して新しい階層リストを作成し、この新しいリストのリスト参照番号を戻します。

BLOBデータはコマンドと整合性を保っていなければなりません。通常、**LIST TO BLOB**コマンドを使用して入力したBLOBを使用します。

オプション引数＜オフセット＞を指定しない場合には、変数データはBLOBの最初から読み込まれます。複数の変数やリストが格納されているBLOBを操作する場合には、＜オフセット＞の他に、数値変数も渡さなければなりません。

呼び出しの前に、この数値変数を適切なオフセットに設定します。呼び出しの後で、この同じ数値変数はBLOB内に格納されている次の変数のオフセットを戻します。

呼び出し後、階層リストが正常に作成された場合には、OK変数は1に設定されます。階層リストを作成するために必要なメモリがない、などの理由で処理が実行できなかった場合には、OK変数は0に設定されます。

プラットフォームからの独立性に関する注意点：

BLOB to list関数および **LIST TO BLOB**関数は、4D内部形式を使用してBLOB内に格納された変数を処理します。このメリットとして、これら2つのコマンドを使用していればプラットフォーム間でのバイトのスワップに配慮する必要がありません。つまり、これらのコマンドのうちいずれかを使用してWindows上で作成されたBLOBはMacintosh上でも使用でき、その逆も可能です。

例：

この例では、データ入力フォームが画面に表示される前に、このフォームのフォームメソッドがBLOBフィールドからリストを抽出し、データ入力が検証されるとこのリストをBLOBフィールドに再び格納します：

```
` [テーブル1];"入力"フォームのフォームメソッド
Case of
   \ (Form event=On Load)
      hList:=BLOB to list([テーブル1]アイデア)
      If (OK=1)
         hList:=New list
      End if
   \ (Form event=On Unload)
      CLEAR LIST(hList;*)
   \ (bValidate=1)
      LIST TO BLOB(hList ; [テーブル1]アイデア)
End case
```

参照：LIST TO BLOB

システム変数またはシステムセット

変数が正常に格納された場合には、OK変数は1に設定されます。それ以外の場合には、OK変数は0に設定されます。

INTEGER TO BLOB

INTEGER TO BLOB (整数 ; blob ; バイト順序 {;オフセット | *})

引数	タイプ		説明
整数	数値	→	BLOBの中に書き込む整数の値
blob	BLOB	→	整数の値を受信するBLOB
バイト順序	数値	→	0=ネイティブバイト順 1=Macintoshバイト順 2=PCバイト順
オフセット	*	変数	*	→	値を追加するためのBLOB内でのオフセット(バイト単位)または *
		←	* でなければ、書き込み後の新しいオフセット

INTEGER TO BLOBコマンドは、引数＜blob＞に2バイトの＜整数＞の値を書き込みます。

引数＜バイト順序＞は、2バイトの＜整数＞の値が書き込まれる際のバイト順序を決定します。4th Dimensionにある次の定義済み定数のうちいずれか1つを渡します：

定数	タイプ	値
Native byte ordering	倍長整数	0
Macintosh byte ordering	倍長整数	1
PC byte ordering	倍長整数	2

プラットフォームからの独立性に関する注意点：

MacintoshとPCのプラットフォーム間でBLOBを変換する場合には、このコマンドを使用する時にバイトのスワップを管理するかどうかは任意です。

オプション引数＜ * ＞を指定した場合には、2バイトの整数rの値がBLOBに追加され、これに合わせてBLOBのサイズも拡張されます。オプション引数＜ * ＞を使用すれば、BLOBがメモリ容量内であれば、整数、倍長整数、実数、またはテキストの値 (他のBLOB コマンドを参照してください) をいくつでも順番にBLOBの中に格納することができます。

オプション引数＜ * ＞や＜オフセット＞変数を指定しない場合には、2バイトの整数の値はBLOBの最初に格納され、それ以前にそこにあった内容を上書きします。これに合わせてBLOBのサイズも調整されます。

＜オフセット＞変数を渡した場合には、2バイトの整数の値はBLOB内のオフセット(ゼロから始まります)に書き込まれます。2バイトの整数の値を書き込む位置に関わらず、BLOBのサイズは渡した位置にしたがって(必要に応じてさらに最大2バイトまで)増加します。現在書き込んでいるバイト以外の新しく割り当てられるバイトは、ゼロに初期化されます。

呼び出し後、＜オフセット＞変数は返され、書き込まれたバイト数分だけ増分されます。従って、その同じ変数を別のBLOB書き込みコマンドにも使用して別の値を書き込むことができます。

例：
1. 次のコードを実行すると、

```
INTEGER TO BLOB (0x0206 ; vxBlob)
```

■ vxBlobのサイズは、2バイトです。
■ Macintoshでは、vxBLOB{0} = $02およびvxBLOB{1} = $06
■ PCでは、vxBLOB{0} = $06およびvxBLOB{1} = $02

2. 次のコードを実行すると、

```
INTEGER TO BLOB (0x0206 ; vxBlob ; Macintosh byte ordering)
```

■ vxBlobのサイズは、2バイトです。
■ すべてのプラットフォームで、vxBLOB{0} = $02 およびvxBLOB{1} = $06

3. 次のコードを実行すると、

```
INTEGER TO BLOB (0x0206 ; vxBlob ; PC byte ordering)
```

■ vxBlobのサイズは、2バイトです。
■ すべてのプラットフォームで、 vxBLOB{0} = $06およびvxBLOB{1} = $02

4. 次のコードを実行すると、

```
SET BLOB SIZE (vxBlob ; 100)
INTEGER TO BLOB (0x0206 ; vxBlob ; PC byte ordering;*)
```

■ vxBlobのサイズは、102バイトです。
■ すべてのプラットフォームで、vxBLOB{100} = $06およびvxBLOB{101} = $02
■ BLOBの他のバイトは変更されません。

5. 次のコードを実行すると、

```
SET BLOB SIZE (vxBlob ; 100)
vlOffset:=50
INTEGER TO BLOB (518 ; vxBlob ; Macintosh byte ordering ; vlOffset)
```

■ vxBlobのサイズは100バイトです。
■ すべてのプラットフォームでvxBLOB{50} = $02 およびvxBLOB{51} = $06
■ BLOBの他のバイトは変更されません。
■ 変数vlOffsetは2だけ増分されました。

参照：BLOB to integer、BLOB to longint、BLOB to real、BLOB to text、LONGINT TO BLOB、REAL TO BLOB、TEXT TO BLOB

LONGINT TO BLOB

LOMGINT TO BLOB (倍長整数 ; blob ; バイト順序 {;オフセット | *})

引数	タイプ		説明
倍長整数	数値	→	BLOBの中に書き込む倍長整数の値
blob	BLOB	→	倍長整数の値を受信するBLOB
バイト順序	数値	→	0=ネイティブバイト順 1=Macintoshバイト順 2=PCバイト順
オフセット	*	変数	*	→	値を追加するためのBLOB内でのオフセット(バイト単位)または *
		←	* でなければ、書き込み後の新しいオフセット

LONGINT TO BLOBコマンドは、引数＜blob＞に4バイトの＜倍長整数＞の値を書き込みます。

引数＜バイト順序＞は、4バイトの＜倍長整数＞の値が書き込まれる際のバイト順序を決定します。4th Dimensionにある次の定義済み定数のうちいずれか1つを渡します：

定数	タイプ	値
Native byte ordering	倍長整数	0
Macintosh byte ordering	倍長整数	1
PC byte ordering	倍長整数	2

プラットフォームからの独立性に関する注意点：

MacintoshとPCのプラットフォーム間でBLOBを変換する場合には、このコマンドを使用する時にバイトのスワップを管理するかどうかは任意です。

オプション引数＜ * ＞を指定した場合には、4バイトの倍長整数の値がBLOBに追加され、これに合わせてBLOBのサイズも拡張されます。オプション引数＜ * ＞を使用すれば、BLOBがメモリ容量内であれば、整数、倍長整数、実数、またはテキストの値 (他のBLOB コマンドを参照してください) をいくつでも順番にBLOBの中に格納することができます。

オプション引数＜ * ＞や＜オフセット＞変数を指定しない場合には、4バイトの倍長整数の値はBLOBの最初に格納され、それ以前にそこにあった内容を上書きします。これに合わせてBLOBのサイズも調整されます。

＜オフセット＞変数を渡した場合には、4バイトの整数の値はBLOB内のオフセット(ゼロから始まります)に書き込まれます。4バイトの整数の値を書き込む位置に関わらず、BLOBのサイズは渡した位置にしたがって(必要に応じてさらに最大4バイトまで)増加します。現在書き込んでいるバイト以外の新しく割り当てられるバイトは、ゼロに初期化されます。

呼び出し後、＜オフセット＞変数は返され、書き込まれたバイト数分だけ増分されます。従って、その同じ変数を別のBLOB書き込みコマンドにも使用して別の値を書き込むことができます。

例：
1. 次のコードを実行すると、

LONGINT TO BLOB (0x01020304 ; vxBlob)

■ vxBlobのサイズは、4バイトです。
■ Macintoshでは、vxBLOB{0}=$01、vxBLOB{1}=$02、vxBLOB{2}=$03、vxBLOB{3}=$04。
■ PCでは、vxBLOB{0}=$04、vxBLOB{1}=$03、vxBLOB{2}=$02、vxBLOB{3}=$01。

2. 次のコードを実行すると、

LONGINT TO BLOB (0x01020304 ; vxBlob ; Macintosh byte ordering)

■ vxBlobのサイズは、4バイトです。
■ すべてのプラットフォームで、vxBLOB{0}=$01、vxBLOB{1}=$02、vxBLOB{2}=$03、vxBLOB{3}=$04。

3. 次のコードを実行すると、

LONGINT TO BLOB (0x01020304 ; vxBlob ; PC byte ordering)

■ vxBlobのサイズは、4バイトです。
■ すべてのプラットフォームで、vxBLOB{0}=$04、vxBLOB{1}=$03、vxBLOB{2}=$02、vxBLOB{3}=$01。

4. 次のコードを実行すると、

```
SET BLOB SIZE (vxBlob ; 100)
LONGINT TO BLOB (0x01020304 ; vxBlob ; PC byte ordering ; *)
```

■ vxBlobのサイズは、104バイトです。
■ すべてのプラットフォームで、vxBLOB{100}=$04、vxBLOB{101}=$03、vxBLOB{102}=$02、vxBLOB{103}=$01。
■ BLOBの他のバイトは、変更されません。

5. 次のコードを実行すると、

```
SET BLOB SIZE (vxBlob;100)
vlOffset:=50
LONGINT TO BLOB (0x01020304 ; vxBlob ; Macintosh byte ordering ; vlOffset)
```

■ vxBlobのサイズは、100Kです。
■ すべてのプラットフォームで、vxBLOB{50}=$01、vxBLOB{51}=$02、vxBLOB{52}=$03、vxBLOB{53}=$04。
■ BLOBの他のバイトは、変更されません。
■ 変数vlOffsetは、4だけ増分されました。

参照：BLOB to integer、BLOB to longint、BLOB to real、BLOB to text、INTEGER TO BLOB、REAL TO BLOB、TEXT TO BLOB

REAL TO BLOB

REAL TO BLOB (**実数 ; blob ; 実数形式 {;オフセット | *}**)

引数	タイプ		説明
実数	数値	→	BLOBの中に書き込む実数の値
blob	BLOB	→	実数の値を受信するBLOB
実数形式	数値	→	0=ネイティブ実数形式 1=拡張実数形式 2=Macintosh倍精度実数形式 3=PC倍精度実数形式
オフセット	*	変数	*	→	値を追加するためのBLOB内でのオフセット(バイト単位)または *
		←	* でなければ、書き込み後の新しいオフセット

REAL TO BLOBコマンドは、引数＜blob＞に＜実数＞の値を書き込みます。

引数＜実数フォーマット＞は、内部形式とバイト順序を決定します。4th Dimensionにある次の定義済み定数のうちいずれか1つを渡します。：

定数	タイプ	値
Native real format	倍長整数	0
Extended real format	倍長整数	1
Macintosh double real format	倍長整数	2
PC double real format	倍長整数	3

プラットフォームからの独立性に関する注意点：

MacintoshとPCのプラットフォーム間でBLOBを変換する場合には、このコマンドを使用する時にバイトのスワップを管理するかどうかは任意です。

オプション引数＜ * ＞を指定した場合には、実数値がBLOBに追加され、これに合わせてBLOBのサイズも拡張されます。オプション引数＜ * ＞を使用すれば、BLOBがメモリ容量内であれば、整数、倍長整数、実数、またはテキストの値 (他の BLOB コマンドを参照してください) をいくつでも順番にBLOBの中に格納することができます。

オプション引数＜ * ＞や＜オフセット＞変数を指定しない場合には、実数値はBLOBの最初に格納され、それ以前にそこにあった内容を上書きします。これに合わせてBLOBのサイズも調整されます。

＜オフセット＞変数を渡した場合には、実数値はBLOB内のオフセット(ゼロから始まります)に書き込まれます。実数値を書き込む位置に関わらず、BLOBのサイズは渡した位置にしたがって(必要に応じてさらに最大8または10バイトまで)増加します。現在書き込んでいるバイト以外の新しく割り当てられるバイトは、ゼロに初期化されます。

呼び出し後、＜オフセット＞変数は返され、書き込まれたバイト数分だけ増分されます。従って、その同じ変数を別のBLOB書き込みコマンドにも使用して別の値を書き込むことができます。

例：
1. 次のコードを実行すると、

```
C_REAL (vrValue)
vrValue := ...
REAL TO BLOB (vrValue ; vxBlob)
```

■ PCおよびPower Macintoshでは、vxBlobのサイズは8バイトです。

■ 68KのMacintoshでは、vxBlobのサイズは10バイトです。

2. 次のコードを実行すると、

```
C_REAL (vrValue)
vrValue := ...
REAL TO BLOB (vrValue ; vxBlob ; Extended real format)
```

■ すべてのプラットフォームで、vxBlobのサイズは10バイトです。

3. 次のコードを実行すると、

```
C_REAL (vrValue)
vrValue := ...
REAL TO BLOB (vrValue ; vxBlob ; Macintosh Double real format)
```

■ すべてのプラットフォームで、vxBlobのサイズは8バイトです。

4. 次のコードを実行すると、

```
SET BLOB SIZE (vxBlob;100)
C_REAL (vrValue)
vrValue := ...
REAL TO BLOB (vrValue ; vxBlob ; Windows Double real format)
```

■ すべてのプラットフォームで、vxBlobのサイズは8バイトです。

5. 次のコードを実行すると、

```
SET BLOB SIZE (vxBlob ; 100)
REAL TO BLOB (vrValue ; vxBlob ; Extended real format ; *)
```

■ すべてのプラットフォームで、vxBlobのサイズは110バイトです。

■ すべてのプラットフォームで、実数値はバイト番号100～109にあるバイトに格納されます。

■ BLOBの他のバイトは、変更されません。

6. 次のコードを実行すると、

```
SET BLOB SIZE (vxBlob;100)
C_REAL (vrValue)
vrValue := ...
vlOffset:=50
REAL TO BLOB (518 ; vxBlob ; Windows Double real format ; vlOffset)
```

■ すべてのプラットフォームで、vxBlobのサイズは100バイトです。

■ すべてのプラットフォームで、実数値はバイト番号50～57にあるバイトに格納されます。

■ BLOBの他のバイトは、変更されません。

■ 変数vlOffsetは、8だけ増分されました。

参照：BLOB to integer、BLOB to longint、BLOB to real、BLOB to text、INTEGER TO BLOB、LONGINT TO BLOB、TEXT TO BLOB

TEXT TO BLOB

TEXT TO BLOB (**テキスト ; blob ; テキスト形式 {;オフセット | *}**)

引数	タイプ		説明
実数	数値	→	BLOBの中に書き込むテキストの値
blob	BLOB	→	テキストの値を受信するBLOB
テキスト形式	数値	→	0=ネイティブ実数形式 1=C言語文字列 2=Pascal言語文字列 3=長さ属性付きテキスト 4=長さ属性なしテキスト
オフセット | *	変数 | *	→	値を追加するためのBLOB内でのオフセット(バイト単位)または *
		←	* でなければ、書き込み後の新しいオフセット

TEXT TO BLOBコマンドは、引数＜テキスト＞の値を＜blob＞の中に書き込みます。

引数＜テキスト形式＞は、＜テキスト＞の値が書き込まれる際の内部形式を決定します。4th Dimensionにある次の定義済み定数のうちいずれか1つを渡します。

定数	タイプ	値
C string	倍長整数	0
Pascal string	倍長整数	1
Text with length	倍長整数	2
Text without length	倍長整数	3

次の表は、これらの内部形式それぞれについて説明しています。

テキスト形式	説明と例
C言語文字列	テキストはNULL文字(ASCII コード $00)で終了します。 "" → $00 "Hello World!" → $48 65 6C 6C 6F 20 57 6F 72 6C 64 21 00
Pascal 言語文字列	テキストの前には 1バイト長が追加されます。 "" → $00 "Hello World!" → $0C 48 65 6C 6C 6F 20 57 6F 72 6C 64 21
長さ属性付きテキスト	テキストの前には2バイトの長さ属性が追加されます。 "" → $00 00 "Hello World!" → $00 0C 48 65 6C 6C 6F 20 57 6F 72 6C 64 21
長さ属性なしテキスト	テキストは文字列だけで構成されます。 "" → データなし "Hello World!" → $48 65 6C 6C 6F 20 57 6F 72 6C 64 21

注：このコマンドは、テキスト(C_TEXTコマンドで宣言されます)および文字列(C_STRINGコマンドで宣言されます)の両方の式に対応しています。テキスト変数には最大で32,000文字、文字列変数にはその宣言にある文字の数だけ、最大で255文字まで含むことができることを覚えておいてください。

オプション引数＜＊＞を指定した場合には、テキストの値がBLOBに追加され、これに合わせてBLOBのサイズも拡張されます。オプション引数＜＊＞を使用すれば、BLOBがメモリ容量内であれば、整数、倍長整数、実数、またはテキストの値 (他の BLOB コマンドを参照してください) をいくつでも順番にBLOBの中に格納することができます。

オプション引数＜＊＞や＜オフセット＞変数を指定しない場合には、テキストの値はBLOBの最初に格納され、それ以前にそこにあった内容を上書きします。これに合わせてBLOBのサイズも調整されます。

＜オフセット＞変数を渡した場合には、テキストの値はBLOB内のオフセット(ゼロから始まります)に書き込まれます。テキストの値を書き込む位置に関わらず、BLOBのサイズは渡した位置にしたがって(必要に応じてさらに最大8または10バイトまで)増加します。現在書き込んでいるバイト以外の新しく割り当てられるバイトは、ゼロに初期化されます。

呼び出し後、＜オフセット＞変数は返され、書き込まれたバイト数分だけ増分されます。従って、その同じ変数を別のBLOB書き込みコマンドにも使用して別の値を書き込むことができます。

例：

次のコードを実行すると、

```
SET BLOB SIZE (vxBlob ; 0)
C_TEXT (vtValue)
vtValue := "Hello World!" ` 「vtValue」変数の長さは12バイト
TEXT TO BLOB (vtValue ; vxBlob) `BLOBのサイズは13バイトになる
TEXT TO BLOB (vtValue ; vxBlob;C string) ``BLOBのサイズは13バイトになる
TEXT TO BLOB (vtValue ; vxBlob;Pascal string) `BLOBのサイズは13バイトになる
TEXT TO BLOB (vtValue ; vxBlob;Text with length) ``BLOBのサイズは14バイトになる
TEXT TO BLOB (vtValue ; vxBlob;Text without length) ``BLOBのサイズは12バイトになる
```

参照：BLOB to integer、BLOB to longint、BLOB to real,、BLOB to text、INTEGER TO BLOB、LONGINT TO BLOB、REAL TO BLOB

BLOB to integer

BLOB to integer (**blob ; バイト順序 {;オフセット}**) **→ 数値**

引数	タイプ		説明
blob	BLOB	→	整数の値を受信するBLOB
バイト順序	数値	→	0=ネイティブバイト順 1=Macintoshバイト順 2=PCバイト順
オフセット	変数	→ ←	BLOB内でのオフセット(バイト単位で表す) 読み込み後の新しいオフセット
関数の返す値	数値	←	2バイトの整数値

BLOB to integerコマンドは、引数＜blob＞から読み込まれた2バイトの整数の値を返します。

引数＜バイト順序＞は、2バイトの＜整数＞の値が書き込まれる際のバイト順序を決定します。4th Dimensionにある次の定義済み定数のうちいずれか1つを渡します：

定数	タイプ	値
Native byte ordering	倍長整数	0
Macintosh byte ordering	倍長整数	1
PC byte ordering	倍長整数	2

プラットフォームからの独立性に関する注意点：

MacintoshとPCのプラットフォーム間でBLOBを変換する場合には、このコマンドを使用する時にバイトのスワップを管理するかどうかは任意です。

オプション引数＜オフセット＞変数を渡した場合には、2バイトの整数の値はBLOB内のオフセット(ゼロから始まります)に読み込まれます。オプション引数＜オフセット＞変数を指定しなかった場合には、BLOBの最初の2バイトが読み込まれます。

注：ゼロから「BLOBのサイズ - 2」の範囲のオフセット(バイト単位)の値を渡す必要があります。このようにしない場合には、この関数の結果は予期しない値になります。

呼び出し後、変数は読み込まれたバイト数分だけ増分されます。従って、その同じ変数を別のBLOB読み込みコマンドにも使用して別の値を読み込むことができます。

▼ 次の例では、あるBLOBから、オフセット0x200から開始して整数値を20個読み込んでいます：

```
$vlOffset:=0x200
For ($viLoop ; 0 ;19)
   $viValue:=BLOB to integer(vxSomeBlob ; PC byte ordering ; $vlOffset)
   `$viValueを使った何らかの処理を行う
End for
```

参照：BLOB to longint、BLOB to real、BLOB to text、INTEGER TO BLOB、LONGINT TO BLOB、REAL TO BLOB、TEXT TO BLOB

BLOB to longint

BLOB to longint (blob ; バイト順序 {;オフセット}) → 数値

引数	タイプ		説明
blob	BLOB	→	整数の値を受信するBLOB
バイト順序	数値	→	0=ネイティブバイト順 1=Macintoshバイト順 2=PCバイト順
オフセット	変数	→ ←	BLOB内でのオフセット(バイト単位で表す) 読み込み後の新しいオフセット
関数の返す値	数値	←	4バイトの倍長整数値

BLOB to longintコマンドは、引数＜blob＞から読み込まれた4バイトの倍長整数の値を返します。

引数＜バイト順序＞は、4バイトの＜倍長整数＞の値が書き込まれる際のバイト順序を決定します。4th Dimensionにある次の定義済み定数のうちいずれか1つを渡します：

定数	タイプ	値
Native byte ordering	倍長整数	0
Macintosh byte ordering	倍長整数	1
PC byte ordering	倍長整数	2

プラットフォームからの独立性に関する注意点：

MacintoshとPCのプラットフォーム間でBLOBを変換する場合には、このコマンドを使用する時にバイトのスワップを管理するかどうかは任意です。

オプション引数＜オフセット＞変数を渡した場合には、2バイトの整数の値はBLOB内のオフセット(ゼロから始まります)に読み込まれます。オプション引数＜オフセット＞変数を指定しなかった場合には、BLOBの最初の4バイトが読み込まれます。

注：ゼロから「BLOBのサイズ - 4」の範囲のオフセット(バイト単位)の値を渡す必要があります。このようにしない場合には、この関数の結果は予期しない値になります。

呼び出し後、変数は読み込まれたバイト数分だけ増分されます。従って、その同じ変数を別のBLOB読み込みコマンドにも使用して別の値を読み込むことができます。

▼ 次の例では、あるBLOBから、オフセット0x200から開始して倍長整数値を20個読み込んでいます：

```
$vlOffset:=0x200
For ($viLoop ; 0 ;19)
   $viValue:=BLOB to longint(vxSomeBlob ; PC byte ordering ; $vlOffset)
   `$viValueを使った何らかの処理を行う
End for
```

参照：BLOB to integer、BLOB to real、BLOB to text、INTEGER TO BLOB、LONGINT TO BLOB、REAL TO BLOB、TEXT TO BLOB

BLOB to real

BLOB to real (**blob ; 実数形式 {;オフセット}**) **→ 数値**

引数	タイプ		説明
blob	BLOB	→	整数の値を受信するBLOB
実数形式	数値	→	0=ネイティブ実数形式 1=拡張実数形式 2=Macintosh倍精度実数形式 3=PC倍精度実数形式
オフセット	変数	→ ←	BLOB内でのオフセット(バイト単位で表す) 読み込み後の新しいオフセット
関数の返す値	数値	←	実数値

BLOB to realコマンドは、引数＜blob＞から読み込まれた実数の値を返します。

引数＜実数形式＞は、内部形式とバイト順序を決定します。4th Dimensionにある次の定義済み定数のうちいずれか1つを渡します。：

定数	タイプ	値
Native real format	倍長整数	0
Extended real format	倍長整数	1
Macintosh double real format	倍長整数	2
PC double real format	倍長整数	3

プラットフォームからの独立性に関する注意点：

MacintoshとPCのプラットフォーム間でBLOBを変換する場合には、このコマンドを使用する時にバイトのスワップを管理するかどうかは任意です。

オプション引数＜オフセット＞変数を渡した場合には、2バイトの整数の値はBLOB内のオフセット(ゼロから始まります)に読み込まれます。オプション引数＜オフセット＞変数を指定しなかった場合には、BLOBの最初の8または10バイトが読み込まれます。

注：ゼロから「BLOBのサイズ - 8」または「BLOBのサイズ - 10」の範囲のオフセットの値を渡す必要があります。このようにしない場合には、この関数の結果は予期しない値になります。

呼び出し後、変数は読み込まれたバイト数分だけ増分されます。従って、その同じ変数を別のBLOB読み込みコマンドにも使用して別の値を読み込むことができます。

▼ 次の例では、あるBLOBから、オフセット0x200から開始して実数値を20個読み込んでいます：

```
$vlOffset:=0x200
For ($viLoop ; 0 ;19)
   $viValue:=BLOB to real(vxSomeBlob ; PC byte ordering ; $vlOffset)
   `$viValueを使った何らかの処理を行う
End for
```

参照：BLOB to integer、BLOB to longint、BLOB to text、INTEGER TO BLOB、LONGINT TO BLOB、REAL TO BLOB、TEXT TO BLOB

BLOB to text

BLOB to text (**blob ; テキスト形式 {;オフセット {;テキストの長さ}}**) → **テキスト**

引数	タイプ		説明
blob	BLOB	→	テキストの値を受信するBLOB
テキスト形式	数値	→	0=ネイティブ実数形式 1=C言語文字列 2=Pascal言語文字列 3=長さ属性付きテキスト 4=長さ属性なしテキスト
オフセット	変数	→	BLOB内でのオフセット(バイト単位で表す)
		←	読み込み後の新しいオフセット
テキストの長さ	数値	→	読み込まれるバイト数
関数の返す値	数値	←	テキストの値

BLOB to textコマンドは、＜blob＞から読み込まれたテキストの値を返します。

引数＜テキスト形式＞は、＜テキスト＞の値が書き込まれる際の内部形式を決定します。4th Dimensionにある次の定義済み定数のうちいずれか1つを渡します。

定数	タイプ	値
C string	倍長整数	0
Pascal string	倍長整数	1
Text with length	倍長整数	2
Text without length	倍長整数	3

次の表は、これらの内部形式それぞれについて説明しています。

テキスト形式	説明と例
C言語文字列	テキストはNULL文字(ASCII コード $00)で終了します。 "" → $00 "Hello World!" → $48 65 6C 6C 6F 20 57 6F 72 6C 64 21 00
Pascal 言語文字列	テキストの前には 1バイト長が追加されます。 "" → $00 "Hello World!" → $0C 48 65 6C 6C 6F 20 57 6F 72 6C 64 21
長さ属性付きテキスト	テキストの前には2バイトの長さ属性が追加されます。 "" → $00 00 "Hello World!" → $00 0C 48 65 6C 6C 6F 20 57 6F 72 6C 64 21
長さ属性なしテキスト	テキストは文字列だけで構成されます。 "" → データなし "Hello World!" → $48 65 6C 6C 6F 20 57 6F 72 6C 64 21

警告：読み込まれる文字の数は、引数＜テキスト形式＞によって決まります。ただし、形式が長さ属性なしテキストの場合には、オプション引数＜テキストの長さ＞に読み込まれる文字の数を必ず指定しなければなりません。それ以外の形式では、＜テキストの長さ＞は無視されるため、省略しても構いません。

テキスト変数には最高で32,000文字、文字列変数にはその宣言にある文字の数だけ、最大255文字まで含むことができることを忘れないようにします。変数に含むことのできる文字数以上のデータを読み込もうとすると、4Dは結果を変数に割り当てる時に途中で切り捨ててしまいます。

オプション引数＜オフセット＞を指定した場合には、テキストの値はBLOB内のオフセット(ゼロから始まります)で読み込まれます。オプション引数＜オフセット＞を指定しなかった場合には、＜テキスト形式＞に渡した値にしたがってBLOBの最初から読み込まれます。

長さ属性なしテキストを読み込む場合には、オプション引数＜オフセット＞を必ず渡さなければならないことに注意してください。

注：ゼロから「BLOBのサイズ - 読み込まれるテキストのサイズ」の範囲のオフセットの値を渡す必要があります。このようにしない場合には、この関数の結果は予期しない値になります。

呼び出し後、変数は読み込まれたバイト数分だけ増分されます。従って、その同じ変数を別のBLOB読み込みコマンドにも使用して別の値を書き込むことができます。

▼ 次の例では、内部形式が'STR#'リソースの内部形式とまったく同一である、架空のMacOSベースのリソースを読み込みます：

```
GET RESOURCE ('ABCD' ; viResID ; vxResData ; viMyResFile)
vlSize:=BLOB Size (vxResData)
If (vlSize>0)
   `リソースは、文字列の数を指定している2バイトの整数から開始します
   vlOffset:=0
   viNbEntries:=BLOB to integer(vxResData ; Macintosh Byte Ordering ; vlOffset)
   `リソースには、埋め込みではなく連結されたPascal言語文字列が含まれている
   For (viEntry ; 1 ; viNbEntries)
      If (vlOffset<vlSize)
         vsEntry:=BLOB to text(vxResData ; Pascal string ; vlOffset)
         ``vsEntryを使った何らかの処理を行う
      Else
         `リソースデータは無効であり、ループを終了する
         viEntry:=viNbEntries+1
      End if
   End for
End if
```

参照：BLOB to integer、BLOB to longint、BLOB to real、INTEGER TO BLOB、LONGINT TO BLOB、REAL TO BLOB、TEXT TO BLOB

INSERT IN BLOB

INSERT IN BLOB (**blob ; オフセット ; 挿入数 {;フィルタ}**)

引数	タイプ		説明
blob	BLOB	→	バイトが挿入されるBLOB
オフセット	変数	→	挿入される開始位置
		←	挿入直後の位置
挿入数	数値	→	挿入されるバイト数
フィルタ	数値	→	デフォルトのバイト値（0x00..0xFF） 省略した場合は、0x00

INSERT IN BLOBコマンドは、引数＜挿入数＞で指定されたバイト数を＜blob＞の＜オフセット＞で指定された位置に挿入します。BLOBは、＜挿入数＞で指定されたバイト数分だけ拡大されます。

オプション引数の＜フィルタ＞を指定しない場合には、BLOBに挿入されるバイトは 0x00 に設定されます。それ以外の場合には、＜フィルタ＞に渡した値に設定されます(モジューロ256_ 0..255)。

呼び出しの前に、オプション引数の＜フィルタ＞変数内に、挿入位置をBLOBの最初から相対的な位置として渡します。呼び出し後、＜フィルタ＞変数は挿入直後の位置を返します。

参照：DELETE FROM BLOB

DELETE FROM BLOB

DELETE FROM BLOB (**blob ; オフセット ; 削除数**)

引数	タイプ		説明
blob	BLOB	→	バイトが削除されるBLOB
オフセット	数値	→	削除される開始位置
削除数	数値	→	削除されるバイト数

DELETE FROM BLOBコマンドは、＜削除数＞で指定されたバイト数を＜blob＞の＜オフセット＞で指定された位置(BLOBの最初から相対的な位置として表される)から削除します。BLOBは、＜削除数＞で指定されたバイト数分だけ縮小されます。

参照：INSERT IN BLOB

COPY BLOB

COPY BLOB (コピー元BLOB ; コピー先BLOB ; コピー元オフセット ; コピー先オフセット ; コピー数)

引数	タイプ		説明
コピー元BLOB	BLOB	→	ソース元BLOB
コピー先BLOB	BLOB	→	ソース先BLOB
コピー元オフセット	変数	→	コピー元の位置
		←	コピー直後の位置
コピー先オフセット	変数	→	コピー先の位置
		←	コピー直後の位置
コピー数	数値	→	コピーされるバイト数

COPY BLOBコマンドは、引数＜コピー数＞で指定されたバイト数を＜コピー元BLOB＞から＜コピー先BLOB＞にコピーします。

コピーは＜コピー元オフセット＞で指定された位置(コピー元のBLOBの最初から相対的な位置として表されます)から開始され、＜コピー先オフセット＞で指定された位置(コピー先のBLOBの最初から相対的な位置として表されます)にコピーされます。

注：コピー先のBLOBのサイズは、必要に応じて変更できます。

呼び出し後、＜コピー元オフセット＞変数および＜コピー元オフセット＞変数は、それぞれコピー直後のコピー元のBLOBおよびコピー先のBLOBでの位置を返します。

参照：DELETE FROM BLOB、INSERT IN BLOB

ブールコマンド　16

この章では、3種類のブール関数について説明します。ブールコマンドは、「ルーチン」エディタの「Boolean」テーマ内にあります。これらの関数は、論理計算に用いられます。

True　　**False**　　**Not**

▼ 次の例は、ボタン“私のボタン”がクリックされた時に“True”を返し、クリックされなかった時には“False”を返します。ボタンがクリックされると、ボタン変数に１を代入します。この例は、ボタンの値を基にボタン変数をセットします：

```
If(私のボタン=1)                    ` ボタンがクリックされたら...
   私のボタン変数:=True             ` ボタン変数にTrueがセットされる
Else                                ` ボタンがクリックされなかったら...
   私のボタン変数:=False            ` ボタン変数にFalseがセットされる
End if
```

上の例をもっと簡単に1行で記述することができます。こちらの方が一層効率的です：

```
私のボタン変数:=(私のボタン=1)
```

さらに、次の4Dコマンドもブール値を返します。

Activated、After、Before、Before selection、Before subselection、Caps lock down、Compiled application、Deactivated, During、End selection、End subselection、In break、In footer、In header、In transaction、Is a list、Is a variable、Is in set、Is user deleted、Locked、 Macintosh command down、Macintosh control down、Macintosh option down、Modified、Modified record、Nil、Outside call、Read only state、Semaphore、Shift down、True、Undefined、User in group、Windows Alt down、Windows Ctrl down

True

True → ブール値 (True)

True関数は、ブール値のTrueを返します。

▼ 次の例は、v変数にTrueを代入します：

```
v変数:=True
```

False

False → ブール値 (False)

False関数は、ブール値のFalseを返します。

▼ 次の例は、v変数にFalseを代入します：

```
v変数:=False
```

Not

Not (ブール値) → ブール値

引数	タイプ	説明
ブール値	ブール	否定を求めるブール値

Not関数は、＜ブール値＞の否定を返します。TrueをFalseに、FalseをTrueにして返します。

▼ 次の例は、変数にTrueを代入し変数の値をFalseにした後、再びTrueにします：

```
v結果:=True                    ` v結果にTrueを代入
v結果:=Not (v結果)             ` v結果にFalseを代入
v結果:=Not (v結果)             ` v結果にTrueを代入
```

クリップボードコマンド　17

この章では、「ルーチン」エディタの「Clipboard」テーマ内にあるクリップボードコマンドについて説明します：

APPEND TO CLIPBOARD
CLEAR CLIPBOARD
GET CLIPBOARD
Get text from clipborad
GET PICTURE FROM CLIPBOARD
SET PICTURE TO CLIPBOARD
SET TEXT TO CLIPBOARD
Test clipborad

APPEND TO CLIPBOARD

APPEND TO CLIPBOARD (**データタイプ ; データ**)

引数	**タイプ**		**説明**
データタイプ	文字列	→	4バイトの文字列データタイプ
データ	BLOB	→	クリップボードに追加するデータ

APPEND TO CLIPBOARDコマンドは、引数＜データタイプ＞で指定されたデータタイプでBLOB＜データ＞内にあるデータをクリップボードに追加します。

警告：＜データタイプ＞に渡す値は、大文字と小文字が区別されます。例えば、"abcd"は"ABCD"と同じではありません。

BLOBデータがクリップボードに正しく追加された場合には、OK変数は1に設定されます。それ以外の場合には、OK変数は0に設定され、エラーが生成される可能性があります。

通常、同一データの複数のインスタンスをクリップボードに追加したり、テキストやPICT以外のタイプのデータを追加したりするときには、**APPEND TO CLIPBOARD**コマンドを使用します。クリップボードに新しいデータを追加するには、まず最初に**CLEAR CLIPBOARD**コマンドを使用してクリップボードを消去する必要があります。

消去と追加を行う場合には、以下に従います：

■ クリップボードに対してテキストを消去または追加する場合には、**SET TEXT TO CLIPBOARD**コマンドを使用します。

■ クリップボードに対してピクチャを消去または追加する場合には、**SET PICTURE TO CLIPBOARD**コマンドを使用します。

ただし、BLOBに実際にテキストまたはピクチャが含まれている場合には、**APPEND TO CLIPBOARD**コマンドを使用して、クリップボードにテキストまたはピクチャを追加できることに注意してください。

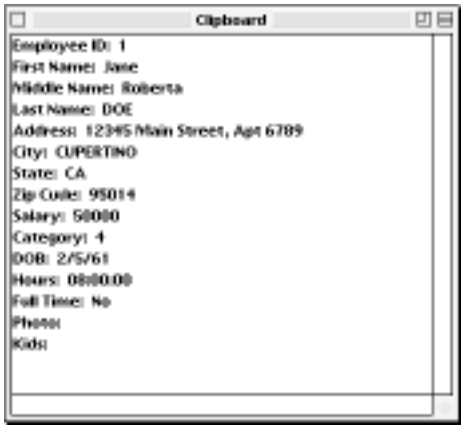
例：

クリップボードコマンドとBLOBを使用すると、一意な1つのデータよりも構造化されたデータを扱う洗練された切り取り／コピー／貼り付けの方式を構築することができます。次の例では、2つのプロジェクトメソッド「SET RECORD TO CLIPBOARD」と「GET RECORD FROM CLIPBOARD」は、クリップボードとの間でコピーするためにレコード全体を1つのデータとして扱うことができます：

```
` 「SET RECORD TO CLIPBOARD」プロジェクトメソッド
` SET RECORD TO CLIPBOARD (数値)
` SET RECORD TO CLIPBOARD (テーブル番号)
C_LONGINT($1 ; $vlField ; $vlFieldType)
C_POINTER($vpTable ; $vpField)
C_STRING(255 ; $vsDocName)
C_TEXT($vtRecordData ; $vtFieldData)
C_BLOB($vxRecordData)
` クリップボードの消去(カレントレコードがない場合は空のまま)
CLEAR CLIPBOARD
`パラメータとして渡される数値を持つテーブルへのポインタを取得する
$vpTable:=Table($1)
`そのテーブルのカレントレコードがある場合
If ((Record number($vpTable->) >=0)  | (Record number($vpTable->)=-3))
`レコードのテキストイメージを保持するテキスト変数を初期化する
  $vtRecordData:=""
  `レコードの各フィールドに対してループする
  For ($vlField ; 1 ; Count fields($1))
    `フィールドタイプを取得する
    GET FIELD PROPERTIES($1 ; $vlField ; $vlFieldType)
    `フィールドへのポインタを取得する
    $vpField:=Field($1 ; $vlField)
    `フィールドのタイプに従って、適切な方法でそのデータをコピーする(またはしない)
    Case of
      \ (($vlFieldType=Is Alpha field )  |  ($vlFieldType=Is Text ))
          $vtFieldData:=$vpField->
      \ (($vlFieldType=Is Real )  |  ($vlFieldType=Is Integer )  |  ($vlFieldType=Is LongInt )
        |  ($vlFieldType=Is Date )  |  ($vlFieldType=Is Time ))
          $vtFieldData:=String($vpField->)
      \ ($vlFieldType=Is Boolean )
          $vtFieldData:=String(Num($vpField->) ; "Yes ; ; No")
    Else
      `他のフィールドデータタイプは読み飛ばすか、無視する
      $vtFieldData:=""
    End case
```

```
      `レコードのテキストイメージを保持しているテキスト変数にフィールドデータを積算する
      $vtRecordData:=$vtRecordData + Field name($1 ; $vlField)+":"+Char(9)+$vtFieldData+ CR
      `注：このメソッドでは、CRはMacintoshではChar(13)を返し、
      `Windowsでは、Char(13)+Char(10)を返す
   End for
   `クリップボードにレコードのテキストイメージを入れる
   SET TEXT TO CLIPBOARD($vtRecordData)
   ` テンポラリ（中間）フォルダにあるスクラップファイルの名前
   $vsDocName:=Temporary folder+"スクラップ"+String(1+(Random%99))
   `スクラップファイルが存在する場合には削除する(エラーはここでテストする)
   DELETE DOCUMENT($vsDocName)
   `スクラップファイルを作成する
   SET CHANNEL(10 ; $vsDocName)
   `レコード全体をスクラップファイルに送信する
   SEND RECORD($vpTable->)
   `スクラップファイルを閉じる
   SET CHANNEL(11)
   `BLOBのなｋにスクラップファイルをロードする
   DOCUMENT TO BLOB($vsDocName ; $vxRecordData)
   `このスクラップファイルはもう必要ない
   DELETE DOCUMENT($vsDocName)
   `レコード全体のイメージをクリップボードに追加する
   `注：任意に"4Drc"をデータタイプとして使用する
   APPEND TO CLIPBOARD("4Drc" ; $vxRecordData)
   `この時点で、クリップボードには以下が含まれる
   `(1) レコードのテキストイメージ(以下の画面のとおり)
   `(2) レコードの全体イメージ(ピクチャ、サブテーブル、BLOBフィールドが含まれる)
End if
```

「SET RECORD TO CLIPBOARD」プロジェクトメソッドに[Employees]テーブルを適用する場合には、クリップボードには次に示すようにレコードのテキストイメージとレコード全体のイメージが含まれます。



次のように、「GET RECORD FROM CLIPBOARD」プロジェクトメソッドを使用して、このレコードのイメージを別のレコードに貼り付けることができます。

```
` 「GET RECORD FROM CLIPBOARD」プロジェクトメソッド
` GET RECORD FROM CLIPBOARD (数値)
` GET RECORD FROM CLIPBOARD (テーブル番号)
C_LONGINT($1 ; $vlField ; $vlFieldType ; $vlPosCR ; $vlPosColon)
C_POINTER($vpTable ; $vpField)
C_STRING(255 ; $vsDocName)
C_BLOB($vxClipboardData)
C_TEXT($vtClipboardData ; $vtFieldData)
`パラメータとして渡される数値を持つテーブルへのポインタを取得する
$vpTable:=Table($1)
`カレントレコードがある場合、
If ((Record number($vpTable->)>=0)  | (Record number($vpTable->)=-3))
   Case of
      `クリップボードにフルイメージレコードが含まれているか？
      \ (Test clipboard("4Drc")>0)
         `含んでいる場合は、クリップボードの内容を取り出す
         GET CLIPBOARD("4Drc" ; $vxClipboardData)
         `テンポラリ（中間）フォルダにあるスクラップファイルの名前
         $vsDocName:=Temporary folder+"スクラップ"+String(1+(Random%99))
         `スクラップファイルが存在する場合には削除する(エラーはここでテストする)
         DELETE DOCUMENT($vsDocName)
         `スクラップファイルの中にBLOBを保存する
         BLOB TO DOCUMENT($vsDocName ; $vxClipboardData)
         `スクラップファイルを開く
         SET CHANNEL(10 ; $vsDocName)
         `スクラップファイルからレコード全体を受信する
         RECEIVE RECORD($vpTable->)
         `スクラップファイルを閉じる
         SET CHANNEL(11)
         `このスクラップファイルはもう必要ない
         DELETE DOCUMENT($vsDocName)
         `テキストをクリップボードが含んでいるか？
      \ (Test clipboard("TEXT")>0)
         `クリップボードからテキストを取り出す
         $vtClipboardData:=Get text from clipboard
      `増分するフィールド番号を初期化する
         $vlField:=0
         Repeat
            `テキスト内の次のフィールド行を探す
            $vlPosCR:=Position(CR ; $vtClipboardData)
            If ($vlPosCR>0)
            `フィールド行を取り出す
               $vtFieldData:=Substring($vtClipboardData ; 1 ; $vlPosCR-1)
               `コロン":"があるか？
               $vlPosColon:=Position(":" ; $vtFieldData)
```

```
            If ($vlPosColon>0)
               `データタイプのみ取得する（フィールド名は削除）
               $vtFieldData:=Substring($vtFieldData ; $vlPosColon+2)
            End if
            `フィールド番号をインクリメントする
            $vlField:=$vlField+1
            `クリップボードは我々が望むより多いデータを含んでいるかもしれない...
            If ($vlField<=Count fields($vpTable))
               `フィールドタイプを取得する
               GET FIELD PROPERTIES($1 ; $vlField ; $vlFieldType)
               `フィールドポインタを取得する
               $vpField:=Field($1 ; $vlField)
               `フィールドのタイプに従って、適切な方法でそのデータを
               `コピーする(またはしない)
               Case of
                  \ (($vlFieldType=Is Alpha field ) | ($vlFieldType=Is Text ))
                     $vpField->:=$vtFieldData
                  \ (($vlFieldType=Is Real ) | ($vlFieldType=Is Integer ) |
                        ($vlFieldType=Is LongInt ))
                     $vpField->:=Num($vtFieldData)
                  \ ($vlFieldType=Is Date )
                     $vpField->:=Date($vtFieldData)
                  \ ($vlFieldType=Is Time )
                     $vpField->:=Time($vtFieldData)
                  \ ($vlFieldType=Is Boolean )
                     $vpField->:=($vtFieldData="Yes")
               Else
                  `他のフィールドデータタイプは読み飛ばすか、無視する
               End case
            Else
               ` 全フィールドが割り当てられた場合、ループを抜ける
               $vtClipboardData:=""
            End if
            ` 取り出されたテキストを削除する
            $vtClipboardData:=Substring($vtClipboardData ; $vlPosCR+Length( CR ))
         Else
            ` フィールド区切りが見つからない場合、ループを抜ける
            $vtClipboardData:=""
         End if
         ` データがあるまでループ
      Until (Length($vtClipboardData)=0)
   Else
      ALERT("レコードとして貼り付けられるデータがクリップボードにありません。")
  End case
End if
```

参照：CLEAR CLIPBOARD、SET PICTURE TO CLIPBOARD、SET TEXT TO CLIPBOARD

システム変数
BLOBデータがクリップボードに正しく追加された場合には、OKは1に設定されます。それ以外の場合には、OKは0に設定され、エラーが生成される可能性があります。

エラー処理
BLOBデータをクリップボードにアペンドするための十分なメモリがない場合には、エラーコード-108が生成されます。

CLEAR CLIPBOARD

CLEAR CLIPBOARD

引数　　タイプ　　説明

このコマンドには、引数はありません。

CLEAR CLIPBOARDコマンドは、クリップボードの内容を消去します。クリップボードに同じデータの複数のインスタンスが含まれる場合には、すべてのインスタンスが消去されます。**CLEAR CLIPBOARD**コマンドを呼び出した後、クリップボードは空になります。

APPEND TO CLIPBOARDコマンドを使用して新しいデータをクリップボードに追加する前に、**CLEAR CLIPBOARD**コマンドを1回呼び出す必要があります。これは、**APPEND TO CLIPBOARD**コマンドが新しいデータを追加する前にクリップボードを消去しないためです。

CLEAR CLIPBOARDコマンドを1回呼び出してから、**APPEND TO CLIPBOARD**コマンドを数回呼び出すと、異なるフォーマットで同じデータを切り取りまたはコピーすることができます。

一方、**SET TEXT TO CLIPBOARD**コマンドと**SET PICTURE TO CLIPBOARD**コマンドは、クリップボードにテキストデータやPICTデータを追加する前に自動的にクリップボードを消去します。

▼ 次の例は、クリップボードを消去し、次にデータをクリップボードに追加します。

```
CLEAR CLIPBOARD `クリップボードが空になっているか確認する
APPEND TO CLIPBOARD('XWKZ' ; $vxSomeData) `'XWKZ'タイプのデータを追加する
APPEND TO CLIPBOARD('SYLK' ; $vxSylkData) `Sylkデータだけデータを追加する
```

▼ **APPEND TO CLIPBOARD**コマンドの例を参照してください。

参照：APPEND TO CLIPBOARD

GET CLIPBOARD

GET CLIPBOARD (データタイプ ; データ)

引数	タイプ		説明
データタイプ	文字列	→	4バイトの文字列データタイプ
データ	BLOB	←	クリップボードから取り出されたデータ

GET CLIPBOARDコマンドは、クリップボードに存在するデータでユーザが＜データタイプ＞で指定したタイプのものをBLOBフィールドまたは変数データに返します。

警告：＜データタイプ＞に渡す値は、大文字と小文字が区別されます。例えば、"abcd"は"ABCD"と同じではありません。

データがクリップボードから正しく取り出された場合には、コマンドはOK変数を1に設定します。クリップボードが空である場合や、指定したタイプのデータが存在しない場合には、コマンドは空のBLOBを返し、OK変数を0に設定してからエラー-102を生成します。クリップボードからデータを取り出すための十分なメモリがない場合には、コマンドはOK変数を0に設定し、エラーコード-108を生成します。

▼ 2つのボタンの次のオブジェクトメソッドは、フォームに存在する配列asOptions(ポップアップメニュー、ドロップダウンリストなど)との間でデータをコピーまたは貼り付けます：

```
` 「bCopyasOptions」ボタンのオブジェクトメソッド
If (Size of array(asOptions)>0) コピーする物がある？
   ARRAY TO BLOB (asOptions ; $vxClipData) `BLOB内の配列要素を積算する
   CLEAR CLIPBOARD `クリップボードを空にする
   APPEND TO CLIPBOARD ("artx" ; asOptions) `注：任意のデータタイプが選択される
End if

` 「bPasteasOption」ボタンのオブジェクトメソッド
If (Test clipboard ("artx")>0) ` クリップボードに"artx"データが存在するか？
   GET CLIPBOARD ("artx" ; $vxClipData) `クリップボードからデータを取り出す
   BLOB TO ARRAY ($vxClipData ; asOptions) `BLOBデータを配列に登録する
   asOptions:=0 `配列用に選択した要素をリセットする
End if
```

参照：GET PICTURE FROM CLIPBOARD、Get text from clipboard、Test clipboard

システム変数

データが正しく取り出された場合には、OKは1に設定されます。それ以外の場合には、OKは0に設定され、エラーが生成されます。

エラー処理

■ データを取り出すための十分なメモリがない場合には、エラーコード-108が生成されます。
■ クリップボードに要求されたタイプのデータがない場合には、エラーコード-102が生成されます。

GET PICTURE FROM CLIPBOARD

GET PICTURE FROM CLIPBOARD (ピクチャ)

引数	タイプ		説明
ピクチャ	ピクチャ	←	クリップボードから取り出したピクチャ

GET PICTURE FROM CLIPBOARDコマンドは、クリップボード内に存在するピクチャをピクチャフィールドやピクチャ変数に返します。

ピクチャがクリップボードから正しく取り出された場合には、コマンドはOK変数を1に設定します。クリップボードが空である場合や、ピクチャが存在しない場合には、コマンドは空のピクチャを返し、OK変数を0に設定して、エラーコード-102を生成します。クリップボードからピクチャを取り出すための十分なメモリがない場合には、コマンドはOK変数を0に設定し、エラーコード-108を生成します。

▼ 次のボタンのオブジェクトメソッドは、クリップボードに存在するピクチャをフィールド[従業員]写真フィールドに割り当てます：

```
If (Test clipboard ("PICT")>0)
   GET PICTURE FROM CLIPBOARD ([従業員]写真)
Else
   ALERT ("クリップボードにピクチャはありません。")
End if
```

参照：GET CLIPBOARD、Get text from clipboard、Test clipboard

システム変数

データが正しく取り出された場合には、OKは1に設定されます。それ以外の場合には、OKは0に設定され、エラーが生成されます。

エラー処理

■ データを取り出すための十分なメモリがない場合には、エラーコード-108が生成されます。
■ クリップボードに要求されたタイプのデータがない場合には、エラーコード-102が生成されます。

Get text from clipboard

Get text from clipboard → 文字列

引数	タイプ		説明
このコマンドには、引数はありません。			
関数の返す値	文字列	←	クリップボードにテキストがあればそれを返す

Get text from clipboard関数は、クリップボードに存在するテキストを返します。

テキストがクリップボードから正しく取り出された場合には、コマンドはOK変数を1に設定します。クリップボードが空である場合や、テキストが存在しない場合には、コマンドは空の文字列を返し、OK変数を0に設定して、エラーコード-102を生成します。クリップボードからテキストを取り出すための十分なメモリがない場合には、コマンドはOK変数を0に設定し、エラーコード-108を生成します。

4th Dimensionのテキストフィールドとテキスト変数は、最大32,000文字まで含むことができます。クリップボード内にある文字が32,000文字を超える場合には、**Get text from clipboard**関数から返される結果は値を受け取るフィールドや変数に入るときに切り捨てられます。非常に大量のクリップボードテキスト内容を処理するときには、まず最初に**Test clipboard**関数でデータのサイズを確認します。次に、そのテキストが32,000文字を超えている場合には、**Get text from clipboard**関数ではなく**GET CLIPBOARD**コマンドを使用します。

▼ 次の例は、クリップボード内のテキストがどの程度のサイズかをテストしてから、データのサイズに合わせて、テキストまたはBLOBとしてクリップボードからテキストを取り出します：

```
$vlSize:=Test clipboard ("TEXT")
Case of
   \ ($vlSize<=0)
      ALERT ("クリップボードにテキストはありません。")
   \ ($vlSize<=32000)
      $vtClipData:=Get text from clipboard
      If (OK=1)
         ` テキストを使って何らかの処理を行う
      End if
   \ ($vlSize>32000)
      GET CLIPBOARD ("TEXT" ; $vxClipData)
      If (OK=1)
         ` BLOBを使った何らかの処理を行う
      End if
End case
```

参照：GET CLIPBOARD、GET PICTURE FROM CLIPBOARD、Test clipboard

システム変数

データが正しく取り出された場合には、OKは1に設定されます。それ以外の場合には、OKは0に設定され、エラーが生成されます。

エラー処理

■ データを取り出すための十分なメモリがない場合には、エラーコード-108が生成されます。
■ クリップボードに要求されたタイプのデータがない場合には、エラーコード-102が生成されます。

SET PICTURE TO CLIPBOARD

SET PICTURE TO CLIPBOARD (ピクチャ)

引数	タイプ		説明
ピクチャ	ピクチャ	→	クリップボードに入れるコピーピクチャ

SET PICTURE TO CLIPBOARDコマンドは、クリップボードを消去し、クリップボードの中に<ピクチャ>で渡したピクチャのコピーをクリップボードに入れます。

ピクチャをクリップボードに入れた後に、**GET PICTURE FROM CLIPBOARD**コマンドを使用するか、**GET CLIPBOARD ("PICT" ; ...)**を呼び出すことでそのピクチャを取り出すことができます。

ピクチャが正しくクリップボードに入れられた場合には、OK変数は1に設定されます。クリップボードにピクチャのコピーを入れるための十分なメモリがない場合には、OK変数は0に設定されますが、エラーは生成されません。

▼ 次の例はフローティングウィンドウを使用して、配列asEmployeeNameを含むフォームを表示します。この配列は[従業員]テーブルの従業員名を一覧表示したものです。従業員の名字をクリックすることにより、その従業員の顔写真をクリップボードにコピーします。この配列用のオブジェクトメソッドを次に示します：

```
If (asEmployeeName#0)
   QUERY ([従業員];[従業員]名字=asEmployeeName{asEmployeeName})
   If (Picture size ([従業員]写真)>0)
      SET PICTURE TO CLIPBOARD ([従業員]写真) ` 従業員の顔写真をコピーする
   Else
      CLEAR CLIPBOARD ` 顔写真がないか、またはレコードが見つからない場合
   End if
End if
```

参照：APPEND TO CLIPBOARD、GET PICTURE FROM CLIPBOARD

システム変数またはシステムセット

ピクチャのコピーが正しくクリップボードに入れられた場合には、OK変数は1に設定されます。

SET TEXT TO CLIPBOARD

SET TEXT TO CLIPBOARD (テキスト)

引数	タイプ		説明
テキスト	テキスト	→	クリップボードに入れるコピーテキスト

SET TEXT TO CLIPBOARDコマンドは、クリップボードを消去し、クリップボードの中に<テキスト>で渡したテキストのコピーをクリップボードに入れます。

テキストをクリップボードに入れた後に、**Get text from clipboard**関数を使用するか、**GET CLIPBOARD ("テキスト" ; ...)**を呼び出すことでそのテキストを取り出すことができます。

テキストが正しくクリップボードに入れられた場合には、OK変数は1に設定されます。クリップボードにテキストのコピーを入れるための十分なメモリがない場合には、OK変数は0に設定されますが、エラーは生成されません。

4th Dimensionのテキスト式は最大32,000文字まで含めることができます。それよりも大きなテキスト値をコピーするには、テキストをBLOB内に積算して、**CLEAR CLIPBOARD**コマンドを呼び出してから**APPEND TO CLIPBOARD("テキスト" ; ...)**を呼び出してください。

▼ APPEND TO CLIPBOARDコマンドの例を参照してください。

参照：APPEND TO CLIPBOARD、Get text from clipboard

システム変数またはシステムセット

テキストが正しくクリップボードに入れられた場合には、OK変数は1に設定されます。

Test clipboard

Test clipboard (データタイプ)

引数	タイプ		説明
データタイプ	文字列	→	4バイトの文字列データタイプ
関数の返す値	数値	←	クリップボードに格納されたデータサイズ(バイト単位)、またはエラーコード

Test clipboard関数では、引数<データタイプ>にユーザが渡したデータがクリップボード内に存在するかどうかをテストできます。

警告：<データタイプ>に渡す値は、大文字と小文字が区別されます。例えば、"abcd"は"ABCD"と同じではありません。

クリップボードが空である場合や、指定したタイプのデータが存在しない場合には、コマンドはエラーコード-102を生成します(下記の定義済定数の表を参照)。クリップボードに指定したタイプのデータが存在する場合には、コマンドはこのデータのサイズをバイト単位で返します。

ユーザが対象としたタイプのデータがクリップボードに存在することを検出した後は、次のいずれか1つのコマンドを使用すると、そのデータをクリップボードから取り出すことができます。

■ クリップボードにあるデータがTEXTタイプの場合には、テキスト値を返す**Get text from clipboard**関数か、BLOBにテキストを返す**GET CLIPBOARD**コマンドを使用してそのデータを取得できます。
■ クリップボードにあるデータがPICTタイプの場合には、ピクチャをピクチャフィールドまたは変数に返す**GET PICTURE FROM CLIPBOARD**コマンドか、ピクチャをBLOBに返す**GET CLIPBOARD**コマンドを使用してそのデータを取得できます。

■ 上記以外の任意のデータタイプに対しては、データをBLOBに返す**GET CLIPBOARD**コマンドを使用します。

4th Dimensionには、次のようなあらかじめ定義された定数があります。

定数	タイプ	値
No such data in clipboard	倍長整数	-102
Text data	文字列	テキスト
Picture data	文字列	PICT

例：
(1) 次のコードは、クリップボードにピクチャが存在するかどうかをテストし、存在する場合にはそのピクチャを4D変数にコピーします：

```
If (Test clipboard (Picture data) > 0) ` クリップボードにピクチャがあるか？
   `ピクチャがある場合、クリップボードからピクチャを取り出す
   GET PICTURE FROM CLIPBOARD ($vPicVariable)
Else
   ALERT("クリップボードにピクチャはありません。")
End if
```

(2) 通常、アプリケーションはテキストタイプまたはPICTタイプのデータをクリップボードに切り取ってコピーします。これは、ほとんどのアプリケーションがこの2つの標準データタイプを認識しているためです。ただし、アプリケーションは1つのデータを異なるフォーマットの複数のインスタンスでクリップボードに追加することができます。例えば、スプレッドシートの一部を切り取りまたはコピーするたびに、スプレッドシートアプリケーションはそのデータをSYLKフォーマットやTEXTフォーマットの他に、仮定的な「SPSH」フォーマットでも追加することができます。「SPSH」インスタンスにはアプリケーションのデータ構造を使用してフォーマットされたデータが含まれているはずです。SYLKフォームには同じデータが含まれているはずですが、それは他の多くのスプレッドシートプログラムでわかるSYLKフォーマットを使用しています。最後に、TEXTフォーマットはSYLKや仮定的な「SPSH」フォーマットに余分な情報を含まずに同一のデータを含むことができます。この時点で、4th Dimensionとその仮定的なスプレッドシートアプリケーションで切り取り／コピー／貼り付けルーチンを記述している間に、「SPSH」フォーマットの内容を知り、SYLKデータの解析準備ができた場合には、以下のようなコードを作成することができます：

```
Case of
      `まず最初に、クリップボードに仮定的なスプレッドシートアプリケーションからの
       データがあるのかをチェックする
   \ (Test clipboard ('SPSH') > 0)
      ` ...
      `次に、クリップボードにSylkデータが存在するかどうかをチェックする
   \ (Test clipboard ('SYLK') > 0)
      ` ...
      `最後に、クリップボードにTextデータが存在するかどうかをチェックする
   \ (Test clipboard ('TEXT') > 0)
` ...
End case
```

つまり、オリジナルの情報のほとんどを含むデータのインスタンスをクリップボードから取り出そうとしています。

(3) **APPEND TO CLIPBOARD**コマンドの例を参照してください。

参照：GET CLIPBOARD、GET PICTURE FROM CLIPBOARD、Get text from clipboard

通信コマンド　18

この章では、「ルーチン」エディタの「Communications」テーマ内にある通信コマンドについて説明します。この章のコマンドは、文書ファイルやシリアルポートから、データの書き出しと読み込みを行います。これらのコマンドのほとんどがディスク上の文書と共に作動します。文書上での動作に関する詳細は、第52章の「システム文書コマンド」を参照してください。

RECEIVE BUFFER
RECEIVE PACKET
RECEIVE RECORD
RECEIVE VARIABLE
SEND PACKET
SEND RECORD
SEND VARIABLE
SET CHANNEL
SET TIMEOUT
USE ASCII MAP

SET CHANNEL

SET CHANNEL (ポート ; セットアップ)

引数	タイプ	説明
ポート	数値	使用するポート
セットアップ	数値	受信データを格納する変数

SET CHANNEL (処理 ; {文書})

引数	タイプ	説明
処理	数値	ファイル処理コード
文書	文字列	処理対象の文書ファイル

SET CHANNELコマンドには、2つの形式があります。第1の形式は、シリアルポートを開きます。第2の形式は、文書ファイルを開きます。このコマンドは、同時に1つのポートまたは1つの文書ファイルしか開くことができません。

SET CHANNELコマンドの第1の形式は、シリアルポートを開きます。同時にプロトコルや、ポートに対するその他の情報を設定します。

データの送信は、**SEND PACKET**コマンド、**SEND RECORD**コマンド、**SEND VARIABLE**コマンドで行います。データの受信は、**RECEIVE BUFFER**コマンド、**RECEIVE PACKET**コマンド、**RECEIVE RECORD**コマンド、**RECEIVE VARIABLE**コマンドで行います。

引数＜ポート＞で、ポートとプロトコルを指定します。ポートとプロトコルは、次の表に示した＜ポート＞の値を加算して指定します。例えば、モデムポートでXON/XOFFを使用する場合には、1+20=21と計算します。引数＜ポート＞の値として21を指定します。

設定対象	ポート	設定
ポート	**0**	**プリンタ**
	1	**モデム**
プロトコル	**0**	**なし**
	20	**XON/XOFF**
	30	**DTR**

引数＜セットアップ＞で、転送速度、データビット、ストップビット、パリティを指定します。次の表に示した＜セットアップ＞の値を加算して指定します。例えば、転送速度を1200ボー、データビットを8ビット、ストップビットを1、パリティをなし、と設定する場合には、94+3072+16384+0と計算します。引数＜セットアップ＞の値として19550を指定します。

制御対象	セットアップ	設定
転送速度	380	300
	189	600
	94	1200
	62	1800
	46	2400
	30	3600
	22	4800
	14	7200
	10	9600
	4	19200
	0	57600
データビット	0	5
	2048	6
	1024	7
	3072	8
ストップビット	16384	1
	-32768	1.5
	-16384	2
パリティ	0	なし
	4096	奇数
	12288	偶数

注：転送速度の算出方法は、次の式を使用します：

セットアップ=(57600 / 転送速度) * 2) - 2

SET CHANNELコマンドの第2の形式は、文書ファイルの作成やオープンおよび、閉じます。文書ファイルコマンドと異なり、同時に1つの文書ファイルしか開くことができません。文書ファイルは、読み込みと書き込みの両方が可能です。文書コマンドに関する詳細は、第45章を参照してください。

引数＜処理＞で、オプション引数＜文書＞で指定した文書ファイルに対する処理を指定します。次ページの表に、＜処理＞の値と、その結果として文書ファイルに対して行われる処理を示します。“ 処理 ” の欄に引数＜処理＞の値を示します。“ 文書 ” の欄に＜文書＞の値を示します。“ 結果 ” の欄は指定した＜処理＞の結果を示します。

例えば、「ファイルを開く」ダイアログボックスを表示して、テキストファイルを開く場合には、次のように記述します：

SET CHANNEL (13 ; "")

処理	文書	結果
10	文字列	文字列で指定した文書ファイルを開きます。指定した文書ファイルが存在しない場合には、その文書ファイルを作成します。
10	""(空の文字列)	「ファイルを開く」ダイアログボックスを表示してファイルを開きます。この場合すべてのタイプの文書ファイルが表示されます。
11	なし	開いていたファイルを閉じます。
12	""(空の文字列)	「ファイル作成」ダイアログボックスを表示して新しいファイルを作成します。
13	""(空の文字列)	「ファイルを開く」ダイアログボックスを表示してファイルを開きます。TEXTタイプの文書ファイルだけが表示されます。

この表に示した処理は、すべてシステム変数Documentに値を代入します。また、処理が正常に行われると、システム変数OKに1が代入されます。それ以外の場合には 0が代入されます。

通常の文書ファイルの処理には、文書ファイルコマンドを使用するべきです。**SEND RECORD**コマンド、**RECEIVE RECORD**コマンド、**SEND VARIABLE**コマンド、**RECEIVE VARIABLE**コマンドのいずれかを使用する場合にのみ、文書ファイルに対して**SET CHANNEL**コマンドを使用します。文書ファイルコマンドでは、これらのコマンドを操作することができません。

SEND PACKETコマンドと**RECEIVE PACKET**コマンドは、**SET CHANNEL**コマンドや第52章で説明されているシステム文書コマンドを用いれば、作成またはオープンされた文書から使用可能です。

▼ 次の例は、ImageWriterIIに接続されたプリンタポートを開き、“ こんにちは ” という単語を出力し、フォームフィードを行って、ポートを閉じます：

```
SET CHANNEL (0 ; 10+3072+16834+0)      ` プリンタポートを開く
$単語:="こんにちは"
SEND PACKET ($単語)                    `プリンタに“ こんにちは ”という単語を送信する
SEND PACKET (Char (12))                ` フォームフィードを行う
SET CHANNEL (11)                       ` ポートを閉じる
```

SET CHANNELコマンドの第2の形式の例を下記に記します。

引数に10 を指定し、ファイル拡張子が .TXT 以外の場合、プログラムでは拡張子とは無関係に既存のファイルをオープンできますが、ファイル拡張子は無視され、ファイルが存在しない場合は .TXTファイルが作られます。例えば、**SET CHANNEL (10;"Archive.BAK")** によって、Archive.BAK ファイルが存在している場合はオープンされますが、ファイルが存在しない場合は、Archive.TXT が作られます。次のコードは、文書が存在していてもいなくても .BAK を強制的に作ります。

```
 ` この例では次のような想定に基づいています
 ` MAP FILE TYPES ("BACK" ; "BAK" ; "Backup documents") がコールされている
$DocRef:=Create document ("" ; "BACK")
If(OK=1)
   ` SEND/RECEIVE RECORD/VARIABLE は Create/Open/Append document
   ` でオープンされた文書を処理しない
  CLOSE DOCUMENT ($DocRef)
  SET CHANNEL (10 ; Document)
  If (OK=1)
    ` ここでは SEND/RECEIVE RECORD/VARIABLE が使用できる
    SET CHANNEL (11)
  End if
End if
```

SEND PACKET

SEND PACKET({文書ファイル参照番号} ; パケット)

引数	タイプ	説明
文書ファイル参照番号	時間	文書ファイルの参照番号
パケット	文字列	送信するパケット

SEND PACKETコマンドは、シリアルポートや文書ファイルに＜パケット＞を送ります。＜文書ファイル参照番号＞を指定した場合は、文書ファイル参照番号の示すMacintosh文書ファイルにパケットを書き込みます。＜文書ファイル参照番号＞を指定しない場合は、**SET CHANNEL**コマンドで開かれたシリアルポートまたは文書ファイルにパケットを前もって書き込みます。**SET CHANNEL**コマンドでシリアルポートを開くか、あるいは文書ファイルコマンドで文書ファイルを開く必要があります。＜パケット＞はデータの一部であり、一般的には文字列です。

SEND PACKETコマンドを使用する前に**SET CHANNEL**コマンドでシリアルポートを開くか、または文書ファイルコマンドで文書ファイルを開く必要があります。

既存の文書ファイルに書き込む場合は、**Append document**関数で開かれていない限り、最初の**SEND PACKET**コマンドでその文書ファイルの内容がすべて消去されます。それに続く**SEND PACKET**コマンドは、文書ファイルが閉じられるまでパケットを書き加えます。

SEND PACKETコマンドは、WindowsまたはMacintoshのASCIIデータを書き込みます。WindowsまたはMacintoshのASCIIは8ビットを使用しています。標準ASCIIでは下位の7ビットしか使用しませんので、装置によっては、WindowsまたはMacintoshのように8ビットを使用していないものもあります。このような装置として、PC-DOSやMS-DOSを使用したパーソナルコンピュータや、ImageWriterプリンタがあります。このような装置に、8ビットデータを送信する場合には、ASCIIコードを変換するために、ASCIIテーブルを定義し、USE ASCII MAPコマンドを実行してからSEND PACKETコマンドでデータ送信を行ってください。ただし、日本語DOSまたは日本語仕様のMS-DOSを使用したパーソナルコンピュータや、日本仕様のImageWriterプリンタに対しては、この処理は必要ありません。XON/XOFFのようなプロトコルでは、マシン間の接続を確立するのに下位のASCIIコードを使うことがあります。ASCIIコードによって干渉したり、通信を中断する場合でも、そのようなプロトコル通信を送らないように注意してください。

▼ 次の例は、フィールドのデータを文書ファイルに書き込みます。この例では、固定長データとして書き込みます。フィールドのデータがフィールド長(15バイト)よりも短い場合には、その分のスペースを埋め込みます(つまり、指定された長さになるまでスペースを付け加えます)。固定長データの使用は、合理的な方法とはいえませんが、一部のコンピュータシステムやアプリケーションでは、広く使用されています：

```
文書:=Create document ("")              ` 文書ファイルを作成
If (OK=1)                               ` ユーザが新しい文書ファイルを開いた？
  FIRST RECORD ([従業員])
  For ($i ; 1 ; Records in selection ([従業員]))` レコード数だけ繰り返す
    ` スペース15個と名字フィールドからパケットを 作成し文書ファイルに書き込む
    SEND PACKET (文書 ; Change string (15*Char (32);[従業員]名字 ; 1))
    ` スペース15個と名前フィールドからパケットを作成し、文書ファイルに書き込む
    SEND PACKET (文書 ; Change string (15*Char (32);[従業員]名前 ; 1))
    NEXT RECORD ([従業員])
  End for
    ` control-zを文書ファイルに書き込むこのコードはEOF(End-of-file)の意味を持つ
  SEND PACKET (文書 ; Char (26))
  CLOSE DOCUMENT (文書)                 ` 文書ファイルを閉じる
End if
```

RECEIVE PACKET

RECEIVE PACKET ({文書ファイル参照番号} ; 受信変数 ; 文字数)

引数	タイプ	説明
文書ファイル参照番号	時間	文書ファイルの参照番号
受信変数	変数	受信データを格納する変数
文字数	数値	受信する文字数(バイト)

RECEIVE PACKET ({文書ファイル参照番号} ; 受信変数 ; 終了文字)

引数	タイプ	説明
文書ファイル参照番号	時間	番号文書ファイルの参照番号
受信変数	変数	受信データを格納する変数
終了文字	文字列	受信を終了する文字または記号

RECEIVE PACKETコマンドには、2つの形式があります。

第1の形式は、受信する文字数(＜文字数＞)を指定します。**RECEIVE PACKET**コマンドは、＜受信変数＞に指定した文字数分のデータを格納します。

第2の形式は、指定した文字(＜終了文字＞)が現われるまでデータを読み込みます。**RECEIVE PACKET**コマンドは、最初の＜終了文字＞を発見するまで＜受信変数＞にデータを格納します。**RECEIVE PACKET**コマンドは、＜終了文字＞を＜受信変数＞に格納しません。＜受信変数＞は、文字またはテキストタイプです。

文書を読み込む時に＜終了文字＞が現れなければ、**RECEIVE PACKET**コマンドは文書の最後で読み込みを終了します。文字列変数は指定文字数(最大32000まで)になるまで、テキスト変数は受け付けます。シリアルポートから読み込む場合は、＜終了文字＞が現われるまで待ち続けます。

どちらの形式も、シリアルポートまたは文書ファイルからデータを読み込みます。**RECEIVE PACKET**コマンドを実行する前に、**SET CHANNEL**コマンドでシリアルポートを開くか、あるいは文書ファイルコマンドで文書ファイルを開く必要があります。＜文書ファイル参照番号＞を指定した場合には、文書ファイル参照番号の示す文書ファイルからデータを読み込みます。＜文書ファイル参照番号＞を指定しない場合は、**SET CHANNEL**コマンドで開かれたシリアルポートか文書ファイルからデータを前もって読み込みます。

文書ファイルを読み込む場合、**RECEIVE PACKET**コマンドは、文書ファイルの先頭から読み込みを開始します。その後のデータ読み込みは、最後に読み込まれた文字の次から開始します。

RECEIVE PACKETコマンドへの呼び出しの後に、データをエラーもなく読み込むと、システム変数OKに1が代入されます。

ファイルの最後を越えて読み込もうとした場合は、**RECEIVE PACKET**コマンドは、そのポイントまでに読み込んだデータを返します。次の**RECEIVE PACKET**コマンドは空の文字列を返すか、あるいは変数OKに 0を代入します。

RECEIVE PACKETコマンドでシリアルポートからデータを読み込む場合は、**ON ERR CALL**メソッドをインストールしていない限り“ Alt（Macintosh版では、option）”キーを押しながらスペースバーを押すことによって、**RECEIVE PACKET**コマンドを中断することができます。また、**SET TIMEOUT**コマンドで**RECEIVE PACKET**コマンドを中断することもできます。どちらの場合でも、**RECEIVE PACKET**コマンドが中断された時点で、システム変数OKに 0が代入されます。

▼ 次の例は、20バイトのデータをシリアルポートから読み込み、変数“ 読込20 ”に格納します。

```
RECEIVE PACKET (読込20 ; 20)
```

▼ 次の例は、変数“ 文書 ”に格納された文書ファイル参照番号の示す文書ファイルからデータを読み込み、変数“ vデータ ”に格納します。ここでは改行(**Char**(13))を発見するまで読み込みます。

```
RECEIVE PACKET (文書 ; vデータ ; Char (13))
```

▼ 次の例は、文書ファイルから読み込んだデータをフィールドに格納します。データは、固定長形式で格納されています。このメソッドは、サブルーチンを呼び出してデータの後ろに付随する不要なスペースを取り除きます：

```
文書:=Open document ("" ; "TEXT")      ` “ テキスト ”ファイルを開く
If (文書 # ?00:00:00?)                  ` ユーザが文書ファイルを開いたら？
 Repeat                                 ` データがなくなるまで繰り返す
  RECEIVE PACKET (文書 ; $Var1 ; 15)       `名字のデータを15バイト読み込む
  RECEIVE PACKET (文書 ; $Var2 ; 15)       ` 名前のデータを15バイト読み込む
  If (OK=1)                             ` まだ文書ファイルを最後まで読み込んでない場合
   CREATE RECORD ([従業員])             ` 新しいレコードを作成
   [従業員]名字:=Strip ($Var1)          ` 名字をフィールドに格納
   [従業員]名前:=Strip ($Var2)          ` 名前をフィールドに格納
   SAVE RECORD ([従業員])               ` レコードを保存
  End if
 Until (OK=0)
 CLOSE DOCUMENT (文書)                  ` 文書ファイルを閉じる
End if
```

データに付随する不要なスペースを取り除くためのサブルーチン“ Strip ”を次に示します：

```
For ($i ; Length ($1) ; 1 ; -1)                  ` 文字列の最後からループを開始
  If ($1[[$i]]#" ")                              ` スペース以外の場合...
    $i:=-$i                                      ` ループを強制的に終了
  End if
End for
$0:=Delete string ($1 ; -$i ; Length ($1))       ` スペースを削除
```

SET TIMEOUT

SET TIMEOUT (秒)

引数	タイプ	説明
秒	数値	タイムアウトになるまでの秒数

SET TIMEOUTコマンドは、シリアルポートコマンドの許容する待ち時間を設定します。シリアルポートコマンドが指定した時間<秒>以内に終了しないと、そのシリアルポートコマンドは、取り消されて、システム変数OKに 0が代入されます。待ち時間は、コマンドが実行されるまでの時間です。データの送受信に要する時間ではない点に注意してください。

以前の設定値を取り消し、シリアルポートに対するタイムアウトの管理を中止する場合は、<秒>に 0を指定します。

次の表は、タイムアウトを適用することができるコマンドを示します：

コマンド	コマンド
RECEIVE PACKET	SEND PACKET
RECEIVE RECORD	SEND RECORD
RECEIVE VARIABLE	SEND VARIABLE

▼ 次の例は、シリアルポートからデータを受信します。タイムアウトを設定し、RECEIVE PACKETでデータを受け取ります。データを設定した時間内に受け取れない場合は、エラーが発生します：

```
` 設定：モデムポート；9600ボー；データビット8；ストップビット1
SET CHANNEL (1 ; 10+3072+16384+0)
SET TIMEOUT (10)                                 ` タイムアウトを10秒に設定
RECEIVE PACKET (v ; Char (13))                   ` 改行まで受信
If (OK=0)                                        ` エラー発生...
  ALERT ("受信データエラー")                     ` ユーザに通知
Else                                             ` その他の場合 ...
  [従業員]名前:=v                                ` データ格納
End if
```

RECEIVE BUFFER

RECEIVE BUFFER (受信変数)

引数	タイプ	説明
受信変数	変数	受信データを格納する変数

RECEIVE BUFFERコマンドは、**SET CHANNEL**コマンドで前もって開いたシリアルポートからデータを読み込みます。シリアルポートは、コマンドで読み込まれるまでバッファの内容を保持しています。**RECEIVE BUFFER**コマンドは、バッファ中の文字を＜受信変数＞に格納して、バッファを消去します。バッファ中に文字が存在しなければ、＜受信変数＞は何も含みません。

シリアルポートのバッファサイズは64バイトです。これは、バッファがオーバーフローするまで64バイトのデータを維持できることを意味します。オーバーフローした文字は、失われます。従って、バッファに文字が入力された時点で迅速に読み込む必要があります。

RECEIVE BUFFERコマンドは、**ON SERIAL PORT CALL**コマンドでインストールした割り込みメソッドでよく使用されます。そのメソッドが呼び出された時点でシリアルポートバッファを読み込みます。

RECEIVE BUFFERコマンドは、バッファ中のデータを即座に返す点が**RECEIVE PACKET**コマンドと異なります。**RECEIVE PACKET**コマンドはバッファ中に指定した文字を発見するまで、または指定した数の文字が入力されるまで待ちます。

▼次の例は、シリアルポートメソッドをインストールします：

```
<>Buffer:=""
<>Len=0
` モデムポート、9600ボー、8データビット、1ストップビット
SET CHANNEL (1 ; 10+3072+16384)
ON SERIAL PORT CALL ("割り込み")
```

シリアルポートからテキストを読み込むには、シリアルポートを呼び出すメソッドに**RECEIVE BUFFER**コマンドを使用します。次の例は、シリアルポートメソッド“割り込み”の内容です：

```
RECEIVE BUFFER ($バッファ)                ` シリアルポートバッファを獲得
If((Length ($バッファ) + <>Len) > 32000)  ` テキスト変数は32000バイトまで受取り可能
   <>Buffer:=""
End if
<>Buffer:=<>Buffer + $バッファ            ` バッファに追加
<>Len:=Length (<>Buffer)
CALL PROCESS (-1)                         ` 別のプロセスを更新
```

SEND RECORD

SEND RECORD ({テーブル})

引数	タイプ	説明
テーブル	テーブル	カレントレコードを送信するテーブル

SEND RECORDコマンドは、＜テーブル＞のカレントレコードを**SET CHANNEL**コマンドで開いたシリアルポートまたは文書ファイルに送信します。レコードは**RECEIVE RECORD**コマンドでなければ読み込むことのできない特別な内部フォーマットで送信します。カレントレコードが存在しなければ、**SEND RECORD**コマンドは何も行いません。

SEND RECORDコマンドは、サブレコードやピクチャも含めてレコードのすべての内容を送信します。

警告：レコードを送受信する場合、送信したテーブルと受信するテーブルの構造がすべて同じでなければなりません。つまり、どちらのファイルも、同じ数と同じ順番のフィールドを持ち、すべてのフィールドタイプが同じでなければなりません。

▼ 次の例は、テーブルのすべてのレコードを送信します：

```
SET CHANNEL (10 ; "保管")                          ` テーブルを開く
ALL RECORDS ([私のテーブル])                       ` すべてのレコードを選択
For ($i ; 1 ; Records in file ([私のテーブル]))    ` レコードの数だけ繰り返す
  SEND RECORD ([私のテーブル])                     ` レコード送出
  NEXT RECORD ([私のテーブル])                     ` 次のレコードへ移動
End for
SET CHANNEL (11)                                   ` テーブルを閉じる
```

RECEIVE RECORD

RECEIVE RECORD ({テーブル})

引数	タイプ	説明
テーブル	テーブル	レコードを受信するテーブル

RECEIVE RECORDコマンドは、**SET CHANNEL**コマンドで開いたシリアルポートまたは文書ファイルからレコードを＜テーブル＞に受信します。受信するレコードは、**SEND RECORD**コマンドで送信したものでなければなりません。**RECEIVE RECORD**コマンドを実行すると、テーブルに新しいレコードが自動的に作成されます。レコードを正しく受信した時点で、**SAVE RECORD**コマンドを使用して新しいレコードをディスクに保存します。

RECEIVE RECORDコマンドは、サブレコードやピクチャも含めてレコードのすべての内容を受信します。

ON ERR CALLメソッドをインストールしていなければ、ユーザは、**RECEIVE RECORD**コマンドの実行中に“ Alt（Macintosh版では、option）”キーを押しながらスペースバーを押すことによって、実行中の**RECEIVE RECORD**コマンドを中断することができます。システム変数OKの値を調べることにより、ユーザの割り込みを判定することができます。レコードを正常に受信すると、システム変数OKに1が代入されます。それ以外の場合は、システム変数OKに 0が代入されます。

警告：レコードを送受信する場合、送信したテーブルと受信するテーブルの構造がすべて同じでなければなりません。つまり、どちらのファイルも、同じ数と同じ順番のフィールドを持ち、すべてのフィールドタイプが同じでなければなりません。

▼ 次の例は、テーブルのすべてのレコードを送信します：

```
SET CHANNEL (10 ; "保管")                ` テーブルを開く
RECEIVE RECORD ([私のテーブル])          ` レコードを読み込む
While (OK=1)                             ` レコードがなくなるまで繰り返す
  SAVE RECORD ([私のテーブル])           ` 新しいレコードを保存
  RECEIVE RECORD ([私のテーブル])        ` レコードを読み込む
End while
SET CHANNEL (11)                         ` テーブルを閉じる
```

SEND VARIABLE

SEND VARIABLE (変数)

引数	タイプ	説明
変数	変数	送信する変数

SEND VARIABLEコマンドは、**SET CHANNEL**コマンドで開いたシリアルポートまたは文書ファイルに＜変数＞を送信します。＜変数＞は**RECEIVE VARIABLE**コマンドでなければ読み込むことのできない特別な内部フォーマットで送信します。**SEND VARIABLE**コマンドは、変数のすべての情報(タイプと値)を送信します。

▼次の例は、シリアルポートに変数を送信します。

```
SET CHANNEL (1 ; 10+3072+16384+0)
SEND VARIABLE (私の変数)
```

RECEIVE VARIABLE

RECEIVE VARIABLE (変数)

引数	タイプ	説明
変数	変数	受信する変数

RECEIVE VARIABLEコマンドは、**SET CHANNEL**コマンドで開いたシリアルポートまたは文書ファイルから**SEND VARIABLE**コマンドで送信した＜変数＞を受信します。受信した変数のタイプは、送信時のタイプに設定されます。既存の変数を受信すると、変数のタイプは送信時のタイプに変換されます。

ON ERR CALLメソッドをインストールしていない限り、ユーザは、**RECEIVE VARIABLE**コマンドの実行中に“Alt（Macintosh版では、option）”キーを押しながらスペースバーを押すことによって、実行中の**RECEIVE VARIABLE**コマンドを中断することができます。システム変数OKの値を調べることにより、ユーザの割り込みを判定することができます。レコードを正常に受信した場合、システム変数OKに1が代入されます。それ以外の場合は、システム変数OKに 0が代入されます。

▼次の例は、シリアルポートから変数を受信します：

```
SET CHANNEL(1 ; 10+3072+16384+0)
SET TIMEOUT (20)                    ` エラーによる割り込みの備え
RECEIVE VARIABLE (私の変数)
```

USE ASCII MAP

USE ASCII MAP (テーブル名 ; I/O)

引数	タイプ	説明
ファイル名	文字列	使用するASCIIテーブルの文書
I/O	数値	出力：0、　入力：1

USE ASCII MAP (* ; I/O)

引数	タイプ	説明
*	文字列	使用するデフォルトのASCIIテーブル
I/O	数値	出力：0、　入力：1

USE ASCII MAPコマンドには2つの形式があります。第1の形式は、ディスクから＜テーブル名＞で指定したASCIIテーブルをロードし、それを使用します。＜I/O＞が0の場合は、ASCIIテーブルを出力用としてロードします。＜I/O＞が1の場合、ASCIIテーブルを入力用としてロードします。

ASCIIテーブルは、「ユーザ」モードの“ ASCII出力テーブル編集 ”ダイアログボックスまたは“ ASCII入力テーブル編集 ”ダイアログボックスで作成します。4th Dimensionは、ロードしたASCIIテーブルをデータベースと文書ファイルまたはシリアルポート間のデータの転送に使用します。転送処理には、TEXT(ASCII)、DIF、SYLKの各形式のファイルの“ 読み込み ”と“ 書き出し ”が含まれます。また、ASCIIテーブルは、**SEND PACKET**コマンド、**RECEIVE PACKET**コマンド、**RECEIVE BUFFER**コマンドで転送するデータにも適用されます。ただし、**SEND RECORD**コマンド、**SEND VARIABLE**コマンド、**RECEIVE RECORD**コマンド、**RECEIVE VARIABLE**コマンドには適用されません。

＜テーブル名＞に空の文字列を指定した場合は、**USE ASCII MAP**コマンドは、標準の「ファイルを開く」ダイアログボックスを表示します。ユーザは、ここでASCIIテーブルの文書ファイルを選択することができます。ASCIIテーブルを正常にロードすると、システム変数OKに1が代入されます。それ以外の場合は、システム変数OKに 0が代入されます。

USE ASCII MAPコマンドの第2の形式は、＜テーブル名＞の代わりにアスタリスク(*)を指定します。この場合、デフォルトのASCIIテーブルを復元します。＜I/O＞が0の場合は、出力用のASCIIテーブルを復元します。＜I/O＞が1の場合は、入力用のASCIIテーブルを復元します。

▼ 次の例は、ディスクからASCIIテーブルをロードし、データを書き出します。最後にデフォルトのASCIIテーブルを復元します：

```
USE ASCII MAP ("私のテーブル" ; 0)          ` ロードしたASCIIテーブルを使用
EXPORT TEXT ([従業員] ; "私のテキスト")     ` ASCIIﾃｰﾌﾞﾙを使ってデータ書き出し
USE ASCII MAP (* ; 0)                       ` デフォルトのASCIIテーブルを復元
```

4D Compilerは、アセンブラ命令でデータベースアプリケーションを解析することができます。4D Compilerの利点には、次のようなものがあります：

■**「スピード」：**データベースの実行速度を3倍から1000倍速くします。

■**「コードチェック」：**データベースアプリケーションのコードを系統的にチェックし、論理的矛盾や構文的矛盾を検出します。

■**「データベースの保護」：**コンパイルされたデータベースは、ストラクチャやメソッドを表示したり、修正することができないこと以外は、オリジナルのデータベースと機能上は同一のものです。データベースをコンパイルすることにより、安全性を高めます。

■**「単体アプリケーション」：**4D Compilerはカスタムアイコンを持った単体アプリケーションを作成することができます。

この節のコマンドは、コンパイラの使用方法に関連があります。これらのコマンドは、データベース中のデータタイプを定義します。IDLEコマンドは、コンパイルされたデータベースで特別に使用されるコマンドです。

C_BLOB	**C_LONGINT**	**C_STRING**
C_BOOLEAN	**C_PICTURE**	**C_TEXT**
C_DATE	**C_POINTER**	**C_TIME**
C_GRAPH	**C_REAL**	**IDLE**
C_INTEGER		

注：変数タイプを明示的に宣言する必要がある場合、非コンパイルのデータベースの中でコンパイラ命令を使用することもできます。

IDLEコマンド以外のこれらのコマンドは、変数を宣言し、それらを指定したデータタイプとしてキャストします。変数を宣言することによって、変数のデータタイプに関連する曖昧さが解決されます。変数がこれらのコマンドのいずれかで宣言されていない場合には、コンパイラが変数のデータタイプを判断しようとします。フォームで使用される変数のデータタイプは、多くの場合、コンパイラで判断するのは困難です。このため、開発者がこれらのコマンドによってフォームで使用される変数を宣言することが特に重要です。

倍長整数変数および整数変数に対する数値演算は、デフォルトの数値タイプ(実数)に対する演算よりも通常は高速になります。

コンパイラの一般規則

■ 変数の間接参照はできません。パーセント記号(%)を使って間接的に変数を参照する文字型間接参照も、中括弧({...})を用いる数値型間接参照も行うことはできません。中括弧({...})は定義された配列の要素を指定する場合にのみ使用されます。

■ グローバルな変数や配列のデータのタイプは、変更することはできません。

■ 1次元配列を2次元配列に、また2次元配列を1次元配列に変更することはできません。

■ 文字(列)変数や、文字(列)の配列の要素の長さは変えられません。

■ データタイプが明確でない場合は、コンパイラ命令を使用し、変数のデータタイプを指定するべきです。

■ 最大限の能力を得るために、可能な限り倍長整数型の変数を使用してください。

■ 変数をクリアする(ヌルに初期化する)には、変数の名前を用いて**CLEAR VARIABLE**コマンドを使用します。**CLEAR VARIABLE**コマンド内で変数の名前を表わす文字列を使用してはいけません。

■ **Undefined**関数は、常にFalseを返します。変数は常に定義されています。

例：

(1) 以下は、4D Compiler用の基本変数宣言の例です：

```
C_BLOB(vxMyBlob) プロセス変数vxMyBlobはBLOBタイプの変数として宣言される
C_BOOLEAN(<>OnWindows) ｀インタープロセス変数<>OnWindowsはBooleanタイプ
                       ｀の変数として宣言される
C_DATE($vdCurDate) ｀ローカル変数$vdCurDateはDateタイプの変数として宣言される
C_GRAPH(vg1 ; vg2 ; vg3) ｀3つのプロセス変数vg1、vg2、vg3はGraphタイプの変数
                         ｀ として宣言される
```

(2) 次の例では、プロジェクトメソッド「OneMethodAmongOthers」は3つの引数を宣言します：

```
｀「OneMethodAmongOthers」プロジェクトメソッド
｀ OneMethodAmongOthers ( 実数 ; 整数 { ; 倍長整数 } )
｀ OneMethodAmongOthers ( 合計 ; 割合 { ; 比率 } )
C_REAL($1) ｀ 実数タイプの第1引数
C_INTEGER($2) ｀ 整数タイプの第2引数
C_LONGINT($3) ｀ 倍長整数タイプの第3引数
```

(3) 次の例では、プロジェクトメソッド「Capitalize」は文字列引数を受け付け、文字列の結果を返します：

```
` 「Capitalize」プロジェクトメソッド
` Capitalize ( 文字列 ) → 文字列
` Capitalize ( 元文字列 ) → 大文字に変換された文字列
C_STRING(255 ; $0 ; $1)
$0:=Uppercase(Substring($1 ; 1 ; 1))+Substring (Substring($1 ; 2))
```

(4) 次の例では、プロジェクトメソッド「SEND PACKETS」はテキスト引数の変数番号が後ろにある時間引数を受け付けます：

```
` 「SEND PACKETS 」プロジェクトメソッド
` SEND PACKETS ( 時間 ; テキスト { ; テキスト2... ; テキストN } )
` SEND PACKETS ( 文書参照番号 ; データ { ; データ2... ; データN } )
C_TIME ($1)
C_TEXT (${2})
C_LONGINT ($vlPacket)
For ($vlPacket ; 2 ; Count parameters)
   SEND PACKET ($1 ; ${$vlPacket})
End for
```

(5) 次の例では、プロジェクトメソッド「COMPILER_Param_Predeclare28」は4D Compiler用の別のプロジェクトメソッドのシンタックスを事前に宣言します：

```
` 「COMPILER_Param_Predeclare28」プロジェクトメソッド
C_REAL(OneMethodAmongOthers ; $1) `OneMethodAmongOthers(実数;整数 {;倍長整数 } )
C_INTEGER(OneMethodAmongOthers ; $2) ` ...
C_LONGINT(OneMethodAmongOthers ; $3) ` ...
C_STRING(Capitalize ; 255 ; $0 ; $1) ` Capitalize (文字列) → 文字列
C_TIME(SEND PACKETS ; $1) `SEND PACKETS ( 時間;テキスト{;テキスト2 ;テキストN})
C_TEXT(SEND PACKETS ; ${2}) ` ...
```

C_BLOB

C_BLOB ({メソッド ;}変数 **{; 変数2 ; …; 変数N}**)

引数	タイプ		説明
メソッド	メソッド	→	メソッドの名前（オプション）
変数	${...}の変数	→	定義する変数の名前

C_BLOBコマンドは、BLOB変数として指定されたそれぞれの変数をキャストします。

コマンドの第1の形式は、オプション引数＜メソッド＞が渡されない形式であり、プロセス変数、インタープロセス変数、ローカル変数の宣言とタイプに使用されます。

注：この形式は、インタープリタを使用したデータベースで使用できます。

コマンドの第2の形式は、オプション引数＜メソッド＞が渡される形式であり、メソッドの結果またはパラメータ($0、$1、$2など)またはその両方を4D Compiler用に事前に定義するために使用されます。このコマンドのシンタックスは、データベースのコンパイル中に、コンパイル時間を節約し、変数句の入力をスキップするために使用してください。

警告：2番目の形式はインタープリタモードでは実行できません。このため、このシンタックスは、インタープリタモードでは実行されないメソッドでだけ使用するようにしてください。このメソッドの名前は「COMPILER」で開始する必要があります。

上級ヒント：シンタックス「C_BLOB(${...})」では、パラメータ群がメソッドの最後のパラメータ群である場合には、同一タイプのパラメータの変数番号を宣言できます。例えば、「C_BLOB(${5})」宣言は、4Dと4D Compilerに対して、5番目のパラメータで始まり、そのメソッドがそのタイプのパラメータの変数番号を受け付けられることを示しています。詳細については、Count parameters関数を参照してください。

▼ 前節での例を参照してください。

レポートブレーク処理

Subtotal関数は、コンパイルされたメソッドの中ではブレークを起こしません。ブレークを起こすためには**BREAK LEVEL**コマンドを使用します。何の小計を算出するのかを指示するには、**ACCUMULATE**コマンドを使用する必要があります。

コンパイラ命令

コンパイラコマンドは、メソッドに用いられる変数のタイプを定義します。これらのコマンドは、通常通りに実行するメソッドに記述する必要はありません。例えば、実行されることのない“ Compiler ”というメソッドに記述することもできます。

インタプリタのアプリケーションにタイプ（型）宣言を使用するには、コマンドを実行しなければなりません。

注：変数タイプを明示的に宣言する必要がある場合、非コンパイルのデータベースの中でコンパイラ命令を使用することもできます。

C_BOOLEAN
C_DATE
C_GRAPH
C_INTEGER
C_LONGINT
C_PICTURE
C_POINTER
C_REAL
C_TEXT
C_TIME
C_STRING

C_BOOLEAN (変数1 {;...; 変数N})
C_DATE (変数1 {;...; 変数N})
C_GRAPH (変数1 {;...; 変数N})
C_INTEGER (変数1 {;...; 変数N})
C_LONGINT (変数1 {;...; 変数N})
C_PICTURE (変数1 {;...; 変数N})
C_POINTER (変数1 {;...; 変数N})
C_REAL (変数1 {;...; 変数N})
C_TEXT (変数1 {;...; 変数N})
C_TIME (変数1 {;...; 変数N})

引数	タイプ	説明
変数	変数	定義する変数の名前

C_STRING (サイズ ; 変数1 {;...; 変数N})

引数	タイプ	説明
サイズ	数値	文字(列)の長さ(バイト)
変数	変数	定義する変数の名前

C_BOOLEAN (メソッド; 変数1 {;...; 変数N})
C_DATE (メソッド; 変数1 {;...; 変数N})
C_GRAPH (メソッド; 変数1 {;...; 変数N})
C_INTEGER (メソッド; 変数1 {;...; 変数N})
C_LONGINT (メソッド; 変数1 {;...; 変数N})
C_PICTURE (メソッド; 変数1 {;...; 変数N})
C_POINTER (メソッド; 変数1 {;...; 変数N})
C_REAL (メソッド; 変数1 {;...; 変数N})
C_TEXT (メソッド; 変数1 {;...; 変数N})
C_TIME (メソッド; 変数1 {;...; 変数N})

引数	タイプ	説明
メソッド	文字列	メソッド名
変数	変数	あらかじめ定義する変数の名前

C_STRING (メソッド; サイズ ; 変数1 {;...; 変数N})

引数	タイプ	説明
メソッド	文字列	メソッド名
サイズ	数値	文字(列)の長さ(バイト)
変数	変数	あらかじめ定義する変数の名前

これらのコマンドは、変数を定義し、指定した型のデータに変換します。定義されていれば、変数のタイプに関するあいまいさが解決されます。変数がこれらのコマンドを使用してあらかじめ定義されていないときは、コンパイラ自体が変数のタイプを決定しようとします。フォームで使われている変数は、コンパイラにとって変数のタイプを決定するのが困難です。そのため、フォームに用いる変数を定義することは特に重要です。

■ **C_BOOLEAN**コマンドは、指定した変数のデータ型をブールの変数に変換します。

■ **C_DATE**コマンドは、指定した変数のデータ型を日付に変換します。

■ **C_GRAPH**コマンドは、指定した変数のデータ型をグラフに変換します。これは、グラフがレイアウト上で作成されたとき用いられます。

■ **C_INTEGER**コマンドは、指定した変数のデータ型を整数に変換します。

■ **C_LONGINT**コマンドは、指定した変数のデータ型を倍長整数に変換します。

■ **C_PICTURE**コマンドは、指定した変数のデータ型をピクチャに変換します。ピクチャデータの操作は、データ型をピクチャに定義した変数でのみ行うことがます。

■ **C_POINTER**コマンドは、指定した変数のデータ型をポインタに変換します。

■ **C_REAL**コマンドは、指定した変数のデータ型を実数に変換します。

■ **C_TEXT**コマンドは、指定した変数のデータ型をテキストに変換します。テキスト変数は、32000バイトまで含むことができます。実行速度の面から言うと、テキスト変数は、文字変数よりも低速です。

■ **C_TIME**コマンドは、指定した変数のデータ型を時間に変換します。

■ **C_STRING**コマンドは、指定した変数のデータ型を文字(列)に変換します。引数＜サイズ＞は、定義した変数が含むことのできる文字(列)の最大の長さを指定します。文字は最大255バイトまでです。データ型が文字(列)変数は、テキスト変数よりも高速に実行されます。

注：Type関数を使用することにより、変数のタイプを調べることができます。

通常の場合、倍長整数や整数での数値操作は、デフォルトの数値タイプ(実数)よりもかなり速く実行されます。

第1の形式は、任意の変数のタイプに使用されます。この形式は、インタプリタのデータベースで使用されます。

第2の形式は、＜メソッド＞に渡される変数(例えば、$0、$1、$2など)のタイプに使用されます。4th Dimensionはステートメント内のメソッドを実行しようとするので、この形式はインタプリタモードでは実行されません。それは、この2番目のシンタックスを使用した場合、インタプリタモード内で実行されないメソッドにそれを保持してしまうからです。

▼ 次の例は、変数「vSalary」と「vEmpcode」を整数型にし、「vLastname」を20バイトの文字変数、「vComment」を32000バイトのテキスト変数にします：

```
C_LONGINT (vSalary ; vEmpcode)
C_STRING (20 ; vLastname)
C_TEXT (vComment)
```

▼ 次の例は、メソッド“私のﾌﾟﾛｼｰｼﾞｬ”で引数“$1”の型を整数型にし、引数“$2”を25バイトを含んだ文字型にします：

```
C_INTEGER (私のメソッド ; $1)
C_STRING (25 ; 私のメソッド ; $2)
```

IDLE

IDLE

IDLEコマンドは、4D Compilerのために作成されたコマンドです。このコマンドは、制御を4th Dimensionエンジンに戻す命令が、コンパイルしたデータベース中のメソッドにない場合に使用します。例えば、ループ内に全くコマンドを含まない**For**ループを持ったメソッドを実行している場合に、**ON SERIAL PORT CALL**コマンドまたは**ON EVENT CALL**コマンドで割り込みメソッドをインストールしていてもループを抜け出すことはできません。このような状態では、たとえマルチファインダの下で実行されていてもユーザは他のアプリケーションに切り替えることができません。**IDLE**コマンドを、イベントによる割り込みを起こしたい箇所に挿入します。

コンパイルを前提とした無限ループを持つデータベースには、次の例のように必ず**IDLE**コマンドを使用します：

```
ON EVENT CALL ("EventProc")
<>Done:=False
MESSAGE ("実行中..."+Char (13)+"どれかキーを押すと実行を停止します。")
Repeat
   ` 4Dコマンドのないループ
  IDLE
Until (<>Done)
```

データ入力コマンド　20

この章では、「ルーチン」エディタの「Data Entry」テーマ内にあるデータ入力コマンドについて説明しますカスタムデータベースを作成するときには、この章のコマンドを使用します。この章のコマンドを使用してデータの入力、画面表示、レポート印刷を行います。これらのコマンドは、フォームに関連します。

ADD RECORD	**Modified**	**Old**
ADD SUBRECORD	**MODIFY RECORD**	
DIALOG	**MODIFY SUBRECORD**	

データ入力は、データベースの基本的な機能の1つです。ユーザが、データベースにデータを入力するための重要な処理手順です。**ADD RECORD**コマンドと**MODIFY RECORD**コマンドは、データ入力で最もよく使用されるコマンドです。これらのコマンドは、「ユーザ」モードの「新規レコード」や「レコード修正」メニューコマンドと同じ動作をします。

入力フォームは、データを入力するために使用します。入力フォームは、複数のページを持ち、それぞれのページに異なるデータまたは異なる方法で示された同じデータを持つことができます。入力フォームに関する詳細は、『4th Dimension / 4D First デザインリファレンス』を参照してください。

ADD SUBRECORDコマンドと**MODIFY SUBRECORD**コマンドはサブレコードを追加したり、修正します。それらはサブレコードに関して、**ADD RECORD**コマンドや**MODIFY RECORD**コマンドと同じ動作をします。

DIALOGコマンドは、ユーザにデータのやり取りを行わせる際によく使用されます。また、レコードに直接関係のないデータを入力する際にも使用されます。

Modified関数は、データ入力時に修正されたフィールドの場合に“ True(真) ”を返します。

Old関数は、カレントレコードの＜フィールド＞の修正する前の内容を返します。

データ入力時にカレントレコードを変更する

各テーブルは、カレントセレクションとカレントレコードを持っています。この節では、カレントレコードについて説明することにします。**ADD RECORD**コマンド、 **MODIFY RECORD**コマンド、 **MODIFY SELECTION**コマンドのいずれかのコマンドを使用してレコードを表示する場合に、表示されたレコードがカレントレコードになります。**NEXT RECORD**コマンドなどを使用してカレントレコードを変更した場合や、表示レコードへの変更をディスクに保存する必要がある場合は、**SAVE RECORD**コマンドを使用しなければなりません。ただし、「次レコード」ボタンをクリックしたり、その他の自動動作ボタンをクリックした場合には、レコードを自動的に保存するために**SAVE RECORD**コマンドを使用する必要はありません。レコードを自動的に保存することのできる自動動作ボタンには、「登録」、「次レコード」、「前レコード」、「先頭レコード」、「最終レコード」の各ボタンがあります。

コマンドでカレントレコードを変更しても、Beforeフェーズは発生しません。しかし、自動動作ボタンを使用して他のレコードに移動すると、Beforeフェーズが発生します。

ADD RECORD
MODIFY RECORD

ADD RECORD ({テーブル} ; {*})
MODIFY RECORD ({テーブル} ; {*})

引数	タイプ	説明
テーブル	テーブル	データ入力を行うテーブル
*		スクロールバーとサイズボックスの表示取り消し

ADD RECORDコマンドは、データベースに新しいレコードを追加します。また、＜テーブル＞に対して新しいレコードを作成し、それをカレントレコードとして設定し、カレント入力フォームを表示します。ユーザが新しいレコードを受け入れると、新しいレコードがカレントセレクションにある唯一のレコードになります。

次の図は、データ入力に使用する一般的なフォームです。

ADD RECORDコマンドによって、最前面に入力フォームが表示されます。ウインドウには、スクロールバーとサイズボックスがあります。オプションのアスタリスク(*)を指定すると、スクロールバーのないウインドウを表示します。

ADD RECORDコマンドは、ユーザがレコードを受け入れるか、あるいは取り消すまでフォームを表示します。ユーザが複数のレコードを追加する場合は、新しいレコードごとに1回ずつコマンドを実行しなければなりません。

MODIFY RECORDコマンドは、入力フォーム上でレコードを修正するために使用します。**MODIFY RECORD**コマンドは、ディスクから＜テーブル＞のカレントレコードをロードし、カレント入力フォームにレコードを表示します。カレントレコードがなければ、**MODIFY RECORD**コマンドは何も行いません。また、**MODIFY RECORD**コマンドはカレントセレクションに影響を与えません。

MODIFY RECORDコマンドを使用するには、カレントレコードが読み込み／書き込み可能で、またレコードはロックされていてはいけません。

入力フォームにカレントセレクションのレコード内を移動するためのボタンを設定すると、ユーザはレコードを修正したり他のレコードへ移動することができます。

どちらのコマンドも、ユーザが「登録」ボタンをクリックするか、あるいは“ Enter ”キーを押すとレコードを保存します。この場合に、システム変数OKに1が代入されます。

ユーザが「キャンセル」ボタンをクリックするか、キャンセルのキー割り当て（Ctrl(コマンド)キーとピリオドキーを同時に押す)を行ったり、または**CANCEL**コマンドが実行されると、レコードは保存されません。この場合に、システム変数OKに 0が代入されます。しかし、この場合でもレコードはメモリ上に残されたままとなります。カレントレコードポインタが変更される前に**SAVE RECORD**コマンドを実行すれば、レコードは保存されます。システム変数OKは、レコードが受け入れられるか、あるいはキャンセルされた後にのみ値が代入されます。

MODIFY RECORDコマンドを使用しても、レコードのデータが全く変更されなければ、レコードが修正されたことにはなりません。再度レコードを保存し直すことは意味を持ちません。また、変数の変更、チェックボックスのチェック、ラジオボタンの選択も修正にはなりません。フィールドのデータが変更された場合にのみレコードを保存します。

ADD RECORDコマンドまたは**MODIFY RECORD**コマンドを使ってレコードがオープンされると、フォームメソッドが実行され、**Before関数**はTrueを返します。フォームメソッドのフォームイベントは、フォームにフォームメソッドが存在する場合にのみ発生します。また、フォーム上に存在するオブジェクトメソッドも、ユーザの操作に応じて実行されます。

▼次の例は、データベースに新しいレコードを追加するためのループです：

```
INPUT FORM ([顧客] ; "顧客入力")      ` 顧客テーブルの入力フォームをセット
Repeat                                ` ユーザにキャンセルされるまでループ
  ADD RECORD ([顧客])                 ` 顧客テーブルへ新しいレコードを追加
Until (OK=0)                          ` ユーザがキャンセルし、OK=0になるまで
```

▼ 次の例は、顧客データを検索し、その検索結果により、2つのステートメントうちの1つを実行します。顧客が全く見つからない場合は、**ADD RECORD**コマンドで新しい顧客を追加します。複数の顧客が見つかった場合には、**MODIFY RECORD**コマンドでそれぞれの顧客のレコードを修正します：

```
INPUT FORM ([顧客] ; "入力1")                    ` 入力フォームを設定
OUTPUT FORM ([顧客] ; "出力1")                   ` 出力フォームを設定
v番号:=Request ("顧客番号を入力してください。")` 顧客番号を獲得
If (OK=1)
   SEARCH ([顧客] ; [顧客]番号=Num (v番号))      ` 顧客を検索する
   If (Records in selection ([顧客])=0)          ` 顧客が見つからない場合
      ADD RECORD ([顧客])                        ` 新規の顧客を追加
   Else
      MODIFY RECORD ([顧客])                     ` レコードを修正
   End if
End if
```

参照：CREATE RECORD

ADD SUBRECORD
MODIFY SUBRECORD

ADD SUBRECORD (サブテーブル ; フォーム ; {*})
MODIFY SUBRECORD (サブテーブル ; フォーム ; {*})

引数	タイプ	説明
サブテーブル	サブテーブル名	データ入力を行うサブテーブル
フォーム	文字列	データ入力に使用するフォーム
*		スクロールバーとサイズボックスの非表示指定

ADD SUBRECORDコマンドは、＜サブテーブル＞に対してフォームを使用して新しいサブレコードを追加します。**ADD SUBRECORD**コマンドは、メモリ上に新しいサブレコードを作成し、それをカレントサブレコードとします。必ず、カレント親レコードが存在しなければなりません。カレント親レコードが存在しない場合には、**ADD SUBRECORD**コマンドは何も行いません。＜フォーム＞は、必ず＜サブテーブル＞に属するフォームでなければなりません。

MODIFY SUBRECORDコマンドは、修正するカレントサブレコードを表示すること以外は**ADD SUBRECORD**コマンドと同じです。カレントサブレコードが存在しなかったり、またはカレント親レコードが存在しない場合には、**MODIFY SUBRECORD**コマンドは何も行いません。

ユーザが「登録」ボタンをクリックしたり、数値キーパッド上の“ Enter ”キーを押したり、または**ACCEPT**コマンドが実行された場合は、サブレコードはメモリに格納されます。サブレコードを格納するとシステム変数OKに1が代入されます。

サブレコードがいったん、追加されたり、修正された場合には、親レコードはサブレコードを保存するために保存しなければなりません。

サブレコードは、「キャンセル」ボタンがクリックされたり、キャンセルのキー割り当て(Ctrl(コマンド)キーとピリオドキーを同時に押す)、または**CANCEL**コマンドが実行された場合には、サブレコードはメモリに格納されません。この場合に、システム変数OKに0が代入されます。

サブレコードは、常にカレントの親レコードに追加されます。サブテーブル内にあるサブテーブルの場合には、該当する親レコード(サブレコード)を最初に選択しておく必要があります。

＜フォーム＞は、スクロールバーとサイズボックス付きのウインドウと共に表示されますが、オプション引数のアスタリスク(*)を指定するとスクロールバーのないウインドウが表示されます。

ADD SUBRECORDコマンドまたは**MODIFY SUBRECORD**コマンドを使ってレコードがオープンされると、フォームメソッドが実行され、**Before関数**はTrue（真）を返します。フォームメソッドのフォームイベントは、フォームメソッドが＜フォーム＞に対して存在する場合にのみ行われます。＜フォーム＞に対するオブジェクトメソッドもユーザの動作に応じて実行されます。

▼ 次の例は、プロジェクトメソッドの一部です。これは、[社員]テーブルの[社員]子供サブテーブルにサブレコードを追加します。複数の子供に関するデータは、[社員]子供というサブテーブルに保存します。サブレコード([社員]子供)の内容を保存するには、親レコード([社員])を保存する必要があります：

```
ADD SUBRECORD ([社員]子供 ; "子供追加")
If (OK=1)                                  ` レコード登録されているか？
  SAVE RECORDS ([社員])                    ` [社員]レコード保存
End if
```

DIALOG

DIALOG ({テーブル} ; フォーム)

引数	タイプ	説明
テーブル	テーブル名	フォームの属するテーブル
フォーム	文字列	ダイアログとして表示するフォーム

DIALOGコマンドは、ユーザに対して＜フォーム＞を提示します。このコマンドは、変数を用いてユーザから、あるいはユーザに情報を交換するために使用します。

Open window関数で開いた“ タイプ1 ”のウインドウ(形式的ウインドウ)にフォームを表示するのが最も一般的な使用方法です。この種のウインドウは1ページしかなく、スクロールバーもありません。一般的なダイアログの例を次に示します。

ダイアログにおいて、データの入力は変数を介してのみ行うことができます。フィールドには現在値を表示することはできますが、入力することはできません。ダイアログは、**ADD RECORD**コマンドのようなことができます。この場合に、フォームが受け入れられるとレコードはそのテーブルに追加されます。また、**DIALOG**コマンドを使用して、データ入力を行うことができます。この場合は、ユーザはレコードを作成し、保存しなければなりません。**DIALOG**コマンドはレコードを操作しません。

データを表示したり、情報を得る際に**ALERT**コマンドや**CONFIRM**コマンドを、**Request**関数の代わりに**DIALOG**コマンドを使用すると、これらのコマンドよりも取り扱いが複雑になります。

DIALOGコマンドは、**ADD RECORD**コマンドや**MODIFY RECORD**コマンドと異なり、フォームを表示するのにカレントの入力フォームを使用しません。デフォルトボタンのパネルは、ボタンを省略した場合にも表示されません。その代わりに「OK」ボタンと「キャンセル」ボタンを自動的に生成します。任意のカスタムボタンを作成した場合には、「OK」ボタンと「キャンセル」ボタンのどちらも生成されません。

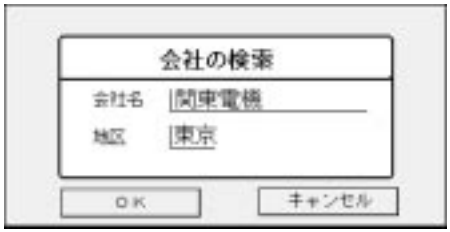
「登録」ボタンをクリックするか、あるいは登録キー(通常は、数値キーパッド上の“ Enter ”キー)を押すとシステム変数OKに1が代入されます。ユーザが「キャンセル」ボタンをクリックするか、あるいはキャンセルのキー割り当て（Ctrl(コマンド)キーとピリオドキーを同時に押す）を行うとシステム変数OKに 0が代入されます。システム変数OKは、ダイアログが閉じられるまで代入されません。

フォームメソッドが存在する場合には、BeforeフェーズとDuringフェーズが実行されます。オブジェクトメソッドが存在する場合には適宜、実行します。

▼ 次の例は、検索条件を入力するための**DIALOG**コマンドの使用方法を示しています。この例は、**QUERY BY EXAMPLE**コマンドとほぼ同じ機能を**DIALOG**コマンドで実現しています。ユーザは、表示されたカスタムフォームに検索条件を入力します。表示されるボタンは、デフォルトボタンです：

```
OPEN WINDOW (10 ; 40 ; 370 ; 220)          ` ウインドウを開く
DIALOG ([会社] ; "検索フォーム")             ` 検索ダイアログを表示する
CLOSE WINDOW                               ` ウインドウを閉じる
If (OK=1)                                  ` ダイアログが受け入れられると...
  QUERY ([会社] ; [会社]会社名=v名称 ; *)
  QUERY ([会社] ; & ; [会社]地区=v地区)
End if
```

次の図に、表示されたカスタムダイアログボックスを示します。



Modified

Modified (フィールド) → ブール

引数	タイプ	説明
フィールド	フィールド	判定に対するフィールド

Modified関数は、データ入力時に修正されたフィールドの場合に、“ True(真) ” を返します。フィールドはユーザがフィールドのデータを変更し、“ Tab ” キーの押下、別フィールドのクリック、ボタンのクリック、別エリア(スクロールエリアや外部ルーチンエリアなど)への移動などの操作で、そのフィールドを離れた場合に修正されたものと認識します。**Modified**関数は、組み込みエリアのフィールドで使用することはできません。

Modified関数をフォームメソッドで使用するよりも、オブジェクトメソッドで使用する方が簡単です。フィールドが修正された時点でオブジェクトメソッドが実行されるため、フォームメソッドで**Modified**関数を使用したのと同じ意味を持つからです。

フィールドから “ Tab ” キーで移動するだけでは、**Modified**関数は “ True(真) ” になりません。フィールドが変更されてなければ、**Modified**関数は “ True(真) ” になりません。

▼ 次の例は、フィールドの “ 数量 ” または “ 価格 ” が変更されたかどうかを判定します。どちらかが変更されると、“ 合計 ” フィールドを計算し直します。数量フィールドと価格フィールドのオブジェクトメソッドに2行目を使用しても同じことを実行することができます：

```
` ユーザがいずれかのフィールドを変更した場合
If ((Modified (数量)) | (Modified (価格)))
  合計:=数量 * 価格      ` 計算し直す。この行はオブジェクトメソッドでも構わない
End if
```

Old

Old (フィールド) → 文字列、数値、日付、ブール、時間

引数	タイプ	説明
フィールド	フィールド	古い内容を得るフィールド

Old関数は、カレントレコードの＜フィールド＞の修正する前の内容を返します。言い換えると、ディスク上に保存されているフィールドの内容を返すということです。**Old**関数は、メソッドによって修正した場合にも、ユーザによって修正された場合にも関係なく機能します。

新しく作成されたレコードの場合には、**Old**関数はそのフィールドに応じた空の値を返します。新しいレコードのフィールドが、文字フィールドの場合は空の文字列(ヌル)を返します。数値フィールドの場合はゼロ(0)を返します。日付フィールドの場合は!00.00.00!を返します。時間フィールドの場合は?00:00:00?を返します。ブールフィールドの場合に、**Old**関数は " False(偽) " を返します。

Old関数は、テキストフィールドとピクチャフィールドに関しては適用されません。その他のフィールドタイプに関しては、サブフィールドも含んで適用されますが、サブテーブル全体に対しては適用されません。

日付関数と時間関数　21

この章では、「ルーチン」エディタの「Date and Time」テーマ内にある日付と時間のコマンドおよび関数について説明します。日付と時間は、数値と同様に計算に用いることができます。西暦2000年に対応したコマンドが新たに追加されました。

次のような式を用いて、時間、分、秒を計算することができます：

```
時間:=変数時間 // 3600          ` 時間を計算
分:=(変数時間 // 60) % 60       ` 分を計算
秒:=変数時間 % 60               ` 秒を計算
```

この章では、次の関数について説明します：

Add to date	**Day number**	**Time String**
Current date	**Day of**	**SET DEFAULT CENTURY**
Current time	**Month of**	**Year of**
Date	**Time**	**Tickcount**
Milliseconds		

Current date

Current date ({*}) → 日付

引数	タイプ	説明
*		サーバから現在の日付を返す

Current date関数は、使用しているオペレーションシステム（OS）のシステムクロックの日付を現在の日付として返します。

▼ 次の例は、現在の日付をアラートボックスに表示します：

```
ALERT ("今日は、"+String (Current date)+"です。")
```

Date

Date (日付文字列) → 日付

引数	タイプ	説明
日付文字列	文字列	文字列日付に変換する文字列

Date関数は＜日付文字列＞を解釈し、日付に変換して返します。

＜日付文字列＞は通常の日付のフォーマットの規則に従って、YY.MM.DD(年.月.日)の順に指定します。年は 2桁でも 4桁でも構いません。また、月は 1桁か 2桁の数値です。年に 2桁の数値を指定すると、**Date**関数は 19を前に付け加えます。セパレータには、スラッシュ(/)、スペース、ピリオド(.)、ハイフン(-)等の文字を使用します。

＜日付文字列＞に誤りがある場合の結果は保証しません。**Date**関数は、"Jan 1,97"のような文字列を解釈できません。

▼ 次の例は、「リクエスト」ダイアログボックスを使用して、ユーザから日付の入力を得ます。入力された文字列を日付に変換し、変数 " ReqDate " に格納します：

```
ReqDate:=Date (Request ("日付を入力してください" ; String (Current date)))
```

▼ 次の例は、文字列 " 97.06.10 " を日付に変換します：

```
vDate:=Date ("97.06.10")
```

Day number

Day number (日付) → 数値

引数	タイプ	説明
日付	日付	曜日に対応する数値を得る日付

Day number関数は、＜日付＞に対応するの曜日を数値として返します。次の表は、曜日に対応する数値の一覧を示します。＜日付＞が空の場合に、**Day number**関数は 2 を返します。

曜日	数値
日曜日	1
月曜日	2
火曜日	3
水曜日	4
木曜日	5
金曜日	6
土曜日	7

▼次の例は、現在の日付を文字列にして返す関数です：

```
$曜日:=Day number (Current date)  ` $曜日に現在の曜日に対応する数値を代入
Case of
  \ ($曜日=1)
    $0:="日曜日"
  \ ($曜日=2)
    $0:="月曜日"
  \ ($曜日=3)
    $0:="火曜日"
  \ ($曜日=4)
    $0:="水曜日"
  \ ($曜日=5)
    $0:="木曜日"
  \ ($曜日=6)
    $0:="金曜日"
  \ ($曜日=7)
    $0:="土曜日"
End case
```

Day of

Day of (日付) → 数値

引数	タイプ	説明
日付	日付	日を取り出す日付

Day of関数は、＜日付＞から日を取り出して返します。

▼ 次の例は、**Day of**関数の使用方法を示しています。結果は変数 “ v結果 ” に代入されます。変数 “ v結果 ” に代入される内容についての説明がコメントされています：

```
v結果:=Day of (!97.07.21!)          ` v結果に21を代入
v結果:=Day of (Current date)        ` v結果に現在の日付の日を代入
```

Month of

Month of (日付) → 数値

引数	タイプ	説明
日付	日付	月を取り出す日付

Month of関数は、＜日付＞から月を取り出して返します。

▼ 次の例は、**Month of**関数の使用方法を示しています。結果は変数 “ v結果 ” に代入されます。変数 “ v結果 ” に代入される内容についての説明がコメントされています：

```
v結果:=Month of (!97.07.21!)        ` v結果に7を代入
v結果:=Month of (Current date)      ` v結果に現在の日付の月を代入
```

Year of

Year of (日付) → 数値

引数	タイプ	説明
日付	日付	年を取り出す日付

Year of関数は、＜日付＞から年を取り出して返します。

▼ 次の例は、**Year of**関数の使用方法を示しています。結果は変数“ v結果 ”に代入されます。変数“ v結果 ”に代入される内容についての説明がコメントされています：

```
v結果:=Year of (!97.07.21!)          ` v結果に1997を代入
v結果:=Year of (Current date)        ` v結果に現在の日付の年を代入
```

Add to date

Add to date (日付 ; 年 ; 月 ; 日) → 日付

引数	タイプ		説明
日付	日付	←	日、月、年を追加する日付
年	数値	←	日付に追加する年
月	数値	←	日付に追加する月
日	数値	←	日付に追加する日
関数の返す値	日付	←	返される日付

Add to date関数は、引数＜日付＞に＜年＞、＜月＞、＜日＞を追加して、その結果の日付を返します。

通常、任意の日付に日を追加する場合、日付演算子を使用しますが、**Add to date**関数は(“ + ”日付加算演算子を使用する場合のように）1ケ月の日数やうるう年の取り扱い方法を気にすることなく月や年をすぐに追加することができます。

▼ 次の例は、翌年の同じ日を計算します：

```
$vdInOneYear:=Add to date(Current date ; 1 ; 0 ; 0)
```

▼ 次の例は、翌月の同じ日を計算します：

```
$vdNextMonth:=Add to date(Current date ; 0 ; 1 ; 0)
```

▼ 次の例は、「$vdTomorrow:=Current date+1」と同じ結果を示します：

```
$vdTomorrow:=Add to date(Current date ; 0 ; 0 ; 1)
```

SET DEFAULT CENTURY

SET DEFAULT CENTURY (世紀 { ; ピボット年度})

引数	タイプ		説明
世紀	数値	←	デフォルトの世紀（マイナス1）
ピボット年度	数値	←	2桁年度の日付入力におけるピボット年度

SET DEFAULT CENTURYコマンドは、2桁の年度で日付を入力する場合に4th Dimensionによってデフォルトの世紀を指定することができます。

デフォルトでは、4th Dimensionは引数＜世紀＞を20世紀をセットします。例えば、

■ 97.01.25は、1997年1月25日を意味します。

■ 07.01.25は、1907年1月25日を意味します。

このデフォルトを変更するには、引数＜世紀＞にマイナス1した新規デフォルトの世紀を受け渡します。例えば、次のコードを呼び出すと、

```
SET DEFAULT CENTURY (20) `デフォルトの世紀を21世紀に切り替える
```

この場合、

■ 97.01.25は、2097年1月25日を意味します。

■ 07.01.25は、2007年1月25日を意味します。

また、オプション引数の＜ピボット年度＞を指定すると、4th Dimensionは2桁年度の日付入力を次のように解釈します：

■ 入力した2桁年度がピボット年度以上の場合は、4th Dimensionは現在のデフォルト世紀を使用します。

■ 入力した2桁年度がピボット年度より小さい場合は、4th Dimensionは現在のデフォルト世紀に1プラスした世紀を使用します。

例えば、ピボット年度が1995である次のコードを呼び出すと、

```
SET DEFAULT CENTURY (19 ; 95) `入力した年度が95より小さい場合はデフォルト世紀
                              `を21世紀に切り替える
```

この場合、

■ 97.01.25は、1997年1月25日を意味します。

■ 07.01.25は、2007年1月25日を意味します。

注：このコマンドは、2桁年度で入力された日付における4th Dimensionの解釈方法にのみ影響を与えます。

4桁年度の日付入力では、

- 1997.01.25は、1997年1月25日を意味します。
- 2097.01.25は、2097年1月25日を意味します。
- 1907.01.25は、1907年1月25日を意味します。
- 2007.01.25は、2007年1月25日を意味します。

このコマンドは、データ入力にのみ影響を与えます。データの保存および計算等では無効です。

SET DEFAULT CENTURYコマンドの効果は、すぐに現れます。

Current time

Current time ({*}) → 時間

引数	タイプ	説明
*		サーバから現在の時間を返す

Current time関数は、Macintosh のシステムクロックの現在の時間を返します。現在の時間は、常に?00:00:00? から?23:59:59? の間の値です。時間を文字列に変換する場合は、**String**関数を使用します。

▼ 次の例は、処理時間の計測方法を示しています。“ Long Operation ” は、計測の対象となるプロジェクトメソッドです：

```
所要時間:=Current time                                  ` 開始時間を保存
Long operation                                          ` 計測対象の操作
所要時間:=Current time - 所要時間                       ` 所要時間を計算
ALERT ("処理の所要時間は" + String (所要時間 ; 4))      ` 所要時間を表示
```

Time

Time (時間文字列) → 時間

引数	タイプ	説明
時間文字列	文字列	時間を表す文字列

Time関数は、時間文字列で指定した時間を返します。時間文字列は“ HH:MM:SS ”の形式で、24時間以内の値を指定します。

▼ 次の例は、アラートボックスに“ 1:00 P.M. = 13時0分 ”というメッセージを表示します：

```
ALERT ("1:00 P.M. = " + String (Time ("13:00:00") ; 4))
```

Time string

Time string (秒) → 文字列

引数	タイプ	説明
秒	数値	午前0時からの秒数

Time string関数は、秒を24時間制で表現する文字列に変換して返します。文字列は“ HH:MM:SS ”のフォーマットです。

1日の秒数は86,400秒ですが、この値を越えると、**Time string**関数は、時間、分、秒を加算します。例えば、**Time string** (86401)は、文字列“ 24:00:01 ”を返します。

▼ 次の例は、“ 46800秒は13:00:00です。”というメッセージをアラートボックスに表示します：

```
ALERT ("46800秒は" + Time string (46800) + "です。")
```

Tickcount

Tickcount → 数値

引数	タイプ		説明
このコマンドには、引数はありません。			
関数の返す値	数値	←	コンピュータが起動されてから経過したティック数(60分の1秒)

Tickcount関数は、コンピュータが起動されてから経過したティック数(60分の1秒)を返します。

注：Tickcount関数は、倍長整数型の値を戻します。

▼ **Milliseconds**関数の例を参照してください。

参照：Current time、Milliseconds

Milliseconds

Milliseconds → 数値

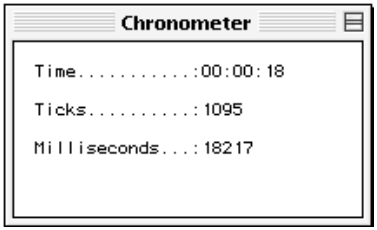
引数	タイプ		説明
このコマンドには、引数はありません。			
関数の返す値	数値	←	コンピュータが起動されてから経過したミリ秒数

Milliseconds関数は、コンピュータが起動されてから経過したミリ秒(1000分の1秒)を返します。

注：Milliseconds関数は、実数型の値を戻します。

▼ 次の例は、"Chronometer"ウィンドウを1分間表示します：

```
Open window (100 ; 100 ; 300 ; 200 ; 0 ; "Chronometer")
$vhTimeStart:=Current time
$vlTicksStart:=Tickcount
$vrMillisecondsStart:=Milliseconds
Repeat
   GOTO XY (2;1)
   MESSAGE ("Time.....:"+String (Current time -$vhTimeStart))
   GOTO XY (2;3)
   MESSAGE ("Ticks.....:"+String (Tickcount -$vlTicksStart))
   GOTO XY (2;5)
   MESSAGE ("Milliseconds...:"+String (Milliseconds -$vrMillisecondsStart))
Until ((Current time -$vhTimeStart) >=?00:01:00?)
CLOSE WINDOW
```



参照：Current time、Tickcount

ドラッグ&ドロップコマンド　22

この章では、「ルーチン」エディタの「Drag and Drop」テーマ内にあるドラッグアンドドロップコマンドについて説明します：

DRAG AND DROP PROPERTIES　　**Drop position**

4th Dimensionのバージョン6では、フォーム内でのオブジェクト間で組み込みのドラッグ&ドロップ機能を導入しました。同一のウィンドウ内でも、別のウィンドウへも、あるオブジェクトから別のオブジェクトへドラッグ&ドロップすることができます。

バージョン6では、デスクトップや別のアプリケーションとの組み込みドラッグ&ドロップ機能はありません。ただし、この機能はACIのパートナーによって開発されたプラグインで提供されます。

注：まず最初に、ドラッグ&ドロップアクションがある点から別の点までいくつかのデータを「トランスポート」するものであると想定します。次に、ドラッグ&ドロップが操作のメタファにもなれることを確認します。

ドラッグ可能およびドロップ可能なオブジェクトプロパティ

あるオブジェクトから別のオブジェクトにドラッグ&ドロップを行うには、「オブジェクトプロパティ」ウィンドウでそのオブジェクト用の「ドラッグ有効」プロパティを選択する必要があります。ドラッグ&ドロップ処理では、ドラッグされるオブジェクトがソース（送信元）オブジェクトになります。

あるオブジェクトをドラッグ&ドロップ処理の送信先にするには、「オブジェクトプロパティ」ウィンドウでそのオブジェクト用の「ドロップ有効」プロパティを選択する必要があります。ドラッグ&ドロップ処理では、データを受け取る（ドロップされた）オブジェクトが送信先オブジェクトになります。

デフォルトでは、新しく作成されたオブジェクトはドラッグもドロップもできません。これらのプロパティを設定するかどうかはユーザに任されています。

入力フォームまたはダイアログのフォームにあるすべてのオブジェクトは、ドラッグまたはドロップの対象にできます。配列の個別の要素(例えば、スクロール可能エリア)または階層リストの項目はドラッグ&ドロップができます。また逆に、配列の個別の要素や階層リストの項目に対してオブジェクトをドラッグ&ドロップすることもできます。ただし、出力フォームのディテイルエリアからオブジェクトをドラッグ&ドロップすることはできません。

4th Dimensionではアクティブオブジェクトの任意のタイプ(フィールドまたは変数)をソース（送信元）オブジェクトとしても送信先オブジェクトとしても使用できるので、ドラッグ&ドロップのユーザインタフェースは簡単に作成できます。例えば、ボタンのドラッグ&ドロップができます。

注：ドラッグとドロップの両方ができるオブジェクトは、ユーザが禁止しない限り、それ自体にもドロップできます。詳細については、以下の説明を参照してください。

次の図は、選択したオブジェクトに対して「オブジェクトプロパティ」ウィンドウで「ドラッグ有効」プロパティと「ドロップ有効」プロパティを設定した状態を示しています。

ドラッグ&ドロップのユーザインタフェース処理

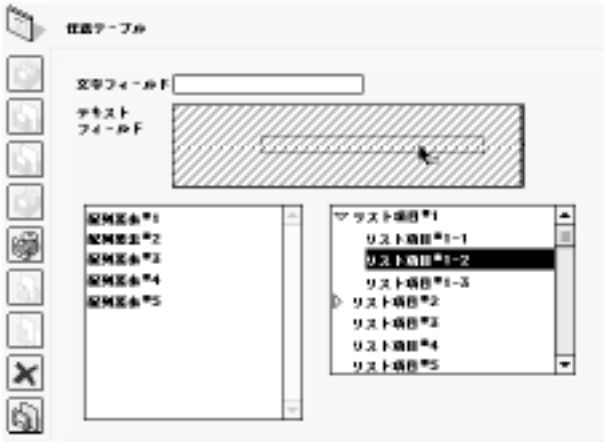
4th Dimensionはドラッグ&ドロップ機能のユーザインタフェース部分を保証しています。ドラッグ可能なオブジェクトをクリックしてから、マウスをドラッグすると、4th Dimensionがオブジェクトをドラッグします。4th Dimensionはマウスの動きに従って点線の矩形を画面に表示することでこの操作を反映します。次の図では、階層リストの項目がテキストフィールド上にドラッグされています。

テキストフィールドエリアの周りに反転した灰色のフレームのハイライト（点滅）があることに注目してください。このハイライトは、送信先オブジェクト(上の図では、テキストフィールド)を示します。この箇所でマウスボタンを放すと、4th Dimensionは、ユーザがハイライトされた送信先オブジェクトにドラッグしたオブジェクトをドロップしたいものと想定します。

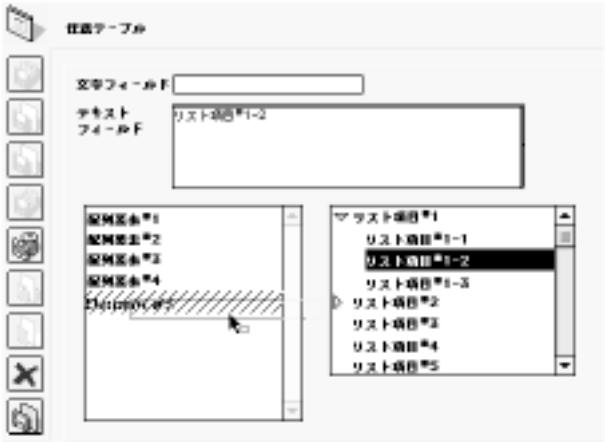
「データベースプロパティ」ダイアログボックスでは、次の図のように送信先オブジェクトのドラッグ&ドロップ時のハイライト（点滅）をフレームまたはパターン(または、その両方)に設定することができます。

デフォルトのハイライト（点滅）は、「フレーム」です。これは、オブジェクトの周りを囲む灰色の反転した矩形です。色付きの背景または色付きのオブジェクトフレームを使用する場合、このハイライト（点滅）を使用するとわかりにくくなります。この場合は、別の手段として、「パターン」ハイライトを使用します。これは、以下に示すように送信先オブジェクトを斜線パターンで塗りつぶします。

次の図は、テキストフィールドにドラッグされる階層リストの項目です。



以下では、配列要素がその配列にドラッグされています。



また、ハイライトの両方のタイプを選択することもできます。

注：送信先オブジェクトが配列(スクロール可能エリア)または階層リストのときには、送信先オブジェクトのハイライトは、要素または項目に「従い」ます。

ドラッグ&ドロップのプログラムによる処理

4th Dimensionはドラッグ&ドロップのユーザインタフェース部分を実行します。そのプログラム部分の実行はユーザに任されています。それを行うために、4th Dimensionは2つのフォームイベント、「On Drag Over」と「On Drop」を用意しています。両方のイベントとも送信先オブジェクトに送信されます。ドラッグ&ドロップの処理中、ソースオブジェクトのオブジェクトメソッドはまったく関係しません。

On Drag OverとOn Dropを受け付けるために、送信先オブジェクトは次の図で示すようにこれらの2つのイベントを「オブジェクトプロパティ」ウィンドウで起動している必要があります。

On Drag Overイベント

On Drag Overイベントは、マウスポインタがオブジェクトの上に移動したときに、繰り返し送信先オブジェクトに送信されます。このイベントの応答として、開発者は通常、以下のことを行います。

■ **DRAG AND DROP PROPERTIES**コマンドを呼び出します。これによって、ソースオブジェクトに関する情報を得られます。

■ (オブジェクトメソッドが現在実行されている)送信先オブジェクトとソースオブジェクトの両方の性質とタイプに従って、ドラッグ&ドロップの受け付けまたは拒否を行います。

ドラッグを受け付けるには、送信先オブジェクトメソッドが 0(ゼロ)を返す必要があるので、「$0:=0」と記述します。ドラッグを拒否するには、オブジェクトメソッドが -1(マイナス1)を返す必要があるので、「$0:=-1」と記述します。On Drag Overイベント中に、4th Dimensionはこのオブジェクトメソッドを関数として扱います。結果が返されない場合には、4th Dimensionはドラッグが受け付けられたのだと認識します。

ドラッグを受け付けた場合には、送信先オブジェクトがハイライトされます。ドラッグを拒否した場合には、送信先オブジェクトはハイライトされません。ドラッグを受け付けることは、ドラッグされたデータが送信先オブジェクトに挿入されるという意味にはなりません。これは、単にマウスボタンをこの場所で放したときに、ドラッグされたデータが送信先オブジェクトによって受け付けられることを意味するだけです。

ドロップ可能なオブジェクトに対して開発者が On Drag Overイベントを処理しない場合には、そのオブジェクトは、ドラッグされたデータの性質やタイプに関係なく、すべてのドラッグ処理に対してハイライトされます。

On Drag Overイベントは、ドラッグ&ドロップ処理の最初の段階を制御する手段です。ドラッグされたデータが送信先オブジェクトと互換性のあるタイプかどうかをテストでき、また、ドラッグの受け付けや拒否をできるだけでなく、4th Dimensionがユーザの判断に基づいて送信先オブジェクトをハイライトした(またはしなかった)ので、この処理が行われたことを同時にユーザに通知することができます。

On Drag Overイベントはマウスの移動に従って、現在の送信先オブジェクトに対して繰り返し送信されるので、このイベントのコード処理は短く、短期間で実行できるようにしてください。

警告：ドラッグ&ドロップがプロセス間のドラッグ&ドロップである場合には、つまり、ソースオブジェクトが送信先オブジェクトのプロセス(ウィンドウ)とは異なるプロセス(ウィンドウ)にある場合は、On Drag Over**イベントに対する送信先オブジェクトのオブジェクトメソッドはソースプロセスのコンテキスト内(ソースオブジェクトのプロセス)で実行され、送信先オブジェクトのプロセスでは実行されません。これはそのような実行が発生した場合に限ります。この種の実行の長所については、この章の最後で説明します。**

On Dropイベント

On Dropイベントはマウスポインタが送信先オブジェクトに対して放されたときにそのオブジェクトに一度送信されます。このイベントはドラッグ&ドロップ処理の第2段階であり、ユーザアクションの応答として処理を実行します。

このイベントは、On Drag Overイベント中にドラッグが受け付けられなかった場合には、オブジェクトには送信されません。オブジェクトに対してOn Drag Overイベントを処理し、ドラッグを拒否した場合には、On Dropイベントは発生しません。つまり、On Drag Overイベント中にソースオブジェクトと送信先オブジェクト間のデータタイプの互換性をテストして、可能なドロップを受け付けた場合には、On Drop中にデータの再テストをする必要はありません。データが送信先オブジェクトに対して適切であることをすでに知っているためです。

4th Dimensionのドラッグ&ドロップインプリメントの特長は、必要なことは何でもできるという点です。次の例を参照してください。

■ 階層リストの項目がテキストフィールドに対してドロップされる場合には、テキストフィールドの最初、最後、または途中にリスト項目のテキストを挿入できます。

■ フォームには2つの状態のピクチャボタンが含まれており、これはごみ箱が空であるか、いっぱいであるかを表すことができます。オブジェクトをそのボタンにドロップすることは、(ユーザインタフェースの立場からすると)「ドラッグされ、ドロップされたオブジェクトをごみ箱に削除すること」です。このとき、ドラッグ&ドロップはある場所から別の場所にデータをトランスポートしません。その代わりにアクションを実行します。

■ フローティングウィンドウからフォーム内のオブジェクトに配列要素をドラッグすることは、「このウィンドウで、データベースに格納されている顧客をリストしているフローティングウィンドウからドラッグ&ドロップしたばかりの顧客レコードの名前を表示せよ」という意味になります。

■ その他。

上記の例からもおわかりのように、4th Dimensionのドラッグ&ドロップインタフェースはユーザが考えた任意のユーザインタフェースメタファをインプリメントできるフレームワークです。

ドラッグ&ドロップのコマンド

DRAG AND DROP PROPERTIESコマンドは、以下を返します。

■ ドラッグされたオブジェクトへのポインタ(フィールドまたは変数)

■ ドラッグされたオブジェクトが配列要素またはリスト項目の場合には、配列要素番号または項目番号。

■ ソースプロセスのプロセス番号

Drop positionコマンドは、送信先オブジェクトが配列(スクロール可能エリア)または階層リストの場合には、目的の要素またはリスト項目の項目位置を示す要素番号を返します。

ドラッグ&ドロップがデータを移動するためではなく、特定の処理のためのユーザインタフェースメタファである場合には、何でも希望することを実行できます。

参照：Drop position、DRAG AND DROP PROPERTIES、Form event

Drop position

Drop position → 整数

引数	タイプ		説明
このコマンドには、引数はありません。			
関数が返す値	数値	←	送信先配列要素番号、リスト項目位置、あるいは、最後の配列要素または最後のリスト項目を超えてドロップが発生した場合は-1

Drop position関数は、オブジェクトがドラッグされ、ドロップされた配列要素番号またはリスト項目位置を返します。

一般に、配列または階層リストに対して発生したドラッグ&ドロップイベントを処理しているときに**Drop position**関数を使用します。

送信先オブジェクトが配列の場合には、この関数は配列要素の番号を返します。送信先オブジェクトが階層リストの場合には、この関数はリスト項目の位置を返します。どちらの場合も、ソースオブジェクトが最後の配列要素または最後のリスト項目を超えてドロップされた場合には、この関数は -1を返します。

ドラッグ&ドロップイベントでははく、配列や階層リストに対して発生したイベントを処理している間に **Drop position**関数をコールする場合には、この関数は -1を返します。

重要：フォームオブジェクトは、その「ドロップ有効」プロパティが選択されている場合にはドロップされたデータを受け付けます。また、そのオブジェクトメソッドは、On Drag Over**または**On Drop**あるいはその両方に対して、これらのイベントを処理するために起動される必要があります。**

▼ **DRAG AND DROP PROPERTIES**コマンドの例を参照してください。

参照：Drop position、DRAG AND DROP PROPERTIES、Form event

DRAG AND DROP PROPERTIES

DRAG AND DROP PROPERTIES (ソースオブジェクト ; ソース要素 ; ソースプロセス)

引数	タイプ		説明
ソースオブジェクト	ポインタ	←	ドラッグ&ドロップされたソースオブジェクトのポインタ
ソース要素		←	ドラッグされた配列要素番号、または、ドラッグされた階層リスト項目、あるいは、ソースオブジェクトが配列でもリストでもない場合には-1
ソースプロセス		←	ソースプロセス番号

DRAG AND DROP PROPERTIESコマンドは、オブジェクトに対するOn Drag Overイベントまたは On Dropイベントが発生したときにソースオブジェクトについての情報を取得することができます。

一般に、On Drag Overイベントまたは On Dropイベントが発生する(送信先オブジェクト)オブジェクトのオブジェクトメソッド(または、呼び出すサブルーチンの1つ)の内部から**DRAG AND DROP PROPERTIES**コマンドを使用します。

重要：フォームオブジェクトは、その「ドロップ有効」プロパティが選択されている場合にはドロップされたデータを受け付けます。また、そのオブジェクトメソッドは、On Drag Over**または**On Drop**あるいはその両方に対して、これらのイベントを処理するために起動される必要があります。**

このコマンドの呼び出しの後には、以下が発生します：

■ 引数<ソースオブジェクト>は、ソースオブジェクト(ドラッグ&ドロップされたオブジェクト)へのポインタです。このオブジェクトは、送信先オブジェクト(On Drag Overイベントまたは On Dropイベントが発生するオブジェクト)または異なるオブジェクトにできることに注意してください。同一のオブジェクト間でデータのドラッグとドロップを行うことは、配列や階層リストでは便利です。これは、ユーザが配列またはリストを手作業でソートできる簡単な方法です。

■ドラッグ&ドロップされたデータが配列要素(配列になるソースオブジェクトの要素)である場合には、引数<ソース要素>はその配列要素の番号を返します。それ以外の場合には、ドラッグ&ドロップされたデータがリスト項目(階層リストになるソースオブジェクトの項目)である場合には、引数<ソース要素>はそのリスト項目の位置を返します。それ以外の場合、ソースオブジェクトが配列でも階層リストでもない場合には、引数<ソース要素>は -1を返します。

■ ドラッグ&ドロップ処理はプロセス間でも発生します。引数＜ソースプロセス＞は、ソースオブジェクトが属する番号プロセスと同じです。このパラメータの値をテストすることは重要です。単にソースデータから送信先オブジェクトにコピーするだけで、同一プロセス内のドラッグ&ドロップに応答できます。通常、データ入力エリア(フィールドまたは変数)に対してソース変数(配列およびリスト)からドラッグ&ドロップユーザインタフェースをインプリメントします。

ドラッグ&ドロップイベントがないところで**DRAG AND DROP PROPERTIES**コマンドをコールする場合には、引数＜ソースオブジェクト＞はヌル値を返し、＜ソース要素＞は -1を、＜ソースプロセス＞は 0を返します。

Tips： 4th Dimensionはドラッグ&ドロップのグラフィカルな部分を自動的に処理します。次に適切な方法でイベントに応答する必要があります。次の例では、その応答はドラッグされたデータをコピーすることです。別の方法として、例えば、フローティングウィンドウから配列要素をドラッグ&ドロップすることが送信先ウィンドウ(送信先オブジェクトが存在するウィンドウ)に構造化されたデータ(例: ソース配列要素によって一意に特定されたレコードからのいくつかのフィールド)を入れるといった、洗練されたユーザインタフェースをインプリメントできます。

On Drag Overイベント中に送信先オブジェクトがドラッグ&ドロップ処理を受け付けるかどうかをソースオブジェクトのタイプや性質(または他の理由)に従って判断するには、**DRAG AND DROP PROPERTIES**コマンドを使用します。ドラッグ&ドロップを受け付ける場合には、オブジェクトメソッドは「$0:=0」を返す必要があります。ドラッグ&ドロップを受け付けない場合には、オブジェクトメソッドは「$0:=-1」を返す必要があります。ドラッグ&ドロップを受け付けたか、拒否したかは、画面に反映されます。つまり、オブジェクトがドラッグ&ドロップ処理の潜在的な送信先として受け付けられればハイライトされ、拒否されればハイライトされません。

Tips：On Drag Over**イベント中に、送信先オブジェクトのオブジェクトメソッドは、ソースオブジェクトのプロセスのコンテキスト内で実行されます。プロセス間ドラッグ&ドロップのソースオブジェクトがフィールドである場合には、このイベントの機会を利用してソースデータをインタープロセス変数にコピーできます。そうすることによって、後で、**On Drop**イベント中に、ドラッグされたフィールドの値を取得するためにソースプロセスとのプロセス間通信を開始する、という必要がなくなります。プロセス間ドラッグ&ドロップにソースオブジェクトとして変数が含まれる場合には、**On Drop**イベント中にGET PROCESS VARIABLEコマンドを使用できます。**

▼ データベースフォームの中には、スクロール可能エリアがあって、そのスクロール可能エリアのある部分からそのデータベース内の別の部分にドラッグ&ドロップするだけで要素の順序を手作業で変えたい場合があります。それぞれの状況に応じてコードを書くよりも、これらのスクロール可能エリアの任意のものを処理する汎用プロジェクトメソッドをインプリメントしたいと思っています。この場合には、以下のようなコードを作成できます：

```
` 配列自体のドラッグ&ドロッププロジェクトメソッドの処理
` 配列自体のドラッグ&ドロップの処理 ( ポインタ ) → ブール値
` 配列自体のドラッグ&ドロップの処理 ( -> 配列 ) → 配列自体のドラッグ&ドロップ
` だったもの
Case of
   \ (Form event=On Drag Over)
      DRAG AND DROP PROPERTIES ($vpSrcObj ; $vlSrcElem ; $vlPID)
      If ($vpSrcObj=$1)
         `配列からそれ自体に対するドラッグ&ドロップである場合にはそれを受け付ける
         $0:=0
      Else
         $0:=-1
      End if
   \ (Form event=On Drop)
      `ドラッグ&ドロップソースオブジェクトについての情報を取得する
      DRAG AND DROP PROPERTIES ($vpSrcObj ; $vlSrcElem ; $vlPID)
      `送信先の配列要素番号を取得する
      $vlDstElem:=Drop position
      `配列要素が配列要素自体にドロップされなかった場合
      If ($vlDstElem   #   $vlSrcElem)
         ` 配列の要素0にドラッグされた要素を保存する
         $1->{0}:=$1->{$vlSrcElem}
         `ドラッグされた配列要素を削除する
         DELETE ELEMENT ($1-> ; $vlSrcElem)
         `送信先要素がドラッグされた要素を超える場合
         If ($vlDstElem>$vlSrcElem)
            `送信先の配列要素番号から1引く
            $vlDstElem:=$vlDstElem-1
         End if
         `ドラッグ&ドロップが最後の要素を超えて発生した場合
         If ($vlDstElem=-1)
            ` 送信先の配列要素番号を配列の最後で新しい要素に設定する
            $vlDstElem:=Size of array($1->)+1
         End if
            `この新しい配列要素を挿入する
         INSERT ELEMENT ($1-> ; $vlDstElem)
```

```
            `配列の要素ゼロに先ほど保存した値を設定する
         $1->{$vlDstElem}:=$1->{0}
         `要素は新しく選択された配列の要素になる
         $1->:=$vlDstElem
      End if
End case
```

参照：Drop position、DRAG AND DROP PROPERTIES、Form event

この章では、「ルーチン」エディタの「Entry Control」テーマ内にあるド入力制御コマンドについて説明します。この章のコマンドは、データの入力を制御したり、フォームを閉じるために使用します。

ACCEPT	**FILTER KEYSTROKE**	**Keystroke**
CANCEL	**GOTO AREA**	**REJECT**

ACCEPT

ACCEPT

ACCEPTコマンドは、次の目的のために入力フォーム内で使用します：

■ 新規または修正されたレコードやサブレコードを受け入れるため

■ **DIALOG**コマンドで表示されたフォームを閉じるため

■ 例えば、**DISPLAY SELECTION**コマンドを使ってレコードのセレクションを表示しているフォームを終了するため

ACCEPTコマンドは、“ Enter ” キーを押すのと同じ動作を実行します。

Duringフェーズは**ACCEPT**コマンドが呼び出されたフェーズが終了した後、発生します。さらに、そのフォームがデータ入力用に使用されると、Afterフェーズが発生します。

フォームが受け入れられた後で、システム変数OKに1が代入されます。

一般に**ACCEPT**コマンドは、選択されているメニューコマンドの結果として実行されます。また、「動作なし」属性ボタンのオブジェクトメソッド内でも使用されます。

また、**Open window**関数におけるオプションの「クローズメソッド」内でも使用されます。もし、あるウインドウにコントロールメニューボックス（Macintosh版では、クローズボックス）があると、**ACCEPT**コマンドはコントロールメニューボックスがクリックされたり、または「閉じる」メニューコマンドが選択された際に実行されるメソッドの中で呼び出されます。

ACCEPTコマンドは、待ち行列を作成することはできません。例えば、2つの**ACCEPT**コマンドを実行しても、1つの**ACCEPT**コマンドを実行したのと同じ効果しか得られません。

▼ 次の例は、メニューコマンドから呼び出される1行のメソッドです。これは、カレントデータの入力のフォームまたはダイアログボックスを受け入れます：

```
ACCEPT
```

CANCEL

CANCEL

CANCELコマンドは、カレントの入力フォームまたは出力フォームを取り消します。**CANCEL**コマンドは、キャンセルのキー割り当て（Ctrl(コマンド)キーとピリオドキーを同時に押す）を行った場合や「キャンセル」ボタンをクリックした場合と同じ意味を持ちます。**MODIFY SELECTION**コマンドや**DISPLAY SELECTION**コマンドで表示された出力フォームを**CANCEL**コマンドで取り消すことができます。

フォームを取り消した後で、システム変数OKに 0がセットされます。

もし、あるウインドウにコントロールメニューボックス（Macintosh版では、クローズボックス）があると、**CANCEL**コマンドはコントロールメニューボックスがクリックされたり、または「閉じる」メニューコマンドが選択された際に実行されるメソッドの中で呼び出されます。

一般に**CANCEL**コマンドは、選択されているメニューコマンドの結果として実行されます。**CANCEL**コマンドは「動作なし」属性ボタンのオブジェクトメソッド内で使用することができます。データ入力時に使用すると、オブジェクトメソッド内の**SAVE RECORD**コマンドでレコードを保存することができます。レコードが保存された後で、**CANCEL**コマンドを使用するとAfterフェーズの実行が無効になります。

CANCELコマンドは、待ち行列を作成することができません。例えば、2つの**CANCEL**コマンドを実行しても、1つの**CANCEL**コマンド実行をした場合と同じ効果しか得られません。

▼ 次の例は、メニューコマンドから呼び出される1行のメソッドです。これは、カレントフォームまたはダイアログボックスを取り消します：

```
CANCEL
```

REJECT

REJECT (フィールド)

引数	タイプ	説明
フィールド	フィールド	取り消すフィールド

このコマンドを使用することは、ほとんどありません。このコマンドを使用する前に、4th Dimension内に組み込まれているデータチェックツールを使用してください。

REJECTコマンドには、2つの形式があります。第1の形式は、引数がありません。これは、データ入力全体を取り消し、レコードを受け入れません。第2の形式は、データ入力エリアだけを取り消します。

REJECTコマンドの第1の形式では、完全でないレコードをユーザが受け入れないようにします。「動作なし」ボタンと数値キーパッド上の " Enter " キーの組み合わせに、**ACCEPT**コマンドと**CANCEL**コマンドを使用してレコードの受け入れや取り消しを制御すれば、**REJECT**を使用しなくても同じ効果を得ることができます。**REJECT**コマンドの第1の形式よりも、次の第2の形式を使用することをお勧めします。

ユーザが入力したレコードが完全でなかったり、正しくない場合に、そのレコードを受け入れられないように第1の形式の**REJECT**コマンドを実行すると、受け入れようとしたレコードのフォームが表示されたままになります。従って、ユーザはレコードが受け入れられるか、あるいは取り消されるまでデータの入力を続行しなければなりません。

REJECTコマンドを実行するのに最適な場所は、数値キーパッド上の " Enter " キーを伴う「登録」ボタンのオブジェクトメソッドです。この場合のデータのチェックは、レコードが受け入れられた場合にのみ実行されます。ユーザが " Enter " キーを押しても、データをチェックしないで済ませることはできません。

REJECTコマンドの第2の形式は、引数＜フィールド＞で実行します。カーソルは、データ入力(フィールド)エリアに停止したままになります。この形式は、ユーザが正しい値を入力するように強制します。この場合、フィールドの修正の直後に**REJECT**コマンドを使用しなければなりません。**Modified**関数を使用して、修正されたかどうかを判定します。データ入力エリアのオブジェクトメソッドで、**REJECT**コマンドを使用することもできます。このコマンドは、組み込みエリアのフィールドに何の影響も与えません。

REJECTコマンドは、入力フォームメソッドのDuringフェーズでのみ機能します。**REJECT**コマンドのいずれかの形式をフォームメソッドまたはフォームのオブジェクトメソッドに記述しなければなりません。サブフォームに**REJECT**コマンドを使用する場合は、それをサブフォームのメソッドまたはオブジェクトメソッドに記述します。

フィールドで取り消されたデータを選択するために、**HIGHLIGHT TEXT**コマンドを使用することもできます。

▼ 次の例は、「登録」ボタンのオブジェクトメソッドで使用する第1の形式の**REJECT**コマンドです。数値パッドキー上の“ Enter ”キーは、「登録」ボタンと同義になるように設定します。つまり、ユーザがレコードを受け入れるために“ Enter ”キーを押した場合でも、このボタンのオブジェクトメソッドが実行されるようにします。このレコード例は、銀行取引のためのものです。取引が小切手で行われる場合には、小切手番号があるはずです。小切手番号がない場合は、ここのチェックで取り消します：

```
Case of
  \ ((種別="小切手") & (小切手番号=""))        ` 小切手に番号がない場合
    ALERT ("小切手番号を入力してください。" )  ` ユーザに警告する
    REJECT                                    ` 入力を取り消す
    GOTO AREA (小切手番号)                    ` “ 小切手番号 ” フィールドに戻る
End case
```

▼ 次の例は、フィールド“ 給与 ”のオブジェクトメソッドの一部です。このオブジェクトメソッドは、フィールド“ 給与 ”が200万円以下であるかを判定します。200万円以下の場合は、このフィールドを取り消します。「フォーム」エディタでフィールドの最小値を指定しても、同じ処理を実現することができます：

```
If (給与 < 2000000)
  ALERT ("給与を200万円以上にしてください。")
  REJECT (給与)
End if
```

GOTO AREA

GOTO AREA（入力エリア)

引数	タイプ	説明
入力エリア	フィールド	移動先のフィールドまたは変数

GOTO AREAコマンドは、入力フォーム内の＜入力エリア＞に挿入ポインタ(カーソル)を移動します。このコマンドは、ユーザがフィールドや変数の中をクリックしたり、“ Tab ”キーを押すのと同じことを行うことができます。このコマンドは、組み込みエリア上のフィールドでは何も行いません。

REJECTコマンドの例を参照ください。

Keystroke

Keystroke → 文字列

引数	タイプ		説明
このコマンドには、引数はありません。			
関数が返す値	数値	←	ユーザによって入力された文字

Keystroke関数は、ユーザがフィールドや入力可能エリアに入力したキャラクタを返します。

通常、On Keystrokeイベントフォームを操作中に、フォームやオブジェクトメソッド内で**Keystroke**関数を呼び出します。キーストロークイベントを見つけるには、**Form event**関数を使用します。

ユーザが実際に入力した文字を他の文字と置き換えるには、**FILTER KEYSTROKE**コマンドを使用します。

注：新しく入力する文字同様に、編集中の入力可能エリアの現在値に応じて on the fly 操作を行いたい場合、次のことを覚えていてください。つまり、画面に表れているテキストは、データソースフィールドの値、または編集中のエリア用の変数にはまだなっていません。そのエリア用のデータ入力が有効になってから、データソースフィールドや変数は入力値に割り当てられるのです (例として、他のエリアへの図表化、ボタンのクリック等)。よってデータ入力を変数に“ 投影 ”し、この投影値を使用して動作を行うかどうかはユーザ次第となります。特有の機能を実行するためのカレントテキスト値を知る必要ができた場合は、上記を行う必要があります。

以下のことを行う場合に**Keystroke** 関数を使用します。

■ カスタマイズされた方法で文字をフィルタする場合

■ データ入力フィルタを使用しない方法でデータ入力をフィルタする場合

■ 動的ルックアップ をインプリメントする場合

▼ **FILTER KEYSTROKE**コマンドの例題を参照してください。

▼ On Keystrokeイベントを処理する際、(カーソルがある) カレントテキストエリアを編集してください。このエリア用のデータソース (フィールドまたは変数)の“ 未来値 ”は使用しないでください。下記の“ キーストローク処理 ”プロジェクトメソッドを使用すれば、どのテキストエリアのデータ入力も2番目の変数に投影できます。ユーザはこの2番目の変数を使用して、そのエリアに文字を入力しながら機能を実行できます。ユーザは

ポインタを最初の引数としてエリアのデータソースに渡し、それからポインタを第2引数として投影変数に渡します。メソッドは投影変数内のテキストエリアの新しい値を返し、その値が最後に入力した文字が挿入される前の値と異なる場合はTrueを返します：

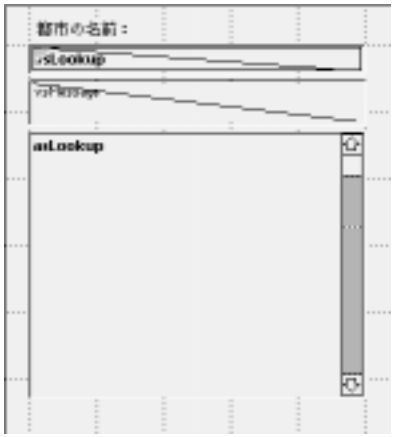
```
` 「キーストローク処理」プロジェクトメソッド
`キーストローク処理 (ポインタ ; ポインタ) → ブール
`キーストローク処理 (->ソースエリア ; -> 現在値) → 新しい値
`入力可能エリア内のテキスト選択範囲を取得する
GET HIGHLIGHT ($1-> ; $vlStart ; $vlEnd)
` 現在値で作業を開始する
$vtNewValue:=$2->
` キーを押すか、文字を入力するかで、それぞれに応じた機能が実行される
Case of
    `Backspace (Delete) キーを押う
    \ (Ascii (Keystroke)=Backspace )
        `選択した文字またはテキストカーソルの左にある文字が削除される
        $vtNewValue:=Substring ($vtNewValue ; 1 ; $vlStart-1-Num($vlStart=$vlEnd))
                            +Substring ($vtNewValue ; $vlEnd)
        `受け付け可能なキャラクタが入力される
    \ (Position (Keystroke ; "abcdefghjiklmnopqrstuvwxyz -0123456789")>0)
        If ($vlStart#$vlEnd)
            `1 文字以上の文字が選択され、キーストロークがこれらを無効にする
            ` $vtNewValue:=Substring ($vtNewValue ; 1 ; $vlStart-1)
                    +Keystroke+Substring ($vtNewValue ; $vlEnd)
        Else
            `テキスト選択がテキストカーソルになる
            Case of
                `テキストカーソルをテキストの冒頭に置く
                \ ($vlStart<=1)
                    `テキストの冒頭にキャラクタを挿入する
                    $vtNewValue:=Keystroke+$vtNewValue
                        ` カレントテキストカーソルはテキストの末尾にある
                \ ($vlStart>=Length ($vtNewValue))
                    `テキストの末尾にキャラクタを加える
                    $vtNewValue:=$vtNewValue+Keystroke
            Else
                `テキストカーソルはテキスト内の任意の場所にあり、
                `そこに新しいキャラクタを挿入する
                $vtNewValue:=Substring ($vtNewValue ; 1 ; $vlStart-1)+Keystroke
                        +Substring ($vtNewValue ; $vlStart)
            End case
        End if
```

```
        `矢印キーを押す
        `キーストロークを受け付けるだけで何も行わない
    \ (Ascii (Keystroke)=Left Arrow Key )
    \ (Ascii (Keystroke)=Right Arrow Key )
    \ (Ascii (Keystroke)=Up Arrow Key )
    \ (Ascii (Keystroke)=Down Arrow Key )
Else
    `文字、アラビア数字、スペース、ダッシュ以外の文字を受け付けない
    FILTER KEYSTROKE ("")
End case
`異なる値がある場合
$0:=($vtNewValue#$2->)
    `次のキーストローク操作用の値を返す
$2->:=$vtNewValue
```

このプロジェクトメソッドをあなたのアプリケーションに追加すると、次のようにそのメソッドを使用することができます：

```
`「MyObject」 入力可能エリアのオブジェクトメソッド
Case of
    \ (Form event=On Load)
        MyObject:=""
        MyShadowObject:=""
    \ (Form event=On Keystroke)
        If ( Handle keystroke (->MyObject ; ->MyShadowObject))
            `MyShadowObject 内に記憶されている値を使用しての最適機能の実行
        End if
End case
```

次のフォームの一部を検証してみましょう。



これは、入力可能エリア「vsLookup」、入力不可エリア「vsMessage」、スクロール可能エリア「asLookup」オブジェクトで構成されています。「vsLookup」内に文字を入力しながら、オブジェクト用のメソッドは [米国郵便番号] テーブル上のクエリを実行します。これによりユーザは都市名の最初の文字を入力するだけで 米国の都市を見つけることができます。下記にに「vsLookup 」入力可能エリアのオブジェクトメソッドを示します：

```
` 「vsLookup」入力可能エリアのオブジェクトメソッド
Case of
    \ (Form event=On Load)
        vsLookup:=""
        vsResult:=""
        vsMessage:="検索したい都市の最初の文字を入力してください。"
        CLEAR VARIABLE(asLookup)
    \ (Form event=On Keystroke )
        If ( キーストローク処理 (->vsLookup ; ->vsResult))
            If (vsResult#"")
                QUERY([米国郵便番号];[米国郵便番号]都市=vsResult+"@")
                MESSAGES OFF
                DISTINCT VALUES ([米国郵便番号]都市 ; asLookup)
                MESSAGES ON
                $vlResult:=Size of array(asLookup)
                Case of
                    \ ($vlResult=0)
                        vsMessage:="都市はありません。"
                    \ ($vlResult=1)
                        vsMessage:="1つの都市を見つけました"
                Else
```

```
                vsMessage:=String($vlResult)+" 都市見つかりました"
            End case
        Else
            DELETE ELEMENT (asLookup ; 1 ; Size of array(asLookup))
            vsMessage:="検索したい都市の最初の文字を入力してください。"
        End if
    End if
End case
```

次の図は、「ユーザ」モードでフォームを表示したものです。

参照：FILTER KEYSTROKE、Form event

FILTER KEYSTROKE

FILTER KEYSTROKE (フィルタ文字)

引数	タイプ		説明
フィルタ文字	文字列	→	フィルタしたキーストローク文字 またはキーストロークをキャンセル する空の文字列

FILTER KEYSTROKEコマンドを使用すれば、ユーザがフィルードや入力可能エリアに入力した文字を引数＜フィルタ文字＞ の最初の文字と置き換えることができます。

空の文字列を受け渡すと、キーストロークはキャンセルされ無視されます。

通常、On Keystrokeフォームイベントを操作中に、フォームやオブジェクトメソッド内で**FILTER KEYSTROKE**コマンドをコールします。キーストロークイベントを見つけるには、Form event関数を使用します。実際のキーストロークを得るには、**Keystroke**関数を使用します。

注：FILTER KEYSTROKEコマンドを使用すれば、ユーザが入力した文字をキャンセルしたり、他の文字と置き換えることができます。一方、特定のキーストローク用に1文字以上の文字を挿入したい場合は、次のことを覚えていてください。つまり、画面に表れているテキストは、データソースフィールドの値、または編集中のエリア用の変数にはまだなっていません。エリア用のデータ入力が有効になってから、データソースフィールドや変数は入力値に割り当てられるのです。よってデータ入力を変数に投影し、この投影値を使用して動作を行い、そして入力可能エリアの再割り当てを行うかどうかはユーザ次第となります (下記の例題を参照してください)。

以下のことを行う場合に**FILTER KEYSTROKE**コマンドを使用します。

- カスタマイズされた方法で文字をフィルタする場合
- データ入力フィルタを使用しない方法でデータ入力をフィルタする場合
- 動的ルックアップを実施する場合

警告：FILTER KEYSTROKEコマンドを呼び出してからKeystroke関数を呼び出すと、このコマンドに渡した文字が実際に入力した文字の代わりに戻されてしまいます。

▼ 次のコードを使用します：

```
`「MyObject」入力可能エリアのオブジェクトメソッド
Case of
    \ (Form event=On Load )
        MyObject:=""
    \ (Form event=On Keystroke )
        If (Position(Keystroke ; "0123456789")>0)
            FILTER KEYSTROKE("*")
        End if
End case
```

「MyObject」エリアに入力した数字は、すべてアスタリスク文字に変換されます。

▼ 次のコードは、「パスワード入力」エリアの働きを実施します。このエリア内では、入力した文字がすべて (画面上は)ランダム文字群に置き換えられます：

```
「vsPassword」入力可能エリアのオブジェクトメソッド
Case of
    \ (Form event=On Load )
        vsPassword:=""
        vsActualPassword:=""
    \ (Form event=On Keystroke )
        キーストローク処理 (->vsPassword ; ->vsActualPassword)
        If (Position(Keystroke ; Char(Backspace )+Char(Left Arrow Key )+
            Char(Right Arrow Key )+Char(Up Arrow Key )+Char(Down Arrow Key ))=0)
            FILTER KEYSTROKE(Char(65+(Random%26)))
        End if
End case
```

データ入力が有効になったら、変数「vsActualPassword」内でユーザが入力した正確なパスワードを取り出します。

注：「キーストローク処理」プロジェクトメソッドは、Keystroke関数の例題の項に示されています。

▼ ユーザのアプリケーション内には、ユーザがいくつかの文を入力できるテキストエリアがあります。また、このアプリケーションには、ユーザのデータベースを通じて通常に使用される用語の辞書テーブルが含まれています。自分のテキストエリアを編集しながら、あるテキストエリア内で選択されたキャラクタ群を基にした辞書入力の取り出しや挿入をすぐに行いたい場合、次の2つの方法があります。

■ 割り当てキーを持つボタンを準備する
または
■ テキストエリア編集中に特定のキーストロークを禁止する

上記の説明にあるように、データ入力を有効にした後、テキストエリア編集中にこのエリア用のデータソースを入力値に再割り当てします。このエリアの編集中に辞書入力を取り出したりテキストエリアに挿入したりするためには、データ入力を投影することが必要となります。第1、第2の引数として入力可能エリアと投影変数にポインタを渡します。そして第3の引数として“ forbidden ”文字の文字列を渡します。キーストロークをどのように扱った場合でも、メソッドは元のキーストロークを返します。“ forbidden ”文字とは、ユーザが入力可能エリアに挿入したくない文字また特殊文字として扱いたい文字を指します。

```
` 「投影キーストローク」プロジェクトメソッド
` 投影キーストローク(ポインタ ; ポインタ ; 文字 ) → 文字列
` 投影キーストローク( ->ソースエリア;->現在値;フィルタ) →オリジナルキーストローク
` オリジナルのキーストロークを返す
$0:=Keystroke
  `入力可能領域内のテキスト選択範囲を取得してください。
GET HIGHLIGHT($1-> ; $vlStart ; $vlEnd)
  `現在値で作業を開始してください。
$vtNewValue:=$2->
  `キーを押すか、文字を入力するかで、それぞれに応じた機能が実行されます。
Case of
    `Backspace (Delete) キーを押した場合。
    \ (Ascii ($0)=Backspace )
        `選択した文字またはテキストカーソルの左にある文字が削除されます。
        $vtNewValue:= Delete text ($vtNewValue;$vlStart;$vlEnd)
        `矢印キーを押します。
        `キーストロークを受け付けるだけで何も行わない。
    \ (Ascii ($0)=Left Arrow Key )
    \ (Ascii ($0)=Right Arrow Key )
    \ (Ascii ($0)=Up Arrow Key )
    \ (Ascii ($0)=Down Arrow Key )
    ` 受け付け可能な文字が入力された場合
    \ (Position($0 ; $3)=0)
    $vtNewValue:= Insert text ($vtNewValue ; $vlStart ; $vlEnd ; $0)
Else
    ` 文字が受け付けられない。
    ˝ FILTER KEYSTROKE("")
End case
` 次のキーストローク処理用の値を返す。
$2->:=$vtNewValue
```

これらのプロジェクトにユーザのプロジェクトを加えると、以下のように使用できます。

```
` 「Delete text」プロジェクトメソッド
` Delete text (文字 ; 倍長整数 ; 倍長整数 ) → 文字列
` Delete text ( ->テキスト ; 選択開始 ; 選択最終 ) → 次のテキスト
$0:=Substring($1 ; 1 ; $2-1-Num($2=$3))+Substring($1 ; $3)

` 「Insert text」プロジェクトメソッド
` Insert text (文字 ; 倍長整数 ; 倍長整数 ) → 文字列
` Insert text ( ->ソーステキスト ; 選択開始 ; 選択最終 ;挿入テキスト) -> 次のテキスト
$0:=$1
If ($2#$3)
    $0:=Substring($0 ; 1 ; $2-1)+$4+Substring($0 ; $3)
Else
    Case of
    \ ($2<=1)
        $0:=$4+$0
    \ ($2>Length($0))
        $0:=$0+$4
    Else
        $0:=Substring($0 ; 1 ; $2-1)+$4+Substring($0;$2)
    End case
End if
```

これらのプロジェクトメソッドをあなたのアプリケーションを追加したら、そのメソッドをこのように使用することができます：

```
` 「vsDescription」入力可能エリアのオブジェクトメソッド
Case of
    \ (Form event=On Load )
        vsDescription:=""
        vsShadowDescription:=""
        `特殊キーとして扱う「forbidden」文字のリストを作成できます。
        ` (この例題では、ヘルプキーのみがフィルタされます。)
        vsSpecialKeys:=Char(HelpKey)
    \ (Form event=On Keystroke )
        $vsKey:= Shadow keystroke (->vsDescription ; ->vsShadowDescription ;vsSpecialKeys)
        Case of
            \ (Ascii($vsKey)=Help Key )
            `Help キーを押して何かを行います。
            `この例題では、任意の辞書入力が検索され挿入されます。
                LOOKUP DICTIONARY (->vsDescription ; ->vsShadowDescription)
        End case
End case
```

「LOOKUP DICTIONARY」プロジェクトメソッドを下記に示します。この目的は投影変数を使用して、編集中の入力可能エリアを再割り当てすることです。

```
` 「LOOKUP DICTIONARY」プロジェクトメソッド
` LOOKUP DICTIONARY ( ポインタ ; ポインタ)
` LOOKUP DICTIONARY ( -> 入力可能エリア ; ->投影変数)
`入力可能領域内のテキスト選択範囲を取得してください。
GET HIGHLIGHT($1-> ; $vlStart ; $vlEnd)
    `選択したテキストまたはテキストカーソルの左にある単語を取得します。
$vtHighlightedText:= Get highlighted text ($2-> ; $vlStart ; $vlEnd)
    `検索したいものが何かありますか？
If ($vtHighlightedText#"")
        `テキスト選択がテキストカーソルで行われる場合、
        `その選択はテキストカーソルの後に続く単語から始まります。
    If ($vlStart=$vlEnd)
        $vlStart:=$vlStart-Length($vtHighlightedText)
    End if
        `最初に使用可能な辞書入力を検索します。
    QUERY([辞書];[辞書]入力=$vtHighlightedText+"@")
        `1つあった場合
    If (Records in selection([辞書])>0)
        `あれば、投影テキスト内に挿入します。
        $2->:= Insert text ($2-> ; $vlStart ; $vlEnd ; [辞書]入力)
        `投影テキストを編集中の入力可能テキストにコピーします。
        $1->:=$2->
        `挿入した辞書入力のすぐ後に選択を設定します。
        $vlEnd:=$vlStart+Length([辞書]入力)
        HIGHLIGHT TEXT(vsComments ; $vlEnd ; $vlEnd)
    Else
        `辞書テーブルに合致する入力がない場合。
        BEEP
    End if
Else
` 反転表示されたテキストがない場合
BEEP
End if
```

「Get highlight text」メソッドを次に示します：

```
` 「Get highlighted text」プロジェクトメソッド
` Get highlighted text (文字 ; 倍長整数 ; 倍長整数 ) -> 文字列
` Get highlighted text (テキスト ; 選択開始 ; 選択最終) -> 反転表示されたテキスト
If ($2<$3)
    $0:=Substring($1 ; $2 ; $3-$2)
Else
    $0:=""
    $2:=$2-1
    Repeat
        If ($2>0)
            If (Position($1[[$2]] ; "    „!?:;()-_ミ‟")=0)
                $0:=$1[[$2]]+$0
                $2:=$2-1
            Else
                $2:=0
            End if
        End if
    Until ($2=0)
End if
```

参照：Form event、Keystroke

フォームイベント関数　24

この章では、「ルーチン」エディタの「Form event」テーマ内にあるフォームイベント関数について説明します。この章の関数は、フォームフォームイベントを判定するために使用します。しかし、できる限り、フォームメソッドのフォームイベントに対する判定の代わりにオブジェクトメソッドを使用してください。ここの関数は、フォームメソッドとオブジェクトメソッド以外では無効です。これらはより効果的なコマンドです。

Activated
After
Before
Deactivated
During
Form event
In break
In footer
In header
Outside call

Before

Before → ブール

Before関数は、フォームが画面上に表示される前または印刷する前に“ True(真) ”を返します。Beforeフェーズは、通常、変数やフィールドを初期化するために使用します。

入力フォームがサブフォームを含んでいる場合、Beforeフェーズは最初に組み込まれたレコードまたはサブレコードに対して発生します。そして、最後に親レコードに対して発生します。Beforeフェーズは、ユーザがスクロールした任意のレコードに対しても発生します。また、Beforeフェーズは、サブフォームの新しいレコードまたはサブレコードに対しても発生します。マルチラインのフォームでもフルページのフォームでも発生します。

PRINT SELECTIONコマンド、または、「ユーザ」モードで「プリント」メニューコマンドを選択してレポートを印刷すると、各レコードが印刷される前にBeforeフェーズが発生します。サブフォームの各レコードとサブレコードのBeforeフェーズは、親レコードのBeforeフェーズの後で発生します。データ入力の場合と順序が逆になります。これにより、親レコードのBeforeフェーズで組み込みフォームのレコードまたはサブレコードが選択できるようになります。

DISPLAY SELECTIONコマンド、**MODIFY SELECTION**コマンドまたは「ユーザ」モードで画面上にカレントセレクションを表示する場合に、各レコードが表示されると、BeforeとDuringの両フェーズが同時に“ True(真) ”になります。

▼ 次の例は、フォームが表示される前に、サブレコードのカレントセレクションをソートします：

```
Case of
  \ (Before)
     ORDER SUBRECORDS BY ([親]子供 ; [親]子供'名前 ; >)
End case
```

During

During → ブール

During関数は、オブジェクト(フィールド、変数、ボタン、その他のアクティブエリア)を修正すると、“ True(真) ” を返します。入力フォームのDuringフェーズは通常、データの属性チェック、計算処理、フィールドや変数の更新に使用します。

入力フォームのDuringフェーズは、次の条件で発生します：

■ フィールドや変数を修正し、そこから移動する場合

■ ボタンやチェックボックスをクリックした場合

■ 外部ルーチンエリアをクリックした場合

■「編集」メニュー(ピクチャフィールドで貼り付けを実行した場合は除く)以外のカスタムメニューを選択した場合

■ スクロールエリアから選択した場合

■ 組み込みレコードやサブレコードにデータを入力した場合

■ 組み込みレコードやサブレコードを更新した場合

■ フォームを受け入れた場合

■ フォームを取り消した場合

PRINT SELECTIONコマンドまたは「ユーザ」モードで「プリント」を選択してレポートを印刷する場合に、**During**関数は各レコードごとに “ True ” を返します。

DISPLAY SELECTIONコマンドや**MODIFY SELECTION**コマンドまたは「ユーザ」モードで、画面上にカレントセレクションを表示する場合に、各レコードが表示されるときに、**Before**と**During**の両フェーズが同時に “ True(真) ” になります。また、レコードをダブルクリックしたり、ボタンをクリックした時点で、出力フォームで “ True(真) ” を返します。

▼ 次の例は、**DISPLAY SELECTION**コマンドや**MODIFY SELECTION**コマンドのフォームイベントで発生するすべてのフェーズを判定します。このメソッドは、出力フォームのフォームメソッドです。これを実行させるためには、フッタエリアにカスタムボタンを設定する必要があります。それは、デフォルトの「終了」ボタンはDuringフェーズしか発生しないからです：

```
Case of
   \ (Before & During)
      $イベント:="レコード表示：" + [コマンド]名前
   \ (Before)
      $イベント:="今にもフォームが現われようとしています"
      FromInput:=False                    ` 入力フォームに“True(真)”をセットする
      RecordOK:=True                      ` レコードを表示する
   \ (In header)
      $イベント:="ヘッダが描画されます"
   \ (bボタン1=1)
      $イベント:="ボタン1がクリックされました"
   \ (bボタン1=2)
      $イベント:="ボタン2がクリックされました"
      `必要に応じて、ここで多くのボタンについての検査を追加できます
   \ (b終了=1)
      $イベント:="フォームを離れます"
      CANCEL
   \ (During)
      `このフェーズはレコードがダブルクリックされると実行され、入力レイアウ
      `トを離れたり、または入力フォーム上のレコード間を移動します。
      `各ケースを見つけるには、それらの間のパス情報に使用可能なプロセス変数
      `を保持します。この例では、3番目のケースを管理しません。
      If (FromInput)
         $イベント:="入力フォームを離れました"
         FromInput:=False
      Else
         $イベント:="入力フォームにレコードが表示されます"
         RecordOK:= (Selected record number ([コマンド]) % 2)=0
         If (RecordOK)
            INPUT FORM ([コマンド] ; "あり")
         Else
            INPUT FORM ([コマンド] ; "なし")
         End if
      End if
   Else
      $イベント:="何を行いますか？"
End case
SET WINDOW TITLE ($イベント)
```

この例では、実行しているフェーズを示すために**SET WINDOW TITLE**コマンドを使用しています。プロセス変数“RecordOK”は、レコードを表示するかどうかといった入力フォーム情報に使用されます。例えば、レコードのある値によって、入力フォームのレコードを隠す場合などです。

▼ 次の例は、選択されたレコード番号の決定が基になっています。レコードが偶数の位置にあれば、入力フォームを“ なし ”フォームに変更します。このフォームは、標準のコントロールパネルを表示させないための「透明」ボタンがあること以外はオブジェクトはありません。この“ なし ”入力フォームのフォームメソッドを次に示します：

```
If (Before)
   FromInput:=True                    ` 出力フォームに知らせる
   CANCEL                             ` ただちにフォームを離れる
End if
```

その他の入力フォームにもプロセス変数“ FromInput ”に“ True ”を代入します。これにより、出力フォームのDuringフェーズで入力フォームへ入ったり、離れたりする違いがわかります。

“ Ctrl（Macintosh版では、コマンド）-ピリオド ”キーや“ Enter ”キーは「b終了」ボタンによって機能しなくなります。

フォームメソッドのコメントで説明されているように、入力フォームに移動ボタンが用意されていれば、それらのボタンをクリックすることにより、出力フォームで実行されるDuringフェーズが発生します。そして、これらのボタンのオブジェクトメソッドに1つ以上のプロセス変数を設定することによって、フォームの別のページに移動したり、入力フォームを離れても、それを見つけることができます。

この例は、出力フォームメソッドの構成についてだけでなく、レコードをダブルクリックした際のコマンドの標準的な取り扱いの修正方法についても記述しています。

After

After → ブール

After関数は、新規または修正されたレコードが受け入れられた時点で“ True(真) ”を返します。フォーム上に組み込みテーブルやサブテーブルがある場合に、Afterフェーズは、まず組み込みテーブルまたはサブテーブルの各レコードに対して発生します。

Afterフェーズは、**ADD RECORD**コマンド、 **MODIFY RECORD**コマンド、 **MODIFY SELECTION**コマンドが実行されたときとレコードが受け入れられたときだけに発生します。Afterフェーズは、レコードが「ユーザ」モードで受け入れられた場合にも発生しますが、**DIALOG**コマンドや出力フォームでは発生しません。

カレントレコードを修正しないでレコードを受け入れた場合は、そのレコードはディスクに上書きしません。従って、Afterフェーズも発生しません。この場合、Beforeフェーズまたは**During**フェーズで、それ自体のフィールドに代入し直して修正されたかのように見せかけることによってAfterフェーズを強制的に発生させることができます。

▼ 次の例は、レコードのフィールドに修正された日付を代入するためのステートメントです：

```
Case of
  \ (Before)
...
  \ (During)
...
  \ (After)                                  ` レコードを保存している場合
    Last Modified:=Current Date              ` 修正した日付を保存する
End case
```

In header

In header → ブール

In header関数は、出力フォームのヘッダエリアが印刷される直前に“ True ”を返します。

フォームのヘッダエリアは、ヘッダマーカ(Hが付いています)の上のフォームの最上部のエリアです。ヘッダはレポートの各ページの一番上に印刷されます。各ブレークレベルにヘッダエリアを設定することもできます。

Before selection関数を使用してレポートの先頭を判定することができます。**Before selection**関数は最初のヘッダが印刷されるときに“ True ”を返します。

In header関数は、出力フォームのヘッダエリアが画面上に表示される直前にも“ True ”を返します。

▼ 次の例は、フォームメソッドのテンプレートで、それぞれの出力フェーズを判定します：

```
Case of
   \ (In header)
      Case of
         \ (Before selection)
            ` ここに最初のヘッダのコードが入る
         \ (Level=1)
            ` ここにブレークヘッダレベル1のコードが入る
            ` 必要ならば何個でもブレークレベルを判定することができる
      End case
   \ (During)
      ` ここにレコードのコードが入る
   \ (In break)
      Case of
         \ (Level=0)
            ` ここにブレークレベル0のコードが入る
         \ (Level=1)
            ` ここにブレークレベル1のコードが入る
            ` 必要ならば何個でもブレークレベルを判定することができる
      End case
   \ (In footer)
      If (End selection)
         ` ここに最後のフッタのコードが入る
      Else
         ` ここにフッタのコードが入る
      End if
End case
```

In break

In break → ブール

In break関数は、出力フォームのブレークエリアが印刷される直前に“ True ”を返します。

In break関数は、各ブレークに対して“ True ”を返します。つまり、ブレークレベルが変更されるたびに“ True ”を返します。

フォームレポートのブレーク処理を開始する方法が2つあります。

前ページの**In header**関数の例を参照してください。

In footer

In footer → ブール

In footer関数は、出力フォームのフッタエリアが印刷される直前に“ True ”を返します。

フォームのフッタエリアは、フッタマーカ(Fが付いています)の上でブレークマーカの下(Bが付いています)までの部分です。フッタは、レポートの各ページの一番下に印刷されます。

End selection関数を使用して、レポートの最後を判定することができます。**End selection**関数は、最後のフッタが印刷される時点で“ True ”を返します。

前ページの**In header**関数の例を参照してください。

Form event

Form event → 数値

引数	タイプ		説明
このコマンドには、引数はありません。			
関数が返す値	数値	←	フォームイベント番号

Form eventt関数は、生じたばかりのフォームイベントの型を認識している絶対値を返します。通常、ユーザはフォームやオブジェクトメソッド内から**Form event**t関数を使用します。

4th Dimensionは、前もって定義された次のような定数を持っています：

定数	値	説明
On Load	1	フォームは表示または印刷されようとしています。
On Unload	24	フォームは終了または解放されようとしています。
On Validate	3	レコードのデータ入力が有効となりました。
On Clicked	4	オブジェクトがクリックされました。
On Double Clicked	13	オブジェクトがダブルクリックされました。
On Keystroke	17	文字がフォーカスを持つオブジェクト内に入力されようとしています。
On Getting Focus	15	フォームオブジェクトがフォーカスを獲得します。
On Losing Focus	14	フォームオブジェクトがフォーカスを失います。
On Activate	11	フォームのウインドウが最前列のウインドウになります。
On Deactivate	12	フォームのウインドウはもはや最前列のウインドウではありません。
On Outside	10	フォームが **CALL PROCESS** コマンド呼び出しを受け取りました。
On Drop	16	データがオブジェクト上にドロップしました。
On Drag Over	21	データがオブジェクト上にドロップする可能性があります。
On Menu Selected	18	メニューアイテムが選択されました。
On Data Change	20	ObjectData が変更されました。
On External Area	19	外部ルーチンエリアが自身の実施すべきオブジェクトメソッドを要求しました。
On Printing Header	5	フォームのヘッダエリアが印刷されようとしています。
On Printing Detail	23	フォームのディテイルエリアが印刷されようとしています。
On Printing Break	6	フォームのブレークエリアの1つが印刷されようとしています。

On Printing Footer	7	フォームのフッタエリアが印刷されようとしています。
On Close Box	22	ウインドウのクローズボックスがクリックされました。
On Display Detail	8	レコードがリスト内に表示されようとしています。
On Open Detail	25	レコードがダブルクリックされ、ユーザは入力フォームに移ります。
On Close Detail	26	ユーザは入力フォームを残したまま出力フォームに戻ります。

イベントとメソッド

フォームイベントが生じると、4th Dimensionは次のようなアクション（動作）を実行します。

■ 4th Dimensionはまず、フォームのオブジェクトを順々にブラウズしていき、次に (イベント内に含まれている) オブジェクト用のオブジェクトメソッドを呼び出します。このとき、そのオブジェクトに対応するオブジェクトのイベントプロパティは選択されています。

■ 次に、対応するフォームのイベントプロパティが選択されている場合、4th Dimension はフォームメソッドを呼び出します。

オブジェクトメソッドは、それが存在する場合、特別な順序で呼び出されることはありません。大雑把に言えば、オブジェクトメソッドは常にフォームメソッドの前に呼び出されます。オブジェクトがサブフォームである場合、サブフォームのリストフォームのオブジェクトメソッドが呼び出され、それからリストフォームのフォームメソッドが呼び出されます。その後、4th Dimension は親フォームのオブジェクトメソッドを呼び出します。つまり、オブジェクトがサブフォームである場合、4th Dimensionはサブフォームオブジェクト内のオブジェクトとフォームメソッドを上述と同じように使用します。

フォームのイベントプロパティが与えられたイベント用に選択されていない場合でも、同じイベントプロパティが選択されているオブジェクトであれば、このオブジェクト用のオブジェクトメソッドへの呼び出しが妨げられることはありません。つまり、フォームレベルでのイベントの有効または無効はオブジェクトのイベントプロパティに影響を与えないということです。

1つのイベント内に含まれるオブジェクト数は、イベントの性質によって異なります。

■ On Loadイベント - On Loadオブジェクトイベントの属性が選択されている(あらゆるページからの) フォームのオブジェクトは、すべてこれらのオブジェクトメソッドを呼び出させます。そして、On Loadフォームイベントプロパティが選択される場合、そのフォームは自身のフォームメソッドが呼び出されることを確認します。

■ On Activateイベント - どのオブジェクトメソッドも呼び出されません。なぜならこのイベントは特定のオブジェクトに適用されるのではなく、全体としてのフォームに適用

されるからです。必然的に On Activateフォームイベントプロパティが選択される場合、フォームだけがフォームメソッドが呼び出されることを確認します。

■ On Drag Overイベント - On Drag Overオブジェクトイベントの属性が選択される場合、イベント内に含まれているドロップ可能なオブジェクトだけが、自身のオブジェクトメソッドが呼び出されることを確認します。フォームメソッドが呼び出されることはありません。

警告：他のイベントとは対照的に、On Drag Over**イベント中は、オブジェクト用のオブジェクトメソッドがドラッグ&ドロップソースオブジェクトのプロセスコンテキスト内で実施されます。ドラッグ&ドロップの送信先オブジェクトのプロセスコンテキスト内で実施されることはありません。詳細は、DRAG AND DROP PROPERTIESコマンドとDrop position関数を参照してください。**

次の表は、オブジェクトメソッドとフォームメソッドを各イベントタイプごとに呼び出す方法を要約したものです。

定数	オブジェクトメソッド	フォームメソッド	対象オブジェクト
On Load	はい	はい	全オブジェクト
On Unload	はい	はい	全オブジェクト
On Validate	はい	はい	全オブジェクト
On Clicked	はい(クリック可能な場合) (*)	はい	関連するオブジェクトのみ
On Double Clicked	はい(クリック可能な場合) (*)	はい	関連するオブジェクトのみ
On Keystroke	はい(キーボード入力可能な場合) (*)	はい	関連するオブジェクトのみ
On Getting Focus	はい(クリック可能な場合) (*)	はい	関連するオブジェクトのみ
On Losing Focus	はい(クリック可能な場合) (*)	はい	関連するオブジェクトのみ
On Activate	なし	はい	なし
On Deactivate	なし	はい	なし
On Outside	なし	はい	なし
On Drop	はい(ドロップ可能な場合) (*)	はい	関連するオブジェクトのみ
On Drag Over	はい(ドロップ可能な場合) (*)	なし	関連するオブジェクトのみ
On Menu Selected	はい	はい	全オブジェクト
On Data Change	はい(変更可能な場合) (*)	はい	関連するオブジェクトのみ
On External Area	はい	はい	関連するオブジェクトのみ
On Printing Header	はい	はい	全オブジェクト
On Printing Detail	はい	はい	全オブジェクト
On Printing Break	はい	はい	全オブジェクト
On Printing Footer	はい	はい	全オブジェクト
On Close Box	なし	はい	なし
On Display Detail	はい	はい	全オブジェクト
On Open Detail	なし	はい	なし
On Close Detail	なし	はい	なし

(*)の印の付いた箇所に関する詳細は、後述の「イベント、オブジェクト、プロパティ」の節を参照してください。

重要：どのイベントであっても、対応するイベントのプロパティがそのフォームやオブジェクト用に選択されている場合は、そのフォームやオブジェクトのメソッドが呼び出されることを覚えていください。「デザイン」モードで (「フォームプロパティ」および「オブジェクトプロパティ」ウインドウを使用して)イベントを無効にすると、ユーザがメソッドを呼び出す回数をかなり縮小でき、これによりユーザはフォームの実施速度をかなり速めることができるというメリットがあります。

警告：On Load**と**On Unload**イベントのオブジェクトとフォーム両方にイベントが有効となる場合、**On Load**と**On Unload**イベントはオブジェクト用に生成されます。イベントがオブジェクト用にだけ有効となる場合、これは起こりません。また、この 2つのイベントが有効になるにはフォームレベルである必要があります。**

イベント、オブジェクト、プロパティ

イベントが、その性質とプロパティに応じて、オブジェクト用に実際に生じうる場合、オブジェクトメソッドが呼び出されます。次の節では、ユーザが様々なオブジェクトタイプを操作する際に一般的に使用するイベントを詳しく説明します。

クリック可能なオブジェクト

クリック可能なオブジェクトは、主にマウスで操作します。これには、次のものが含まれます。

- ブール入力可能フィールドや変数
- 表示フォーマットが「バックグラウンド」に設定されているピクチャフィールドやピクチャ変数
- ボタン、デフォルトボタン、ラジオボタン、チェックボックス、ボタングリッド
- 3Dボタン、3Dラジオボタン、3Dチェックボックス
- ポップアップメニュー、階層ポップアップメニュー、ピクチャメニュー
- ドロップダウンリスト、メニューまたはドロップダウンリスト
- スクロール可能エリア、階層リスト
- 透明ボタン、ハイライトボタン、ラジオピクチャ
- サーモメータ、ルーラ、ダイアル (これらは、スライダーオブジェクトとも呼ばれます)
- タブコントロール

On Clickedと On Double Clickedオブジェクトのイベントプロパティが、これらオブジェクトの 1つ用に選択されると、ユーザはオブジェクト内またはオブジェクト上のクリックを見つけて操作することができるようになります。この場合、状況に応じて On ClickedやOn Double Clickedを返す**Form event**関数を使用します。

すべてのオブジェクトに対し、いったんマウスボタンがリリースされると、On Clickedイ

ベントが生じます。ただし次の 2つの例外があります：

■ **透明ボタン** - クリックされると、すぐにOn Clickedイベントが生じます。マウスボタンがリリースされることを待ちません。

■ **スライダーオブジェクト(サーモメータ、ルーラ、ダイアル)** - ユーザがコントロールをスライドしている最中にオブジェクトメソッドを呼び出す必要ができたことを表示フォームが示した場合、クリックされると、すぐにOn Clickedイベントが生じます。

注：キーボードで使用できるオブジェクトもあります。例えば、チェックボックスがフォーカスを獲得すると、スペースバーを使用してこれを入力できます。このような場合でも、On Clicked**イベントはそのまま生成されます。**

警告：コンボボックスはクリック可能なオブジェクトとは考えられていません。コンボボックスは、入力可能なテキストエリアとして扱う必要があります。この入力可能なテキストエリアに関連するドロップダウンリストはデフォルトの値を提供します。必然的に、ユーザは On Keystroke**と** On Data Change**イベントを通じて、コンボボックス内でデータ入力操作を行うことになります。**

キーボード入力可能なオブジェクト

キーボード入力可能なオブジェクトとは、キーボードを使用してユーザがデータを入力するオブジェクトのことであり、ユーザはそのオブジェクト用に On Keystrokeイベントを見つけることにより最下位レベルでデータ入力をフィルタします。これには、次のものが含まれます：

■ すべての入力可能なフィールドオブジェクト(ピクチャ、サブテーブル、BLOBは除く)
■ すべての入力可能な変数(ピクチャ、BLOB、ポインタ、配列は除く)
■ コンボボックス
■ 階層リスト

On Keystrokeオブジェクトのイベントプロパティがオブジェクトの 1つ用に選択されると、ユーザはこのオブジェクト内でキーストロークを見つけて操作できます。この場合、On Keystrokeイベントを返す**Form event**関数を使用します。

変更可能なオブジェクト

変更可能なオブジェクトはデータソースを持っており、このデータソースの値はマウスやキーボードで変更できます。ただし、これらは、実際には On Clickedイベントを通じて操作するユーザインタフェースのコントロールとは考えられていません。これには、次のものが含まれます：

■ すべての入力可能なフィールドオブジェクト(サブテーブル、BLOB は除く)
■ すべての入力可能な変数(BLOB、ポインタ、配列は除く)
■ コンボボックス
■ 外部オブジェクト (外部オブジェクト用のデータ全体入力は 4D Extension で受け付けられます)

これらのオブジェクトが On Data Changeイベントを受け取ります。On Data Changeオブジェクトのイベント属性がこれらのオブジェクトの 1 つ用に選択された後、ユーザはデータソースの値の変更を見つけて操作することができます。この場合、On Data Changeを返す **Form event**関数を使用します。

タブ使用可能なオブジェクト

ユーザがタブ使用可能なオブジェクトに到達するためにそして / またはこれらをクリックするために Tab キーを使用すると、タブ使用可能なオブジェクトはフォーカスを獲得します。フォーカスを持っているオブジェクトは (キーボード上でタイプされた) 文字を受け取ります。この文字は、メニューコマンドやボタンのようなオブジェクトへの (Windows での)アクセラレータや (MacOs での)ショートカットではありません。

以下のものを除き、オブジェクトはすべてタブ使用可能です：

■ 入力不可能なフィールドや変数
■ (MacOS で使用される)ボタン
■ ボタングリッド
■ 3Dボタン、3Dラジオボタン、3D チェックボックス
■ ポップアップメニュー、階層ポップアップメニュー
■ (MacOS で使用される)メニューまたはドロップダウンリスト
■ ピクチャメニュー
■ スクロール可能エリア
■ 透明ボタン、ハイライトボタン、ラジオピクチャ
■ グラフ
■ 外部オブジェクト (外部オブジェクト用の全データ入力は、 4D Extension で受け付けられません)
■ タブコントロール

On Getting Focusと / または On losing Focus オブジェクトのイベント属性が、タブ使用可能なオブジェクト用に選択された後、ユーザはフォーカスの変更を見つけて操作することができるようになります。この場合、状況に応じてOn Getting FocusまたはOn losing Focusを返す**Form event**関数を使用します。

フォームイベントは、次のカテゴリーに分類できます。

- 一般的なイベント
- フォームに適応したイベント
- ユーザアクション（動作）に関連したイベント
- ドラッグ&ドロップのイベント
- 印刷用のイベント

バージョン6とバージョン3の間の互換性について

次の表は、バージョン6のフォームイベントとバージョン3の レイアウト実行サイクルの間で同等の意味を持つものを示しています。

バージョン6のイベント	バージョン3の実行サイクル	バージョン3のコマンド
On Load	Beforeフェーズ	**Before**
On Unload	同等の実行サイクルはありません	なし
On Validate	Beforeフェーズ	**Before**
On Clicked	一般的なDuringフェーズ	**During**
On Double Clicked	一般的なDuringフェーズ	**During**
On Keystroke	同等の実行サイクルはありません	なし
On Getting Focus	同等の実行サイクルはありません	なし
On Losing Focus	同等の実行サイクルはありません	なし
On Activate	Activatedフェーズ	**Activated**
On Deactivate	Deactivatedフェーズ	**Deactivated**
On Outside Call	Outside Callフェーズ	**Outside Call**
On Drop	同等の実行サイクルはありません	なし
On Drag Over	同等の実行サイクルはありません	なし
On Menu Selected	一般的なDuringフェーズ	**During**と**Menu selected**
On Data Change	一般的なDuringフェーズ	**During**
On External Area	一般的なDuringフェーズ	**During**
On Printing Header	印刷時のヘッダフェーズ	**In header**
On Printing Detail	一般的なDuringフェーズ	**During**
On Printing Break	印刷時のブレークフェーズ	**In break**
On Printing Footer	印刷時のフッタフェーズ	**In footer**
On Close Box	同等の実行サイクルはありません	**OPEN WINDOW**
On Display Detail	一般的なDuringフェーズ	**Before**と**During**
On Open Detail	一般的なDuringフェーズ	**During**
On Close Detail	一般的なDuringフェーズ	**During**

ユーザがバージョン6の4th Dimensionを使用してバージョン3 のデータベースを開くと、プログラムは 2つのオペレーションを実行します。

- ストラクチャファイルの新規フォーマットへの変換
- データファイルの新規フォーマットへの変換

変換後のデータベースを使用する際、もし「デザイン」モードでフォームが編集または変更されていない場合、フォームはバージョン3 のときと同じ状態でストラクチャファイル内にまだ記憶されていることになります。ユーザのバージョン3の既存 アプリケーションとの互換性を確実にするために、フォームとオブジェクトのイベントプロパティは自動的に設定され、その設定は「ala V1」に反映します。つまり、バージョン6のイベントプロパティが自動的に選択され、そして「古い V1 コマンド」はバージョン 1 のときと同じように機能します。

バージョン3の「変更時のみ実行」オプションを持った任意のオブジェクト（フィールドまたは変数）を選択した場合、そのイベントプロパティはバージョン3のデータ入力中に発生する**During**実行サイクルに適合するプロパティに縮小されます：

バージョン6のイベント	バージョン3の実行サイクル	バージョン3のコマンド
On Clicked	一般的なDuringフェーズ	**During**
On Double Clicked	一般的なDuringフェーズ	**During**
On Data Change	一般的なDuringフェーズ	**During**
On External Area	一般的なDuringフェーズ	**During**

いったん、ユーザがバージョン6でフォームとそのオブジェクトの編集を開始すると、フォームとオブジェクトのイベントプロパティは、デフォルトでは同じ方式で設定されます。バージョン6の新規イベントを利用するには、「デザイン」モードでフォームとオブジェクト用のイベントプロパティを選択し、それから **Form event**関数を使用してフォームメソッドとオブジェクトメソッドを変更します。

バージョン3の実行サイクルに対応しないバージョン6の新規イベントは、次の通りです：

バージョン6のイベント	バージョン3の実行サイクル	バージョン3のコマンド
On Unload	同等の実行サイクルはありません	なし
On Keystroke	同等の実行サイクルはありません	なし
On Getting Focus	同等の実行サイクルはありません	なし
On Losing Focus	同等の実行サイクルはありません	なし
On Drop	同等の実行サイクルはありません	なし
On Drag Over	同等の実行サイクルはありません	なし
On Close Box	同等の実行サイクルはありません	**OPEN WINDOW**

新規イベントを使用することで、ユーザはイベントの性質により適合する動作を実行できるようになります：

バージョン6のイベント	バージョン3の実行サイクル	バージョン3のコマンド
On Clicked	一般的なDuringフェーズ	**During**
On Double Clicked	一般的なDuringフェーズ	**During**
On Menu Selected	一般的なDuringフェーズ	**During**と**Menu selected**
On Data Change	一般的なDuringフェーズ	**During**
On External Area	一般的なDuringフェーズ	**During**
On Printing Detail	一般的なDuringフェーズ	**During**
On Display Detail	一般的なDuringフェーズ	**Before**と**During**
On Open Detail	一般的なDuringフェーズ	**During**
On Close Detail	一般的なDuringフェーズ	**During**

ここで説明される例題は、すべてフォームとオブジェクトのプロパティが適切に選択されていることを前提としています。

▼ 次の例は、[親]テーブルのフォームが画面上に表示される前に、[親]子供サブテーブルのサブレコードの選択がソートします：

```
` [親]テーブルのフォームメソッド
Case of
    \ (Form event=On Load)
        ORDER SUBRECORDS BY([親]子供 ; [親]子供'名字 ; >)
        ...
End case
```

▼ 次の例は、レコードが変更される日付を (フィールドに) 自動的に割り当てるために使用されている On Validateイベントを示しています：

```
`任意のフォームメソッド
Case of
        ...
    \ (Form event=On Validate)
        [テーブル]最新修正日:=Current date
End case
```

▼ 次の例は、ドロップダウンリスト (初期化、ユーザによるクリック、オブジェクトのリリース) の操作はすべてオブジェクトのメソッド内に含まれていることを示しています：

```
`「asBurgerSize」ドロップダウンリストのオブジェクトメソッド
Case of
    \ (Form event=On Load)
        ARRAY STRING(31 ; asBurgerSize ; 3)
        asBurgerSize{1}:="小"
        asBurgerSize{1}:="中"
        asBurgerSize{1}:="大"
    \ (Form event=On Clicked)
        If (asBurgerSize#0)
            ALERT("サイズ "+asBurgerSize{asBurgerSize}+" のハンバーガーを1つ！")
        End if
    \ (Form event=On Unload)
        CLEAR VARIABLE(asBurgerSize)
End case
```

▼次の例は、ピクチャ値を受け取るだけのフィールドオブジェクト用のドラッグ&ドロップ操作のオブジェクトメソッド内での受け取りとその後の操作方法を示しています：

注：下記の例では、4th Dimensionに用意されていないコマンドが使用されているため、4th Dimensionでは行うことができません。参考までに掲載してます。

```
` [テーブル]ピクチャ入力可能ピクチャフィールドのオブジェクトメソッド
Case of
   \ (Form event=On Drag Over)
      `ドラッグ&ドロップ操作を開始します。マウスはカレントフィールド中に渡っています。
      `ソースオブジェクトについての情報を獲得してください。
      DRAG AND DROP PROPERTIES ($vpSrcObject ; $vlSrcElement ; $lSrcProcess)
      `オブジェクトメソッドが例外的にプロセスのコンテキスト内で実施されているた
      `めユーザ側はソースプロセスの ID 番号をテストする必要がないことに注意して
      `ください。
      $vlDataType:=Type ($vpSrcObject->)
      `ソースデータはピクチャ (フィールド、変数、配列 ) ですか？
      If (($vlDataType=Is Picture) | ($vlDataType=Picture Array))
         `正しい場合、ドラッグを受け付けてください。
         `マウスボタンが押されたままであることに注意してください。ドラッグが
         `4th Dimensionにオブジェクトをハイライトさせていることを受け付けている
         `間だけ効果があります。これでユーザはソースデータがオブジェクト上に
         `ドロップしたかを知ります。
         $0:=0
      Else
         `ドラッグを受け付けない場合
         $0:=-1
         `オブジェクトはハイライトしません。
      End if
   \ (Form event=On Drop)
      `ソースデータがオブジェクト上にドロップしました。よってユーザ側はこれを
      `そのオブジェクトにコピーする必要があります。
      `ソースオブジェクトについての情報を獲得してください。
      DRAG AND DROP PROPERTIES ($vpSrcObject ; $vlSrcElement ; $lSrcProcess)
      $vlDataType:=Type ($vpSrcObject->)
      Case of
         `ソースオブジェクトはピクチャ フィールドまたはピクチャ変数です。
         \ ($vlDataType=Is Picture)
            `ソースオブジェクトは同じプロセスから (つまり同じウインドウと
            `フォームから)来ていますか？
            If ($lSrcProcess=Current process)
               `正しい場合、ソースの値のコピーだけを行ってください。
               [テーブル]ピクチャ:=$vpSrcObject->
            Else
               ` 違う場合、ソースオブジェクトは変数ですか？
```

```
                If (Is a variable ($vpSrcObject))
                    `正しい場合、ソースプロセスからその値を獲得してください。
                    GET PROCESS VARIABLE ($lSrcProcess ; $vpSrcObject-> ;
                        $vgDraggedPict)
                    [テーブル]ピクチャ:=$vgDraggedPict
                Else
                    `違う場合、ソースプロセスからフィールドの値を獲得するために
                    `CALL PROCESSコマンドを使用してください。
                End if
            End if
            `ソースオブジェクトはピクチャ配列です。
        \ ($vlDataType=Picture Array)
            `ソースオブジェクトは同じプロセスから ( つまり同じウインドウと
            `フォームから ) 来ていますか ?
            If ($lSrcProcess=Current process)
                `正しい場合、ソースの値のコピーだけを行ってください。
                [テーブル]ピクチャ:=$vpSrcObject->{$vlSrcElement}
            Else
                `違う場合、ソースプロセスからその値を獲得してください。
                GET PROCESS VARIABLE ($lSrcProcess ; $vpSrcObject
                    ->{$vlSrcElement} ; $vgDraggedPict)
                [テーブル]ピクチャ:=$vgDraggedPict
            End if
    End case
End case
```

注：On Drag Over**と** On Drop**イベントを操作する方法を示す例題に関しては、DRAG AND DROP PROPERTIESコマンドの例題を参照してください。**

▼次の例は、フォームメソッド用のテンプレートです。ここでは、集計リポートフォームを出力フォームとして使用している間に生じる可能性のある各イベントを示しています：

```
`集計レポート用の出力フォームとして使用されるフォームのメソッドM
$vpFormTable:=Current form table
Case of
   ...
   \ (Form event=On Printing Header)
      `ヘッダエリアが印刷されようとしています。
      Case of
         \ (Before selection($vpFormTable->))
            `最初のブレークヘッダ用のコードはここです。
         \ (Level = 1)
            `ブレークヘッダレベル 1 用のコードはここです。
         \ (Level = 2)
            `ブレークヘッダレベル 2 用のコードはここです。
            ...
      End case
   \ (Form event=On Printing Details)
      `レコードが印刷されようとしています。
      ` 各レコード用のコードはここです。
   \ (Form event=On Printing Break)
      ` ブレークが印刷されようとしています。
      Case of
         : (Level = 0)
            `ブレークレベル 0 用のコードはここです。
         : (Level = 1)
            `ブレークレベル 1 用のコードはここです。
            ...
      End case
   \ (Form event=On Printing Footer)
      If(End selection($vpFormTable->))
         `最後のフッタ用のコードはここです。
      Else
         ` フッタ用のコードはここです。
      End if
End case
```

▼ 次の例は、イベントを操作するフォームメソッドのテンプレートを示しています。このイベントは、**DISPLAY SELECTION**コマンドや **MODIFY SELECTION**コマンドを使用して表示されるフォーム用に生じる可能性があるイベントです。教授的な目的で、これはフォームウインドウのタイトルバー内にあるイベントの性質を表示しています：

```
`任意のフォームメソッド
Case of
   \ (Form event=On Load)
      $vsTheEvent:="フォームが表示されようとしています。"
   \ (Form event=On Unload)
      $vsTheEvent:="出力フォームが終了し、画面から消えようとしています。"
   \ (Form event=On Display Details)
      $vsTheEvent:="表示しているレコード番号#" +String (Selected record number([テーブル]))
   \ (Form event=On Menu Selected)
      $vsTheEvent:="メニューコマンドが選択されています。"
   \ (Form event=On Printing Header)
      $vsTheEvent:="ヘッダエリアが描かれようとしています。"
   \ (Form event=On Open Details)
      $vsTheEvent:="レコード番号 #"+String(Selected record number([テーブル]))+
                  " がダブルクリックされました。"
   \ (Form event=On Close Details)
      $vsTheEvent:="出力フォームに戻ります。"
   \ (Form event=On Activate)
      $vsTheEvent:="フォームのウインドウが最前面ウインドウになります。"
   \ (Form event=On Deactivate)
      $vsTheEvent:="フォームのウインドウはもはや最前面ウインドウではありません。"
   \ (Form event=On Menu Selected)
      $vsTheEvent:="メニューコマンドが選択されました。"
   \ (Form event=On Outside call)
      $vsTheEvent:="他からの呼び出しが受け付けられました。"
Else
   ` $vsTheEvent:="What's going on? Event #"+String(Form event)
End case
SET WINDOW TITLE ($vsTheEvent)
```

▼ On Keystrokeイベントの操作方法の例題に関しては、**Keystroke** 関数と **FILTER KEYSTROKE**コマンドの例題を参照してください。

▼次の例は、スクロール可能エリアとしてクリックとダブルクリックを同じように扱う方法を示しています：

```
`「asChoices」スクロールエリアのオブジェクトメソッド
Case of
   \ (Form event=On Load)
      ARRAY STRING (...; asChoices ;...)
       ...
      asChoices:=0
   \ ((Form event=On Clicked) | (Form even=On Double Clicked))
      If (asChoices#0)
         ` アイテムをクリックすると、ここで何かを行います。
          ...
      End if
       ...
End case
```

▼次の例は、それぞれ異なる応答を使用するクリックとダブルクリックを扱う方法を示しています。選択した要素のトラックを維持するために要素ゼロを使用するすることに注意してください：

```
`「asChoices」スクロールエリアのオブジェクトメソッド
Case of
   \ (Form event=On Load)
      ARRAY STRING (...; asChoices;...)
       ...
      asChoices:=0
      asChoices{0}:="0"
   \ (Form event=On Clicked)
      If (asChoices#0)
         If (asChoices # Num(asChoices))
            `新規アイテムをクリックすると、ここで何かを行います。
            ...
            `次回のために新しく選択した配列要素を保存してください。
            asChoices{0}:=String (asChoices)
         End if
      Else
         asChoices:=Num(asChoices{0})
      End if
   \ (Form event=On Double Clicked)
      If (asChoices#0)
         `新規アイテムをダブルクリックすると、ここで今までと違う何かを行います。
      End if
          ...
End case
```

▼ この例題では、フォームメソッド内からステータスのテキスト情報領域を維持する方法を示しています。この場合、On Getting Focus と On Losing Focusイベントを使用します：

```
` [交渉];"データ入力" フォームメソッド
Case of
   \ (Form Event=On Load)
      C_TEXT (vtStatusArea)
      vtStatusArea:=""
   \ (Form Event=On Getting Focus)
      RESOLVE POINTER (Last object ; $vsVarName ; $vlTableNum ; $vlFieldNum)
      If (($vlTableNum#0) & ($vlFieldNum#0))
         Case of
            \ ($vlFieldNum=1) `名字フィールド
               vtStatusArea:="交渉した人の名字を入力してください。これは自動的
                              に大文字になります。"
               ...
            \ ($vlFieldNum=10) ` 郵便番号フィールド
               vtStatusArea:="5桁の郵便番号を入力してください。これは自動的
                              にチェックされ有効になります。"
               ...
         End case
      End if
   \ (Form Event=On Losing Focus)
      vtStatusArea:=""
      ...
End case
```

▼ この例題では、レコードデータ入力用に使用されるフォームを持つクローズウインドウイベントへの応答方法を示しています：

```
`データ入力用のフォームメソッド
$vpFormTable:=Current form table
Case of
        ...
    \ (Form Event=On Close Box)
        If (Modified record($vpFormTable->))
            CONFIRM ("このレコードは変更されました。変更を保存しますか？")
            If (OK=1)
                ACCEPT
            Else
                CANCEL
            End if
        Else
            CANCEL
        End if
            ...
End case
```

▼ この例題では、テキストや文字・数字両方からなるフィールドのデータソースの値が変更される度に、テキストや文字・数字両方からなるフィールドを大文字にする方法を示しています：

```
` [交渉]名字フィールドのオブジェクトメソッド
Case of
        ...
    \ (Form event=On Data Change)
        [交渉]名字:= Uppercase (Substring ([交渉]名字 ; 1 ; 1))
                +Lowercase (Substring ([交渉]名字 ; 2))
        ...
End case
```

参照：CALL PROCESS、Current form table、DRAG AND DROP PROPERTIES、FILTER KEYSTROKE、Keystroke

Outside call

Outside call → ブール

当該プロセスに対して**CALL PROCESS**コマンドが呼び出されている場合に、**Outside call**関数は“ True ”を返します。**Outside call**関数は**CALL PROCESS**コマンドで呼び出されたプロセスの最前面のウインドウにのみ“ True ”を返します。**Outside call**関数はフォームメソッドまたはオブジェクトメソッドで使用されます。

▼ 次の例は、クロックの継続的な更新方法を示しています。これは、プロセス間通信のさまざまな側面を示します。クロックのようなルーチンを実現するには、2つのプロセスが必要です。1つは現在時刻を表示し、もう1つは最初のプロセスに更新時期を知らせます。

最初のプロセス“ Show Clock ”は、プロジェクトメソッド“ クロック表示 ”をメニューアイテムに割り当て、そのメニューコマンドの「新規プロセス」チェックボックスをチェックして作成します。“ クロック表示 ”メソッドは、インタープロセス変数“ <>Done ”にRefreshClockプロセスのコントローラの役割を課します。そして、RefreshClockプロセスで使用するインタープロセス変数に自分のプロセス参照番号を保存します。次に、別のRefreshClockプロセスを起動します。

このプロセスはインタフェースを持たず、メソッド呼び出しやレコードをスタックしませんので、最小スタックサイズを指定します。次に**OPEN WINDOW**コマンドを使用してフローティングウインドウを開きます。フローティングウインドウのクローズボックスメソッドは、**CANCEL**コマンドを使ってダイアログボックスを閉じます。次に、**DIALOG**コマンドを使用してクロックを表示します。このダイアログのフォームメソッドは現在時間を変数“ vTime ”に割り当てます。“ クロック表示 ”メソッドは次のようになります：

```
C_BOOLEAN (<>Done)
C_LONGINT (<>pClock)
`<>Done=Trueならば、RefreshClockプロセスは停止する
<>Done:=False
`プロセス参照番号を得る
<>pClock:=Current process
`RefreshClockを別プロセスとして起動する
$RefreshID:=New process ("RefreshClock" ; 32000 ; "RefreshClock")
`フローティングパレットを開く
OPEN WINDOW (50 ; 50 ; 110 ; 70 ; -724 ; "" ; "CloseProc")
`クロックフォームを表示する
DIALOG ([従業員]" ; クロック")
`ウインドウを閉じる
CLOSE WINDOW
```

```
`RefreshClockプロセスを停止する
<>Done:=True
`遅延している場合は実行を再開する
RESUME PROCESS ($RefreshID)
```

「CloseProc」メソッドを次に示します：

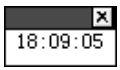
```
CANCEL
```

RefreshClockメソッドは、開始と同時に**Repeat**ループを繰り返してShow Clockプロセスを呼び出します。変数 " <>pClock " にはShow Clockプロセスのプロセス参照番号情報が入っています。RefreshClockは他のプロセスを呼び出すごとに実行を1秒停止します。これによって、RefreshClockが1秒間に何回も実行されて他のプロセスの処理を妨げるのを防ぎます。クロックは1秒に1回だけ更新しますのでこれで十分です。秒のない形でクロックを表示すると、RefreshClockを1分ずつ停止させることができ、さらに効率が上がります。以下にRefreshClockプロセスを示します：

次は、「RefreshClock」メソッドです：

```
C_TIME (<>Delay)
<>Delay:=?00:00:01?
Repeat
`ShowClockプロセスを呼び出す
   CALL PROCESS (<>pClock)
   `Updaterプロセスを1秒停止する
   DELAY PROCESS (Current process ; <>Delay)
   `クロックが閉じられると<>DoneがTrueになる
Until (<>Done)
```

このクロックは、フローティングウインドウに表示されます：



次にクロックダイアログボックスのフォームメソッドを示します。これが呼び出されるごとにクロックが更新されます。このフォームには透明ボタンがありますのでデフォルトボタンは表示されません。このフォームは、ウインドウのクローズボックスをクリックして閉じます：

```
Case of
   \ (Before)
      `Beforeフェーズ
...
   \ (Outside call)
     vTime:=Current time
End case
```

この例は、処理を終了する時に、関連するすべてのプロセスを閉じなければならないことを示しています。クロックを停止しないで閉じると、ユーザがデータベースを終了するまでRefreshClockは動き続けます。

プロセスに関する詳細は、第10章と第39章、第40章、第41章を参照してください。

Activated

Activated

Activated関数は、フォームまたはオブジェクトメソッドを含むウインドウが最前面のプロセスの最前面のウインドウになると、フォームメソッドとオブジェクトメソッドが“ True ”を返します。オブジェクトメソッドに対してはActivatedフェーズのみ実行されます。

フォームやオブジェクトメソッドのActivatedフェーズに**TRACE**コマンドまたは**ALERT**コマンドを入れると、無限ループになりますのでこれは避けてください。オブジェクトメソッドのActivatedフェーズが呼び出されるようするには、「変更時のみ実行」チェックボックスのチェックマークを外してください。

▼ 次の例は、追加、変更、削除、印刷などの基本的レコード操作用ボタンを含むフローティングウインドウの作成方法を示しています。このフローティングウインドウはファイル名も表示します。ウインドウの順序が変わるごとに**CALL PROCESS**コマンドを用いてフローティングウインドウ内のファイル名が更新されます。各ウインドウのフォームメソッドでこのコードを繰り返すよりも、これを含むプロジェクトメソッドを呼び出す方法がやり方としては良いでしょう：

```
Case of
   \ (Deactivated)
      <>Action:=0                      ` ウインドウが前面でなくなった
      CALL PROCESS (<>Dashboard)       ` フローティングウインドウプロセスを呼ぶ
   \ (Activated)
      <>Action:=1                      ` ウインドウがアクティブになった
      <>TableShown:=->[顧客]           ` テーブルが表示される
      CALL PROCESS (<>Dashboard)       ` フローティングウインドウプロセスを呼ぶ
   \ (Outside call)
      Case of
      ...
         \ (<>Action=2)                ` 印刷
            PRINT SELECTION ([顧客])
      End case
End case
```

フローティングウインドウのフォームメソッドは、次のようになります：

```
Case of
   \ (Before)
   ` ...
   \ (Outside call)
      Case of
         \ (<>Action=0)                    ` ウインドウが前面でなくなった
            vファイル名:="ファイルが選択されていません"
            DISABLE BUTTON (b印刷)   `ここで他のボタンを処理する
         \ (<>Action=1)                    ` ウインドウがディアクティブになった
            vファイル名:=Table name (<>TableShown)
            If (Records in selection (<>TableShown->)>0)
               ENABLE BUTTON (b印刷)
            Else
               DISABLE BUTTON (b印刷)
            End if
               `ここで他のボタンを処理する
      Else
         ` ...
      End case
   \ (During)
      Case of
         \ (b印刷=1)
            <>Action:=2
            CALL PROCESS (Frontmost process(*))
      End case
End case
```

インタープロセス変数“ <>Action ”と“ <>TableShown ”および**CALL PROCESS**コマンドを用いてウインドウ間のメッセージシステムを作成しているところに注目してください。

Deactivated

Deactivated

Deactivated関数は、フォームまたはオブジェクトメソッドを含む最前面プロセスの最前面ウインドウが背後に配置されると、フォームメソッドとオブジェクトメソッドに“ True ”を返します。

Activatedコマンドの例を参照してください。

この章では、「ルーチン」エディタの「Form Pages」テーマ内にあるフォームページコマンドについて説明します。この章のコマンドは、フォームページを操作します。**FIRST PAGE**、**LAST PAGE**、**NEXT PAGE**、**PREVIOUS PAGE**の各コマンドと同じ処理を実行する自動動作ボタンが4th Dimension内部に用意されています。適宜、これらのコマンドの代わりに自動動作ボタンを使用することもできます。

Current form page	**GOTO PAGE**	**NEXT PAGE**
FIRST PAGE	**LAST PAGE**	**PREVIOUS PAGE**

ページコマンドは、入力フォームやダイアログに表示されるフォームでのみ使用することができます。出力フォームは先頭のページしか使用できません。フォームは常に、最低でも先頭の1ページが存在します。

フォームのページ数に関係なく、1つのフォームには1つのフォームメソッドしか存在しません。

FIRST PAGE

FIRST PAGE

FIRST PAGEコマンドは、現在表示されているフォームページを先頭のフォームページに変更します。フォームが表示されていない場合や最初のフォームページが表示されている場合には、**FIRST PAGE**コマンドは何も行いません。

▼ 次の例は、メニューコマンドから呼び出される1行のメソッドです。
これは、先頭のフォームページを表示します：

```
FIRST PAGE
```

LAST PAGE

LAST PAGE

LAST PAGEコマンドは、現在表示されているフォームページを最終のフォームページに変更します。フォームが表示されていない場合や最終のフォームページが表示されている場合には、**LAST PAGE**コマンドは何も行いません。

▼ 次の例は、メニューコマンドから呼び出される1行のメソッドです。これは、最終のフォームページを表示します：

```
LAST PAGE
```

NEXT PAGE

NEXT PAGE

NEXT PAGEコマンドは、現在表示されているフォームページを次のフォームページに変更します。フォームが表示されていない場合や最終のフォームページが表示されている場合には、**NEXT PAGE**コマンドは何も行いません。

▼ 次の例は、メニューコマンドから呼び出される1行のメソッドです。現在表示されているフォームページの次のフォームページを表示します：

```
NEXT PAGE
```

PREVIOUS PAGE

PREVIOUS PAGE

PREVIOUS PAGEコマンドは、現在表示されているフォームページを前のフォームページに変更します。フォームが表示されていない場合や先頭のフォームページが表示されている場合には、**PREVIOUS PAGE**コマンドは何も行いません。

▼ 次の例は、メニューコマンドから呼び出される1行のメソッドです。現在表示されているフォームページの前のフォームページを表示します：

```
PREVIOUS PAGE
```

GOTO PAGE

GOTO PAGE (ページ番号)

引数	タイプ	説明
ページ番号	数値	表示するフォームページ

GOTO PAGEコマンドは、現在表示されているフォームページを＜ページ番号＞で指定したフォームページに変更します。

フォームが表示されていない場合には、**GOTO PAGE**コマンドは何も行いません。＜ページ番号＞が実際のフォームページの数よりも大きければ、最終ページを表示します。＜ページ番号＞に1以下の数が指定されると、先頭のフォームページを表示します。指定したフォームページを呼び出すために、ボタンのオブジェクトメソッドに**GOTO PAGE**コマンドを使用することもできます。

▼ 次の例はボタンのオブジェクトメソッドです。これは指定したフォームページ 3を表示します：

```
GOTO PAGE (3)
```

Current form page

Current form page → 数値

引数	タイプ		説明
このコマンドには、引数はありません。			
関数が返す値	数値	←	フォームイベント番号

Current form page関数は、現在表示されているフォームページの番号を返します。

▼ フォーム内において、メニューバーから任意のメニューコマンドを選択したり、そのフォームが別プロセスの呼び出しを受信すると、現在表示されているフォームページによって異なる動作を実行することができます。これ例を次に示します：

```
` 任意フォームのフォームオブジェクト
Case of
   \ (Form event=On Load)
      ` ...
   \ (Form event=On Unload)
      ` ...
   \ (Form event=On Menu selected)
      $vlMenuNumber:=Menu Selected -> 16
      $vlItemNumber:=Menu Selected & 0xFFFF
      Case of
         \ ($vlMenuNumber=...)
            Case of
               \ ($vlItemNumber=...)
                  \ (Current form page=1)
                        `ページ1に合った処理を行う
                  \ (Current form page=2)
                        `ページ2に合った処理を行う
                        ` ...
               \ ($vlItemNumber=...)
                        ` ...
            End case
         \ ($vlMenuNumber=...)
            ` ...
      End case
   \ (Form event=On Outside call)
      Case of
         \ (Current form page=1)
            `ページ1に合った処理を行う
         \ (Current form page=2)
            ` ページ2に合った処理を行う
   End case
   ` ...
End case
```

参照：FIRST PAGE, GOTO PAGE, LAST PAGE, NEXT PAGE, PREVIOUS PAGE

グラフコマンド　26

この章では、「ルーチン」エディタの「Graphs」テーマ内にあるグラフコマンドについて説明します。バージョン6からグラフ機能は4th Dimensionの中にあらかじめ組み込まれている4D Chartプラグインでサポートされるようになりました。バージョン3の4th Dimensionで使用されていたグラフコマンドは4D Chartで透過的に転送されます。

グラフは、2種類の方法で作成します。レコードのデータからグラフ化（**GRAPH TABLE**コマンド）したり、サブフィールドまたは配列の情報からグラフ化（**GRAPH**コマンド）します。**GRAPH TABLE**コマンドは、グラフを作成するためにレコード上のフィールドのデータを使用します。このコマンドは、自分自身のウインドウにグラフを表示します。**GRAPH**コマンドは、配列またはサブフィールドの情報を使用して、フォーム上のグラフエリアにグラフを作成します。

GRAPH　　**GRAPH SETTINGS**　　**GRAPH TABLE**

GRAPH

GRAPH (グラフエリア ; グラフ番号 ; xラベル ; y要素1 { ;...;y要素8})

引数	タイプ		説明
グラフエリア	変数	→	フォーム上のグラフエリア
グラフ番号	数値	→	グラフタイプ番号
xラベル	配列またはサブフィールド	→	X軸のラベル
y要素	配列またはサブフィールド	→	グラフへのデータ（最大8個）

GRAPHコマンドは、フォームのグラフエリアにグラフを作成します。データは、配列またはサブフィールドから取り出されます。

引数＜グラフエリア＞は、グラフを表示するグラフエリアの名前です。グラフエリアは、「フォーム」エディタでグラフオブジェクトタイプを使用して作成します。グラフ名は、その変数に入力された名前です。グラフエリアの作成に関する詳細は、『4th Dimension / 4D First デザインリファレンス』を参照してください。

引数＜グラフ番号 ＞は、作成されるグラフタイプを定義します。これは1から8までの数値でなければなりません。グラフタイプは次ページの例題1を参照してください。グラフタイプを変更する場合は、グラフを作成した後で＜グラフ番号＞を変更したり、**GRAPH**コマンドをもう一度実行します。

引数＜xラベル＞は、X軸(グラフの一番下)に使用するラベルを定義します。このデータは、文字列、日付、時間、数値タイプのいずれでも構いません。＜xラベル＞は、＜y要素＞のサブレコードや配列要素と同じ数のサブレコードまたは配列要素がなければなりません。

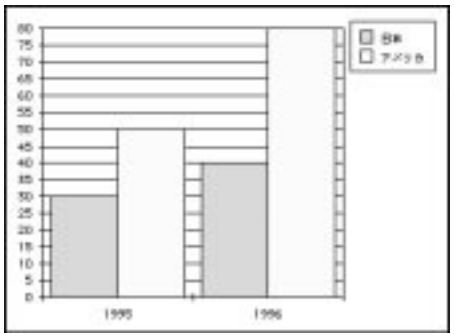
引数＜y要素＞で指定するデータは、グラフにするデータです。このデータは数値でなければなりません。最大8つのデータセットをグラフ化することができます。それぞれをセミコロン(;)で区切って指定します。円グラフは、最初の＜y要素＞のみをグラフ化します。

▼ 次の例は、変数を使用してグラフを作成します。このステートメントは、フォームメソッドやオブジェクトメソッドに使用します。データは定数ですので、あまり現実的ではありません：

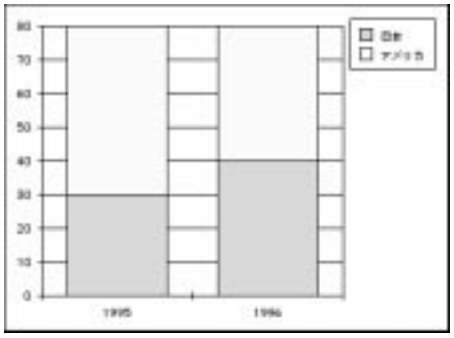
```
ARRAY STRING (4 ; X ; 2)                    ` X軸の配列作成
X{1}:="1995"                                ` X Label #1
X{2}:="1996"                                ` X Label #2
ARRAY REAL (A ;2)                           ` Y軸の配列作成
A{1}:=30                                    ` データを挿入
A{2}:=40
ARRAY REAL (B ; 2)                          ` Y軸の配列作成
B{1}:=50                                    ` データを挿入
B{2}:=80
GRAPH (vグラフ ; vタイプ ; X ; A ; B)       ` グラフ作成
` グラフ範囲をセット
GRAPH SETTINGS (vグラフ ; 0 ; 0 ; 0 ; 0 ; False ; False ; True ;"日本" ; "アメリカ")
```

下図は、実行結果のグラフを表しています。

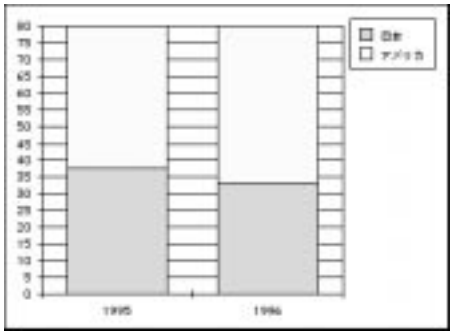
■「vタイプ」を1にすると、「棒グラフ」で表示します：



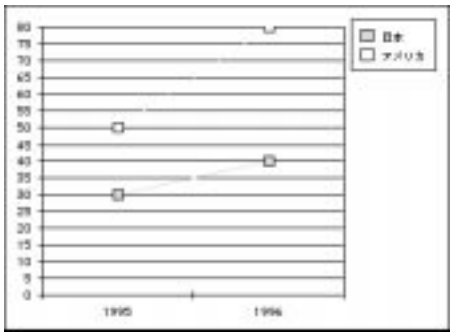
■「vタイプ」を2にすると、「棒・比率グラフ」で表示します：



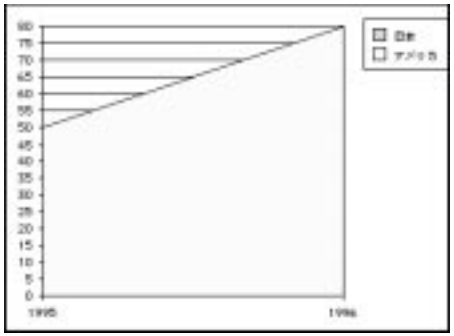
■「vタイプ」を3にすると、「棒・累積グラフ」で表示します：



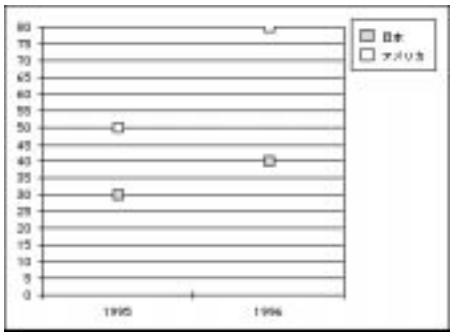
■「vタイプ」を4にすると、「線グラフ」で表示します：



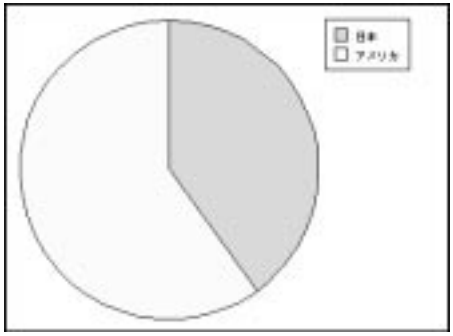
■「vタイプ」を5にすると、「面グラフ」で表示します：



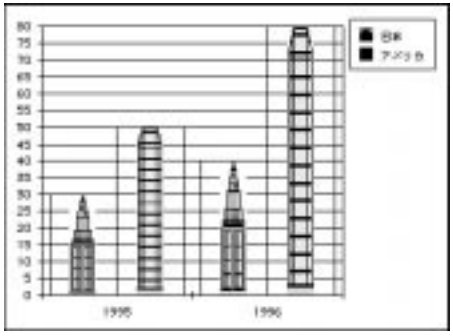
■「vタイプ」を6にすると、「点グラフ」で表示します：



■「vタイプ」を7にすると、「円グラフ」で表示します：



■「vタイプ」を8にすると、「ピクチャグラフ」で表示します：



▼ 次の例は、サブテーブルに入っているセールスマンの売上金額をグラフ化します。サブテーブルには名前、去年の合計売上金額、今年の合計売上金額の3つのサブフィールドがあります。このグラフは最後の2年間のセールスマンのそれぞれの売上を示します：

GRAPH (売上グラフ ; 1 ; セールス'名前 ; セールス'昨年度合計 ; セールス'本年度合計)

GRAPH SETTINGS

GRAPH SETTINGS (g ; x最小値 ; x最大値 ; y最小値 ; y最大値 ; xプロップ ; xグリッド ; yグリッド ; {タイトル1 ; ... ; タイトル8})

引数	タイプ		説明
g	変数	→	グラフエリアの名前
x最小値	数値／日付／時間	→	比例グラフのｘ軸の最小値 (線グラフまたは点グラフのプロットのみ)
x最大値	数値／日付／時間	→	比例グラフのｘ軸の最大値 (線グラフまたは点グラフのプロットのみ)
y最小値	数値	→	最小のY軸の値
y最大値	数値	→	最大のY軸の値
xプロップ	ブール	→	比例X軸に対して“ True(真) ” ノーマルX軸に対して“ False(偽) ” (線グラフまたは点グラフのプロットのみ)
xグリッド	ブール	→	X軸グリッドに対して“ True(真) ” 非X軸グリッドに対して“ False(偽) ” (xプロップが“ True(真) ”の場合のみ)
yグリッド	ブール	→	Y軸グリッドに対して“ True(真) ” 非Y軸グリッドに対して“ False(偽) ”
タイトル	文字列	→	グラフ凡例のタイトル

GRAPH SETTINGSコマンドは、引数<g>のグラフ設定値を変更します。このコマンドは、**GRAPH**コマンドの例で既に紹介しました。このコマンドによるグラフ設定値は、円グラフ(タイプ7)に対しては無効です。

引数<x最小値>、<x最大値>、< y最小値>、< y最大値>にグラフのそれぞれの軸の最大値と最小値を指定します。これらの引数の値が空(ヌル)の場合(0、?00:00:00?、!00.00.00!など、データ型によって異なります)、デフォルトのグラフ値を使用します。

引数<xプロップ>は、線グラフ(タイプ4)と点グラフ(タイプ6)に対する比例プロットを変更します。“ True ”であれば、点の値に従ってX軸上の各点をプロットします。ただし、値が数値、時間、日付の場合に限ります。

<xグリッド>と<yグリッド>は、グリッドラインを表示したり、非表示にします。X軸のグリッドは、プロットが比例する点グラフまたは線グラフの場合にのみ表示します。

引数<タイトル>は、指定した文字列を凡例のラベルとして表示します。

前ページの**GRAPH**コマンドの例を参照してください。

参照：GRAPH、GRAPH TABLE

GRAPH TABLE

GRAPH TABLE ({テーブル})

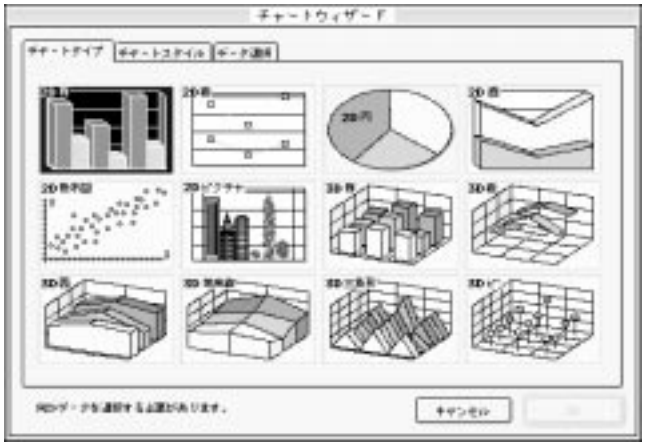
引数	タイプ		説明
テーブル	テーブル	→	グラフ化するテーブル

GRAPH TABLE({テーブル} ; グラフ番号 ; xフィールド ; yフィールド1{ ;...; yフィールド8})

引数	タイプ		説明
テーブル	テーブル	→	グラフ化するテーブル
グラフ番号	数値	→	グラフタイプ番号
xフィールド	フィールド	→	X軸に対するラベル
yフィールド	フィールド	→	グラフ化するフィールド(最大8個)

GRAPH TABLEコマンドには、2つのシンタックスがあります。第1のシンタックスは、「4D Chartウィザード」ウインドウを表示し、ユーザがグラフにするフィールドを選択します。第2のシンタックスは、グラフ化するフィールドを指定し、「4D Chartウィザード」ウインドウを表示しません。**GRAPH TABLE**コマンドは、テーブルのフィールドのデータをグラフにします。また、カレントセレクションのデータだけをグラフにします。

第1のシンタックスは、「ユーザ」モードの「レポート」メニューから「チャート...」を選択した場合と同じです。次の図は「4D Chartウィザード」ウインドウです。ユーザは、このウインドウでグラフを定義します。



GRAPH TABLEコマンドの第2のシンタックスは、＜テーブル＞のフィールドを指定してグラフにします。

引数＜グラフ番号＞は、1から8の数値でグラフのタイプを定義します。グラフタイプに関する詳細は、**GRAPH**コマンドの例題を参照してください。

引数＜xフィールド＞は、X軸(グラフの底部)につけるラベルを定義します。フィールドは文字、整数、倍長整数、実数、日付のいずれでも構いません。

引数＜yフィールド＞は、グラフのデータです。フィールドは文字、整数、倍長整数、実数のいずれかの数値のデータです。＜yフィールド＞は、最大8個まで指定することができ、各引数はセミコロン(;)で区切ります。

いずれのタイプでも、**GRAPH TABLE**コマンドは、新しく作成されたグラフで作業できるように任意のチャートウインドウを開きます。チャートウインドウの使用方法に関する詳細は、『4th Dimension / 4D First ユーザリファレンス』を参照してください。

「クイックレポート」エディタの「出力先」メニューコマンドを選択して、フィールドデータからグラフを作成することができます。グラフに関する詳細は、『4th Dimension / 4D First ユーザリファレンス』を参照してください。

▼ 次の例は、**GRAPH TABLE**コマンドの第1のシンタックスの説明です。これは、「4D Chartウィザード」ウインドウを表示し、グラフ化するフィールドをユーザが選択します。この例ではユーザが検索とソートを行い、グラフ上のレコードの順番を決定します：

```
QUERY ([従業員])                          ` 従業員テーブルを検索
If (OK=1)
  ORDER BY ([従業員])                     ` 従業員テーブルをソート
  If (OK=1)
    GRAPH TABLE ([従業員])                ` 従業員テーブルをグラフにする
  End if
End if
```

▼ 次の例は、**GRAPH TABLE**コマンドの第2のシンタックスの説明です。最初に[従業員]テーブルから課長のレコードを検索して並び替え、その給与をグラフ化します：

```
QUERY ([従業員] ; [従業員]役職名="課長")    ` 従業員テーブルから課長を検索
ORDER BY ([従業員] ; [従業員]給与 ; >)      ` 給与でソート
GRAPH TABLE ([従業員] ; 1 ; [従業員]名字 ; [従業員]給与) ` 課長の給与をグラフにする
```

階層リストコマンド　27

この章では、「ルーチン」エディタの「Clipboard」テーマ内にあるクリップボードコマンドについて説明します：

Load list
SAVE LIST
New list
Copy list
CLEAR LIST
Count list items
Is a list
REDRAW LIST
SET LIST PROPERTIES
GET LIST PROPERTIES
SORT LIST
APPEND TO LIST
INSERT LIST ITEM
SET LIST ITEM PROPERTIES
GET LIST ITEM PROPERTIES
List item position
List item parent
DELETE LIST ITEM
GET LIST ITEM
SET LIST ITEM
Selected list item
SELECT LIST ITEM
SELECT LIST ITEM BY REFERENCE

Load list

Load list (リスト名) → リスト参照番号

引数	タイプ		説明
リスト名	文字列	→	「デザイン」モードの「リスト」エディタ内で作成されたリストの名前
関数の返す値	リスト参照番号	←	新しく作成されたリストの参照番号

Load list関数は、内容をリストからコピーし、引数＜リスト名＞で渡した名前を付けた階層リストを新たに作成します。これは、新たに作成されたリストにリストの参照番号を返します。

＜リスト名＞で指定されたリストが存在しない場合、リストは作成されず、**Load list**関数はゼロを返します。

新規リストは「デザイン」モードで定義されたリストのコピーであることに注意してください。従って、新規リストに変更を加えても、「デザイン」モードで定義されたリストには作用しません。逆に、「デザイン」モードで定義されたリストに変更を加えても、作成したリストに作用しません。

新たに作成したリストを変更し、その変更を永久に保存する場合は、**SAVE LIST**コマンドを呼び出します。

新たに作成したリストを使いおわったら、**CLEAR LIST**コマンドを呼び出してそれを廃棄するのを忘れないでください。そうでなければ、作業セッションが終了するまで、またはそれを作成したプロセスが終了するかアボートされるまで、そのリストはメモリ内に留まります。

Tips：「フォームエディタオブジェクトプロパティ」ウィンドウの「選択リスト」プロパティを使用して、リストをフォームオブジェクト(階層リスト、タブコントロール、階層ポップアップメニュー)に関連付ける場合、オブジェクトメソッドからLoad list関数またはCLEAR LISTコマンドを呼び出す必要はありません。4th Dimensionはリストを自動的にロードして消去します。

例：

国際市場に対応するデータベースを作成し、そのデータベースの使用中に異なる言語に切り替える必要があるとします。フォームで「hlList」という名前の、標準オプションのリストを示す階層化リストを指定します。「デザイン」モードでは、英語版の"Std Options US"、フランス語版の"Std Options FR"、日本語版の"Std Options JP"など、さまざまなリストを準備しました。これに加えて、<>gsCurrentLanguageという名前のインタープロセス変数を持ち、これに、英語版には"US"、フランス語版には"FR"、日本語版には"JP"というように2文字の言語コードを格納します。現在選択されている言語を使用してリストがロードされることを保証するために、以下のように記述します：

```
`「hlList」階層リストのプロジェクトメソッド
Case of
   \ (Form event = On Load)
      C_LONGINT (hlList)
      hlList:=Load list("Std Options"+<>gsCurrentLanguage)
   \ (Form event = On Unload)
      CLEAR LIST(hlList ; *)
End case
```

参照：CLEAR LIST、SAVE LIST

SAVE LIST

SAVE LIST (リスト ; **リスト名**) → **リスト参照番号**

引数	**タイプ**		**説明**
リスト	リスト参照番号	→	リスト参照番号
リスト名	文字列	→	「デザイン」モードの「リスト」エディタに表示されるリストの名前

SAVE LISTコマンドは、「デザイン」モードのリストエディタの中にあって、引数＜リスト＞によって渡された参照番号を持つリストを＜リスト名＞に渡される名前で保存します。

すでにその名前のリストが存在する場合は、その内容が置き換えられます。

参照：Load list

New list

New list → リスト参照番号

引数	タイプ		説明
このコマンドには、引数はありません。			
関数の返す値	リスト参照番号	←	リスト参照番号

New list関数は、新しい空の階層リストをメモリに作成し、その一意のリスト参照番号を返します。

警告：階層リストはメモリに保持されます。階層リストを使い終わったら、CLEAR LIST コマンドを使用してそれを廃棄し、メモリを解放することが重要です。

階層リストを作成するコマンドは、この他にもあります：

■ **Copy list**関数は、既存のリストからリストを複製します。

■ **Load list**関数は、「デザイン」モードのリストエディタで作成された(手動またはプログラムによって)選択リストをロードすることによりリストを作成します。

■ **BLOB to list**関数は、リストが前回保存されたBLOBのコンテンツから作成されます。

New list関数を使用して階層リストを作成すれば、次のことが行えます：

■ **APPEND LIST ITEM**コマンドまたは**INSERT LIST ITEM**コマンドを使用して、項目をそのリストに追加できる

■ **DELETE LIST ITEM**コマンドを使用して、そのリストから項目を削除できる

▼ **APPEND TO LIST**コマンドの例を参照してください。

参照：APPEND TO LIST、BLOB to list、CLEAR LIST、Copy list、DELETE LIST ITEM、INSERT LIST ITEM、Load list.Copy list

Copy list

Copy list (リスト)→ リスト参照番号

引数	タイプ		説明
リスト	リスト参照番号	→	コピーされるリスト参照番号
関数の返す値	リスト参照番号	←	複製したリストのリスト参照番号

Copy list関数は、引数＜リスト＞で渡した参照番号を持つリストを複製し、新しいリストのリスト参照番号を返します。

新しいリストを使い終わったら、**CLEAR LIST**コマンドを呼び出して削除します。

参照：CLEAR LIST、Load list、New list

CLEAR LIST

CLEAR LIST (リスト {; *})

引数	タイプ		説明
リスト	リスト参照番号	→	リスト参照番号
*		→	指定した場合、メモリからサブリストをクリア 省略した場合、サブリストはクリアしない

CLEAR LISTコマンドは、引数＜リスト＞によって渡されたリスト参照番号を持つ階層リストを廃棄します。

通常、オプション引数の＜ * ＞を付けて、リストの項目またはサブ項目にサブリストが添付されている場合、それも一緒に廃棄します。

フォームオブジェクトに添付されているリストを「オブジェクトプロパティ」ウィンドウによって消去する必要はありません。4Dがリストを自動的にロードして消去します。他方、BLOBからプログラムによってリストのロード、コピー、抽出、または作成を行った場合は、リストを使い終わるたびに**CLEAR LIST**コマンドを呼び出します。

カレントフォームに表示されているもう1つのリストの(あらゆるレベルの)項目に添付されているサブリストを消去するには、次のように行います：

1. 親項目でGET LIST ITEMコマンドを呼び出して、サブリストのリスト参照を取得する。

2. 親項目でSET LIST ITEMコマンドを呼び出して、サブリストを消去する前にサブリストをリストから切り離す。

3. CLEAR LISTコマンドを呼び出して、GET LIST ITEMコマンドで取得した参照番号を持つサブリストを消去する。

4. REDRAW LISTコマンドを呼び出して、フォームに表示されているリストについてその項目とサブリストを再計算する。

例：

1. アプリケーションのある時点で、もはや必要のない(例えば、ウィンドウが終了し、フォームがアンロードされる場合)オブジェクトとデータをすべて消去するクリーンアップルーチンの中で、フォームにおけるユーザの動作によっては、すでに消去した階層リストを最後に消去することがあるでしょう。必要に応じて、**Is a list**関数を使用してリストを消去します：

```
`「クリーンアップ」ルーチン
If (Is a list(hlList))
   CLEAR LIST(hlList ; *)
End if
```

2. **Load list**関数の例を参照してください。

3. **BLOB to list**関数の例を参照してください。

参照：BLOB to list、Load list、New list

Count list items

Count list items (リスト)→ 倍長整数

引数	タイプ		説明
リスト	リスト参照番号	→	リスト参照番号
関数の返す値	倍長整数	←	拡げられたリストの中にある項目数

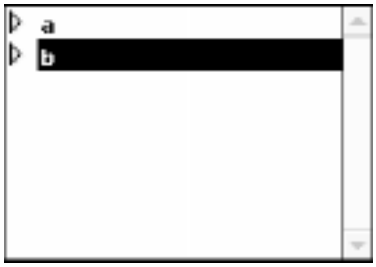
Count list items関数は、引数＜リスト＞によって渡された参照番号を持つリストにおいて、現在"表示されている"項目の数を返します。

Count list items関数は、リストにある項目の総数を返すものではありません。これは、リストおよびサブリストの現在の展開／縮小の状態に応じて、表示されている項目の数を返します。

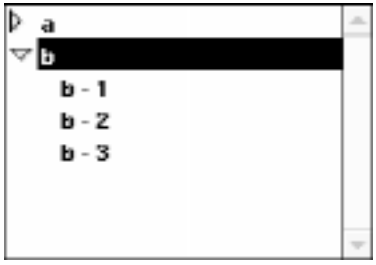
このコマンドは、フォームに表示されているリストに適用します。

例：

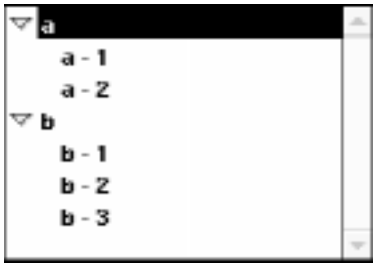
次は、「ユーザ」モードで表示された「hList」という名前のリストです。



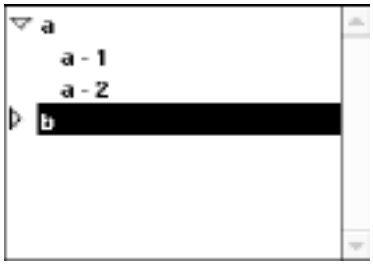
$vlNbItems:=Count list items(hList) `この時点で、「$vlNbItems」変数は2を取得する



$vlNbItems:=Count list items(hList) `この時点で、「$vlNbItems」変数は5を取得する



$vlNbItems:=Count list items(hList) `この時点で、「$vlNbItems」変数は7を取得する



$vlNbItems:=Count list items(hList) `この時点で、「$vlNbItems」変数は4を取得する

参照：List item position、Selected list item

Is a list

Is a list (リスト)→ ブール

引数	タイプ		説明
リスト	リスト参照番号	→	検査されるリスト参照番号の値
関数の返す値	ブール	←	リストが階層化リストの場合はTRUE リストが階層化リストでない場合はFALSE

Is a list関数は、引数＜リスト＞によって渡された値が階層化リストに対して有効な参照の場合TRUEを返します。その以外の場合は、FALSEを返します。

▼ **CLEAR LIST**コマンドの例を参照してください。

▼ **DRAG AND DROP PROPERTIES**コマンドの例を参照してください。

参照：DRAG AND DROP PROPERTIES

REDRAW LIST

REDRAW LIST (リスト)

引数	タイプ		説明
リスト	変数	→	リスト参照番号

REDRAW LISTコマンドは、引数<リスト>によって渡された参照番号を持つリストの項目やサブリストの位置を再計算します。

フォームにおいて、リストの1つ以上の要素またはそのサブリストを変更した場合、最低1回はこのコマンドを呼び出す必要があります。

警告：式や変数ではなく、リストの実際の変数インスタンスを渡してください。例えば、フォームに「hList」という名前のリストがある場合、

```
`変更が行われた後でリストを再計算する
REDRAW LIST (hList) ` 良い例
   ` ...

$vlList:=hList
   ` ...
`変更が行われた後でリストを再計算する
REDRAW LIST ($vlList) ` 悪い例
` ...
```

SET LIST PROPERTIES

SET LIST PROPERTIES (リスト ; 表示形式 ; アイコン {; 線の高さ)

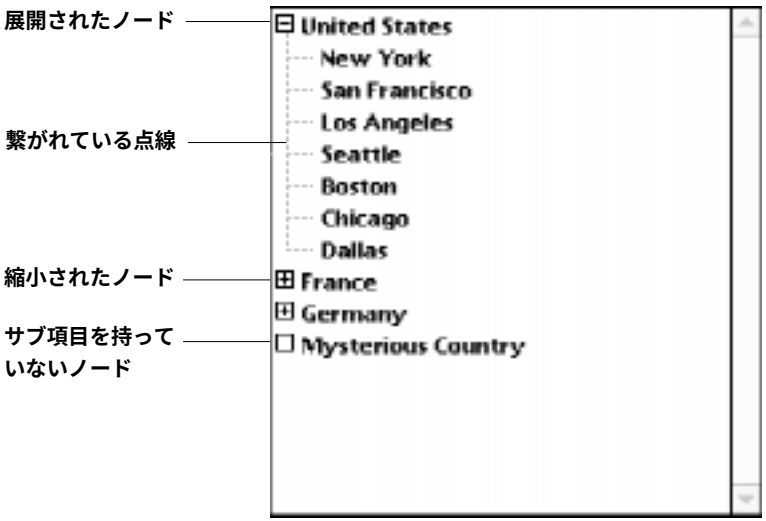
引数	タイプ		説明
リスト	リスト参照番号	→	リスト参照番号
表示形式	数値	→	リストの表示形式 1=Macintoshの階層リスト 2=Windowsの階層リスト
アイコン	数値	→	'cicn' MacOSベースのリソースIDまたは デフォルトプラットフォームのノード アイコンの場合は0
線の高さ	数値	→	線の高さの最小値（ピクセル単位）

SET LIST PROPERTIESコマンドは、引数＜リスト＞によって渡されたリスト参照を持つ階層リストの表示様式を設定します。

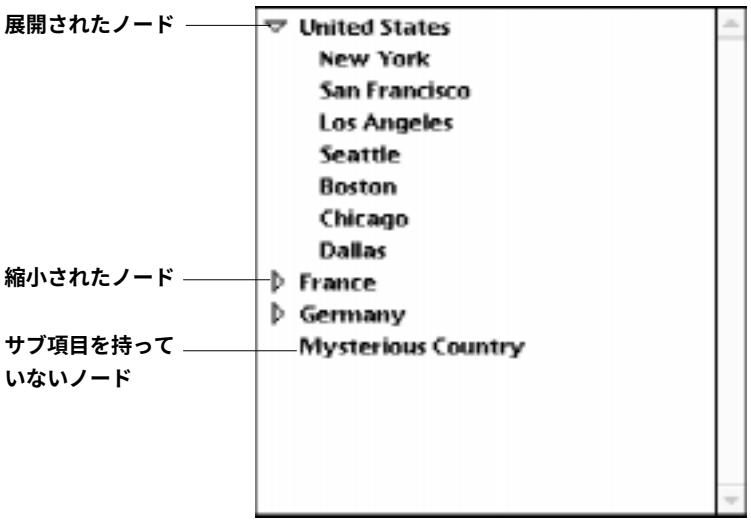
引数＜表示形式＞は、4th Dimensionが提供する以下の定義済定数のいずれかです。

定数	タイプ	値
ala Macintosh	倍長整数	1
ala Windows	倍長整数	2

Windows表示では、リストはノードとブランチの間を点線で繋ぎます。1つのアイコンは縮小されたノード、別のアイコンは展開されたノード、もう1つのアイコンはサブ項目を持たないことを示します。以下は、Windows表示のデフォルトの階層リストです。



Macintosh表示では、リストには接続のための点線はありません。1つのアイコンは縮小されたノード、別のアイコンは展開されたノードを示します。サブ項目を持たないノードにはアイコンがありません。以下はMacintosh表示のデフォルトの階層リストです。



注：SET LIST PROPERTIESコマンドを呼び出さずに階層リストオブジェクトを表示すると、リストは「デザイン」モードのフォームエディタでそのオブジェクトについて選択した「プラットフォームインタフェース」プロパティに応じて、デフォルトのWindowsまたはMacintoshの様式で表示されます。

引数＜アイコン＞は、それぞれのノードを表示するアイコンを示します。アイコンに渡される値は縮小されたノードのための＜アイコン＞、＜アイコン＞+1 は展開されたノードのためのアイコン、＜アイコン＞+2 はサブ項目を持たないノード(Windows表示に設定されている場合)を設定します。

例えば、15000を渡した場合、カラーアイコン'cicn' ID=15000は縮小されたノード用に表示され、カラーアイコン'cicn' ID=15001は展開されたノード用に表示され、カラーアイコン'cicn' ID=15002はサブ項目のないノード用に表示されます。

従って、これらの'cicn'カラーアイコンリソースがデータベースのストラクチャファイルに存在することが重要です。カラーアイコンがないと、対応するノードはアイコンなしで表示されます(リストをアイコンなしで表示するには、この方法を利用します)。

警告：cicn'カラーアイコンリソースを作成する場合、15000以上のリソースIDを使用してください。15000未満のリソースIDは4th Dimensionのために予約されています。

デフォルトのMacintoshおよびWindowsノードのリソースIDは、4th Dimensionの以下の定義済定数によって表されます：

定数	タイプ	値
Macintosh node	倍長整数	860
Windows node	倍長整数	138

つまり、4th Dimensionは次の'cicn'リソースを提供します。

ID番号	説明
860	Macintosh表示の縮小されたノード
861	Macintosh表示の展開されたノード
138	Windows表示の縮小されたノード
139	Windows表示の展開されたノード
140	Windows表示のサブ項目のないノード

引数＜アイコン＞を渡さない場合、ノードは選択した表示様式のデフォルトのアイコンで表示されます。

カラーアイコンリソースにはさまざまなサイズがあります。例えば、16*16または32*32のカラーアイコンを作成することができます。

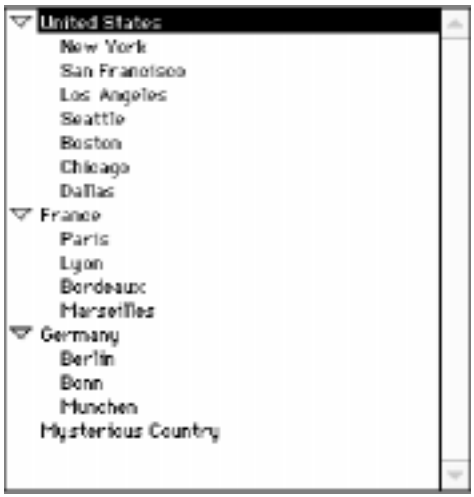
引数＜線の高さ＞を渡さない場合、階層リストの線の高さは、オブジェクトに使用されるフォントおよびフォントサイズによって決定されます。縦または横に大き過ぎるカラーアイコンを使用すると、端が切れて表示されるか、接続のための点線(Windows表示の場合)やその上下のテキストによって消されるか、またはその両方になります。

カラーアイコンのサイズ、フォント、フォントサイズを適切に選択するか、そうでなければ＜線の高さ＞で階層リストの線の高さの最小値を渡します。渡した値が、使用するフォントやフォントサイズから求められる線の高さより大きい場合、階層リストの線の高さは渡された値になります。

注：SET LIST PROPERTIESコマンドは、階層リストの中でのノードの表示方法に影響します。リストの中の個々の項目のアイコンをカスタマイズするには、SET LIST ITEM PROPERTIESコマンドを使用します。

例：

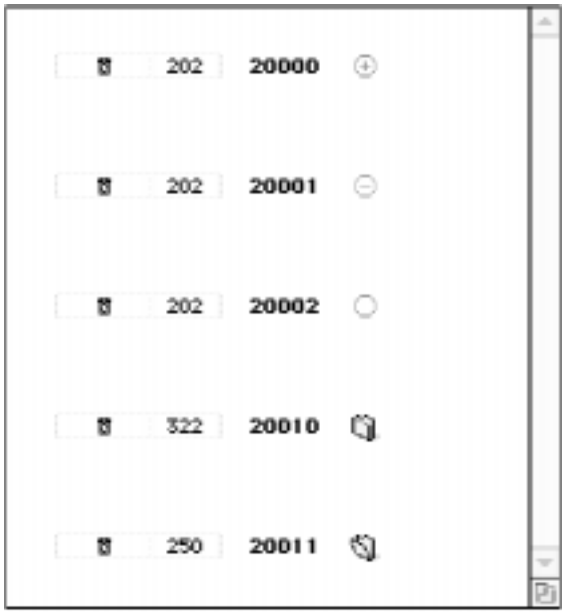
次の階層リストが、「デザイン」モードのリストエディタで定義されています。



フォームの中で、「hlCities」階層リストオブジェクトがそのリストをこのオブジェクトメソッドで再利用します。

```
Case of
   \ (Form event=On Load)
      hlCities:=Load list("Cities")
      SET LIST PROPERTIES(hlCities ; vlAppearance ; vlIcon)
   \ (Form event=On Unload)
      CLEAR LIST(hlCities ; *)
End case
```

加えて、データベースのストラクチャファイルが次の'cicn'カラーアイコンリソースを含むように編集されています。



1. 次の行により、

SET LIST PROPERTIES(hlCities ; ala Macintosh ; Macintosh node)

階層リストは、次のように表示されます。

2. 次の行により、

SET LIST PROPERTIES(hlCities ; ala Windows ; Windows node)

階層リストは、次のように表示されます。

3. 次の行により、

SET LIST PROPERTIES(hlCities ; ala Windows ; 20000)

階層リストは、次のように表示されます。

4. 次の行により、

SET LIST PROPERTIES(hlCities ; ala Macintosh ; 20000)

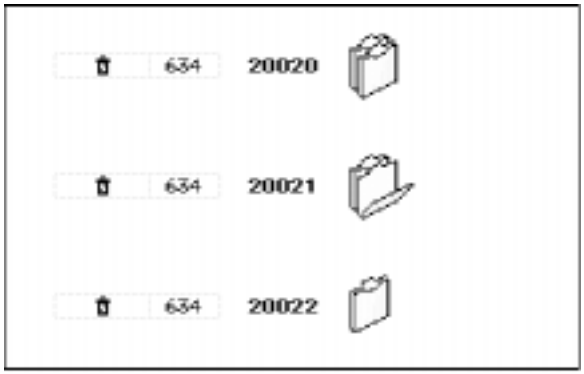
階層リストは、次のように表示されます。

5. 次の行により、

SET LIST PROPERTIES(hlCities ; ala Macintosh ; 20010)

階層リストは、次のように表示されます。

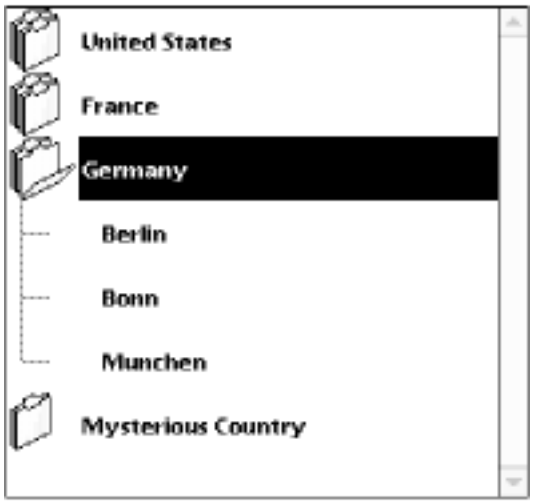
次に、以下に示す'cicn'カラーアイコンリソースがデータベースのストラクチャファイルに追加されます。



6. 次の行により、

SET LIST PROPERTIES(hlCities ; ala Windows ; 20020 ; 32)

階層リストは、次のように表示されます。



GET LIST PROPERTIES

GET LIST PROPERTIES (リスト ; 表示形式 ; アイコン {; 線の高さ)

引数	タイプ		説明
リスト	リスト参照番号	→	リスト参照番号
表示形式	数値	←	リストの表示形式 1=Macintoshの階層リスト 2=Windowsの階層リスト
アイコン	数値	←	'cicn' MacOSベースのリソースIDまたは デフォルトプラットフォームのノード アイコンの場合は0
線の高さ	数値	←	線の高さの最小値（ピクセル単位）

GET LIST PROPERTIESコマンドは、引数＜リスト＞によって渡された参照番号を持つリストについての情報を返します。

引数＜表示形式＞は、リストの表示形式を返します。

引数＜アイコン＞は、リストに表示されるノードアイコンのリソースIDを返します。

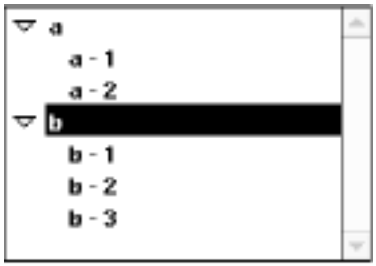
オプション引数＜線の高さ＞は、線の高さの最小値を返します。

これらのプロパティは、**SET LIST PROPERTIES**コマンドおよび/またはリストが「デザイン」モードのリストエディタで作成された場合、または**SAVE LIST**コマンドを使用して保存された場合は、リストエディタで設定することができます。

リストの表示様式、ノードアイコン、線の高さの最小値についての詳細は、**GET LIST PROPERTIES**コマンドを参照してください。

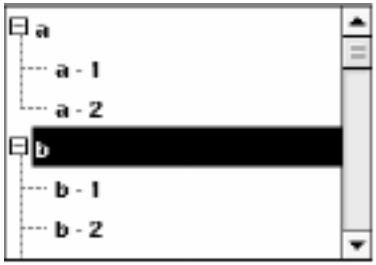
例：

次に示す「hList」という名前のリストが「ユーザ」モードにあるとします(Macintosh表示)。



```
`「bMacOrWin」ボタンのオブジェクトメソッド
GET LIST PROPERTIES(hList ; $vlAppearance ; $vlIcon ; $vlLH)
If ($vlAppearance=Ala Macintosh)
   $vlAppearance:=Ala Windows
   $vlIcon:=Windows node
   $vlLH:=20
Else
   $vlAppearance:=Ala Macintosh
   $vlIcon:=Macintosh node
   $vlLH:=0
End if
SET LIST PROPERTIES(hList ; $vlAppearance ; $vlIcon ; $vlLH)
REDRAW LIST(hList) `REDRAW LISTコマンドを忘れずに呼び出します。
                   `さもなければ、リストが更新されません。
```

上記のコードを実行すると、次のようなWindows表示形式で表示されます。



参照：SET LIST PROPERTIES

SORT LIST

SORT LIST (リスト {; > または <})

引数	タイプ		説明
リスト	リスト参照番号	→	リスト参照番号
>または<		→	ソート順位： >=昇順ソート <=降順ソート

SORT LISTコマンドは、引数＜リスト＞によって渡された参照番号を持つリストをソートします。

昇順にソートするにはオプション引数「 > 」を渡し、降順にソートするには「 < 」を渡します。ソート順パラメータを省略した場合、**SORT LIST**コマンドはデフォルトで昇順にソートします。

SORT LISTコマンドは、あらゆるレベルのリストをソートします。まずリストの項目をソートし、次にサブリストがあればそれらをソートし、サブリストの中の項目をソートするというようにリストのすべてのレベルに降りていきます。通常、**SORT LIST**コマンドをフォームに表示されているリストに適用するのはこのためです。サブリストのソートは、上位レベルを呼び出したときに順序が変更されるので、ほとんど意味がありません。

SORT LISTコマンドは、選択されたリスト項目またはリストやサブリストの現在の展開/縮小状態は変更しません。ただし、選択された項目がソートによって移動することはあります。

選択されたリスト項目がソートの前後で異なる位置を返すことがあります。

例：

リスト名を「hList」とし、「ユーザ」モードで表示します(Macintosh表示)。

次のコードを実行すると、

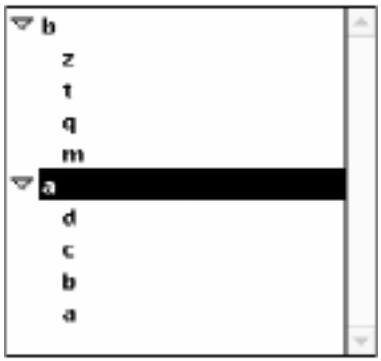
```
` リストとそのサブリストを昇順でソートする
SORT LIST(hList ; >)
REDRAW LIST(hList) `REDRAW LISTコマンドを忘れずに呼び出します。
                   `さもなければ更新されません。
```

リストは、次のようになります。

次のコードを実行すると、

```
`リストとそのサブリストを昇順でソートする
SORT LIST(hList ; <)
REDRAW LIST(hList) `REDRAW LISTコマンドを忘れずに呼び出します。
                   `さもなければ更新されません。
```

リストは、次のようになります。



参照：Selected list item

APPEND TO LIST

APPEND TO LIST (リスト ; 項目テキスト ; 項目参照番号 {; サブリスト{ ; 展開}})

引数	タイプ		説明
リスト	リスト参照番号	→	リスト参照番号
項目テキスト	文字列	→	新規リスト項目のテキスト(最大31文字)
項目参照番号	数値	→	新規リスト項目の一意の参照番号
サブリスト	リスト参照番号	→	新規リスト項目に添付するオプションのサブリスト
展開	ブール	→	そのサブリストが展開されるか縮小されるかを示す

APPEND TO LISTコマンドは、引数＜リスト＞によって渡された参照番号を持つ階層リストに新規項目を追加します。

項目のテキストは、引数＜項目テキスト＞によって渡します。31文字までの文字列またはテキスト式を渡すことができます。これより長い値を渡した場合は切り捨てられます。

項目の一意の参照番号は、引数＜項目参照番号＞によって渡します。この項目の参照番号を一意としましたが、実際にはどのような値でも渡すことができます。詳細については、下記の「項目参照番号」の節を参照してください。

項目にサブ項目を設定する場合は、引数＜サブリスト＞によってサブ階層リストへの有効なリスト参照を渡します。サブリストを展開または縮小するには、オプション引数＜展開＞によってTRUEまたはFALSEを渡します。

＜サブリスト＞によって渡されたリスト参照は既存のリストを参照しなければなりません。既存のリストは、空のリスト、1階層のリスト、サブリストを持つリストのいずれでも構いません。新規項目にサブリストを添付しない場合は、この引数を省略するか0を渡します。＜サブリスト＞を渡すと、デフォルトではサブリストは展開されません。

Tips：

■ リストに新規項目を挿入するには、**INSERT LIST ITEM**コマンドを使用します。既存の項目のテキストの変更、そのサブリストや展開／縮小状態の変更を行うには、**SET LIST ITEM**コマンドを使用します。

■ 新たに追加された項目の表示様式を変更するには、**SET LIST ITEM PROPERTIES**コマンドを使用します。

警告：フォームに現在表示されているリスト、またはフォームに現在表示されているリストを持つ項目に添付されているリストに項目を追加する場合は、REDRAW LISTコマンドを呼び出す必要があります。これにより、4Dは変更内容を反映してリストを再計算し、表示します。規則は簡単です。リストのどのレベルを操作しようと、REDRAW LISTコマンドをメインリスト、すなわちフォームでオブジェクトに参照されるリストに適用します。

項目参照番号：これは何に使用するのでしょうか?

階層リストの各項目には、倍長整数の参照番号があります。この値は専用の値で4th Dimensionだけがそれを持ちます。以下では、この参照番号をどのように使用するかについて説明します。

1. 各項目を一意に識別する必要はありません(初心者レベル)。

■ **第1の例：**タブコントロール(例えば、回転式卓上カードファイル)をプログラムによって作成します。タブコントロールは選択されたタブの数を返すので、おそらくそれ以上の情報は必要ありません。この場合、項目参照番号のことを考慮する必要もないので、引数<項目参照番号>に0を渡します。回転式卓上カードファイルのタブコントロールについては、「デザイン」モードでA、B、...、Zリストをあらかじめ定義することができます。しかし、レコードのない文字を削除するために(例えば、Qで始まるキーフィールドが1つもないなど)、プログラムによって作成したい場合もあるでしょう。

■ **第2の例：**データベースで作業しながら、徐々にキーワードのリストを作成します。各セッションの終わりに、必要に応じて**SAVE LIST**コマンドまたは**LIST TO BLOB**コマンドを使用してリストを保存し、各セッションの始めに**Load list**関数または**BLOB to list**関数を使用してそれを再度ロードします。このリストを「パレット」ウィンドウに表示し、項目をクリックすると、クリックされたキーワードが最前面にあるプロセスの現在入力可能なエリアに挿入されます。使用できるのはドラッグアンドドロップだけです。いずれにせよ、重要なのは本質的には選択した項目(クリックまたはドラッグした項目)を扱うということです。関数**Selected list item**(クリック)および**DRAG AND DROP PROPERTIES**コマンドにより、取得すべき項目の位置がわかります。この位置を使用して、**GET LIST ITEM**コマンドを使用する項目のテキストを取得することができます。これで終わりです。したがって、各項目を一意に識別する必要はなく、引数<項目参照番号>によって0を渡せばよいのです。

2. リスト項目を部分的に識別する必要があります。

項目参照番号を使用して、項目を操作する場合に必要な情報を格納します。後述の例がこのような場合にあたります。この例では、項目参照番号を使用してレコード番号を格納します。ただし、[部門]のレコードに対応する項目を[従業員]のレコードに対応する項目と区別する必要があります。これは、このコマンドの2番目の例で行われているので参照してください。

3. リスト項目を一意に識別する必要があります。

これは、リストの各レベルが何であっても、作成しているプログラムでは、それぞれを絶対に一意に識別する必要があるからです。これを行う最も簡単な方法は、プライベートカウンタを持つことです。ここで、**New list**関数を使用して、「hlList」というリストを作成するとしましょう。この時点では、カウンタvlhCounterを0に初期化します。**APPEND TO LIST**コマンドまたは**INSERT LIST ITEM**コマンドを呼び出すたびに、このカウンタ(vlhCounter:=vlhCounter+1)がインクリメントされ、そのカウンタを項目参照番号として渡します。秘訣は項目を削除してもカウンタをデクリメントせず、カウンタが増加しかしないことです。これにより、事実上、項目参照番号の一意性が保証されます。項目参照番号は倍長整数値なので、再初期化されたリストには、項目を20億回以上追加、挿入することができます(この限界に達するような場合は、リストではなくテーブルを使用する必要があるということになります)。

注：ビットワイズ演算子を使用する場合、項目参照番号も使用して倍長整数値にあたる情報を格納します。すなわち、整数値なら2つ、バイト値なら4つ、論理値なら32です。

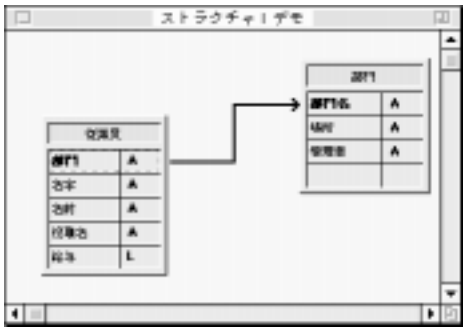
なぜ一意の参照番号が必要なのでしょうか?

ほとんどの場合、階層リストをユーザインターフェイスの目的で使用し、選択した項目(クリックまたはドラッグした項目)だけを扱う場合、項目参照番号を使用する必要はまったくありません。選択したリスト項目と**GET LIST ITEM**コマンドを使用する場合、現在選択されている項目を扱うのに必要なものはすべて揃っています。加えて、**INSERT LIST ITEM**コマンドや**DELETE LIST ITEM**コマンドのようなコマンドでは、選択されている項目に"相対的に"リストを操作することができます。

基本的に、参照番号を使った処理が必要になるのは、必ずしもリストで現在選択されている項目に限らず、プログラムによってリストの任意の項目に直接アクセスする場合です。

例：

次は、データベースストラクチャの一部です。



[部門]テーブルと[従業員]テーブルには、次のレコードが入っています。

部門

部門名	場所	管理者
営業	本社3階	太田恵男
技術開発	技術センター	山田幸二
経理	本社7階	井上洋介
活動休眠中	未定	未定

従業員

部門	名字	名前
営業	会田	一郎
経理	佐藤	祐子
技術開発	田中	豊
技術開発	名取	利雄
経理	水野	あやの
経理	高岡	由美
営業	佐藤	幸太郎

ここで、部門を表示するとともに、各部門についてその部門で働いている従業員が入ったサブリストを示す階層リストを表示します。

「hlList」のオブジェクトメソッドは、次のように記述されます。

```
` 「hlList」階層リストのオブジェクトメソッド
Case of
    \ (Form event=On Load)
        C_LONGINT(hlList ; $hSubList ; $vlDepartment ; $vlEmployee)
        `新たな空の階層リストを作成する
        hlList:=New list
        `[部門]テーブルのすべてのレコードを選択する
        ALL RECORDS([部門])
        ` 各部門に対してループする
        For ($vlDepartment ; 1 ; Records in selection([部門]))
            `この部門のすべての従業員を選択する
            RELATE MANY([部門]部門名)
            ` 従業員は何人？
            $vlNbEmployees:=Records in selection([従業員])
            `この部門には最低1人の従業員がいるかどうかを確認する
            If ($vlNbEmployees>0)
                `部門項目のサブリストを作成する
                $hSubList:=New list
                ` 各従業員に対してループする
                For ($vlEmployee ; 1 ; Records in selection([従業員]))
                    `従業員項目をサブリストに追加する
                    `[従業員]レコードのレコード番号が項目参照番号として渡されること
                    `に注意してください。
                    APPEND TO LIST($hSubList ; [従業員]名字+"   "+[従業員]名前 ;
                            Record number([従業員]))
                    `次の[従業員]レコードに進む
                    NEXT RECORD([従業員])
                End for
            Else
                `部門項目に従業員もサブリストも存在しない場合
                $hSubList:=0
            End if
            `部門項目をメインリストに追加する
            `[部門]レコードのレコード番号が項目参照番号として渡されることに注意して
            `ください。項目参照番号の31番目のビットが1になり、部門項目と従業員項目
            `が区別できるようになります。このビットを項目についての補足情報として
            `使用できる理由については、後述の注を参照してください。
            APPEND TO LIST(hlList ; [部門]部門名 ;0x80000000   |
                            Record number([部門]) ; $hSublist ; $hSubList   #   0)
            `部門項目を太字に設定して、リストの階層を強調する
            SET LIST ITEM PROPERTIES(hlList ; 0 ; False ; Bold ; 0)
```

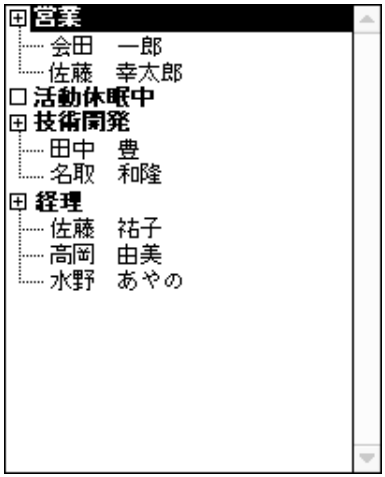
```
            `次の部門に進む
            NEXT RECORD([部門])
        End for
        `リスト全体を昇順にソートする
        SORT LIST(hlList ; >)
        ` リストをWindowsスタイルで表示し、線の高さの最小値を14ポイントに設定する
        SET LIST PROPERTIES(hlList ; ala Windows ; Windows node ; 14)
    \ (Form event=On Unload)
        `リストはもう必要ありません。忘れずに消去します！
        CLEAR LIST(hlList ; *)
    \ (Form event=On Double Clicked)
        `ダブルクリックが生じたら、選択した項目の位置を取得する
        $vlItemPos:=Selected list item(hlList)
        `念のために位置を確認する
        If ($vlItemPos  #  0)
            `リスト項目の情報を取得する
            GET LIST ITEM(hlList ; $vlItemPos ; $vlItemRef ; $vsItemText
                    ; $vlItemSubList ; $vbItemSubExpanded)
            `項目が部門項目かを確認する
            If ($vlItemRef  ??  31)
                `部門項目の場合は、部門項目をダブルクリックする
                ALERT("部門の項目がダブルクリックされました "+Char(34)
                    +$vsItemText+Char(34)+".")
            Else
                `そうでない場合は、従業員項目をダブルクリックする
                ` 親の項目参照番号を使用して、[部門]レコードを検索する
                GOTO RECORD([部門] ; List item parent(hlList ; $vlItemRef)  ?-  31)
                ` その従業員の勤務場所と直属上司を取得する
                ALERT("従業員項目がダブルクリックされました"+Char(34)+
                    $vsItemText+Char(34)+" 部門内の従業員名："+Char34)
                    +[部門]部門名+Char(34)+" 管理者名： "+Char(34)+
                    [部門]管理者+Char(34)+".")
            End if
        End if
End case
`注：4th Dimensionは、テーブルあたり最高1600万(正確には16,777,215)のレコードを
`格納することができます。この値は2^24-1です。レコード番号は、24ビットで表され
`ます。この例では、未使用の上位バイトの31番目のビットを使用して、従業員項目
`および部門項目を区別します。
```

この例では、部門項目と従業員項目を区別する必要がある理由は1つだけです。

1. レコード番号は項目参照番号に格納しますので、おそらく従業員項目と同じ項目参照番号を持つ部門項目で終わるでしょう。

2. **List parent item**関数を使用して選択した項目の親を取得します。対応するレコード番号が例えば10番で、同じ番号を持つ部門項目も存在し、項目参照番号を渡して項目を検索するためにリストを参照したときに、リストの親項目によってそれが最初に検索された場合、この関数は従業員項目の親ではなく、部門項目の親を返します。結論として、一意の項目参照番号が作成されましたが、これは項目参照番号が一意である必要があったからではなく、部門と従業員を区別する必要があったからです。

「ユーザ」モードまたは「カスタムメニュー」モードでは、リストは次のように見えます。



最後に：上記の例は、少ないレコードを扱う場合には、ユーザインターフェイスの作成に役立ちます。リストはメモリに置かれるので、何千という項目を持つ階層リストでユーザインターフェイスを作成すべきではありません。

INSERT LIST ITEM

INSERT LIST ITEM (リスト ; 前項目参照番号 | * ; 項目テキスト ; 項目参照番号 {サブリスト { ; 展開}})

引数	タイプ		説明
リスト	リスト参照番号	→	リスト参照番号
前項目参照番号 | *	数値 | *	→	項目の参照番号 *の場合、現在選択されているリスト項目
項目テキスト	文字列	→	新規リスト項目のテキスト(最大31文字)
項目参照番号	数値	→	新規リスト項目の一意の参照番号
サブリスト	リスト参照番号	→	新規リスト項目に添付するオプション のサブリスト
展開	ブール	→	そのサブリストが展開されるか縮小 されるかを示す

INSERT LIST ITEMコマンドは、引数＜リスト＞によって渡される参照番号を持つリストに新規項目を挿入します。

2番目の引数として＜ * ＞を渡す場合、項目は現在リストで選択されている項目の前に挿入されます。この場合、新たに挿入された項目もまた選択された項目になります。

そうでない場合、特定の項目の前に項目を挿入するには、その項目の項目参照番号を渡します。この場合、新たに挿入された項目が自動的に選択されることはありません。この項目参照番号を持つ項目が存在しない場合、このコマンドは何も行いません。

新規項目のテキストと項目参照番号は、引数＜項目テキスト＞と＜項目参照番号＞によって渡します。

例：

(1) 次のコードは、リスト「hList」の現在選択されている項目の直前に項目を挿入します(サブリストは添付されていません)。

```
vlUniqueRef:=vlUniqueRef+1
INSERT LIST ITEM(hList ; * ; "新規項目" ; vlUniqueRef)
REDRAW LIST(hList)
```

参照：APPEND TO LIST

SET LIST ITEM PROPERTIES

SET LIST ITEM PROPERTIES (リスト ; 項目参照番号 ; 入力可 ; スタイル ; アイコン)

引数	タイプ		説明
リスト	リスト参照番号	→	リスト参照番号
項目参照番号	数値	→	項目参照番号、または 0の場合、リストに最後に追加された項目
入力可	ブール	→	True=入力可、Faule=入力不可
スタイル	数値	→	項目のフォントスタイル
アイコン	数値	→	Macintoshベースの'cicn'リソースID、または Macintoshベースの65536+'PICT'リソースID、 または、131072+ピクチャ参照番号

SET LIST ITEM PROPERTIESコマンドは、引数＜リスト＞によって渡されたリストにあって、＜項目参照番号＞によって渡された参照番号を持つ項目を変更します。

このような項目参照番号を持つ項目が存在しない場合、コマンドは何も動作しません。オプションとして＜項目参照番号＞で0を渡すことによって、**APPEND TO LIST**コマンドを使用してリストに追加した最後の項目を変更することができます。

項目参照番号を使用して作業を行う場合、項目が一意の参照番号を持つリストを作成します。そうでなければ、項目を区別できません。詳細については、**APPEND TO LIST**コマンドの説明を参照してください。

注：項目のテキストまたはそのサブリストを変更するには、SET LIST ITEMコマンドを使用します。

項目を入力可能にするには、引数＜入力可＞でTrueを渡し、そうでない場合はFalseを渡します。

重要：入力可能にするためには、その項目が入力可能なリストに属する必要があります。SET ENTERABLEコマンドを使用すると、リスト全体を入力可能または入力不可能にすることもできます。SET LIST ITEM PROPERTIESコマンドを使用すると、個々のリスト項目を入力可能または入力不可能にすることができます。「入力可」プロパティをリストレベルで変更しても、項目の「入力可」プロパティは変更されません。項目が入力可能になるのは、リストと項目の両方が入力可能な場合のみです。

項目のフォントスタイルは、引数＜スタイル＞で指定します。次の定義済定数の1つまたは複数を組み合わせて渡します。

定数	タイプ	値
Plain	倍長整数	0
Bold	倍長整数	1
Italic	倍長整数	2
Underline	倍長整数	4
Outline	倍長整数	8
Shadow	倍長整数	16
Condensed	倍長整数	32
Extended	倍長整数	64

注：Windowsでは、標準 (Plain)、太字 (Bold)、斜体 (Italic)、アンダーライン (Underline) などのスタイルしか使用できません。

項目にアイコンを関連付ける場合、次の数値のいずれか1つを渡します。

■ N、NはMacOSベースの'cicn'リソースのリソースIDです。

■ Use PICT resource+N、NはMacOSベースの'PICT'リソースのリソースIDです。

■ Use PicRef+N、Nは「デザイン」モードのピクチャライブラリのピクチャの参照番号です。

項目にグラフィックスを使用しない場合は、ゼロを渡します。

注：「Use PICT resource」および「Use PicRef」は、4Dであらかじめ定義されている定数です。

▼ **APPEND TO LIST**コマンドの例を参照してください。

参照：GET LIST ITEM PROPERTIES、SET LIST ITEM

GET LIST ITEM PROPERTIES

GET LIST ITEM PROPERTIES (リスト ; 項目参照番号 ; 入力可 ; スタイル ; アイコン)

引数	タイプ		説明
リスト	リスト参照番号	→	リスト参照番号
項目参照番号	数値	→	項目参照番号、または 0の場合、リストに最後に追加された項目
入力可	ブール	←	True=入力可、Faule=入力不可
スタイル	数値	←	項目のフォントスタイル
アイコン	数値	←	Macintoshベースの'cicn'リソースID、または Macintoshベースの65536+'PICT'リソースID、 または、131072+ピクチャ参照番号

GET LIST ITEM PROPERTIESコマンドは、引数＜リスト＞によって渡されたリスト参照番号を持つリストにあって、＜項目参照番号＞によって渡された参照番号を持つ項目のプロパティを返します。

このコマンドを呼び出すと、次のような結果が生じます。

■ その項目が入力可能な場合、引数＜入力可＞がTrueを返します。
■ 引数＜スタイル＞が、項目のフォントスタイルを返します。
■ 引数＜アイコン＞がその項目に割り当てられているアイコンまたはピクチャを返し、何も割り当てられていない場合は0を返します。

これらのプロパティについての詳細は、**SET LIST ITEM PROPERTIES**コマンドの説明を参照してください。

このような項目参照番号を持つ項目が存在しない場合、コマンドは引数を変更しません。

項目参照番号を使用して作業を行う場合、項目が一意の参照番号を持つリストを作成します。そうでなければ、項目を区別できません。詳細については、**APPEND TO LIST**コマンドの説明を参照してください。

参照：GET LIST ITEM、SET LIST ITEM、SET LIST ITEM PROPERTIES

List item position

List item position (リスト ; 項目参照番号) → 数値

引数	タイプ		説明
リスト	リスト参照番号	→	リスト参照番号
項目参照番号	数値	→	項目参照番号
関数の返す値	数値	←	展開されたリスト内の項目の位置

List item position関数は、引数＜リスト＞によって渡されたリスト参照番号を持つリストにあって、＜項目参照番号＞によって渡された参照番号を持つ項目の位置を返します。

この位置は、メインリストの最初の項目に相対的に表され、リストとそのサブリストの現在の展開／縮小状態が使用されます。

従って、結果は1から**Count list items**関数によって返される値の間の数値です。

項目が縮小されているリストにあるために表示されていない場合、**List item position**関数が適切なリストを展開してその項目を表示します。

項目が存在しない場合、**List item position**関数は 0を返します。

参照：Count list items、SELECT LIST ITEM BY REFERENCE

List item parent

List item parent (リスト ; 項目参照番号) → 数値

引数	タイプ		説明
リスト	リスト参照番号	→	リスト参照番号
項目参照番号	数値	→	項目参照番号
関数の返す値	数値	←	親項目の項目参照番号、親項目がない場合= 0

List item parent関数は、親項目の項目参照番号を返します。

リスト参照番号を引数＜リスト＞で、リストの項目参照番号を引数＜項目参照番号＞で渡します。項目参照番号がリストの既存の項目を参照し、この項目がサブリストにある場合、すなわち親項目を持つ場合、この結果として親項目の項目参照番号が取得されます。

渡した項目参照番号を持つ項目が存在しない場合、または項目が親を持たない場合、**List item parent**関数は 0を返します。

項目参照番号を使用して作業を行う場合、項目が一意の参照番号を持つリストを作成します。そうでなければ、項目を区別できません。詳細については、**APPEND TO LIST**コマンドの説明を参照してください。

例：

リスト名を「hList」とし、「ユーザ」モードで表示します。

加えて、項目の項目参照番号を次のように設定します。

項目	項目参照番号
a	100
a - 1	101
a - 2	102
b	200
b - 1	201
b - 2	202
b - 3	203

▼次のコードを使って、項目“b - 3”を選択すると、「$vlParentItemRef」変数は200を取得しまう。この数値は、項目“b”の項目参照番号です。

```
$vlItemPos:=Selected list item(hList)
GET LIST ITEM(hList ; $vlItemPos ; $vlItemRef ; $vsItemText)
$vlParentItemRef:=List item parent(hList ; $vlItemRef) ` $vlParentItemRefは200を取得する
```

▼項目“a - 1”が選択された場合、「$vlParentItemRef」変数は100を取得するでしょう。この数値は、項目“b”の項目参照番号です。

▼項目“a”または“a”が選択された場合、「$vlParentItemRef」変数は親項目を持っていないので0を取得するでしょう。

参照：GET LIST ITEM、List item position、SELECT ITEM BY REFERENCE、SET LIST ITEM

DELETE LIST ITEM

DELETE LIST ITEM (リスト ; 項目参照番号 | * {; *})

引数	タイプ		説明
リスト	リスト参照番号	→	リスト参照番号
項目参照番号	*	数値	*	→	項目の参照番号 *の場合、現在選択されているリスト項目
*		→	指定した場合、メモリにサブリストがある場合はそれを消去 省略した場合、サブリストがあっても消去されません

DELETE LIST ITEMコマンドは、引数＜リスト＞によって渡されたリスト参照番号を持つリストから項目を削除します。

2番目の引数として＜＊＞を渡す場合、リストで現在選択されている項目が削除されます。

そうでない場合は、削除する項目の項目参照番号を指定します。指定した項目参照番号を持つ項目が存在しない場合、コマンドは何も行いません。

項目参照番号を使用して作業を行う場合、項目が一意の参照番号を持つリストを作成します。そうでなければ、項目を区別できません。詳細については、**APPEND TO LIST**コマンドの説明を参照してください。

どの項目を削除しようと、オプション引数の＜＊＞を指定して、その項目にサブリストが添付されている場合、4Dにそれを自動的に削除させることができます。＜＊＞を指定しない場合、あらかじめ項目に添付されているサブリストのリスト参照番号を取得するとよいでしょう。というのは、結局**CLEAR LIST**コマンドを使用してそれを削除する必要があるからです。

▼ 次の例は、「hList」というリストで現在選択されている項目を削除します。項目にサブリストが添付されている場合、サブリスト(およびサブリストのサブリスト)が消去されます：

```
DELETE LIST ITEM(hList ; * ; *)
REDRAW LIST(hList) `REDRAW LISTコマンドを忘れずに呼び出してください。
                   `さもなければ、リストが更新されません。
```

参照：CLEAR LIST、GET LIST ITEM

GET LIST ITEM

GET LIST ITEM (リスト ; 項目位置 ; 項目参照番号 ; 項目テキスト {; サブリスト ; {; 展開}})

引数	タイプ		説明
リスト	リスト参照番号	→	リスト参照番号
項目位置	数値	→	展開されたリスト内の項目の位置
項目参照番号	数値	←	項目の参照番号
項目テキスト	文字列	←	リスト項目のテキスト
サブリスト	リスト参照番号	←	サブリスト参照番号
展開	ブール	←	サブリストがある場合、 サブリストが現在展開されている場合はTrue サブリストが現在縮小されている場合はFalse

GET LIST ITEMコマンドは、引数＜リスト＞によって渡された参照番号を持つリストにあって、＜項目参照番号＞によって渡された位置を持つ項目についての情報を返します。

位置は、リストとそのサブリストの展開／縮小の現在の状態を使用して相対的に表されます。位置としては、1から**Count list items**関数によって返された値までの数値を渡します。この範囲外の値を渡すと、**GET LIST ITEM**コマンドは引数を変更せずに返します。

このコマンド呼び出すと、次の情報を取得できます。

■ 引数＜項目参照番号＞の項目の項目参照番号

■ 引数＜項目テキスト＞の項目のテキスト

オプション引数の＜サブリスト＞と＜展開＞を渡すと、以下のようになります。

■ ＜サブリスト＞が項目に添付されているサブリストのリスト参照番号を返します。項目にサブリストがない場合、subListはゼロを返します。

■ 項目にサブリストがある場合、サブリストが現在展開されていれば、＜展開＞は Trueを返し、縮小されていれば Falseを返します。

例：

(1) リスト「hList」が一意の参照番号が付いた項目を持つ場合、次のコードは現在選択されている項目に添付されているサブリストが存在すれば、その展開／縮小の状態をプログラムによって切り換えます。

```
$vlItemPos:=Selected list item(hList)
If ($vlItemPos>0)
   GET LIST ITEM(hList;$vlItemPos;$vlItemRef;$vsItemText;$hSublist;$vbExpanded)
   If (Is a list($hSublist))
      SET LIST ITEM (hList ; $vlItemRef ; $vsItemText ; $vlItemRef ; $hSublist ; Not ($vbExpanded))
      REDRAW LIST(hList)
   End if
End if
```

(2) **APPEND TO LIST**コマンドの例を参照してください。

参照：GET LIST ITEM PROPERTIES、List item parent、List item position、Selected list item、SET LIST ITEM、SET LIST ITEM PROPERTIES

SET LIST ITEM

SET LIST ITEM (リスト ; 項目参照番号 ; 新規項目テキスト ; 新規項目参照番号{; サブリスト ; {; 展開}})

引数	タイプ		説明
リスト	リスト参照番号	→	リスト参照番号
項目参照番号	数値	→	項目の参照番号 0の場合、リストに最後に追加された項目
新規項目テキスト	文字列	→	新規リスト項目のテキスト
項目参照番号	数値	→	新規項目の参照番号
サブリスト	リスト参照番号	→	項目に付着する新規サブリスト 項目に添付されるサブリストまたは サブリストがない場合は 0、 または変更がない場合は -1
展開	ブール	→	サブリストが展開されるか縮小されるか を示す

SET LIST ITEMコマンドは、引数＜リスト＞によって渡される参照番号を持つリストにあって、＜項目参照番号＞によって渡される項目参照番号を持つ項目を変更します。

このような項目参照番号を持つ項目が存在しない場合、コマンドは何も行いません。オプションで＜項目参照番号＞で0を渡して、**APPEND TO LIST**コマンドを使用してリストに追加した最後の項目を変更することができます。

項目参照番号を使用して作業を行う場合、項目が一意の参照番号を持つリストを作成します。そうでなければ、項目を区別できません。詳細については、**APPEND TO LIST**コマンドの説明を参照してください。

項目の新たなテキストは、＜新規項目テキスト＞によって渡します。項目参照番号を変更する場合は、＜新規項目参照番号＞で新たな値を渡します。そうでない場合は、＜項目参照番号＞と同じ値を渡します。

項目にリストを添付する場合、リスト参照番号を＜サブリスト＞で渡します。この場合、新たなサブリストが展開されている場合は＜展開＞でTrueを、そうでない場合はFalseを渡します。

項目に添付されているサブリストを分離する場合は、＜サブリスト＞で 0を渡します。この場合、**GET LIST ITEM**コマンドを使用して、そのリストのあらかじめ取得された参照番号を調べておくとよいでしょう。そうすれば、それが必要なくなったときに**CLEAR LIST**コマンドを使用して削除することができます。

項目の＜サブリスト＞プロパティを変更しない場合は、＜サブリスト＞で-1を渡します。

例：

(1) リスト「hList」が一意の参照番号が付いた項目を持つ場合、次のボタン用オブジェクトメソッドは、リスト「hList」の中で現在選択されているリスト項目にサブ項目を追加します。

```
$vlItemPos:=Selected list item(hList)
If ($vlItemPos>0)
   GET LIST ITEM(hList ; $vlItemPos ; $vlItemRef ; $vsItemText ; $hSublist ; $vbExpanded)
   $vbNewSubList:=Not(Is a list($hSublist))
   If ($vbNewSubList)
      $hSublist:=New list
   End if
   vlUniqueRef:=vlUniqueRef+1
   APPEND TO LIST($hSubList ; "新規項目" ; vlUniqueRef)
   If ($vbNewSubList)
      SET LIST ITEM(hList ; $vlItemRef ; $vsItemText ; $vlItemRef ; $hSublist ; True)
   End if
   SELECT LIST ITEM BY REFERENCE(hList ; vlUniqueRef)
   REDRAW LIST(hList)
End if
```

(2) **GET LIST ITEM**コマンドの例を参照してください。

(3) **APPEND TO LIST**コマンドの例を参照してください。

参照：GET LIST ITEM、GET LIST ITEM PROPERTIES、SET LIST ITEM PROPERTIES

Selected list item

Selected list item (リスト) → 倍長整数

引数	タイプ		説明
リスト	リスト参照番号	→	リスト参照番号
関数の返す値	倍長整数	←	展開されたリストの現在選択されているリスト項目の位置

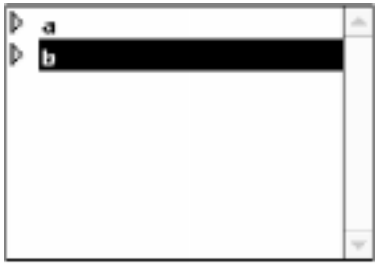
Selected list item関数は、引数＜リスト＞によって渡された参照番号を持つリストの中の選択された項目の位置を返します。

この関数は、フォームに表示されるリストに適用されて、ユーザがどの項目を選択したかを検出します。

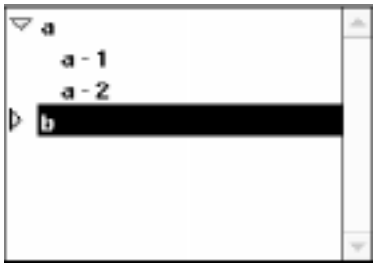
リストにサブリストがある場合、この関数はメインリスト(フォームで実際に定義されたリスト)のサブリストではなく、メインリストに適用します。位置は、リストとそのサブリストの現在の展開／縮小状態を使用して、メインリストの最初の項目に相対的に表現されます。

例：

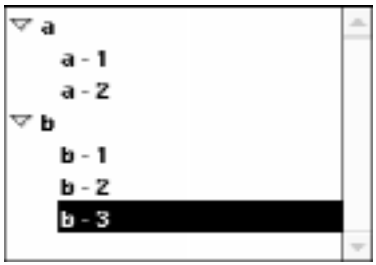
次は、「ユーザ」モードに表示された「hList」という名前のリストです。



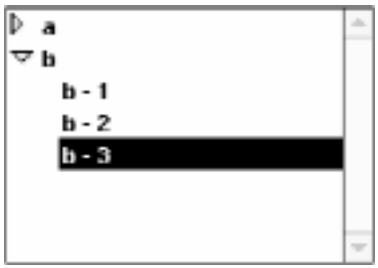
$vlItemPos:=Selected list item(hList) ` この時点では$vlItemPosは2を取得する



$vlItemPos:=Selected list item(hList)`この時点では$vlItemPosは4を取得する



$vlItemPos:=Selected list item(hList)`この時点では$vlItemPosは7を取得する



$vlItemPos:=Selected list item(hList)`この時点では$vlItemPosは5を取得する

参照：SELECT LIST BY REFERENCE、SELECT LIST ITEM

SELECT LIST ITEM

SELECT LIST ITEM (リスト ; 項目位置)

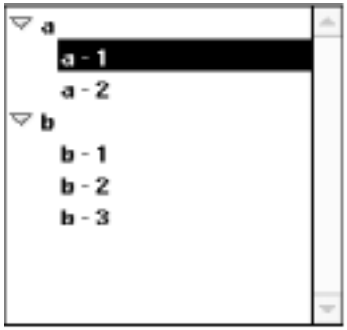
引数	タイプ		説明
リスト	リスト参照番号	→	リスト参照番号
項目位置	数値	→	展開されたリスト内の項目の位置

SELECT LIST ITEMコマンドは、引数＜リスト＞によって渡された参照番号を持つリストにあって、＜項目位置＞によって渡された位置にある項目を選択します。

引数＜項目位置＞は、リストとそのサブリストの展開／縮小の現在の状態を使用して相対的に表される位置です。位置は、1から**Count list items**関数によって返された値までの数値を渡します。この範囲外の値を渡すと、デフォルトでは最初の項目が選択されます。

例：

次は、「ユーザ」モードに表示された「hList」という名前のリストです。



次のコードを実行すると、

```
SELECT LIST ITEM(hList ; Count list items(hList))
REDRAW LIST(hList) `REDRAW LISTコマンドを忘れずに呼び出します。
                   `そうでなければ、リストが更新されません。
```

表示されているリスト項目の最後の項目が選択されます。

参照：SELECT LIST ITEM BY REFERENCE、Selected list item

SELECT LIST ITEM BY REFERENCE

SELECT LIST ITEM BY REFERENCE (リスト ; 項目参照番号)

引数	タイプ		説明
リスト	リスト参照番号	→	リスト参照番号
項目参照番号	数値	→	項目参照番号

SELECT LIST ITEM BY REFERENCEコマンドは、引数＜リスト＞によって渡された参照番号を持つリストの中の＜項目参照番号＞によって渡された項目参照番号を持つ項目を選択します。

渡した項目参照番号の項目が存在しない場合、コマンドは何も行いません。

項目が縮小されているリストにあるために表示されていない場合、このコマンドが必要なサブリストを展開して選択した項目を表示します。

項目参照番号を使用して作業を行う場合、項目が一意の参照番号を持つリストを作成します。そうでなければ、項目を区別できません。詳細については、**APPEND TO LIST**コマンドの説明を参照してください。

例：

「hList」が一意の参照番号が付いた項目を持つリストとし、現在選択されている項目に親項目が存在する場合には、次のボタン用オブジェクトメソッドがその親項目を選択するとします。

```
$vlItemPos:=Selected list item(hList) ` 選択された項目の位置を取得する
`選択された項目の項目参照番号を取得
GET LIST ITEM(hList ; $vlItemPos ; $vlItemRef ; $vsItemText)
`親項目が存在する場合は、親項目の項目参照番号を取得
$vlParentItemRef:=List item parent(hList ; $vlItemRef)
If ($vlParentItemRef>0)
   `親項目を選択する
   SELECT LIST ITEM BY REFERENCE(hList ; List item parent(hList ; $vlItemRef))
   REDRAW LIST(hList) `REDRAW LISTを忘れずに呼び出します。
                      `そうでなければ、リストが更新されません。
End if
```

参照：SELECT LIST ITEM、Selected list item

データ読み込みとデータ書き出しコマンド　28

この章では、「ルーチン」エディタの「Import and Export」テーマ内にあるデータ読み込み／データ書き出しコマンドについて説明します。この章のコマンドは、データを読み込んだり、書き出すために使用します。どちらも、フォームを介して処理します。

EXPORT DIF
IMPORT DIF
EXPORT SYLK
IMPORT SYLK
EXPORT TEXT
IMPORT TEXT

EXPORT DIF
EXPORT SYLK
EXPORT TEXT

EXPORT DIF ({テーブル} ; 文書)
EXPORT SYLK ({テーブル} ; 文書)
EXPORT TEXT ({テーブル} ; 文書)

引数	タイプ	説明
テーブル	テーブル名	データを書き出すテーブル
文書	文字列	データが書き出される文書テーブル

データの書き出しコマンドは、＜テーブル＞のカレントセレクションのレコードをディスクに書き出します。このデータは＜文書＞に書き込まれます。＜文書＞は、標準的なテキスト形式の文書テーブルです。

データの書き出し処理は、カレント出力フォームを介して実行します。このデータの書き出し処理は、出力フォーム上のフィールドや変数をその入力順序に従って書き出します。このため、書き出したい入力可能オブジェクトとフィールドのみを含んだ出力フォームを使用する必要があります。書き出し用のフォームにボタンやその他関係ないオブジェクトを配置しないでください。

注：サブフォームは無視されます。

書き出すレコードに対して、フォームメソッドをそれぞれ1回ずつ実行します。この場合は、Beforeフェーズだけが“ True（真）”となります。

引数＜文書＞には、新規または既存の文書ファイルを指定することができます。＜文書＞が既存の文書ファイルと同じ名前の場合には、その文書ファイルは上書きされます。また、＜文書＞にはボリューム名やフォルダ名などのパスを指定することもできます。

データの書き出し処理中にその進捗状況を表すインジケータ(ナンバーやサーモメータ)を表示します。インジケータの「中止」ボタンをクリックすると、処理を中断します。中断されたかどうかは、システム変数OKの値を調べます。書き出しが完了していれば1が、中断されていれば 0が代入されています。インジケータを表示したくない場合には、**MESSAGES OFF**コマンドを使用してください。

データの書き出し処理は、カレントASCII出力テーブルを使用して行います。ASCII出力テーブルは、他のマシンや他のソフトウェアに用いるデータ変換のために使用します。ASCII出力テーブルに関する詳細は、『4th Dimension / 4D First ユーザリファレンス』を参照してください。

EXPORT TEXTコマンドのデフォルトのフィールド区切り文字とレコード区切り文字は、それぞれタブ(ASCIIコードの9)とキャリッジリターン(ASCIIコードの13)ですが、2つのシステム変数FldDelimitとRecDelimitに値を代入することによって、これらの区切り文字を変更することができます。またデータ書き出しダイアログボックスでこれらの指定を修正することもできます。付録Bの「ASCIIコード」を参照してください。

テキストフィールドには、キャリッジリターンを含めることができますので、テキストフィールドを書き出す場合には、区切り文字としてキャリッジリターンを使用していないかどうか注意してください。

▼ 次の例は、データを文書ファイルに書き出します。まず、メソッドの最初で出力フォームを設定し、次にシステム変数を使用して区切り文字を修正します：

```
OUTPUT FORM ([従業員] ; "書き出し")    ` データ書き出しのフォームを設定
FldDelimit:=27                         ` フィールド区切り文字にエスケープ
RecDelimit:=10                         ` レコード区切り文字にラインフィード
EXPORT TEXT ([従業員] ; "従業員文書")  ` 従業員文書にデータを書き出す
```

IMPORT DIF
IMPORT SYLK
IMPORT TEXT

IMPORT DIF ({テーブル} ; 文書)
IMPORT SYLK ({テーブル} ; 文書)
IMPORT TEXT ({テーブル} ; 文書)

引数	タイプ	説明
テーブル	テーブル名	データが読み込まれるテーブル
文書	文字列	データを読み込むMacintosh文書テーブル

データの読み込みコマンドは、標準的なテキスト形式の文書テーブルである＜文書＞から＜テーブル＞にデータを読み込みます。

データの読み込み処理は、カレント入力フォームを介して実行します。このデータの読み込み処理は、入力フォーム上のフィールドや変数をその入力順序に従って読み込みます。このため、読み込みたい入力可能オブジェクトとフィールドのみを含んだ入力フォームを使用する必要があります。読み込み用のフォームにボタンやその他関係ないオブジェクトを配置しないでください。

注：サブフォームは無視されます。

入力フォーム上のフィールドや変数の数が読み込んだレコードのフィールドと一致しない場合には、余分なものが無視されます。組み込みフォームは無視されます。フォームメソッドは、読み込んだレコードに対して、それぞれ1回ずつ実行します。この場合はAfterフェーズだけが“ True（真）”となります。

引数＜文書＞には、ボリューム名やディレクトリ（Macintosh版では、フォルダ）名などのパスを指定できます。

データの読み込み処理中にその進捗状況を表わすインジケータ(ナンバーやサーモメータ)を表示します。インジケータの「中止」ボタンをクリックすると、処理を中断します。既に読み込まれたレコードは、ユーザが「中止」ボタンをクリックした場合に消去されます。中断されたかどうかは、システム変数OKの値を調べます。読み込みが完了していれば1が、中断されていれば 0が代入されています。インジケータを表示したくない場合には、**MESSAGES OFF**コマンドを使用してください。

データの読み込み処理は、カレントASCII出力テーブルを使用して行います。ASCII出力テーブルは、他のマシンや他のソフトウェアに用いるデータ変換のために使用します。ASCII出力テーブルに関する詳細は、『4th Dimension / 4D Firstユーザリファレンス』を参照してください。

IMPORT TEXTコマンドの、デフォルトのフィールド区切り文字とレコード区切り文字は、それぞれタブ(ASCIIコードの9)とキャリッジリターン(ASCIIコードの13)ですが、2つのシステム変数FldDelimitとRecDelimitに値を代入することによって、これらの区切り文字を変更することができます。また、「データ読み込み」ダイアログボックスでこれらの指定を修正することもできます。付録Bの「ASCIIコード」を参照してください。

テキストフィールドにはキャリッジリターンを含めることができますので、テキストフィールドを読み込む場合には、区切り文字としてキャリッジリターンを使用していないかどうか注意してください。

▼ 次の例は、データを文書ファイルから読み込みます。まず、メソッドの最初で入力フォームを設定し、次にシステム変数を使用して区切り文字を修正します：

```
INPUT FORM ([従業員] ; "読み込み")      ` データ読み込みのフォームを設定
FldDelimit:=27                          ` フィールド区切り文字にエスケープ
RecDelimit:=10                          ` レコード区切り文字にラインフィード
IMPORT TEXT ([従業員] ; "従業員文書")   ` 従業員文書からデータを読み込む
```

割り込みコマンド　29

この章では、「ルーチン」エディタの「Interruptions」テーマ内にある割り込みコマンドについて説明します。この章のコマンドは、メソッド実行の監視、エラーとイベントの管理を行います。

ON ERR CALL　　**ABORT**　　**ON SERIAL PORT CALL**
ON EVENT CALL　　**FILTER EVENT**

ON ERR CALL

ON ERR CALL (エラープメソッド)

引数	タイプ	説明
エラーメソッド	文字列	インストールするエラー処理メソッド

ON ERR CALLコマンドは、エラーを管理するエラー処理メソッドである＜エラーメソッド＞をインストールします。＜エラーメソッド＞に空の文字列を指定すると、エラーの管理を4th Dimensionに返します。インストール後は、エラーが発生するたびに4th Dimensionが＜エラーメソッド＞を呼び出します。

エラーの種別は、システム変数Errorの値で判断します。通常、エラー処理メソッドからは、エラーメッセージが表示されるはずです。このマニュアルの付録Cに、4th Dimensionのエラーコードが記載されています。また、付録Cには一般的なエラーコードも記載されています。

エラーは、次のようなものから生成されます：

■ **4th Dimensionのデータベースエンジン：**例えば、レコードを保存している最中に、重複不可のインデックスを複製しようとした場合。

■ **4th Dimension環境：**例えば、配列に割り当てるために必要なメモリーを持っていない場合。

■ **データベースが起動しているオペレーションシステム：**例えば、ディスクに空きがなかったり、I/O (入出力)エラーの場合。

実行を終了するには、**ABORT**コマンドを使用します。インストールしたメソッドに**ABORT**コマンドを使用しないと、4th Dimensionは割り込みをかけたメソッドに制御を戻し、メソッドを実行し続けます。

ON ERR CALLコマンドでインストールしたエラー処理メソッドの内部にエラーが発生した場合は、エラーの管理は4th Dimensionに返ります。従って、エラー処理メソッドの内部でエラーが発生しないように十分注意してください。また、エラー処理メソッドの内部では、**ON ERR CALL**コマンドを使用することはできません。

カスタムデータベース内のすべてのデータベースアプリケーションのエラーに対処するために**ON ERR CALL**コマンドは通常、「On Startup」データベースメソッドの中で使用します。また、あるメソッド固有のエラー処理を行うために、そのメソッドの先頭で使用する場合もあります。

ON ERR CALLメソッドをインストールすると、Windows上では Altキー、Macintosh上では optionキーを押しながらマウスボタンをクリックして、デバッガを起動することができません。“ Alt (option) ”キーを押しながらマウスボタンをクリックすると、エラーコードを生成し、**ON ERR CALL**メソッドが起動されてしまうためです。

▼ 次の例は、エラー処理メソッド“ エラー処理 ”をインストールします：

```
ON ERR CALL ("エラー処理")
```

エラー処理メソッド“ エラー処理 ”のコードを次に示します。このメソッドは、ダイアログボックスにエラーコードを表示します。ユーザが、ダイアログボックスの「OK」ボタンをクリックすると、**ABORT**コマンドが実行され、すべてのメソッドの実行が終了します：

```
` エラー処理：ON ERR CALLでインストールされる
CONFIRM ("エラー番号" + String (Error) + "  終了しますか？")
If (OK=1)
  ABORT                                  ` メソッドを終了し、メニューへ戻る
End if
```

ABORT

ABORT

ABORTコマンドは、メソッドの実行を中止します。「シンタックスエラー」ウインドウやデバッガの「アボート」ボタンをクリックした場合と同じです。

ABORTコマンドは、ほとんど使用されません。また、データベースアプリケーションを終了するために使用するべきではありません。通常、**ABORT**コマンドはアプリケーション開発時の予測できないエラーに対してデバッグするために使用します。

フォームが表示されている状態で**ABORT**コマンドを実行すると、メソッドは実行を中止しフォームに制御を返します。しかし、フォームがAfterフェーズの場合は、**ABORT**コマンドはフォームメソッドの実行を中止します。そして、フォームの表示を取りやめ、レコードは保存されます。**ABORT**コマンドは、システム変数OKに対して影響を与えません。単にフォームメソッドの実行を中止するだけです。

フォームが表示されていない状態で**ABORT**コマンドを実行すると、メソッドは実行を中止し、メニューに制御を戻します。

ABORTコマンドは慎重に使用してください。エラーが発生しても、エラーの発生したメソッドに戻った方が良い場合もあります。例えば、**RECEIVE PACKET**コマンドでテーブルを読み込んでいるときにエラーが発生した場合は、元のメソッドに戻ってテーブルを閉じる必要があります。

ON EVENT CALL

ON EVENT CALL (イベントメソッド ; {プロセス})

引数	タイプ	説明
イベントメソッド	文字列	インストールするイベントメソッド
プロセス	文字列	プロセス名

ON EVENT CALLコマンドは、イベントを管理するメソッドである＜イベントメソッド＞をインストールします。＜イベントメソッド＞に空の文字列を指定すると、イベントの管理を4th Dimensionに返します。＜イベントメソッド＞は別プロセスとしてインストールされるので、4th Dimensionのメソッドが1つも実行されなくても、常にアクティブになります。インストール後はイベントが発生するたびに4th Dimensionが＜イベントメソッド＞を呼び出します。管理できるイベントは、マウスボタンのクリックとキーボードからの入力です。引数＜プロセス＞は、**ON EVENT CALL**コマンドで作成されます。＜プロセス＞の先頭にドル記号($)を付けると、＜プロセス＞はローカルプロセスになります。

警告：＜イベントメソッド＞にTRACEコマンドを記述することはできません。なぜなら、イベントがデバッグウインドウを開くたびに無限ループになってしまうからです。

メソッドを実行させたり、イベントを認識させたりする必要はありません。いったんイベントがインストールされると、イベントは「デザイン」モードに含まれると同時にすべてのモードからイベントプロセスがパスされます。

“ Ctrl - Shift -Alt - Backspace ”（Macintosh版では、「コマンド - シフト - オプション - コントロール- delete」）キーでイベントプロセスを通常のプロセスに変換します。従って、それ以降、発生するすべてのイベントがこのイベントメソッドに自動的に渡されなくなります。例えば、誤って**FILTER EVENT**コマンドがイベントメソッド内の唯一のコマンドになってしまったときなどにこれを利用します。

イベント処理メソッドでは、MouseDown、KeyCode、Modifiers、MouseX、MouseY、MouseProcの6つのシステム変数を読み取ることができます。これらの変数がプロセス変数であるという点に注目してください。

システム変数MouseDownには、マウスボタンがクリックされると、1が代入されます。それ以外の場合は、0が代入されます。

システム変数KeyCodeには、押されたキーのASCIIコードが代入されます。付録Bに、ASCIIコード表が記載されています。

システム変数Modifiersは、モデファイキーの値が入ります。システム変数Modifiersはイベントが発生した時点で、次のモディファイアキーが押されていたかどうかを示します：

プラットフォーム	モディファイアキー
Windows	Shift key、Caps Lock、Alt key、Right mouse button
Macintosh	Shift key、Caps Lock、Option key、Control key、Command key

モディファイアキーだけでは、イベントは発生しません。マウスボタンをクリックするか、あるいはキーを押した時にだけイベントが発生します。

システム変数MouseXとMouseYは、マウスの縦と横の位置が入ります。ウインドウの左上隅の位置が0.0です。これらはマウスクリックした場合にのみ有効です。システム変数MouseProcには、マウスクリックでも打鍵でも、イベントが発生したプロセスのプロセス参照番号が入ります。

▼ 次の例は、ユーザがどのモディファイアアキーを押したかを調べ、5つのインタープロセス変数で結果を返します。5つのブール変数(<>Ctrl、<>Shift、<>Caps Lock、<>Alt)はキーが押された場合に“ True ”を、キーが押されなかった場合に“ False ”を代入します。また、変数<>RightMは、右マウスボタンがクリックされた場合に“ True ”を、クリックされなかった場合に“ False ”を代入します：

```
C_BOOLEAN (<>Ctrl ; <>Shift ; <>CapsLock ; <>Alt ; <>RightM)
$M:=Modifiers // 256  ` $Mは任意の変数名
` $Mのモジューロが1であれば、任意のモデファイキーが押されたことになる
<>Ctrl:=(($M % 2) =1) `Macintosh版では、「<>Command」
$M:=$M // 2
<>Shift:=(($M % 2) =1)
$M:=$M // 2
<>CapsLock:=(($M % 2) =1)
$M:=$M // 2
<>Alt:=(($M % 2) =1) `Macintosh版では、「<>Option」
$M:=$M // 2
<>RightM:=(($M % 2) =1)
```

MouseDown、KeyCode、Modifiers、MouseX、MouseY、およびMouseProcの各システム変数は、ON EVENT CALLメソッド内でのみ有効な値を持ちます。

ON EVENT CALLコマンドは乱用しないでください。通常、すべてのイベントを監視するコード部分の前に入れ、その部分の直後に空の文字列引数でクリアします。

▼ 次の例は、ユーザが “ コマンド-ピリオド ” を押したら、印刷を中止します。まず、イベントを操作するメソッドをインストールします。次にユーザにメッセージを表示して、印刷を中止できることを知らせます。イベントメソッド内でインタープロセス変数 “ <>vCancel ” に “ True ” が代入されると、既に印刷されたレコードの数をユーザに知らせます：

最後にイベントメソッドをクリアします：

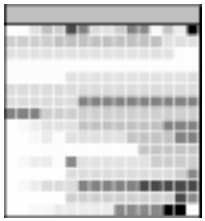
```
<>vCancel:=False                                    ` インタープロセス制御フラグ
ON EVENT CALL ("EvtProc1")
ALL RECORDS ([従業員])
MESSAGE ("プリントを中止するには、Ctrl (コマンド) - ピリオドを押します")
For ($i;1;Records in selection ([従業員]))          ` 印刷ループ
   If (<>vCancel)
      ALERT ("レコード数："+ String ($i)+" ／ "+String (Records in selection ([従業
員]))+"で印刷が中止されました")
   $i:=Records in selection ([従業員])+1            ` 印刷ループを中止する
   Else                                             ` ユーザによって印刷が中止されてない場合
      PRINT LAYOUT ([従業員] ; "レポート")
   End if
End for
FORM FEED
ON EVENT CALL ("")
```

イベント処理メソッドはキーが押されたかどうかをチェックします。キーが押された場合は、インタープロセス変数 “ <>vCancel ” に “ True ” を代入します：

```
$ﾓﾃﾞｨﾌｧｲｱｷｰ:=Modifiers // 256                       ` modifier変数の値
$ｺﾏﾝﾄﾞｷｰ:=$ﾓﾃﾞｨﾌｧｲｱｷｰ % 2                         ` ブールに変換
` 「コマンド-ピリオド」キーが押された場合
If ($ｺﾏﾝﾄﾞｷｰ=1) & (KeyCode=46)
   CONFIRM ("本当に、印刷を中止してもいいですか？")
   If (OK=1)
      <>vCancel:=True
   End if
End if
```

▼次の例は、**ON EVENT CALL**コマンドを使用してカラーパレットを管理します。カラーパレットはフローティングウインドウ内にあります。ユーザが色をクリックすると、他のメソッドのサーモメータがその色に設定されます。イベントプロセス“ EventHandler ”がイベント処理メソッドとしてインストールされます。フローティングウインドウは、**OPEN WINDOW**コマンドで開きます。

パレットは次のようにダイアログで表示されます：



この例では、4th Dimensionの標準カラーパレットを使用します。次の呼び出しでカラーパレットプロセスを作成します：

```
<>Pパレット:=New process ("カラーパレット" ; 16000 ; $ColorPalette)
```

“ カラーパレット ”メソッドがイベントメソッドをインストールします：

```
` イベントメソッドをインストールする
ON EVENT CALL ("EventHandler" ; "$EventHandler")
` パレットを開く
OPEN WINDOW (250 ; 50 ; 394 ; 193 ;-(45*16) ; "Color" ; "G_CloseCancel")
` ウインドウにパレットを表示する
DIALOG ([ファイル1] ; "カラーパレット")
` イベントプロセスを取り除く
ON EVENT CALL ("")
` 変数をゼロにリセットする
<>Pパレット:=0
```

イベント処理メソッド“ EventHandler ”は、パレット内で発生するイベントを管理します。パレット内でイベントが発生すると、クリックされた色を計算し、その番号をインタープロセス変数“ <>ColorID ”に代入します。別のプロセスを呼び出して色を更新します。“ EventHandler ”プロセスは次のようになります：

```
If (MouseDown = 1)                                  ` マウスクリックされたか？
   If (<>Pパレット=MouseProc)                       ` カラーパレットプロセス内か？
      If ((MouseX >= 0) & (MouseY >= 0))            ` タイトルバー内か？
         ` カラーを計算する
         $ColorID:=(MouseX // 9) + ((MouseY // 9) * 16)
         If ($ColorID <= 255)                       ` カラーパレットかどうかを確かめる？
```

```
            <>ColorID:=$ColorID              ` 値を代入する
            CALL PROCESS (<>Pパレット) ` カラーパレットウインドウを呼び出す
            FILTER EVENT                     ` イベントを取り除く
         End if
      End if
   End if
End if
```

FILTER EVENT

FILTER EVENT

FILTER EVENTコマンドは、**ON EVENT CALL**メソッドの中から呼び出すことができます。このコマンドを使用すると、イベントキューからイベントを取り除きます。

すべてのイベントはイベントメソッドに渡されます。イベントメソッドが**FILTER EVENT**コマンドを呼び出すと、そのイベントは4th Dimensionでは処理されません。

FILTER EVENTコマンドだけから成るイベントメソッドを作成しないようにしてください。このメソッドはすべてのイベントを取り除きますので、そのようなイベントメソッドがある場合には“ Ctrl - Shift -Alt - Backspace ”（Macintosh版では、「コマンド - シフト - オプション - コントロール- delete」）キーでそのイベントを終了します。これにより、イベントプロセスを通常のプロセスに変換します。

前述の**ON EVENT CALL**コマンドの例を参照してください。

ON SERIAL PORT CALL

ON SERIAL PORT CALL (シリアルメソッド; {プロセス})

引数	タイプ	説明
シリアルメソッド	文字列	呼び出すメソッド
プロセス	文字列	プロセス名

このコマンドは、旧バージョンの4th Dimensionデータベースとの互換性を保つ目的でのみ提供されています。ほとんどのケースでは、RECEIVE BUFFERコマンドを使用する方が有効です。

ON SERIAL PORT CALLコマンドは、別プロセスのシリアルポートイベントを管理するための割り込みメソッドをインストールします。4th Dimensionは、**SET CHANNEL**コマンドで開くシリアルポートバッファに文字が入力された時点で、自動的に割り込みメソッドを呼び出します。＜シリアルメソッド＞に空の文字列を指定すると、シリアルポートのイベントに対する割り込み処理をオフにします。オプション引数＜プロセス＞を指定すると、シリアルメソッドは＜プロセス＞という名前の別プロセスとしてインストールされます。＜プロセス＞の先頭にドル記号($)を付けるとローカルプロセスになります。

割り込みメソッドが別プロセスとしてインストールされていると、その他のすべてのプロセスと同時に実行されます。

4th Dimensionは、シリアルポートバッファに1つまたは複数の文字を含む場合に割り込みメソッドを自動的に呼び出します。バッファが何を含んでいるのかわからない場合は、**RECEIVE BUFFER**コマンドを実行してバッファをクリアしてください。バッファをクリアしないと何度も割り込みメソッドを呼び出します。

警告：4th Dimensionバージョン6では、シリアルポートをハンドルできる独自のプロセスを作成する方が得策です。このプロセスで、シリアルポートからデータを受け取ったり、またはシリアルポートにデータを送ることができます。そして、インタープロセス通信を使って、他のプロセスとのデータのやり取りを行うことができます。ON SERIAL PORT CALLコマンドの使用には、十分注意が必要です。

▼次のステートメントは、割り込みメソッド“ 割り込み ”をインストールします：

```
ON SERIAL PORT CALL ("割り込み")
```

メソッド“ 割り込み ”は、シリアルポートバッファの内容をすべて取り出し、インタープロセス変数“ <>受け取り ”に格納します。“ <>受け取り ”は、アプリケーションの他の部分で読み取ることができます。次に、割り込みメソッド“ 割り込み ”を示します：

```
RECEIVE BUFFER ($v)                 ` シリアルポートバッファを読み込む
<>受け取り:=<>受け取り + $v          ` データを変数に格納
```

次のステートメントは、インストールした割り込みメソッドを削除します。

```
ON SERIAL PORT CALL ("")
```

ランゲージコマンド　30

この章では、「ルーチン」エディタの「Language」テーマ内にあるランゲージコマンドについて説明します。この章のコマンドは、さまざまなデータベース関数に使用されます。

Command name	**Is a variable**	**Self**
Count parameters	**Nil**	**TRACE**
EXECUTE	**NO TRACE**	**Type**
Get pointer	**RESOLVE POINTER**	

EXECUTE

EXECUTE (ステートメント)

引数	タイプ	説明
ステートメント	文字列	実行するコード

EXECUTEコマンドは、1行の＜ステートメント＞を実行します。ステートメントの文字列は1行だけです。＜ステートメント＞に空の文字列を指定した場合には、**EXECUTE**コマンドは何も行いません。

実行できるステートメントが1行でも、プロジェクトメソッドを指定すれば、適切な処理を実行することができます。**EXECUTE**コマンドを使用すると実行速度が遅くなりますので、なるべく使用を控えるようにしましょう。

使用できるステートメントを次に記します：

- プロジェクトメソッド
- コマンド
- 代入式
- プロセス（グローバル）変数

ステートメントにフロー制御文(If文など)を使用することはできません。

▼ 次の例は、入力されたステートメントを実行するメソッドです。最後の**ABORT**コマンドは、デバッガの中でエラーを発生させないためのものです：

```
$コマンド:=""                                  ` コマンド用の文字列変数を初期化
Repeat
   ` ユーザにステートメントを入力してもらう
 $コマンド:=Request ("実行：" ; $コマンド)
 If (OK=1)                                     ` OKがクリックされた
   EXECUTE ($コマンド)                         ` コマンドを実行
 End if
Until (OK=0)                                   ` キャンセルがクリックされるまで
ABORT
```

TRACE
NO TRACE

TRACE
NO TRACE

データベースの開発時に、**TRACE**コマンドと**NO TRACE**コマンドを使用してメソッドをトレースすることができます。

TRACEコマンドは、4th Dimensionのデバッガを起動します。コードの次の行が実行される前に“ デバッグ ”ウインドウを表示します。実行するコードを表示しながらメソッドの実行を続けることができます。メソッドの実行中にWindows上では Altキー、Macintosh上では optionキーを押しながらマウスボタンをクリックして、デバッガを起動することもできます。

NO TRACEコマンドは、**TRACE**コマンド、エラー、ユーザによって起動されたデバッガを終了します。“ デバッグ ”ウインドウの「トレースなし」ボタンをクリックしても同じ動作をします。

Count parameters

Count parameters → 数値

Count parameters関数は、メソッドに渡された引数の数を返します。**Count parameters**関数は、サブルーチンとして他のメソッドから呼び出されるプロジェクトメソッド内でのみ意味を持ちます。**Count parameters**関数を含んだプロジェクトメソッドがメニューに割り当てられた場合、**Count parameters**関数は -1を返します。

▼ 次の例は、受け渡された文字列の引数をすべて連結して返す関数です：

```
$0:=""                                  ` 返す値を空の文字列で初期化
For ($i ; 1 ; Count parameters)         ` 引数の数だけ繰り返す
  $0:=$0 + $ {$i}                       ` 引数を連結
End for
```

Is a variable

Is a variable (引数) → ブール

引数	タイプ	説明
引数	ポインタ	判定するデータオブジェクト

Is a variable関数は、通常、関数に渡された引数が変数かどうかを判定するために使用します。＜引数＞が変数へのポインタの場合は、“ True(真) ” を返します。＜引数＞がフィールドへのポインタの場合には、“ False(偽) ” を返します。

▼ 次の例は、配列またはフィールドの合計値を返す関数 “ SumIt ” です。引数がフィールドの場合、**Sum**関数を使用します。引数が配列の場合には、合計を計算します。**Type**関数を使用して、引数が正しいデータタイプかどうかを検査している点に注目してください：

```
$0:=0                                                    ` 合計値を初期化
If (Is a variable ($1))                                  ` 引数が変数
  ` 数値配列 ?
  If ((Type ($1->)=14) | (Type ($1->)=15) | (Type ($1->)=16))
    For ($i ; 1 ; Size of array ($1->))                  ` 配列の要素の数だけ繰り返す
      $0:=$0 + $1->{$i}                                  ` 配列の各要素を加算
    End for
  End if
Else                                                     ` 引数がフィールド
  ` 数値フィールド ?
  If ((Type ($1->)=1) | (Type ($1->)=8) | (Type ($1->)=9))
    $0:=Sum ($1->)                                       ` フィールドを合計
  End if
End if
```

数値フィールド “ [従業員]給与 ” の合計を計算する場合は、次のように記述します：

SumIt **(->[従業員]給与)**

数値配列 “ 明細合計 ” の合計を計算する場合は、次のように記述します：

SumIt **(->明細合計)**

Nil

Nil (変数) → ブール

引数	タイプ	説明
変数	変数	ニルポインタかどうかを判定する変数

Nil関数は、＜変数＞がニルポインタ(->[ ])の場合は " True(真) " を返します。この関数は、グローバル変数、ローカル変数、配列要素に対して使用できます。ただし、＜変数＞の型はポインタでなければなりません。

▼ 次の例は、**Nil**関数を使用して " $1 " が有効なポインタであるかどうかを判定します。" $1 " が有効なポインタであれば " $1 " の指す値に1を加算します：

```
If (Not (Nil ($1)))
  $1->:=$1-> + 1
End if
```

Get pointer

Get pointer (名前) → ポインタ

引数	タイプ	説明
名前	文字列	変数の名前

Get pointer関数は、引数＜名前＞で指定した変数のポインタを返します。**Get pointer**関数を実行する前に、その変数が存在している必要はありません。

▼次の例は、CB1、CB2、... 、CB20という一連の変数に数値の1を代入します：

```
For ($i ; 1 ; 20)
  $P:=Get pointer ("CB" + String ($i))      ` 変数のポインタを求める
  $P->:=1                                   ` 変数に値をセット
End for
```

Type

Type (引数) → 数値

引数	タイプ	説明
引数	フィールドまたは変数	タイプを求めるデータオブジェクトまたは変数

Type関数は、＜引数＞のタイプを返します。**Type**関数は、通常、引数が正しいタイプかどうか判定するために使用します。

4th Dimensionは、次のようなあらかじめ定義された定数を持っています：

定数	タイプ	値
Is Alpha Field	倍長整数	0
Is String Var	倍長整数	24
Is Text	倍長整数	2
Is Real	倍長整数	1
Is Integer	倍長整数	8
Is LongInt	倍長整数	9
Is Date	倍長整数	4
Is Time	倍長整数	11
Is Boolean	倍長整数	6
Is Picture	倍長整数	3
Is Subtable	倍長整数	7
Is BLOB	倍長整数	30
Is Undefined	倍長整数	5
Is Pointer	倍長整数	23
String array	倍長整数	21
Text array	倍長整数	18
Real array	倍長整数	14
Integer array	倍長整数	15
LongInt array	倍長整数	16
Date array	倍長整数	17
Boolean array	倍長整数	22
Picture array	倍長整数	19
Pointer array	倍長整数	20
Array 2D	倍長整数	13

互換性に関する注意点：バージョン3までの4Dでは、C_GRAPHコマンドを使ってタイプ宣言した任意のグラフ変数に対してType関数は 3(Is Picture)を返していました。しかし、バージョン6からは、Type関数はそのグラフ変数に 9(Is Longint）を返します。

Type関数をフィールド、インタープロセス変数、プロセス変数、ローカル変数、およびこれらのオブジェクトタイプを参照するポインタで使用することができます。

バージョン6の注意点：バージョン6から、Type関数をパラメータ（$1、$2...,${..})、プロジェクトメソッド、関数の返す値（$0)で使用することができるようになりました。

▼ **APPEND TO CLIPBOARD**コマンドの例を参照してください。

▼ **DRAG AND DROP PROPERTIES**コマンドの例を参照してください。

参照：Is a variable、Undefined

Self

Self → ポインタ

Self関数は、現在実行中のオブジェクトメソッドの属するオブジェクトのポインタを返します。**Self**関数が使用できるのは、オブジェクトメソッドだけです。オブジェクトメソッドから呼び出したプロジェクトメソッドにも使用できません。**Self**関数は、オブジェクトメソッド内で変数を参照するために使用します。

Self関数は、フォーム上の複数のオブジェクトに同じ処理を実行させる場合に便利です。オブジェクトメソッドに**Self**関数を使用したオブジェクトを1つ作成すれば、そのオブジェクトを複製し名前を付け直すだけで済みます。

▼ 次の例は、フィールドに対するオブジェクトメソッドです。これは、ユーザが作成した関数 “ Caps ” にフィールドの内容を渡し、その処理結果をフィールドに代入します。このオブジェクトを複製したり、オブジェクトメソッドを他のフィールドに複製してもオブジェクトメソッドは正しく実行されます：

```
Self->:=Caps (Self->)
```

RESOLVE POINTER

RESOLVE POINTER (ポインタ ; 変数名 ; テーブル番号 ; フィールド番号)

引数	タイプ		説明
ポインタ	ポインタ	→	**参照オブジェクトを検索するためのポインタ**
変数名	文字列	←	**参照される変数名または空の文字列**
テーブル番号	**数値**	←	**参照されるテーブルの番号、または 0 か -1**
フィールド番号	**数値**	←	**参照フィールドまたは配列要素の番号、または 0**

RESOLVE POINTERコマンドは、＜ポインタ＞式によって参照されるオブジェクトを検索し、引数＜変数名＞、＜テーブル番号＞、＜フィールド番号＞に返します。

参照されるオブジェクトの種類によって、**RESOLVE POINTER**コマンドは、次のような値を返します。

参照オブジェクト	引数（パラメータ）		
	＜変数名＞	＜テーブル名＞	＜フィールド番号＞
なし (NIL ポインタ)	**"" (空の文字列)**	**0**	**0**
変数	**変数の名前**	**-1**	**0**
配列	**配列の名前**	**-1**	**0**
配列要素	**配列の名前**	**-1**	**要素番号**
テーブル	**"" (空の文字列)**	**テーブル番号**	**0**
フィールド	**"" (空の文字列)**	**テーブル番号**	**フィールド番号**

注：ポインタに渡す値がポインタ数式でない場合には、シンタックスエラーが発生します。

例：

1. フォーム内で、v1～v100 までの入力可能な変数 100 個を作成します。
これを行うには、次のような手順を実行します。

a. 入力可能な変数を 1つ作成し、“ v ” と名付ける。

b. オブジェクトのプロパティを設定する。

c. オブジェクトに次のメソッドを付着する。

DoSomething (**Self**) `**DoSomething がデータベース内でのプロジェクトメソッドになる**

d. この時点で、必要な回数だけ変数を複製することも、「フォーム」エディタの「グリッドで変数作成」機能を使用することもできます。

e.「DoSomething」メソッド内で、このメソッドが呼び出される変数のインデックスを知る必要がある場合には、次のように書き込みます。

```
RESOLVE POINTER($1 ; $vsVarName ; $vlTableNum ; $vlFieldNum)
$vlVarNum:=Num(Substring($vsVarName ; 2))
```

フォームをこのような方法で構築すると、100 個の変数のメソッドを一度で記述することになることに注意が必要です。DoSomething (1)、DoSomething (2) から DoSomething (100) までを書き込む必要はありません。

2. デバッグのためには、メソッドへの2番目の引数 ($2) がテーブルへのポインタであることを確認する必要があります。このメソッドの最初で、次のように記述します。

```
If (<>DebugOn)
   RESOLVE POINTER($2 ; $vsVarName ; $vlTableNum ; $vlFieldNum)
   If (Not(($vlTableNum>0)&($vlFieldNum=0)))
      `警告：ポインタは、テーブルへの参照ではない
      TRACE
   End if
End if
```

3. **DRAG AND DROP PROPERTIES**コマンドの例を参照してください。

参照：DRAG AND DROP PROPERTIES、Field、Get pointer、Is a variable、Nil、Table

Command name

Command name (コマンド番号)→ 文字列

引数	タイプ		説明
コマンド番号	数値	→	コマンド番号
関数の返す値	文字列	←	ローカライズされたコマンドの名前

Command name関数は、引数<コマンド番号>に渡したコマンド番号のコマンドのローカライズされた名前を返します。

4th Dimension は、メソッドで使用するキーワード、定数、およびコマンド名をダイナミックに翻訳して統合します。例えば、4Dの英語版（コマンド名は日本語版も英語版と同じです）を使用している場合は、次のように記述します：

```
DEFAULT TABLE ([MyTable])
ALL RECORDS ([MyTable])
```

この同じコードは、4D のフランス語版でも再オープンされ、次のように読み込まれます：

```
TABLE PAR DEFAUT ([MyTable])
TOUT SELECTIONNER ([MyTable])
```

ただし、4th Dimension には、ユニークな機能である **EXECUTE**コマンドがあり、これを使用すると、データベースがコンパイルされていてもただちにコードを組み込み、このコードを実行することができます。

英語版（日本語版）で**EXECUTE** 文を使用して作成されたコードの例は、次のようになります。

```
EXECUTE ( "DEFAULT TABLE([MyTable])")
EXECUTE  ( "ALL RECORDS([MyTable])")
```

この同じコードは、4D のフランス語版でも再オープンされ、次のようになります：

```
EXECUTER ( "DEFAULT TABLE([MyTable])")
EXECUTER ( "ALL RECORDS([MyTable])")
```

4D は、**EXECUTE** (英語版／日本語版) を **EXECUTER** (フランス語版) に自動的に翻訳しますが、コマンドに渡したテキスト文を翻訳することはできません。

アプリケーションで **EXECUTE** コマンドを使用する場合には、**Command name**関数を使用すると、異なる言語間のローカライズにおける実行文の問題をこのようにして解決し、文を言語に依存しないようにすることができます。コードの例は、次のようになります：

```
EXECUTE (Command name (46)+"([MyTable])")
EXECUTE (Command name (47)+"([MyTable])")
```

4D のフランス語版を使用すると、このコードは次のようになります：

```
EXECUTER (Nom commande (46)+"([MyTable])")
EXECUTER (Nom commande (47)+"([MyTable])")
```

注：コマンド名およびコマンド番号は、後述の「コマンド名によるコマンド」 および「コマンド番号によるコマンド」の節 に一覧表示されています。

例：

1. データベースのすべてのテーブルについて、各テーブルの標準データ入力に使用するための“ 入力フォーム ”というフォームがあります。そこで、ポインタまたはテーブル名を渡すテーブルのカレント入力フォームとしてこのフォームを設定する、汎用的なプロジェクトメソッドを追加する場合には、次のように記述します：

```
` 「STANDARD INPUT FORM」プロジェクトメソッド
` STANDARD INPUT FORM (ポインタ {; 文字列 })
` STANDARD INPUT FORM (->テーブル {; テーブル名 })
C_POINTER ($1)
C_STRING (31;$2)
If (Count parameters>=2)
   EXECUTE (Command name (55)+"(["+$2+"];"入力フォーム")")
Else
   If (Count parameters>=1)
      INPUT FORM ($1-> ; "l入力フォーム")
   End if
End if
```

このプロジェクトメソッドがデータベースに追加された後、次のように記述します：

```
STANTARD INPUT FORM (->[従業員])
STANTARD INPUT FORM ("従業員")
```

注：通常、汎用ルーチンを記述する場合には、ポインタを使用することをお勧めします。その理由としては、まず、データベースがコンパイルされると、コードはコンパイルされるためです。次に、4D Insiderが、ポインタを渡すオブジェクトへの参照を検索するためです。最後に、前述の例でもあるようにテーブルの名前を変更すると、コードは正しく動作しなくなるためです。ただし、EXECUTEコマンドを使用すれば問題が解決する場合もあります。

2. フォームでは、ドロップダウンリストを一般的なサマリーレポートコマンドと共存させるには、そのドロップダウンリストのオブジェクトメソッドに、次のように記述します：

```
Case of
   \ (Form event =On Before)
      ARRAY TEXT (asCommand ; 4)
      asCommand{1}:=Command name (1) ` Sum関数
      asCommand{2}:=Command name (2) ` Average関数
      asCommand{3}:=Command name (4) ` Min関数
      asCommand{4}:=Command name (3) ` Max関数
      ` ...
End case
```

4D の英語版／日本語版では、ドロップダウンリストには、Sum、Average、Min、Max が表示されます。フランス語版では、ドロップダウンリストには、Somme、Moyenne、Min、Max が表示されます。

参照：EXECUTE

コマンド名によるコマンド

次の表は、4Dコマンドを名前ごとに一覧表示し、一緒にコマンド番号も示しています。コマンド番号は、引数<コマンド番号>で使用する必要があります。

コマンド名	コマンド番号
ABORT	156
Abs	99
ACCEPT	269
ACCUMULATE	303
ACI folder	485
Activated	346
ADD DATA SEGMENT	361
ADD RECORD	56
ADD SUBRECORD	202
Add to date	393
ADD TO SET	119
After	31
ALERT	41
ALL RECORDS	47
ALL SUBRECORDS	109
Append document	265
APPEND MENU ITEM	411
APPEND TO CLIPBOARD	403
APPEND TO LIST	376
Application file	491
Application type	494
Application version	493
APPLY TO SELECTION	70
APPLY TO SUBSELECTION	73
Arctan	20
ARRAY BOOLEAN	223
ARRAY DATE	224
ARRAY INTEGER	220
ARRAY LONGINT	221
ARRAY PICTURE	279
ARRAY POINTER	280
ARRAY REAL	219
ARRAY STRING	218
ARRAY TEXT	222
ARRAY TO LIST	287
ARRAY TO SELECTION	261
ARRAY TO STRING LIST	512

参照：Command name、EXECUTE

コマンド番号によるコマンド

次の表は、4D コマンドを番号ごとに一覧表示し、一緒にコマンドの名前も示しています。コマンド番号は、引数＜コマンド番号＞で使用する必要があります。

注：リストされていない番号は、現在使用されていません。

コマンド番号	コマンド名
1	Sum
2	Average
3	Max
4	Min
5	PRINT FORM
5	Year of
6	PAGE BREAK
7	Records in subselection
8	Int
9	Dec
10	String
11	Num
12	Substring
13	Uppercase
14	Lowercase
15	Position
16	Length
17	Sin
18	Cos
19	Tan
20	Arctan
21	Exp
22	Log
23	Day of
24	Month of
26	Std deviation
27	Variance
28	Sum squares
29	Before
30	During
31	After
32	Modified
33	Current date
34	Not
35	Old

36	End selection
37	End subselection
38	REJECT
39	PRINT LABEL
40	DIALOG
41	ALERT
42	RELATE ONE
43	SAVE RELATED ONE
44	OLD RELATED ONE
45	SAVE OLD RELATED ONE
46	DEFAULT TABLE
47	ALL RECORDS
48	QUERY BY FORMULA
49	ORDER BY
50	FIRST RECORD
51	NEXT RECORD
52	LOAD RECORD
53	SAVE RECORD
54	OUTPUT FORM
55	INPUT FORM
56	ADD RECORD
57	MODIFY RECORD
58	DELETE RECORD
59	DISPLAY SELECTION
60	PRINT SELECTION
61	FIRST SUBRECORD
62	NEXT SUBRECORD
63	EXECUTE
64	SEARCH BY INDEX
65	CREATE RELATED ONE
66	DELETE SELECTION
67	MENU BAR
68	CREATE RECORD
70	APPLY TO SELECTION
71	PRINT RECORD
72	CREATE SUBRECORD
73	APPLY TO SUBSELECTION
74	LOAD VARIABLES
75	SAVE VARIABLES
76	Records in selection
77	SET CHANNEL
78	SEND RECORD
79	RECEIVE RECORD

80	SEND VARIABLE
81	RECEIVE VARIABLE
82	Undefined
83	Records in table
84	EXPORT DIF
85	EXPORT SYLK
86	IMPORT DIF
87	IMPORT SYLK
88	MESSAGE
89	CLEAR VARIABLE
90	Char
91	Ascii
93	INVERT BACKGROUND
94	Round
95	Trunc
96	DELETE SUBRECORD
97	Subtotal
98	Mod
99	Abs
99	TRIGGER PROPERTIES 3
100	Random
101	Level
102	Date
103	SEND PACKET
104	RECEIVE PACKET
105	DISPLAY RECORD
106	PRINT SETTINGS
107	ORDER SUBRECORDS BY
108	QUERY SUBRECORDS
109	ALL SUBRECORDS
110	PREVIOUS RECORD
111	PREVIOUS SUBRECORD
112	In header
113	In break
114	Day number
116	CREATE SET
117	CLEAR SET
118	USE SET
119	ADD TO SET
120	UNION
121	INTERSECTION
122	DIFFERENCE
140	CREATE EMPTY SET

143	Semaphore
144	CLEAR SEMAPHORE
145	READ ONLY
146	READ WRITE
147	Locked
148	GRAPH TABLE
149	ENABLE MENU ITEM
150	DISABLE MENU ITEM
151	BEEP
152	Menu selected
153	Open window
154	CLOSE WINDOW
155	ON ERR CALL
156	ABORT
157	TRACE
158	NO TRACE
159	DELETE DOCUMENT
160	ERASE WINDOW
161	GOTO XY
162	CONFIRM
163	Request
164	FONT
165	FONT SIZE
166	FONT STYLE
167	EXPORT TEXT
168	IMPORT TEXT
169	GRAPH
170	SORT BY INDEX
171	ON SERIAL PORT CALL
172	RECEIVE BUFFER
174	REDRAW
175	MESSAGES OFF
176	PUSH RECORD
177	POP RECORD
178	Current time
179	Time
180	Time string
181	MESSAGES ON
182	Current user
184	SAVE SET
185	LOAD SET
186	CHANGE PASSWORD
187	Screen width

188	Screen height
189	ONE RECORD SELECT
190	ON EVENT CALL
191	In footer
192	ENABLE BUTTON
193	DISABLE BUTTON
194	BUTTON TEXT
195	Records in set
197	REPORT
198	Before selection
199	Before subselection
200	LAST RECORD
201	LAST SUBRECORD
202	ADD SUBRECORD
203	MODIFY SUBRECORD
204	MODIFY SELECTION
205	USE ASCII MAP
206	GOTO AREA
207	QUERY SELECTION BY FORMULA
208	SET MENU ITEM MARK
209	GET HIGHLIGHT
210	HIGHLIGHT TEXT
212	UNLOAD RECORD
213	SET WINDOW TITLE
214	True
215	False
218	ARRAY STRING
219	ARRAY REAL
220	ARRAY INTEGER
221	ARRAY LONGINT
222	ARRAY TEXT
223	ARRAY BOOLEAN
224	ARRAY DATE
225	DUPLICATE RECORD
226	COPY ARRAY
227	INSERT ELEMENT
228	DELETE ELEMENT
229	SORT ARRAY
230	Find in array
231	Insert string
232	Delete string
233	Replace string
234	Change string

235	SET FILTER
236	SET FORMAT
237	SET CHOICE LIST
238	SET ENTERABLE
239	START TRANSACTION
240	VALIDATE TRANSACTION
241	CANCEL TRANSACTION
242	GOTO RECORD
243	Record number
244	Sequence number
245	GOTO SELECTED RECORD
246	Selected record number
247	GOTO PAGE
248	NEXT PAGE
249	PREVIOUS PAGE
250	FIRST PAGE
251	LAST PAGE
252	Table
253	Field
254	Count tables
255	Count fields
256	Table name
257	Field name
258	GET FIELD PROPERTIES
259	Count parameters
260	SELECTION TO ARRAY
261	ARRAY TO SELECTION
262	RELATE MANY
263	OLD RELATED MANY
264	Open document
265	Append document
266	Create document
267	CLOSE DOCUMENT
268	SET TIMEOUT
269	ACCEPT
270	CANCEL
271	SET COLOR
273	Is in set
274	Size of array
275	Printing page
276	Current form page
277	QUERY
278	Last object

279	ARRAY PICTURE
280	ARRAY POINTER
281	EDIT ACCESS
282	C_INTEGER
283	C_LONGINT
284	C_TEXT
285	C_REAL
286	C_PICTURE
287	ARRAY TO LIST
288	LIST TO ARRAY
289	CHANGE ACCESS
290	PLAY
291	QUIT 4D
292	QUERY BY EXAMPLE
293	C_STRING
294	Is a variable
295	Type
297	FLUSH BUFFERS
298	GRAPH SETTINGS
299	PAGE SETUP
300	ORDER BY FORMULA
301	C_POINTER
302	BREAK LEVEL
303	ACCUMULATE
304	Get pointer
305	C_BOOLEAN
306	C_TIME
307	C_DATE
308	Self
309	Open external window
310	AUTOMATIC RELATIONS
311	IDLE
314	Modified record
315	Nil
316	SET ABOUT
317	New process
319	PAUSE PROCESS
320	RESUME PROCESS
321	FILTER EVENT
322	Current process
323	DELAY PROCESS
324	HIDE PROCESS
325	SHOW PROCESS

326	BRING TO FRONT
327	Frontmost process
328	Outside call
329	CALL PROCESS
330	Process state
331	COPY NAMED SELECTION
332	USE NAMED SELECTION
333	CLEAR NAMED SELECTION
334	CUT NAMED SELECTION
335	Count tasks
336	PROCESS PROPERTIES
338	User in group
339	DISTINCT VALUES
340	RELATE MANY SELECTION
341	QUERY SELECTION
342	Count users
343	Count user processes
344	SET INDEX
345	SELECT LOG FILE
346	Activated
347	Deactivated
348	SET MENU ITEM
349	RELATE ONE SELECTION
350	SCAN INDEX
351	REDUCE SELECTION
352	C_GRAPH
353	LOCKED ATTRIBUTES
355	COMPRESS PICTURE
356	Picture size
357	LOAD COMPRESS PICTURE FROM FILE
358	SAVE PICTURE TO FILE
359	COMPRESS PICTURE FILE
361	ADD DATA SEGMENT
362	Read only state
363	Current default table
364	SET PRINT PREVIEW
365	PLATFORM PROPERTIES
366	MAP FILE TYPES
367	SET PLATFORM INTERFACE
368	SUBSELECTION TO ARRAY
369	Database event
370	SET PROCESS VARIABLE
371	GET PROCESS VARIABLE

372	Process number
373	Execute on server
375	New list
376	APPEND TO LIST
377	CLEAR LIST
378	GET LIST ITEM
379	Selected list item
380	Count list items
381	SELECT LIST ITEM
382	REDRAW LIST
383	Load list
384	SAVE LIST
385	SET LIST ITEM
386	SET LIST ITEM PROPERTIES
387	SET LIST PROPERTIES
388	Form event
389	FILTER KEYSTROKE
390	Keystroke
391	SORT LIST
392	SET DEFAULT CENTURY
393	Add to date
394	RESOLVE POINTER
395	SET QUERY LIMIT
396	SET QUERY DESTINATION
397	In transaction
398	Trigger level
400	Test clipboard
401	GET CLIPBOARD
402	CLEAR CLIPBOARD
403	APPEND TO CLIPBOARD
404	Count menus
405	Count menu items
411	APPEND MENU ITEM
412	INSERT MENU ITEM
413	DELETE MENU ITEM
422	Get menu item
423	SET MENU ITEM KEY
424	Get menu item key
425	SET MENU ITEM STYLE
426	Get menu item style
428	Get menu item mark
430	Get menu title
431	SHOW MENU BAR

432	HIDE MENU BAR
433	SHOW TOOLBAR
434	HIDE TOOLBAR
435	SHOW WINDOW
436	HIDE WINDOW
437	Count screens
438	SCREEN COORDINATES
439	SCREEN DEPTH
440	Menu bar height
441	Menu bar screen
442	WINDOW LIST
443	GET WINDOW RECT
444	SET WINDOW RECT
445	Window kind
446	Window process
447	Frontmost window
448	Next window
449	Find window
450	Get window title
452	DRAG WINDOW
453	MAXIMIZE WINDOW
454	MINIMIZE WINDOW
456	REDRAW WINDOW
457	PICTURE PROPERTIES
458	Tickcount
459	Milliseconds
460	FONT LIST
461	Font number
462	Font name
463	Mac to Win
464	Win to Mac
465	POST KEY
466	POST CLICK
467	POST EVENT
468	GET MOUSE
469	SET CURSOR
470	Get platform interface
471	VOLUME LIST
472	VOLUME ATTRIBUTES
473	FOLDER LIST
474	DOCUMENT LIST
475	CREATE DIRECTORY
476	Test path name

477	GET DOCUMENT PROPERTIES
478	SET DOCUMENT PROPERTIES
479	Get document size
480	SET DOCUMENT SIZE
481	Get document position
482	SET DOCUMENT POSITION
483	Current machine
484	Current machine owner
485	ACI folder
486	Temporary folder
487	System folder
488	Gestalt
489	Structure file
490	Data file
491	Application file
492	Compiled application
493	Application version
494	Application type
495	Version type
496	Create resource file
497	Open resource file
498	CLOSE RESOURCE FILE
499	RESOURCE TYPE LIST
500	RESOURCE LIST
501	DELETE RESOURCE
502	GET PICTURE RESOURCE
503	SET PICTURE RESOURCE
504	Get text resource
505	SET TEXT RESOURCE
506	Get string resource
507	SET STRING RESOURCE
508	GET RESOURCE
509	SET RESOURCE
510	Get indexed string
511	STRING LIST TO ARRAY
512	ARRAY TO STRING LIST
513	Get resource name
514	SET RESOURCE NAME
515	Get resource properties
516	SET RESOURCE PROPERTIES
517	GET ICON RESOURCE
519	Mac to ISO
520	ISO to Mac

521	SET PICTURE TO CLIPBOARD
522	GET PICTURE FROM CLIPBOARD
523	SET TEXT TO CLIPBOARD
524	Get text from clipboard
525	DOCUMENT TO BLOB
526	BLOB TO DOCUMENT
527	DATA SEGMENT LIST
528	Document type
529	Document creator
530	SET DOCUMENT TYPE
531	SET DOCUMENT CREATOR
532	VARIABLE TO BLOB
533	BLOB TO VARIABLE
534	COMPRESS BLOB
535	EXPAND BLOB
536	BLOB PROPERTIES
537	SET SCREEN DEPTH
538	Command name
539	Square root
540	MOVE DOCUMENT
541	COPY DOCUMENT
542	Pop up menu
543	Shift down
544	Macintosh control down
545	Macintosh option down
546	Macintosh command down
547	Caps lock down
548	INTEGER TO BLOB
549	BLOB to integer
550	LONGINT TO BLOB
551	BLOB to longint
552	REAL TO BLOB
553	BLOB to real
554	TEXT TO BLOB
555	BLOB to text
556	LIST TO BLOB
557	BLOB to list
558	COPY BLOB
559	INSERT IN BLOB
560	DELETE FROM BLOB
561	REMOVE FROM SET
562	Windows Ctrl down
563	Windows Alt down

600	COPY SET
601	SET TABLE TITLES
602	SET FIELD TITLES
603	SET VISIBLE
604	C_BLOB
605	BLOB size
606	SET BLOB SIZE
607	DRAG AND DROP PROPERTIES
608	Drag and drop position
609	GET USER LIST
610	GET GROUP LIST
611	GET USER PROPERTIES
612	SET USER PROPERTIES
613	GET GROUP PROPERTIES
614	SET GROUP PROPERTIES
615	DELETE USER
616	Is user deleted
617	START WEB SERVER
618	STOP WEB SERVER
619	SEND HTML FILE
620	SET WEB DISPLAY LIMIT
621	Is a list
622	SET WEB TIMEOUT
623	SET REAL COMPARISON LEVEL
624	DELETE LIST ITEM
625	INSERT LIST ITEM
626	Copy list
627	Current form table
628	SET RGB COLOR
629	List item position
630	SELECT LIST ITEM BY REFERENCE
631	GET LIST ITEM PROPERTIES
632	GET LIST PROPERTIES
633	List item parent
634	SET HTML ROOT

参照：Command name、EXECUTE

算術関数　31

この章では、「ルーチン」エディタの「Math」テーマ内にある算術関数について説明します。この章では、標準的な算術演算を行なう関数について説明します。これらの関数は、数値を返します。

Abs	**Int**	**SET REAL COMPARISON LEVEL**
Arctan	**Log**	**Square root**
Cos	**Mod**	**Sin**
Dec	**Random**	**Tan**
Exp	**Round**	**Trunc**

Abs

Abs (数値) → 数値

引数	タイプ	説明
数値	数値	絶対値をとる数値

Abs関数は、＜数値＞の絶対値(符号なしの正の値)を返します。

＜数値＞が負の場合、正の数を返します。＜数値＞が正の数の場合は、値は変わりません。

▼ 次の例は、-10.3の正の値である10.3を返します：

```
vベクトル:=Abs (-10.3)                    ` 変数 “ vベクトル ” に10.3を代入する
```

Dec

Dec (数値) → 数値

引数	タイプ	説明
数値	数値	小数点以下の数をとる数値

Dec関数は、＜数値＞の小数部(端数)を返します。返す値は、常に正の数またはゼロになります。

▼ 次の例は、キログラム単位で実数になっている重量を、整数のキログラムとグラムに変換します。重量が7.311の場合は、7キログラムと311グラムなります：

```
キログラム:=Int (重量)                    ` キログラムの部分7を得る
グラム:=Dec (重量)                        ` グラムの部分0.311を得る
```

Exp

Exp (数値) → 数値

引数	タイプ	説明
数値	数値	評価する数値

Exp関数は、自然対数の底(e=2.71828182845904524)の＜数値＞乗の値を計算します。**Exp**関数は、**Log**関数の逆の関数です。

▼ 次の例は、eの２乗を変数“ｖ”に代入します：

```
v:=Exp (2)
```

Int

Int (数値) → 数値

引数	タイプ	説明
数値	数値	整数部を取り出す数値

Int関数は、＜数値＞の整数部を取り出して返します。＜数値＞が負の場合は、ゼロから遠い数値に丸めます。

▼ 次の例は、負の数値と正の数値に対して **Int**関数がどのように機能するかを示しています：

```
x:=Int (123.4)      ` xに123を代入
y:=Int (-123.4)     ` yに-124を代入
```

Log

Log (数値) → 数値

引数	タイプ	説明
数値	数値	自然対数を取る数値

Log関数は、＜数値＞の自然対数を返します。**Log**関数は、**Exp**関数の逆の演算を行う関数です。

次の数式を使って、ある数値の対数を変換します：

LogX(n) = LogY(Y) / LogE(X)

“ Y ”が“ e ”と等しいければ、

LogX(n) = LogE(n) * LogE(e) / LogE(X)

になりますので、“ LogE(e) ”は１と等しくなり、“ Exp(1) ”は“ e ”と等しいので、この数式はさらに簡単にすることができます。つまり、このようになります：

LogX(n) = LogE(n) / LogE(X)

式“ LogE(n) / LobE(10) ”は“ Log10(n) ”の値を返します。“ 1/LogE(10) ”の値は、0.434294481903251828...と等しいので、“ Log10(n) ”は“ LogE(n) ”にこの定数を掛けたものなります。同様に、“ LogE(n) ”は、次の数式で求められます：

LogE(n) = Log10(n) * 2.3025850092794045

メソッドで定数を使用するか、あるいは式を使用するかは、式“ Log(10) ”を計算する時間を節約したいか否かに依ります。

Random

Random → 数値

Random関数は、0から32,767までの範囲の整数(乱数)を返します。

整数の範囲は、次のように記述して変えることができます：

Random % (最大 - 最小 + 1) + 最小

最小は範囲の最小値で、最大は最大値です。

▼ 次の例は、10から30までの範囲の整数の乱数を変数“ v結果 ”に代入します：

v結果:=Random % 21 + 10

Round

Round (数値 ; 桁位置) → 数値

引数	タイプ	説明
数値	数値	丸める数値
桁位置	数値	丸める桁

関数は、＜数値＞を指定された＜桁位置＞で四捨五入します。

＜桁位置＞が正の数の場合は、＜数値＞の小数部を丸め、＜桁位置＞が負の場合には、整数部を丸めます。

＜桁位置＞で指定した桁位置の数字を四捨五入します。＜桁位置＞が0の場合は、小数第１位を、-1の場合には、整数部の１の位を四捨五入します。

▼ 下記は、さまざまな引数を使用して**Round**関数の機能を示します。結果を変数 “ v結果 ” に代入します。コメントは、変数 “ v結果 ” に代入される値についての説明です：

```
v結果:=Round (16.857 ; 2)          ` v結果に16.86を代入
v結果:=Round (32345.67 ; -3)       ` v結果に32000を代入
v結果:=Round (29.8725 ; 3)         ` v結果に29.873を代入
v結果:=Round (-1.5 ; 0)            ` v結果に-2を代入
```

Trunc

Trunc (数値 ; 桁位置) → 数値

引数	タイプ	説明
数値	数値	切り捨てる数値
桁位置	数値	切り捨てる桁

Trunc関数は、指定された＜桁位置＞の小数部を切り捨てた＜数値＞を返します。**Trunc**関数は、常に元の値よりも小さい値を返します。

＜桁位置＞が正の数の場合は、＜数値＞の小数部を切り捨て、＜桁位置＞が負の場合には、整数部を切り捨てます。

▼ さまざまな引数を使用した**Trunc**関数の機能を次に示します。結果を変数 " v結果 " に代入します。コメントは、変数 " v結果 " に代入される値についての説明です：

```
v結果:=Trunc (216.897 ; 1)      ` v結果に216.8を代入
v結果:=Trunc (216.897 ; -1)     ` v結果に210を代入
v結果:=Trunc (-216.897 ; 1)     ` v結果に-216.9を代入
v結果:=Trunc (-216.897 ; -1)    ` v結果に-220を代入
```

Mod

Mod (数値1 ; 数値2) → 数値

引数	タイプ	説明
数値1	数値	割り算される数値
数値2	数値	割り算する数値

Mod関数は、＜数値2＞で＜数値1＞を割り算し、その余りの整数を返します。＜数値1＞または＜数値2＞が実数の場合は、**Mod**関数はまずその値を丸め、計算を実行します。
また、余りを計算するためにモジューロ演算子(%)を使用することができます。

▼ 次の例は、**Mod**関数が異なる引数でどのように機能するかを紹介しています。各行は変数 " v結果 " に値を代入し、コメント行にその結果を記述しています：

```
v結果:=Mod (3 ; 2)      ` 変数 " v結果 " は１を得る
v結果:=Mod (4 ; 2)      ` 変数 " v結果 " は0を得る
v結果:=Mod (3.5 ; 2)    ` 変数 " v結果 " は0を得る
```

三角関数

この節では、三角関数について説明します。これらの関数は、単位にラジアンを使用して計算します。1度を0.0174532925199432958として扱います。

Arctan

Arctan (数値) → 数値

引数	タイプ	説明
数値	数値	ラジアンで返すタンジェントの値

Arctan関数は、＜数値＞のアークタンジェント(Arctangent、逆正接)をラジアンで返します。＜数値＞はタンジェントです。

▼ 次の例は、変数 “ y ” にπの値を代入します：

```
y:=Arctan (1)
```

Cos

Cos (数値) → 数値

引数	タイプ	説明
数値	数値	コサイン(cosine)で返す値(単位...ラジアン)

Cos関数は、＜数値＞のコサイン(Cosine、余弦)を返します。＜数値＞はラジアンで指定します。

Sin

Sin (数値) → 数値

引数	タイプ	説明
数値	数値	サイン(sine)で返す値(単位...ラジアン)

Sin関数は、＜数値＞のサイン(Sine、正弦)を返します。＜数値＞はラジアンで指定します。

Tan

Tan (数値) → 数値

引数	タイプ	説明
数値	数値	タンジェント(Tangent)で返す値(単位...ラジアン)

Tan関数は、＜数値＞のタンジェント(Tangent、正接)を返します。＜数値＞はラジアンで指定します。

Square root

Square root (数値) → 数値

引数	タイプ		説明
数値	数値	→	平方根を求める数値
関数の返す値	数値	←	平方根の値

Square root関数は、＜数値＞の平方根を返します。

▼ 次の行は、値 1.414213562373 を「$SquareRoot」変数に割り当てます：

```
$SquareRoot:=Square root (2)
```

▼ 次のメソッドは右側の三角形の斜辺を返します。この三角形の 2つの斜辺以外の辺は引数として渡されます：

```
` 「Hypotenuse」メソッド
` Hypotenuse (実数 ; 実数) ->実数
` Hypotenuse (辺A ; 辺B) -> 斜辺
C_REAL($0 ; $1 ; $2)
$0 := Square root(($1^2)+($2^2))
```

例えば、斜辺 (4;3) は 5 を返します。

SET REAL COMPARISON LEVEL

SET REAL COMPARISON LEVEL (イプシロン値)

引数	タイプ		説明
イプシロン値	数値	→	実数の同等性の比較に使用するイプシロン値

SET REAL COMPARISON LEVELコマンドは、4th Dimensionが実数と数式の同等性を比較するために使用するイプシロン値を設定します。

コンピュータは常に実数の計算を概算で実行するため、実数の同等性をテストする時には、この概算があることを考慮する必要があります。4th Dimensionは、実数を比較する時に2つの実数の差が一定の値より大きいかどうかをテストすることによって、概算を確認します。この値はイプシロン値と呼ばれ、次のような機能があります。

2つの実数aとbがある時、**Abs(a-b)**がイプシロン値より大きい場合には、これら2つの実数は等しくないとみなされます。それ以外の場合には、これらの実数は等しいとみなされます。

デフォルトでは、4th Dimensionはイプシロン値を10の2乗から6を減じた値(10^-6)に設定しています。

例：

- 0.00001=0.00002は、Falseを戻します。誤差0.00001が10^-6より大きいためです。
- 0.000001=0.000002は、Trueを戻します。誤差0.000001が10^-6より大きくないためです。
- 0.000001=0.000003は、Falseを戻します。誤差0.000002が10^-6より大きいためです。

SET REAL COMPARISON LEVELコマンドを使用すると、必要に応じてイプシロン値を増やしたり減らすことができます。

警告：通常、デフォルトのイプシロン値を変更するためにこのコマンドを使用する必要はありません。

重要：イプシロン値を変更しても、実数の同等性の比較に影響があるだけで、他の実数計算や実数値の表示には影響はありません。

実数の表示について

注意：

複数のプラットフォームによる開発に関わっていない場合には、この項は参照しなくても構いません。

コンピュータでは、浮動小数点計算は、数学的知識ではなく技法の1つにすぎません。例えば、3分の1(1/3)は、小数点の後に数字3が無限に続く形で表記できると学習したことでしょう。一方、コンピュータはこのことを認識しないため、式を計算しなければなりません。同様に、3×1/3が1に等しいことは概念的に理解していますが、コンピュータはその解を得るために式を計算します。使用するコンピュータの種類によっては、1/3は小数点の後に数字の3が限られた桁数だけ続く有限小数ととして計算されます。この桁数は、そのコンピュータの精度と呼ばれます。

68KのMacintoshでは、精度数は19で、1/3は有効数字19桁で計算されます。WindowsやPower Macintoshでは、この数は15になり、1/3は有効数字15桁で表示されます。式1/3を4th Dimensionの「デバッグ」ウィンドウで表示すると、68KのMacintoshでは0.3333333333333333333と表示され、WindowsやPower Macintoshでは 0.3333333333333333148のように表示されます。WindowsおよびPower Macintoshでの精度数は 68KのMacintoshでの精度数より小さいため、最後の3桁が異なっていることに注意が必要です。それでも、(1/3)*3と表示すると、いずれのコンピュータでも解は1になります。

浮動小数点計算で、例えば、裏庭の広さを平方フィートを算出する場合には、小数点以下の桁数などは関係ないため、無視できることでしょう。一方、IRS(内国歳入庁:日本の国税庁に相当)の申告書を作成している場合には、時にはコンピュータが小数点以下も正確に計算しているかどうかが気になるところでしょう。それでも、小数点以下19桁または15桁まで計算していれば、何十億ドルもの所得があったとしても、十分に正確な結果が得られます。

68KのMacintoshとWindowsやPower Macintoshでは、1/3の値が異なるのはなぜでしょうか?

68KのMacintoshでは、オペレーティングシステム（OS）は実数を10バイト(80ビット)で格納しているのに対し、WindowsやPower Macintoshでは、実数を8バイト(64ビット)で格納しています。この結果、68KのMacintoshでは実数の有効数字が19桁で、WindowsやPower Macintoshではそれが15桁になります。

それでも、式(1/3)*3によりいずれのコンピュータでも1が戻るのはなぜでしょうか?

コンピュータは近似的に計算を実行できるだけにすぎません。したがって、数字を比較したり計算する時には、コンピュータは実数を数学的なオブジェクトとしてではなく、近似値として処理します。この例では、0.3333...を3倍すると0.9999...になります。0.9999...と1の差は非常に小さいため、コンピュータはこの結果を1と等しいとみなし、1

を戻します。この問題については、**SET REAL COMPARISON LEVEL**コマンドについての説明を参照してください。

実数には2つの側面があるため、次の2つの点について区別しておかなければなりません。

- 実数がどのように計算され、比較されるか
- 実数が画面やプリンタにどのように出力されるか

もともと4th Dimensionは、68KのMacintoshで提供されていた標準の10バイトデータ型を使用して実数を処理していました。その結果、ディスクのデータファイルに格納される実数は、この形式を使用して保存されます。68KのMacintosh、Windows、Power Macintoshのそれぞれの4th Dimensionのバージョン間で互換性を保つために、4th Dimensionのデータファイルでは引き続き10バイトデータ型を使用して実数を保存しています。浮動小数点計算はWindowsまたはPower Macintoshで8バイト形式を使用して実行されるため、4th Dimensionは値を10バイト形式から8バイト形式に変換したり、8バイト形式から10バイト形式に変換しています。このため、68KのMacintoshで保存された実数が含まれるレコードをWindowsやPower Macintoshにロードする場合には、精度の一部が失われる(有効数字が19桁から15桁になる)可能性があります。一方、WindowsやPower Macintoshで保存された実数が含まれるレコードを68KのMacintoshにロードした場合には、精度が失われることはありません。基本的には、68KのMacintoshのデータベースを使用した場合でも、WindowsやPower Macintoshのデータベースを使用した場合でも、浮動小数点計算では、有効数字19桁ではなく15桁までを信頼するようにします

Customizer Plusユーティリティを使用すると、68KのMacintoshまたはWindowsやPower Macintoshで実数の表示を簡略化する場合には、省略する桁数を設定することができます。デフォルトの設定では、68KのMacintoshでは省略する桁数はゼロ、WindowsやPower Macintoshでは 5桁です。

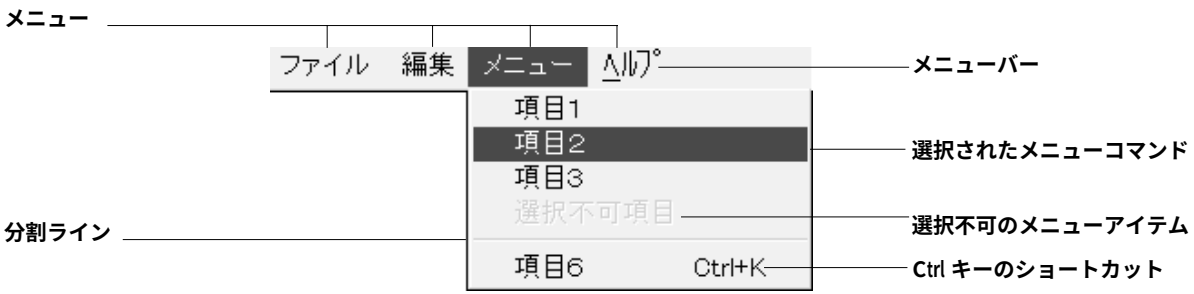
メニューコマンド　32

この章のコマンドは、メニュバーの切り替え、メニューコマンドでのテキストの変更、メニューコマンドのチェック、選択可と選択不可の切り替えを行なうためのものです。

APPEND MENU ITEM
Count menus
Count menu items
DELETE MENU ITEM
DISABLE MENU ITEM
ENABLE MENU ITEM
Get menu item
Get menu item key
Get menu item mark
Get menu item style
Get menu title
HIDE MENU BAR
INSERT MENU ITEM
MENU BAR
Menu selected
SET ABOUT
SET MENU ITEM
SET MENU ITEM KEY
SET MENU ITEM MARK
SET MENU ITEM STYLE
SHOW MENU BAR

メニューの要素

画面の上部にあるバーは、“ メニューバー ” と呼ばれます。バー上の名前は、各メニューを表します。メニューをプルダウンすると、“ メニューコマンド ” が表示されます。「デザイン」モードの「メニュー」エディタでメニューバーを作成します。メニューバーは、名前ではなく番号で識別されます。最初のメニューバーは、メニューバー#1です。これは、デフォルトのメニューバーでもあります。アプリケーションを開いた時点でメニューバー#1でないメニューバーを表示したい場合には、On Startupデータベースメソッドで**MENU BAR**コマンドを使用して強制的にメニューバーを切り替る必要があります。



各メニューコマンドには、それに付着する1つのプロジェクトメソッドがあります。このメソッドを“ マスターメソッド ” と呼びます。「メニュー」エディタウインドウの「メソッド名」エリアにメソッドの名前を入力することによって、メソッドをメニューコマンドに割り当てます。ユーザは、実行したいメソッドが属すメニューコマンドを選択することによりメソッドを起動することができます。

任意のメニューコマンドにメソッドを割り当てていない場合には、4th Dimensionはメニューシステム(カスタムモード)を抜けて、「ユーザ」モードに戻ります。

メニューコマンドにプロジェクトメソッドを割り当てなくても、メニューシステムを作成することができます。つまり、メソッドがまったく存在しない状態でも「メニュー」エディタでメニューバーを作成することができます。ただし、各メニューコマンドに目的のメソッドを割り当てるまでは、意図した処理を実行することはできません。

すべてのメニューバーは、前もって「ファイル」メニュー、「編集」メニューの2つのメニューを持っています。

■「ファイル」メニューには、「終了」メニューコマンドしかありません。「終了」アイテムにメソッドが割り当てられていないことに注目してください。つまり、これが4th Dimensionメニューシステム(「カスタム」モード)を終了させる方法です。「ファイル」メニューは、名前を変えたり、メニューコマンドを追加することもできます。「ファイル」メニューの名前を変更すると、「編集」メニューの左に表示されなくなります。メニューコマンド「終了」は、「ファイル」メニューの一番最後のメニューコマンドにすることをお勧めします。

■「編集」メニューには、標準的な編集コマンドがあります。この「編集」メニューは「メニュー」エディタに表示されない上、修正することもできません。

■ Windows版では、上記の他に「ヘルプ」メニューがあります。この「ヘルプ」メニューの中に「4th Dimension® について...」メニューコマンドとそのアプリケーションで利用できる「Windowsヘルプ」ファイルが含まれています。尚、「ヘルプ} メニューは「メニュー」エディタに表示されない上、修正することもできません。

■ Macintosh版では、上記の他に「アップル」メニューがあります。この「アップル」メニューの中に「4th Dimension® について...」メニューコマンドが含まれています。尚、「アップル」メニューは「メニュー」エディタに表示されない上、修正することもできません。

メニュー番号とメニューコマンド番号

メニューバーと同様に、メニューにも番号が付けられています。ただし、「編集」メニューは変更できないので番号は付いていません。従って、「ファイル」メニューを1として左から右に(2、3、4˜という具合に)番号が付けられています。メニューがフォームメニュー番号に付着すると変更されます。最初に追加されたメニューは、2049番で始まります。追加されたメニューを参照するには、通常のメニュー番号に 2048を追加します。

各メニュー内のメニューコマンドは、項目の1番上から1番下まで連続番号が付けられています。1番上のメニューコマンドが 1です。

カスタムメニュー

カスタムメニューを使用すれば、まるで専用に設計したかのようにアプリケーションを作成することができます。4th Dimensionは強力なメニュー構築ツールを備えています。メニューを作成すれば、ユーザはマウスを使用しないでCtrl（Macintosh版では、コマンド）キーの組み合わせでメニューを選択することができます。また、パスワードで保護されたメニューコマンドを作成し、フォーム、選択可、選択不可をメソッドで設定することもできます。

メニューバーには、“ スプラッシュスクリーン ” メニューバーと “ フォーム ” メニューバーを用いることができます。

スプラッシュスクリーンメニューバーは、「カスタム」モードでスプラッシュスクリーンが表示されている場合に使用可能です。このメニューバーのメニューからメニューコマンドを選択することによって、メニューコマンドに割り当てられたメソッドが実行されます。

フォームメニューバーは、「フォーム」エディタの「フォーム」メニューの「メニューバー連結」を使用してフォームに付着します。フォームメニューバーのメニューは、フォームが表示された時点でカレントメニューに付加され、フォームメニューバーのメニューコマンドに割り当てられたメソッドが実行できるようになります。入力フォームについては、「ユーザ」モードと「カスタム」モードの両方でカスタムメニューが追加されます。出力フォームについては、「カスタム」モードでのみカスタムメニューが追加されます。

フォームメニューは、メニューバー番号で指定します。スプラッシュスクリーンメニューバーと同じ番号を指定した場合には、メニューは追加されません。

フォームメニューバーに対して負の数を指定した場合には、4th Dimensionはその絶対値を使用します。例えば、-3を指定した場合には、メニューバー#3を指定したものとみなします。ただし、フォームメニューバーを負の数で指定した場合には、メニューバー(スプラッシュスクリーンメニューバーにフォームメニューバーのメニューが追加されたメニューバー)のすべてのメニューコマンドは、ユーザの選択により割り当てられたメソッドを実行することができます。この方法は、「カスタム」モードのメニューバーを使用するユーザにとって望ましい環境といえるでしょう。

フォームメニューバーに対して正の数を指定した場合には、メニューバーからメニューコマンドを選択しても割り当てられたメソッドを実行することはできません。その代わりに、On Menu Selectedイベントがフォームメソッドに送られます。

連結メニュー

メニューは、メニューバーに連結することができます。連結されたメニューコマンドがチェックされると、選択可または選択不可になり、メニューのその他のすべてがこれらの変更を反映します。連結メニューに関する詳細は、『4th Dimension / 4D First デザインリファレンス』を参照してください。

MENU BAR

MENU BAR (メニューバー番号 {; プロセス} {; *})

引数	タイプ	説明
メニューバー番号	数値	メニューバーの番号
プロセス	数値	プロセス参照番号
*		メニューバーの状態保存

MENU BARコマンドは、カレントメニューバーを＜メニューバー番号＞で指定したメニューバー変更します。

オプション引数＜プロセス＞は、指定したプロセスのメニューバーを＜メニューバー＞変更します。すべてのメニューコマンドは、「メニュー」エディタで定義した状態(選択可または選択不可)に戻ります。すべてのメニューコマンドは、チェックマークなしで表示します。

オプション引数＜*＞により、メニューバーの現在の状態を保存することができます。この引数が省略された場合、このコマンドが実行されると、**MENU BAR**コマンドはメニューバーを元の状態に戻します。

例えば、**MENU BAR (1)** を実行したとします。次に、**DISABLE MENU ITEM**コマンドを使い、任意のアイテムを使用不可にします。

MENU BAR (1)を 2度目に実行すると、メニューコマンドはすべて、元の使用可の状態になっています。

MENU BAR (1 ; *) を実行すると、メニューバーは前と同じ状態を保っており、使用不可にしたメニューコマンドは使用不可のままになっています。

注：省略可能な引数を指定しない場合、* は 2番目の引数になります。つまり MENU BAR (1 ; 2 ; *) と MENU BAR (1 ; *) は両方とも有効な命令文です。

ユーザが「カスタム」モードに入ると、最初のメニューバー(メニューバー#1)が開かれます。データベースが開かれるときに起動されるOn Startupデータベースメソッドでメニューバーを変更することができます。

▼ 次の例は、カレントメニューバーをメニューバー#3に変更し、メニューの状態を元に戻します：

```
MENU BAR (3)
```

▼ 次の例は、カレントメニューバーをメニューバー#3に変更し、メニューの状態を保存します：前に使用不可にされたメニューコマンドが、使用不可で現れます。

```
MENU BAR (3 ; *)
```

▼ 次の例は、レコードの変更中にフォームのメニューバーをメニューバー#3に変更します。レコードの変更が済むと、メニューの状態を保存してメニューバーをメニューバー#2に戻します：

```
MENU BAR (3)                ` 次のフォームにメニューバー#3を設定する
ALL RECORDS (顧客)
MODIFY SELECTION (顧客)     ` フォームを開く
MENU BAR (2 *)              ` 変更後メニューバーを戻す
```

DISABLE MENU ITEM
ENABLE MENU ITEM

DISABLE MENU ITEM (メニュー番号 ; メニューコマンド番号)
ENABLE MENU ITEM (メニュー番号 ; メニューコマンド番号)

引数	タイプ	説明
メニュー番号	数値	メニューの番号
メニューコマンド番号	数値	メニューコマンドの番号

DISABLE MENU ITEMコマンドは、＜メニュー番号＞と＜メニューコマンド番号＞で指定したメニューコマンドを選択不可の状態(淡色表示)にします。＜メニューコマンド番号＞に 0を指定した場合には、そのメニューを選択不可の状態にします。

ENABLE MENU ITEMコマンドは、＜メニュー番号＞と＜メニューコマンド番号＞で指定したメニューコマンドを選択可の状態にします。＜メニューコマンド番号＞に 0を指定した場合には、そのメニューを選択可の状態にします。

メニューコマンドは、**MENU BAR**コマンドでメニューバーを切り替えるまで選択可または選択不可の状態を保ちます。

同じアイテムを頻繁に選択不可にする場合は、「メニュー」エディタで選択不可に設定しておくことをお勧めします。

「ファイル」メニューがメニュー番号1になります。対応するメニューバーのメニュー番号は2049から始まります。「編集」メニュー（Macintosh版の場合は「「アップル」メニューも）は、あらかじめ組み込まれており、メニューの一部としてカウントされません。

▼ 次の例は、2番目のメニュー(「編集」メニューではない)の4番目のメニューコマンドにレコードを削除するメソッドが割り当てられていると想定しています。このメニューコマンドは、削除すべきレコードが存在するとき以外は、選択不可の状態にする必要があります。この例は、メニューコマンドをに選択可と選択不可の状態に切り替えます：

```
If (Records in selection ([従業員]) # 0)   ` 従業員がいる？
  ENABLE MENU ITEM (2 ; 4)                 ` レコード削除のメニューコマンドを選択可にする
Else                                       ` 従業員が1人もいない？
  DISABLE MENU ITEM (2 ; 4)                ` レコード削除のメニューコマンドを選択不可にする
End if
```

HIDE MENU BAR

HIDE MENU BAR → 数値

引数	タイプ	説明
このコマンドには、引数はありません。		

HIDE MENU BARコマンドは、メニューバーを隠します。

すでにメニューバーが隠れている場合は、このコマンドは何も行いません。

▼ 次のメソッドは、マウスボタンをクリックするまでフルスクリーン表示（Macintosh）でレコードを表示します：

```
HIDE MENU BAR
Open window (-1;-1;1+Screen width ; 1+Screen height ; ダイアログ)
INPUT FORM([描画];"フルスクリーン")
DISPLAY RECORD([描画])
Repeat
   GET MOUSE($vlX ; $vlY ; $vlButton)
Until ($vlButton#0)
CLOSE WINDOW
SHOW MENU BAR
```

注：Windows上では、ウインドウはアプリケーションウインドウの範囲内に制限されます。

参照；SHOW MENU BAR

SHOW MENU BAR

SHOW MENU BAR → 数値

引数	タイプ	説明
このコマンドには、引数はありません。		

SHOW MENU BARコマンドは、メニューバーを隠します。

すでにメニューバーが表示さている場合は、このコマンドは何も行いません。

▼ **HIDE MENU BAR**コマンドの例を参照してください。

参照；HIDE MENU BAR

SET MENU ITEM

SET MENU ITEM (メニュー番号 ; メニューコマンド番号 ; メニューテキスト ; {プロセス})

引数	タイプ	説明
メニュー番号	数値	メニュー番号
メニューコマンド番号	数値	メニューコマンド番号
メニューテキスト	文字列	メニューコマンドの新テキスト
プロセス	数値	プロセスID

SET MENU ITEMコマンドは、＜メニュー番号＞の＜メニューコマンド番号＞のテキストを設定します。＜メニュー番号＞がメニュー接続の場合、＜メニュー番号＞のすべてのインスタンスの＜メニューコマンド番号＞のテキストを＜メニューテキスト＞に変更します。

オプション引数＜プロセス＞は、任意のメニューコマンドを異なるプロセス内のメニューコマンドで用いるために使用されます。しかし、そのコマンドに及ぼす影響がこのパラメータによって指定されたプロセスに限定されるわけではないことに注意してください。つまり、それが使用されるプロセスのすべてでメニューコマンドが変更されます。

対応するメニューバーのメニュー番号は、2049番から始まります。

▼ 次の例では、最初に追加されたメニューの3番目のメニューコマンドにメニューコマンドを設定します。ブール変数 “ Bool_Flag ” が “ True ” の場合は、メニューは “ 顧客ソート ” に変わります。“ Bool_Flag ” が “ False ” の場合には、メニューは “ 送り状ソート ” に変わります：

```
If (Bool_Flag)
   SET MENU ITEM (2049 ; 3 ; "顧客ソート" ; <>Process1)
Else
   SET MENU ITEM (2049 ; 3 ; "送り状ソート" ; <>Process1)
End if
```

▼ 次の例では、カレントファイルが “ 顧客 ” の場合に、最前面にあるプロセスの最初のメニューの5番目のメニューコマンドを変更します：

```
If (FilePointer = (->[顧客]))
   SET MENU ITEM (1 ; 5 ; "顧客印刷" ; Frontmost process)
Else
   SET MENU ITEM (1 ; 5 ; "送り状印刷" ; Frontmost process)
End if
```

Menu selected

Menu selected → 数値

Menu selected関数は、入力フォームまたは出力フォームが表示さている状態でのみ使用可能です。このコマンドは、メニューから選択されたメニューコマンドを検出します。

特別な場合以外は、**Menu selected**関数を使用しないで、メニューバーのメニューコマンドに割り当てられたメソッドを使用する方が良いでしょう。選択されたメニューコマンドに割り当てられたメソッドが直接実行されますので、メニューコマンドを検出する必要はありません。

Menu selected関数は、選択されたメニューとメニューコマンドの番号を倍長整数型で返します。選択されたメニュー番号を知るためには、返された数値を65,536で割り、整数型に変換します。選択されたメニューコマンド番号を知るには、返された数値を65,536で割り、その余りを求めます：

```
メニュー:=Menu selected // 65536
メニューコマンド:=Menu selected % 65536
```

メニューコマンドが選択されていない場合には、 0を返します。

「ファイル」メニューがメニュー番号1になります。対応するメニューバーのメニュー番号は2049から始まります。「編集」メニューと「ヘルプ」メニュー（Macintosh版の場合は「アップル」メニュー）は、あらかじめ組み込まれており、メニューの一部としてカウントされません。

▼ 次の例は、**SET MENU ITEM MARK**コマンドの引数を求めるために**Menu selected**関数を使用しています：

```
Case of
  \ (During)
  ` Duringフェーズ
  \ (Menu selected # 0)
    SET MENU ITEM MARK (Menu selected // 65536 ; Menu selected 65536 ; Char(18))
End case
```

SET ABOUT

SET ABOUT (アイテム ; メソッド)

引数	タイプ	説明
アイテム	文字列	メニュー上の文字列
メソッド	文字列	呼び出すメソッド

SET ABOUTコマンドは、「ヘルプ」(Macintosh版では、「アップル」)メニューの「4th Dimension®について...」の<アイテム>を置き換えます。**SET ABOUT**コマンドを実行すると、「ヘルプ (アップル)」メニューにアイテムが表示されます。ユーザがこのアイテムを選択すると、<メソッド>が実行されます。<メソッド>はダイアログボックスを開き、データベースに関するバージョンやその他の情報を表示することができます。ダイアログの上部は固定表示エリアとなり、4th Dimensionアイコン、4th Dimensionバージョン番号、4D Compilerバージョン番号、著作権情報および1本の線が表示されます。

▼ 次の例は、「4th Dimension®について」メニューコマンドを「スケジュールについて...」に置き換えます。メソッド“ ABOUT ”は、カスタムのアバウトボックスを表示します：

```
SET ABOUT ("スケジュールについて..." ; "ABOUT")
```

▼ 次の例は、「4th Dimension®について」メニューコマンドを元のアバウトボックスに戻します：

```
SET ABOUT ("4th Dimension®について..." ; "")
```

Count menus

Count menus ({プロセス}) → 数値

引数	タイプ		説明
プロセス	数値	→	プロセス参照番号
関数の返す値	数値	←	カレントメニューバー上のメニューの数

Count menus関数は、カレントメニューバー上にあるメニューの数を返します。

オプション引数＜プロセス＞を省略すると、**Count menus**関数はカレントプロセスのメニューバーに適用します。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューバーに適用します。

参照：Count menu item

Count menu items

Count menu items (メニュー {; プロセス}) → 数値

引数	タイプ		説明
メニュー	数値	→	メニュー番号
プロセス	数値	→	プロセス参照番号
関数の返す値	数値	←	メニュー内のメニューコマンドの数

Count menu items関数は、引数＜メニュー＞に渡されたメニュー番号を持つメニュー内にあるメニューコマンドの数を返します。

オプション引数＜プロセス＞を省略すると、**Count menu items**関数はカレントプロセスのメニューに適用します。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューに適用します。

参照：Count menu item

Get menu title

Get menu title (メニュー {; プロセス}) → 数値

引数	タイプ		説明
メニュー	数値	→	メニュー番号
プロセス	数値	→	プロセス参照番号
関数の返す値	数値	←	メニューのタイトル

Get menu title関数は、引数＜メニュー＞に渡されたメニュー番号を持つメニューのタイトルを返します。

オプション引数＜プロセス＞を省略すると、**Get menu title**関数はカレントプロセスのメニューに適用します。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューに適用します。

参照：Count menu

Get menu item

Get menu item (メニュー ; メニューコマンド {; プロセス}) → 文字列

引数	タイプ		説明
メニュー	数値	→	メニュー番号
メニューコマンド	数値	→	メニューコマンド番号
プロセス	数値	→	プロセス参照番号
関数の返す値	文字列	←	メニューコマンドのテキスト

Get menu item関数は、引数＜メニュー＞と＜メニューコマンド＞に渡されたメニュー番号およびメニューコマンド番号を持つメニューコマンドのテキストを返します。

オプション引数＜プロセス＞を省略すると、**Get menu item**関数はカレントプロセスのメニューに適用します。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューに適用します。

参照：SET MENU ITEM

Get menu item style

Get menu item style (メニュー ; メニューコマンド {; プロセス}) → 数値

引数	タイプ		説明
メニュー	数値	→	メニュー番号
メニューコマンド	数値	→	メニューコマンド番号
プロセス	数値	→	プロセス参照番号
関数の返す値	数値	←	メニューコマンドのフォントスタイル

Get menu item style関数は、引数＜メニュー＞と＜メニューコマンド＞に渡されたメニュー番号およびメニューコマンド番号を持つメニューコマンドのフォントスタイルを返します。

オプション引数＜プロセス＞を省略すると、**Get menu item style**関数はカレントプロセスのメニューに適用します。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューに適用します。

Get menu item style関数は、次のようなあらかじめ定義された定数の組み合わせを加算した数値を返します。

定数	タイプ	値
Plain	倍長整数	0
Bold	倍長整数	1
Italic	倍長整数	2
Underline	倍長整数	4
Outline	倍長整数	8
Shadow	倍長整数	16
Condensed	倍長整数	32
Extended	倍長整数	64

注；Windows版では、“ Plain ”、“ Bold ”、“ Italic ”、“ Underline ” しか利用できません。

▼ 次の例は、任意のメニューコマンドが太字（ボールド）で表示されるかどうかをテストします：

```
If ((Get menu item style($vlMenu ; $vlItem) & Bold)#0)
   `...
End if
```

参照：SET MENU ITEM STYLE

SET MENU ITEM STYLE

SET MENU ITEM STYLE (メニュー ; メニューコマンド ; スタイル{; プロセス})

引数	タイプ		説明
メニュー	数値	→	メニュー番号
メニューコマンド	数値	→	メニューコマンド番号
スタイル	数値	→	新しいメニューコマンドのフォントスタイル
プロセス	数値	→	プロセス参照番号

SET MENU ITEM STYLEコマンドは、引数＜メニュー＞と＜メニューコマンド＞に渡されたメニュー番号およびメニューコマンド番号を持つメニューコマンドのフォントスタイルを変更します。

オプション引数＜プロセス＞を省略すると、**SET MENU ITEM STYLE**コマンドはカレントプロセスのメニューに適用します。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューに適用します。

SET MENU ITEM STYLEコマンドは、次のようなあらかじめ定義された定数の組み合わせを加算した数値を返します。

定数	タイプ	値
Plain	倍長整数	0
Bold	倍長整数	1
Italic	倍長整数	2
Underline	倍長整数	4
Outline	倍長整数	8
Shadow	倍長整数	16
Condensed	倍長整数	32
Extended	倍長整数	64

注；Windows版では、“ Plain ”、“ Bold ”、“ Italic ”、“ Underline ” しか利用できません。

参照：Get menu item style

Get menu item mark

Get menu item mark (メニュー ; メニューコマンド {; プロセス}) → 文字列

引数	タイプ		説明
メニュー	数値	→	メニュー番号
メニューコマンド	数値	→	メニューコマンド番号
プロセス	数値	→	プロセス参照番号
関数の返す値	文字列	←	メニューコマンドのマーク

Get menu item mark関数は、引数＜メニュー＞と＜メニューコマンド＞に渡されたメニュー番号およびメニューコマンド番号を持つメニューコマンドのチェックマークを返します。

オプション引数＜プロセス＞を省略すると、**Get menu item mark**関数はカレントプロセスのメニューに適用します。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューに適用します。

メニューコマンドにマークが付いていない場合、**Get menu item mark**関数は空の文字列を返します。

注：SET MENU ITEM MARKコマンド内のMacintoshおよびWindows上のマークの説明参照してください。

▼次の例は、チェックコマンドのチェックマークを切り替えます：

```
SET ITEM MARK($vlMenu ; $vlItem ; Char(18)*Num (Get menu item mark($vlMenu ; $vlItem)=""))
```

参照：SET MENU ITEM MARK

SET MENU ITEM MARK

SET MENU ITEM MARK (メニュー ; メニューコマンド ; マーク {; プロセス})

引数	タイプ		説明
メニュー	数値	→	メニュー番号
メニューコマンド	数値	→	メニューコマンド番号
マーク	数値	→	新しいメニューコマンドのマーク
プロセス	数値	→	プロセス参照番号

SET MENU ITEM MARKコマンドは、引数＜メニュー＞と＜メニューコマンド＞に渡されたメニュー番号およびメニューコマンド番号を持つメニューコマンドのチェックマークを変更します。

オプション引数＜プロセス＞を省略すると、**SET MENU ITEM MARK**コマンドはカレントプロセスのメニューに適用します。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューに適用します。

空の文字列を渡すと、任意のマークはメニューコマンドから取り除かれます。空の文字列でない場合は、

■ Macintosh上では、文字列の最初の文字がメニューコマンドのマークになります。通常、**Char**(18)を受け渡します。これは、Macintoshメニューのチェックマーク文字です。

■ Windows上では、メニューコマンドは標準のチェックマークが割り当てられます。

▼ **Get item mark**関数の例を参照してください。

参照：Get menu item mark

Get menu item key

Get menu item key (メニュー ; メニューコマンド {; プロセス}) → 数値

引数	タイプ		説明
メニュー	数値	→	メニュー番号
メニューコマンド	数値	→	メニューコマンド番号
プロセス	数値	→	プロセス参照番号
関数の返す値	数値	←	メニューコマンドのショートカットキーのASCIIコード

Get menu item key関数は、引数＜メニュー＞と＜メニューコマンド＞に渡されたメニュー番号およびメニューコマンド番号を持つメニューコマンドのキーボード（Windows上ではCtrl、Macintosh上ではコマンド）ショートカットキーのASCIIコードを返します。

オプション引数＜プロセス＞を省略すると、**Get menu item mark**関数はカレントプロセスのメニューに適用します。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューに適用します。

メニューコマンドにショートカットキーがない場合、**Get menu item key**関数は 0を返します。

参照：SET MENU ITEM KEY

SET MENU ITEM KEY

SET MENU ITEM KEY (メニュー ; メニューコマンド ; キー {; プロセス})

引数	タイプ		説明
メニュー	数値	→	メニュー番号
メニューコマンド	数値	→	メニューコマンド番号
キー	数値	→	新しいメニューコマンドのショートカットキー
プロセス	数値	→	プロセス参照番号

Get menu item key関数は、引数＜メニュー＞と＜メニューコマンド＞に渡されたメニュー番号およびメニューコマンド番号を持つメニューコマンドのキーボード（Windows上ではCtrl、Macintosh上ではコマンド）ショートカットキーを引数＜キー＞に渡されたASCIIコードに変更します。

オプション引数＜プロセス＞を省略すると、**Get menu item mark**関数はカレントプロセスのメニューに適用します。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューに適用します。

引数＜キー＞に0を渡すと、任意のショートカットキーはメニューコマンドから取り除かれます。

注：一貫性のあるユーザインタフェースでは、Ctrlキー（Windows）またはコマンドキー（Macintosh）以外のモデファイアキーを使用せずに、キーボード上で利用できる数字や記号に大文字を使用します。

参照：Get menu item key

APPEND MENU ITEM

APPEND MENU ITEM (メニュー ; テキスト {; プロセス})

引数	タイプ		説明
メニュー	数値	→	メニュー番号
テキスト	文字列	→	新しいメニューコマンドのテキスト
プロセス	数値	→	プロセス参照番号

APPEND MENU ITEMコマンドは、引数＜メニュー＞に渡されたメニュー番号を持つメニューに新規メニューコマンドを追加します。

オプション引数＜プロセス＞を省略すると、**APPEND MENU ITEM**コマンドはカレントプロセスのメニューに適用します。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューに適用します。

APPEND MENU ITEMコマンドは、1回の呼び出しで1つまたは複数のメニューコマンドを追加することができます。

追加されるメニューコマンドを次のようにして引数＜テキスト＞で定義します：

■ セミコロン (;) で各メニューコマンドを区切る。例えば、

“ 項目テキスト1 ; 項目テキスト2 ; 項目テキスト3 ”

■ メニューコマンドを使用不可にする場合は、引数＜テキスト＞の中に開き丸カッコ「 (」を配置する。

■ メニューコマンド分割ラインを指定する場合は、引数＜テキスト＞に「(-」を渡す。

■ 行にフォントスタイルを指定する場合は、引数＜テキスト＞の中に小なり記号 (<)の後に続く下記の文字の1つを配置する。

<B　太字（ボールド）
<I　斜体（イタリック）
<U　下線（アンダーライン）
<O　アウトライン（Macintoshのみ）
<S　シャドウ（Macintoshのみ）

■ 任意のメニューコマンドにチェックマークを追加するには、引数＜テキスト＞の中にのクエスチョンマーク（ ? ）の後に続くチェックマークとして使用する文字を配置する。Macintosh上では、その配置された文字が表示されます。Windows上では、どんな文字が渡されようともチェックマークが表示されます。

■ 任意のメニューコマンドにアイコンを追加するには、引数＜テキスト＞の中にカレット (^) の後にASCIIコードから48を引いたMacOSベースのリソースIDの文字を配置する。

■ 任意のメニューコマンドにキーボードショートカットを追加するには、引数＜テキスト＞の中にカレット (/) の後にショートカット用の文字を配置する。

注：程よいメニューコマンドの数を持ったメニューを使用してください。もし、50以上のメニューコマンドを表示したい場合は、メニューの代わりにスクロールエリアの使用を検討する必要がありかもしれません。

注：ITEM MENU ITEMコマンドは255バイトまでですが、APPEND MENU ITEMコマンドは最大32000バイトまで使用することができます。

重要：新しいメニューコマンドは、割り当てられたメソッドは持っていません。そのため、新しいメニューコマンドはMenu selected関数を使ってフォームメソッドの中から管理する必要があります。

▼ 次の例は、「フォント」メニューに利用可能なフォントの名前を追加します。この例では、カレントメニューバーに 6つのメニューがあります：

```
` 「On Startup」データベースメソッド内
` フォントリストがロードされ、メニューコマンドが作成される
FONT LIST(<>asAvailableFont)
<>atFontMenuItems:=""
For ($vlFont ; 1 ; Size of array(<>asAvailableFont))
   <>atFontMenuItems:=<>atFontMenuItems+";"+<>asAvailableFont{$vlFont}
End for
```

そして、任意のフォームメソッドまたはプロジェクトメソッド内に次のコードを記述します。

```
APPEND MENU ITEM(6 ; <>atFontMenuItems)
```

参照：DELETE MENU ITEM、INSERT MENU ITEM

INSERT MENU ITEM

INSERT MENU ITEM (メニュー ; メニューコマンド ; テキスト {; プロセス})

引数	タイプ		説明
メニュー	数値	→	メニュー番号
メニューコマンド	数値	→	メニューコマンド番号
テキスト	文字列	→	挿入されるメニューコマンドのテキスト
プロセス	数値	→	プロセス参照番号

INSERT MENU ITEMコマンドは、引数＜メニュー＞に渡されたメニュー番号を持つメニューの中の引数＜メニューコマンド＞に渡された番号を持つメニューコマンドの前に新しいメニューコマンドを挿入します。

オプション引数＜プロセス＞を省略すると、**INSERT MENU ITEM**コマンドはカレントプロセスのメニューに適用します。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューに適用します。

INSERT MENU ITEMコマンドは、1回の呼び出しで1つまたは複数のメニューコマンドを挿入することができます。

INSERT MENU ITEMコマンドは、下記の相違点を除いて、**APPEND MENU ITEM**コマンドと同じように機能します。

■ **INSERT MENU ITEM**コマンドはメニュー内のどんな場所にでもメニューコマンドを挿入できるのに対して、**APPEND MENU ITEM**コマンドは常にメニューの最後にしかメニューコマンドを追加することができない。

■ **INSERT MENU ITEM**コマンドは引数＜テキスト＞には最大255バイトまでしか使用できないのに対して、**APPEND MENU ITEM**コマンドは＜テキスト＞には最大32000バイトまで使用できる。

引数＜テキスト＞に渡すメニューコマンドの定義情報に関する詳細は、**APPEND MENU ITEM**コマンドの節を参照してください。

重要：新しいメニューコマンドは、割り当てられたメソッドは持っていません。そのため、新しいメニューコマンドはMenu selected関数を使ってフォームメソッドの中から管理する必要があります。

参照：APPEND MENU ITEM

DELETE MENU ITEM

DELETE MENU ITEM (メニュー ; メニューコマンド {; プロセス})

引数	タイプ		説明
メニュー	数値	→	メニュー番号
メニューコマンド	数値	→	メニューコマンド番号
プロセス	数値	→	プロセス参照番号

DELETE MENU ITEMコマンドは、引数＜メニュー＞と＜メニューコマンド＞に渡されたメニュー番号およびメニューコマンド番号を持つメニューコマンドを削除します。

オプション引数＜プロセス＞を省略すると、**DELETE MENU ITEM**コマンドはカレントプロセスのメニューに適用します。＜プロセス＞を指定した場合は、＜プロセス＞に渡されたプロセス参照番号のプロセスのメニューに適用します。

注：一貫性のあるユーザインタフェースでは、メニューコマンドのないメニューを保持してはいけません。

参照：APPEND MENU ITEM、INSERT MENU ITEM

メッセージコマンド　33

この章では、「ルーチン」エディタの「Messages」テーマ内にあるメッセージコマンドについて説明します。この章のコマンドは、ユーザに対してメッセージを表示するものです。メッセージには、標準ダイアログボックスとメッセージウインドウ、およびカスタムのメッセージウインドウとインジケータなども含みます。

ダイアログボックスは形式的なもので、ユーザがボタンをクリックするか、または“ Enter ”キーを押すことによって処理を続行します。

ALERTコマンドは、ユーザに情報を伝える場合にのみ使用します。

CONFIRMコマンドは、ユーザに情報を伝え、動作を行うべきかどうかの指示を得る場合に使用します。

Request関数は、ユーザからのテキスト情報を得る場合に使用します。

ALERT
CONFIRM
GOTO XY
MESSAGE
MESSAGES OFF
MESSAGES ON
Request

ALERT

ALERT (メッセージ {; OKボタンタイトル})

引数	タイプ		説明
メッセージ	文字列	→	「注意」ダイアログボックスに表示するメッセージ
OKボタンタイトル	文字列	→	OKボタンのタイトル

ALERTコマンドは、注意アイコンとメッセージ、OKボタンで構成される警告ダイアログボックスを表示します。

引数＜メッセージ＞をアラートボックス上に表示します。このメッセージは最大255バイトまで表示可能ですが、表示されるメッセージがメッセージエリアに表示しきれない場合は、文字の幅にもよりますが途中で切り取られます。

デフォルトでは、OKボタンのタイトル（ラベル）は“ OK ”です。このOKボタンタイトルを変更するには、オプション引数＜OKボタンタイトル＞の中に任意のボタンタイトルを指定します。

▼ 次の例は、会社に関する情報を示すアラートボックスを表示します。表示する文字列中にキャリッジリターンが含まれてることに注目してください。これは、キャリッジリターンから後ろの文字列を次の行に改行するためです：

```
CR:=Char (13)
ALERT ("会社：" + [会社]名称 + CR +
  "会社の従業員数：" + String (Records in selection ([従業員])) + CR +
  "供給する商品の数：" + String (Records in selection ([部品])))
```

次の図は上記の**ALERT**コマンドで表示される「注意} ダイアログボックスを示しています。

注：このALERTコマンドのステートメント行がマニュアルの関係上、3行に渡って記述されていますが、4th Dimensionの「メソッド」エディタでは、ステートメントはすべて1行で記述する必要があります。

▼ 次の例は、OKボタンのタイトルを任意のボタンタイトルに変更したものを示しています：

ALERT ("このレコードを削除するアクセス権はあなたにはありません。" ; "ごめんなさい")

参照：CONFIRM、Request

CONFIRM

CONFIRM (メッセージ {; OKボタンタイトル{; キャンセルボタンタイトル}})

引数	タイプ		説明
メッセージ	文字列	→	「確認」ダイアログボックスに表示するメッセージ
OKボタンタイトル	文字列	→	「OK」ボタンのタイトル
キャンセルボタンタイトル	文字列	→	「キャンセル」ボタンのタイトル

CONFIRMココマンドは、確認アイコンとメッセージ、OKボタンで構成される「確認」ダイアログボックスを表示します。

引数＜メッセージ＞をアラートボックス上に表示します。このメッセージは最大255バイトまで表示可能ですが、表示されるメッセージがメッセージエリアに表示しきれない場合は、文字の幅にもよりますが途中で切り取られます。

デフォルトでは、OKボタンのタイトル（ラベル）は“ OK ”、「キャンセル」ボタンのタイトル（ラベル）は“ キャンセル ”です。この2つのボタンタイトルを変更するには、オプション引数＜OKボタンタイトル＞と＜キャンセルボタンタイトル＞の中に任意のボタンタイトルを指定します。

「OK」ボタンは、デフォルトボタンです。ユーザが「OK」ボタンをクリックするか、または“ Enter ”キーを押すとシステム変数OKに1が代入されます。ユーザが「キャンセル」ボタンをクリックするとシステム変数OKに 0が代入されます。

▼ 次の例は、レコード削除処理の確認をユーザに要請するための「確認」ダイアログボックスを表示します。システム変数OKの状態を示すために **If**文と**ALERT**コマンドを使用します：

```
CONFIRM ("警告！：この処理は元に戻すことはできません。")
If (OK=1)
   ALL RECORD ([元従業員])
   DELETE SELECTION ([元従業員)
Else
  ALERT ("処理が取り消されました。")
End if
```

次の図は、この**CONFIRM**コマンドで表示される「確認」ダイアログボックスを示しています。

▼ 次の例は、引数オプションの＜OKボタンタイトル＞と＜キャンセルボタンタイトル＞に任意のボタンラベルを指定しています：

```
CONFIRM("警告：この処理を続行したら、レコードを元に戻すことができません。"
                    + Char(13) + "どうしますか？" ; "続行しない" ; "続行")
```

次の図は、この**CONFIRM**コマンドで表示される「確認」ダイアログボックスを示しています。

参照：ALERT、 Request

Request

Request (メッセージ {; デフォルト応答 {; OKボタンタイトル{; キャンセルボタンタイトル}}}) → 文字列

引数	タイプ		説明
メッセージ	文字列	→	「リクエスト」ダイアログボックスに表示するメッセージ
デフォルト応答	文字列	→	テキスト入力エリアに表示するデータ
OKボタンタイトル	文字列	→	「OK」ボタンのタイトル
キャンセルボタンタイトル	文字列	→	「キャンセル」ボタンのタイトル
関数の返す値	文字列	←	ユーザによって入力される値

Request関数は、メッセージ、テキスト入力エリア、OKボタンで構成される「リクエスト」ダイアログボックスを表示します。

引数＜メッセージ＞をアラートボックス上に表示します。このメッセージは最大255バイトまで表示可能ですが、表示されるメッセージがメッセージエリアに表示しきれない場合は、文字の幅にもよりますが途中で切り取られます。

デフォルトでは、OKボタンのタイトル（ラベル）は“OK”、「キャンセル」ボタンのタイトル（ラベル）は“キャンセル”です。この2つのボタンタイトルを変更するには、オプション引数＜OKボタンタイトル＞と＜キャンセルボタンタイトル＞の中に任意のボタンタイトルを指定します。

「OK」ボタンは、デフォルトボタンです。ユーザが「OK」ボタンをクリックするか、または“Enter”キーを押すとシステム変数OKに1が代入されます。ユーザが「キャンセル」ボタンをクリックするとシステム変数OKに 0が代入されます。

ユーザは、テキスト入力エリアに文字列データを入力することができます。ユーザが「OK」ボタンをクリックするか、または“Enter”キーを押すと**Request**関数はその文字列を返します。ユーザが「キャンセル」ボタンをクリックすると**Request**関数は空の文字列(ヌル)を返します。返される値が数値または日付のいずれかでなければならない場合は、**Request**関数が返した文字列に対して**Num**関数や **Date**関数を使用して正しいデータタイプに変換する必要があります。

ユーザから複数の情報を得なければならない場合には、「リクエスト」ダイアログボックスを何度も表示するのではなく、フォームを作成し、**DIALOG**コマンドを使用してそれを表示するべきでしょう。

▼次の例は、一般的な「リクエスト」ダイアログボックスを表示します。ユーザが入力した情報は、変数 “ vリターン ” に格納されます：

vリターン:=Request ("名前を入力してください")

▼ 次の例は、引数オプションの＜OKボタンタイトル＞と＜キャンセルボタンタイトル＞に任意のボタンラベルを指定して「リクエスト」ダイアログボックスを表示します：

```
vsPrompt:=Request ("従業員の名前：" ; "" ; "レコード作成" ; "キャンセル")
If (OK=1)
   ADD RECORD ([従業員])
   `変数「vsPrompt」は、フォームメソッド内のOn Loadイベント中に
   [従業員]名字フィールドにコピーされます。
End if
```

▼ 次の例は、デフォルト値として今日の日付を入力し、その日付文字列を日付タイプに変換して「リクエスト」ダイアログボックスを表示します：

$vdPrompt := Date (Request ("検索日付を入力してください：" ; String (Current date)))

参照：ALERT、CONFIRM

MESSAGE

MESSAGE (メッセージ)

引数	タイプ	説明
メッセージ	文字列	表示するメッセージ

MESSAGEコマンドは、特殊なウインドウ上(デフォルトのメッセージウインドウ上)に<メッセージ>を表示します。このメッセージは一時的なもので、レイアウトを表示したりプロシージャの実行が終了するとすぐに消去されます。別の**MESSAGE**コマンドを実行すると古いメッセージは、消去されます。

MESSAGEコマンドは、動作状態をユーザに伝えるのが一般的な使用法です。

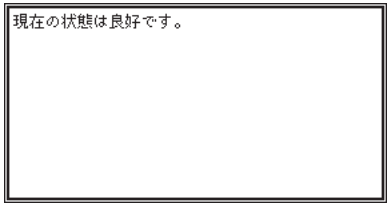
Open window関数でウインドウを開いている場合は、そのウインドウはメッセージを表示するための終点となります。全角文字のみを使用した場合には、すべての文字の幅が一定になりますのでウインドウの配置が正確になります。

Open window関数は開いたウインドウ上では、前のメッセージを消さないで連続的にメッセージを表示します。その代わりに、新しいメッセージは現在表示中のメッセージの後ろに来ます。メッセージがウインドウの幅よりも長い場合、4th Dimensionは自動的に改行を行います。メッセージ中にキャリッジリターンを挿入して、行の制御を行うこともできます。この場合のキャリッジリターン記号には、**Char**(13)を使用します。ウインドウの行を越えてメッセージを表示した場合は、4th Dimensionは自動的にメッセージをスクロールします。

ウインドウ上でメッセージの位置決めをするために、**ERASE WINDOW**コマンドと**GOTO XY**コマンドを使用することができます。新しいメッセージの文字は、同じ位置に既に表示されていた文字を上書きしたり消去します。ウインドウ上では、カーソルを表示しません。

▼次の例は、デフォルトのメッセージウインドウ上にメッセージを表示します：

MESSAGE ("現在の状態は良好です。")



MESSAGES ON
MESSAGES OFF

MESSAGES ON
MESSAGES OFF

MESSAGES ONコマンドと**MESSAGES OFF**コマンドは、時間のかかる処理を行っている際に表示するインジケータを表示または非表示にします。デフォルト状態では、表示されます。インジケータを表示する「ユーザ」モードのメニューコマンドを次の表に示します。

処理	処理	処理
フォーミュラで更新	クイックレポート	並び替え
データ書き出し	データ読み込み	チャート
フォームで検索	フォーミュラで検索	検索

インジケータを表示するコマンドを次の表に示します。

コマンド	コマンド	コマンド
APPLY TO SELECTION	IMPORT SYLK	QUERY BY FORMULA
DISTINCT VALUES	IMPORT TEXT	QUERY BY EXAMPLE
EXPORT DIF	ORDER BY	SCAN INDEX
EXPORT SYLK	ORDER BY FORMULA	
EXPORT TEXT	REPORT	
GRAPH FILE	QUERY	
IMPORT DIF	QUERY SELECTION BY FORMULA	

▼ 次の例は、「並び替え...」コマンドを実行する前にインジケータを非表示にし、「並び替え...」コマンドの実行が完了した時点で表示に戻します：

```
MESSAGES OFF
ORDER BY ([住所] ; [住所]郵便番号 ; > ; [住所]名前2 ; >)
MESSAGES ON
```

GOTO XY

GOTO XY (x ; y)

引数	タイプ	説明
x	数値	カーソルの x (水平)位置
y	数値	カーソルの y (垂直)位置

GOTO XYコマンドは、**Open window**関数で開いたウインドウ上のカーソル (見えないカーソル) の位置を指定します。**GOTO XY**コマンドは、フォームを表示していない場合にのみ機能します。

ウインドウの左上隅の位置が0,0 です。カーソルは、ウインドウを開いたときと、**ERASE WINDOW**コマンドを実行した後には、自動的に0.0 に置かれます。

Open window関数でカーソルの位置を指定してから、**MESSAGE**コマンドで文字を表示することができます。**Open window**関数、**GOTO XY**コマンド、**MESSAGE**コマンドを使用して、文字端末をエミュレートすることもできます。**Open window**関数はテキストを表示するためのウインドウを開きますので、**GOTO XY**コマンドはテキストを表示する位置を指定するために、そして**MESSAGE**コマンドはカーソル位置にデータを表示するために使用します。

GOTO XYコマンドは、フォントを“ システムフォント ”で表示します。全角文字のみを使用した場合には、すべての文字の幅が一定になりますのでウインドウ上で正確に配置することができます。

命名セレクションコマンド　34

この章では、「ルーチン」エディタの「Named selections」テーマ内にある命名セレクションコマンドについて説明します。命名セレクションを使用することによってカレントでなくなったセレクションを簡単に扱うことができます。命名セレクションはプロセスのテーブルの順序付けされたリストです。このレコードの順序リストは名前を付けてメモリに残すことができます。命名セレクションによって簡単にセレクションをメモリに置くことができます。セレクションの順序とセレクションのカレントレコードも保存されます。

次のコマンドを命名セレクションとして使用することができます。

COPY NAMED SELECTION　　**CUT NAMED SELECTION**
CLEAR NAMED SELECTION　　**USE NAMED SELECTION**

命名セレクションは、**COPY NAMED SELECTION**コマンドまたは**CUT NAMED SELECTION**コマンドで作成することができます。一般的に命名セレクションは、1つ以上のセレクションで作業、保存し、後で順序付けしたセレクションを復元するために使用します。プロセス内の各テーブルに対して複数の命名セレクションを持つことができます。カレントセレクションとして命名セレクションを再利用するには、**USE NAMED SELECTION**コマンドを呼び出します。命名セレクションでの作業が終了したら、**CLEAR NAMED SELECTION**コマンドを呼び出します。

命名セレクションには、使用する範囲の違いによるプロセス命名セレクションとインタープロセス命名セレクションの2種類があります。

先頭が“ <> ”文字で始まる名前の命名セレクションがインタープロセス命名セレクションでクライアント上の別プロセスで利用することができます。これに対し、先頭が“ <> ”文字以外の名前で始まるのがプロセス命名セレクションで、それが作成されたプロセス内でしか利用することはできません。

インタープロセス命名セレクションの利用範囲はインタープロセス変数の範囲とまったく同じです。そのため、インタープロセス命名セレクションは任意のプロセスからアクセスすることができます。

4D Clientと4D Serverを使用してる場合、インタープロセス命名セレクションはそれを作成したクライアントのプロセスでしか利用することはできません。つまり、別のクライアントマシンでそれを利用することはできません。

警告：命名セレクションを作成するには、テーブルのセレクションへのアクセスが必要となります。セレクションはサーバ上に保持され、またローカルプロセスはサーバデータへのアクセス権を持っていないので、ローカルプロセス内で命名セレクションを使用することはできません。

命名セレクションとセット

セットと命名セレクションとの違いは、次の通りです。

■ 命名セレクションがレコードの順序付けされたリストであるのに対して、セットは順序付けされていません。

■ セットは、テーブル内の各レコードに対して1ビットしか必要としませんのでメモリの面から見ると効率的です。命名セレクションは、セレクション内の各レコードに対して4バイトを必要とします。

■ セットと違って命名セレクションは、ディスクへ保存することができません。

■ セットには交差、結合、差異といった標準演算があるのに対して、命名セレクションには演算はありません。

命名セレクションとセットには次のような類似点があります。

■ セットと同じように命名セレクションもメモリ内に存在します。

■ 命名セレクションもセットもレコードの参照を格納します。レコードを変更または修正すると、命名セレクションやセットは無効になることがあります。

■ セットと同じように命名セレクションは、その命名セレクションが作成された時点のカレントレコードを“記憶”します。

セットに関する詳細は、第48章の「セットコマンド」を参照してください。

命名セレクションの例

次の例では、データベース内で“名前”フィールドに重複した値があるかどうか調べています。これは“名前”フィールドのスクリプトです。まず、フィールド内の値を変数に割り当てています。そして、そのレコードとセレクションを保存します。同じ値を持つレコードが見つかると、そのセレクションとレコードを取り出します。値を無効にし、ユーザに新しい値を入れるように要求します。

```
` 値を一時変数に保存
$名前:=[従業員]名前
` レコードをスタックにプッシュする
PUSH RECORD ([従業員])
` ソートしたセレクションを保存
CUT NAMED SELECTION ([従業員] ; "ユーザソート")
` フィールドの値を検索する
QUERY ([従業員];[従業員]名前=$名前)
` 同じ値を持つレコードが見つかった場合
$発見:=(Records in selection ([従業員]) > 0)
` 前に保存したセレクションを使用する
USE NAMED SELECTION ("ユーザソート")
` レコードを戻す
POP RECORD ([従業員])
` レコードが見つかったら、ユーザに知らせてフィールド内の値を無効にする
If ($発見)
   ALERT ("名前フィールドに重複データが入力されています")
   REJECT ([従業員]名前)
End if
```

COPY NAMED SELECTION

COPY NAMED SELECTION ({テーブル} ; 名前)

引数	タイプ	説明
テーブル	テーブル	セレクションコピーするテーブル
名前	文字列	作成するセレクションの名前

COPY NAMED SELECTIONコマンドは、＜テーブル＞のカレントセレクションを命名セレクション＜名前＞にコピーします。オプション引数＜テーブル＞が指定されていない場合は、プロセスのデフォルトテーブルを使用します。＜名前＞にはセレクションのコピーが含まれます。このコマンドは、＜テーブル＞内のカレントセレクションに適用されます。プロセスにおける＜テーブル＞のカレントセレクションとカレントレコードは変更されません。

命名セレクションは実際にレコードを含むわけではなく、レコードへの順序付けされたポインタを含みます。各ポインタは4バイトです。従って、**COPY NAMED SELECTION**コマンドを使用してセレクションをコピーすると、セレクション内のレコード数×4バイト分のメモリが必要になります。命名セレクションはメモリに置かれますので、命名セレクションに必要な分とプロセス内のテーブルのカレントセレクションに必要な分のメモリを確保しなければなりません。

4D Server：命名セレクション＜名前＞とカレントセレクションは、両方ともサーバのメモリに置かれますので、サーバには十分なメモリを確保してください。

命名セレクション＜名前＞が使用したメモリを解放するには、**CLEAR NAMED SELECTION**コマンドを使用します。

▼ 次の例では、[従業員]テーブルに未払いの送り状があるかどうかを調べています。セレクションをソートして保存します。送り状が未払いのレコードをすべて検索します。その後、そのセレクションを再利用してメモリ内の命名セレクションを消去します：

ソートしたセレクションを後で使いたい場合には、命名セレクションを消去しなくても構いません。

```
ALL RECORDS ([従業員])
ORDER BY ([従業員])                          ` セレクションをソートする
` ソートしたセレクションを命名セレクションとして保存する
COPY NAMED SELECTION ([従業員] ; "ユーザソート")
QUERY ([従業員] ;[ 従業員]未払い=True)        ` 送り状が未払いのレコードを探す
If(Records in selection ([従業員]) > 0)      ` レコードが見つかった場合
   ALERT ("未払いの送り状があります")          ` ユーザに知らせる
End if
USE NAMED SELECTION ("ユーザソート") ` ソートした命名セレクションを利用
CLEAR NAMED SELECTION ("ユーザソート") ` セレクションをメモリから消去
```

CUT NAMED SELECTION

CUT NAMED SELECTION ({テーブル} ; 名前)

引数	タイプ	説明
テーブル	テーブル	セレクションカットするテーブル
名前	文字列	作成するセレクションの名前

CUT NAMED SELECTIONコマンドは、命名セレクション＜名前＞を作成し、＜テーブル＞のカレントセレクションをそこへ移します。このコマンドは、カレントセレクションをコピーするのではなく、実際に＜テーブル＞のカレントセレクションを移す点が**COPY NAMED SELECTION**コマンドと異なります。

このコマンドを実行した後、カレントプロセス内の＜テーブル＞のカレントセレクションは空になります。そのため、**CUT NAMED SELECTION**コマンドはレコードが修正されている最中は使用できません。

CUT NAMED SELECTIONコマンドは、**COPY NAMED SELECTION**コマンドよりも効率的です。**COPY NAMED SELECTION**コマンドでは選択したレコードの数×4バイトをメモリ内で複製します。**CUT NAMED SELECTION**コマンドではリストの参照だけを移動します。

▼ 次の例では、簡単に[顧客]テーブルのカレントセレクションを空にします：

```
CUT NAMED SELECTION ([顧客] ; "クリア")
CLEAR NAMED SELECTION ("クリア")
```

USE NAMED SELECTION

USE NAMED SELECTION (名前)

引数	タイプ	説明
名前	文字列	使用する命名セレクションの名前

USE NAMED SELECTIONコマンドは、命名セレクション＜名前＞をそれが属するテーブルのカレントセレクションとして使用します。

命名セレクションを作成すると、その命名セレクションはカレントレコードを記憶します。**USE NAMED SELECTION**コマンドはこのレコードの位置を取り出し、そのレコードを新しいカレントレコードにします。このコマンドは、カレントレコードをロードします。命名セレクション＜名前＞を作成した後にカレントレコードが変更された場合は、変更情報を失わないためには**USE NAMED SELECTION**コマンドを実行する前にそのレコードを保存しなければなりません。

■ **COPY NAMED SELECTION**コマンドを使用して命名セレクション＜名前＞を作成した場合は、＜名前＞はそれが属するテーブルのカレントセレクションにコピーされます。＜名前＞は消去されるまでメモリに残ります。命名セレクションをクリックして＜名前＞が使用しているメモリを消去するには、**CLEAR NAMED SELECTION**コマンドを用います。

■ **CUT NAMED SELECTION**コマンドを使用して命名セレクション＜名前＞を作成した場合は、カレントセレクションが＜名前＞に設定され、＜名前＞はメモリから消去されます。

命名セレクションは、それが作成された時点でのレコードのセレクションを表わす点に注意してください。命名セレクションが表わしているレコードが変更されると、命名セレクションは正確ではなくなります。従って、命名セレクションは通常、あまり変更されないレコードの集まりを表わします。命名セレクションが無効になる原因はいくつかあります。例えば、命名セレクションのレコードの変更、削除、および命名セレクションを決定する条件の変更などがあります。

また、トランザクションの際には一時的なレコードアドレスを使用します。トランザクション中に命名セレクションが作成されると、トランザクションが確定または取り消された場合に無効となるアドレスを含むことがあります。これは、トランザクションが受け入れられた後にレコードがその最終アドレスと実アドレスを受け取るためです。

CLEAR NAMED SELECTION

CLEAR NAMED SELECTION (名前)

引数	タイプ	説明
名前	文字列	消去する命名セレクションの名前

CLEAR NAMED SELECTIONコマンドは、命名セレクション<名前>をメモリから消去して、それが使用していたメモリを解放します。**CLEAR NAMED SELECTION**コマンドはテーブル、セレクション、およびレコードには影響しません。命名セレクションはメモリを使用しますので、不要になったら消去する習慣をつけることをお勧めします。

CUT NAMED SELECTIONコマンドを使用して命名セレクション<名前>を作成し、**USE NAMED SELECTION**コマンドで処理した場合には、命名セレクション<名前>は既にメモリ上にはありません。この場合は、**CLEAR NAMED SELECTION**コマンドを使用する必要はありません。

オブジェクトプロパティコマンド 35

この章では、「ルーチン」エディタの「Object Properties」テーマ内にあるオブジェクトプロパティコマンドについて説明します。この章のコマンドは、フィールドや変数などの入力フォームエリアに影響を与えます。例えば、これらのコマンドはフォームが表示や印刷される場合にのみ有効なフォームオブジェクトを変更します。このオブジェクトは新しいフォームやレコードが表示されると、「デザイン」モードで割り当てられた属性に戻ります。これらのコマンドはフォームメソッドやオブジェクトメソッドで使用されます。

BUTTON TEXT
DISABLE BUTTON
ENABLE BUTTON
FONT
FONT SIZE
FONT STYLE
SET CHOICE LIST
SET COLOR
SET ENTERABLE
SET FILTER
SET FORMAT
SET VISIBLE
SET RGB COLOR

「オブジェクトプロパティ」コマンドは、フォームの中にあるオブジェクトのプロパティ上で機能します。「ユーザ」モードまたは「カスタム」モードでフォームを使用しながら、「オブジェクトプロパティ」コマンドによって、オブジェクトの表示様式や動作を変更することができます。

重要：これらのコマンドの範囲は現在使われているフォームです。フォームを終了すると、変更内容が無効になります。

オブジェクト名またはデータベース名によるオブジェクトへのアクセス

「オブジェクトプロパティ」コマンドは、以下で説明するのと同じシンタックスを共有します。

コマンド名 ({*;} オブジェクト {; 各コマンドに指定する引数)

オプション引数<*>を指定する場合は、引数<オブジェクト>によってオブジェクト名(文字列)を指定します。

1回の呼び出しで1つのフォームの複数のオブジェクトを指定するには、そのオブジェクト名の中でワイルドカード記号 “ @ ” を使用します。次の表に、このコマンドを使用して指定できるオブジェクト名の例を示します。

オブジェクト名	呼び出しによって影響を受けるオブジェクト
mainGroupBox	オブジェクト「mainGroupBox」のみ
main@	名前が"main"で始まるオブジェクト
@GroupBox	名前が"GroupBox"で終わるオブジェクト
@Group@	名前が"Group"という文字を含むオブジェクト
main@Btn	名前が"main"で始まり、"Btn"で終わるオブジェクト
@	フォームの中にあるすべてのオブジェクト

オプション引数<*>を省略する場合は、引数<オブジェクト>によって特定のフィールドや変数を指定します。この場合、文字列ではなく、フィールドまたは変数参照(フィールドまたは変数オブジェクトのみ)を指定します。

注：この2番目のシンタックスは、4th Dimensionの旧バージョンと互換性があります。

データ入力属性

この節のコマンドは、入力フォームのデータの入力属性を設定します。これらのコマンドは、テキスト、数値、日付、時間データ入力エリアだけに使用します。フォームが画面上に表示されている間だけ有効です。新しいレコードまたはフォームが表示されるとデフォルト値の設定が有効となります。

画面上への表示と印刷の両方に、出力フォームを使用することのできる**SET FORMAT**コマンドは例外です。

これらの属性の設定に関する詳細は、『4th Dimension / 4D First デザインリファレンス』を参照してください。

SET FILTER

SET FILTER ({*;} オブジェクト ; 入力フィルタ)

引数	タイプ		説明
*		→	指定した場合、オブジェクトは オブジェクトの名前（文字列） 省略した場合は、フィールドまたは変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（*を指定した場合） フィールドまたは変数（*を省略した場合）
入力フィルタ	文字列	→	使用する入力フィルタ

SET FILTERコマンドは、引数＜オブジェクト＞で指定されたオブジェクトの入力フィルタを引数＜入力フィルタ＞に変更します。

オプション引数＜*＞を指定すると、＜オブジェクト＞にオブジェクト名を指し示します。オプション引数＜*＞を省略すると、＜オブジェクト＞にフィールドまたは変数を指し示します。この場合、文字列ではなく、フィールドまたは変数参照（フィールドや変数のみ）を指定します。

引数＜入力フィルタ＞に空の文字列を指定すると、オブジェクトのカレント入力フィルタを消去します。

注：このコマンドは、サブフォームのリストフォームに配置されたオブジェクトでは機能しません。

注：引数＜入力フィルタ＞で「データベースプロパティ」ダイアログボックスで前もって定義した入力フィルタを使用するには、入力フィルタ名の前に縦棒（|）を挿入します。

▼ 次の例は、郵便番号フィールドに対する文字フィルタを設定します。住所が米国の場合は、米国の郵便番号フィルタを設定します。それ以外の場合は、任意の入力ができるように設定します：

```
If (国名="US")
   ` 郵便番号フォーマットのフィルタを設定
  SET FILTER (郵便番号 ; "&9#####")
Else
   ` 文字、数値、大文字を受け入れるためフィルタを設定
  SET FILTER (郵便番号 ; |"~@")
End if
```

▼ 次の例は、フィールド“ フィールド ”内に入力できる2文字を“ a ”、“ b ”、“ c ”、“ g ”に制限します：

```
$フィルタ:="&" + Char(Double quote) + "a;b;c;g" + Char(Double quote) + "##"
SET FILTER ([テーブル]フィールド ; $フィルタ)
```

注：この例では、入力フィルタに“ &"a;b;c;g"## ”を設定します。

参照：SET FORMAT

SET CHOICE LIST

SET CHOICE LIST ({*;} オブジェクト ; リスト)

引数	タイプ		説明
*		→	指定した場合、オブジェクトは オブジェクトの名前（文字列） 省略した場合は、フィールドまたは変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（*を指定した場合） フィールドまたは変数（*を省略した場合）
リスト	文字列	→	選択リストに使用するリスト名

SET CHOICE LISTコマンドは、引数＜オブジェクト＞で指定されたオブジェクトの選択リストを引数＜リスト＞で渡した階層リスト（「デザイン」モードのリストエディタで定義）に変更します。

このコマンドは、入力フォームまたはダイアログフォームで使用できます。フォーム内のフィールドおよび入力可変数はテキストとして入力されます。

オプション引数＜*＞を指定すると、＜オブジェクト＞にオブジェクト名を指し示します。オプション引数＜*＞を省略すると、＜オブジェクト＞にフィールドまたは変数を指し示します。この場合、文字列ではなく、フィールドまたは変数参照（フィールドや変数のみ）を指定します。

注：このコマンドは、サブフォームのリストフォームに配置されたオブジェクトでは機能しません。

▼ 次の例は、“ 船積み業者 ” フィールドに対する選択項目リストを設定します。船積みを夜間に行う場合に夜間に船積みすることができる船積み業者のリストを選択項目リストに設定します。それ以外の場合は標準的な船積み業者に設定します：

```
If (夜間)
  SET CHOICE LIST (船積み業者 ; "夜間可")
Else
  SET CHOICE LIST (船積み業者 ; "通常")
End if
```

SET ENTERABLE

SET ENTERABLE ({*;} オブジェクト ; 入力可)

引数	タイプ		説明
*		→	指定した場合、オブジェクトは オブジェクトの名前（文字列） 省略した場合は、フィールドまたは変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（*を指定した場合） フィールドまたは変数（*を省略した場合）
入力可	ブール	→	True = 入力可、False = 入力不可

SET ENTERABLEコマンドは、引数＜オブジェクト＞を入力可または入力不可に設定します。

オプション引数＜*＞を指定すると、＜オブジェクト＞にオブジェクト名を指し示します。オプション引数＜*＞を省略すると、＜オブジェクト＞にフィールドまたは変数を指し示します。この場合、文字列ではなく、フィールドまたは変数参照（フィールドや変数のみ）を指定します。

このコマンドの使用方法は、「フォーム」エディタの「オブジェクトプロパティ」ウインドウ内でフィールドや変数に入力可／入力不可を選択する機能とまったく同じものです。このコマンドは、サブフォームのオブジェクトメソッド内にこのコマンドが使用されている場合にのみそのサブフォーム内で機能します。

引数＜入力可＞が入力可能（True）な場合、ユーザはそのエリアの中でカーソルを移動したりデータを入力することができます。引数＜入力可＞が入力不可（False）の場合、ユーザはそのエリアの中でカーソルを移動したりデータを入力することはできません。

▼ 次の例は、船積みの重量に応じて、船積みフィールドを設定します。船積みが1オンス以下の場合は、船積み業者に米国郵便を設定し、このフィールドを入力不可にします。それ以外の場合には、入力可に設定します：

```
If (重量<=1)
  船積み業者:="米国郵便"
  SET ENTERABLE (船積み業者 ; False)
Else
  SET ENTERABLE (船積み業者 ; True)
End if
```

参照：DISABLE BUTTON、ENABLE BUTTON、SET VISIBLE

SET FORMAT

SET FORMAT ({*;} オブジェクト ; 表示フォーマット)

引数	タイプ		説明
*		→	指定した場合、オブジェクトは オブジェクトの名前（文字列） 省略した場合は、フィールドまたは変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（*を指定した場合） フィールドまたは変数（*を省略した場合）
表示フォーマット	文字列	→	オブジェクトに使用する表示フォーマット

SET FORMATコマンドは、引数＜オブジェクト＞で指定されたオブジェクトの表示フォーマットを引数＜表示フォーマット＞で渡したフォーマットに変更します。

オプション引数＜*＞を指定すると、＜オブジェクト＞にオブジェクト名を指し示します。オプション引数＜*＞を省略すると、＜オブジェクト＞にフィールドまたは変数を指し示します。この場合、文字列ではなく、フィールドまたは変数参照（フィールドや変数のみ）を指定します。

SET FORMATコマンドは、入力フォームと出力フォームの両方で使用することができます。また、フィールド、入力可／入力不可変数、数値のスクロールエリアに適用できます。

オブジェクトに表示するデータタイプに合った表示フォーマットを使用します：

■ ブールフィールドをフォーマットするには、2つの値をセミコロン(;)で区切って渡します。
■ 日付フィールドまたは変数をフォーマットするには、引数＜表示フォーマット＞に次の表にある4th Dimensionにあらかじめ定義されている定数または値を渡します。

定数	タイプ	値
Short	倍長整数	1
Abbreviated	倍長整数	2
Long	倍長整数	3
MM DD YYYY	倍長整数	4
Month Date Year	倍長整数	5
Abbr Date, year	倍長整数	6
MM DD YYYY Forcedr	倍長整数	7

■ 時間フィールドまたは変数をフォーマットするには、引数＜表示フォーマット＞に次の表にある4th Dimensionにあらかじめ定義されている定数または値を渡します。

定数	タイプ	値
HH MM SS	倍長整数	1
HH MM	倍長整数	2
Hour Min Sec	倍長整数	3
Hour Min	倍長整数	4
HH MM AM PM	倍長整数	5

■ ピクチャフィールドまたは変数をフォーマットするには、引数＜表示フォーマット＞に次の表にある4th Dimensionにあらかじめ定義されている定数または値を渡します。

定数	タイプ	値
Truncated Centered	倍長整数	1
Scaled to Fit	倍長整数	2
On Background	倍長整数	3
Truncated non Centered	倍長整数	4
Scaled to fit proportional	倍長整数	5
Scaled to fit prop centered	倍長整数	6

文字および数値の表示フォーマットに関する詳細は、『4th Dimension / 4D First デザインリファレンス』を参照してください。

注：引数＜入力フィルタ＞で「データベースプロパティ」ダイアログボックスで前もって定義した入力フィルタを使用するには、入力フィルタ名の前に縦棒（|）を挿入します。

▼次の例は、変数 “ vDate ” に対して、定数「Short」の日付フォーマットを設定します：

```
SET FORMAT (vDate ; Char (Short))
```

▼ 次の例は、郵便番号の長さによってフィールド “ [会社]郵便番号 ” に対するフォーマットを変更します：

```
If (Length ([会社]郵便番号)=5)
   SET FORMAT ([会社]郵便番号 ; "###-##")
Else
   SET FORMAT ([会社]郵便番号 ; "###")
End if
```

▼ 次の例は、ブールフィールドのフォーマットにデフォルトの “ True (真) ” と “ False(偽) ” の代わりに、“ 既婚 ” と “ 未婚 ” を表示するためにセットします：

```
SET FORMAT ([従業員]既婚歴 ; "既婚;未婚")
```

参照：SET FILTER

BUTTON TEXT

BUTTON TEXT ({*;} オブジェクト ; ボタンテキスト)

引数	タイプ		説明
*		→	指定した場合、オブジェクトは オブジェクトの名前（文字列） 省略した場合は、フィールドまたは変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（*を指定した場合） 変数（*を省略した場合）
ボタンテキスト	文字列	→	ボタンのタイトル（ラベル）

BUTTON TEXTコマンドは、引数＜オブジェクト＞で指定されたオブジェクトのボタンタイトルを引数＜ボタンテキスト＞で渡した値と変更します。

オプション引数＜*＞を指定すると、＜オブジェクト＞にオブジェクト名を指し示します。オプション引数＜*＞を省略すると、＜オブジェクト＞にフィールドまたは変数を指し示します。この場合、文字列ではなく、フィールドまたは変数参照（フィールドや変数のみ）を指定します。

BUTTON TEXTコマンドは、テキストを表示するボタン：ボタン、チェックボックス、ラジオボタンにのみ影響を与えます。

一般的に、このコマンドは1度に1つのボタンに適用します。ボタンエリアは、ボタンテキストを表示するだけの十分な大きさが必要です。ボタンエリアが小さすぎると、テキストを途中までしか表示しません。引数＜ボタンテキスト＞にキャリッジリターンは使用しないでください。

▼ 次の例は、**MODIFY SELECTION**コマンドを使って表示された出力フォームのフッタエリアにある検索ボタンのオブジェクトメソッドです。このメソッドは、[従業員]テーブルを検索し、その検索結果によって“ b削除 ”とラベルの付いたボタンを使用可または使用不可にして、そのボタンタイトルを変更します：

```
QUERY ([従業員]; [従業員]名前 = vName)
Case of
   \ (Records in selection ([従業員]) = 0) `削除する従業員がいない場合
      BUTTON TEXT (b削除;" 削除")
      DISABLE BUTTON (b削除)
   \ (Records in selection ([従業員]) = 1) ` 削除する従業員が1人の場合
      BUTTON TEXT (b削除;"単一削除")
      ENABLE BUTTON (b削除)
   \ (Records in selection ([従業員]) > 1) ` 削除する従業員が複数の場合
      BUTTON TEXT (b削除;"複数削除")
      ENABLE BUTTON (b削除)
End case
```

参照：DISABLE BUTTON, ENABLE BUTTON

ENABLE BUTTON

ENABLE BUTTON ({*;} オブジェクト)

引数	タイプ		説明
*		→	指定した場合、オブジェクトは オブジェクトの名前（文字列） 省略した場合は、フィールドまたは変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（*を指定した場合） 変数（*を省略した場合）

ENABLE BUTTONコマンドコマンドは、引数＜オブジェクト＞で指定されたフォームオブジェクトを使用可能にします。

使用可能なボタンまたはオブジェクトは、マウスクリックおよびキーボードショートカットに対応します。

オプション引数＜*＞を指定すると、＜オブジェクト＞にオブジェクト名を指し示します。オプション引数＜*＞を省略すると、＜オブジェクト＞にフィールドまたは変数を指し示します。この場合、文字列ではなく、フィールドまたは変数参照（フィールドや変数のみ）を指定します。

このコマンドは、次のようなオブジェクトタイプに適用されます。

- ボタン、デフォルトボタン、3Dボタン、透明ボタン、ハイライトボタン
- ラジオボタン、3Dラジオボタン、ラジオピクチャ
- チェックボックス、3Dチェックボックス
- ポップアップメニュー、ドロップダウンリストリスト、コンボボックス、メニュードロップダウンリストボックス
- サーモメータ、ルーラ

注：自動動作属性を割り当てられたオブジェクトでこのコマンドを使用するのは現実的ではありません。それは、必要な時にその制御のステータスを4th Dimensionが変更するからです。

▼ 次の例は、「b有効」ボタンを使用可能にします：

```
ENABLE BUTTON (b有効)
```

▼ 次の例は、“btn”を含んだ名前を持つフォームオブジェクトのすべてを使用可能にします：

```
ENABLE BUTTON (* ; "@btn@")
```

▼ **BUTTON TEXT**コマンドの例を参照してください。

参照：BUTTON TEXT、DISABLE BUTTON

DISABLE BUTTON

DISABLE BUTTON ({*;} オブジェクト)

引数	タイプ		説明
*		→	指定した場合、オブジェクトは オブジェクトの名前（文字列） 省略した場合は、フィールドまたは変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（*を指定した場合） 変数（*を省略した場合）

DISABLE BUTTONコマンドコマンドは、引数＜オブジェクト＞で指定されたフォームオブジェクトを使用不可にします。

使用不可のボタンまたはオブジェクトは、マウスクリックおよびキーボードショートカットに対応します。

オプション引数＜*＞を指定すると、＜オブジェクト＞にオブジェクト名を指し示します。オプション引数＜*＞を省略すると、＜オブジェクト＞にフィールドまたは変数を指し示します。この場合、文字列ではなく、フィールドまたは変数参照（フィールドや変数のみ）を指定します。

このコマンドは、次のようなオブジェクトタイプに適用されます。

■ ボタン、デフォルトボタン、3Dボタン、透明ボタン、ハイライトボタン
■ ラジオボタン、3Dラジオボタン、ラジオピクチャ
■ チェックボックス、3Dチェックボックス
■ ポップアップメニュー、ドロップダウンリストリスト、コンボボックス、メニュードロップダウンリストボックス
■ サーモメータ、ルーラ

注：自動動作属性を割り当てられたオブジェクトでこのコマンドを使用するのは現実的ではありません。それは、必要な時にその制御のステータスを4th Dimensionが変更するからです。

▼ 次の例は、「b有効」ボタンを使用不可にします：

```
DISABLE BUTTON (b有効)
```

▼ 次の例は、“ btn ” を含んだ名前を持つフォームオブジェクトのすべてを使用不可にします：

```
DISABLE BUTTON (* ; "@btn@")
```

▼ **BUTTON TEXT**コマンドの例を参照してください。

参照：BUTTON TEXT、ENABLE BUTTON

SET COLOR

SET COLOR ({*;} オブジェクト ; カラー)

引数	タイプ		説明
*		→	指定した場合、オブジェクトは オブジェクトの名前（文字列） 省略した場合は、フィールドまたは変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（*を指定した場合） 変数（*を省略した場合）
ボタンテキスト	数値	→	オブジェクトに使用するカラー

SET COLORコマンドは、引数＜オブジェクト＞で指定されたフォームオブジェクトの前景色と背景色を設定します。

オプション引数＜*＞を指定すると、＜オブジェクト＞にオブジェクト名を指し示します。オプション引数＜*＞を省略すると、＜オブジェクト＞にフィールドまたは変数を指し示します。この場合、文字列ではなく、フィールドまたは変数参照（フィールドや変数のみ）を指定します。

引数＜カラー＞で、前景色と背景色を設定します。カラーは次の計算式で求められます。

カラー:= - (前景色 + (256 * 背景色))

前景色と背景色はカラーパレット内部の（0から255までの）カラー番号です。引数＜カラー＞は、常に負の数値です。例えば、前景色が20で、背景色が10の場合、＜カラー＞は、- (20 + (256 * 10))または-2580となります。

注：「フォーム」エディタの「オブジェクトプロパティ」ウインドウでカラーパレットを見ることができます。

カラーで使用される共通の番号は、下記のように前もって定義された定数で提供されます。

定数	タイプ	値
White	倍長整数	0
Yellow	倍長整数	1
Orange	倍長整数	2
Red	倍長整数	3
Purple	倍長整数	4
Dark Blue	倍長整数	5
Blue	倍長整数	6
Light Blue	倍長整数	7
Green	倍長整数	8
Dark Green	倍長整数	9
Dark Brown	倍長整数	10

Dark Grey	**倍長整数**	**11**
Light Grey	**倍長整数**	**12**
Brown	**倍長整数**	**13**
Grey	**倍長整数**	**14**
Black	**倍長整数**	**15**

注：SET COLORコマンドがデフォルトの4th Dimensionカラーパレットのインデックスカラーを使って作業するのに対して、バージョン6で新しく追加されたSET RGB COLORコマンドは任意のRGBカラーで作業することができます。

▼ 次の例は、“ 私のボタン ” という名前のボタンに対して色を設定します。この色は “ v前景色 ” と “ v背景色 ” という名前の変数に値を設定しています：

```
SET COLOR (私のボタン; - (v前景色 + (256 * v背景色)))  ` 私のボタンに色を設定
```

参照：SET RGB COLOR

FONT

FONT ({*;} オブジェクト ; フォント)

引数	**タイプ**		**説明**
*		→	指定した場合、オブジェクトは オブジェクトの名前（文字列） 省略した場合は、フィールドまたは変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（*を指定した場合） 変数（*を省略した場合）
フォント	文字列／数値	→	フォント名またはフォント番号

FONTコマンドは、引数＜オブジェクト＞で指定されたフォームオブジェクトに引数＜フォント＞で渡したフォント名またはフォント番号を設定します。

オプション引数＜*＞を指定すると、＜オブジェクト＞にオブジェクト名を指し示します。オプション引数＜*＞を省略すると、＜オブジェクト＞にフィールドまたは変数を指し示します。この場合、文字列ではなく、フィールドまたは変数参照（フィールドや変数のみ）を指定します。

▼ 次の例は、“ bOK ” という名前のボタンに対して “ Times ” フォントをセットします：

```
FONT (bOK ; "Times")
```

▼ 次の例は、“ info ” という名前を含んだすべてのフォームオブジェクトに対して “ Times ” フォントをセットします：

```
FONT (* ; @info@ ; "Times")
```

参照：FONT SIZE、FONT STYLE

FONT SIZE

FONT SIZE ({*;} オブジェクト ; フォントサイズ)

引数	タイプ		説明
*		→	指定した場合、オブジェクトは オブジェクトの名前（文字列） 省略した場合は、フィールドまたは変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（*を指定した場合） 変数（*を省略した場合）
フォントサイズ	数値	→	フォントサイズ（ポイント）

FONT SIZEコマンドは、引数＜オブジェクト＞で指定されたフォームオブジェクトに引数＜フォントサイズ＞で渡したフォントサイズ(ポイント数)を設定します。

オプション引数＜*＞を指定すると、＜オブジェクト＞にオブジェクト名を指し示します。オプション引数＜*＞を省略すると、＜オブジェクト＞にフィールドまたは変数を指し示します。この場合、文字列ではなく、フィールドまたは変数参照（フィールドや変数のみ）を指定します。

引数＜サイズ＞は、1から255までの整数です。指定したフォントサイズが実在しない場合には、文字をスケーリングします。

フォーム上のオブジェクトは、指定したフォントサイズを表示するのに十分な大きさがなければなりません。十分な大きさがない場合には、テキストが途中までしか表示されなかったり、全く表示されなくなります。

▼ 次の例は、“ v情報 ” と名付けられた変数に対して14ポイントのフォントサイズを設定します：

```
FONT SIZE (v情報 ; 14)
```

▼ 次の例は、“ hl ” という名前で始まるすべてのフォームオブジェクトに対して14ポイントのフォントサイズを設定します：

```
FONT SIZE (* ; "hl@" ; 14)
```

参照：FONT、FONT STYLE

FONT STYLE

FONT STYLE ({*;} オブジェクト ; フォントスタイル)

引数	タイプ		説明
*		→	指定した場合、オブジェクトは オブジェクトの名前（文字列） 省略した場合は、フィールドまたは変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（*を指定した場合） 変数（*を省略した場合）
フォントスタイル	数値	→	フォントスタイル

FONT STYLEコマンドは、引数＜オブジェクト＞で指定されたフォームオブジェクトに引数＜フォントスタイル＞で渡したフォントスタイルを設定します。

オプション引数＜*＞を指定すると、＜オブジェクト＞にオブジェクト名を指し示します。オプション引数＜*＞を省略すると、＜オブジェクト＞にフィールドまたは変数を指し示します。この場合、文字列ではなく、フィールドまたは変数参照（フィールドや変数のみ）を指定します。

下記で示したフォントスタイルの定数を加算した数値を引数＜フォントスタイル＞で渡します。フォントスタイルの定数は、次の表のようにあらかじめ定義されています：

定数	タイプ	値
Plain	倍長整数	0
Bold	倍長整数	1
Italic	倍長整数	2
Underline	倍長整数	4
Outline	倍長整数	8
Shadow	倍長整数	16
Condensed	倍長整数	32
Extended	倍長整数	64

注；Windows版では、“ Plain ”、“ Bold ”、“ Italic ”、“ Underline ” しか利用できません。

▼ 次の例は、“ b追加 ” という名前のボタンに対して “ ボールド（太字）+イタリック（斜体）” のフォントスタイルを設定します：

```
FONT STYLE (b追加 ; Bold + Italic)
```

▼ 次の例は、“ vt ” という名前で始まるすべてのフォームオブジェクトに対して “ Plain(標準) ” フォントスタイルを設定します：

```
FONT STYLE (* ; "vt@" ; Plain)
```

参照：FONT、FONT SIZE

SET VISIBLE

SET VISIBLE ({*} ; オブジェクト ; 表示)

引数	タイプ		説明
*		→	指定した場合、オブジェクトはオブジェクト名省略した場合、引数＜オブジェクト＞はフィールドまたは変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（*を指定した場合）、またはフィールドまたは変数（*を省略した場合）
表示	ブール	→	表示の場合はTrue。非表示の場合はFalse

SET VISIBLEコマンドは、**引数＜**オブジェクト＞によって指定されるオブジェクトを表示したり、隠したりします。

オプション引数の“ * ”を指定した場合は、引数＜オブジェクト＞にオブジェクト名(文字列)を指示します。省略した場合は、引数＜オブジェクト＞にフィールドまたは変数を指示します。この場合、文字列の代わりに、フィールドまたは変数参照(フィールドまたは変数オブジェクトのみ)を指定してください。オブジェクト名に関する詳細は、「オブジェクトプロパティ」コマンドの節を参照してください。

引数＜表示＞にTrue渡すと、オブジェクトが表示されます。引数＜表示＞にFalseを渡すと、オブジェクトが隠されます。

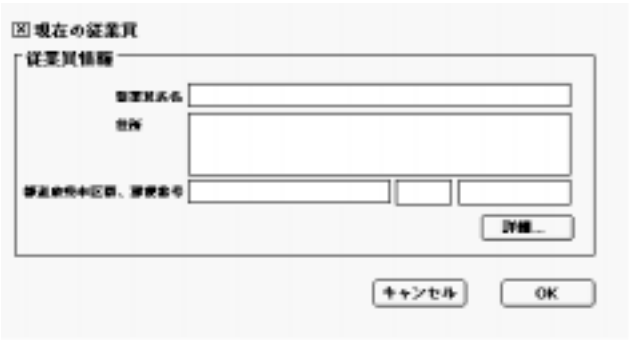
例：

次の図は、「デザイン」モードにおける標準的なフォームです。

「従業員情報」グループボックスにあるオブジェクトは、(グループボックスを含めて)どれもオブジェクトの名前に“ Employer ”という文字が入っています。「現在の従業員」チェックボックスをオンにすると、オブジェクトが表示されます。チェックボックスをオフにすると、オブジェクトが表示されなくなります。以下は、チェックボックスのオブジェクトメソッドです。

```
`「現在の従業員」(cbCurrentlyEmployed)チェックボックスのオブジェクトメソッド
Case of
   \ (Form event=On Load)
      cbCurrentlyEmployed:=1
   \ (Form event=On Clicked)
      `オブジェクト名に " Emp " を含んだオブジェクトをすべて表示または非表示する
      SET VISIBLE (* ; "@emp@" ; cbCurrentlyEmployed  #  0)
      `ただし、常にチャックボックス自身は表示されていなければならない
      SET VISIBLE (cbCurrentlyEmployed ; True)
End case
```

したがって、「ユーザ」モードまたは「カスタム」モードでは、フォームは次のように表示されます。



または、

参照：DISABLE BUTTON、ENABLE BUTTON、SET ENTERABLE

SET RGB COLOR

SET RGB COLOR ({*} ; オブジェクト ; 前景色 ; 背景色)

引数	タイプ		説明
*		→	指定した場合、オブジェクトはオブジェクト名省略した場合、引数<オブジェクト>はフィールドまたは変数
オブジェクト	フォームオブジェクト	→	オブジェクト名（*を指定した場合)、またはフィールドまたは変数（*を省略した場合）
前景色	数値	→	前景色におけるRGBカラーの値
背景色	数値	→	背景色におけるRGBカラーの値

SET RGB COLORコマンドは、引数<オブジェクト>とオプション引数の<*>によって指定されるオブジェクトの前景色と背景色を変更します。

オプション引数の “ * ” を指定した場合は、引数<オブジェクト>にオブジェクト名(文字列)を指示します。省略した場合は、引数<オブジェクト>にフィールドまたは変数を指示します。この場合、文字列の代わりに、フィールドまたは変数参照(フィールドまたは変数オブジェクトのみ)を指定してください。オブジェクト名に関する詳細は、「オブジェクトプロパティ」コマンドの節を参照してください。

引数<前景色>と<背景色>でRGB のカラー値を指定します。RGBの値は、4バイトの倍長整数です。そのフォーマット(0x00RRGGBB)は、以下の表で説明します(バイトには、左から右に0から3まの数字が付されます)。

バイト	説明
3	絶対RGBカラーの場合は、ゼロでなければなりません。
2	色のうちの赤の成分(0〜255)
1	色のうちの緑の成分(0〜255)
0	色のうちの青の成分(0〜255)

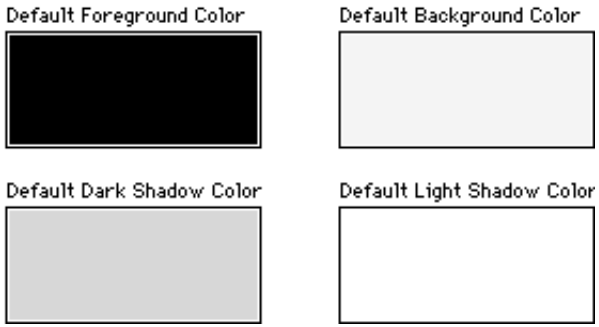
次の表は、RGBのカラー値の例をいくつか示します。

値	説明
0x00000000	黒
0x00FF0000	赤
0x0000FF00	緑
0x000000FF	青
0x007F7F7F	グレー
0x00FFFF00	黄
0x00FF7F7F	赤（パステル）
0x00FFFFFF	白

この代わりに4th Dimensionが使用する4つの自動色のいずれかを指定して、色を自動設定してオブジェクトを描くことができます。4th Dimensionでは、次のような定義済みの定数が用意されています。

定数	タイプ	値
Default foreground color	倍長整数	-1
Default background color	倍長整数	-2
Default dark shadow color	倍長整数	-3
Default light shadow color	倍長整数	-4

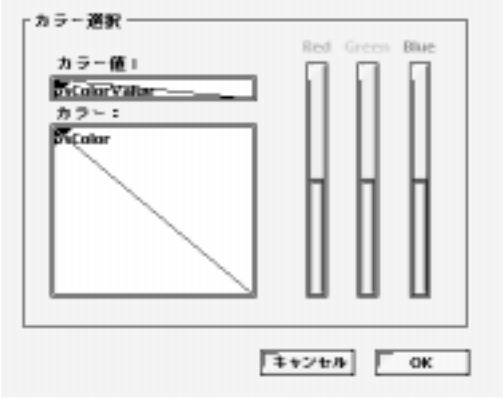
(標準システムにおける)これらの色を次に示します。



注：Windows上では、これらの自動色はシステムに依存します。「カラー」コントロールパネルでシステム色を変更すると、4th Dimensionは、それに従ってその自動色を調整します。自動色の値を使用すると、オブジェクトは上に示すサンプル色ではなく、システム色で設定されます。

例：

次のフォームには、「vsColorValue」と「vsColor」という 2つの入力不可変数と「thRed」、「thGreen」、「thBlue」という 3つのサーモメータが含まれます。



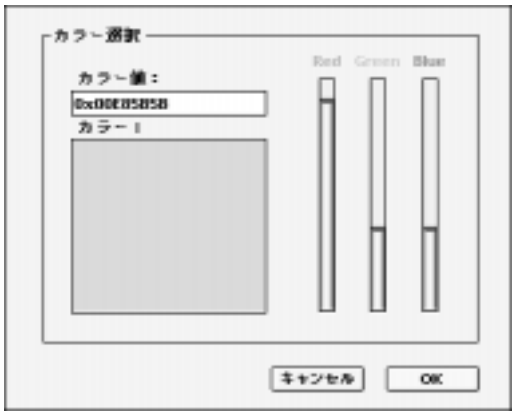
以下は、このオブジェクト用のメソッドです：

```
`「vsColorValue」入力不可変数のオブジェクトメソッド
Case of
   \ (Form event=On Load)
      vsColorValue:="0x00000000"
End case

`「vsColor」入力不可変数のオブジェクトメソッド
Case of
   : (Form event=On Load)
      vsColor:=""
      SET RGB COLOR (vsColor ; 0x00FFFFFF ; 0x0000)
End case

`「thRed」サーモメータのオブジェクトメソッド
CLICK IN COLOR THERMOMETER

`「thGreen」サーモメータのオブジェクトメソッド
CLICK IN COLOR THERMOMETER

`「thBlue」サーモメータのオブジェクトメソッド
CLICK IN COLOR THERMOMETER
```

3つのサーモメータによって呼び出されるプロジェクトメソッドは、以下のようになります。

```
`「CLICK IN COLOR THERMOMETER」プロジェクトメソッド
SET RGB COLOR (vsColor ; 0x00FFFFFF ; (thRed << 16)+(thGreen << 8)+thBlue)
vsColorValue:=String((thRed << 16)+(thGreen << 8)+thBlue;"&x")
If (thRed=0)
   vsColorValue:=Substring (vsColorValue ; 1 ;2) +"0000"+Substring(vsColorValue ; 3)
End if
```

「ユーザ」モードまたは「カスタム」モードでは、フォームが以下のようになります。



参照：SET COLOR

リスト(統計)上の関数　36

この章では、「ルーチン」エディタの「On a Series」テーマ内にあるリスト上の統計関数について説明します。統計関数は、一連の値に関する計算を行います。**Average**、**Max**、**Min**、**Sum**、**Sum squares**、**Std deviation**、そして**Variance**の各関数はフィールドやサブフィールドに使用されます。フィールドに用いる場合、レコードのセレクションに使用されます。サブフィールドに使用する場合には、カレントレコードのサブフィールドのセレクションに使用されます。**Sum squares**、**Std deviation**、**Variance**の各関数は印刷時のみにフィールドで使用されます。

Average
Max
Min
Std deviation
Sum
Sum squares
Variance

これらの関数は、数値データに対してだけ機能します。これらの関数は、すべて数値を返します。

フィールドを使用する

Average、**Max**、**Min**、**Sum**の各関数をフィールドに対して使用する場合は、カレントセレクションのレコードをロードします。レコード数が多いと、処理に時間がかかります。

レポート作成時にこれらの関数を使用すると、他の場合とは異なる動作をします。印刷時には、レコードを毎回ロードしなければならないためです。**PRINT SELECTION**コマンドによって印刷したり、「ユーザ」モードで「ファイル」メニューの「プリント...」を選択して印刷する場合に、フォームメソッドやオブジェクトメソッドの中でこれらの関数を使用します。

レポートに統計関数を使用する際に、関数から返される値が意味を持つのは、フッタやブレーク、ブレークレベル0にこの関数が使用される場合だけです。つまり、これらの関数はレポートの最後、全てのレコードを処理し終えた時にのみ有効です。

通常、B0ブレークの入力不可エリアのオブジェクトメソッドで使用します。

統計関数に引数として渡されるフィールドは、数値フィールドでなければならないことを覚えておいてください。

Average

Average (一連の値) → 数値

引数	タイプ	説明
一連の値	フィールドまたはサブフィールド	平均を計算するためのデータ

Average関数は、＜一連の値＞の平均値を返します。＜一連の値＞がインデックスフィールドにある場合に、インデックスは平均値を求めるために使用されます。

▼ 次の例は、出力フォームの B0ブレークエリアの変数に値を代入します。変数のオブジェクトメソッドにこの行を入れます。オブジェクトメソッドは、レベル0のブレークが発生したときに実行されます：

```
v平均売上:=Average ([従業員]売上)
```

▼ 次のメソッドは、ブレーク処理をアクティブにしてセレクション内のレコードを印刷するために呼び出されます：

```
ALL RECORDS ([従業員])
ORDER BY ([従業員] ; [従業員]名字 ; >)
BREAK LEVEL (1)
ACCUMULATE ([従業員]給与)
OUTPUT FORM ([従業員] ; "印刷")
PRINT SELECTION ([従業員])
```

注：BREAK LEVELコマンドの引数は、印刷するレポートのブレーク数と同じ数でなければなりません。ブレーク処理に関する詳細は、第38章の「印刷コマンド」を参照してください。

Max

Max (一連の値) → 数値

引数	タイプ	説明
一連の値	フィールドまたはサブフィールド	最大値を得るためのデータ

Max関数は、＜一連の値＞の最大値を返します。＜一連の値＞がインデックスフィールドにある場合に、インデックスは最大値を求めるために使用されます。

▼ 次の例は、変数のオブジェクトメソッドです。変数“ v最大 ”は、フォームのブレーク0のエリアに配置されています。変数は、レポートの最後に印刷されます。オブジェクトメソッドは、フィールドの値の最大値を変数に代入します。この変数は、レポートの最後のブレークが発生したときに印刷されます：

```
v最大:=Max ([従業員]給与)
```

▼ 次の例は、ブレーク処理をアクティブにしてセレクション内のレコードを印刷するために呼び出されます：

```
ALL RECORDS ([従業員])
ORDER BY ([従業員] ; [従業員]名字 ; >)
BREAK LEVEL (1)
ACCUMULATE ([従業員]給与)
OUTPUT FORM ([従業員] ; "印刷")
PRINT SELECTION ([従業員])
```

注：BREAK LEVELコマンドの引数は、印刷するレポートのブレーク数と同じ数でなければなりません。ブレーク処理に関する詳細は、第38章の「印刷コマンド」を参照してください。

Min

Min (一連の値) → 数値

引数	タイプ	説明
一連の値	フィールドまたはサブフィールド	最小値を得るためのデータ

Min関数は、＜一連の値＞の最小値を返します。＜一連の値＞がインデックスフィールドにある場合に、インデックスは最小値を求めるために使用されます。

▼ 次の例は、変数のオブジェクトメソッドです。変数“ v最小 ”は、フォームのブレーク0のエリアに配置されています。変数は、レポートの最後に印刷されます。オブジェクトメソッドは、フィールドの値の最小値を変数に代入します。この変数は、レポートの最後のブレークが発生したときに印刷されます：

```
v最小:=Min ([従業員]給与)
```

▼ 次のメソッドは、ブレーク処理をアクティブにしてセレクション内のレコードを印刷するために呼び出されます：

```
ALL RECORDS ([従業員])
ORDER BY ([従業員] ; [従業員]名字 ; >)
BREAK LEVEL (1)
ACCUMULATE ([従業員]給与)
OUTPUT FORM ([従業員] ; "印刷")
PRINT SELECTION ([従業員])
```

注：BREAK LEVELコマンドの引数は、印刷するレポートのブレーク数と同じ数でなければなりません。ブレーク処理に関する詳細は、第38章の「印刷コマンド」を参照してください。

▼ 次の例は、従業員売上の最小値を求め、結果をアラートボックスに表示します。売上高は、サブフィールド[社員]売上'金額に格納されています：

```
ALERT ("売上高の最低は" + String (Min ([社員]売上'金額)) + "円です。")
```

Sum

Sum (一連の値) → 数値

引数	タイプ	説明
一連の値	フィールドまたはサブフィールド	合計を得るためのデータ

Sum関数は、＜一連の値＞の合計値を返します。＜一連の値＞がインデックスフィールドにある場合は、インデックスは合計値を求めるために使用されます。

▼ 次の例は、変数のオブジェクトメソッドです。変数“ v合計 ”がフォームに配置されています。変数に従業員全員の給与合計を代入します：

```
v合計:=Sum ([従業員]給与)
```

▼ 次のメソッドは、ブレーク処理をアクティブにしてセレクション内のレコードを印刷するために呼び出されます：

```
ALL RECORDS ([従業員])
ORDER BY ([従業員] ; [従業員]名字 ; >)
BREAK LEVEL (1)
ACCUMULATE ([従業員]給与)
OUTPUT FORM ([従業員] ; "印刷")
PRINT SELECTION ([従業員])
```

注：BREAK LEVELコマンドの引数は、印刷するレポートのブレーク数と同じ数でなければなりません。ブレーク処理に関する詳細は、第38章の「印刷コマンド」を参照してください。

Sum squares

Sum squares (一連の値) → 数値

引数	タイプ	説明
一連の値	フィールドまたはサブフィールド	平方和を得るためのデータ

Sum squares関数は、＜一連の値＞の平方和を返します。＜一連の値＞がインデックスフィールドにある場合にインデックスは平方和を求めるために使用されます。この関数をレポート印刷に使用する場合には、フィールドに対してのみ使用することができます。

▼ 次の例は、変数のオブジェクトメソッドです。一連のデータに対する平方和を変数“v平方和”に代入します。このオブジェクトメソッドはレポートの最後のブレークが発生したときに印刷されます：

```
v平方和:=Sum squares ([ファイル]一連データ)
```

▼ 次のメソッドは、ブレーク処理をアクティブにしてセレクション内のレコードを印刷するために呼び出されます：

```
ALL RECORDS ([従業員])
ORDER BY ([従業員] ; [従業員]名字 ; >)
BREAK LEVEL (1)
ACCUMULATE ([従業員]給与)
OUTPUT FORM ([従業員] ; "印刷")
PRINT SELECTION ([従業員])
```

注：BREAK LEVELコマンドの引数は、印刷するレポートのブレーク数と同じ数でなければなりません。ブレーク処理に関する詳細は、第38章の「印刷コマンド」を参照してください。

Std deviation

Std deviation (一連の値) → 数値

引数	タイプ	説明
一連の値	フィールドまたはサブフィールド	標準偏差を得るためのデータ

Std deviation関数は、＜一連の値＞の標準偏差を返します。＜一連の値＞がインデックスフィールドにある場合に、インデックスは標準偏差を求めるために使用されます。この関数をレポートの印刷に使用する場合、フィールドに対してのみ使用することができます。

▼ 次の例は、変数のオブジェクトメソッドです。一連のデータに関する標準偏差を変数“ v偏差 ”に代入します：

```
v偏差:=Std deviation ([ファイル]一連データ)
```

▼ 次のメソッドは、ブレーク処理をアクティブにしてセレクション内のレコードを印刷するために呼び出されます：

```
ALL RECORDS ([従業員])
ORDER BY ([従業員] ; [従業員]名字 ; >)
BREAK LEVEL (1)
ACCUMULATE ([従業員]給与)
OUTPUT FORM ([従業員] ; "印刷")
PRINT SELECTION ([従業員])
```

注：BREAK LEVELコマンドの引数は、印刷するレポートのブレーク数と同じ数でなければなりません。ブレーク処理に関する詳細は、第38章の「印刷コマンド」を参照してください。

Variance

Variance (一連の値) → 数値

引数	タイプ	説明
一連の値	フィールドまたはサブフィールド	分散を得るためのデータ

Variance関数は、一連の値の分散を返します。一連の値がインデックスフィールドにある場合に、インデックスは分散を求めるために使用されます。この関数をレポートの印刷に使用する場合には、フィールドに対してのみ使用することができます。

▼ 次の例は、変数のオブジェクトメソッドです。一連のデータに関する分散を変数 “ v分散 ” に代入します：

```
v分散:=Variance ([ファイル]一連データ)
```

▼ 次のメソッドは、ブレーク処理をアクティブにしてセレクション内のレコードを印刷するために呼び出されます：

```
ALL RECORDS ([従業員])
ORDER BY ([従業員] ; [従業員]名字 ; >)
BREAK LEVEL (1)
ACCUMULATE ([従業員]給与)
OUTPUT FORM ([従業員] ; "印刷")
PRINT SELECTION ([従業員])
```

注：BREAK LEVELコマンドの引数は、印刷するレポートのブレーク数と同じ数でなければなりません。ブレーク処理に関する詳細は、第38章の「印刷コマンド」を参照してください。

ピクチャコマンド　37

この章では、「ルーチン」エディタの「Picture」テーマ内にあるピクチャコマンドについて説明します。この節のコマンドは、QuickTimeを使ってピクチャを圧縮するために使用し、文書としてそれを保存します。

COMPRESS PICTURE
COMPRESS PICTURE FILE
LOAD COMPRESS PICTURE FROM FILE
GET PICTURE FROM LIBRARY
PICTURE LIBRARY LIST
Picture size
PICTURE PROPERTIES
REMOVE PICTURE FROM LIBRARY
SAVE PICTURE TO FILE
SET PICTURE TO LIBRARY

また、この章では、QuickTimeの圧縮タイプとピクチャ圧縮エラーにおけるエラーコードを示します。

重要：QuickTimeのピクチャ圧縮は、Windows上では利用できません。しかし、上記のコマンドは4th Dimensionの中で使用することができます。Windows上では利用できませんが、Macintosh上で利用できます。

COMPRESS PICTURE

COMPRESS PICTURE (ピクチャ ; 方法 ; 品質)

引数	タイプ	説明
ピクチャ	ピクチャ	圧縮するピクチャ
方法	文字列	圧縮方法(4バイト)
品質	整数	圧縮の品質

このコマンドは、ピクチャフィールドまたは変数に入っているピクチャを圧縮します。

引数＜方法＞は、圧縮するタイプを 4 バイトの文字列で指定します。詳細は、後述の「QuickTimeの圧縮タイプ」を参照してください。

引数＜品質＞は、圧縮されたピクチャの品質を 1 から 1000 までの整数で指定します。一般に、品質を下げると、ピクチャの圧縮率が増加します。

警告：指定された品質に対して可能な圧縮率は、圧縮するピクチャのサイズと種類によって異なります。小さなピクチャを圧縮すると、サイズが減少しないこともあります。

LOAD COMPRESS PICTURE FROM FILE

LOAD COMPRESS PICTURE FROM FILE (vRef ; 方法 ; 品質 ; ピクチャ)

引数	タイプ	説明
vRef	Docref	文書ファイル参照番号
方法	文字列	圧縮方法(4バイト)
品質	整数	圧縮の品質
ピクチャ	ピクチャ	圧縮するピクチャ

このコマンドは、ディスク上の文書からロードされたピクチャを圧縮します。

Open document関数を使って、PICT 文書を開くことができます。次に、この関数から返される文書ファイル参照番号を使い、文書中の PICT データを圧縮します。このコマンドはピクチャをメモリにロードし、指定した方法と品質に従って圧縮し、それを 4番目の引数に返します。

ピクチャは圧縮される前にメモリへロードされます。ピクチャをロードするためのメモリが足りない場合は、**LOAD COMPRESS PICTURE FROM FILE** コマンドを呼び出す前に **COMPRESS PICTURE FILE** コマンドを使ってください。

＜方法＞は、圧縮するタイプを 4バイトの文字列で指定します（詳細は、この章の後で説明する「QuickTimeの圧縮タイプ」を参照してください)。

＜品質＞は、圧縮されたピクチャの品質を 1 から1000までの整数で指定します。一般に、品質を下げると、ピクチャの圧縮率が増加します。

警告：指定された品質に対して可能な圧縮率は、圧縮するピクチャのサイズと種類によって異なります。小さなピクチャを圧縮すると、サイズが減少しないこともあります。

COMPRESS PICTURE FILE

COMPRESS PICTURE FILE (vRef ; 方法 ; 品質)

引数	タイプ	説明
vRef	Docref	文書ファイル参照番号
方法	文字列	圧縮方法(4バイト)
品質	整数	圧縮の品質

このコマンドは、ディスク上のピクチャファイルを圧縮します。単にピクチャを圧縮するためだけにこのコマンドを使うと、空いているメモリを使ってロードできなくなります。圧縮した場合は、**LOAD COMPRESS PICTURE FROM FILE**コマンドを使えば、メモリにロードすることができます。

＜方法＞は、圧縮するタイプを 4バイトの文字列で指定します。詳細は、後述の「QuickTimeの圧縮タイプ」を参照してください。

＜品質＞は、圧縮されたピクチャの品質を 1 から 1000 までの整数で指定します。一般に、品質を下げると、ピクチャの圧縮率が増えます。

警告：指定された品質に対して可能な圧縮率は、圧縮するピクチャのサイズと種類によって異なります。小さなピクチャを圧縮すると、サイズが減少しないこともあります。

SAVE PICTURE TO FILE

SAVE PICTURE TO FILE (vRef ; ピクチャ)

引数	タイプ	説明
vRef	Docref	文書ファイル参照番号
ピクチャ	ピクチャ	圧縮するピクチャ

SAVE PICTURE TO FILEコマンドは、**Create document** 関数を使って作成された文書中に＜ピクチャ＞を保存します。

Picture size

Picture size (ピクチャ) → 整数

引数	タイプ	説明
ピクチャ	ピクチャ	ピクチャフィールドまたは変数

この関数は、バイト単位で指定したピクチャのサイズを返します。

PICTURE PROPERTIES

PICTURE PROPERTIES (ピクチャ;幅;高さ;{水平オフセット{;垂直オフセット{;モード}}})

引数	タイプ		説明
ピクチャ	ピクチャ	→	情報を取得するピクチャ
幅	数値	←	ピクセルで表されたピクチャの幅
高さ	数値	←	ピクセルで表されたピクチャの高さ
水平オフセット	数値	←	背景に表示された際の水平オフセット
垂直オフセット	数値	←	背景に表示された際の垂直オフセット

PICTURE PROPERTIESコマンドは、引数＜ピクチャ＞に渡すピクチャに関する情報を返します。

引数＜幅＞と＜高さ＞は、ピクチャの幅と高さを返します。

オプション引数＜水平オフセット＞、＜垂直オフセット＞、＜モード＞は、フォームのバックグラウンド（背景）に表示されると、ピクチャの水平、垂直位置と転送モードを返します。

参照：Picture size

PICTURE LIBRARY LIST

PICTURE LIBRARY LIST(ピクチャ参照番号 ; ピクチャ名)

引数	タイプ		説明
ピクチャ参照番号	数値配列	←	ピクチャライブラリ内のピクチャ参照番号
ピクチャ名	文字列配列	←	ピクチャライブラリ内のピクチャの名前

PICTURE LIBRARY LISTコマンドは、データベースのピクチャライブラリの中に現在格納されているピクチャの参照番号と名前を返します。

このコマンドを呼び出すと、引数＜ピクチャ参照番号＞配列の中に参照番号、引数＜ピクチャ名＞配列の中にピクチャ名を返します。この2つの配列は、同期します。つまり、＜ピクチャ参照番号＞配列のｎ番目の要素は、＜ピクチャ名＞配列のｎ番目の要素内に返されるピクチャライブラリ内にあるピクチャ名が持つ参照番号になります。

＜ピクチャ参照番号＞配列は、実数、倍長整数または整数の配列です。インタプリタモードでは、その配列が**PICTURE LIBRARY LIST**コマンドへの呼び出しの前に宣言されていない場合、倍長整数型の配列がデフォルトとして作成されます。

＜ピクチャ名＞配列は、文字列またはテキストの配列です。インタプリタモードでは、その配列が**PICTURE LIBRARY LIST**コマンドへの呼び出しの前に宣言されていない場合、テキスト型の配列がデフォルトとして作成されます。

ピクチャライブラリに格納されるピクチャの名前は、最大31バイトです。＜ピクチャ名＞配列に文字列配列を使用する場合、返されるピクチャ名が途中で切り取られないように、十分な固定の長さを持った配列を宣言します。

ピクチャライブラリの中にピクチャがない場合、両方の配列は空で返されます。

ピクチャライブラリの中に現在格納されているピクチャの参照番号を取得するには、Size of array関数を使って、2つの配列の1つのサイズを取得します。

例：

1. 次のコードは、配列 “ alPicRef ” と “ alPicName ” の中にピクチャライブラリのカタログを返します：

```
PICTURE LIBRARY LIST(alPicRef ; asPicName)
```

2. 次の例は、ピクチャライブラリが空であるかどうかを検査します：

```
PICTURE LIBRARY LIST(alPicRef ; asPicName)
If (Size of array(alPicRef)=0)
   ALERT("ピクチャライブラリは空です")
Else
   ALERT("ピクチャライブラリは、"+String(Size of array(alPicRef))+"ピクチャを含んでいます。")
End if
```

3. 次の例は、ピクチャライブリをディスク上の任意の文書に書き出します：

```
PICTURE LIBRARY LIST($alPicRef ; $asPicName)
$vlNbPictures:=Size of array($alPicRef)
If ($vlNbPictures>0)
   SET CHANNEL(12 ; "")
   If (OK=1)
      $vsTag:="4DV6PICTURELIBRARYEXPORT"
      SEND VARIABLE($vsTag)
      SEND VARIABLE($vlNbPictures)
      Error:=0
      For($vlPicture ; 1 ; $vlNbPictures)
         $vlPicRef:=$alPicRef{$vlPicture}
         $vsPicName:=$asPicName{$vlPicture}
         GET PICTURE FROM LIBRARY(alPicRef{$vlPicture} ; $vgPicture)
         If (OK=1)
            SEND VARIABLE($vlPicRef)
            SEND VARIABLE($vsPicName)
            SEND VARIABLE($vgPicture)
         Else
            $vlPicture:=$vlNbPictures+1
            gError:=-108
         End if
      End for
      SET CHANNEL(11)
      If (gError#0)
         ALERT("ピクチャライブラリは書き出せませんでした。メモリを増やしてください。")
         DELETE DOCUMENT (Document)
      End if
   End if
Else
   ALERT("ピクチャライブラリは空です。")
End if
```

参照：GET PICTURE FROM LIBRARY, REMOVE PICTURE FROM LIBRARY, SET PICTURE TO LIBRARY

GET PICTURE FROM LIBRARY

GET PICTURE FROM LIBRARY (**ピクチャ参照番号 ; ピクチャ名**)

引数	**タイプ**		**説明**
ピクチャ参照番号	数値	→	ピクチャライブラリ内のピクチャ参照番号
ピクチャ名	ピクチャ変数	←	ピクチャライブラリ内のピクチャ

GET PICTURE FROM LIBRARYコマンドは、引数＜ピクチャ参照番号＞に渡された参照番号を持つピクチャライブラリ内のピクチャを引数＜ピクチャ＞内に返します。

参照番号を持つピクチャがない場合は、**GET PICTURE FROM LIBRARY**コマンドは＜ピクチャ＞はそのまま変わりません。

例：

1. 次の例は、参照番号がローカル変数 “ $vlPicRef ” 変数に格納されたピクチャを変数 “ vgMyPicture ” 変数に返します：

```
GET PICTURE FROM LIBRARY($vlPicRef ; vgMyPicture)
```

2. **PICTURE LIBRARY LIST**コマンドの例を参照してください

参照：PICTURE LIBRARY LIST、REMOVE PICTURE FROM LIBRARY、SET PICTURE TO LIBRARY

システム変数とシステムセット：

ピクチャライブラリが存在する場合、OKシステム変数に1が設定されます。そうでない場合は、0が設定されます。

エラー処理：

ピクチャに返すための十分なメモリがない場合、エラーコード -108が生成されます。エラー処理メソッドを使って、このエラーを受け取ることができます。

SET PICTURE TO LIBRARY

SET PICTURE TO LIBRARY (**ピクチャ** ; **ピクチャ参照番号 ; ピクチャ名**)

引数	タイプ		説明
ピクチャ	ピクチャ	→	新規ピクチャ
ピクチャ参照番号	数値	→	ピクチャライブラリ内のピクチャ参照番号
ピクチャ名	文字列	→	ピクチャの新しい名前

SET PICTURE TO LIBRARYコマンドは、新規ピクチャを作成するか、またはピクチャライブラリにあるピクチャを置き換えます。

このコマンドを呼び出す前に、下記の引数を渡します；

- 引数＜ピクチャ参照番号＞にピクチャ参照番号（1~32767の範囲）
- 引数＜ピクチャ＞にピクチャ自身
- 引数＜ピクチャ名＞ピクチャの名前（最大31バイト）

同じ参照番号を持つ既存のピクチャライブラリがある場合、そのピクチャの内容は置き換えられ、引数＜ピクチャ＞と＜ピクチャ名＞に渡された値を基にピクチャ名が変更されます。

引数＜ピクチャ参照番号＞に渡された参照番号を持つピクチャライブラリ図形がない場合、新規ピクチャがピクチャライブラリに追加されます。

4D Server：SET PICTURE TO LIBRARYコマンドは（ストアドプロシージャまたはトリガを含む）サーバマシン上で実行されるメソッドの中から使用することはできません。SET PICTURE TO LIBRARYコマンドをサーバマシン上で呼び出しても、無視され、何も行われません。

警告：デザインオブジェクト（階層リスト項目、メニュー項目など）は、ピクチャライブラリ図形を参照することができます。プログラムによってピクチャライブラリ図形を修正する際は、注意して使用する必要があります。

注：引数＜ピクチャ＞に空のピクチャを渡すか、または引数＜ピクチャ参照番号＞に負数またはスル値を渡すと、SET PICTURE TO LIBRARYコマンドは何も行いません。

例：

1. 次の例は、ピクチャライブラリの現在の内容が何であろうと、最初に一意のピクチャ参照番号を探すことによってピクチャライブラリに新規ピクチャを追加します：

```
PICTURE LIBRARY LIST($alPicRef ; $asPicNames)
Repeat
   $vlPicRef:=1+Abs(Random)
Until (Find in array($alPicRef ; $vlPicRef)<0)
SET PICTURE TO LIBRARY(vgPicture ; $vlPicRef ; "新規ピクチャ")
```

2. 次の例は、ピクチャライブラリの中に**PICTUTE LIBRARY LIST**コマンドの3番目の例で作成した（ディスク上の文書に格納された）ピクチャを読み込みます：

```
SET CHANNEL(10 ; "")
If (OK=1)
    RECEIVE VARIABLE ($vsTag)
    If ($vsTag="4DV6PICTURELIBRARYEXPORT")
        RECEIVE VARIABLE ($vlNbPictures)
        If ($vlNbPictures)
            For ($vlPicture ; 1 ; $vlNbPictures)
                RECEIVE VARIABLE ($vlPicRef)
                If (OK=1)
                    RECEIVE VARIABLE ($vlPicName)
                End if
                If (OK=1)
                    RECEIVE VARIABLE ($vgPicture)
                End if
                If (OK=1)
                    SET PICTURE TO LIBRARY ($vgPicture ; $vlPicRef ; $vlPicName)
                Else
                    $vlPicture:=$vlNbPictures+1
                    ALERT("このファイルは、ダメージを受けているみたいです。")
                End if
            End for
        Else
            ALERT("このファイルは、ダメージを受けているみたいです。")
        End if
    Else
        ALERT(Document+"ファイルは、ピクチャライブラリの書き出しファイルでは
                ありません。")
    End if
    SET CHANNEL(11)
End
```

参照：GET PICTURE FROM LIBRARY、PICTURE LIBRARY LIST、REMOVE PICTURE FROM LIBRARY

システム変数とシステムセット：

何も影響を与えません。

エラー処理：

ピクチャライブラリにピクチャを追加するための十分なメモリがない場合、エラーコード -108が生成されます。また、I / Oエラーが返される（例えば、ストラクチャファイルがロックされている等）点にも注意してください。エラー処理メソッドを使って、このエラーを受け取ることができます。

REMOVE PICTURE FROM LIBRARY

REMOVE PICTURE FROM LIBRARY(ピクチャ参照番号)

引数	タイプ		説明
ピクチャ参照番号	数値	→	ピクチャライブラリ図形の参照番号

REMOVE PICTURE FROM LIBRARYコマンドは、引数＜ピクチャ参照番号＞に渡した参照番号を持つピクチャをピクチャライブラリから消去します。

参照番号を持つピクチャがない場合は、このコマンドは何も行いません。

4D Server：REMOVE PICTURE FROM LIBRARYコマンドは（ストアドプロシージャまたはトリガを含む）サーバマシン上で実行されるメソッドの中から使用することはできません。REMOVE PICTURE FROM LIBRARYコマンドをサーバマシン上で呼び出しても、無視され、何も行われません。

警告：デザインオブジェクト（階層リスト項目、メニュー項目など）は、ピクチャライブラリ図形を参照することができます。プログラムによってピクチャライブラリ図形を修正する際は、注意して使用する必要があります。

例：

1. 次の例は、ピクチャライブラリから参照番号4444のピクチャを削除します：

```
REMOVE PICTURE FROM LIBRARY(4444)
```

2. 次の例は、ドル記号 ($)で始まる名前を持つピクチャをピクチャライブラリから削除します：

```
PICTURE LIBRARY LIST($alPicRef ; $asPicName)
For($vlPicture ; 1 ; Size of array($alPicRef))
   If ($asPicName{$vlPicture}="$@")
      REMOVE PICTURE FROM LIBRARY($alPicRef{$vlPicture})
   End if
End for
```

参照：GET PICTURE FROM LIBRARY、PICTURE LIBRARY LIST、SET PICTURE TO LIBRARY

サポートされるピクチャのフォーマット

以下の表に、MacintoshとWindowsプラットフォームにおいてサポートされるピクチャのフォーマットを要約します。

切り抜きと貼り付け：サポートされるフォーマット

	PICT	EMF	WMF	BITMAP
Macintosh	○	-	-	-
Windows	○	○ PicCommentに埋め込み	○ PicCommentに埋め込み	○ Macintosh PICTに変換

表示：サポートされるフォーマット

	PICT	QuickTime	embedded WMF	embedded EMF
Macintosh	○	○	×	×
Windows	○	○ NT & WIN 95 + QT 32ビット	○	○

ACI_Pack ReadPictureFile サポートされる形式 (注を参照)

	PICT	BMP	WMF	EMF	JPEG
Macintosh	○	○ Mac PICTに変換	×	×	○ Mac PICTに拡張
Windows	○	○ Mac PICTに変換	○ PicCommentに埋め込み	○ PicCommentに埋め込み	○ Mac PICTに拡張

注：ACI Packは、ACIの4Dプラグインで4th Dimensionパッケージに付属しています。

Apple QuickTime圧縮

AppleはQuickTimeを使用して、JPEGのような新たな圧縮技術をインプリメントします。Appleは、オリジナルのPICT使用に新たなopcodesを追加して、Macintoshアプリケーションがquicktime画像を何らの変更することなく扱えるようにしました。アプリケーションがシステムに埋め込まれたQuickTimeデータを含む画像を描画するよう指示すると、QuickTimeが存在する場合にはビットマップが拡張されて表示されます。QuickTimeがインストールされていなければ、QuickTimeのopcodeは無視されます。この技術はユーザに対して透過的です。また1メガバイトの画像が40キロバイトのPICTに格納でき、表示する前に拡張する必要がないのでメモリが最小限で済みます。

QuickTimeの圧縮タイプ

次に示すのは、QuickTime で扱うことのできる圧縮のタイプすべてとその簡単な説明です。

■ **写真圧縮 - JPEG（'jpeg'）：**Photo compressor　ピクチャの圧縮に Joint Photographic Experts Group（JPEG）アルゴリズムを採用します。JPEG は静止画を圧縮するための国際標準です。

■ **ビデオ（`rpza'）：**Video compressor　良質のピクチャを維持したまま高速な解凍を行うことができます。

■ **アニメーション（'rle '）：**Animation compressor　動画やコンピュータ処理のビデオデータを処理する場合に用います。

■ **なし （'raw '）：**Raw compressor　ピクチャのピクセルデプスを変換して、ピクチャの格納容量を減らします。

■ **グラフィックス（'smc '）：**Graphics compressor　「アニメーション」の改良版で、結果の品質は劣ります。

■ **コンパクト・ビデオ（'cdvs'）：**Compact video compressor　「ビデオ」圧縮に似ていますが、より高度な圧縮率と、良質のピクチャ、高速な再生を行うことができます。

注：「アニメーション」と「グラフィックス」タイプの機能は、「圧縮設定」ダイアログボックスの「品質」スライドバーによる設定が「最高」になっている場合にのみ有効です。

圧縮のタイプを指定するとき、タイプ名に含まれているスペースも忘れずに、引数＜方法＞に指定してください。圧縮方法が空の文字列の場合、ピクチャはロードされますが圧縮されません。

画像圧縮エラー

画像圧縮コマンドを使用するが、QuickTimeがシステムにインストールされていない場合、4th Dimensionはエラーコード-9955を返します。QuickTimeが生成する他のエラーも返されます。

これらのエラーは、**ON ERR CALL**でインストールされるエラー処理メソッドを使用して受け取ることができます。

Windows上でApple QuickTimeを使用する

■ Alturaおよび4th Dimensionはいずれも純正32ビットアプリケーションなので、32ビットのQuickTime for Windows (version 2.1.1 b50以上)をインストールする必要があります。

■ QuickTime for Windowsは、Windows 3.1をサポートしないので、QuickTime画像をWindows 3.1で表示する方法はありません。

■ この画像を表示するには、QuickTime for Windowsがディスク上の作業ファイルを使用する必要があります。QuickTime画像が画面に表示されるたびに、それが最初にディスクに書き込まれ、次に削除されます。通常、テンポラリファイルはキャッシュに残り、実際にディスクに書き込まれることはないので、操作が非常に高速に行われます。ただし、低速なPCの場合またはメモリが不足している場合、この操作も低速になる可能性があります。

■ QuickTime for Windowsは、QuickTime画像を表示するだけです。画像を圧縮することはできません。QuickTime圧縮を行う4th DimensionコマンドがWindows上で機能しないのはこのためです(将来も、QuickTime for Windowsが圧縮をサポートするまで機能しません)。Windowsで機能しないコマンドは以下のとおりです：

COMPRESS PICTURE、COMPRESS PICTURE FILE、LOAD COMPRESS PICTURE FROM FILE

SAVE PICTURE TO FILEコマンドはQuickTimeを使用しないので、Windows上でもMacintosh上と同様に機能します。これが、Photoshopのような高度グラフィックスアプリケーションによってWindows上でオープンできるMacintoshのPICTファイルを生成します。

印刷コマンド　38

この章では、「ルーチン」エディタの「Printing」テーマ内にある印刷コマンドについて説明します。レポートの印刷は、データベースの最も重要な作業です。この章のコマンドで、4th Dimensionの柔軟性の高いレポート機能を使用することができます。

ACCUMULATE
BREAK LEVEL
FORM FEED
Level
PAGE SETUP
PRINT LABEL
PRINT FORM
PRINT RECORD
PRINT SELECTION
PRINT SETTINGS
Printing page
REPORT
SET PRINT PREVIEW
Subtotal

PRINT SELECTIONコマンド、**PRINT FORM**コマンド、**PRINT LABEL**コマンドを使用して、「デザイン」モードで作成したフォームからレポートを作成することができます。また、フォームを作成しなくても、**REPORT**コマンドや**PRINT LABEL**コマンドを使用してレポートを作成することができます。

フォームを使用すれば、ほとんどのレポートをデザインすることができます。レポートにはグラフィックスを入れることもできます。また、レポートを構成する要素を調整するためのさまざまな方法が用意されています。フォームで印刷されるレポートは、フォームメソッドを実行できるので、大きな処理能力が得ることができます。

「クィックレポート」エディタや「ラベル」ウィザードで作成するレポートは、一般に単純なデザインになっています。ユーザはこれらのエディタを使用して、レポートを設計することができます。また、レポートのデザインをディスクに保存することもできます。これらのレポートはフォームを使用しませんので、フォームメソッドを実行することはできません。

REPORT、**PRINT SELECTION**、**PRINT FORM**の 3つの主要なレポートコマンドは、柔軟性の度合がそれぞれ異なります。一般に、コマンドの柔軟性が高いほど、レポート作成時にそのデザイン手腕が要求されます。

REPORTコマンドは、最も簡単なレポート作成コマンドです。このコマンドは、「ユーザ」モードで使用するものと同じ「クィックレポート」エディタを使用します。レポートのスタイルは、行とカラムで構成されます。これには、ヘッダ、フッタ、さまざまなフォントや書体、フォーマット、フォーミュラ、ブレーク、集計、計算、可変長のテキストを含む複数カラムなどがあります。レポートはグラフやASCIIファイルにも変換可能です。

PRINT SELECTIONコマンドは、最もよく使用される印刷コマンドです。これには、出力フォームを使用します。このフォームのデザインに制約はありません。このコマンドにもヘッダ、フッタ、さまざまなフォントや書体、フォーマット、フォーミュラ、ブレーク、集計、計算が使用されます。簡単なレポートは、フォームメソッドがなくても作成することはできますが、ほとんどの場合では、レポートを処理するためのフォームメソッドやオブジェクトメソッドを作成します。

PRINT FORMコマンドは、すべての印刷コマンドの中で最も柔軟性の高いものです。また、最もデザイン手腕が要求されるものです。これは、同じページに異なるフォームを配置することができます。さらに、印刷中の任意の時点で改ページ(フォーム・フィード)を挿入することができます。

PRINT LABELコマンドは、フォームを使用してラベルを印刷します。これは、高い柔軟性を持っています。「ユーザ」モードに用意された「ラベル」ウィザードを使用するので、必要に応じてラベルをデザインすることができます。このコマンドは、レコードを並列に印刷することができるため、特殊なレポートを作成する場合にも利用することができます。

すべての印刷コマンドは、カレントセレクションを印刷します。印刷前にセレクションをソートするのが一般的です。

ユーザはレポートを印刷する場合に、それを印刷するかどうかを画面上で選択することができます。4th Dimensionは、印刷中に、印刷されるカレントページと印刷状態を表示します。ユーザが画面上のレポートを“印刷”する場合に、ユーザは「プリント」ボタンをクリックして、カレントページを印刷することができます。

印刷を取り消す場合は、「プリント中止」ボタンを押します。ユーザがこのボタンをクリックするか、あるいはプリンタダイアログボックスを取り消して印刷を中止すると、システム変数OKに 0が代入されます。印刷がうまくいくと、システム変数OKに1が代入されます。

フォームレポートにおけるブレーク処理の生成

出力フォームのブレーク処理は、2つの方法で生成されます。1つは、**Subtotal**関数を使用する方法です。もう1つは、**BREAK LEVEL**コマンドと**ACCUMULATE**コマンドを使用する方法です。両方とも同じ結果を得られますが、それぞれに異なる利点を持っています。

Subtotal関数を使用したブレーク処理

Subtotal関数でブレーク処理を実行するためには、フォームメソッドまたはオブジェクトメソッドで使用しなければなりません。レポートを印刷する前に、4th Dimensionはフォームメソッドと**Subtotal**関数のオブジェクトメソッドを実行します。

ブレーク処理をオンにするために、**Subtotal**関数を実行する必要はありません。例えば、フッタ行の下のオブジェクトにオブジェクトメソッドを付随した場合は、決して印刷または実行されることはありません。実際に**Subtotal**関数が実行されない場合には、この関数に対する引数が有効である必要はありません。例えば、次の行は、ブレーク処理をオンにします：

```
x:=Subtotal (x)
```

Subtotal関数をブレーク処理を生成させるために使用した場合に、ブレークを行うものよりも1つ多いレベルでソートしなければなりません。例えば、レポート中に2つのソートレベルを必要とする場合は、ソートは3レベルで行います。

BREAK LEVELコマンドおよびACCUMULATEコマンドを使用したブレーク処理

ブレーク処理をオンにするために、**BREAK LEVEL**コマンドと**ACCUMULATE**コマンドを使用することもできます。この場合、レポートを印刷する前にこれらのコマンドの両方を実行しなければなりません。この方法を使用する場合、**Subtotal**関数は必要ありません。

またこの方法が使用される場合、1つ多いレベルでソートする必要はありません。ただし、ブレークする数と同じ数のレベルでソートする必要はあります。

2つの方法の比較

ブレーク処理を開始するために**Subtotal**関数を使用する主な利点は、レポートの印刷前にメソッドを実行する必要がないことです。これは、「ユーザ」モードにおいては特に有効です。「ユーザ」モードでレポートを印刷するための行程は、次のようになります。

1. 印刷するレコードを選択する。

2. レコードのソートを行う。1つ多いレベルでソートを行う。

3.「ファイル」メニューから「プリント...」を選択する。

4th Dimensionは、フォームメソッドとオブジェクトメソッドを調べ、**Subtotal**関数を検索します。そして、ブレーク処理をオンにし、レポートを印刷します。しかし、ブレーク処理に**Subtotal**関数を使用することについては、次の2つの欠点があります。

■ **Subtotal**関数は、コンパイルされたメソッドでブレーク処理を生成することができない。

■ 1つ多いレベルでソートしなければならないため、レコードが多いと、処理に時間がかかる。

BREAK LEVELコマンドと**ACCUMULATE**コマンドによるブレーク処理の生成方法は、レポートの作成にメソッドを使用する場合に有効です。この方法を使用したレポートの印刷プロセスは、一般的に次のようになります。

1. 印刷するレコードを選択する。

2. レコードをソートする。少なくともブレークと同数のレベルでソートする。

3. BREAK LEVELコマンドとACCUMULATEコマンドを実行する。

4. レポートを印刷する。

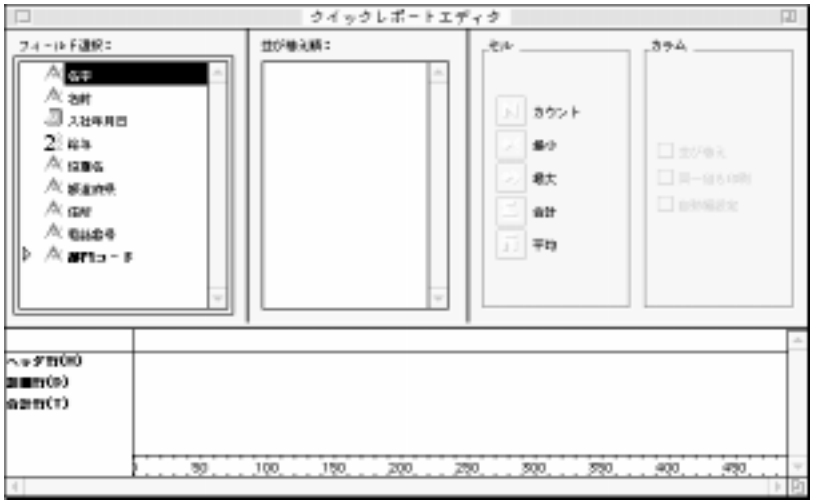
コンパイルされたメソッドでブレーク処理を生成するには、**BREAK LEVEL**コマンドと**ACCUMULATE**コマンドを使用しなければなりません。ただし、**Subtotal**関数は、フォーム上の値を表示するために今までどおり必要です。

REPORT

REPORT ({テーブル} ; 文書 ; {*})

引数	タイプ	説明
テーブル	テーブル	印刷するテーブル
文書	文字列	クィックレポートの文書
*		プリンタダイアログボックスの表示取り消し

REPORTコマンドは、＜テーブル＞に対するレポートを下図に示した「クィックレポート」エディタを使用して印刷します。



＜文書＞は、「クィックレポート」エディタで作成され、ディスクに保存された文書です。「クィックレポート」エディタ内の「ファイル」メニューから「保存」または「新規保存」を選択してレポート文書を保存します。これはレポートの形式を格納するだけで、印刷されるレコードを格納するわけではありません。＜文書＞に対して空の文字列("")を指定した場合に、**REPORT**コマンドは「ファイルを開く」ダイアログボックスを表示し、ユーザは、印刷するレポートの形式を選択することができます。レポートの形式が選択されると「用紙設定」ダイアログボックスが表示されます。オプション引数にアスタリスク(*)を指定すると、プリンタダイアログボックスは表示されません。レポートは、その後で印刷されます。

＜文書＞に存在しない文書名を指定した場合は、「クィックレポート」エディタが表示されます。ユーザは、「クィックレポート」エディタでカスタムレポートを作成することができます。「クィックレポート」エディタが表示されると、メニューバーは「ユーザ」モードのエディタと同じ「ファイル」、「編集」、「フォント」、「書体」の 4つのメニューを表示します。「クィックレポート」エディタを使用したレポートの作成に関する詳細は、『4th Dimension / 4D Firstユーザリファレンス』を参照してください。

▼次の例は、ユーザがデータベースを検索し、レポートを印刷するクィックレポート文書の選択のためのダイアログボックスを開きます：

```
QUERY ([従業員])                    ` レコードを検索する
REPORT ([従業員] ; "")              ` ユーザがレポートを生成できるようにする
```

PAGE SETUP

PAGE SETUP ({テーブル} ; フォーム)

引数	タイプ	説明
テーブル	テーブル	フォームを含むテーブル
フォーム	文字列	ページセットアップに使用されるフォーム

PAGE SETUPコマンドは、＜フォーム＞に格納される用紙設定情報を設定します。このコマンドは、**PRINT SELECTION**コマンドの前に実行します。**PRINT SETTINGS**コマンドは**PRINT FORM**コマンドの前に使用します。

▼ 次の例は、フォーム " レポート " に格納される用紙設定情報を設定します：

```
PAGE SETUP ([従業員] ; "レポート")
```

PRINT SELECTION

PRINT SELECTION({テーブル} ; {*})

引数	タイプ	説明
テーブル	テーブル	印刷するテーブル
*		プリンタダイアログボックスの表示取り消し

PRINT SELECTIONコマンドは、テーブルのカレントセレクションを印刷します。レコードは、カレント出力フォームを介して印刷します。**PRINT SELECTION**コマンドは、「ユーザ」モードの「プリント...」メニューコマンドと同じ動作を実行します。

デフォルトでは、**PRINT SELECTION**コマンドは、印刷前にプリンタダイアログボックスを表示します。オプション引数のアスタリスク(*)を使用して、ダイアログボックスの表示を取り消すことができます。ユーザがプリンタダイアログボックスの「キャンセル」ボタンをクリックすると、このコマンドが取り消され、レポートは印刷されません。

アスタリスク(*)を設定した場合は、フォームを作成した時点のページ設定情報でレポートが印刷されます。また、**PAGE SETUP**コマンドを使用してページ設定を行うことができます。

印刷中に、出力フォームのフォームメソッドとオブジェクトメソッドを実行します。つまり、ヘッダを印刷する場合はHeaderフェーズ、各レコードを印刷する場合はBeforeフェーズとDuringフェーズ、ブレークエリアを印刷する場合はBreakフェーズ、フッタを印刷する場合はFooterフェーズが発生します。

PRINT SELECTIONコマンドが最初のヘッダを印刷しているかどうかは、Headerフェーズで**Before selection**関数を判定することによって調べることができます。またFooterフェーズで**End selection**関数を判定することによって、最後のフッタかどうかをチェックすることができます。

PRINT SELECTIONコマンドが異なる2つのプロセス内で同時に使用される場合は、2番目のプロセスからの印刷は最初のプロセスからの印刷が終了するまで待つことになります。

PRINT SELECTIONコマンドを使用してセレクションの小計を印刷するには、そのセレクションを最初にソートしなければなりません。また、レポートの各ブレークエリアに小計を変数に代入するオブジェクトメソッドを持つ変数を設定する必要もあります。変数に値を代入する際に、**Sum**関数や**Average**関数のような統計関数と算術関数を使用することもできます。**PRINT SELECTION**コマンドと一緒に**FORM FEED**コマンドを使用してはいけません。

▼ 次の例は、最初に[従業員]テーブルのすべてのレコードを選択します。次に**DISPLAY SELECTION**コマンドを使用して、すべてのレコードを表示し、そのレコードを**PRINT SELECTION**コマンドで印刷します：

```
ALL RECORDS ([従業員])              ` 全レコードを選択
DISPLAY SELECTION ([従業員] ; *)    ` レコードを表示
PRINT SELECTION ([従業員])          ` ユーザが選択したレコードを印刷
```

BREAK LEVEL

BREAK LEVEL (レベル ; {ページブレーク})

引数	タイプ	説明
レベル	数値	ブレークレベルの数
{ページブレーク}	数値	ページブレークを行うブレークの数

BREAK LEVELコマンドは、レポート中のブレークの数を指定します。

レポートに対してブレーク処理をオンにするための方法が2つあります。これらの方法に関しては、前述の「フォームレポートにおけるブレーク処理の生成」の節を参照してください。

ブレーク処理を行うレポートを印刷する前に、**BREAK LEVEL**コマンドと**ACCUMULATE**コマンドを実行しなければなりません。これらのコマンドはレポートに対するブレーク処理を生成します。

＜レベル＞は、ブレーク処理を実行するレベルのことです。少なくとも同数のレベルでレコードをソートしなければなりません。ブレークレベルよりも多いレベルでソートすると、これらのレベルはソートされたものとして印刷されますが、ブレークに対しての意味は持ちません。

生成される各ブレークレベルは、フォーム中の対応するブレークエリアやヘッダエリアを印刷します。フォーム中のブレークエリアは、＜レベル＞の数だけ存在しなければなりません。フォーム中により多くのブレークエリアがある場合、それらは無視され、印刷されません。

2番目の(オプションの)引数＜ページブレーク＞は、印刷中にページブレークを発生させるために使用します。

▼ 次の例は、2つのブレークレベルを持つレポートを印刷します。このセレクションは 4つのレベルに対してソートされますが、**BREAK LEVEL**コマンドは 2つのレベルだけにブレークを指定します。[従業員]給与フィールドは**ACCUMULATE**コマンドで累計されます：

```
` 4つのレベルでソート
ORDER BY ( [ 従業員] ; [従業員]部門 ;> ; [従業員]役職名 ; > ; [ 従業員]名字 ; > ;
[従業員]名前 ; >)
` 部門と役職名のフィールドでブレーク処理を行う
BREAK LEVEL (2)
ACCUMULATE ([従業員]給与)                ` 給与の累計
OUTPUT FORM ([従業員] ; "部門別給与")     ` 出力フォームの設定
PRINT SELECTION ([従業員])               ` レポートの印刷
```

ACCUMULATE

ACCUMULATE (データ1 {;...; データN})

引数	タイプ	説明
データ	フィールドまたは変数	累計する数値フィールドまたは変数

ACCUMULATEコマンドは、レポート中で累計するフィールドまたは変数を指定します。

レポートに対してブレーク処理をオンにするための方法が2つあります。これらの方法に関しては、前述の「フォームレポートにおけるブレーク処理の生成」の節を参照してください。ブレーク処理を行うレポートを印刷する前に、**BREAK LEVEL**コマンドと**ACCUMULATE**コマンドを実行しなければなりません。これらのコマンドはレポートに対するブレーク処理を生成します。

レポート内の数値フィールドまたは変数に対する小計を求める場合は、**ACCUMULATE**コマンドを使用します。**ACCUMULATE**コマンドは、4th Dimensionに対して、引数<データ>に対する小計を記憶するように指示します。小計は**BREAK LEVEL**コマンドで指定された各ブレークレベルに対して累計されます。

PRINT SELECTIONコマンドを使ってレポートを印刷する前か、あるいは「ユーザ」モードで「プリント...」メニューコマンドを選択する前に、**ACCUMULATE**コマンドを実行します。フォームメソッドかオブジェクトメソッドで**Subtotal**関数を使用して、引数<データ>の1つの小計を求めます。

ACCUMULATEコマンドの使用例は、前ページ**BREAK LEVEL**コマンドの例を参照してください。

Subtotal

Subtotal (データ ; {ページブレーク}) → 数値

引数	タイプ	説明
データ	フィールド	小計を求める数値フィールドまたは変数
ページブレーク	数値	ページブレークを発生させるブレークレベル

Subtotal関数は、現在または最後のブレークレベルに対するデータの小計を返します。**Subtotal**関数は、**PRINT SELECTION**コマンドで印刷する場合と、「ユーザ」モードで「プリント...」メニューコマンドを使用して印刷を行う場合にのみ機能します。データのタイプは実数、整数、倍長整数のいずれかでなければなりません。印刷するフォームのブレークエリアの変数には、**Subtotal**関数の結果を代入します。

レポートに対してブレーク処理をオンにするための方法が2つあります。これらの方法に関しては、前述の「フォームレポートにおけるブレーク処理の生成」の節を参照してください。

フォームメソッドまたはフォームのオブジェクトメソッドに**Subtotal**関数を記述しなければなりません。4th Dimensionは印刷前にフォームメソッドとオブジェクトメソッドをチェックし、ブレーク処理を開始します。

Subtotal関数の2番目の引数(オプション)で印刷中にページブレークを行います。ブレークレベルが 0の場合は、**Subtotal**関数はページブレークを行いません。ブレークレベルが1の場合には**Subtotal**関数はページブレークを行い、ブレークレベルが2の場合では、**Subtotal**関数は1と2の各レベルに対してページブレークを行います。

レベルｎに対してブレークを行いたい場合、ｎ+1のレベルでカレントセレクションをソートしなければなりません。これによって、最終レベルのフィールドまでソートすることにより、思わぬブレークが発生することを防ぎます。最終レベルのソートフィールドでブレークを行う場合は、そのフィールドを2回ソートします。

▼ 次の例は、フォームのブレークエリア(B0、B0マーカの上のエリア)のオブジェクトメソッドです。変数“ v給与 ”は、ブレークエリアにあります。このブレークレベルが発生すると、変数に“ 給与 ”フィールドの小計が代入されます：

```
v給与:=Subtotal (給与)
```

Level

Level → 数値

Level関数は、カレントブレークまたはヘッダのレベルを調べるために使用します。この関数は、実行サイクルのIn BreakフェーズまたはIn Headerフェーズのレベル数を返します。

レベル0は、印刷する最後のレベルです。合計を印刷するのに適しています。**Level**関数は、最初のソートフィールドのブレークを印刷するときに1を返し、2番目ソートフィールドでブレークを印刷するときに2を返します。

前述の**In header**関数の例を参照してください。

Printing page

Printing page → 数値

Printing page関数は、印刷中のページ番号を返します。この関数は、**PRINT SELECTION**コマンドまたは「ユーザ」モードの「プリント...」メニューコマンドの選択によって印刷する場合にのみ使用することができます。

▼ 次の例は、両面印刷フォーマットのレポートにページ番号を設定します。ページ番号の位置を変更するために、フォームはページ番号を表示する変数を2つ持っています。変数 “ v左 ” は、偶数のページ番号を印刷します。変数 “ v右 ” は、奇数のページ番号を印刷します。この例は、偶数ページを判定し適切な変数に値を代入します：

```
If ((Printing page % 2) = 0)      ` 印刷中のページを2で割った時の余りが0の場合
  v左:=String (Printing page)     ` 左ページの番号を設定
  v右:=""                         ` 右ページの番号をクリア
Else
  v右:=String (Printing page)     ` 右ページの番号を設定
  v左:=""                         ` 左ページの番号をクリア
End if
```

PRINT FORM

PRINT FORM ({テーブル} ; フォーム)

引数	タイプ	説明
テーブル	テーブル	印刷するテーブル
フォーム	文字列	印刷するフォーム

PRINT FORMコマンドは、カレントフィールドの値を変数とともに＜フォーム＞に印刷します。このコマンドは、フォームのディテイルエリア(ヘッダ行とディテイル行の間のエリア)だけを印刷します。通常は、印刷のプロセスをメソッドで完全に制御する必要のある非常に複雑なレポートを印刷するために使用します。**PRINT FORM**コマンドはレコード処理、ブレーク処理、フォームフィード(改ページ)処理、ヘッダ処理、フッタ処理を全く行いません。これらの処理は、すべてデザイナが行います。**PRINT FORM**コマンドは固定された大きさの枠のなかでフィールドや変数を印刷します。

PRINT FORMコマンドは、フォームの印刷の後にフォームフィード(改ページ)を行いませんので、同じページに異なるフォームを容易に配置することができます。**PRINT FORM**コマンドは、異なるテーブルや異るフォームを含む複雑な印刷処理を完全に制御します。フォームフィード(改ページ)を強制的に行うには**FORM FEED**コマンドを使用してください。

PRINT FORMコマンドを使用する場合、プリンタダイアログボックスは表示されません。「デザイン」モードでレポートのフォームを作成した時点で、「ファイル」メニューから「用紙設定...」を選択して印刷形式を設定します。プリンタダイアログボックスを表示させる場合は、**PRINT FORM**コマンドを実行する前に**PRINT SETTINGS**コマンドを使用してください。

PRINT FORMコマンドは、メモリ中にそれぞれ印刷するページを作成します。各ページはメモリのページがいっぱいになると印刷されます。**PRINT FORM**コマンドの使用後の最後のページの印刷を確実に行うためには、**FORM FEED**コマンドで終了しなければなりません。そうでないと、最後のページはメモリ中に残り印刷されません。

サブフォームや外部ルーチンエリアは、**PRINT FORM**コマンドでは印刷できません。

PRINT FORMコマンドは、フォームメソッドのBeforeフェーズとDuringフェーズのみを実行します。

▼次の例は、単純な**PRINT SELECTION**コマンドをシミュレートします。ブレーク処理はありません。このレポートは、小切手レジスタに対するものです：

レコードが小切手用か預金用であるかによって2種類のフォーム内の1つを使用します：

```
QUERY ([レジスタ])                                    ` レコードの選択
ORDER BY ([レジスタ])                                 ` レコードのソート
PRINT SETTINGS                                        ` 用紙の設定
For ($i ; 1 ; Records in selection ([レジスタ]))       ` 選択した全レコードのループ
 If ([レジスタ]タイプ="小切手")` “タイプ”フィールドが"小切手"の場合
   PRINT FORM ([レジスタ] ; "小切手印刷")              ` 小切手の印刷
 Else ` “タイプ”フィールドが"小切手"でない場合
   PRINT FORM ([レジスタ] ; "預金印刷")                ` 預金の印刷
 End if
 NEXT RECORD ([レジスタ])                             ` 次のレコードに移動
End for
FORM FEED                                             ` 最後のページを印刷
```

PRINT SETTINGS

PRINT SETTINGS

PRINT SETTINGSコマンドはプリンタダイアログボックスを表示します。最初に「用紙設定」ダイアログボックスを表示し、その次に「印刷設定」ダイアログボックスを表示します。ユーザが両方のダイアログボックスで「OK」ボタンをクリックすると、システム変数OKに1が代入されます。それ以外の場合は、システム変数OKに 0が代入されます。**PRINT SETTINGS**コマンドまたは**PAGE SETUP**コマンドは、**PRINT FORM**コマンドの前に実行しなければなりません。**PRINT SETTINGS**コマンドは、**PRINT SELECTION**コマンドや**PRINT LABEL**コマンドに対しては全く効果がありません。

「印刷設定」ダイアログボックスには、「プレビュー（Macintosh版では、スクリーンへプリント）」チェックボックスがあります。このチェックボックスでレポートを画面に印刷するように指定することができます。

前述の**PRINT FORM**コマンドの例を参照してください。

FORM FEED

FORM FEED
FORM FEED ({ * })
FORM FEED ({ > })

FORM FEEDコマンドは、プリンタへ送出したページを排出、印刷します。**FORM FEED**コマンドは、**PRINT FORM**コマンドとともに使用し、ページブレークを強制し、最終ページを印刷します。**FORM FEED**コマンドは、**PRINT SELECTION**コマンドとともに使用しないでください。ページブレークを行うためには、**Subtotal**関数または**BREAK LEVEL**コマンドにオプション引数を使用してください。

＜＊＞と＜＞＞引数は両方とも省略できます。

＜＊＞引数により、**PRINT FORM**コマンドによって開始したプリントジョブをキャンセルすることができます。このコマンドを実行すると、進行中のプリントジョブが直ちに中止されます。

＜＞＞引数は、**FORM FEED**コマンドの振る舞いを変更します。この形式は 2種類の効果を持ちます。

■ **FORM FEED**コマンドが引数なしで再度実行されるまで、プリントジョブの開始を止めます。

■ プリントジョブに優先権を与えます。プリントジョブが終了するまで、他のプリントは行われません。

2番目のオプションは、スプールされるプリントジョブとともに使用すると、特に有効です。＜＞＞引数により、プリントジョブは 1 つのテーブルにスプールされます。これは、プリント時間を減少させます。

前述の**PRINT FORM**コマンドの例を参照してください。

PRINT LABEL

PRINT LABEL ({テーブル} ; {*})

引数	タイプ	説明
テーブル	テーブル	印刷するテーブル
*		プリンタダイアログボックスの表示取り消し

PRINT LABEL ({テーブル} ;ラベル文書)

引数	タイプ	説明
テーブル	テーブル	印刷するテーブル
ラベル文書	文字列	ディスクラベル文書の名前

PRINT LABELコマンドには、2つの形式があります。

第1の形式は、カレント出力フォームを使用して、ラベルとして＜テーブル＞のカレントセレクションを印刷します。ラベルにサブテーブルを印刷することはできません。ラベルのフォーム作成に関する詳細は、『4th Dimension / 4D First デザインリファレンス』を参照してください。

第1の形式を使用する場合、**PRINT LABEL**コマンドは、印刷前にプリンタダイアログボックスを表示します。オプション引数のアスタリスク(*)を使用して、ダイアログボックスの表示を取り消すことができます。ユーザがプリンタダイアログボックスのいずれかで「キャンセル」ボタンをクリックすると、このコマンドは取り消され、ラベルは印刷されません。4th Dimensionは、印刷中に出力フォームのフォームメソッドとオブジェクトメソッドを実行します。BeforeフェーズとDuringフェーズが各レコードの印刷時に発生します。

コマンドの第2の形式は、「ラベル」エディタを使用してラベルを印刷します。

このコマンドの第2の形式を使用する場合には、ラベルは＜ラベル文書＞で定義されたラベル設定情報に従って印刷されます。＜ラベル文書＞が空のストリング("")の場合に、「ファイルを開く」ダイアログボックスを表示します。ユーザは、ここで、ラベル設定情報を定義することができます。

PRINT LABELコマンドが異なる2つのプロセス内で同時に使用されると、2番目のプロセスからのラベル印刷は最初のプロセスからの印刷が終了するまで待つことになります。

ラベルエディタを使ったラベル作成に関する詳細は、『4th Dimension / 4D First ユーザリファレンス』を参照してください。

▼ 次の例は、**PRINT LABEL**コマンドの第1の形式の使用方法について説明しています。これは、フォーム“ ラベル出力 ”を使用して、ラベルを印刷します。この例では、2つのメソッドを使用します。最初のものは、出力フォームをセットし、ラベル印刷を実行するプロジェクトメソッドです：

```
` プロジェクトメソッド
vCR:=Char (13)                                  ` 改行を代入
ALL RECORDS ([住所])                            ` すべてのレコードを選択
OUTPUT FORM ([住所] ; "ラベル出力")              ` 出力フォームの設定
PRINT LABEL ([住所])                            ` ラベルの印刷
OUTPUT FORM ([住所] ; "出力1")                   ` デフォルト出力フォームの設定
```

▼ 次の例は、フォーム“ ラベル出力 ”のフォームメソッドです。このフォームは、各フィールドの内容を連結した結果を格納するための1つの変数“ vラベル ”を含みます。変数“ vラベル ”は、フィールド“ 住所2 ”が空白の場合、メソッドで読み飛ばします。(「ラベル」ウィザードを使用しても、この処理が自動的に実行される点に注目してください。) フォームメソッドは、各レコードに対して1つずつラベルを作成します：

```
` フォーム“ ラベル出力 ”のフォームメソッド
` “ vラベル ”はフォーム上の変数
`郵便番号、都道府県、市町村、住所、名前を連結する
vラベル:=住所1
If (住所2 # "")                                 ` 住所2が空白でない場合
  vラベル:=vラベル+"  "+住所2                    ` vラベルに住所2を連結する
End if
vラベル:=郵便番号+vCR+都道府県+"  "+市町村+vCR+vラベル+vCR+名前1+"  "+名前2
```

▼ 次の例は、**PRINT LABEL**コマンドの第2の形式の使用方法について説明しています。これは、文書ファイル“ 住所ラベル ”に格納されている「ラベル」ウィザードの設定情報を使用して印刷します：

```
PRINT LABEL ([住所 ]; "住所ラベル")
```

PRINT RECORD

PRINT RECORD (**{テーブル} ; {*}**)

引数	タイプ	説明
テーブル	テーブル	カレントレコードを印刷するテーブル
*	ピクチャ	プリンタダイアログボックスの表示取り消し

このコマンドは＜テーブル＞のカレントレコードを、カレントセレクションを変更せずに印刷します。カレント出力フォームが印刷に用いられます。

PRINT RECORDコマンドを使って組み込みフォームや外部ルーチンエリアを印刷することができます。これは、**PRINT FORM** コマンドではできない機能です。

注：レコードに対して行われた修正が保存されていない場合、修正前のフィールド値ではなく、修正後の値が印刷されます。

▼ 次の例では、カレントレコードを印刷します。このコードは入力フォームのボタン内に記述されています。ユーザがそのボタンをクリックすると、レコードは指定した出力フォームで印刷されます：

```
OUTPUT FORM ([テーブル1] ; "レコード印刷")
PRINT RECORD ([テーブル1] ; *)
OUTPUT FORM ([テーブル1] ; "出力")
```

SET PRINT PREVIEW

SET PRINT PREVIEW (プレビュー)

引数	タイプ	説明
プレビュー	ブール	True = スクリーン上でプレビューする False = プレビューしない

このコマンドは、「プリント」ダイアログボックスの「プレビュー（Macintosh版では、スクリーンへプリント）」チェックボックスのオン／オフをメソッドで切り替えるためのものです。コマンドに「True」を渡すと、「プレビュー（スクリーンへプリント）」チェックボックスはチェックされ、「False」を渡すと、チェックが外されます。

この設定はローカルなプロセスであり、他のプロセスの印刷やユーザに影響を与えません。

▼ 次の例は、検索結果を表示するために、まず「プレビュー（スクリーンへプリント）」チェックボックスをオンにし、それからオフに切り替えます：

```
QUERY ([顧客])
If (OK =1)
   SET PRINT PREVIEW (True)
   PRINT SELECTION ([顧客] ; *)
   SET PRINT PREVIEW (False)
End if
```

プロセス（通信）コマンド　39

この章では、「ルーチン」エディタの「Process (Communications)」テーマ内にあるプロセス（通信）コマンドについて説明します。

CALL PROCESS
CLEAR SEMAPHORE
GET PROCESS VARIABLE
Semaphore
SET PROCESS VARIABLE
VARIABLE TO VARIABLE

Semaphore

Semaphore (セマフォ) → ブール

引数	タイプ	説明
セマフォ	文字列	設定するセマフォ

セマフォは、マルチユーザユーザデータベースのワークステーション(各ユーザのコンピュータ)間の簡単なメッセージです。セマフォは、単に存在したり存在しなかったりするだけです。セマフォを作成したり、その存在の有無を調べることにより、ワークステーション間でのメソッドの通信が可能になります。

Semaphore関数は、＜セマフォ＞が存在する場合に“ True ”を返します。**Semaphore**関数は＜セマフォ＞が存在しない場合にセマフォを作成し、“ False ”を返します。同時には、1人のユーザしかセマフォを作成することはできません。セマフォが“ False ”を返すということは、セマフォが存在しなかったことを意味すると同時に、呼び出したプロセスに対して新たにそれが設定されたことを意味します。

Semaphore関数は、セマフォが設定されていなければ“ False ”を返します。また、呼び出したプロセスがすでにセマフォを設定している場合も、“ False ”を返します。セマフォは15文字以内に制限されています。これより長いセマフォを指定すると、短縮されたセマフォでテストされます。

4th Dimensionバージョン6には、ローカルセマフォとグローバルセマフォの2種類があります。

ローカルセマフォは、同じワークステーション上のすべてのプロセスでアクセスすることができます(同一ワークステーション上に限られます)。ローカルセマフォは、セマフォ名の先頭にドル記号($)を付けて作成します。また、ローカルセマフォは、同一ワークステーション上で実行しているプロセス間で互いに処理を監視します。例えば、ローカルセマフォを使用して、シングルユーザデータベース、またはワークステーション上のすべてのプロセスで共用するインタープロセス配列のアクセスを監視します。

グローバルセマフォは、すべてのユーザとそのプロセスからアクセスすることができます。グローバルセマフォはマルチユーザデータベースのユーザ間で互いに処理を監視するために用います。

グローバルセマフォとローカルセマフォは理論的には同じものです。違いは、その範囲にあります。4D Serverでは、グローバルセマフォはすべてのクライアントで実行しているすべてのプロセス間で共用します。ローカルセマフォは、それが作成されたクライアントと同じクライアントで実行しているプロセス間でのみ共用します。

4th Dimensionでは、ユーザは一人だけですので、グローバルセマフォの範囲もローカルセマフォの範囲も同じになります。ただし、シングルとマルチの両方の形でデータベースを使用する場合は、用途によってグローバルセマフォとローカルセマフォを使い分けてください。

セマフォはレコードのアクセスの保護目的には使用しません。これは4th Dimensionと4D Serverが自動的に行います。セマフォは、複数のユーザが同じ処理を同時に実行するのを防ぐために用います。

▼ 次の例では、2人のユーザが“製品”ファイルの価格を一括更新するのを防ぎます。以下のメソッドではセマフォを用いて、これを実現しています：

```
If (Semaphore ("価格更新"))                    ` セマフォの作成を試みる
   ALERT ("他のユーザが既に価格を更新しています。再度トライしてください。")
Else
   DoUpdatePrices                              ` ここで価格を更新する
   CLEAR SEMAPHORE ("価格更新")                ` セマフォを消去する
End if
```

▼ 次の例は、ローカルセマフォの使い方です。複数のプロセスを持つデータベースでは、“To Do”リストを管理する必要があります。このリストはファイルではなく、インタープロセス配列で管理します。セマフォを使って同時にアクセスされるのを防ぎます。このような場合には、“To Do”リストは自分だけのものですので、ローカルセマフォで十分です：

インタープロセス配列は、「On Startup」データベースメソッドで初期化します：

```
ARRAY TEXT (<>ToDoリスト ; 0)                  ` 最初To Doリストは空
```

“To Do”リストに項目を追加するメソッドを次に示します：

```
C_TEXT ($1)                                    ` コマンドに渡されるパラメータ
While (Semaphore ("$AccessToDoList"))          ` 必要なら待つ
End while
INSERT ELEMENT (<>ToDoリスト ; 1 + Size of array (<>ToDoリスト))
<>ToDoリスト{Size of array (<>ToDoリスト)}:=$1  ` 値を保存
CLEAR SEMAPHORE ("$AccessToDoList")            ` セマフォをクリア
```

このメソッドは、任意のプロセスから呼び出すことができます。

CLEAR SEMAPHORE

CLEAR SEMAPHORE (セマフォ)

引数	タイプ	説明
セマフォ	文字列	消去するセマフォ

CLEAR SEMAPHOREコマンドは、**Semaphore**関数で設定された＜セマフォ＞を消去します。

一般に、作成されたすべてのセマフォは消去するべきです。セマフォが消去されない場合には、セマフォを作成したプロセスが終了するまで、作成されたセマフォはメモリ上に残ります。プロセスは作成したセマフォしか消去することはできません。セマフォを作成していないプロセス内からセマフォを消去しようとしても、何も行いません。

前出の**Semaphore**関数の例を参照してください。

CALL PROCESS

CALL PROCESS (プロセス)

引数	タイプ	説明
プロセス	数値	プロセス番号

CALL PROCESSコマンドは、＜プロセス＞の最前面に表示されたフォームの**Outside call**フェーズを呼び出します。＜プロセス＞がウインドウを持たないか、あるいはフォームが表示されていない場合には何も行いません。

インタープロセス変数表示の更新

メソッドの実行速度が遅くならないように、4th Dimensionはインタープロセス変数が変更されるたびに再描画することはしません。もし、プロセス参照番号の代わりに-1を**CALL PROCESS**コマンドの引数＜プロセス＞に渡すと、プロセス内のウインドウに表示されるすべてのインタプロセス変数は更新されます。4D Serverでは、クライアントから実行される**CALL PROCESS(-1)**は、そのクライアント上で実行されているプロセスのインタープロセス変数のみを更新します。

Outside call関数の説明を参照してください。

GET PROCESS VARIABLE

GET PROCESS VARIABLE (プロセス ; ソース変数 ; 送信先変数 { ; ソース変数2 ; 送信先変数2 ; ...; ソース変数N ; 送信先変数N})

引数	タイプ		説明
プロセス	数値	→	**ソースプロセス番号**
ソース変数	変数	→	**ソース変数**
送信先変数	**変数**	←	**送信先変数**

GET PROCESS VARIABLEコマンドは、引数＜プロセス＞で渡す番号のソースプロセスから＜ソース変数＞(ソース変数2など)プロセス変数を読み込み、その現在の値を現在のプロセスの＜送信先変数＞(送信先変数など)変数に返します。

それぞれのソース変数は、変数、配列、配列要素のいずれかを指定できます。ただし、このセクションで後述する制限事項を参照してください。

＜ソース変数＞と＜送信先変数＞変数の組では、この2つの変数はみな互換性のあるタイプである必要があり、互換性がない場合には、値を取得しても意味がなくなります。

現在のプロセスはソースプロセスの変数を「のぞき見」しています。ソースプロセスは別のプロセスが自分の変数のインスタンスを読み込んでいることについては何も警告されません。

制限事項：

GET PROCESS VARIABLEコマンドは、ソース変数としてインタープロセス変数またはローカル変数は受け付けません。

言い換えると、送信先変数は、インタープロセス変数、プロセス変数、ローカル変数になれます。この値は変数だけで「受け取る」ことができ、フィールドで受け取ることはできません。

GET PROCESS VARIABLEコマンドは、次のタイプ以外の任意のソースプロセス変数を受け付けます。

- ポインタ
- ポインタの配列
- 2次元配列

ソースプロセスは、ユーザプロセスである必要があります。カーネルプロセスは、ソースプロセスにはなれません。ソースプロセスが存在しない場合には、エラーが生成されます。**ON ERR CALL**コマンドでインストールされたエラー処理メソッドを使用すると、このエラーを受け取ることができます。

注：インタープリタモードでは、ソース変数が存在しない場合には未定義値が返されます。これをType関数を使って検出し、対応する送信先変数をテストすることができます。

例：
1. 次のコードは、番号が$vlProcessであるプロセスのテキスト変数「vtCurStatus」の値を読み込み、その値を現在のプロセスのプロセス変数「vtInfo」に返します：

```
GET PROCESS VARIABLE ($vlProcess ; vtCurStatus ; vtInfo)
```

2. 次のコードは上記の例と同じことをしますが、現在のプロセスで実行しているメソッドに対してローカル変数「$vtInfo」に値を返します：

```
GET PROCESS VARIABLE ($vlProcess ; vtCurStatus ; $vtInfo)
```

3. 次のコードは上記の例と同じことをしますが、現在のプロセスの「vtCurStatus」変数に値を返します：

```
GET PROCESS VARIABLE ($vlProcess ; vtCurStatus ; vtCurStatus)
```

注：最初のvtCurStatusは、ソースプロセスにある変数のインスタンスを示しています。2番目のvtCurStatusは現在のプロセスにある変数のインスタンスを示しています。

4. 次の例は、$vlProcessで示されるプロセスのプロセス配列の要素を順番に読み込みます：

```
GET PROCESS($vlProcess ; vl_IPCom_Array ; $vlSize)
For ($vlElem ; 1 ; $vlSize)
   GET PROCESS VARIABLE ($vlProcess ; at_IPCom_Array{$vlElem} ; $vtElem)
   `$vtElemを使った何らかの処理を行う
End for
```

注：この例では、プロセス変数「vl_IPCom_Array」には配列「at_IPCom_Array」のサイズが含まれており、ソースプロセスによって管理されている必要があります。

5. 次の例は上記の例と同じことをしますが、配列の要素を順番に読み込む代わりに配列を全体として読み込みます：

```
GET PROCESS($vlProcess;at_IPCom_Array;$anArray)
For ($vlElem ; 1 ; Size of array($anArray))
   `$anArray{$vlElem}を使った何らかの処理を行う
End for
```

6. 次の例は、変数v1、v2、v3のソースプロセスインスタンスを読み込み、それらの値を現在のプロセスの同じ変数のインスタンスに返します：

```
GET PROCESS VARIABLE ($vlProcess ; v1 ; v1 ; v2 ; v2 ; v3 ; v3)
```

7. **DRAG AND DROP PROPERTIES**コマンドの例を参照してください。

参照：CALL PROCESS、DRAG AND DROP PROPERTIESSET、PROCESS VARIABLE

SET PROCESS VARIABLE

SET PROCESS VARIABLE (プロセス ; 送信先変数 ; ソース式 { ; 送信先変数2 ; ソース式2 ; ...; 送信先変数N ; ソース式N})

引数	タイプ		説明
プロセス	数値	→	**送信先プロセス番号**
送信先変数	**変数**	→	**送信先変数**
ソース式	変数	→	**ソース式（またはソース変数）**

SET PROCESS VARIABLEコマンドは、引数＜ソース式＞(＜ソース式2＞など)で渡す値を使用して＜プロセス＞に渡す番号を持つ送信先プロセスの＜送信先変数＞(＜送信先変数2＞など)プロセス変数を書き込みます。

それぞれのソース変数は、変数、配列、配列要素のいずれかを指定できます。ただし、このセクションで後述する制限事項を参照してください。

＜送信先変数＞と＜ソース式＞変数の組では、この式は送信先変数と互換性のあるタイプである必要があり、互換性がない場合には、変数に意味のない値が入ってしまうことになります。インタープリタモードでは、送信先変数が存在しない場合には、この式で送信先変数が作成され、割り当てられます。

現在のプロセスは送信先プロセスの変数を「のぞき見」しています。送信先プロセスは別のプロセスが自分の変数のインスタンスを読み込んでいることについては何も警告されません。

制限事項：

SET PROCESS VARIABLEコマンドは、送信先変数としてインタープロセス変数またはローカル変数は受け付けません。

SET PROCESS VARIABLEコマンドは、次のタイプ以外の任意の送信先プロセス変数を受け付けます。

■ ポインタ

■ 任意のタイプの配列：あるプロセスから別のプロセスに配列を全体として書き込むには、**VARIABLE TO VARIABLE**コマンドを使用します。ただし、**SET PROCESS VARIABLE**コマンドは配列の要素を書き込むことはできます。

■ ポインタ配列の要素または2次元配列の要素を書き込むことはできません。

送信先プロセスは、ユーザプロセスである必要があります。カーネルプロセスは、送信先プロセスにはなれません。送信先プロセスが存在しない場合には、エラーが生成されます。**ON ERR CALL**コマンドでインストールされたエラー処理メソッドを使用すると、このエラーを受け取ることができます。

例：

1. 次のコードは、番号が$vlProcessであるプロセスのテキスト変数「vtCurStatus」を(空の文字列に)設定します：

```
SET PROCESS VARIABLE ($vlProcess ; vtCurStatus ; "")
```

2. 次のコードは、番号が$vlProcessであるプロセスのテキスト変数「vtCurStatus」を現在のプロセスで実行中のメソッドの変数「$vtInfo」の値に設定します：

```
SET PROCESS VARIABLE ($vlProcess ; vtCurStatus ; $vtInfo)
```

3. 次のコードは、番号が$vlProcessであるプロセスのテキスト変数「vtCurStatus」を現在のプロセスの同じ変数の値に設定します：

```
SET PROCESS VARIABLE ($vlProcess ; vtCurStatus ; vtCurStatus)
```

注：最初のvtCurStatusは、送信先プロセスにある変数のインスタンスを示しています。2番目のvtCurStatusは現在のプロセスにある変数のインスタンスを示しています。

4. 次の例は、$vlProcessで示されるプロセスのプロセス配列の要素を大文字で順番に設定します：

```
GET PROCESS VARIABLE ($vlProcess ; vl_IPCom_Array ; $vlSize)
For($vlElem ; 1 ; $vlSize)
   GET PROCESS VARIABLE($vlProcess;at_IPCom_Array{$vlElem};$vtElem)
   SET PROCESS VARIABLE ($vlProcess ; at_IPCom_Array{$vlElem} ; Uppercase($vtElem))
End for
```

注：この例では、プロセス変数「vl_IPCom_Array」には配列「at_IPCom_Array」のサイズが含まれており、ソース／送信先プロセスによって管理されている必要があります。

5. 次の例は、現在のプロセスからの同じ変数のインスタンスを使用して、変数v1、v2、v3の送信先プロセスインスタンスを書き込みます：

```
SET PROCESS VARIABLE ($vlProcess ; v1 ; v1 ; v2 ; v2 ; v3 ; v3)
```

参照：CALL PROCESS、GET PROCESS VARIABLE

VARIABLE TO VARIABLE

VARIABLE TO VARIABLE (プロセス ; 送信先変数 ; ソース変数 {; 送信先変数2 ; ソース変数2 ; ...; 送信先変数N ; ソース変数N})

引数	タイプ		説明
プロセス	数値	→	**送信先プロセス番号**
送信先変数	**変数**	→	**送信先変数**
ソース変数	変数	→	**ソース変数**

VARIABLE TO VARIABLEコマンドは、引数＜ソース変数＞(＜ソース変数2＞など)で渡す値を使用して＜プロセス＞に渡す番号を持つ送信先プロセスの＜送信先変数＞(＜送信先変数2＞など)プロセス変数を書き込みます。

VARIABLE TO VARIABLEコマンドは、次の違いを除いて、**SET PROCESS VARIABLE**コマンドと同じ動作をします。

■ **SET PROCESS VARIABLE**コマンドは引数に＜ソース式＞を渡すので、全体として配列を渡すことができない。これに対して、**VARIABLE TO VARIABLE**コマンドは引数に＜ソース変数＞を渡すので、配列を全体として渡すことができる。

■ **SET PROCESS VARIABLE**コマンドの各送信先変数は、任意の変数または任意の配列要素になることはできるが、配列の全体になることはできない。これに対して、**VARIABLE TO VARIABLE**コマンドの各送信先変数は、任意の変数、任意の配列または任意の配列要素になることはできる。

＜送信先変数＞と＜ソース変数＞変数の組では、このソース変数は送信先変数と互換性のあるタイプである必要があり、互換性がない場合には、変数に意味のない値が入ってしまうことになります。インタープリタモードでは、送信先変数が存在しない場合には、ソース変数のタイプと値を使って、送信先変数が作成され、割り当てられます。

現在のプロセスは、送信先プロセスの変数を「のぞき見」しています。送信先プロセスは別のプロセスが自分の変数のインスタンスを読み込んでいることについては何も警告されません。

制限事項：

VARIABLE TO VARIABLEコマンドは、送信先変数としてインタープロセス変数またはローカル変数は受け付けません。

VARIABLE TO VARIABLEコマンドは、次のタイプ以外の任意の送信先プロセス変数またはインタフェースプロセス変数を受け付けます。

■ ポインタ
■ ポインタの配列
■ 2次元配列

送信先プロセスは、ユーザプロセスである必要があります。カーネルプロセスは、送信先プロセスにはなれません。送信先プロセスが存在しない場合には、エラーが生成されます。**ON ERR CALL**コマンドでインストールされたエラー処理メソッドを使用することにより、このエラーを受け取ることができます。

例：

次の例は、ローカル変数 “ $vlProcess ” で示されたプロセスから任意のプロセスを読み込み、配列要素を順番に大文字に変換して、配列を全体として書き込みます：

```
GET PROCESS VARIABLE($vlProcess ; at_IPCom_Array ; $anArray)
For($vlElem ; 1 ; Size of array($anArray))
   $anArray{$vlElem}:=Uppercase($anArray{$vlElem})
End for
VARIABLE TO VARIABLE($vlProcess ; at_IPCom_Array ; $anArray)
```

参照：GET PROCESS VARIABLE、SET PROCESS VARIABLE

プロセス（ユーザインタフェース）コマンド　40

この章では、「ルーチン」エディタの「Process (User Interface)」テーマ内にあるプロセス（ユーザインタフェース）コマンドについて説明します。この章のコマンドは、プロセスのユーザインタフェース要素に影響します。プロセスとそのインタフェース要素は、**HIDE PROCESS**コマンドや**SHOW PROCESS**コマンドによって、表示または非表示になります。メッセージプロセスのようなプロセスが必要な場合は、**BRING TO FRONT**コマンドを使用して、プロセスを前面に配置することができます。

BRING TO FRONT
Frontmost process
HIDE PROCESS
SHOW PROCESS

HIDE PROCESS

HIDE PROCESS (プロセス)

引数	タイプ	説明
プロセス	数値	非表示にするプロセスのプロセス参照番号

HIDE PROCESSコマンドは＜プロセス＞に属するすべてのウインドウを非表示にします。＜プロセス＞のすべてのインタフェース要素は、次に**SHOW PROCESS**コマンドを実行するまで非表示となります。従って、プロセスが非表示になっているときにウインドウを開いても画面が再描画されたり表示されません。

ただし、デバッグウインドウは例外です。＜プロセス＞を非表示にしてもデバッグウインドウが表示されると、＜プロセス＞は表示され最前プロセスとなります。

＜プロセス＞を全く表示したくなければ、**HIDE PROCESS**コマンドをプロセスプロシージャの最初のコマンドにします。このコマンドは、「ユーザ／カスタムメニュー」プロセスおよび「キャッシュマネージャ」プロセスを非表示にすることはできません。

プロセスを非表示にしてもそれを実行し続けることができます。

▼次の例は、カレントプロセスのすべてのウインドウを非表示にします：

```
HIDE PROCESS (Current process)
```

SHOW PROCESS

SHOW PROCESS (プロセス)

引数	タイプ	説明
プロセス	数値	表示するプロセスのプロセス参照番号

SHOW PROCESSコマンドは＜プロセス＞に属する全ウインドウを表示します。このコマンドは、＜プロセス＞のウインドウを最前面ウインドウにするわけではありません。これには**BRING TO FRONT**コマンドを使用します。

プロセスが既に表示されている場合は、このコマンドは何も行いません。

▼ 次の例は、以前に非表示になっていれば「デザイン」プロセスを表示します。「デザイン」プロセスのプロセス参照はインタープロセス変数 “ <>DesignProc ” に格納されます：

```
SHOW PROCESS (<>DesignProc)
```

BRING TO FRONT

BRING TO FRONT (プロセス)

引数	タイプ	説明
プロセス	数値	最前レベルに持って行くプロセスのプロセス参照番号

BRING TO FRONTコマンドは＜プロセス＞のウインドウを最前に配置します。＜プロセス＞に属するウインドウはすべて最前に配置されます。ウインドウの順序は変わりません。このプロセスが既に最前のプロセスの場合は、このコマンドは何も行いません。プロセスが非表示の場合に、**SHOW PROCESS**コマンドでプロセスを表示しないと**BRING TO FRONT**コマンドは効果がありません。

「ユーザ／カスタムメニュー」プロセスは、このコマンドで前面に配置することができます。

▼ 次の例は、メニューから実行できるプロジェクトメソッドです。これは、最前のプロセスが “ <>Customers ” プロセスかどうかを調べています。そうでなければ、それを前面に配置します：

```
If (Frontmost process # <>Customers)    ` 顧客リストが表示されていない場合
   BRING TO FRONT (<>Customers)         ` リストを前面に持ってくる
End if
```

Frontmost process

Frontmost process ({*}) → 整数

引数	タイプ	説明
*		最初の非フローティングウインドウのプロセス番号

Frontmost process関数は、ウインドウが最前面にあるプロセスの番号を返します。

1つまたは複数のフローティングウインドウがある場合は、次の2種類のウインドウレイヤがあります。

- 通常のウインドウ
- フローティングウインドウ

フローティングウインドウのフォームメソッドやオブジェクトメソッドから**Frontmost process**関数を使用すると、この関数はフローティングウインドウレイヤ内の最前面のフローティングウインドウのプロセス参照番号を返します。引数アスタリスク(*)を指定すると、この関数は、通常ウインドウレイヤ内の最前面のアクティブウインドウのプロセス参照番号を返します。

前述の**BRING TO FRONT**コマンドの例を参照してください。

この章では、「ルーチン」エディタの「Process」テーマ内にあるプロセスコマンドについて説明します。この章のコマンドは、プロセスを管理します。

New process
RESUME PROCESS
DELAY PROCESS
Process state
PROCESS PROPERTIES
Execute on server
PAUSE PROCESS
Current process
Count tasks
Count user processes
Count users
Process number

New process

New process (メソッド ; スタック ;{プロセス}) → プロセス参照番号

引数	タイプ	説明
メソッド	文字列	呼び出すメソッド
スタック	数値	バイト単位のスタックサイズ
プロセス	文字列	作成したプロセスの名前

New process関数は、＜プロセス＞という名前で新しいプロセスを開始し、そのプロセスの参照番号を返します。

＜メソッドは、そのプロセスで実行されるメソッドの名前を指定する文字列です。

＜スタック＞は、プロセスのスタックに割り当てられたメモリの量です。このメモリ領域にメソッド呼び出し、ローカル変数、サブルーチンの引数、スタックレコード、再帰ルーチンなどのプロセスオブジェクトが保存されます。この大きさはバイトで表され、プロセスが作成したオブジェクトの数によって決まります。ほとんどのプロセスでは、16000から32000の値が妥当です。最小スタックサイズは16000です。

オプション引数の＜プロセス＞は、プロセスの名前です。＜プロセス＞は「デザイン」モードの「プロセスリスト」エディタ内でプロセスを特定するために使用します。名前を指定しなければ、そのプロセスは「プロセスリスト」エディタ内で無名プロセスとなります。プロセス名は最大31バイトまで付けることができます。

プロセス名の先頭に“ $ ”記号を付けることにより、プロセスをローカルにすることができます。ただし、このローカルプロセスでは、データにアクセスすることができないので注意が必要です。ローカルプロセスに関する詳細は、第10章の「ローカルプロセスとグローバルプロセス」の節を参照してください。

4D Server：4D Serverでは、名前は個々のユーザ用のサーバのメインウインドウに表示されます。プロセス名の先頭にドル記号($)を付けることによってローカルプロセスを作成することができます。これによって4D Serverはサーバを多用する操作の処理を続けることができます。ローカルプロセスを使ってデータをアクセスするべきではありません。

プロセスを開始すると、そのプロセスは独自のメニューバーを持ちます。デフォルトでは、これは空白のメニューバーになります。**MENU BAR**コマンドを使用してプロセスのメニューバーを設定します。プロセスは一度に複数のウインドウを開くことができますが、プロセスは1つのアクティブウインドウしか持つことができません。

各プロセスは独自のカレントセレクションを持っています。各プロセスは各ファイルに対して独自のカレントレコードを持つことができます。プロセスを開始したときの新プロセス内の各ファイルのカレントセレクションは空で、カレントレコードもありません。

すべてのプロセスには独自のプロセス変数があります。

DEFAULT TABLEコマンドを実行するまで、プロセスにはデフォルトテーブルはありません。**DEFAULT TABLE**コマンドを実行した後は独自のデフォルトテーブルを持つことができます。各プロセスは、各テーブルに対して別々の入力フォームと出力フォームを持つことができます。「デザイン」モードの「フォーム」ダイアログボックス内でテーブルのデフォルト入力フォームと出力フォームを指定することができます。

各プロセスは独自のプロセスセットを持つことができます。各プロセスは独自のUserSetとLockedSetを持ちます。各プロセスは独自のデバッガを持つことができます。プロセスのデバッガを表示するには、**TRACE**コマンドを使用するか、あるいは「プロセスリスト」エディタ内でプロセスを選択して「プロセス」メニューから「トレース」を選択します。

作成できるプロセスの数はメモリによって決まります。4th Dimensionの場合は、これはマシンのメモリによって決まります。4D Serverの場合では、これはサーバメモリとクライアントメモリによって決まります。プロセスを作成できない場合、**New process**関数は 0 を返します。

▼ 次の例では、“ $プロセス名 ”というローカルプロセスを作成します。スタックサイズは32000バイトで、プロセスメソッド“ MyProcedure ”がプロセスを制御します：

```
pId:=New process ("MyProcedure" ; 32000 ; "$プロセス名")
```

警告：ローカルプロセスをサーバアクセス用に決して使用してはいけません。ローカルプロセス内で実行されるコードはストラクチャファイルまたはデータファイルをアクセスしてはいけないことを意味します。

▼次の例では、2つの異なるプロセスを作成します。最初のプロセス“ 顧客リスト ”は、メソッド“ 顧客表示 ”を別プロセスとして開始します。“ 顧客表示 ”メソッドは[顧客]テーブル内の全レコードを表示します。“ ベンダーリスト ”プロセスは同じ方法でベンダーの一覧を表示します：

```
C_LONGINT (<>顧客修正 ; <>ベンダー修正)
<>顧客修正:= New process ("顧客表示" ; 32000 ; "顧客リスト")
<>ベンダー修正:= New process ("ベンダー表示" ; 32000 ; "ベンダーリスト")
```

メソッド“ 顧客表示 ”は、[顧客]ファイル内のレコードを表示します：

```
ALL RECORDS ([顧客])                    ` 全レコードを選択
MODIFY SELECTION ([顧客] ; *)           ` カレントセレクションを表示
```

▼次の例では、4つのプロセスを続けて開始します。2番目のプロセスは最初のプロセスがインタープロセス変数“ <>開始 ”を“ True(真) ”にするまで開始しません。最初のプロセスが“ <>開始 ”を“ True(真) ”にするので、最初のプロセスを作成中に2番目のプロセスが開始されることはありません。同様に、3番目のプロセスが2番目のプロセスを作成中に作られることもありません。このプロセスの作成方法を実現するために、まずプロセスが作られるごとにプロセス参照番号を格納する配列を作成します：

```
C_BOOLEAN (<>開始)
ARRAY LONGINT (<>ProcArray ; 4)         ` 4要素を持つ配列を初期化する
   ` ループして各プロセス参照を配列の各要素に入れる
For ($ i; 1 ; 4)
  <>ProcArray{$i}:= New process ("メソッド：" + String ($i) ; 32000)
  Repeat
    IDLE                                ` コンパイル済みのデータベース用
    ` これによって各プロセスはその前のプロセスを待って開始される
    ` 開始されるメソッドが変数をTrueに設定
  Until (<>開始)
  <>開始:= False                        ` 次のプロセスのためにリセットする
End for
```

DELAY PROCESS

DELAY PROCESS (プロセス ; 遅延時間)

引数	タイプ	説明
プロセス	数値	プロセス参照番号
遅延時間	数値	遅延時間

DELAY PROCESSコマンドはプロセスの実行を指定したtick(処理単位＝1/60秒)の数だけ遅らせます。この間、そのプロセスは4th Dimensionおよびユーザのマシンの処理時間を使用しません。プロセスの実行を遅延しても、そのプロセスはメモリ内に残ります。

プロセスが既に遅延状態の場合、このコマンドはそれを再度遅延します。この場合は、＜遅延時間＞は残時間に加算されるのではなく、それに置き換わります。従って、これ以上遅延させたくなければゼロ(0)を渡します：

```
DELAY PROCESS (pID ; 0)
```

プロセスが存在しなければ、このコマンドは何も行いません。

警告：DELAY PROCESSコマンドは、開始したプロセス内でしか使用しないようにしてください。DELAY PROCESSコマンドを「ユーザ／カスタムメニュー」プロセス内で使用してはいけません。

▼ 次の例では、プロセスの開始を真夜中まで遅延してその後レポートを印刷します。これによって、昼間の貴重な時間を節約することができます：

```
<>レポート:=New process ("ReportProc" ; 32000 ; "レポート")
```

“ ReportProc ”メソッドは、真夜中までのtickの数を計算してその分プロセスを遅延します。

```
` 真夜中までのtick数
$真夜中:=?24:00:00? - Current time * 60
DELAY PROCESS (Current process ; $真夜中)
レポート印刷　` レポートを印刷するメソッド
```

▼ 次の例では、[顧客]テーブルのカレントレコードがロックされているかを調べています。ロックされていればプロセスが1秒遅延されます。この技法はマルチユーザ環境およびマルチプロセス環境で使用することができます：

```
` 特定のレコードを検索
SEARCH ([顧客] ; [顧客]名前="山田")
If (Records in selection ([顧客]) > 0)   ` 少なくとも1レコードが見つかった場合
   While (Locked ([顧客]))               ` カレントレコードがロックされている場合
      MESSAGE( "そのレコードはロックされています")` メッセージを表示
      DELAY PROCESS (Current process ; 60)      ` 1秒待ってレコードを
      LOAD RECORD ([顧客])                      ` 再度ロードする
   End while
   ` カレントレコードがロック解除の状態
   MODIFY RECORD ([顧客])               ` ユーザにレコードを変更させる
   UNLOAD RECORD ([顧客])               ` レコードロックを解除する
Else
   ALERT ("検索条件と一致するレコードはありません")
End if
```

PAUSE PROCESS

PAUSE PROCESS (プロセス)

引数	タイプ	説明
プロセス	数値	プロセス参照番号

PAUSE PROCESSコマンドは、**RESUME PROCESS**コマンドで再開されるまで＜プロセス＞の実行を停止します。この間＜プロセス＞は4D Serverあるいはユーザマシンの処理時間を使用しません。プロセスは停止されてもメモリ内に残ります。

＜プロセス＞が既に停止していた場合は、**PAUSE PROCESS**コマンドは何も行いません。プロセスが**DELAY PROCESS**コマンドで遅延されている場合もそのプロセスは停止されます。**DELAY PROCESS**コマンドで指定した時間が過ぎる前にそのプロセスが再開されると、時間がくるまで遅延されてからアクティブになります。**RESUME PROCESS**コマンドは、即座にプロセスを再開します。

プロセスの実行を停止している間はそのプロセスのウインドウに入力することはできません。この場合には、ユーザの混乱を避けるためにそのプロセスを非表示にするとよいでしょう。＜プロセス＞が存在しなければ、このコマンドは何も行いません。

警告：PAUSE PROCESSコマンドは、開始したプロセス内でしか使用しないようにしてください。PAUSE PROCESSコマンドを「ユーザ／カスタムメニュー」プロセス内で使用してはいけません。

▼次の例は、文字フィールドとテキストフィールド内の入力項目を複製するためのチェックです：

「On startup」データベースメソッドは、“ P_Duplicates ” メソッドの起動中に「Duplicate Finder」プロセスを開始します：

```
C_POINTER (<>vフィールド)
C_STRING (80 ; <>v値)
C_LONGINT (<>v複製 ; <>vステータス)
<>v複製 := New process ("P_Duplicates" ; 32000 ; "Duplicate Finder")
```

下記に “ P_Duplicates ” メソッドを示します：

```
   `複製項目を検索し、プロセスを停止する
   `文字とテキストでのみ機能する
C_POINTER ($theTable)
HIDE PROCESS (Current process) `プロセスを隠す
While (True) `ループを初期化する
` “ CheckDups ” メソッドがコールされるまでプロセスを停止する
   PAUSE PROCESS (Current process)
   $theFile := Table (Table (<>theField)) `フィールドポインタのテーブルポインタ
   QUERY ($theTable-> ; <>theField-> = <>v値) `複製項目の検索
   `` “ CheckDup ” にステータスを返す
   <>vステータス := Records in selection ($theTable->))
End while
```

次のフォーム内にあるオブジェクトメソッドは、“ CheckDups ” メソッドをアクティブにします：

```
CheckDups (Self)
```

下記は、“ CheckDups ” メソッドのコードです：

```
C_POINTER ($1)
<>vステータス := -1              `vステータスのデフォルトのステータス
<>theFIeld := $1
<>v値 := $1->
RESUME PROCESS (<>v複製)
Repeat
   IDLE
Until (<>vステータス#0)          ` “ Duplicate Finder ” プロセスが終了するまで待機する
If (<>vステータス#0)
   ALERT ($1-> + "複製項目です！")          `ユーザに警告する
   $1-> := ""                    `前回の入力項目をクリアする
   GOTO AREA ($1->)              `そのフィールドにカーソルを戻す
End if
```

RESUME PROCESS

RESUME PROCESS (プロセス)

引数	タイプ	説明
プロセス	数値	プロセス参照番号

RESUME PROCESSコマンドは、実行が停止しているプロセスを再開します。プロセスが停止していない場合、このコマンドは何も行いません。

プロセスが事前に遅延されてる場合は、**PAUSE PROCESS**コマンドまたは**DELAY PROCESS**コマンドを参照してください。プロセスが存在しない場合、このコマンドは何も行いません。

PAUSE PROCESSコマンドの例を参照してください。

Current process

Current process → プロセス参照番号

Current process関数は、このコマンドを呼び出したプロセスのプロセス参照番号を返します。

DELAY PROCESSコマンドと**PROCESS ATTRIBUTES**コマンドの例を参照してください。

Process state

Process state (プロセス) → 整数

引数	タイプ	説明
プロセス	数値	プロセス参照番号

Process state関数は、プロセスのステータスコードを返します。プロセスのステータスとしては次のようなものがあります。

ステータス	値	説明
実行中	**0**	プロセスメソッドが実行中
延期中	**1**	プロセスが遅延されている
ユーザイベント待ち	**2**	一般的にデータ入力待ち
入出力(I/O)待ち	**3**	レコードのキャッシュ中
セマフォ待ち	**4**	データベースアクセス待ち
停止	**5**	プロセスが停止されている
ダイアログ非表示	**6**	モーダルダイアログを持つプロセスが隠れている
中止	**-1**	プロセスが中止された
	-100	プロセスが存在しない

4D Server：このコマンドは、サーバ上ではなくワークステーション上でプロセスステータスを返します。

▼ 次の例は、各プロセスの名前とプロセス参照番号を配列 “ arProcName ” と “ arProcNum ” に入れます。このメソッドは、プロセスが中止されたかを調べます。この場合、プロセス名と参照番号は配列に追加されません：

```
$タスク:= Count tasks
ARRAY STRING (15 ; arProcName ; $タスク)
ARRAY INTEGER (arProcNum ; $タスク)
C_INTEGER ($i ; vCount ; vState ; vTime)
For ($i ; 1 ; $タスク)
   If (Process state # -1)                    ` プロセスが中止されていなければ
      vCount:=vCount + 1                      ` 要素数をカウントする
      PROCESS ATTRIBUTES (arProcNum{$i} ; arProcName{$i} ; vState ; vTime)
   End if
End for
  ` プロセスの数に応じて配列の大きさを変える
ARRAY STRING (15 ; arProcName ; vCount)
ARRAY INTEGER (arProcNum ; vCount)
```

Count users

Count users → 整数

Count users関数は、サーバに接続されているユーザの数を返します。シングルユーザの4th Dimensionの場合は、**Count users**関数は1を返します。

Count tasks

Count tasks → 整数

Count tasks関数は、ワークステーション上またはシングルユーザの4th Dimension上で開いているプロセスの数を返します。この数には、4th Dimensionが自動的に管理するものも含めてすべてのプロセスが含まれます。この中には、「ユーザ／カスタム」プロセス、「デザイン」プロセス、「キャッシュマネージャ」プロセス、および「インデックス」プロセスが含まれます。

Count user processes

Count user processes → 整数

Count user processes関数は、4th Dimensionが自動的に管理するプロセス以外の開いているプロセスの数を返します。「ユーザ／カスタム」プロセスは「ユーザ」プロセスとみなします。従って、「ユーザ／カスタム」プロセスも「ユーザ」プロセスの数に入ります。

Execute on server

Execute on server (メソッド ; スタック {; 名前 {; パラメータ {; パラメータ2 ;...; パラメータN}{; *}})

引数	タイプ		説明
メソッド	文字列	→	プロセスの中で実行されるメソッド
スタック	数値	→	スタックサイズ（バイト単位）
名前	文字列	→	作成されるプロセスの名前
パラメータ	式	→	作成されるプロセスの名前
*		→	メソッドのパラメータ（引数） ユニークプロセス
関数の返す値	数値		新しく作成されるプロセスのプロセス番号 またはすでに実行しているプロセス

この関数は、バージョン6の4D Serverがリリースされる時にドキュメント化されます。

参照：New process

PROCESS PROPERTIES

PROCESS PROPERTIES (プロセス ; 名前 ; ステータス ; 時間 {;表示})

引数	タイプ		説明
プロセス	ポインタ	→	プロセス番号
名前	文字列	←	プロセス名
ステータス	数値	←	プロセスステータス
時間	数値	←	プロセスの累積時間
表示	ブール	←	表示の場合はTrue、非表示の場合はFalse

PROCESS PROPERTIESコマンドは、引数＜プロセス＞でプロセス番号を渡すプロセスに関する情報を返します。

このコマンドを呼び出すと、以下が起こります。

■ 引数＜名前＞はプロセスの名前を返します。プロセス名で注意すべき点は、次のとおりです。

■ プロセスが「ユーザ」モードの「メソッド実行」ダイアログボックスから(「新規プロセス」オプションを選択した状態で)起動された場合に、プロセスの名前は「P_」で始まり、その後に数字が続く形になります。

■ 「プロセス開始」プロパティが選択された任意のカスタムメニューコマンドから起動されたプロセスの場合には、プロセスの名前は「M_」または「ML_」で始まり、その後に数字が続く形になります。

■ プロセスがアボートしていた場合(そして、その「スロット」がまだ再使用されていない場合)、プロセスの名前はそのまま返されます。プロセスがアボートしたかどうかを検出するには、＜ステータス＞=-1をテストします(下記の表を参照)。

■ ＜ステータス＞は、呼び出しを行ったときのプロセスの状態を返します。この引数は、次の定義済定数で提供される値のいずれか1つを返すことができます。

定数	タイプ	値
Aborted	倍長整数	-1
Delayed	倍長整数	1
Does not exist	倍長整数	-100
Executing	倍長整数	0
Hidden modal dialog	倍長整数	6
Paused	倍長整数	5
Waiting for input output	倍長整数	3
Waiting for internal flag	倍長整数	4
Waiting for user event	倍長整数	2

■ 引数<時間>は、プロセスが起動されたときから使用している累積時間を(1/60秒単位で)返します。

■ オプション引数<表示>が指定された場合には、プロセスが表示されている場合はTRUE、隠されている場合はFALSEを返します。

プロセスが存在しない場合、つまり、**Count tasks**関数に範囲1の番号を渡さなかったという意味ですが、この場合、**PROCESS PROPERTIES**コマンドは変数パラメータの変更は行いません。

例：

1. 次の例は、変数vName、vState、VtimeSpentに現在のプロセスのプロセス名、プロセスステータス、プロセス時間を返します：

```
C_STRING(80; vName) ` Initialize the variables
C_INTEGER(vState)
C_INTEGER(vTime)
PROCESS PROPERTIES (Current process ; vName ; vState ; vTimeSpent)
```

2. 「**On Exit**」データベースメソッドの例を参照してください。

参照：Count tasks、Process state

Process number

Process number (プロセス名)→ 数値

引数	タイプ		説明
プロセス名	数値	→	プロセス番号を取り出すためのプロセスの名前
関数の返す値	数値	←	プロセス番号

Process number関数は、引数<プロセス名>で名前を渡すプロセスの番号を返します。プロセスが見つからない場合には、**Process number**関数は 0を返します。

例：

独立したプロセスで実行するカスタムフローティングウィンドウを作成します。このプロセスでは、「デザイン」モードでやり取りができる独自のツールを実装します。例えば、キーワードの階層リストで項目を選択するときに、「デザイン」モードの最前面ウィンドウにテキストを貼り付けたいとします。これを行うには、クリップボードを使用できますが、イベントの貼り付けは「デザインプロセス」の内部で発生する必要があります。次の関数は、「デザインプロセス」(が実行している場合)のプロセス番号を返します：

```
` 「Design process number」プロジェクトメソッド
` Design process number → 倍長整数
` Design process number → 「デザインプロセス」のプロセス番号
$0:=Process number(Get indexed string (170 ; 3))
` 「デザインプロセス」の名前は、4D内部のSTR#リソースIDが170で、ストリング番号
`  が3の中に格納される
`  注：これは、将来リソースが変わると変更される可能性がある
```

この関数を使用して、下記のプロジェクトメソッドは、引数として受け取ったテキストを「デザイン」モードの最前面のウィンドウに貼り付けます(適用可能な場合)：

```
` 「PASTE TEXT TO DESIGN」プロジェクトメソッド
` PASTE TEXT TO DESIGN (テキスト)
` PASTE TEXT TO DESIGN (最前面ウインドウに貼り付けられるテキスト )
C_TEXT($1)
C_LONGINT($vlDesignPID ; $vlCount)
$vlDesignPID:= Design process number
If ($vlDesignPID  #  0)
   `クリップボードにテキストを配置する
   SET TEXT TO CLIPBOARD($1)
   `「Ctrl-V」または「Command-V」イベントをポスとする
   POST KEY(Ascii("v") ; Command key mask ; $vlDesignPID)
   ` DELAY PROCESSコマンドを繰り返しコールすることでスケジューラは
   `「デザインプロセス」にイベントを渡す機会を取得する
   For ($vlCount ; 1 ; 5)
      DELAY PROCESS(Current process ; 1)
   End for
End if
```

参照：PROCESS PROPERTIES、Process state

検索とソートコマンド　42

この章では、「ルーチン」エディタの「Queries」テーマ内にあるクエリ（検索）と並べ替え（ソート）コマンドについて説明します。4th Dimensionのプログラミング言語には、レコードを検索するための多くのコマンドが用意されています。これらのコマンドは基本的には同じです。つまり、テーブル上のレコードを検索し、検索条件に適合するレコードを見つけ出すという機能です。各コマンドは、実行を完了した時点で見つけ出したレコードの集合（以降では、この集合のことを“セレクション”と呼びます)を生成します。また、レコードをソートするためにいくつかのコマンドがあります。ソートコマンドは、指定されたソートキーに従ってレコードを並べ変えます。

QUERY
QUERY BY FORMULA
QUERY BY EXAMPLE
QUERY SELECTION
QUERY SELECTION BY FORMULA
ORDER BY FORMULA
ORDER BY
SET QUERY LIMIT
SET QUERY DESTINATION

検索

検索する条件には、単一条件の場合も、複合条件の場合もあります。また“従業員番号57を検索する”などのように単一のレコードを探し出す場合や、“ニューヨークにあるすべての企業を検索する”などのように複数のレコードを探し出す場合もあります。

自動リレートを持ったテーブルのレコードを検索する場合には、リレート先テーブルのフィールドを使用することもできます。テーブルの自動リレートの定義に関する詳細は、『4th Dimension / 4D First デザインリファレンス』を参照してください。

インデックスを使用して検索すると極めて高速に処理することができます。特に大量のレコードを検索する場合に有用です。

できる限り**QUERY**コマンドや**QUERY SELECTION**コマンドを使用することをお勧めします。これらのコマンドは、「ユーザ」モードにおける「クエリ」エディタと同じ検索方法を使用します。つまり最初にインデックスを使用してから、必要な検索を順次行う、という検索処理の最適化を自動的に行います。(以降では、このように自動的に最適化される検索方法を“インテリジェントな検索”と呼びます。)

QUERY BY FORMULAコマンドと**QUERY SELECTION BY FORMULA**コマンドは、非常に強力で柔軟性のある検索コマンドです。これらは、**QUERY**コマンド、**QUERY SELECTION**コマンドのような検索条件の制約がありません。フィールド内のサブストリングに対する検索や、計算結果に基づいた検索など、より高度な検索が行うことができます。ただし、これらのコマンドの実行する検索処理は、常にシーケンシャルな検索のためインデックスを使用した検索よりも遅くなります。

QUERY SUBRECORDSコマンドは、1つのレコードのサブレコード内で検索します。このコマンドはサブレコードに対してのみ有効で、レコードに対しては何も行いません。サブレコードに対して検索可能なのはこのコマンドだけです。他の検索コマンドは、全てのレコードを検索します。

検索処理の実行中に検索の進捗状況を表すインジケータ(ナンバーやサーモメータ)を表示します。これらのインジケータを表示したくない場合は、**MESSAGES OFF**コマンドを使用してください。インジケータの「中止」ボタンや検索ウインドウの「キャンセル」ボタンをクリックすると検索を中止します。中止されたかどうかは、システム変数OKの値を調べるとわかります。検索が完了していれば1が、中止されていれば 0が代入されています。検索コマンドで検索した結果を表示する場合には、**DISPLAY SELECTION**コマンドまたは**MODIFY SELECTION**コマンドを使用します。

ソート

ソートは、データベースの処理の中でも重要な操作の1つです。セレクションのソートに関するコマンドは、レコードの選択が表示されたり、印刷される前によく使用されます。ソートコマンドは、“ あ ”から“ ん ”への昇順ソートや、逆に“ ん ”から“ あ ”への降順ソートを行います。また、複数のレベルでソートすることもできます。**ORDER BY FORMULA**コマンドは、計算値を基にしたソートを実行することができます。また、ブレークレベルを作成するためにレコードをソートする必要があるかもしれません。

コマンドによるソートは、一時的にレコードをソートするだけです。恒久的にソートしたい場合は、4D Toolsを使用します。

警告：テキストフィールドをソートすることはできません。

テキストフィールドをソートしたい場合、次のようなコードを使用することにより、そのフィールドの先頭の80バイトをソートすることができます：

```
ORDER BY FORMULA ([テーブル] ; Substring ([テーブル]ﾃｷｽﾄﾌｨｰﾙﾄﾞ ; 1 ; 80))
```

QUERY BY EXAMPLE

QUERY BY EXAMPLE ({テーブル})

引数	タイプ	説明
テーブル	テーブル	検索対象となるテーブル

QUERY BY EXAMPLEコマンドは、「ユーザ」モードの「フォームで検索」メニューコマンドと同じ処理を実行します。検索ウインドウとしてカレント入力フォームを表示します。**QUERY BY EXAMPLE**コマンドは、フォームに入力された情報を使用して＜テーブル＞を検索します。このときに使用するフォームは、検索対象となるテーブルに属するフォームでなければなりません。フォームの指定を省略した場合には、カレントの入力フォームを使用します。検索はインテリジェントな検索を行います。つまり、インデックスが付けられたフィールドは自動的にそれを使用するといった最適化された検索を行います。

「登録」ボタンをクリックするか、または“ Enter ”キーを押すとシステム変数OKに1を代入し、検索を開始します。また「キャンセル」ボタンをクリックするか、またはキャンセルのキー割り当てを行うとシステム変数OKに 0を代入し、検索を中止します。

「ユーザ」モードの「フォームで検索」アイテムに関する詳細は、『4th Dimension / 4D First ユーザリファレンス』を参照してください。

▼ 次のメソッドの例は、まず“ 検索 ”という名前のフォームを表示します。次にユーザがフォームに検索データを入力し、検索の実行を要求した場合(システム変数OKに1が代入された場合)に、検索結果を表示するようにします：

```
INPUT FORM ([従業員] ; "検索")        ` フォームを検索するために切り換える
QUERY BY EXAMPLE ([従業員])           ` フォームを表示し、検索を実行する
If (OK=1)                             ` ユーザが検索実行を指示した場合
  DISPLAY SELECTION ([従業員])        ` レコードを表示する
End if
```

QUERY

QUERY ({テーブル})

引数	タイプ	説明
テーブル	テーブル	検索対象となるテーブル

QUERY ({テーブル};検索条件;{*})

引数	タイプ	説明
テーブル	テーブル	検索対象となるテーブル
検索条件		検索条件
*		継続検索フラグ

QUERYコマンドには2つの形式があります。どちらの形式も＜テーブル＞を検索して検索結果をセレクションとして返します。インデックスフィールドを指定すると検索は最適化されます。つまり、インデックス検索を最初に実行することで、最小限の時間で検索します。このコマンドは、＜テーブル＞のカレントセレクションを変更します。そして、セレクションの先頭のレコードをカレントレコードにします。

第1の形式は、＜テーブル＞用の「クエリ」エディタを表示します。ここでユーザはエディタ内で検索条件を設定し、検索処理を実行することができます。設定した検索条件をディスクに保存することもできます。

QUERYコマンドのオプション引数であるアスタリスク(*)を指定した検索を実行すると、「クエリ」エディタは表示されません。その代わりに、オプション引数アスタリスク(*)を指定した検索を実行します。

QUERYコマンドの第2の形式は検索対象テーブルに対して、＜検索条件＞を使用したインテリジェントな検索を実行しセレクションを生成します。

複合条件を使用して検索することもできます。複数の**QUERY**コマンドを実行することによってこのような検索を行うことができます。各**QUERY**コマンドにオプション引数アスタリスク(*)を指定して、複合条件の検索であることを宣言します。コマンド記述の後で、各検索条件は論理演算子によって互いに結合されます。

QUERYコマンドによる検索に時間がかかる場合は、4th Dimensionは自動的にサーモメータで進捗状況のメッセージを表示します。**MESSAGES ON**コマンドと**MESSAGES OFF**コマンドを使用して、メッセージを表示または非表示にすることができます。

検索条件の指定方法

“ 検索条件 ” には、次のような書式を使用します：

{論理演算子} {;} フィールド {;} 比較演算子 {;} 比較値

“ 論理演算子 ” はオプションです。これは、前に実行した**QUERY**コマンドの検索条件と次に実行する**QUERY**コマンドの検索条件を結合するために使用します。

使用できる “ 論理演算子 ” は、「ユーザ」モードの「クエリ」エディタで使用できるものと同じです。“ 論理演算子 ” を次に示します。

検索用論理演算子	**記号**
かつ	&
または	|
以外	#

“ 論理演算子 ” は定数です。文字列ではありません。

“ 論理演算子 ” は、複合条件を使用して行う検索処理の最初の**QUERY**コマンド上および**QUERY**コマンドが1つしかない場合には使用することはできません。

“ フィールド ” は、検索対象となるフィールドのことです。自動リレートを持つテーブルの場合には、リレート先のテーブルのフィールドも使用可能です。

“ 比較演算子 ” は、フィールドと比較値とを比較するために使用します。

“ 比較演算子 ” を次に示します。

検索用比較演算子	記号
等号	=
不等号	#
より小さい	<
より大きい	>
より小さい、または等しい	<=
より大きい、または等しい	>=

“ 比較演算子 ” は定数または文字列のいずれでも使用できます。定数の場合には、セミコロン(;)はオプションとなり “ 比較演算子 ” を囲む必要はありません。文字列の場合には、必ずセミコロン(;)で “ 比較演算子 ” を囲む必要があります。

“ 比較値 ” はフィールドと比較するためのデータです。“ 比較値 ” はフィールドと同じデータタイプとして記述することも、文字列として記述することもできます。文字列として記述した場合には、自動的にフィールドのデータタイプと同じ形式に変換してから比較します。

データタイプの変換は検索を開始する前に一度だけ実行します。各レコードに対して実行されるわけではありません。

文字列中に特定の文字列を含んでいるかどうかを検索する場合(包含検索)には、“ 比較値 ” にワイルドカード記号(@)を使用することができます。

複合条件検索の使用方法

最終引数にアスタリスク(*)を指定した一連の**QUERY**コマンドを使用して、検索処理の実行を一時保留することができます。この方法により、複数の検索条件を使用した検索処理が可能です。検索処理を実行するためには、引数にアスタリスク(*)を指定しない**QUERY**コマンドを使用するだけです。また、引数の全くない**QUERY**コマンドを使用しても同様に実行することができます。

複合条件検索を行うための規則を次に示します：

■ 最初の**QUERY**コマンドは論理演算子を含んでいてはいけません。

■ 最初以外の**QUERY**コマンドは論理演算子を含んでいなければいけません。

■ 最後の**QUERY**コマンド以外は引数にアスタリスク(*)を指定しなければいけません。

■ 検索を実行するためには、最後の**QUERY**コマンドで引数アスタリスク(*)を指定してはいけません。また、引数の全くない**QUERY**コマンドを使用しても同様に実行することができます(**QUERY**コマンドの第1形式)。

■ 複合条件検索の処理シーケンスは各テーブルに固有のものです。従って、処理シーケンス内で使用する**QUERY**コマンドに対しては、引数＜テーブル＞を指定する**DEFAULT TABLE**コマンドを使用して、検索対象テーブルを明確に特定する必要があります。

QUERYコマンドによる検索例

▼次の例は、“ smith ” という名字の人のレコードをすべて検索します。4th Dimensionは大文字、小文字の種別を考慮しない点に注意してください。名字のフィールドにはインデックスが設定されています。**QUERY**コマンドは高速な検索のために、自動的にインデックスを使用します：

```
QUERY ([従業員] ; [従業員]名字 = "smith")
```

▼次の例は、“ John Smith ” という名前の人のレコードを全て検索します。名字のフィールドにはインデックスが設定されていますが、名前のフィールドにはインデックスが設定されていません。検索を実行すると、まず、最初にインデックスを持った名字のフィールドを検索し、“ Smith ” という名前の人のレコードのみに検索対象を絞ります。次に、名前のフィールドに対してシーケンシャルな検索を行い、“ John ” という名前のレコードを探します。

1番目の**QUERY**コマンドは、最後の引数にアスタリスク(*)を指定します。“ * ” を指定すると、検索の実行を一時保留することができます。2番目の**QUERY**コマンドは、論理演算子 “ かつ(&) ” によって1番目の**QUERY**コマンドと結合されます。これで、名字が “ Smith ” で名前が “ John ” の人を探すという検索条件が設定されます。2番目の**QUERY**コマンドには引数にアスタリスク(*)の指定がないので、実際の検索処理はこのコマンドで実行します：

```
QUERY ([従業員] ; [従業員]名字 = "smith";*)        ` 名字がSmithで...
QUERY ([従業員] ; & [従業員]名前 = "john")         ` 名前がJohnという人を探す
```

▼ 次の例は、“ Smith ” または “ Jones ” という名字の人のレコードをすべて検索します。名字のフィールドにはインデックスが設定されています。**QUERY**コマンドは両方の検索にインデックスを使用します。2番目の**QUERY**コマンドは、論理演算子 “ または(|) ” によって1番目の**QUERY**コマンドと結合されます。これで、名字が “ Smith ” または “ Jones ” の人を探すという検索条件が設定されます：

```
QUERY ([従業員] ; [従業員]名字;" = ";"smith";*)    ` 名字がSmithという人か...
QUERY ([従業員] ; | [従業員]名字;" = ";"jones")    ` 名字がJonesという人を探す
```

▼ 次の例は、会社員でない人のレコードを全て検索します。これは空のフィールド(空の文字列)を検索することによって行います：

```
QUERY ([従業員] ; [従業員]会社 = "")          ` 会社員でない人を探す
```

▼ 次の例は、“ Smith ” という名字で “ NY(ニューヨーク) ” に会社のある人のレコードを全て検索します。2番目の**QUERY**コマンドは、他のテーブルのフィールドを使用しています。これは、[従業員]テーブルから[会社]テーブルへリレートされているために可能になっています：

```
QUERY ([従業員] ; [従業員]名字 = "smith";*)  `名字がSmithで...
QUERY ([従業員] ; &[会社]所在地 = "NY")      `会社所在地がニューヨークの人を探す
```

▼ 次の例は、名前のイニシャルがAからMの人のレコードを全て検索します：

```
QUERY([従業員] ; [従業員]名前 < "n")          ` 名前のイニシャルがAからMの人を探す
```

▼ 次の例は、住所が “ 94(サンフランシスコ) ” または “ 90(ロサンゼルス) ” の人のレコードをすべて検索します(郵便番号が94または90で始まる)：

```
QUERY ([従業員] ; [従業員]郵便番号 = "94@";*)  ` 住所がサンフランシスコか...
QUERY ([従業員] ; | [従業員]郵便番号 = "90@")  ` ロサンゼルスの人を探す
```

▼ 次の例は、インデックスの設定されたサブフィールドを全て検索します。この検索は親レコード([従業員]テーブルのレコード)の検索です。サブレコードの検索ではありません。検索結果は “ Sabra ” という名前の子供がいる人のレコードすべてとなります：

```
` 名前が “ Sabra ” という子供を探す
QUERY ([従業員] ; [従業員]子供'名前 = "Sabra")
```

▼ 次の例は、「リクエスト」ダイアログボックスに入力された送り状番号と一致するレコードを検索します：

```
v番号:=Request ("送り状番号を入力してください。")` 送り状番号を入力する
If (OK=1)` 検索実行？
` 入力された番号と同じ送り状番号を探す
 QUERY ([送り状] ; [送り状]番号 = v番号)
End if
```

▼ 次の例は、1996年に作成された送り状のレコードを全て検索します。これは、1995年12月31日以降で1997年1月1日以前のレコードを全て検索します。2番目の**QUERY**コマンドの日付が文字列になっている点に注目してください。**QUERY**コマンドは自動的に文字列を正しいデータタイプに変換します(この場合は、日付)：

```
QUERY ([送り状] ; [送り状]作成日付 > !95.12.31!;*)` 作成日付が95年12月31日から...
QUERY ([送り状] ;&[送り状]作成日付 < "97.01.01")` 97年7月1日までの送り状を探す
```

▼ 次の例は、\2,500,000≦給与＜\5,000,000の条件に当てはまる従業員のレコードをすべて検索します：

```
` \2,500,000≦給与＜\5,000,000の従業員を探す
QUERY ([従業員] ; [従業員]給与 >= 2500000;*)
QUERY [[従業員] ; &[従業員]給与 < 5000000)
```

▼ 次の例は、3,000,000円以上の給与を得ている営業部に所属する従業員のレコードをすべて検索します。“ 給与 ” フィールドにはインデックスが設定されていますので最初に検索します。2番目の**QUERY**コマンドが、他のテーブルのフィールドを使用している点に注意してください。[従業員]テーブルから[部門]テーブルの “ 名称 ” フィールドへ自動リレートされているためにこのようなことが可能になっています。[従業員]テーブルのレコードの順番に従ってリレート処理を行うために、[部門]テーブルの “ 名称 ” フィールドには、インデックスが設定されているにも関わらず、インデックスを使用した検索は行われません：

```
  ` 給与が3,000,000円よりも多い、営業部に所属する社員を探す
QUERY ([従業員] ; [従業員]給与 > 3000000;*)
QUERY ([従業員] ; &[部門]名称 = "営業部")
```

▼ 次の例は、変数 “ Var ” に入力された情報を検索します：

```
QUERY ([規定] ; [規定]テキスト = Var)          `変数「Var」と一致するすべての値を探す
```

この検索処理では変数 “ Var ” の内容により多くの異なる検索結果が得られます。例えば、次のようなものです(ワイルドカードの例)：

■ 変数 “ Var ” の内容が “ @Copyright ” の場合には、[規定]テーブルの “ テキスト ” フィールドが “ Copyright ” という文字列で終わるすべてのレコードを検索します。

■ 変数 “ Var ” の内容が “ Copyright@ ” の場合には、[規定]テーブルの “ テキスト ” フィールドが “ Copyright ” という文字列で始まるすべてのレコードを検索します。

■ 変数 “ Var ” の内容が “ @Copyright@ ” の場合には、[規定]テーブルのテキストフィールドに “ Copyright ” という文字列が最低1つ以上含まれるすべてのレコードを検索します。

QUERY SELECTION

QUERY SELECTION ({テーブル})

引数	タイプ	説明
テーブル	テーブル	検索対象となるテーブル

QUERY SELECTION ({テーブル}; 検索条件 ; {*})

引数	タイプ	説明
テーブル	テーブル	検索対象となるテーブル
検索条件		検索条件
*		継続検索フラグ

QUERY SELECTIONコマンドは、**QUERY**コマンドに似ています。相違点は、検索する範囲が異なるだけです。**QUERY**コマンドはテーブル全体のレコードを検索するのに対して、**QUERY SELECTION**コマンドはテーブルのカレントセレクションのレコードを検索します。**QUERY SELECTION**コマンドには2つの形式があります。両方の形式ともプロセス内の＜テーブル＞のカレントセレクションに対して検索します。カレントプロセス内の＜テーブル＞のレコードのセレクションが返されます。インデックスフィールドを指定すると検索は最適化されます。インデックス検索を最初に実行しますので、最小限の時間で検索します。このコマンドは、プロセス内の＜テーブル＞のカレントセレクションを変更します。そして、セレクションの先頭のレコードをカレントレコードにします。

第1の形式は、＜テーブル＞用の「クエリ」エディタを表示します。ユーザはエディタ内で検索条件を構築し、絞り込み検索を実行することができます。設定した検索条件をディスクに保存することもできます。

オプション引数であるアスタリスク(*)を指定した検索を実行すると、「クエリ」エディタは表示されません。その代わりに、オプション引数アスタリスク(*)を指定した検索を実行します。

QUERY SELECTIONコマンドの第2の形式は検索対象テーブルに対して、＜検索条件＞を使用したインテリジェントな検索を実行しセレクションを生成します。

複合条件を使用して検索することもできます。複数の**QUERY SELECTION**コマンドを実行することによってこのような検索を行うことが可能です。各**QUERY SELECTION**コマンドにオプション引数アスタリスク(*)を指定することによって複合条件の検索であることを宣言します。各検索条件は論理演算子によって互いに結合されます。検索の設定方法と引数の指定方法に関する詳細は、**QUERY**コマンドを参照してください。

QUERY SELECTIONコマンドによる検索に時間がかかる場合は、4th Dimensionは自動的にサーモメータで進捗状況のメッセージを表示します。**MESSAGES ON**コマンドと**MESSAGES OFF**コマンドを使用して、メッセージを表示または非表示にすることができます。

QUERY BY FORMULA
QUERY SELECTION BY FORMULA

QUERY BY FORMULA ({テーブル};{検索条件式})
QUERY SELECTION BY FORMULA ({テーブル};{検索条件式})

引数	タイプ	説明
テーブル	テーブル	検索対象となるテーブル
検索条件式	ブール	検索条件式

QUERY BY FORMULAコマンドと**QUERY SELECTION BY FORMULA**コマンドは、＜テーブル＞からレコードを検索し、新しいセレクションを生成します。**QUERY BY FORMULA**コマンドと**QUERY SELECTION BY FORMULA**コマンドは、全く同じように機能しますが、**QUERY BY FORMULA**コマンドがテーブルのすべてのレコードを検索対象とするのに対して、**QUERY SELECTION BY FORMULA**コマンドはカレントセレクションのレコードのみを検索対象とします。

両方のコマンドは、テーブルまたはセレクションのレコードに対して＜検索条件式＞を適用します。＜検索条件式＞は、“ True(真) ” か “ False(偽) ” のいずれかの状態に評価されるブール式です。＜検索条件式＞で “ True(真) ” に評価されたレコードを新しいセレクションに追加します。

＜検索条件式＞は、フィールドと変数を比較するだけの単純なものから、計算したりリレート先テーブルの情報を評価するような複雑なものまで処理します。＜検索条件式＞には、4th Dimensionの関数やユーザが作成した関数やファーミュラを使用することができます。文字フィールドやテキストフィールドに対して作業を行う場合は、＜検索条件式＞にワイルドカードを使用することもできます。

両方のコマンドとも、検索処理が完了した時点で新しいセレクションを生成し、その先頭のレコードをカレントレコードとします。

これらのコマンドではシーケンシャルな検索を行い、インデックスを使用した検索は行いません。**QUERY BY FORMULA**コマンドと**QUERY SELECTION BY FORMULA**コマンドは、インデックスが設定されたフィールドを検索する場合でも、**QUERY**コマンドを使用した場合よりも処理速度は遅くなります。検索に要する時間は、テーブルやセレクション中のレコード数に比例します。

最初の3つの例は、**QUERY**コマンドの最初の3つの例と同じです。**QUERY**コマンドとの違いは、常にシーケンシャルな検索を行い、**QUERY**コマンドのような最適化された検索は行わないことです。**QUERY SELECTION BY FORMULA**コマンドを使用した検索では、処理対象は常にカレントセレクションとなります。

▼ 次の例は、“ Smith ” という名字の人のレコードをカレントセレクション内で検索します。4th Dimensionは大文字、小文字の種別を考慮しない点に注意してください。名字のフィールドにはインデックスが設定されていますが、インデックスを無視しシーケンシャルな検索を行います：

```
` 名字が “ Smith ” という人を探す
QUERY SELECTION BY FORMULA ([従業員] ; [従業員]名字 = "smith")
```

▼ 次の例は、“ John Smith ” という人のレコードをカレントセレクション内で検索します。ブール式の評価を制御するために括弧を使用することに注意してください：

```
` 名前が “ John ” で、名字が “ Smith ” という人を探す
QUERY SELECTION BY FORMULA ([従業員] ; ([従業員]名前 = "john") & ([従業員]名字 = "smith"))
```

▼ 次の例は、名字が “ Smith ” または “ Jones ” という人のレコードをカレントセレクション内で検索します：

```
` 名字が “ Smith ” または “ Jones ” という人を探す
QUERY SELECTION BY FORMULA ([従業員] ; ([従業員]名字 = "smith") | ([従業員]名字 = "jones"))
```

▼ 次の例は、すべての年度の12月に作成された送り状のレコードを検索します。これは、**Month of**関数を各レコードに適用して検索します。このような検索は月の情報を別のフィールドとして持たない限り、他の方法では実現できません：

```
` 12月に作成された送り状を探す
QUERY BY FORMULA ([送り状] ; Month of ([送り状]作成日付) = 12)
```

▼次の例は、名前が全角で5文字(半角で10文字)以上の人のレコードを検索します：

```
` 名前が全角で5文字以上の人を探す
QUERY  BY FORMULA ([従業員] ; Length ([従業員]名前) > 10)
```

ORDER BY

ORDER BY ({テーブル})

引数	タイプ	説明
テーブル	テーブル	ソート対象となるテーブル

ORDER BY ({テーブル};フィールド1;{ソート種別1} {;...;フィールドN;{ソート種別N}})

引数	タイプ	説明
テーブル	テーブル	ソート対象となるテーブル
フィールド	フィールド	ソートキーとなるフィールド
ソート種別	>または<	> 昇順 、 < 降順

ORDER BYコマンドには、2つの形式があります。第1の形式は、「ユーザ」モードの「並び替え...」アイテムと同じ「並び替え」ダイアログボックスを表示し、ソートキーを指定をします。

第2の形式は、引数の指示に従って、＜テーブル＞のカレントセレクションをソートします。＜テーブル＞を省略した場合には、**ORDER BY**コマンドは＜フィールド1＞を含むテーブルのカレントセレクションをソートします。1つのステートメント内で複数のソートキーを指定し、ソートすることができます。ソート処理が完了した時点で、ソートしたセレクションの先頭のレコードをカレントレコードとします。

▼ 次の例は、[住所]テーブルのカレントセレクションを“ 郵便番号 ”フィールドを第1ソートキーに、“ 名字 ”フィールドを第2ソートキーに指定しそれぞれ昇順に並べ替えます：

```
ORDER BY ([住所] ; [住所]郵便番号 ; > ; [住所]名字 ; >)
```

＜ソート種別＞で、昇順または降順にソートするのかを指定します。＜ソート種別＞が“ > ”の場合には、ソートを昇順に行います。＜ソート種別＞が“ < ”の場合には、ソートを降順に行います。＜ソート種別＞を省略した場合には、ソートを昇順に行います。

1つのフィールドのみの指定で、そのフィールドにインデックスが設定されている場合は、インデックスを使用してソートします。複数のフィールドを指定した場合には、たとえインデックスに設定されたフィールドが含まれていてもインデックスは使用しないで、ソート処理はシーケンシャルに行います。ｎ対1の自動リレートを持ったフィールドで並び替えを行うことができます。

▼ 次の例は、[送り状]テーブルのレコードを登録した日付で並び替えます：

```
If (Before)
   ORDER BY ([送り状] ; [送り状]登録日付)
End if
```

前もって**MESSAGES OFF**コマンドを実行していない限り、ソート処理を実行すると処理の進捗状況を表すインジケータ(ナンバーやサーモメータ)を表示します。インジケータの「中止」ボタンやソートウインドウの「キャンセル」ボタンをクリックすると、ソートを中止します。中止されたかどうかは、システム変数OKの値を調べるとわかります。ソートが完了していればシステム変数OKに1が、中止されていれば 0が代入されます。

▼ 次の例は、[住所]テーブルの「並び替え」ダイアログボックスを表示します：

```
ORDER BY ([住所])
```

ORDER BY FORMULA

ORDER BY FORMULA (テーブル ; ソート条件式 ; {ソート種別1}{ ;...; ソート条件式N; {ソート種別N}})

引数	タイプ	説明
テーブル	テーブル	ソート対象となるテーブル
ソート条件式	文字列または数値 または　日付 または　時間 または　ブール	ソート条件式
ソート種別	>または<	> 昇順、 < 降順

ORDER BY FORMULAコマンドは、引数の指示に従って、＜テーブル＞のカレントセレクションを並べ替えます。引数には複数のソート条件を指定することができます。

＜テーブル＞を指定しなければならない点に注意してください。デフォルトテーブルを使用することはできません。

ORDER BY FORMULAコマンドは、カレントセレクションを昇順または降順に並び替えます。＜ソート種別＞で、＜テーブル＞のレコードを昇順または降順にソートするのかを指定します。＜ソート種別＞が " > " の場合は、ソートを昇順に行います。＜ソート種別＞が " < " の場合には、ソートを降順に行います。＜ソート種別＞を省略した場合には、並び替えを昇順に行います。

ソート処理が完了した時点で、ソートしたセレクションの先頭のレコードをカレントレコードとします。

前もって**MESSAGES OFF**コマンドを実行していない限り、ソート処理を実行すると処理の進捗状況を表すインジケータ(ナンバーやサーモメータ)を表示します。インジケータの「中止」ボタンやソートウインドウの「キャンセル」ボタンをクリックすると、ソートを中止します。中止されたかどうかは、システム変数OKの値を調べるとわかります。ソートが完了していればシステム変数OKに1が、中止されていれば 0が代入されます。

▼ 次の例は、[従業員]テーブルを " 名字 " フィールドの長さをキーにして降順に並び替えます。最も長い名字を持った人がカレントセレクションの先頭になります：

```
ORDER BY FORMULA ([従業員] ; Length ([従業員]名字) ; <)
```

SET QUERY LIMIT

SET QUERY LIMIT (制限)

引数	タイプ		説明
制限	数値	→	レコード数、0=制限なし

SET QUERY LIMITコマンドは、引数＜制限＞で渡したレコード数が見つかったらすぐにカレントプロセスの次のクエリ（検索）を中止するように4th Dimensionに指示します。

例えば、＜制限＞に 1を受け渡すと、次のクエリはクエリ条件に一致した1件のレコードを見つけるとすぐにインデックスまたはデータファイルのブラウズ作業を中止します。

制限なしのクエリを再実行するには、＜制限＞に 0を渡した**SET QUERY LIMIT**コマンドを再度呼び出します。

警告：SET QUERY LIMITコマンドは、カレントプロセス内で行われた次のクエリのすべてに影響を及ぼします。そのため、SET QUERY LIMIT (制限)「制限 > 0」の呼び出しは、常に制限なしのクエリを再実行するためにSET QUERY LIMIT (0)の呼び出しが対になっていることを覚えておいてください。

SET QUERY LIMITコマンドは、下記のクエリコマンドの動作を変更します：

- **QUERY**
- **QUERY SELECTION**
- **QUERY BY EXAMPLE**
- **QUERY BY FORMULA**
- **QUERY SELECTION BY FORMULA**

これに対して、**SET QUERY LIMIT**コマンドは、**ALL RECORDS**コマンドや**RELATE MANY**コマンドなどのテーブルのカレントセレクションを変更する他のコマンドに影響を与えません。

▼“ 100万円以上の売上を獲得している顧客10人を探せ ”という要求に対応するクエリを実行するには、次のように記述します：

```
SET QUERY LIMIT (10)
QUERY ([顧客] ; [顧客]売上 > 1000000)
SET QUERY LIMIT  (10)
```

▼ **SET QUERY DESTINATION**コマンドの2番目の例を参照してください。

参照：QUERY、QUERY BY EXAMPLE、QUERY BY FORMULA、QUERY SELECTION、QUERY SELECTION BY FORMULA、SET QUERY DESTINATION

SET QUERY DESTINATION

SET QUERY DESTINATION (配置先タイプ ; {; 配置先オブジェクト})

引数	タイプ		説明
配置先タイプ	数値	→	0 = カレントセレクション 1 = セット 2 = 命名セレクション 3 = 変数
配置先オブジェクト	文字列または変数	→	セット名、または命名セレクション名 変数名

SET QUERY DESTINATIONコマンドは、カレントプロセスの次のクエリの結果を配置する場所を4th Dimensionに指示することができます。

引数＜配置先タイプ＞に配置場所のタイプを指定します。4th Dimensionには、次の表のようなあらかじめ定義された定数を持っています：

定数	タイプ	値
Into current selection	倍長整数	0
Into set	倍長整数	1
Into named selection	倍長整数	2
Into variable	倍長整数	3

次の表にしたがって、オプション引数＜配置先オブジェクト＞にクエリ自身の配置先を指定します：

引数＜配置先タイプ＞	引数＜配置先オブジェクト＞
0（カレントセレクション）	引数を省略します。
1（セット）	（既存または作成した）セット名を渡します。
2（命名セレクション）	（既存または作成した）命名セレクション名を渡します。
3（変数）	（既存または作成した）変数名を渡します。

例えば、

SET QUERY DESTINATION (Into current selection)

次のクエリで見つかったレコードは、最終的にはそのクエリで関係したテーブルの新しいカレントセレクションの中に配置されます。

SET QUERY DESTINATION (Into set ; "**myセット**")

次のクエリで見つかったレコードは、最終的には“ Myセット ”の中に配置されます。クエリで関係したテーブルのカレントセレクションとカレントレコードはそのまま変わりません。

SET QUERY DESTINATION (Into named selection ; "**my命名**")

次のクエリで見つかったレコードは、最終的には命名セレクション“ My命名 ”の中に配置されます。クエリで関係したテーブルのカレントセレクションとカレントレコードはそのまま変わりません。

SET QUERY DESTINATION (Into variable ; "**my変数**")

次のクエリで見つかったレコードは、最終的には“ My変数 ”の中に配置されます。クエリで関係したテーブルのカレントセレクションとカレントレコードはそのまま変わりません。

警告：SET QUERY DESTINATIONコマンドは、カレントプロセス内で行われた次のクエリのすべてに影響を及ぼします。そのため、SET QUERY DESTINATION (配置先タイプ#0)の呼び出しは、常に通常のクエリを再実行するためにSET QUERY DESTINATION (0)の呼び出しが対になっていることを覚えておいてください。

SET QUERY DESTINATIONコマンドは、下記のクエリコマンドの動作を変更します：

- **QUERY**
- **QUERY SELECTION**
- **QUERY BY EXAMPLE**
- **QUERY BY FORMULA**
- **QUERY SELECTION BY FORMULA**

これに対して、**SET QUERY DESTINATION**コマンドは、**ALL RECORDS**コマンドや**RELATE MANY**コマンドなどのテーブルのカレントセレクションを変更する他のコマンドに影響を与えません。

▼ 次の例は[電話帳]テーブルにレコードを表示するフォームを作成して、そのフォームに(アルファベット26文字の)“ asRolodex ”と名付けられたタブコントロールと[電話帳]レコードを表示するサブフォームを作成します。タブコントロールから任意のタブを選択することにより、そのタブ上の文字で始まるレコードを表示することができます。

アプリケーションでは、[電話帳]テーブルは一連の固定データを持っているので、任意のタブを選択するたびにクエリを実行する必要はありません。このように、貴重なデータベースエンジン時間を節約することができます。

これを行うために、必要な時に再利用できる命名セレクションの中にクエリを転送することができます。次のコードは「asRolodex」タブコントロールのオブジェクトメソッドです；

```
`「asRolodex」タブコントロールのオブジェクトメソッド
Case of
   \ (Form even=On Load)
      `フォームが画面上に現れる前に文字列配列とブール配列を初期化する。
      `これらの配列はタブ上の文字をクエリを実行する際に使用される。
      ARRAY STRING (1 ; asRolodex ; 26)
      ARRAY BOOLEAN (abQueryDone ; 26)
      For ($vlElem ; 1 ; 26)
         asRolodex{$vlElem}:=Char(64+$vlElem)
         abQueryDone{$vlElem}:=FaIse
      End for
   \ (Form event=On Clicked)
      `タブコントロール上をクリックすると、そのタブ上の文字に対応するクエリが
      `実行されるかどうかをチェックする。t
      If (Not(abQueryDone{asRolodex}))
         `実行されない場合、命名セレクションに次のクエリを転送する。
         SET QUERY DESTINATION(Into named selection ; "Rolodex"+
               asRolodex{asRolodex})
         `クエリを実行する。
         QUERY([電話帳];[電話帳]名字=asRolodex{asRolodex}+"@")
         `通常のクエリ処理に戻す
         SET QUERY DESTINATION(Into current selection)
         `次回その文字を選択した場合は、クエリを再実行しません。
         abQueryDone{asRolodex}:=True
      End if
      `選択された文字に対応したレコードを表示する命名セレクションを使用する。
      USE NAMED SELECTION("Rolodex"+asRolodex{asRolodex})
   \ (Form event=On Unload)
      `フォームが画面から見えなくなったら、作成した命名セレクションを消去する。
      For ($vlElem ; 1 ; 26)
         If(abQueryDone{$vlElem})
            CLEAR NAMED SELECTION("Rolodex"+asRolodex{$vlElem})
         End if
      End for
      `不要になった2つの配列をクリアする。
      CLEAR VARIABLE(asRolodex)
      CLEAR VARIABLE(abQueryDone)
End case
```

▼ 次の例は下記の“ 重複値検査 ”プロジェクトメソッドは、テーブル内のフィールド番号に重複する値がないかどうか検査することができます。カレントレコードは、既存または新しく作成したレコードです：

```
` 「重複値検査」プロジェクトメソッド
`重複値検査 (ポインタ ; ポインタ { ; ポインタ... } ) → ブール
`重複値検査 ( ->ブール ; ->フィールド1 { ; ->フィールド2... } ) → Yes or No
C_BOOLEAN($0 ; $2)
C_POINTER(${1})
C_LONGINT($vlField ; $vlNbFields ; $vlFound ; $vlCurrentRecord)
$vlNbFields:=Count parameters-1
$vlCurrentRecord:=Record number($1->)
If ($vlNbFields>0)
    If ($vlCurrentRecord#-1)
        If ($vlCurrentRecord<0)
            `カレントレコードがまだ保存されていない新規レコードの場合、
            `1件のレコードが見つけるとすぐに次のクエリを中止できます。
            SET QUERY LIMIT(1)
        Else
            `カレントレコードが既存レコードの場合、
            `2件のレコードが見つけるとすぐに次のクエリを中止できます。
            SET QUERY LIMIT(2)
        End if
        `クエリはカレントレコードやカレントセレクションを変更することなく
        `$viFoundの中にクエリ自身の結果を返します。
        SET QUERY DESTINATION(Into variable ; $vlFound)
        `指定されたフィールド番号によってクエリを作成する。
        Case of
            \ ($vlNbFields=1)
                QUERY($1-> ; $2->=$2->)
            \ ($vlNbFields=2)
                QUERY($1->; $2->=$2-> ; *)
                QUERY($1->;  &  ;$3->=$3->)
        Else
                QUERY($1-> ; $2->=$2-> ; *)
                For ($vlField;2;$vlNbFields-1)
                    QUERY($1-> ;  &  ; ${1+$vlField}->=${1+$vlField}->;*)
                End for
                QUERY($1->;  &  ; ${1+$vlNbFields}->=${1+$vlNbFields}->)
        End case
        SET QUERY DESTINATION(Into current selection) `通常のクエリ処理に戻す。
        SET QUERY LIMIT(0) `クエリに制限はありません。
        `クエリ処理の結果
```

```
        Case of
            \ ($vlFound=0)
                $0:=True `重複した値がない場合
            \ ($vlFound=1)
                If ($vlCurrentRecord<0)
                    $0:=False `新規レコードと同じ値を持つ既存レコードが見つかった場合
                Else
                    $0:=True `重複した値がない場合
                End if
            \ ($vlFound=2)
                $0:=False ` どちらの場合であっても、値が重複している場合
        End case
    Else
        If (<>DebugOn) `開発バージョンの場合、デバッグウインドウを開く。
            TRACE `警告！重複しない値がカレントレコードなしで呼び出される。
        End if
        $0:=False `結果は保証されない。
    End if
Else
    If (<>DebugOn) `開発バージョンの場合、デバッグウインドウを開く。
        TRACE `警告！重複しない値がクエリ条件なしで呼び出される。
    End if
    $0:=False `結果は保証されない。
End if
```

このプロジェクトメソッドをあなたのアプリケーションにインプリメントされると、次のように記述します：

```
  ` ...
If (重複値検査(->[交渉];->[交渉]会社);->[交渉]名字 ; ->[交渉]名前)
  `ここで、重複していない値を持つレコードに関する任意の処理を行う。
Else
  ALERT("この会社はすでに交渉を行っています。.")
End if
  ` ...
```

参照：QUERY、QUERY BY EXAMPLE、QUERY BY FORMULA、QUERY SELECTION, QUERY SELECTION BY FORMULA、SET QUERY LIMIT

レコードロックコマンド　43

4th Dimensionと4D Server／4D Clientは、マルチユーザまたはマルチプロセスのコンフリクトを防ぐことによってマルチユーザデータベースを自動的に管理します。2人のユーザまたは2つのプロセスが、同時に同じレコードやオブジェクトを修正することはできませんが、ロード(読み込む)することは可能です。

この章のマルチユーザコマンドを使用しなければならない状況がいくつかあります：

■ プログラミング言語を使用してレコードを更新する。

■ マルチユーザ環境でカスタムユーザインタフェースを使用する。

■ トランザクション内でリレート変更を保存する。

マルチユーザデータベースでコマンドを使用するときに注意すべき重要な概念が3つあります：

■ 各テーブルは、読み込み専用(リードオンリー)または読み書き可能(リードライト)のどちらかに設定される。

■ レコードは、ロードされた時点でロック状態またはロードされてない状態でロック解除状態になる。

■ ロックされたレコードは、更新することはできない。

この章では、マルチユーザデータベースを使用して作業を行う人を“ローカルユーザ”と呼びます。マルチユーザデータベースを使用する他の人を“他のユーザ”と呼びます。この節では、ローカルユーザの観点で説明します。また、マルチプロセスの観点から、データベース上で処理を実行しているプロセスを“カレントプロセス”と呼びます。その他の実行しているプロセスは、“他のプロセス”と呼びます。この章では、カレントプロセスの観点で説明します。この章では、「ルーチン」エディタの「Record Locking」テーマ内にあるレコードロックコマンドについて説明します。

READ WRITE
READ ONLY
Locked
Read only state
LOAD RECORD
UNLOAD RECORD
LOCKED ATTRIBUTES

レコードのロック

ローカルユーザは、ロックされたレコードをロードすることは可能ですが、更新することはできません。他のユーザが、レコードを更新するためにレコードをロードした時点で、そのレコードはロックされます。この場合、レコードを更新しているユーザだけが、そのレコードをロックされていない状態で取り扱うことができます。その他のすべてのユーザは、レコードがロック状態になるので更新することはできません。

テーブルをリードライト状態に設定していない場合、ロードされたテーブルは決してロックされていない状態にはなりません。

リードオンリー状態とリードライト状態

データベース上の各テーブルは、データベースの各ユーザに対してリードオンリー状態とリードライト状態のいずれかになっています。リードオンリーとは、テーブルからレコードをロードすることはできるけど更新することができない状態です。つまり、テーブルが常にロックされローカルユーザが更新できない状態になっていることを意味します。リードライトとは、他のユーザがロックしていない場合にレコードをロードし、さらに更新することができる状態です。

テーブルのステータスを変更すると、その変更が次からロードされるレコードに影響を及ぼすことに注意してください。そのため、テーブルステータスを変更する際にちょうどロードされたレコードがあっても、そのレコードはステータス変更によって影響を受けることはありません。

リードオンリー状態

テーブルがリードオンリー状態に設定されると、ロードしたレコードは常にロックされます。つまり、レコードの表示と印刷を行うことはできますが、更新することはできません。

このリードオンリー状態は、既存レコードを編集中の場合にのみ適用されることに注目してください。つまり、新規レコードを作成する場合には何の意味も持ちません。そのため、「ユーザ」モードの「更新」メニューにある「新規レコード」メニューアイテムを選択したり、あるいは**CREATE RECORD**コマンドや**ADD RECORD**コマンドを使ってリードオンリーテーブルにレコードを追加することができます。

4D Server／4D Clientは、レコードに対する書き込み動作を伴わないコマンドが実行されるテーブルを自動的にリードオンリー状態にします。次の表に、テーブルを自動的にリードオンリー状態にするコマンドを示します。

DISPLAY SELECTION	**EXPORT TEXT**	**REPORT**
DISTINCT VALUES	**GRAPH TABLE**	**SELECTION TO ARRAY**
EXPORT DIF	**PRINT LABELS**	
EXPORT SYLK	**PRINT SELECTION**	

これらのコマンドを実行する前に、4th Dimensionはテーブルのそのときの状態(リードオンリーまたはリードライト)を保存します。コマンドを実行した後でテーブルの状態を復元します。

リードライト状態

リードライト状態のテーブルからロードされたレコードは、他のユーザがそのレコードを先にロックしていなければアンロックになります。レコードが他のユーザによってロックされている場合にはレコードをロードすることはできますが、更新結果を保存することはできません。

テーブルがリードライト状態に設定され、ロードしたレコードがアンロックになった時点で初めて、そのレコードの更新が可能になります。

あるユーザがリードライトモードのテーブルからレコードをロードすると、修正するためにレコードをロードできるユーザは他にはいません。しかし、他のユーザは「ユーザ」モード内で手動により、あるいは**CREATE RECORD**コマンドや**ADD RECORD**コマンドを使ってそのテーブルにレコードを追加することはできます。

データベースが開かれたり、または新規プロセスが起動した際のすべてのテーブルのデフォルト状態はリードライトです。

テーブルステータスの変更

READ ONLYコマンドおよび**READ WRITE**コマンドを使って、テーブルステータスを変更することができます。あるレコードをリードオンリー状態またはリードライト状態にするために任意テーブルのステータスを変更したい場合は、そのレコードをロードする前にコマンドを実行する必要があります。すでにロードされたレコードは、この**READ ONLY**コマンドおよび**READ WRITE**コマンドによって影響を受けることはありません。

各プロセスは、データベース内の各テーブルに対する独自の状態（リードオンリーまたはリードライト）を持っています。

レコードのロード 、更新、アンロード

ローカルユーザがレコードを更新するためには、テーブルがリードライト状態でかつ、ロードしたレコードがアンロックになっていなければなりません。

次のコマンドは、レコードをロードするコマンドです：

注：下記のコマンドは、バージョン3までのコマンドです。バージョン6で新規に追加されたコマンドについて掲載してません。新規コマンドに関しては、各コマンドの節で説明しています。

ALL RECORDS	**MODIFY RECORD**	**SCAN INDEX**
APPLY TO SELECTION	**MODIFY SELECTION**	**QUERY**
ARRAY TO SELECTION	**NEXT RECORD**	**QUERY BY FORMULA**
CREATE RELATED ONE	**OLD RELATED MANY**	**QUERY BY INDEX**
DISTINCT VALUES	**OLD RELATED ONE**	**QUERY EXAMPLE**
FIRST RECORD	**ONE RECORD SELECT**	**QUERY SELECTION**
GOTO RECORD	**PREVIOUS RECORD**	**QUERY SELECTION BY FORMULA**
GOTO SELECTED RECORD	**PROJECT SELECTION**	**ORDER BY FORMULA**
JOIN	**REDUCE SELECTION**	**ORDER BY**
LAST RECORD	**RELATE MANY**	**USE NAMED SELECTION**
LOAD RECORD	**RELATE ONE**	**USE SET**

コマンドは、(該当するものがある場合)カレントレコードをロードし、レコードをロックまたはアンロックにします。レコードは、テーブルのその時の状態(リードオンリーまたはリードライト)に応じてロードされます。

レコードが自動リレートされている場合には、自動リレートを行うコマンドによってリレートテーブルからロードされます。これに関する詳細は、第38章を参照してください。

テーブルがリードオンリー状態になっている場合には、テーブルからロードされたレコードはロックされます。ロックされたレコードは、保存および削除ができません。リードオンリー状態は他のユーザがレコードをロード 、更新、保存などの処理を実行している時には望ましい状態です。

テーブルがリードライト状態になっている場合には、テーブルからロードされたレコードは他のユーザが先にロックしていない限りアンロックになります。アンロックされたレコードは、テーブルに保存することができます。テーブルは、レコードをロード 、更新、保存などの処理を実行する直前にのみリードライト状態にしなければなりません。

レコードがロックされている場合(**Locked**関数がTrueの場合)には、レコードを**LOAD RECORD**コマンドでロードし再びレコードがロックされているかどうかを調べます。レコードがアンロックになるまで(**Locked**関数がFalseになるまで)この処理を繰り返します。

レコードへの修正が完了したら、**UNLOAD RECORD**コマンドを使って、そのレコードを解放(他のユーザに対してアンロック)しなければなりません。レコードがアンロードされないと、他のカレントレコードが選択されるまで、すべての他のユーザに対してロックされた状態になります。テーブルのカレントレコードへの変更は、前のカレントレコードを自動的にアンロックにします。カレントレコードを変更しない場合は、確実に**UNLOAD RECORD**コマンドを呼び出す必要があります。これは既存レコードに適用します。新しいレコードは作成されると、そのレコードが属するテーブルの状態に関係なく保存することができます。

LOCKED ATTRIBUTESコマンドを使用すると、ユーザやプロセスがレコードをロックしているかどうかを調べることができます。

アンロックされたレコードをロードするためのループ処理

次の例は、アンロックされたレコードをロードするための最も単純なループ処理を示しています：

```
READ WRITE ([顧客])                ` テーブルの状態をリードライトにする
Repeat                             ` レコードがアンロックになるまで繰り返す
  LOAD RECORD ([顧客])             ` ロック情報を得るためにレコードをロードする
Until (Not (Locked ([顧客])))
` ここでレコードに対する処理を行う
READ ONLY ([顧客])                 ` テーブルの状態をリードオンリーにする
```

このループ処理はレコードがアンロックされるまで、永久に繰り返します。

このようなループ処理は、ループが終了するまで待たされても、レコードが他のユーザにロックされる可能性がほとんどない場合にのみ使用することができます。

従って、プロシージャを使用してレコードを更新するような場合以外では使用することはできません。

次の例は、アンロックされたレコードをロードして、更新するためのループ処理を示しています：

```
READ WRITE ([在庫])                ` テーブルの状態をリードライトにする
Repeat                             ` レコードがアンロックになるまで繰り返す
  LOAD RECORD ([在庫])             ` ロック情報を得るためにレコードをロードする
Until (Not (Locked ([在庫])))
[在庫]部品数量:=[在庫]部品数量 - 1 ` レコードを更新
SAVE RECORD ([在庫])               ` レコードを保存
UNLOAD RECORD ([在庫])   ` 他のユーザが更新できるようにレコードをアンロック
READ ONLY ([在庫])                 ` テーブルの状態をリードオンリーにする
```

MODIFY RECORDコマンドは、自動的にレコードがロックされているかどうかを調べ、ロックされている場合には、それをユーザに通知します。次の例は、**Locked**関数を使用してレコードの状態を調べ、**MODIFY RECORD**コマンドの自動的な通知処理を無効にします。レコードがロックされている場合には、ユーザが処理を中断できるようにします。

次の例は、[コマンド]テーブルのカレントレコードがロックされているかどうかを調べます。ロックされている場合は、プロセスがメソッドによって2秒間延期されます。この技法は、マルチユーザやマルチプロセスの場合に用いられます：

```
Repeat
   ` リードライト状態にする必要はない
  READ ONLY ([コマンド])
  QUERY ([コマンド])
     ` 検索がうまくいくと、レコードがいくつか返される
  If ((OK=1) & (Records in selection ([コマンド]) > 0))
     ` テーブルをリードライト状態に設定する
    READ WRITE ([コマンド])
    ` レコードがアンロックの場合
    LOAD RECORD ([コマンド])
    ` レコードがアンロックになるまでループする
    While (Locked ([コマンド]) & (OK=1))
       ` どのユーザによってレコードがロックされているのかを調べる
      LOCKED ATTRIBUTES ([コマンド] ; $ﾌﾟﾛｾｽ番号 ; $ユーザ ; $マシン ; $プロセス名)
      If ($ﾌﾟﾛｾｽ番号=-1) ` レコードが削除されたか？
        ALERT ("レコードが合間に削除されてしまいました。")
        OK:=0
      Else    ` シングルユーザ環境？
        If ($ユーザ="")
          $ユーザ:="あなた"
        End if
      CONFIRM ("レコードはすでに " + $ユーザ + " によって " + $プロセス名 +
       " プロセス内で使用されています。")
        If (OK=1)    ` もし、数秒間、延期したい場合
          DELAY PROCESS (Current process ; 120)
           ` レコードをロードしてみる
          LOAD RECORD ([コマンド])
        End if
      End if
    End while
    ` レコードがアンロックされる
    If (OK=1)
       ` レコードを修正することができる
      MODIFY RECORD ([コマンド])
      UNLOAD RECORD ( [コマンド])
    End if
    ` リードオンリー状態に切り替える
    READ ONLY ([コマンド])
    OK:=1
  End if
Until (OK=0)
```

マルチユーザデータベース上でのコマンドの使用

多くのコマンドは、レコードがロックされていることを検知した時点で、特定の処理を実行します。これらのコマンドは、レコードがロックされていない場合には、通常どおりの処理を遂行します。次のリストに、レコードがロックされていることを検知した場合の各コマンドの処理を示します。

■ **MODIFY RECORDコマンド：**レコードが使用されていることを示すダイアログボックスを表示し、該当するレコードを表示しません。従って、ユーザはレコードを修正することができません。「ユーザ」モードでは、レコードはリードオンリー状態で表わされています。

■ **MODIFY SELECTIONコマンド：**ユーザがレコードをダブルクリックして修正しようとした場合を除いて通常どおりの処理を行います。**MODIFY SELECTION**コマンドは、**MODIFY RECORD**コマンドと同様にレコードが使用されていることを示すダイアログボックスを表示します。

■ **APPLY TO SELECTIONコマンド：**ロックされたレコードをロードしますが、そのレコードは更新されません。**APPLY TO SELECTION**コマンドは、特別な処置を行わずにテーブルからレコードを読み取ります。ロックされたレコードを検知すると、そのレコードをLockedSetと呼ばれるシステムセットに格納します。

■ **DELETE SELECTIONコマンド：**ロックされたレコードを削除しないで読み飛ばします。ロックされているレコードを、LockedSetと呼ばれるシステムセットに格納します。

■ **DELETE RECORDコマンド：**レコードがロックされている場合には何も行いません。エラーも全く返しません。このコマンドを実行する前にレコードがアンロックされていることを確認する必要があります。

■ **SAVE RECORDコマンド：**レコードがロックされている場合には何も行いません。エラーも全く返しません。このコマンドを実行する前にレコードがアンロックされていることを確認する必要があります。

■ **ARRAY TO SELECTIONコマンド：**ロックされたレコードは保存しません。

■ **GOTO RECORDコマンド：**マルチユーザデータベース上では、他のユーザがレコードを削除したり追加することができます。従って、レコード番号が変更される場合がありますので、マルチユーザデータベース上でレコード番号を使用して直接レコードを参照する場合には、十分注意してください。

■ **セットコマンド：**セットに従っている情報を他のユーザが変更する可能性がありますので、セットの使用には細心の注意が必要です。

Locked

Locked (テーブル) → ブール

引数	タイプ	説明
テーブル	テーブル	レコードのロックを調べるテーブル

Locked関数は、＜テーブル＞のカレントレコードがロックされているかを調べます。また、この関数はレコードがロックされているかどうかを検出します。そして、レコードへの延期をなくしたり、処理を読み飛ばす選択をユーザに与えます。

Locked関数が " True " を返す場合、レコードは他のユーザにロックされていますため、レコードを保存することはできません。この場合には、**LOAD RECORD**コマンドを使用して、**Locked**関数が " False " を返すまでレコードのロードを繰り返します。

Locked関数が " False " を返す場合、レコードはアンロックされています。これはレコードが他のすべてのユーザに対してロックされることを意味します。ローカルユーザだけがレコードを修正、保存することができます。**Locked**関数が " True " を返すには、テーブルがリードライト状態でなければなりません。

削除されたレコードをロードしようとすると、**Locked**関数は " True " を返し続けます。存在しないレコードへのこれ以上の延期を避けるために、**LOCKED ATTRIBUTES**コマンドを使用します。レコードが削除されている場合には、**LOCKED ATTRIBUTES**コマンドはプロセス引数に -1を返します。

トランザクション処理の実行中にも、レコードがロックされているかどうかを調べるために**LOAD RECORD**コマンドと**Locked**関数を使用します。レコードがロックされている場合には、トランザクション処理をキャンセルするのが一般的です。

前節に示した「アンロックされたレコードをロードするためのループ処理」の例を参照してください。

LOCKED ATTRIBUTES

LOCKED ATTRIBUTES ({テーブル} ; プロセス ; ユーザ ; マシン ; プロセス名)

引数	タイプ	説明
テーブル	テーブル	ロックされたレコードを調べるテーブル
プロセス	倍長整数	プロセス参照番号
ユーザ	文字列	マルチユーザの場合、ユーザ名
マシン	文字列	マルチユーザの場合、マシン名
プロセス名	文字列	プロセス名

LOCKED ATTRIBUTESコマンドは、レコードをロックする＜プロセス＞、＜ユーザ＞、＜マシン＞、および＜プロセス名＞の各変数にプロセス参照番号、ユーザ名、マシン名およびプロセス名を返します。レコードがロックされている場合には、この情報をダイアログボックスに表示して、ユーザに警告します。

レコードがロックされていない場合は、＜プロセス＞は 0を返し、＜ユーザ＞、＜マシン＞、および＜プロセス名＞は空の文字列を返します。リードライト状態でロードしようとしたレコードが削除されている場合には、＜プロセス＞は -1を返し、＜ユーザ＞、＜マシン＞、および＜プロセス名＞は空の文字列を返します。

シングルユーザ環境では、このコマンドはレコードがロックされている場合にのみ＜プロセス＞と＜プロセス名＞に値を返します。この場合に、＜ユーザ＞と＜マシン＞は空の文字列を返します。

引数＜ユーザ＞には、たとえユーザ名が空白であっても、4th Dimensionのパスワードシステムからユーザ名が返されます。もし、パスワードシステムがない場合は、“ 管理者 ” が返されます。

引数＜マシン＞に返されるのは、オペレーションシステムのテーブル共有で設定されているオーナー名です。名前の変更は、マシンが再起動するまで行われません。

前節に示した「アンロックされたレコードをロードするためのループ処理」の例を参照してください。

READ WRITE

READ WRITE ({テーブル})

引数	タイプ	説明
テーブル	テーブル	リードライト状態にするテーブル

READ WRITE ({*})

引数	タイプ	説明
*		リードライト状態にするテーブル

READ WRITEコマンドは、それが呼び出されるプロセス内の＜テーブル＞の状態をリードライトに変更します。オプション引数＜*＞を指定すると、すべてのテーブルはリードライト状態に変更されます。

READ WRITEコマンドを呼び出した後、レコードがロードされると、他のユーザがそのレコードをロックしていない場合にはアンロックになります。このコマンドは、現在ロードしているレコードの状態を変更するものではありません。その次以降にロードしたレコードに対してのみ適用します。

すべてのテーブルのデフォルト状態は、リードライト状態です。

READ WRITEコマンドは、レコードを修正し、その結果を保存しなければならないときに使用します。また、レコードを修正しない場合でも、他のユーザに対してレコードをロックする必要のあるときに使用することもできます。テーブルをリードライトモードに設定することにより、そのテーブル上における他のユーザによる操作編集を防ぐことができます。

前節に示した「アンロックされたレコードをロードするためのループ処理」の例を参照してください。

注：このコマンドは、さかのぼってアクティブになることはありません。そのため、レコードはロードされた時点のテーブルのリードライト状態によってロードされます。リードライトモード内でリードオンリーテーブルのレコードをロードするには、そのレコードをロードする前にそのテーブルステータスをリードライトモードに変更する必要があります。

READ ONLY

READ ONLY ({テーブル})

引数	タイプ	説明
テーブル	テーブル	リードオンリー状態にするテーブル

READ ONLY ({*})

引数	タイプ	説明
*		リードオンリー状態にするテーブル

READ ONLYコマンドは、それが呼び出されるプロセス内の<テーブル>の状態をリードオンリーに変更します。このコマンドを実行した後でロードしたレコードはすべてロックされます。オプション引数<*>を指定すると、すべてのテーブルはリードオンリー状態に変更されます。

1つまたは複数のレコードを修正する必要のない場合に、**READ ONLY**コマンドを使用します。

前節に示した「アンロックされたレコードをロードするためのループ処理」の例を参照してください。

注：このコマンドは、さかのぼってアクティブになることはありません。そのため、レコードはロードされた時点のテーブルのリードライト状態によってロードされます。リードオンリーモード内でリードライトテーブルのレコードをロードするには、そのレコードをロードする前にそのテーブルステータスをリードオンリーモードに変更する必要があります。

Read only state

Read only state ({**テーブル**}) → **ブール**

引数	**タイプ**	**説明**
テーブル	テーブル	読み込み専用状態かどうかを判断するテーブル

この関数は、コールされたプロセスで＜テーブル＞が読み込み専用状態かどうかを調べます。＜テーブル＞が読み込み専用であれば、True（真）を返します。＜テーブル＞が書き込み可能状態であれば、False（偽）を返します。

▼次の例は、[送り状] テーブルの状態を判断するものです。[送り状] テーブルの状態が読み込み専用であれば、書き込み可能状態に設定し、カレントレコードを再度ロードします：

```
If (Read only state ([送り状]))
   READ WRITE ([送り状])
   LOAD RECORD ([送り状])
End if
```

注：送り状のレコードを再度ロードすることにより、ユーザはレコードを修正できるようになります。前回読み込み専用でロードされたレコードは、書き込み可能状態で再度ロードされるまで、ロックされたままです。

LOAD RECORD

LOAD RECORD ({テーブル})

引数	タイプ	説明
テーブル	テーブル	ロードするレコードの属するテーブル

LOAD RECORDコマンドは、＜テーブル＞のカレントレコードをロードします。カレントレコードが存在しない場合は、**LOAD RECORD**コマンドは何も行いません。

レコードの状態は、**Locked**関数で調べることができます：

■ ＜テーブル＞が“ リードオンリー ”状態ならば、**Locked**関数は“ True(真) ”を返し、レコードを修正することはできません。

■ ＜テーブル＞が“ リードライト ”状態でレコードがすでにロックされている場合は、レコードは“ リードオンリー ”状態になり、そのレコードを修正することはできません。

■ ＜テーブル＞が“ リードライト ”状態で、しかもレコードがロックされていない場合は、カレントプロセス内のレコードを修正することができます。その際、**Locked**関数はその他のすべてのユーザとプロセスに対して“ True(真) ”を返します。

一般に、**QUERY**コマンド、**NEXT RECORD**コマンド、**PREVIOUS RECORD**コマンドなどのコマンドは自動的にカレントレコードをロードするので、**LOAD RECORD**コマンドを使用する必要はありません。

マルチユーザ環境やマルチプロセス環境において、既存レコードを修正するには、“ リードライト ”状態でテーブルをアクセスし、しかもそのレコードが他のユーザやプロセスによってロックされていないことが必要です。レコードがロックされても、ロックされていなくていても、**LOAD RECORD**コマンドは再度カレントレコードをロードしようと試みます。また、ループ内で**LOAD RECORD**コマンドを使用することにより、レコードが“ リードライト ”状態になるか、またはユーザの介入を待つことができます。

UNLOAD RECORD

UNLOAD RECORD ({テーブル})

引数	タイプ	説明
テーブル	テーブル	ロードするレコードの属するテーブル

UNLOAD RECORDコマンドは、＜テーブル＞のカレントレコードをアンロードします。

UNLOAD RECORDコマンドは、レコードがローカルユーザに対してアンロックされている場合(他のユーザに対してロックされている場合)は、他のユーザに対してレコードをアンロックします。

UNLOAD RECORDコマンドは、メモリからレコードを解放(アンロック)しますが、カレントレコードはそのまま存在します。他のレコードがカレントレコードになると、前のカレントレコードは自動的にアンロードされ、他のユーザに対してアンロックされます。このコマンドは、レコードを修正したり自分自身のカレントレコードを他のユーザが使えるようにしたい場合は常に実行します。

レコードに大きなデータやピクチャフィールドまたは4D Drawなどの外部文書が含まれている場合に、修正の必要がなくなるまで、そのカレントレコードをメモリ内に保持したくないかもしれません。こういう場合、**UNLOAD RECORD**コマンドを使用して、メモリ内にそういうレコードを持たないでカレントレコードを保持できるようにします。また、そのフィールドの値にアクセスすることはできませんが、そのレコードによって占有されていたメモリを解放することができます。

レコードコマンド　44

この章では、「ルーチン」エディタの「Records」テーマ内にあるレコードコマンドについて説明します。この章のコマンドは、レコードを管理するためのものです。新しいレコードを作成、追加したり、既存のレコードに対し修正、複製、削除などを行うためのコマンドです。

これらのコマンドは、ユーザには見えない形でデータを管理するためのものです。従って、データやフォームは表示しません。

CREATE RECORD
DUPLICATE RECORD
Records in table
GOTO RECORD
Modified record
SAVE RECORD
Record number
DELETE RECORD
DISPLAY RECORD
Sequence number
POP RECORD
PUSH RECORD

DISPLAY RECORD

DISPLAY RECORD ({テーブル})

引数	タイプ	説明
テーブル	テーブル	レコードを表示するテーブル

DISPLAY RECORDコマンドは、カレント入力フォームを通して、＜テーブル＞のカレントレコードを表示します。このレコードは、イベントがウインドウを更新するまで表示します。このイベントとは、**ADD RECORD**コマンドを実行したり、入力フォームへ戻ったり、メニューバーに戻ることです。**DISPLAY RECORD**コマンドは、カレントレコードが存在しない場合には何も行いません。

DISPLAY RECORDコマンドは、よくオリジナルの進捗メッセージを表示するために使用されます。また、自由に動くスライドを生成するために使用されることもあります。

フォームメソッドが存在する場合は、Beforeフェーズが発生します。

▼ 次の例は、レコードを次から次へ表示します：

```
ALL RECORDS ([デモ])                          ` すべてのレコードを選択
INPUT FORM ([デモ] ; "表示")                  ` 表示フォームの設定
For ($i ; 1 ; Records in selection ([デモ]))  ` レコード全体をループ
  DISPLAY RECORD ([デモ])                     ` レコードの表示
  NEXT RECORD ([デモ])                        ` 次のレコードへ移動
End for
```

CREATE RECORD

CREATE RECORD ({テーブル})

引数	タイプ	説明
テーブル	テーブル	新しいレコードを作成するテーブル

CREATE RECORDコマンドは、＜テーブル＞に対して新しい空レコードを作成しますが、このレコードは表示されません。データ入力のために新しいレコードを作成、表示するには**ADD RECORD**コマンドを使用します。詳細は、**ADD RECORD**コマンドの節を参照してください。**CREATE RECORD**コマンドは、データ入力をプログラミング言語上で行う場合に、**ADD RECORD**コマンドの代わりに使用します。新しく作成されたレコードはカレントレコードとなり、さらにカレントセレクション(レコードが1件のみのカレントセレクション)になります。

新しいレコードは、同一テーブルに対する**SAVE RECORD**コマンドを実行するまではメモリ上にのみ存在します。新しいカレントレコードが保存される前に修正された場合(検索処理などによって)には、そのレコードは失われます。

▼ 次の例は、30日分の[伝票]テーブルのデータを[台帳]テーブルに転記します。[台帳]テーブルに対し新しいレコードを登録することによって、これを行っています。転記が終了した時点で、転記し終わった[伝票]レコードを[伝票]テーブルから削除します：

```
`本日から30日前までのレコードを探す
QUERY ([伝票] ; [伝票]作成日付 < (Current date - 30))
For ($i ; 1 ; Records in selection ([伝票]))    ` 転記が終わるまで繰り返す
  CREATE RECORD ([台帳])                        ` 新しい[台帳]レコードを作成
  [台帳]番号:=[伝票]番号                        ` [台帳]レコードへデータを転記
  [台帳]作成日付:=[伝票]作成日付
  [台帳]合計金額:=[伝票]合計金額
  SAVE RECORD ([台帳])                          ` 新しい[台帳]レコード登録
  NEXT RECORD ([伝票])                          ` 次の[伝票]レコードへ移動
End for
DELETE SELECTION ([伝票])                       ` [伝票]レコードを削除
```

DUPLICATE RECORD

DUPLICATE RECORD ({テーブル})

引数	タイプ	説明
テーブル	テーブル	複製元のカレントレコードの属するテーブル

DUPLICATE RECORDコマンドは、カレントレコードを複製して同じ＜テーブル＞内に新しいレコードを作成します。新しいレコードは、カレントレコードとなります。カレントレコードが存在しない場合には、**DUPLICATE RECORD**コマンドは何も行いません。新しいレコードを保存するには、**SAVE RECORD**コマンドを使用しなければなりません。

DUPLICATE RECORDコマンドは、データ入力中にも実行可能です。これにより、現在表示しているレコードのコピーを作ることができます。複写元のレコードを修正した場合には、**SAVE RECORD**コマンドを一番最初に実行しておくことを忘れないでください。また、この場合のフォームのBeforeフェーズは発生しないことに注意してください。

Modified record

Modified record ({テーブル}) → ブール

引数	タイプ	説明
テーブル	テーブル	判定するテーブル

Modified record関数は、レコードが更新されると“ True(真) ”を返します。それ以外は“ False(偽) ”を返します。この関数は、保存する必要があるレコードかどうかを判定する場合に使用します。これは、次のレコードに進む前にカレントレコードを保存するかどうかチェックする入力フォーム上で特に有効です。**Modified record**関数は新規レコードについては、常に“ True(真) ”を返します。

▼ 次の例は、**Modified record**関数の一般的な使用方法です：

```
If (Modified record ([テーブル1]))
  SAVE RECORD ([テーブル1])
End if
```

SAVE RECORD

SAVE RECORD ({テーブル})

引数	タイプ	説明
テーブル	テーブル	カレントレコードを保存するテーブル

SAVE RECORDコマンドは、＜テーブル＞のカレントレコードを保存します。レコードがサブレコードを含む場合には、それらも親レコードとともに保存します。カレントレコードが存在しない場合には、**SAVE RECORD**コマンドは何も行いません。

SAVE RECORDコマンドは、新しく作成または修正したレコードを保存するために使用します。ユーザがフォーム中で修正したレコードは、**SAVE RECORD**コマンドで保存する必要はありません。しかし、ユーザによってフォーム中で修正されたレコードが取り消された場合でも、**SAVE RECORD**コマンドで保存することができます。

SAVE RECORDコマンドが必要とされる場合を次に示します：

■ **CREATE RECORD**コマンドまたは**DUPLICATE RECORD**コマンドで作成した新しいレコードを保存する場合

■ **RECEIVE RECORD**コマンドからのデータを保存する場合

■ メソッドによって修正したレコードを保存する場合

■ **ADD SUBRECORD**コマンド、**CREATE SUBRECORD**コマンド、**MODIFY SUBRECORD**コマンドによって新しく作成または修正したサブレコードを保存する場合

■ カレントレコードを修正するようなコマンドが実行する前に、データ入力途中で表示されているレコードを保存する場合

■ データ入力途中で新しいレコードを保存する場合

受け入れられたフォームのAfterフェーズ内で**SAVE RECORD**コマンドを実行してはいけません。もし、これを行うと、レコードが2つ保存されてしまいます。

DELETE RECORD

DELETE RECORD ({テーブル})

引数	タイプ	説明
テーブル	テーブル	削除するカレントレコードの属するテーブル

DELETE RECORDコマンドは、＜テーブル＞のカレントレコードを削除します。カレントレコードが存在しない場合には、**DELETE RECORD**コマンドは何も行いません。フォーム中では、このコマンドの代わりに**DELETE RECORD**属性のボタンを使用することができます。レコードを削除すると、＜テーブル＞のカレントセレクションは空白になります。

警告：レコードの削除は、一度行うと元に戻すことはできません。

レコードが削除されると、レコード番号は新しいレコードが作成される際に再利用されます。従って、データベースからレコードを削除する場合は、レコード番号をレコードの識別に使用しないでください。

▼ 次の例は、1件の[従業員]レコードを削除します。まず、ユーザにどの従業員レコードを削除するのかを尋ね、[従業員]レコードを検索し、見つかったレコードを削除します：

```
v削除社員:=Request ("社員番号を入力してください。")`社員番号を入力してもらう
If (OK=1)                                          `検索、削除の実行？
  QUERY ([従業員] ; [従業員]社員番号 = v削除社員)      `従業員を探す
  DELETE RECORD ([従業員])                         `従業員を削除する
End if
```

Records in table

Records in table ({テーブル}) → 数値

引数	タイプ	説明
テーブル	テーブル	レコード数を求めるテーブル

Records in table関数は、＜テーブル ＞中のレコード数を返します。**Records in selection**関数は、カレントセレクションのレコード数を返します。**Records in table**関数がトランザクション内で使用される場合は、トランザクション中に作成されたレコードには注意が必要です。

▼ 次の例は、テーブルのレコードの数を示すアラートを表示します。**Records in table**関数によって返されたレコード数を文字列に変換していることに注目してください：

```
ALERT ("従業員テーブルには" + String (Records in table ([従業員])) + "件のレコードがあります。")
```

レコードに付けられた番号の使用

本節のコマンドは、番号を直接参照することによってレコードを管理するためのものです。これらの番号は、テーブルやセレクション中のそれぞれのレコードに付けられたものです。

レコードに付けられた番号には、次の3つがあります。
- レコード番号
- レコード位置番号
- 一連番号

「レコード番号」は、レコードに対する絶対的な番号です。この番号は、レコードが作成されるごとに自動的に付けられます。レコードを削除するかテーブルを恒久的にソートするまで、レコード番号は変わることはありません。レコード番号はゼロから始まります。レコード番号は、削除されたレコード番号が新しいレコードに再利用されるため、それ独自のものではありません。そのため、レコード番号は「テーブルソート」コマンドを使用してテーブルを恒久的にソートしたり、4D Toolsを使用してデータベースを圧縮、修復すると変更されてしまいます。

「レコード位置番号」は、カレントセレクション中のレコードの位置を示す番号です。この番号は、カレントセレクションとは完全に独立した関係にあります。

セレクションを変更したりソートした場合には、レコード位置番号が変更されることがあります。レコード位置番号に番号を付ける場合は1から始まります。このコマンドに関する詳細は、第40章を参照してください。

「一連番号」は、レコードに割り当てることができる、重複することのない番号です。この番号は、各レコードに自動的に付けられるものではありません。一連番号は、1から始まり新しいレコードを作成するごとに加算していきます。レコード番号と違って、一連番号はレコードが削除されたり、またはデータベースの圧縮、修復および恒久ソートの場合でも再利用されません。そのため、一連番号を使用して、レコードに重複しないID番号を持たせることができます。トランザクション処理中に一連番号が増えると、その番号はトランザクション処理が取り消されても、減少することはありません。

レコードに付けられた番号の例

次ページの表は、レコードに付けられた番号について説明しています。表の各行は、レコードについての情報を示しています。各行の順序は、出力リストに表示された順序のレコードです。

■ **データの列**は、各レコードのデータの内容で、人名を表しています。

■ **レコード番号の列**は、絶対的なレコード番号です。この番号は、**Record number**関数によって求められます。

■ **レコード位置番号の列**は、カレントセレクション中の位置を示す番号です。
この番号は、**Selected record number**関数によって求められます。

■ **一連番号の列**は、各レコードに固有の一連番号です。この番号は、**Sequence number**関数によって求められます。この一連番号は各レコードのフィールドに格納されています。

レコードを入力した後

最初の表は、入力された後のセレクションを示しています。

■ レコードのデフォルトの順序は、レコード番号により設定されます。

■ レコード番号は0から始まります。

■ レコード位置番号と一連番号は、ともに1から始まります。

データ	レコード番号	レコード位置番号	一連番号
Tess	0	1	1
Terri	1	2	2
Sabra	2	3	3
Sam	3	4	4
Lisa	4	5	5

注：レコードのデフォルト順序は、任意のコマンドがレコードをソートしない状態でカレントセレクションを変更した後も変わりません。例えば、「ユーザ」モードで「すべてを表示」メニューコマンドを選択した後やALL RECORDSコマンドを実行した後の場合です。

レコードをソートした後

次の表は、上の表と同じセレクションを名前でソートした後の状態を示しています。

■ レコード番号は、各レコードに付けられたままです。

■ レコード位置番号は、ソートされたセレクションの中の新しい位置を示しています。

■ 一連番号は、各レコードが作成されたときに付けられたままなので変わりません。

データ	レコード番号	レコード位置番号	一連番号
Sabra	2	1	3
Lisa	4	2	5
Sam	3	3	4
Terri	1	4	2
Tess	0	5	1

レコードが削除された後

次の表は、“ Sam ”のレコードが削除されたセレクションの状態を示しています。

■ レコード位置番号だけが変更されています。(つまり、レコード位置番号はレコードを表示する順番であるとも言えます。)

データ	レコード番号	レコード位置番号	一連番号
Tess	0	1	1
Terri	1	2	2
Sabra	2	3	3
Lisa	4	4	5

レコードが追加された後

次の表は、“ Liz ”という新しいレコードが追加されたセレクションの状態を示しています。

■ 新しいレコードは、カレントセレクションの最後に加えられます。

■“ Sam ”のレコード番号が新しいレコード(“ Liz ”)に再使用されます。

■ 一連番号は加算され続けます。

データ	レコード番号	レコード位置番号	一連番号
Tess	0	1	1
Terri	1	2	2
Sabra	2	3	3
Lisa	4	4	5
Liz	3	5	6

レコードが変更およびソートされた後

次の表は、3つのレコードが選択され、さらにソートされたセレクションの状態を示しています。

■ 各レコードのレコード位置番号だけが変更されています。

データ	レコード番号	レコード位置番号	一連番号
Sabra	2	1	3
Liz	3	2	6
Terri	1	3	2

Record number

Record number ({テーブル}) → 数値

引数	タイプ	説明
テーブル	テーブル	レコード番号を求めるカレントレコードの属するテーブル

Record number関数は、＜テーブル＞のカレントレコードの絶対レコード番号を返します。カレントレコードがない場合(レコードポインタがカレントセレクションの前後にある場合)には -1を返します。カレントレコードが保存されていない新しいレコードの場合には -3を返します。

4D Toolsを使用してデータベースを圧縮したり、または「検査と修復」メニューの「データをタグにより複製した後、再作成する(数分)」オプションを使用して、データベースを修復すると、レコード番号は変更されます。レコードを削除すると、レコード番号は新しいレコードで再利用されます。トランザクション処理の間、新しく作成されたレコードは一時的なレコード番号を持ちます。トランザクション処理が受け入れられると、そのレコードは正式なレコード番号が割り当てられます。

▼ 次の例は、カレントレコードのレコード番号を変数に格納し、他に同じデータを持つレコードがないかを検索します：

```
$レコード番号:=Record number([従業員])　` レコード番号を求める
QUERY([従業員];[従業員]名字=[従業員]名字)　` 他に同じ名字がないかを検索する
` 同じ名字の従業員が見つかったらアラートボックスにそのレコード数を表示する
ALERT ("同じ名字の従業員が" + String (Records in selection ([従業員])) +
"件あります。")
GOTO RECORD ([従業員];$レコード番号)　` 元のレコードに戻る
```

GOTO RECORD

GOTO RECORD ({テーブル} ; レコード番号)

引数	タイプ	説明
テーブル	テーブル	移動するレコードの属するテーブル
レコード番号	数値	**Record number**関数で求めたレコード番号

GOTO RECORDコマンドは、＜テーブル＞のレコードをロードし選択します。引数＜レコード番号＞は、**Record number**関数で求めたレコード番号です。これは、**Selected record number**関数で求めたレコード位置番号とは異なります。このコマンドを実行するとセレクションは、選択されたレコード1件だけになります。

＜レコード番号＞がデータベースの中で最も小さいレコード番号よりも小さい場合には、最初のレコードがロードされます。＜レコード番号＞がデータベースの中で最も大きいレコード番号よりも大きい場合は、最後のレコードがロードされます。レコードが削除されたレコードのレコード番号と等しい場合には、セレクションは空になります。

注：このコマンドを使ってトランザクションの最中に仮のレコード番号を使用してはいけません。

前節の**Record number**関数の例を参照してください。

Sequence number

Sequence number ({テーブル}) → 数値

引数	タイプ	説明
テーブル	テーブル	一連番号を求めるレコードの属するテーブル

Sequence number関数は、＜テーブル＞の次の一連番号を返します。一連番号は、フォームのフィールドにデフォルト値として#N記号を設定した場合に得られる番号と同じです。(デフォルト値の設定に関する詳細は、『4th Dimension / 4D First デザインリファレンス』の「デフォルト値を設定する」を参照してください。)一連番号は、各テーブルに対して固有のものです。この番号は、テーブルに対して新しいレコードが追加されるたびに加算される決して重複することのない番号です。番号は、1から始まります。この番号は、レコードが削除されても失われません。一連番号は、その番号が使用されているかどうかに関わらず、新しいレコードが保存されるたびに1ずつ加算されます。

#N記号を使用する代わりに**Sequence number**関数を使用しなければならない理由を次に4つ示します：

■ 新しいレコードをフォームを使用せずに、メソッドを使用して作成した場合

■ 1以外の番号から使用する必要がある場合

■ 番号の増分値に1以上の数を使用する必要がある場合

■ 一連番号を他のコードの一部に使用する場合(部品コードの一部に使用する場合など)

メソッドを使用して一連番号を各レコードに格納するには、テーブル上に倍長整数型のフィールドを作成し、そのフィールドに対して一連番号を代入します。

一連番号が1以外の数値から始まる必要がある場合には、一連番号に対してその差を加算するだけで構いません。例えば、番号が1000から始まる必要がある場合には、次のようなステートメントを使用します：

```
一連番号ﾌｨｰﾙﾄﾞ:= Sequence number ([テーブル]) + 999
```

▼ 次の例は、フォームメソッドの一部です。このステートメントは、まずレコードが新しいものかどうか検査(請求書番号が空の文字列であるかどうかで判断)します。新しいレコードであれば、請求書番号を設定します。この請求書番号は、2つの情報から成り立っています。それは、一連番号とデータベースを開くときに入力された操作番号です。一連番号を、5桁の文字列として扱います：

```
If (請求書番号 = "")                              ` 新しい請求書番号を設定？
   ` 新しい請求書番号を作成
   ` 請求書番号は文字化した一連番号と操作番号で構成する
 請求書番号:= String (Sequence number ; "00000") + 操作番号
End if
```

レコードスタックの使用

PUSH RECORD　　POP RECORD

この節のコマンドは、レコードをレコードスタックに格納(プッシュ、PUSH)したり、レコードをレコードスタックから削除(ポップ、POP)したりします。

各テーブルは、固有のレコードスタックを持っています。4th Dimensionは、各レコードスタックを後入れ先出し法(Last-In-First-Out:LIFO)で管理します。レコードスタックの容量は、ハードウエア(マシンに実装されているメモリの容量)によって制限されます。

PUSH RECORDコマンドと**POP RECORD**コマンドを使用する場合には注意が必要です。プッシュされた各レコードは、メモリの空いている部分に格納されます。使用できるメモリの容量を越えて多量のレコードをプッシュした場合には、アウトオブメモリ(Out-of-memory)状態、または“スタックがいっぱいです”状態になります。

4th Dimensionは、レコードスタックにポップされないレコードを放置した状態で、メソッドを終了してメニューに戻った場合にも、レコードスタックを消去します。

PUSH RECORDコマンドと**POP RECORD**コマンドは、データの入力中に同じテーブルの内容を調べるときなどに使用すると非常に便利です。これを行うためには、まずレコードをプッシュします。次に検索を行い、見つかったレコードから情報を収集(例えば、フィールドの内容を変数にコピーする)します。最後にレコードを元の状態に戻すためにポップします。

PUSH RECORD

PUSH RECORD ({テーブル})

引数	タイプ	説明
テーブル	テーブル	プッシュするレコードの属するテーブル

PUSH RECORDコマンドは、＜テーブル＞のカレントレコード(それに付随するサブレコードも含めて)を、そのファイルのレコードスタックにプッシュします。**PUSH RECORD**コマンドは、レコードがディスクに保存される前でも実行することができます。

アンロックされたレコードをプッシュした場合、レコードはポップするかアンロードするまですべてのユーザやプロセスに対してロックされた状態になります。

▼次の例は、“顧客”テーブルのレコードをレコードスタックにプッシュします：

```
PUSH RECORD ([顧客])  ` 顧客レコードをスタックにプッシュする
```

POP RECORD

POP RECORD ({テーブル})

引数	タイプ	説明
テーブル	テーブル	ポップするレコードの属するテーブル

POP RECORDコマンドは、＜テーブル＞のレコード(それに付随するサブレコードも含めて)を、そのテーブルのレコードスタックからポップし、そのレコードを空レコードにします。

レコードをプッシュした後で、カレントセレクションをプッシュしたレコードを含まないセレクションに変更した場合は、レコードをポップしてもそのレコードは新しいカレントセレクションのカレントレコードにはなりません。ポップしたレコードを新しいカレントセレクションのカレントレコードにしたい場合には、**ONE RECORD SELECT**コマンドを実行しなければなりません。レコードを保存する前にレコードポインタを移動するようなコマンドを実行した場合は、メモリ上のコピーを失うことになります。

▼ 次の例は、“ 顧客 ” テーブルのレコードをポップしてレコードスタックからポップします：

```
PUSH RECORD ([顧客])          ` カレントレコードをプッシュする
SEARCH ([顧客])               ` いくつかのレコードを検索する
POP RECORD ([顧客])           ` 顧客レコードをスタックからポップする
ONE RECORD SELECT ([顧客])    `ポップしたレコードをカレントレコードにする
```

リレートコマンド　45

この章では、「ルーチン」エディタの「Relations」テーマ内にあるリレートコマンドについて説明します。この章のコマンド、特に**RELATE ONE**コマンドと**RELATE MANY**コマンドは自動的および自動的でないテーブル間のリレートを設定、管理するものです。テーブル間のリレートの作成に関する詳細は、『4th Dimension / 4D First デザインリファレンス』を参照してください。

AUTOMATIC RELATIONS
CREATE RELATED ONE
OLD RELATED MANY
OLD RELATED ONE
RELATE MANY
RELATE ONE
RELATE MANY SELECTION
RELATE ONE SELECTION
SAVE OLD RELATED ONE
SAVE RELATED ONE

コマンドを使用したテーブルの自動リレート

2つのテーブルは自動リレートで関連付けることができます。一般的にテーブルの自動リレートは、リレート先テーブル(nテーブル)のレコードをロードしたり、選択するために使用します。リレートを使用することで多くの処理を行うことができます。

次のような処理が含まれます：

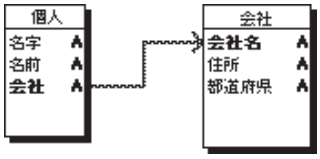
- データ入力
- 出力フォームによる画面上のリスト出力
- 帳票の印刷
- セレクションに対する検索、ソート、フォーミュラでの更新

性能を最大限に引き出すために、4th Dimensionはテーブルに対してカレントレコードとなるレコードのみ自動リレートを設定します。レコードに自動リレートが設定されていると上記のリストに示した各操作を行ったときにリレート先のテーブルからリレートレコードがロードされます。

- リレートテーブルの1レコードのみ選択するリレートの場合は、そのレコードはディスクからロードされます。

- 複数の場合には、新しいカレントセレクションがそのテーブルから作成され、セレクション内の先頭レコードがディスクからロードされます。

次の図のテーブルストラクチャを例にとってみましょう。[個人]テーブルをデータ入力のためにロードした場合は、[会社]テーブルから関連するレコードが1件選択され、ロードされます。同様に[会社]テーブルをロードした場合には、[個人]テーブルから関連する複数のレコードが選択されます。



上図において、[個人]テーブルは“ nテーブル ”として参照され、[会社]テーブルは“ 1テーブル ”として参照されます。これを解かりやすくするために、1つの会社には多くの人がいると考えてください。

同様に[個人]テーブルの“会社”フィールドはｎフィールドとして参照され、[会社]テーブルの“会社名”フィールドは1フィールドとして参照されます。

リレートフィールドが、いつもユニーク(重複しない)とは限りません。例えば、[会社]会社名フィールドは同じ値を含んだ会社のレコードをいくつも持っているかもしれません。こういう場合は、いつもユニークな別のフィールドでリレート作成することにより簡単に処理することができます。このようなフィールドの例として、会社のID番号があります。

次の表に掲げたコマンドは、コマンドの実行中にリレート先のリレートレコードをロードするために自動リレートを使用します。「1対ｎ」の欄が○になっているコマンドのみが1対ｎのリレートに対応することができます。

コマンド	1対ｎ	コマンド	1対ｎ
ADD RECORD	○	ORDER BY	×
ADD SUBRECORD	×	ORDER BY FORMULA	×
APPLY TO SELECTION	×	PRINT LABEL	×
DISPLAY SELECTION	×	PRINT SELECTION	○
EXPORT DIF	×	QUERY	○
EXPORT SYLK	×	QUERY BY FORMULA	○
EXPORT TEXT	×	QUERY SELECTION	○
MODIFY RECORD	○	REPORT	×
MODIFY SUBRECORD	×	SELECTION TO ARRAY	×
MODIFY SELECTION	○(データ入力時)	SUBSELECTION TO ARRAY	×

注：上記のコマンドはバージョン3までのコマンドです。バージョン6で新規に追加されたコマンドは含まれていません。

テーブルリレートを実行するコマンド

自動リレートとは、コマンドがレコードをロードするたびに、そのテーブルに関連する1つまたは複数のレコードを自動的に選択するという意味ではありません。レコードをロードするコマンドを使用した後で、**RELATE ONE**コマンドまたは**RELATE MANY**コマンドを使用して必要なデータを明示的に選択する必要があります。

前ページの表に掲げたコマンドの一部(**QUERY**コマンドなど)は、処理の終了後にカレントレコードをロードします。しかし、最後のレコードをロードした場合には、リレート先の関連するレコードを自動的には選択しません。また、ここでも、**RELATE ONE**コマンドまたは**RELATE MANY**コマンドを使用して必要なデータを明示的に選択する必要があります。

次の表にカレントレコードをロードするコマンドを掲げます。これらは、関連するレコードを自動的に選択するものではありません。

ALL RECORDS	**IMPORT DIF**	**QUERY BY FORMULA**
CREATE RECORD	**IMPORT SYLK**	**QUERY BY EXAMPLE**
CREATE SUBRECORD	**IMPORT TEXT**	**QUERY SELECTION**
CUT NAMED SELECTION	**LAST RECORD**	**QUERY SELECTION**
DELETE RECORD	**LAST SUBRECORD**	**BY FORMULA**
DELETE SELECTION	**NEXT RECORD**	**QUERY SUBRECORDS**
DELETE SUBRECORD	**NEXT SUBRECORD**	**ORDER BY FORMULA**
FIRST RECORD	**ONE RECORD SELECT**	**ORDER BY**
FIRST SUBRECORD	**POP RECORD**	**ORDER BY SUBSELECTION**
GOTO RECORD	**PREVIOUS RECORD**	**SCAN INDEX**
GOTO SELECTED	**PREVIOUS SUBRECORD**	**USE NAMED SELECTION**
RECORD	**QUERY**	**USE SET**

注：上記のコマンドはバージョン3までのコマンドです。バージョン6で新規に追加されたコマンドは含まれていません。

AUTOMATIC RELATIONS

AUTOMATIC RELATIONS (1リレート ; nリレート)

引数	タイプ	説明
1リレート	ブール	n対1のリレート
nリレート	ブール	1対nのリレート

AUTOMATIC RELATIONSコマンドは、データベース全体のマニュアルリレートを1次的に自動リレートに変更します。リレートは、次に**AUTOMATIC RELATIONS**コマンドを使用するまで自動リレートのままになります。

＜1リレート＞が Trueの場合、すべてのn対1のマニュアルリレートを自動リレートに設定します。＜1リレート＞が Falseの場合、前もって自動リレートにしたすべてのn対1リレートがマニュアルリレートに戻ります。

＜nリレート＞も1対nリレートに対して同じように作用します。

このコマンドは、「デザイン」モードで既に自動リレートに設定されたものに対しては無効です。

このコマンドを使用すれば、「デザイン」モードでマニュアルに設定したリレートに対して自動リレートを必要とする処理の直前で自動リレートに切り替えることができます。(リレートを検索コマンドやソートコマンドのように扱います。)自動リレートの必要がなくなった時点で、再度マニュアルリレートに戻します。

▼ 次の例は、すべてのn対1のマニュアルリレートを自動リレートに設定し、前もって自動リレートにした1対nリレートをマニュアルリレートに戻します：

AUTOMATIC RELATIONS (True ; False)

RELATE ONE

RELATE ONE ({nテーブル})

引数	タイプ	説明
nテーブル	テーブル	すべての自動リレートを設定するテーブル

RELATE ONE (nフィールド ; {選択フィールド})

引数	タイプ	説明
nフィールド	フィールド	1テーブルにマニュアルリレートするフィールド
選択フィールド	フィールド	1テーブルから選択したフィールド

RELATE ONEコマンドには、2つの形式があります。

第1の形式は、すべての自動的なn対1のリレートを＜nテーブル＞に対して設定します。このコマンドは、＜nテーブル＞の各フィールドに対して自動的なn対1のリレートを行います。これは、リレートされたテーブルにあるカレントレコードを変更します。

第2の形式は、＜nフィールド＞に関連するレコードを選択します。自動リレートデータベースである必要はありません。**RELATE ONE**コマンドは、関連するレコードを1つメモリにロードし、そのテーブルのカレントレコードをカレントセレクションにします。

オプション引数の＜選択フィールド＞は、＜nフィールド＞が文字フィールドである場合にのみ指定することができます。＜選択フィールド＞は、リレート先のフィールドでなければなりません。＜選択フィールド＞は、文字フィールド、数値フィールド、日付フィールド、時間フィールド、またはブールフィールドでなければなりません。つまりテキストフィールド、ピクチャフィールド、またはサブテーブルフィールドを選択することはできません。

＜選択フィールド＞を指定し、リレート先テーブルで複数のレコードを発見した場合は、**RELATE ONE**コマンドは＜nフィールド＞の内容と一致するレコードを選択リストに表示します。この選択リストは、左の欄にリレート先フィールドの内容を、右の欄に選択フィールドの内容を表示します。

＜nフィールド＞の内容がワイルドカード記号(@)で終了した場合でも、複数のレコードを発見しない限り、選択リストは表示しません。

次の図は、入力されたレコードと、その前面に表示された選択リストを示しています。

次のコマンドを実行すると選択リストを表示します：

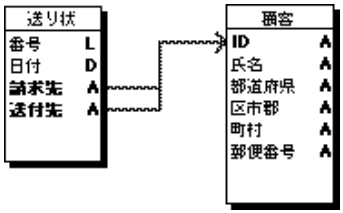
RELATE ONE ([個人]名前 ; [会社]所在地)

“ ACME ” で始まる名称を持つすべての会社のリストを、その会社の所在地と共に表示させるために “ ACME@ ” と入力しています。

＜選択フィールド＞を指定することは、テーブルのリレートを設定する時点でワイルドカード選択を指定するのと同じことです。ワイルドカード選択に関する詳細は、『4th Dimension / 4D First デザインリファレンス』を参照してください。

RELATE ONEコマンドは、サブテーブルに対するリレート機能もありますが、関連付けが適切に行われるように、親テーブルとサブテーブルの両方にリレートを行う必要があります。サブテーブルに対するリレートを使用する場合は、1回目の**RELATE ONE**コマンドで関連する親レコードをロードし、2回目の**RELATE ONE**コマンドで関連するサブレコードをロードします。

▼ 次の例は、2つの自動的でないリレートで[送り状]テーブルと[顧客]テーブルが関連付けられています。1つのリレートは、“ [送り状]請求先 ” フィールドから “ [顧客]ID ” フィールドに対するものです。もう1つのリレートは、“ [送り状]送付先 ” フィールドから “ [顧客]ID ” フィールドに対するものです。



次の図は、[送り状]テーブルの “ 請求先 ” と “ 送付先 ” を表示するための[送り状]テーブルに属するフォームを示しています。

両方とも同じテーブル([顧客]テーブル)であるため得られた情報は、変数を使用して表示します。もし[顧客]テーブルのフィールドを使用して表示した場合には、2番目のリレートから得られたデータしか表示しません。

次の2つのメソッドは、“ [送り状]請求先 ” フィールドと “ [送り状]送付先 ” フィールドに対するオブジェクトメソッドです：次は、“ [送り状]請求先 ” フィールドに対するオブジェクトメソッドです：

```
RELATE ONE ([送り状]請求先 ; [顧客]都道府県)
v都道府県1:=[顧客]都道府県
v区市郡1:=[顧客]区市郡
v町村1:=[顧客]町村
v郵便番号1:=[顧客]郵便番号
```

次は、“ [送り状]送付先 ” フィールドに対するオブジェクトメソッドです：

```
RELATE ONE ([送り状]送付先 ; [顧客]都道府県)
v都道府県2:=[顧客]都道府県
v区市郡2:=[顧客]区市郡
v町村2:=[顧客]町村
v郵便番号2:=[顧客]郵便番号
```

RELATE MANY

RELATE MANY ({1テーブル})

引数	タイプ	説明
1テーブル	テーブル	すべての1対nの自動リレートを設定するテーブル

RELATE MANY (1フィールド)

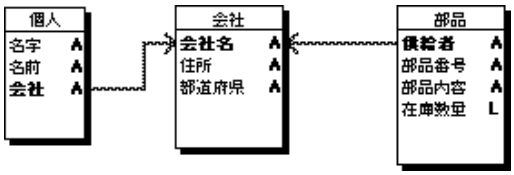
引数	タイプ	説明
1フィールド	フィールド	1フィールド

RELATE MANYコマンドには、2つの形式があります。

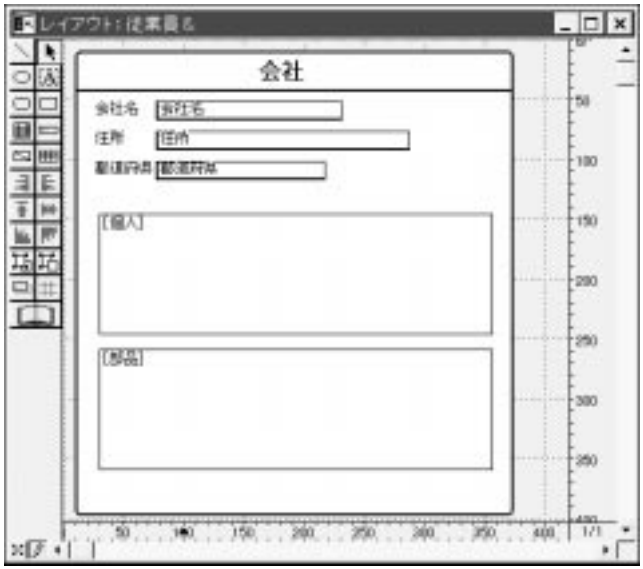
第1の形式は、＜1テーブル＞に対してすべての1対nの自動リレートを設定します。このコマンドは、＜1テーブル＞に対して1対nの自動リレートを持つ各テーブルのカレントセレクションを更新します。つまり、カレントセレクションは、1テーブルのそれぞれの関連するフィールドの現在値を反映します。このコマンドが実行される時はいつでも、nテーブルのカレントセレクションが再生されます。

第2の形式は、＜1フィールド＞に対して1対nのリレートを設定します。これは、フィールドと関連を持つテーブルのみに対しカレントセレクションを修正します。つまり、リレート先テーブルの関連するレコードは、すべてカレントセレクションになることを意味します。

次の例は、3つのテーブルが自動リレートで関連付けられています。
[個人]テーブルと[部品]テーブルは、両方とも[会社]テーブルに対してｎ対1のリレート関係にあります。



次の図は、[個人]テーブルと[部品]テーブルの両方の関連するレコードを表示するための、[会社]テーブルのフォームを示しています。



フォームを表示すると、[個人]テーブルと[部品]テーブルから関連するレコードはロードされます。しかし、[会社]テーブルをメソッド中のコマンドで選択する場合には、関連するレコードはロードされません。このような場合には、**RELATE MANY**コマンドを使用する必要があります。

次のメソッドは、[会社]テーブルの各レコードに対し、アラートボックスを表示します。アラートボックスには、会社にいる人数([個人]テーブル中の関連するレコードの数)、供給する部品の種類数([部品]テーブル中の関連するレコードの数)を表示します。(印刷の関係上**ALERT**コマンドの引数が複数行にわたっています。)**RELATE MANY**コマンドが、たとえ自動リレートの場合でも必要なことに注目してください：

```
ALL RECORDS ([会社])  ` [会社]テーブルのレコードをすべて選択
ORDER BY ([会社] ; [会社]会社名)        ` 会社名でソート
```

```
For ($i ; 1 ; Records in file ([会社]))          ` [会社]レコード数だけ繰り返す
  RELATE MANY ([会社]会社名)                     ` リレートレコードを選択
  ALERT ("会社：" + [会社]会社名 + Char (13) +"会社にいる人数："
  + String (Records in selection ([個人])) + Char (13) + "供給する部品の種類数："
  + String (Records in selection ([部品])))
  NEXT RECORD ([会社])                           ` 次の[会社]レコードへ移動
End for
```

CREATE RELATED ONE

CREATE RELATED ONE (フィールド)

引数	タイプ	説明
フィールド	フィールド	nフィールド

CREATE RELATED ONEコマンドには、2つの機能があります。関連するレコードが＜フィールド＞に対して存在しない場合(つまり、フィールドのカレントな値が適合しない場合)は、**CREATE RELATED ONE**コマンドは関連するテーブルに対して、新しいレコードを作成します。このレコードの各フィールドには何もデータを含みません。nフィールドの値を1フィールドに代入してください。**SAVE RELATED ONE**コマンドを実行することによってレコードが保存されます。

関連するレコードが存在する場合は、**RELATE ONE**コマンドと全く同じようにそのレコードをロードします。

SAVE RELATED ONE

SAVE RELATED ONE (フィールド)

引数	タイプ	説明
フィールド	フィールド	nフィールド

SAVE RELATED ONEコマンドは、＜フィールド＞に関連レコードを保存します。**CREATE RELATED ONE**コマンドで新しく作成したレコードや、**RELATE ONE**コマンドでロードし修正したレコードを保存するには、必ず本コマンドを実行する必要があります。

SAVE RELATED ONEコマンドは、サブテーブルに対しては何も行いません。それは、親レコードを保存すればサブレコードも自動的に保存するからです。

SAVE RELATED ONEコマンドは、ロックされたレコードは保存しません。このコマンドを使用する場合、最初にレコードがアンロックされているかどうかを確認する必要があります。レコードがロックされている場合には、このコマンドは無視され、レコードを保存せず、エラーも返しません。

修正されたデータの管理

レコードがロードされた時点で、4th Dimensionはロードされたレコードのコピーを作成します。レコードが修正された時点で、コピーも修正します。このコピーは、保存の指示があるまでディスクには書き込まれません。コピーが保存されるまでは、この節のコマンドを使用して“ 更新前データ ”(修正前の状態のデータ)をアクセスすることができます。

OLD RELATED ONE

OLD RELATED ONE (フィールド)

引数	タイプ	説明
フィールド	フィールド	nフィールド

OLD RELATED ONEコマンドは、**RELATE ONE**コマンドと同じ方法で処理を行います。ただし、**OLD RELATED ONE**コマンドは、フィールドの更新前の内容を使用してリレート処理を行います。

OLD RELATED ONEコマンドは、更新前のカレントレコードの内容に関連したレコードをロードし、その関連レコードにアクセスできるようにします。関連レコードを修正した場合には、**SAVE OLD RELATED ONE**コマンドを使用して保存する必要があります。新しく作成されたレコードは、更新前の関連するレコードを持たないという点に注意してください。

SAVE OLD RELATED ONE

SAVE OLD RELATED ONE (フィールド)

引数	タイプ	説明
フィールド	フィールド	nフィールド

SAVE OLD RELATED ONEコマンドは、フィールドの更新前の内容を使用してリレート処理を行う以外は、**SAVE RELATED ONE**コマンドと同じ方法で処理を行います。**SAVE OLD RELATED ONE**コマンドを使用する前に、**OLD RELATED ONE**コマンドでレコードをロードしておく必要があります。**OLD RELATED ONE**コマンドでロードしたレコードを修正した場合にも、**SAVE OLD RELATED ONE**コマンドを使用して保存する必要があります。

SAVE OLD RELATED ONEコマンドは、ロックされたレコードを保存しません。このコマンドを使用した場合は、最初にレコードがアンロックされているかどうかを確認する必要があります。レコードがロックされている場合には、このコマンドは無視され、レコードを保存せず、エラーも返しません。

OLD RELATED MANY

OLD RELATED MANY (フィールド)

引数	タイプ	説明
フィールド	フィールド	1フィールド

OLD RELATED MANYコマンドは、フィールドの更新前の内容を使用してリレート処理を行う以外は、**RELATE MANY**コマンドと同じ方法で処理を行います。

OLD RELATED MANYコマンドは、リレートテーブルのセレクションを変更し、カレントレコードとしてそのセレクションの最初のレコードを選択します。

新しく作成されたレコードは、更新前の関連するレコードを持たない点に注意してください。

RELATE ONE SELECTION

RELATE ONE SELECTION (nテーブル ; 1テーブル)

引数	タイプ		説明
nテーブル	テーブル	→	nテーブルの名前（リレート元）
1テーブル	テーブル	→	1テーブルの名前（リレート先）

RELATE ONE SELECTIONコマンドは、引数＜nテーブル＞のレコードセレクションを基にして、＜1テーブル＞用のレコードセレクションを新たに作成します。

このコマンドは、＜nテーブル＞から＜1テーブル＞へのリレートがある場合に限って使用できます。**RELATE ONE SELECTION**コマンドは、リレートの複数レベルを対象に動作できます。＜nテーブル＞と＜1テーブル＞の間には、複数のリレートテーブルがある場合があります。これらのリレートは、マニュアルリレートまたは自動リレートになるように設定することもできます。

警告：このコマンドは、トランザクションの内部で使用してはいけません。

▼ 次の例では、今日が請求書の支払期日であるすべての顧客を検索しています：

次に示されているのは、[送り状]テーブル内のレコードの選択がある場合に、[顧客]テーブル内で選択を作成する方法の1つです。

```
CREATE EMPTY SET([顧客];"支払期日")
QUERY([送り状];[送り状]支払日 = Current date)
While(Not(End selection([送り状])))
   RELATE ONE ([送り状]顧客ID)
   ADD TO SET([顧客];"支払期日")
   NEXT RECORD([送り状])
End while
```

▼ 次の手法では、**RELATE ONE SELECTION**コマンドを使用して同じ結果を出しています：

```
QUERY([送り状];[送り状]支払日 = Current date)
RELATE ONE SELECTION([送り状];[顧客])
```

参照：QUERY、 RELATE MANY SELECTION、RELATE ONE

RELATE MANY SELECTION

RELATE MANY SELECTION (フィールド)

引数	タイプ		説明
フィールド	フィールド	→	nテーブルのフィールド (リレート元の)

RELATE MANY SELECTIONコマンドは、1テーブルのレコードセレクションを基にして、nテーブルのレコードセレクションを作成します。

注：RELATE MANY SELECTIONコマンドは、1テーブルのカレントレコードを変更します。

警告：このコマンドは、トランザクションの内部で使用してはいけません。

▼ 次の例では、貸出金が1,000ドル以上の顧客を対象に作成されたすべての請求書を選択しています。[送り状]テーブルの[送り状]顧客 IDフィールドは、[顧客]テーブルの[顧客]ID番号フィールドにリレートしています：

```
` 顧客を選択する
QUERY ([顧客];[顧客]貸出金>=1000)
`任意の顧客にリレートしているすべての送り状を探す
RELATE MANY SELECTION ([送り状]顧客ID)
```

参照：QUERY、RELATE ONE、RELATE ONE SELECTION

リソースの概念

リソースは、Windowsの「.RSR」ファイル、Macintoshファイルのリソースフォークで定義されたフォーマットに保持されるある種のデータです。一般的に、リソースは文字列、ピクチャ、アイコン等のデータを含んでいます。実際に、独自のタイプのリソースを作成して使用することができます。また、そのリソースの中にどんなデータでも格納することができます。

データフォークとリソースフォーク

Macintosh上では、各ファイルはデータフォークとリソースフォークを持つことができます。Macintoshファイルのデータフォークは、WindowsやUNIX上のファイルと同等のものです。Macintoshのリソースフォークは、そのファイルのMacintoshベースのリソースが含まれています。これは、WindowsやUNIX上に同等のものはありません。

Windowsベースのリソースは、ファイルの他のデータと一緒に混ぜあわされ格納されます。例えば、Windowsのアプリケーションにおいて、「.EXE」ファイルはリソースデータとコードの両方を含むことができます。あなたの4Dアプリケーションをプラットフォームフォームに依存しない形式で管理するには、4th DimensionはMacintoshとWindowsの両方のプラットフォーム上でMacintoshベースのリソースを使って作業します。

4D Transporter

リソースフォークはWindows上には存在しないので、（4th DimensionのMacintosh版で提供される）4D Transporterユーティリティツールを使って、4DデータベースをWindowsからMacintosh（または、MacintoshからWindows）にトランスポートすることができます。

WindowsからMacintoshに4Dデータベースをトランスポートすると、データベースストラクチャファイルの「.4DB」と「.RSR」が1つのMacintoshファイルに“マージ”されます。「.4DB」ファイルが、Macintoshストラクチャファイルのデータフォークになります。一方、「.RSR」ファイルが、Macintoshストラクチャファイルのリソースフォークになります。

逆に、MacintoshからWindowsに4Dデータベースをトランスポートすると、Macintoshストラクチャファイルは2つのファイルに“ 分割 ”されます。Macintoshのデータフォークが「.4DB」ファイル、Macintoshのリソースフォークが「.RSR」ファイルになります。

データフォークおよびリソースフォークファイルを分割またはマージする機能こそが、4D Transporterツールの主要目的です。4D Transporterは、フォークおよびファイルの中に保持されている実際のデータを翻訳したり修正することはできません。プラットフォーム間の4Dデータベースのトランスポートに関する詳細は、『4D Transporter』マニュアルを参照してください。

リソースファイル

使用しているプラットフォームが何であれ、4Dデータベースのストラクチャファイルがリソースを持つ唯一のファイルタイプではありません。4Dアプリケーション自身がリソースを含んでいます。Macintosh上では、4Dリソースはそのアプリケーションのリソースフォークに保持されます。一方、Windows上では4Dリソースは「4D.RSR」ファイルに保持され、実行可能なコードを持つ4Dアプリケーションのリソース部分は「4D.EXE」ファイルに保持されます。

4Dプラグインもリソースを含むことができます。例えば、ACI社の4D Drawプラグインはリソースを含んでいます。Macintosh上では、リソースは「4D Draw6.0」のリソースフォークに保持されます。一方、Windows上では「4DDRAW.RSR」ファイルに保持されます。

また、4Dデータベースのデータファイルもリソースを含むことができます。例えば、(Customizer Plusユーティリティを使って）特定のストラクチャファイルでのみ使用するデータファイルをロックすると、この処理はストラクチャファイルとデータファイルの中に同一のWEDD（“ Wedding ”の“ WEDD ”）リソースを追加します。Macintosh上では、そのリソースはデータファイルのリソースフォークに追加されます。Windows上では、このリソースはデータファイルのリソースファイルである「.4DR」ファイルに保持されます。

注：Customizer Plusは、Windows版およびMacintosh版の4th Dimensionで提供されます。

Windows上では、データファイルの「.4DR」ファイルを除いて、通常、「.RSR」ファイル拡張子を持ったファイルとしてMacintoshベースのリソースを含んでいる標準の4Dファイルを検出します。**Create resource file**関数がデフォルトのファイル拡張子として「.RES」を使用することに注目してください。

ユーザ独自のリソースファイルの作成

4Dによって提供されるリソースファイルの他に、**Create resource file**関数と**Open resource file**関数を使って、ユーザ独自のリソースファイルを作成して使用することができます。この2つの関数は、オープンされるリソースファイルを一意に識別する「リソースファイル参照番号」を返します。このリソース参照番号は、**Open document**関数などのシステム文書コマンドによって返される一般ファイルの文書参照番号と同等のものです。すべての4Dリソースコマンドは、任意にリソースファイル参照番号を求めます。リソースファイルを使い終わったら、**CLOSE RESOURCE FILE**コマンドを使って、そのファイルを閉じることを忘れないでください。

リソースファイル連鎖

4Dのデータベースで作業する際、「現在オープンしているリソースファイルのすべて」または「特定のリソースファイル」のどちらか一方で作業することができます。

複数のリソースファイルを同時にオープンすることができます。これは常に4Dデータベースの内部からの場合です。次のファイルを開くことができます：

- Macintosh上のシステムリソースファイル
- Windows上の「ASIPORT.RSR」ファイル（Macintoshのシステムリソース部分を含む）
- 4Dアプリケーションリソースファイル
- データベースのストラクチャリソースファイル
- リソースファイルを任意にオープンできるデータベースのデータファイル
- **Open resource file**関数を使って、オープンできるユーザ独自のリソースファイル

上記のオープンできるリソースファイル群のリストは、「リソースファイル連鎖」と呼ばれます。次の2つの方法で所定のリソースを検索することができます：

- 4Dリソースコマンドにリソースファイル参照番号を渡すと、そのリソースはリソースファイル内でのみ検索されます。
- 4Dリソースコマンドにリソースファイル参照番号を渡さない場合、そのリソースは現在オープンされているすべてのリソースファイル内で検索されます。この場合、1番最後にオープンされたファイルから1番最初にオープンしたファイルに向かって検索を行います。そのため、リソースファイル連鎖はオープンされた逆の順番でブラウズされます。つまり、最後にオープンされたリソースファイルが最初に検査されます。

次の例題を見てください。

```
$vhResFile:=Create resource file("Just_a_file")
If (OK=1)
   ARRAY STRING(63 ; asSomeStrings ; 0)
   STRING LIST TO ARRAY(8 ; asSomeStrings ; $vhResFile)
   ALERT("配列のサイズは、"+String(Size of array(asSomeStrings))+"要素です。")
   STRING LIST TO ARRAY(8 ; asSomeStrings)
   ALERT("配列のサイズは、"+String(Size of array(asSomeStrings))+"要素です。")
   CLOSE RESOURCE FILE($vhResFile)
End if
```

このメソッドを実行すると、最初に“ 配列のサイズは、0要素です ” という警告を表示します。次に、“ 配列のサイズは、634要素です ” という警告を表示します。

1番目の呼び出し：

```
STRING LIST TO ARRAY(8 ; asSomeStrings ; $vhResFile)
```

Create resource file関数を使って、作成されオープンされたリソースファイルの「"STR#" ID=8」リソース“ のみ ” を探します。このファイルは新規に作成されて空なので、そのリソースは見つかりません。

2番目の呼び出し：

```
STRING LIST TO ARRAY(8 ; asSomeStrings )
```

“ 現在オープンしているすべて ” のリソースファイルの「"STR#" ID=8」リソースを探します。ファイルは、（**Create resource file**関数を使って）作成されオープンされるので、リソースは含んでいません。**STRING LIST TO ARRAY**コマンドは、データベースのストラクチャリソースファイルのリソースを探します。このリソースファイルはどちらもリソースを含んでいないので、**STRING LIST TO ARRAY**コマンドは4Dリソースファイルを調べて、このストラクチャリソースファイルの中にリソースを配置し、そのリソースを配列に割り当てます。

まとめ：リソースファイルを使って作業する際、もし特定のファイルにアクセスしたい場合は、4Dリソースコマンドにリソースファイル参照番号を渡すことを忘れないでください。そうしないと、4Dリソースコマンドは、ユーザがリソースの元であるファイルを気にかけていないと仮定します。

リソースタイプ

リソースファイルは、高度に構築されます。各リソースデータの他に、そのリソース内容を詳細に記述したヘッダとマップを含んでいます。

リソースは、タイプ別に分類されます。リソースタイプは、常に4バイトの文字列で表されます。リソースタイプは、厳格に区別されます。例えば、リソースタイプ“ Hi_! ”、“ hi_! ”、“ HI_! ” は、すべて異なります。

重要：小文字のリソースタイプは、オペレーションシステム（OS）で使用するために予約されています。そのため、ユーザ独自のリソースタイプを小文字で設計するのは避けてください。

下記は、共通で使用されるリソースタイプを示したものです：

■ **“ STR# ”** リソースタイプは、Pascal文字列のリストを含んでいるリソースです。このリソースは、「ストリングリストリソース」と呼ばれます。

■ **“ STR ”（4バイト目にスペースがある点に注意）** リソースタイプは、個別のPascal文字列を含んでいるリソースです。このリソースは、「ストリングリソース」と呼ばれます。

■ **“ TEXT ”** リソースタイプは、長さ制限のないテキストストリングを含んでいるリソースです。このリソースは、「テキストリソース」と呼ばれます。

■ **“ PICT ”** リソースタイプは、4Dを使って、MacintoshとWindowsの両方で使用および表示されるMacintoshベースのQuickDrawピクチャを含んでいるリソースです。このリソースは、「ピクチャリソース」と呼ばれます。

■ **“ cicn ”** リソースタイプは、4Dを使って、MacintoshとWindowsの両方で使用および表示されるMacintoshベースのカラーアイコンを含んでいるリソースです。このリソースは、「カラーアイコンリソース」と呼ばれます。例えば、“ cicn ” リソースは、**SET LIST ITEM PROPERTIES**コマンドを使って階層リストの項目に割り当てることができます。

標準のリソースタイプの他に、ユーザ独自のリソースタイプを作成することができます。例えば、“ MTYP ”（“ My Type ” の “ MTYP ”）リソースタイプを使って作業することができます。

オープンされているすべてのリソースファイルまたは特定のリソースファイル内に現在存在するリソースタイプのリストを取得するには、**RESOURCE TYPE LIST**コマンドを使用します。また、オープンされているすべてのリソースファイルまたは特定のリソースファイル内に現在存在するリソースの指定されたタイプのリストを取得するには、**RESOURCE LIST**コマンドを使用します。このコマンドは、所定の全リソースタイプのID番号と名前を返します。

警告：ほとんどのアプリケーションはリソースの内容を使って作業するためにそのリソースタイプを当てにしています。例えば、アプリケーションが“ STR# ”リソースにアクセスすると同時にそのリソース内のストリングリストを検索します。標準のリソースタイプ内に矛盾したデータを格納してはいけません。これを行うと、4Dアプリケーションまたは他のアプリケーションにおいてシステムエラーを引き起こす原因になります。

警告：リソースは高度に構築されたファイルです。そのため、リソースコマンド以外のコマンドを使ってリソースファイルにアクセスしてはいけません。例えば、SEND PACKET等のコマンドへの（文書参照番号のような4D時間式の）リソースファイル参照番号を受け渡すと、おそらくリソースファイルはダメージを受けます。

警告：リソースファイルは、個別に最大2700個のリソースを持つことができます。この制限を越えるような試みを行ってはいけません。これを行うと、リソースファイルがダメージを受けて、使用できなくなります。

リソース名とリソースID番号

リソースは、「リソース名」を持っています。リソース名は最大255バイトで指定することができ、発音区別記号は識別しますが、それ以外の厳格な区別は行いません。リソース名は任意のリソースを説明するのに有効ですが、実際はリソースのタイプとID番号を使って、任意のリソースにアクセスします。リソース名は、一意（ユニーク）ではありません。複数のリソースで同じ名前を持つことができます。

リソースは、「リソースID番号（短く言えば、リソースIDまたはID」を持っています。このIDは、任意のリソースタイプおよびリソース内において一意です。例えば、

■ 1つのリソースファイルでは、リソース“ ABCD ”ID=1とリソース“ EFGH ”ID=1を持つことができます。

■ 2つのリソースファイルでは、同じタイプとIDのリソースを持つことができます。

4Dコマンドを使って任意のリソースにアクセスする場合、そのリソースのタイプとIDを指定します。このリソースを検索するためのリソースファイルを指定しなかった場合、4Dコマンドは最初に調べたリソースファイル内で見つかったリソースの出現を返します。リソースファイルがオープンされた逆の順番で調べられることを覚えておいてください。

リソースIDの範囲は、-32768から32767までです。

重要：15000から32767の範囲は、ユーザ独自のリソース用に使用できます。しかし、負数のリソースIDを使用してはいけません。これは、オペレーションシステム（OS）で使用されるためにあらかじめ予約されている番号です。また、0から14999の範囲のリソースIDを使用してもいけません。この範囲は、4th Dimensionで使用されるためにあらかじめ予約されている番号です。

所定のリソースタイプのID番号と名前を取得するには、**RESOURCE LIST**コマンドを使用します。

個別のリソース名を取得するには、**Get resource name**関数を使用します。

個別のリソース名を変更するには、**SET RESOURCE NAME**コマンドを使用します。

各4Dコマンドは任意にリソースファイル参照番号をアクセスできるので、異なる2つのリソースファイル内で同じタイプとIDを持つリソースを簡単に取り扱うすることができます。次の例は、あるリソースから別のリソースにすべての“ PICT ”リソースをコピーします：

```
` 既存のリソースファイルを開く
$vhResFileA:=Open resource file("")
If (OK=1)
    ` 新規リソースファイルを作成する
    $vhResFileB:=Create resource file("")
    If (OK=1)
        ` "PICT"タイプのすべてのリソースのIDと名前のリストを取得し、
        ` リソースファイルAの中に配置する
        RESOURCE LIST("PICT" ; $aiResID ; $asResName ; $vhResFileA)
        `各リソースファイルに対してループ
        For($vlElem ; 1 ; Size of array($aiResID))
            $viResID:=$aiResID{$vlElem}
            ` Aファイルからリソースをロードする
            GET RESOURCE ("PICT" ; $viResID ; vxResData ; $vhResFileA)
            ` リソースがロードされた場合、
            If (OK=1)
                ` Bファイルの中にリソースを追加し書き込む
                SET RESOURCE ("PICT" ; $viResID ; vxResData ; $vhResFileB)
                ` リソースが追加および書き込まれた場合
                If (OK=1)
                    `リソースの名前もコピーする
                    SET RESOURCE NAME("PICT ; $viResID ; $asResName{$vlElem}
                    ; $vhResFileB)
                    `リソースプロパティについては、次節の「リソースプロパティ」を
                    `参照してください。
                    $vlResAttr:=Get resource properties("PICT" ; $viResID ; $vhResFileA)
                    SET RESOURCE PROPERTIES("PICT" ; $viResID ; $vlResAttr
                    ; $vhResFileB)
                Else
                    ALERT("リソースPICT ID="+String($viResID)+"は追加されません。")
                End if
```

```
            Else
                ALERT("リソースPICT ID="+String($viResID)+"はロードされません。")
            End if
        End if
        CLOSE RESOURCE FILE($vhResFileB)
    End if
    CLOSE RESOURCE FILE($vhResFileA)
End if
```

リソースプロパティ

リソースは、タイプ、名前、ID番号の他に（属性と呼ばれる）追加プロパティを持っています。例えば、任意のリソースではパージ（消去）できるものもあれば、できないものもあります。この属性は、空きメモリが他のオブジェクトに割り当てるために必要とされている場合に、ロードされるリソースがメモリからパージされるかどうかオペレーションシステムに知らせます。上記の例で示しているように、任意のリソースを作成またはコピーする際、そのリソースのコピーだけでなく、そのリソースの名前およびプロパティもコピーするところが重要です。リソースプロパティに関する詳細は、**Get resource properties**関数および**SET RESOURCE PROPERIES**コマンドの説明を参照してください。

リソース内容の取り扱い

メモリの中に任意のリソースタイプをロードするには、BLOBの中にリソース内容を返す**GET RESOURCE**コマンドを呼び出します。

ディスク上に任意のリソースを追加したり書き込むには、ユーザが渡したBLOBの内容にリソース内容を設定できる**SET RESOURCE**コマンドを呼び出します。

既存のリソースを削除するには、**DELETE RESOURCE**コマンドを使用します。

一般的なリソースタイプを簡単に取り扱うために、4Dは下記のような内蔵コマンドを用意しています。

■ **STRING LIST TO ARRAY**コマンドは、任意の文字列配列またはテキスト配列にストリングリストリソースに含まれる文字列を割り当てます。

■ **ARRAY TO STRING LIST**コマンドは、ストリングリストリソースを文字列配列またはテキスト配列の要素を使って上書きします。

■ **Get string resource**関数は、ストリングリソースから特定の文字列を返します。

■ **Get index string**関数は、ストリングリソースから文字列を返します。

■ **SET STRING RESOURCE**コマンドは、ストリングリソースを作成または上書きします。

■ **Get text resource**関数は、テキストリソースのテキストを返します。
■ **SET TEXT RESOURCE**コマンドは、テキストリソースを作成または上書きします。
■ **GET PICTURE RESOURCE**コマンドは、ピクチャリソースのピクチャを返します。
■ **SET PICTURE RESOURCE**コマンドは、ピクチャリソースを作成または上書きします。
■ **GET ICON RESOURCE**コマンドは、ピクチャのカラーアイコンリソースを返します。

上記のコマンドは一般的なリソースタイプを簡単に管理するために用意されていますが、これらのコマンドは、BLOBを使った**GET RESOURCE**コマンドや**SET RESOURCE**コマンドの使用を防止しない点に注意してください。例えば、次のコードは、

```
ALERT(Get text resource(20000))
```

下記のコードと同等の処理を短くしたものです：

```
GET RESOURCE("TEXT" ; 20000 ; vxData)
If (OK=1)
   $vlOffset:=0
   ALERT(BLOB to text(vxData ; Text without length ; $vlOffset ; BLOB Size(vxData)))
End if
```

4Dコマンドとリソース

この章で説明するリソースコマンドの他に、リソースやリソースファイルを使って作業する4Dコマンドもあります。

■ Macintosh上では、**DOCUMENT TO BLOB**コマンドと**BLOB TO DOCUMENT**コマンドは任意のMacintoshファイルのリソースフォーク全体をロードして書き込むことができます。
■ **SET LIST ITEM PROPERTIES**コマンドと**SET LIST PROPERTIES**コマンドを使って、リストの項目にピクチャやカラーアイコンリソースを関連付けたり、リストのノードとしてカラーアイコンリソースを使用することができます。
■ **PLAY**コマンドは、MacintoshとWindowsの両方で"snd "リソースをプレイします。
■ **SET CURSOR**コマンドは、"CURS"リソースを使って、マウスの外観を変更します。

Open resource file

Open resource file (リソースファイル名 {; ファイルタイプ }) → DocRef

引数	タイプ		説明
リソースファイル名	文字列	→	リソースファイルのショート名またはロング名 空の文字列の場合、標準の「ファイルを開く」ダイアログボックスを表示
ファイルタイプ	文字列	→	MacOSファイルタイプ (4バイト文字)、またはWindowsファイル拡張子 (1~3バイト文字)、 または、省略の場合、すべてのファイル
関数の返す値	DocRef	←	リソースファイル参照番号

Open resource file関数は、引数＜リソースファイル名＞に渡した名前またはパス名を持つリソースファイルをオープンします。

任意のパス名を渡す場合、そのファイルはデータベースのストラクチャファイルと同じフォルダ内に配置されていなければなりません。他のフォルダ内に配置されているリソースを開くには、任意のパス名を渡します。

引数＜リソースファイル名＞に空の文字列を渡すと、**Open resource file**関数は「ファイルを開く」ダイアログボックスが表示されます。このダイアログボックスでリソースを選択し、それを開くことができます。ダイアログボックスを取り消すと、リソースファイルはオープンされません。また、**Open resource file**関数はヌルのDocRefを返し、システム変数OKに0を設定します。

リソースファイルが現在オープンされている場合は、**Open resource file**関数はそのリソースファイル参照番号を返し、システム変数OKに1を設定します。リソースファイルが存在しない場合やオープンしようとしているファイルにリソースファイルがない場合、エラーが生成されます。

Macintosh上で、「ファイルを開く」ダイアログボックスを使用する場合、デフォルトとしてすべてのファイルが表示されます。ファイルの特定タイプを表示するには、オプション引数＜ファイルタイプ＞にそのファイルタイプを指定します。

Windows上で、「ファイルを開く」ダイアログボックスを使用する場合、デフォルトとしてすべてのファイルが表示されます。ファイルの特定タイプを表示するには、1から3バイトのWindowsファイル拡張子または**MAP FILE TYPES**コマンドを使ってマップされた任意のMacintoshファイルタイプをオプション引数＜ファイルタイプ＞に渡します。

リソースファイルを使い終わったら、**CLOSE RESOURCE FILE**コマンドの呼び出しを忘れないでください。ただし、アプリケーションを終了する（または、他のデータベースを開く）場合は、4Dが **Open resource file**関数や**Create resource file**関数を使ってオープンされたすべてのリソースファイルを自動的に閉じます。

排他的なリード-ライトアクセスで任意の文書（Macintosh上のデータフォーク）をオープンする**Open document**関数と違って、**Open resource file**関数は 4Dセッションの内部からすでにオープンされているリソースファイルをユーザがオープンするのを防止しません。例えば、**Open document**関数を使って同じ文書を2度オープンしようとすると、2度目のオープン時にI / Oエラーが返されます。これに対して、4Dセッションの内部からすでにオープンされているリソースファイルをオープンしようとすると、**Open resource file**関数はすでにオープンしているファイルのリソースファイル参照番号を返します。たとえ、何度も任意のリソースファイルをオープンしても、そのリソースファイルを閉じるにはたった1回**CLOSE RESOURCE FILE**コマンドを呼び出すだけで済みます。これが、4Dセッションの内部からリソースファイルがオープンされる場合に許可される点に注目してください。他のアプリケーションですでにオープンされているリソースファイルをオープンしようとすると、I/ Oエラーが発生します。

この複数オープン機能により、通常の4D処理をむやみに変更することなく、4Dアプリケーションとデータベースリソースファイルの参照番号を容易に取得することができます。

警告：

■ 4Dアプリケーションのリソースファイルにアクセスする場合は十分に注意してご使用ください。4Dアプリケーションのリソースを変更してはいけません。プログラムに予想できないダメージを与え、システムエラーを引き起こす原因になります。また、あなたのデータベースがさまざまな4D環境（4D、4D Runtime、4D Engine、4D Serverと4D Client）内で使用されることを覚えておいてください。

■ データベースリソースファイルにアクセスして、プログラムでそのファイルのリソースを追加、削除、修正する場合、動作している4D環境を必ず調べてください。4D Serverでは、これはおそらく深刻な問題を引き起こします。例えば、（データベースメソッドやストアドプロシージャ経由で）サーバマシン上で任意のリソースファイルを修正する場合、ワークステーションにリソースを透過的に配付する内蔵の4D Server管理サービスに確実に影響を及ぼします。4D Clientでは、ストラクチャファイルがサーバマシン上に配置されているため、直接ストラクチャファイルにアクセスできない点に注意してください。

■ この理由により、リソースを使用する場合、そのリソースをあなた自身のファイルに格納します。

■ あなた自身のリソースを使って作業している場合、負数のリソースIDを使用してはいけません。これは、オペレーションシステムで使用されるためにあらかじめ予約されている番号です。また、0から14999の範囲のリソースIDを使用してもいけません。この範囲は、4th Dimensionで使用されるためにあらかじめ予約されている番号です。あなた自身

のリソースには、15000から32767の範囲の番号を使用します。リソースファイル連鎖内で検索される最初のファイルが1つオープンされるのを覚えておいてください。。システムまたは4Dリソースの範囲内のIDを持つリソースファイルに任意のリソースを格納する場合、このリソースは**GET RESOURCE**コマンドまたは4Dアプリケーションの内部ルーチンで見つけることができます。これがあなたが目的とする結果かもしれませんが、もしそうでない場合は、それらはシステムエラーを引き起こすかもしれないので、これらの範囲を使用してはいけません。

■ リソースファイルは高度に構築されたファイルで、しかも1ファイル当たり最高2700以上のリソースにアクセスすることはできません。とても大きい数のリソースを含んでいるリソースファイルで作業する場合、そのリソースファイルに新しいリソースを追加する前にそのリソースの数を調べるのが得策です。これについては、**RESOURCE TYPE LIST**コマンドの例題の**Count resource**関数を参考にしてください。

任意のリソースファイルをオープンすると、**RESOURCE TYPE LIST**コマンドと**RESOURCE LIST**コマンドを使って、そのファイルの内容を解析することができます。

例：

1. 次の例は、Windows上でデータベースフォルダに配置された“ MyPrefs.res ”リソースファイルをオープンします：

```
$vhResFile:=Open resource file("MyPrefs" ; "res ")
```

Macintosh上では、この例は“ MyPrefs ”ファイルをオープンします。

2. 次の例は、Windows上でデータベースフォルダに配置された“ MyPrefs.rsr ”リソースファイルをオープンします：

```
$vhResFile:=Open resource file("MyPrefs" ; "rsr ")
```

Macintosh上では、この例は“ MyPrefs ”ファイルをオープンします。

3. 次の例は、すべてのファイルタイプを表示する「ファイルを開く」ダイアログボックスを表示します：

```
$vhResFile:=Open resource file(" ")
```

4. 次の例は、デフォルトファイルタイプを使って、**Create resource file**関数で作成されたファイルを表示する「ファイルを開く」ダイアログボックスを表示します：

```
$vhResFile:=Open resource file("" ; "res ")
If (OK=1)
   ALERT(Document+"をオープンしました。")
   CLOSE RESOURCE FILE($vhResFile)
End if
```

5. 次の例は、ローカル変数 “ $vhStructureResFile ” にデータベースストラクチャリソースファイルの参照番号を返します：

```
If (On Windows)
   $vhStructureResFile:=Open resource file(Replace string(Structure file ; ".4DB" ; ".RSR"))
Else
   $vhStructureResFile:=Open resource file(Structure file)
End if
```

6. 次の例は、ローカル変数 “ $vhApplResFile ” に4Dアプリケーションリソースファイルの参照番号を返します：

```
If (On Windows)
   $vhApplResFile:=Open resource file(Replace string(Application file;".EXE";".RSR"))
Else
   $vhApplResFile:=Open resource file(Application file)
End if
```

参照：CLOSE RESOURCE FILE、Create resource file

システム変数とシステムセット：

リソースファイルが正常にオープンされた場合は、システム変数OKに1を設定します。リソースファイルがオープンできなかったり、ユーザが「ファイルを開く」ダイアログボックスの「キャンセル」ボタンをクリックした場合は、システム変数OKに0が設定されます。

リソースファイルが「ファイルを開く」ダイアログボックスを使って正常にオープンされた場合は、システム変数Documentにそのリソースファイルのパス名を設定します。

エラー処理：

リソースファイルが任意のリソースをオープンできなかったり、I/O（入出力）に問題が生じた場合、エラーが生成されます。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

Create resource file

Create resource file (リソースファイル名 {; ファイルタイプ }) → DocRef

引数	タイプ		説明
リソースファイル名	文字列	→	リソースファイルのショート名またはロング名 空の文字列の場合、標準の「ファイルを保存」ダイアログボックスを表示
ファイルタイプ	文字列	→	MacOSファイルタイプ (4バイト文字)、またはWindowsファイル拡張子 (1~3バイト文字)、 または、省略の場合、リソース文書("res ' / .RES)
関数の返す値	DocRef	←	リソースファイル参照番号

Create resource file関数は、引数＜リソースファイル名＞に渡した名前またはパス名を持つ新規リソースファイルを作成してオープンします。

任意のパス名を渡す場合、そのファイルはデータベースのストラクチャファイルと同じフォルダ内に配置されていなければなりません。他のフォルダ内に配置されているリソースを開くには、任意のパス名を渡します。

すでに存在するファイルが現在オープンされていない場合、**Create resource file**関数は空の新規リソースファイルでそれを上書きします。ファイルが現在オープンされている場合は、I/Oエラーが返されます。

引数＜リソースファイル名＞に空の文字列を渡すと、**Create resource file**関数は「ファイルを保存」ダイアログボックスが表示されます。このダイアログボックスで作成されるリソースファイルの格納場所と名前を選択することができます。ダイアログボックスを取り消すと、リソースファイルを保存しません。また、**Create resource file**関数はヌルのDocRefを返し、システム変数OKに0を設定します。

リソースファイルが現在オープンされている場合は、**Open resource file**関数はそのリソースファイル参照番号を返し、システム変数OKに1を設定します。リソースファイルが存在しない場合やオープンしようとしているファイルにリソースファイルがない場合、エラーが生成されます。

Macintosh上では、**Create resource file**関数で作成されるリソースファイルのデフォルトファイルタイプは“ res ”です。Windows上では、デフォルトのファイル拡張子は“ .RES ”です。

他のファイルタイプを作成するには、

■ Macintosh上では、オプション引数＜ファイルタイプ＞にそのファイルタイプを渡します。

■ Windows上では、1から3バイトのWindowsファイル拡張子または**MAP FILE TYPES**コマンドを使ってマップされた任意のMacintoshファイルタイプをオプション引数＜ファイルタイプ＞に渡します。

リソースファイルを使い終わったら、**CLOSE RESOURCE FILE**コマンドの呼び出しを忘れないでください。ただし、アプリケーションを終了する（または、他のデータベースを開く）場合、4Dが **Open resource file**関数や**Create resource file**関数を使って、オープンされたすべてのリソースファイルを自動的に閉じます。

例：

1. 次の例は、Windows上でデータベースフォルダに配置された“ MyPrefs.res ”リソースファイルを作成してオープンします：

```
$vhResFile:=Create resource file("MyPrefs")
```

Macintosh上では、この例は“ MyPrefs ”ファイルを作成してオープンします。

2. 次の例は、Windows上でデータベースフォルダに配置された“ MyPrefs.rsr ”リソースファイルを作成してオープンします：

```
$vhResFile:=Create resource file("MyPrefs" ; "rsr ")
```

Macintosh上では、この例は“ MyPrefs ”ファイルを作成してオープンします。

3. 次の例は、「ファイルを保存」ダイアログボックスを表示します：

```
$vhResFile:=Create resource file("" ; "res ")
If (OK=1)
   ALERT(Document+"を保存しました。")
   CLOSE RESOURCE FILE($vhResFile)
End if
```

参照：CLOSE RESOURCE FILE、Open resource file

システム変数とシステムセット：

リソースファイルが正常に作成されオープンされた場合は、システム変数OKに1を設定します。リソースファイルが作成できなかったり、ユーザが「ファイルを保存」ダイアログボックスの「キャンセル」ボタンをクリックした場合は、システム変数OKに0が設定されます。

リソースファイルが「ファイルを保存」ダイアログボックスを使って正常に作成されオープンされた場合は、システム変数Documentにそのリソースファイルのパス名を設定します。

エラー処理：

リソースファイルが任意のリソースを作成またはオープンできなかったり、I/O（入出力）に問題が生じた場合、エラーが生成されます。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

CLOSE RESOURCE FILE

CLOSE RESOURCE FILE (リソースファイル)

引数	タイプ		説明
リソースファイル	DocRef	→	リソースファイル参照番号

CLOSE RESOURCE FILEコマンドは、引数＜リソースファイル＞に渡された参照番号を持つリソースファイルを閉じます。

たとえ、何度も同じリソースファイルをオープンしても、そのリソースファイルを閉じるには**CLOSE RESOURCE FILE**コマンドを1回呼び出すだけで済みます。

4Dアプリケーションやデータベースリソースファイルに**CLOSE RESOURCE FILE**コマンドを適用すると、このコマンドはそれを見つけて何も行いません。

無効なリソースファイル参照番号を渡すと、このコマンドは何も行いません。

Open resource file関数や**Create resource file**関数を使ってオープンされたリソースファイルを終了するために、最終的には**CLOSE RESOURCE FILE**コマンドを呼び出すことを忘れないでください。また、アプリケーションを終了する（または、他のデータベースを開く）場合、4Dがオープンされたリソースファイルを自動的に閉じる点にも注目してください。

▼ 次の例は、任意のリソースファイルを作成し、そのファイルにストリングリソースを追加して閉じます：

```
$vhDocRef:=Create resource file("Just a file")
If (OK=1)
SET STRING RESOURCE(20000 ; "Just a string" ; $vhDocRef)
   CLOSE RESOURCE FILE($vhDocRef)
End if
```

参照：Create resource file、Open resource file

システム変数とシステムセット：

影響を及ぼすものはありません。

RESOURCE TYPE LIST

RESOURCE TYPE LIST (リソースタイプ {; リソースファイル})

引数	タイプ		説明
リソースタイプ	文字列配列	←	リソース可能なリソースタイプのリスト
リソースファイル	DocRef	→	リソースファイル参照番号、または 省略した場合、オープンしているすべての リソースファイル

RESORCE TYPE LISTコマンドは、引数＜リソースタイプ＞の文字列配列に現在オープンしているリソースファイルの中に存在するリソースのリソースタイプを割り当てます。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのファイルのリソースのみが一覧表示されます。＜リソースファイル＞を省略した場合は、現在オープンしているリソースファイルのすべてのリソースが一覧表示されます。

RESORCE TYPE LISTコマンドを呼び出す前に引数＜リソースタイプ＞配列に任意の文字列配列またはテキスト配列を前もって定義することができます。配列の事前定義を行わない場合、このコマンドはテキスト配列の＜リソースタイプ＞を作成します。

このコマンドを呼び出すと、**Size of array**関数で見つけた引数＜リソースタイプ＞のリソースタイプの数を調べることができます。

例：

1. 次の例は、配列 “ atResType ” に現在オープンしているすべてのリソースファイル存在するリソースのリソースタイプを割り当てます：

```
RESOURCE TYPE LIST(atResType)
```

2. 次の例は、使用しているMacintoshの4Dストラクチャファイルが Windows上でデータベースを使用するためにアップデートする必要のある旧バージョンの4Dプラグインを含んでいるかどうかをユーザに知らせます：

```
$vhResFile:=Open resource file(Structure file)
RESOURCE TYPE LIST(atResType ; $vhResFile)
If (Find in array(atResType ; "4DEX")>0)
   ALERT("このデータベースは旧モデルのMacOSに対応した4Dプラグインが含まれて
         います。"+(Char(13)*2)+"Windows上でこのデータベースを使用するには、
         このプラグインをアップデートする必要があります。")
End if
```

注；ストラクチャファイルが旧バージョンの4Dプラグインを格納できる唯一のファイルではありません。データベースは “ Proc.EXT ” ファイルも含むことができます。

3. 次のプロジェクトメソッドは、任意のリソースファイルの中に存在するリソースの数を返します：

```
` 「Count resources」プロジェクトメソッド
` Count resources (時間) → 倍長整数
` Count resources ( DocRef ) → リソースの数
C_LONGINT($0)
C_TIME($1)
$0:=0
RESOURCE TYPE LIST($atResType ; $1)
For ($vlElem ; 1 ; Size of array($atResType))
   RESOURCE LIST($atResType{$vlElem} ; $alResID ; $atResName ; $1)
   $0:=$0+Size of array($alResID)
End for
```

このメソッドをデータベースに組み込むと、次のように記述できます：

```
$vhResFile:=Open resource file("")
If (OK=1)
   ALERT(Document+"ファイルには、"+String(Count resources ($vhResFile))
            +" 個のリソースが存在します。")
CLOSE RESOURCE FILE($vhResFile)
End if
```

参照：RESOURCE LIST

システム変数とシステムセット

影響を及ぼすものはありません。

RESOURCE LIST

RESOURCE LIST (リソースタイプ ; リソースID ; リソース名 {; リソースファイル})

引数	タイプ		説明
リソースタイプ	文字列	→	4バイトリソースタイプ
リソースID	倍長整数配列	←	リソースID番号
リソース名	文字列配列	←	リソース名
リソースファイル	DocRef	→	リソースファイル参照番号、または 省略した場合、オープンしているすべての リソースファイル

RESORCE LISTコマンドは、引数＜リソースID＞と＜リソース名＞の配列に引数＜リソースタイプ＞に渡したタイプのリソースID番号とリソース名を割り当てます。

重要；＜リソースタイプ＞には、4バイトの文字列を受け渡す必要があります。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのファイルのリソースのみが一覧表示されます。＜リソースファイル＞を省略した場合は、現在オープンしているリソースファイルのすべてのリソースが一覧表示されます。

RESORCE LISTコマンドを呼び出す前に配列を事前定義する場合は、引数＜リソースID＞配列に任意の倍長整数配列、引数＜リソースタイプ＞配列に任意の文字列配列またはテキスト配列を前もって定義する必要があります。配列の事前定義を行わない場合、このコマンドは倍長整数配列の＜リソースID＞、テキスト配列の＜リソースタイプ＞を作成します。

このコマンドを呼び出すと、**Size of array**関数で見つけた引数＜リソースID＞と＜リソース名＞のリソースの数を調べることができます。

例：

1. 次の例は、配列 “ $alResID ” と “ $atResName ” にデータベースのストラクチャファイル内に存在するストリングリストリソースのIDと名前を割り当てます：

```
If (On Windows)
   $vhStructureResFile:=Open resource file(Replace string(Structurefile ; ".4DB" ; ".RSR"))
Else
   $vhStructureResFile:=Open resource file(Structure file)
End if
If (OK=1)
   RESOURCE LIST("STR#" ; $alResID ; $atResName ; $vhStructureResFile)
End if
```

2. 次の例は、データベースのピクチャライブラリの中に現在オープンされているリソースファイル内にあるピクチャリソースをコピーします：

```
RESOURCE LIST("PICT" ; $alResID ; $atResName)
Open window(50 ; 50 ; 550 ; 120 ; 5 ; "PICTリソースコピー中...")
For ($vlElem ; 1 ; Size of array($alResID))
   GET PICTURE RESOURCE($alResID{$vlElem} ; $vgPicture)
   If (OK=1)
      $vsName:=$atResName{$vlElem}
      If ($vsName="")
         $vsName:="PICTリソースID="+String($alResID{$vlElem})
      End if
      ERASE WINDOW
      GOTO XY(2 ; 1)
      MESSAGE("DBピクチャライブラリに"+$vsName+"を追加します。")
      SET PICTURE TO LIBRARY($vgPicture ; $alResID{$vlElem} ; $vsName)
   End if
End for
CLOSE WINDOW
```

参照：RESOURCE TYPE LIST

システム変数とシステムセット

影響を及ぼすものはありません。

STRING LIST TO ARRAY

STRING LIST TO ARRAY (リソースID ; ストリング {; リソースファイル})

引数	タイプ		説明
リソースID	数値	→	リソースID番号
ストリング	文字列配列	→	ストリングを取り出すための文字列配列またはテキスト配列
		←	STR#リソースのストリング
リソースファイル	DocRef	→	リソースファイル参照番号、または省略した場合、オープンしているすべてのリソースファイル

STRING LIST TO ARRAYコマンドは、引数＜ストリング＞配列に引数＜リソースID＞に渡されるIDを持つストリングリソースリスト（"STR#"）リソース内に格納されているストリングを割り当てます。

リソースが見つからない場合は、＜ストリング＞配列はそのまま変更されず、システム変数OKに0が設定されます。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。＜リソースファイル＞を省略した場合は、リソースファイル連鎖内で最初に見つかったリソースの出現が返されます。

STRING LIST TO ARRAYコマンドを呼び出す前に、引数＜ストリング＞配列に任意の文字列配列またはテキスト配列を前もって定義することができます。配列の事前定義を行わない場合、このコマンドはテキスト配列の＜ストリング＞を作成します。

注：ストリングリストリソースの各ストリングは、最大255バイトの文字を使用することができます。

Tips：ストリングリストリソースは、合計で32Kまで使用できます。1リソース当たり最大100個程度のストリングにします。

▼ **ARRAY TO STRING LIST**コマンドの例を参照してください。

参照：ARRAY TO STRING LIST、Get indexed string、Get string resource、Get text resource

システム変数とシステムセット

リソースが見つかった場合は、OKに1が設定されます。それ以外は0が設定されます。

ARRAY TO STRING LIST

ARRAY TO STRING LIST (ストリング ; リソースID {; リソースファイル})

引数	タイプ		説明
ストリング	文字列配列	→	文字列配列またはテキスト配列 (STR#リソースの新しい項目)
リソースID	数値	→	リソースID番号
リソースファイル	DocRef	→	リソースファイル参照番号、または 省略した場合、現在のリソースファイル

ARRAY TO STRING LISTコマンドは、引数＜リソースID＞に渡されるIDを持つストリングリソースリスト（"STR#"）リソースを作成または上書きします。

リソースが追加されなかった場合は、システム変数OKに0が設定されます。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイルに追加されます。＜リソースファイル＞を省略した場合は、リソースはリソースファイル連鎖（最後にオープンしたリソースファイル）の上部にある現在のファイルに追加されます。

注：ストリングリストリソースの各ストリングは、最大255バイトの文字を使用することができます。

Tips：ストリングリストリソースは、合計で32Kまで使用できます。1リソース当たり最大100個程度のストリングにします。

例：

あなたのデータベースでは、所定のフォントセットを使用してるとします。

その場合、「On Exit」データベースメソッドにおいて、次のように記述します：

```
`「On Exit」データベースメソッド
If (<>bFontsAreOK)
   FONT LIST($atFont)
   $vhResFile:=Open resource file("FontSet")
   If (OK=1)
      ARRAY TO STRING LIST($atFont ; 15000 ; $vhResFile)
      CLOSE RESOURCE FILE($vhResFile)
   End if
End if
```

「On Startup」データベースメソッドにおいて、次のように記述します：

```
`「On Startup」データベースメソッド
<>bFontsAreOK:=False
FONT LIST($atNewFont)
If (Test path name("FontSet")#Is a document)
   $vhResFile:=Create resource file("FontSet")
Else
   $vhResFile:=Create resource file("FontSet")
End if
If (OK=1)
   STRING LIST TO ARRAY(15000 ; $atOldFont ; $vhResFile)
   If (OK=1)
      <>bFontsAreOK:=True
      For($vlElem ; 1 ; Size of array($atNewFont))
         If ($atNewFont{$vlElem}#$atOldFont{$vlElem})
            $vlElem:=MAXLONG
            <>bFontsAreOK:=False
         End if
      End for
   Else
      <>bFontsAreOK:=True
   End if
   CLOSE RESOURCE FILE($vhResFile)
End if
If(Not(炯bFontsAreOK))
   CONFIRM("同じフォントセットを使用していません。OK?")
   If(OK=1)
      <>bFontsAreOK:=True
   Else
      QUIT 4D
   End if
End if
```

参照：SET STRING RESOURCE、SET TEXT RESOURCE、STRING LIST TO ARRAY

システム変数とシステムセット

リソースが上書きされた場合は、OKに1が設定されます。それ以外は0が設定されます。

Get indexed string

Get indexed string (リソースID ; ストリングID {; リソースファイル}) → 文字列

引数	タイプ		説明
リソースID	数値	→	リソースID番号
ストリングID	数値	→	ストリング番号
リソースファイル	DocRef	→	リソースファイル参照番号、または 省略した場合、オープンしているすべての リソースファイル
関数の返す値	文字列	←	インデックス付きのストリングの値

Get indexed string関数は、引数＜リソースID＞に渡されるIDを持つストリングリスト("STR#")リソースに格納されている1つのストリングを返します。

引数＜ストリングID＞にストリングの番号を返します。ストリングリストリソースのストリングは1からNに番号が振られます。ストリングリストリソースのすべてのストリング（およびそのストリング番号）を取得するには、**STRING LIST TO ARRAY**コマンドを使用します。

リソースまたはそのリソース内のストリングが見つからなかった場合、空の文字列が返され、システム変数OKに0が設定されます。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。＜リソースファイル＞を省略した場合は、リソースファイル連鎖内で最初に見つかったリソースの出現が返されます。

注：ストリングリストリソースの各ストリングは、最大255バイトの文字を使用することができます。

▼ **Month of**関数の例を参照してください。

参照；Get string resource、Get text resource、STRING LIST TO ARRAY

システム変数とシステムセット

リソースが見つかった場合は、OKに1が設定されます。それ以外は0が設定されます。

Get string resource

Get string resource (リソースID {; リソースファイル}) → 文字列

引数	タイプ		説明
リソースID	数値	→	リソースID番号
リソースファイル	DocRef	→	リソースファイル参照番号、または 省略した場合、オープンしているすべての リソースファイル
関数の返す値	文字列	←	STRリソースの内容

Get string resource関数は、引数＜リソースID＞に渡されるIDを持つストリングリスト("STR#")リソースに格納されているストリングを返します。

リソースが見つからなかった場合、空の文字列が返され、システム変数OKに0が設定されます。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。＜リソースファイル＞を省略した場合は、リソースファイル連鎖内で最初に見つかったリソースの出現が返されます。

注：ストリングリソースは、最大255バイトの文字を使用することができます。

▼ 次の例は、ストリングリソースID=20911の内容を表示します。このIDは、現在オープンされているリソースファイルの少なくとも1つに配置されている必要があります：

```
ALERT (Get string resource(20911))
```

参照：Get indexed string、Get text resource、SET STRING RESOURCE、STRING LIST TO ARRAY

システム変数とシステムセット

リソースが見つかった場合は、OKに1が設定されます。それ以外は0が設定されます。

SET STRING RESOURCE

SET STRING RESOURCE (リソースID ; リソースデータ {; リソースファイル})

引数	タイプ		説明
リソースID	数値	→	リソースID番号
リソースデータ	文字列	→	STR リソースの新しい内容
リソースファイル	DocRef	→	リソースファイル参照番号、または 省略した場合現在のリソースファイル

SET STRING RESOURCE関コマンドは、引数＜リソースデータ＞に渡されるストリングを使って引数＜リソースID＞に渡されるIDを持つストリング（"STR "）リソースを作成または上書きします。

リソースが見つからなかった場合、空の文字列が返され、システム変数OKに0が設定されます。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイルに追加されます。＜リソースファイル＞を省略した場合は、リソースはリソースファイル連鎖（最後にオープンしたリソースファイル）の上部にある現在のファイルに追加されます。

注：ストリングリソースは、最大255バイトの文字を使用することができます。

参照：SET TEXT RESOURCE

システム変数とシステムセット

リソースが上書きされた場合は、OKに1が設定されます。それ以外は0が設定されます。

Get text resource

Get text resource (リソースID {; リソースファイル}) → テキスト

引数	タイプ		説明
リソースID	数値	→	リソースID番号
リソースファイル	DocRef	→	リソースファイル参照番号、または 省略した場合、オープンしているすべての リソースファイル
関数の返す値	テキスト	←	TEXTリソースの内容

Get text resource関数は、引数＜リソースID＞に渡されるIDを持つテキスト（"TEXT"）リソースに格納されているテキストを返します。

リソースが見つからなかった場合、空の文字列が返され、システム変数OKに0が設定されます。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。＜リソースファイル＞を省略した場合は、リソースファイル連鎖内で最初に見つかったリソースの出現が返されます。

注：テキストリソースは、最大32000バイトの文字を使用することができます。

▼ 次の例は、テキストリソースID=20800の内容を表示します。このIDは、現在オープンされているリソースファイルの少なくとも1つに配置されている必要があります：

```
ALERT (Get text resource(20800))
```

参照：Get indexed string、Get string resource、SET STRING RESOURCE、STRING LIST TO ARRAY

システム変数とシステムセット

リソースが見つかった場合は、OKに1が設定されます。それ以外は0が設定されます。

SET TEXT RESOURCE

SET TEXT RESOURCE (リソースID ; リソースデータ {; リソースファイル})

引数	タイプ		説明
リソースID	数値	→	リソースID番号
リソースデータ	文字列	→	TEXTリソースの新しい内容
リソースファイル	DocRef	→	リソースファイル参照番号、または 省略した場合現在のリソースファイル

SET TEXT RESOURCEコマンドは、引数＜リソースデータ＞に渡されるテキストまたは文字列を使って引数＜リソースID＞に渡されるIDを持つテキスト（"TEXT"）リソースを作成または上書きします。

リソースが見つからなかった場合、空の文字列が返され、システム変数OKに0が設定されます。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイルに追加されます。＜リソースファイル＞を省略した場合は、リソースはリソースファイル連鎖（最後にオープンしたリソースファイル）の上部にある現在のファイルに追加されます。

注：テキストリソースは、最大32000バイトの文字を使用することができます。

参照：SET STRING RESOURCE

システム変数とシステムセット

リソースが上書きされた場合は、OKに1が設定されます。それ以外は0が設定されます。

GET PICTURE RESOURCE

GET PICTURE RESOURCE (リソースID ; リソースデータ {; リソースファイル})

引数	タイプ		説明
リソースID	数値	→	リソースID番号
リソースデータ	フィールド または変数	→	ピクチャを受信するピクチャフィールド またはピクチャ変数
		←	PICTリソースの内容
リソースファイル	DocRef	→	リソースファイル参照番号、または 省略した場合、オープンしているすべての リソースファイル

GET PICTURE RESOURCEコマンドは、引数＜リソースID＞に渡されるIDを持つピクチャ（"PICT"）リソースに格納されているピクチャを引数＜リソースデータ＞のピクチャフィールドまたはピクチャ変数に返します。

リソースが見つからなかった場合、＜リソースデータ＞はそのまま変わらず、システム変数OKに0が設定されます。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。＜リソースファイル＞を省略した場合は、リソースファイル連鎖内で最初に見つかったリソースの出現が返されます。

注：ピクチャリソースは、少なくとも数メガバイトのサイズを使用することができます。

▼ **RESOURCE LIST**コマンドの例を参照してください。

参照：GET ICON RESOURCE、ON ERR CALL、SET PICTURE RESOURCE

システム変数とシステムセット

リソースが見つかった場合は、OKに1が設定されます。それ以外は0が設定されます。

エラー処理

ピクチャをロードするための十分なメモリがない場合、エラーが生成されます。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

SET PICTURE RESOURCE

SET PICTURE RESOURCE (リソースID ; リソースデータ {; リソースファイル})

引数	タイプ		説明
リソースID	数値	→	リソースID番号
リソースデータ	ピクチャ	→	PICTリソースの新しい内容
リソースファイル	DocRef	→	リソースファイル参照番号、または 省略した場合現在のリソースファイル

SET PICTURE RESOURCEコマンドは、引数＜リソースデータ＞に渡されるピクチャを使って引数＜リソースID＞に渡されるIDを持つピクチャ（"PICT"）リソースを作成または上書きします。

リソースが見つからなかった場合、空の文字列が返され、システム変数OKに0が設定されます。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイルに追加されます。＜リソースファイル＞を省略した場合は、リソースはリソースファイル連鎖（最後にオープンしたリソースファイル）の上部にある現在のファイルに追加されます。

注：ピクチャリソースは、少なくとも数メガバイトのサイズを使用することができます。

参照：GET PICTURE RESOURCE

システム変数とシステムセット

リソースが上書きされた場合は、OKに1が設定されます。それ以外は0が設定されます。

GET ICON RESOURCE

GET ICON RESOURCE (リソースID ; リソースデータ {; リソースファイル})

引数	タイプ		説明
リソースID	数値	→	リソースID番号
リソースデータ	フィールドまたは変数	→	ピクチャを受信するピクチャフィールドまたはピクチャ変数
		←	cicnリソースの内容
リソースファイル	DocRef	→	リソースファイル参照番号、または省略した場合、オープンしているすべてのリソースファイル

GET ICON RESOURCEコマンドは、引数＜リソースID＞に渡されるIDを持つカラーアイコン（"cicn"）リソースに格納されているアイコンを引数＜リソースデータ＞のピクチャフィールドまたはピクチャ変数に返します。

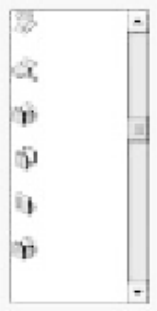
リソースが見つからなかった場合、＜リソースデータ＞はそのまま変わらず、システム変数OKに0が設定されます。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。＜リソースファイル＞を省略した場合は、リソースファイル連鎖内で最初に見つかったリソースの出現が返されます。

▼ 次の例は、稼働している4Dアプリケーション内に配置されているカラーアイコンをピクチャ配列にロードします：

```
If (On Windows)
   $vh4DResFile:=Open resource file(Replace string(Application file ;".EXE" ; ".RSR"))
Else
   $vh4DResFile:=Open resource file(Application file)
End if
RESOURCE LIST("cicn" ; $alResID ; $asResName ; $vh4DResFile)
$vlNbIcons:=Size of array($alResID)
ARRAY PICTURE(ag4DIcon ; $vlNbIcons)
For ($vlElem ; 1 ; $vlNbIcons)
   GET ICON RESOURCE($alResID{$vlElem} ; ag4DIcon{$vlElem} ; $vh4DResFile)
End for
```

このコードを実行すると、配列がフォーム内に表示されると、次のように見えます。



参照：GET PICTURE RESOURCE

システム変数とシステムセット

リソースが見つかった場合は、OKに1が設定されます。それ以外は0が設定されます。

GET RESOURCE

GET RESOURCE (リソースタイプ;リソースID;リソースデータ{;リソースファイル})

引数	タイプ		説明
リソースタイプ	文字列	→	4バイトのリソースタイプ
リソースID	数値	→	リソースID番号
リソースデータ	BLOB	→	データを受信するBLOBフィールドまたはBLOB変数
		←	リソースの内容
リソースファイル	DocRef	→	リソースファイル参照番号、または省略した場合、オープンしているすべてのリソースファイル

GET RESOURCEコマンドは、引数＜リソースタイプ＞と＜リソースID＞に渡されるタイプとIDを持つカリソースの内容を引数＜リソースデータ＞のBLOBフィールドまたはBLOB変数に返します。

重要；＜リソースタイプ＞には、4バイトの文字列を受け渡す必要があります。

リソースが見つからなかった場合、＜リソースデータ＞はそのまま変わらず、システム変数OKに0が設定されます。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。＜リソースファイル＞を省略した場合は、リソースファイル連鎖内で最初に見つかったリソースの出現が返されます。

注：リソースは、少なくとも数メガバイトのサイズを使用することができます。

プラットフォームの独立性：MacOSベースのリソースを使って作業していることを忘れないでください。プラットフォームが何であろうと、倍長整数と同じ内部リソースデータはBLOB用の「Macintosh byte ordering」定義済み定数を使って格納されます。Windows上では、（ストリングリストリソースおよびピクチャリソース等の）標準リソースのデータは必要なときに自動的にバイトスワップされます。これに対して、ユーザ自身の内部データストラクチャを作成および使用する場合は、BLOB（つまり、**BLOB to longint**関数への「Macintosh byte ordering」定数の受け渡し）から取り出すデータをユーザの責任のもとでバイトスワップします。

▼ **SET RESOURCE**コマンドの例を参照してください。

参照：BLOB Commands、Resources、SET RESOURCE

システム変数とシステムセット

リソースが見つかった場合は、OKに1が設定されます。それ以外は0が設定されます。

エラー処理

リソースをロードするための十分なメモリがない場合、エラーが生成されます。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

SET RESOURCE

SET RESOURCE (リソースタイプ; リソースID; リソースデータ {; リソースファイル})

引数	タイプ		説明
リソースタイプ	文字列	→	4バイトのリソースタイプ
リソースID	数値	→	リソースID番号
リソースデータ	BLOB	→	リソースの新しい内容
リソースファイル	DocRef	→	リソースファイル参照番号、または 省略した場合、現在のリソースファイル

SET RESOURCEコマンドは、引数＜リソースデータ＞BLOBに渡されるデータを使って引数＜リソースタイプ＞と＜リソースID＞に渡されるタイプとIDを持つカリソースを作成または上書きします。

重要；＜リソースタイプ＞には、4バイトの文字列を受け渡す必要があります。

リソースが見つからなかった場合、＜リソースデータ＞はそのまま変わらず、システム変数OKに0が設定されます。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイルに追加されます。＜リソースファイル＞を省略した場合は、リソースはリソースファイル連鎖（最後にオープンしたリソースファイル）の上部にある現在のファイルに追加されます。

注：リソースは、少なくとも数メガバイトのサイズを使用することができます。

プラットフォームの独立性：MacOSベースのリソースを使って作業していることを忘れないでください。プラットフォームが何であろうと、倍長整数と同じ内部リソースデータはBLOB用の「Macintosh byte ordering」定義済み定数を使って格納されます。Windows上では、（ストリングリストリソースおよびピクチャリソース等の）標準リソースのデータは必要なときに自動的にバイトスワップされます。これに対して、ユーザ自

身の内部データストラクチャを作成および使用する場合は、BLOB（つまり、**BLOB to longint**関数への「Macintosh byte ordering」定数の受け渡し）から取り出すデータをユーザの責任のもとでバイトスワップします。

例：

4Dセッション中にインタープロセス変数でユーザのプレファレンス（環境設定）を管理すると仮定します。この環境設定をセッション間で保存するには、次のように記述します：

1. **SAVE VARIABLES**コマンドと**LOAD VARIABLES**コマンドを使って、ディスク上の変数文書にその変数を保存して取り出します。

2. **VARIABLE TO BLOB**コマンド、**BLOB TO DOCUMENT**コマンド、**DOCUMEN T TO BLOB**、**BLOB TO VARIABLE**コマンドを使って、ディスク上のBLOB文書にその変数を保存して取り出します。

3. **VARIABLE TO BLOB**コマンド、SET RESOURCEコマンド、GET RESOURCEコマンド、BLOB TO VARIABLEコマンドを使って、ディスク上のリソースファイルにその変数を保存して取り出します。

次の例は、3番目のメソッドの例です。「On Exit」データベースメソッドに記述します：

```
`「On Exit」データベースメソッド
If (Test path name("DB_Prefs")#Is a document)
   $vhResFile:=Create resource file("DB_Prefs")
Else
   $vhResFile:=Open resource file("DB_Prefs")
End if
If (OK=1)
   VARIABLE TO BLOB(<>bAutoRepeat ; $vxPrefData)
   VARIABLE TO BLOB(<>lCurTable ; $vxPrefData ; *)
   VARIABLE TO BLOB(<>sDfltOption ; $vxPrefData ; *)
   `...
   SET RESOURCE("PREF" ; 26500 ; $vxPrefData ; $vhResFile)
   CLOSE RESOURCE FILE($vhResFile)
End if
```

「On Startup」データベースに次のように記述します：

```
`「On Startup」データベースメソッド
C_BOOLEAN(<>bAutoRepeat)
C_LONGINT(<>lCurTable)
$vbDone:=False
$vhResFile:=Open resource file("DB_Prefs")
If (OK=1)
   GET RESOURCE("PREF" ; 26500 ; $vxPrefData ; $vhResFile)
   If (OK=1)
      $vlOffset:=0
      BLOB TO VARIABLE($vxPrefData ; <>bAutoRepeat ; $vlOffset)
      BLOB TO VARIABLE($vxPrefData ; <>lCurTable ; $vlOffset)
      BLOB TO VARIABLE($vxPrefData ; <>sDfltOption ; $vlOffset)
      `...
      $vbDone:=False
   End if
   CLOSE RESOURCE FILE($vhResFile)
End if
If(Not($vbDone))
   <>bAutoRepeat:=False
   <>lCurTable:=0
   ARRY STRING(127 ; 濛sDfltOption ; 0)
End if
```

参照：BLOB Commands、GET RESOURCE

システム変数とシステムセット

リソースが見つかった場合は、OKに1が設定されます。それ以外は0が設定されます。

Get resource name

Get resource name (リソースタイプ ; リソースID {; リソースファイル}) → 文字列

引数	タイプ		説明
リソースタイプ	文字列	→	4バイトのリソースタイプ
リソースID	数値	→	リソースID番号
リソースファイル	DocRef	→	リソースファイル参照番号、または 省略した場合、オープンしているすべての リソースファイル
関数の返す値	文字列	←	リソースの名前

Get resource name関数は、引数＜リソースタイプ＞に渡されるタイプかつ引数＜リソースID＞に渡されるIDを持つリソースの名前を返します。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。＜リソースファイル＞を省略した場合は、現在オープンされているすべてのリソースファイル内で検索されます。

リソースが存在しない場合、**Get resource name**関数は空の文字列を返し、システム変数OKに0が設定されます。

▼ 次の例は、任意のリソースをコピーするプロジェクトメソッドです。そして、あるリソースファイルから別のリソースファイルにそのリソース名と属性をコピーします：

```
` 「COPY RESOURCE」プロジェクトメソッド
` COPY RESOURCE (文字列 ; 倍長整数 ; 時間 ; 時間 )
` COPY RESOURCE ( リソースタイプ ; リソースID ; コピー元ファイル ; コピー先ファイル)
C_STRING (4 ; $1)
C_LONGINT ($2)
C_TIME ($3;$4)
C_BLOB ($vxResData)
GET RESOURCE ($1 ; $2 ; $vxData ; $3)
If (OK=1)
   SET RESOURCE ($1 ; $2 ; $vxData ; $4)
   If (OK=1)
      SET RESOURCE NAME ($1 ; $2 ; Get resource name ($1 ; $2 ; $3) ; $4)
      SET RESOURCE PROPERTIES($1; $2 ; Get resource properties ($1 ; $2 ; $3); $4)
   End if
End if
```

このメソッドをあなたのアプリケーションに組み込むと、次のように記述できます：

```
` ファイルAからファイルBにリソース 'DATA' ID = 15000をコピーする
COPY RESOURCE ("DATA" ; 15000 ; $vhResFileA ; $vhResFileB)
```

参照：SET RESOURCE PROPERTIES

システム変数とシステムセット

リソースが存在する場合は、OKに1が設定されます。それ以外は0が設定されます。

SET RESOURCE NAME

SET RESOURCE NAME (リソースタイプ ; リソースID ; リソース名 {; リソースファイル})

引数	タイプ		説明
リソースタイプ	文字列	→	4バイトのリソースタイプ
リソースID	数値	→	リソースID番号
リソース名	文字列	→	リソースの新しい名前
リソースファイル	DocRef	→	リソースファイル参照番号、または 省略した場合、現在のリソースファイル

SET RESOURCE NAMEコマンドは、引数＜リソースタイプ＞に渡されるタイプかつ引数＜リソースID＞に渡されるIDを持つリソースの名前を変更しします。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。＜リソースファイル＞を省略した場合は、現在オープンされているリソースファイル内で検索されます。

リソースが存在しない場合、**SET RESOURCE NAME**コマンドは何も行わずに、システム変数OKに0が設定されます。

警告：4Dまたは任意のシステムファイルに属しているリソースの名前を変更してはいけません。もし、リソース名を変更すると、予期しないシステムエラーを引き起こす原因になります。

注；リソース名は、最大255バイトまで指定することができます。この名前は厳格な区別を行いませんが、発音記号の区別は行います。

▼ **Get resource name**関数の例を参照してください。

参照：SET RESOURCE PROPERTIES

システム変数とシステムセット

リソースが存在する場合は、OKに1が設定されます。それ以外は0が設定されます。

Get resource properties

Get resource properties(リソースタイプ ; リソースID {; リソースファイル}) → 数値

引数	タイプ		説明
リソースタイプ	文字列	→	4バイトのリソースタイプ
リソースID	数値	→	リソースID番号
リソースファイル	DocRef	→	リソースファイル参照番号、または 省略した場合、オープンしているすべての リソースファイル
関数の返す値	数値	←	リソースの属性

Get resource properties関数は、引数＜リソースタイプ＞に渡されるタイプかつ引数＜リソースID＞に渡されるIDを持つリソースの属性を返します。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。＜リソースファイル＞を省略した場合は、現在オープンされているすべてのリソースファイル内で検索されます。

リソースが存在しない場合、**Get resource properties**関数は 0を返し、システム変数OKに0が設定されます。

Get resource properties関数によって返される数値は、ビットが特別の意味を持っているビットフィールド値として表す必要があります。リソース属性およびその影響に関する詳細は、**SET RESOURCE PROPERTIES**コマンドの節を参照してください。

▼ **Get resource name**関数の例を参照してください。

参照：SET RESOURCE NAME

システム変数とシステムセット

リソースが存在する場合は、OKに1が設定されます。それ以外は0が設定されます。

SET RESOURCE PROPERTIES

SET RESOURCE PROPERTIES (リソースタイプ ; リソースID ; リソース属性 {; リソースファイル})

引数	タイプ		説明
リソースタイプ	文字列	→	4バイトのリソースタイプ
リソースID	数値	→	リソースID番号
リソース属性	数値	→	リソースの新しい属性
リソースファイル	DocRef	→	リソースファイル参照番号、または 省略した場合、現在のリソースファイル

SET RESOURCE PROPERTIESコマンドは、引数＜リソースタイプ＞に渡されるタイプかつ引数＜リソースID＞に渡されるIDを持つリソースの属性を変更します。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。＜リソースファイル＞を省略した場合は、現在オープンされているリソースファイル内で検索されます。

リソースが存在しない場合、**SET RESOURCE PROPERTIES**コマンドは何も行わずに、システム変数OKに0が設定されます。

引数＜リソース属性＞に渡す数値は、ビットが特別の意味を持っているビットフィールド値として表す必要があります。下記は、4th Dimensionによってあらかじめ用意されている定義済みの定数です：

定数	タイプ	値
System heap resource mask	倍長整数	64
System heap resource bit	倍長整数	6
Purgeable resource mask	倍長整数	32
Purgeable resource bit	倍長整数	5
Locked resource mask	倍長整数	16
Locked resource bit	倍長整数	4
Protected resource mask	倍長整数	8
Protected resource bit	倍長整数	3
Preloaded resource mask	倍長整数	4
Preloaded resource bit	倍長整数	2
Changed resource mask	倍長整数	2
Changed resource bit	倍長整数	1

上記の定数を使って、任意のリソース属性を構築することができます。後述の例を参照してください。

リソース属性とその及ぼす影響について

■ Syatem heap

この属性を設定すると、リソースは4Dメモリ内以外のシステムメモリの中にロードされます。もし、あなたが何を行っていいのか“実は”わからない場合は、この属性を使用すべきではありません。

■ Purgeable

この属性を設定すると、リソースがロードされた後、他のデータを割り当てるために空間が必要な場合、そのリソースをメモリからパージ（消去）することができます。4D BLOBの中にリソースをロードするので、メモリの使用を減らすためにパージ可能なユーザ独自のリソースを持つことは良い方法です。ただし、作業セッション中にこのリソースに頻繁にアクセスする場合、パージされたリソースの再ロードのディスクアクセスを減らすためにそれをパージ不可にしたい場合があるかもしれません。

■ Locked

この属性を設定すると、リソースがメモリの中にロードされた後、そのリソースを再配置することができなくなります（移動不可）。ロックされたリソースは、たとえパージ可能であっても、パージされません。リソースのロックは、メモリ空間のフラグメントという望ましくない影響を及ぼします。もし、あなたが何を行っていいのか“実は”わからない場合は、この属性を使用すべきではありません。

■ Protected

この属性を設定すると、リソースの名前、ID番号、およびリソースの内容を変更することができなくなります。また、このリソースを削除することもできなくなります。ただし、**SET RESOURCE PROPERTIES**コマンドを呼び出して、この属性を消去することができます。そして、また、このリソースを変更したり削除することができるようになります。ほとんど、この属性を使用することはありません。

注：この属性は、Windows上では機能しません。

■ Preloaded

この属性を設定すると、リソースが配置されているリソースファイルがオープンされると、そのリソースは自動的にメモリの中にロードされます。この属性は、任意のリソースファイルがオープンされる際にロードされるリソースの最適化に有効です。ほとんど、この属性を使用することはありません。

■ Changed

この属性を設定すると、リソースが配置されているリソースファイルがクローズされると、“ディスク上に保存される必要がある”というマークが付けられます。**SET RESOURCE**コマンドはリソースの書き込みおよび上書きを行うことができるので、もし、

あなたが何を行っていいのか“実は”わからない場合は、この属性を使用すべきではありません。

警告：4Dまたは任意のシステムファイルに属しているリソースの名前を変更してはいけません。もし、リソース名を変更すると、予期しないシステムエラーを引き起こす原因になります。

例：

1. **Get resource name**関数の例を参照してください。

2. 次の例は、リソース「"STR#" ID=17000」をパージ可能にします。ただし、その他の属性はそのまま変更しません：

```
$vlResAttr:=Get resource properties ('STR#' ; 17000 ; $vhResFile)
SET RESOURCE PROPERTIES('STR#' ; 17000 ; $vlResAttr ?+ Purgeable resource bit ; $vhMyResFile)
```

3. 次の例は、リソース「"STR#" ID=17000」の属性を再ロードとパージ不可にします：

```
SET RESOURCE PROPERTIES('STR#' ; 17000 ; Preloaded resource mask ; $vhResFile)
```

4. 次の例は、リソース「"STR#" ID=17000」の属性を再ロードとパージ可にします：

```
SET RESOURCE PROPERTIES('STR#' ; 17000 ; Preloaded resource mask + Purgeable resource mask ; $vhResFile)
```

参照：SET RESOURCE NAME

システム変数とシステムセット

リソースが存在する場合は、OKに1が設定されます。それ以外は0が設定されます。

DELETE RESOURCE

DELETE RESOURCE (リソースタイプ ; リソースID {; リソースファイル})

引数	タイプ		説明
リソースタイプ	文字列	→	4バイトのリソースタイプ
リソースID	数値	→	リソースID番号
リソースファイル	DocRef	→	リソースファイル参照番号、または 省略した場合、現在のリソースファイル

DELETE RESOURCEコマンドは、引数＜リソースタイプ＞に渡されるタイプかつ引数＜リソースID＞に渡されるIDを持つリソースを削除します。

オプション引数＜リソースファイル＞に有効なリソースファイル参照番号を渡すと、そのリソースはそのリソースファイル内でのみ検索されます。＜リソースファイル＞を省略した場合は、現在オープンされているリソースファイル内で検索されます。

リソースが存在しない場合、**DELETE RESOURCE**コマンドは何も行わずに、システム変数OKに0が設定されます。リソースが見つかって削除された場合は、システム変数OKに1が設定されます。

警告：4Dまたは任意のシステムファイルに属しているリソースの名前を変更してはいけません。もし、リソース名を変更すると、予期しないシステムエラーを引き起こす原因になります。

例：

1. 次の例は、リソース「"STR# ID=20000」を削除します：

```
`この例では現在オープンしているすべてのリソースの中で最初に見つかった
`「"STR# ID=20000」リソースを削除する点に注目してください
DELETE RESOURCE ("STR#" ; 20000)
```

2. 次の例は、指定されたリソースファイルの中で見つかったリソース「"STR# ID=20000」を削除します：

```
`この例ではローカル変数 “ $vhResFile ” によって指定されたリソースファイル内にそのリソ
`ースが存在する場合にのみ「"STR# ID=20000」リソースが削除する点に注目してください
DELETE RESOURCE ("STR#" ; 20000 ; $vhResFile)
`$vhResFile ” によって指定された以外の現在オープンされているリソースファイルの中に
`このようなリソースがある場合は、このリソースは何も手をつけられない点に注目してくだ
`さい
```

3. プロジェクトメソッド「DELETE RESOURCE OF TYPE」は、(1番目の引数で）指定されたリソースファイルから（2番目の引数で）指定されたタイプのすべてのリソースを削除します：

```
` 「DELETE RESOURCES OF TYPE」プロジェクトメソッド
` DELETE RESOURCES OF TYPE (時間 ; 文字列 )
` DELETE RESOURCES OF TYPE (リソースファイル ; リソースタイプ)
C_TIME($1)
C_STRING(4 ; $2)
RESOURCE LIST($2 ; $aiResID ; $asResName ; $1)
If(OK=1)
   For($vlElem ; 1 ; Size of array($aiResID))
      DELETE RESOURCE($2 ; $aiResID{$vlElem} ; $1)
   End for
End if
```

このプロジェクトメソッドをデータベースに組み込むと、次のように記述できます：

```
`リソースファイル「$vhResFile」から “ PREF ” タイプのすべてのリソースを削除する
DELETE RESOURCES OF TYPE ($vhResFile ; "PREF")
```

4. プロジェクトメソッド「DELETE RESOURCE BY NAME」は、リソースファイル名がわかっている（特定タイプの）リソースを削除します：

```
` 「DELETE RESOURCE BY NAME」プロジェクトメソッド
` DELETE RESOURCE BY NAME (時間 ; 文字列 ; 文字列 )
` DELETE RESOURCE BY NAME ( リソースファイル ; リソースタイプ ; リソース名)
C_TIME($1)
C_STRING(4 ; $2)
C_STRING(255 ; $3)
RESOURCE LIST($2 ; $aiResID ; $asResName ; $1)
If(OK=1)
   $vlElem:=Find in array($asResName ; $3)
   If($vlElem>0)
      DELETE RESOURCE($2 ; $aiResID{$vlElem} ; $1)
   End if
End if
```

このプロジェクトメソッドをデータベースに組み込むと、次のように記述できます：

```
`リソースファイル「$vhResFile」から “ 標準設定 ” という名前を持つ “ PREF ” リソースを削除する
DELETE RESOURCE BY NAME ($vhResFile ; "PREF" ; "標準設定")
```

参照：RESOURCE LIST、SET RESOURCE PROPERTIES

システム変数とシステムセット：

リソースが存在しない場合は、OKに1が設定されます。リソースが削除された場合は、システム変数OKに 1が設定されます。

カレントセレクションコマンド　47

この章では、「ルーチン」エディタの「Selection」テーマ内にあるカレントセレクションコマンドについて説明します。この章のコマンドは、画面上にレコードを表示するためのものです。リストにデータを入力したり、表示します。

ほとんどのコマンドは、カレントセレクションの削除、修正、カレントセレクション間の移動など、カレントセレクションの操作を実行します。

ALL RECORDS
APPLY TO SELECTION
Before Selection
DELETE SELECTION
DISPLAY SELECTION
End selection
FIRST RECORD
GOTO SELECTED RECORD
ONE RECORD SELECT
LAST RECORD
MODIFY SELECTION
NEXT RECORD
PREVIOUS RECORD
Records in selection
REDUCE SELECTION
SCAN INDEX
Selected record number

DISPLAY SELECTION
MODIFY SELECTION

DISPLAY SELECTION ({テーブル} ; {*})
MODIFY SELECTION ({テーブル} ; {*})

引数	タイプ	説明
テーブル	テーブル	表示するテーブル
*		レコードが1件の場合にも出力フォームを使用し、 入力フォームのスクロールバーを表示しない

DISPLAY SELECTIONコマンドと**MODIFY SELECTION**コマンドは、カレント出力フォームを介して、＜テーブル＞のカレントセレクションを表示します。レコードは、「ユーザ」モードの出力に類似のスクロール可能なリストとして表示されます。ユーザがレコードをダブルクリックすると、そのレコードはカレント入力フォームで表示されます。リストは最前面にあるウインドウに表示されます。

次の図に、**DISPLAY SELECTION**コマンドまたは**MODIFY SELECTION**コマンドで表示された出力フォームの例を示します。

DISPLAY SELECTIONコマンドまたは**MODIFY SELECTION**コマンドを実行した直後は、カレントレコードが存在しない場合があります。レコードを1件選択するために、**FIRST RECORD**コマンドまたは**LAST RECORD**コマンドを使用してください。

MODIFY SELECTIONコマンドは入力フォーム中でレコードを修正することができますが、**DISPLAY SELECTION**コマンドは入力フォーム中でレコードを修正することができません。

MODIFY SELECTIONコマンドでもロックされたレコードは修正できません。その代わり、ユーザにそのレコードが使用されていることを知らせるダイアログボックスを表示します。そして、そのレコードを“ リードオンリー ”(読み込み専用)状態で表示します。

セレクションにレコードが1件しか存在しないときに、オプションのアスタリスク(*)を指定していなければ、そのレコードは出力フォームの代わりに入力フォームで表示されます。アスタリスク(*)を指定した場合は、出力フォームを通して1件のレコードが表示されます。また、ユーザが入力フォームにレコードを表示した場合には、スクロールバーは表示されません。

「終了」ボタンが、自動的にリストの最後に組み込まれます。フォーム上に変数、またはアクティブオブジェクトを追加すると「終了」ボタンが消えます。このボタンをクリックするとコマンドを終了します。カスタムボタンを代わりに使用することもできます。出力フォームのフッタエリアにこのボタンを作成ことができます。また、レコード表示を終了させるために「登録」ボタンや「キャンセル」ボタンを使用することもできます。

ユーザは、カレントセレクションをスクロールし、該当レコードをクリックして選択することができます。ユーザが別のレコードをクリックした場合は、最初のレコードの選択が解除され、そのレコードが選択されます。また、ユーザは連続するレコードをグループとして選択することもできます。この場合は、選択する先頭のレコードをクリックし、最後のレコードを“ Shift ”キーを押しながらマウスをクリック(シフト-クリック)します。不連続のレコードを選択するには、選択したいレコードごとに、“ Ctrl（Macintosh版では、コマンド）”キーを押しながらマウスをクリックします。

DISPLAY SELECTIONコマンドまたは**MODIFY SELECTION**コマンドの実行後、ユーザが選択したレコードは“ UserSet ”という名前のセットに格納されます。UserSetは1つのプロセスに対して1つしか存在しません。このセットは、**DISPLAY SELECTION**コマンドまたは**MODIFY SELECTION**コマンドに付随します。UserSetに関する詳細は、第41章を参照してください。

フォームメソッドまたはオブジェクトメソッドが存在する場合に、表示される各レコードに対して1回ずつBeforeフェーズが発生します。その後でIn Headerフェーズが発生し、続いて、BeforeとDuringフェーズが同時に発生します。また、ユーザがボタンをクリックしたり、メニューを選択したり、レコードをダブルクリックした場合には、出力フォームのフォームメソッドのDuringフェーズが発生します。

▼ 次の例は、最初に、[従業員]テーブルのすべてのレコードを選択します。次に、**DISPLAY SELECTION**コマンドを使用してレコードを表示し、**PRINT SELECTION**コマンドで選択されたレコードを印刷します：

```
ALL RECORDS ([従業員])                ` すべてのレコードの選択
DISPLAY SELECTION ([従業員] ; *)      ` レコードの表示
PRINT SELECTION ([従業員])            ` ユーザが選択したレコードの印刷
```

▼ 次の例は、**DISPLAY SELECTION**コマンドと**MODIFY SELECTION**コマンドのフォームイベントを確認したい場合に、チェックするために必要な検査を紹介しています。下記に示されているメソッドは、表示される出力フォームのフォームメソッドです：

```
Case of
   \ (Before & During)
      $イベント:="Before & Duringフェーズ：レコード表示"
   \ (Before)
      bnxt:=0                                  `次レコードボタンの変数を初期化する
      bprv:=0                                  `前レコードボタンの変数を初期化する
      FromInput:=False
      $イベント:="今にもフォームが現われようとしています"
   \ (In header)
      $イベント:="ヘッダが描画されます"
   \ (Menu selected # 0)
      $イベント:="メニューが選択されました"
   \ (bボタン1=1)
      $イベント:="ボタン1がクリックされました"
   \ (During)
      Case of
         \ (bnxt=1)
            $イベント:="Duringフェーズ：次のレコードに移動"
         \ (bprv=1)
            $イベント:="Duringフェーズ：前のレコードに移動"
         \ (FromInput)
            FromInput:=False
            $イベント:="Duringフェーズ：入力フォームを閉じる"
      Else
         $イベント:="Duringフェーズ：ダブルクリック"
         If ((Selected record number ([テーブル1]) % 2) = 0)
            INPUT FORM ([テーブル1] ; "入力")
         Else
            INPUT FORM ([コマンド] ; "空白")
         End if
      End Case
End case
SET WINDOW TITLE ($イベント)
```

この例では、実行しているフェーズを示すために**SET WINDOW TITLE**コマンドを使用します。

Duringフェーズの最終検査により、ユーザがダブルクリックしたレコードをチェックすることができます。このテストを正確に行うには、フッタエリアにカスタムボタンを使用する必要があります。そうしないと、デフォルトの「終了」ボタンがDuringフェーズを生成し、そのクリックによりユーザはそのDuringフェーズにおける検査によりトラップされます。

▼ 次の例は、選択されたレコード番号の決定が基になっています。もし、レコードが偶数の位置にあれば、入力フォームを“ なし ”フォームに変更します。このフォームは、標準のコントロールパネルを非表示にするための「透明」ボタンがある以外は何のオブジェクトもありません。この“ なし ”入力フォームのフォームメソッドを次に示します：

```
If (Before)
   FromInput :=True                    ` 出力フォームに知らせる
   CANCEL                              ` ただちにフォームを離れる
End if
```

その他の入力フォームにもプロセス変数“ FromInput ”に“ True(真) ”を代入します。これにより、出力フォームのDuringフェーズで入力フォームへ入ったり、離れたりする違いがわかります。

“ Ctrl（Macintosh版では、コマンド）- ピリオド ”キーや“ Enter ”キーは、「b終了」ボタンによって機能しなくなります。フォームのフッタエリアにボタンを1つコピーすると、2つ目のコピーはそのフッタの下側に作成されます。

フォームメソッドのコメントで説明されているように、入力フォームに移動ボタンが用意されている場合に、それらのボタンをクリックすることにより、出力フォームにDuringフェーズが発生します。そして、これらのボタンのオブジェクトメソッドに1つ以上のプロセス変数を設定することによって、フォームの別のページに移動したり、入力フォームを離れても、それを見つけることができます。

この例は、出力フォームメソッドの構成についてだけでなく、レコードをダブルクリックした際のコマンドの標準的な取り扱いの修正方法についても記述しています。

ALL RECORDS

ALL RECORDS ({テーブル})

引数	タイプ	説明
テーブル	テーブル	全レコードを選択するテーブル

ALL RECORDSコマンドは、カレントセレクションとして、＜テーブル＞の全レコードを選択します。**ALL RECORDS**コマンドは、先頭のレコードをディスクからロードし、カレントレコードに設定します。**ALL RECORDS**コマンドは、レコードの順序をデフォルトのレコード順序に戻します。

▼ 次の例は、テーブルのすべてのレコードを表示します：

```
ALL RECORDS ([従業員])                `従業員テーブルのすべてのレコードを選択
DISPLAY SELECTION ([従業員])          `出力フォームにレコードを表示
```

Records in selection

Records in selection({テーブル}) → 数値

引数	タイプ	説明
テーブル	テーブル	カレントセレクションの数を求めるテーブル

Records in selection関数は、＜テーブル＞ のカレントセレクションのレコード数を返します。**Records in file**関数はテーブルの全レコード数を返します。

▼ 次の例は、カレントセレクションのすべてのレコード間を移動するのによく使用されるループ処理です。**APPLY TO SELECTION**コマンドを代用してもよいでしょう：

```
FIRST RECORD ([従業員])               `カレントセレクションのレコード数だけ繰り返す
  `カレントセレクションを1つずつループ
For ($i ; 1 ; Records in selection ([従業員]))
  Do Something                        `レコードに対して何らかの処理を実行
  NEXT RECORD ([従業員])              `次のレコードに移動
End for
```

APPLY TO SELECTION

APPLY TO SELECTION ({テーブル} ; ステートメント)

引数	タイプ	説明
テーブル	テーブル	ステートメントを適用するテーブル
ステートメント	ステートメント	1行のステートメントまたはグローバルメソッド

APPLY TO SELECTIONコマンドは、＜テーブル＞のカレントセレクションに対して＜ステートメント＞を適用します。＜ステートメント ＞は、1行のステートメントまたはプロジェクトメソッドのどちらでも構いません。＜ステートメント＞が＜テーブル＞のレコードを修正した場合は、そのレコードをディスクに保存しますが、レコードを修正しない場合には保存しません。また、カレントセレクションがない場合は、**APPLY TO SELECTION**コマンドは何も行いません。自動リレートの場合に、＜ステートメント＞はリレートされたテーブルのフィールドを含むことができます。

APPLY TO SELECTIONコマンドは、カレントセレクションの情報(例えば、合計など)を求めるため、あるいはカレントセレクションを修正するため(例えば、フィールドの頭文字を大文字に変えるなど)に使用します。このコマンドをトランザクション処理内で使用する場合に、トランザクション処理が取り消されると、すべての変更は無視されます。

APPLY TO SELECTIONコマンドを実行している間、処理の進捗を表すサーモメータが表示されます。**MESSAGES OFF**コマンドで、サーモメータの表示を取り消すことができます。ユーザが処理を取り消した場合は、システム変数OKに 0が代入されます。それ以外の場合には、システム変数OKに1がセットされます。

▼次の例は、テーブル上のすべての(アルファベットの)名前を大文字に変換します：

```
APPLY TO SELECTION ([従業員] ; [従業員]名前:=Uppercase ([従業員]名前))
```

APPLY TO SELECTIONコマンドの実行中にレコードが、他のユーザの修正などによってロックされた場合、そのレコードは保存されません。ロックされたレコードは、すべて“ LockedSet ”と呼ばれるセットに格納されます。**APPLY TO SELECTION**コマンドが実行された後で、LockedSetを判定し、ロックされているレコードがあるかどうかを調べます。

▼次のループは、すべてのレコードを修正するまで実行します：

```
Repeat  ` ロックされたレコードがなくなるまでループ
  APPLY TO SELECTION ([従業員] ; [従業員]名前[[1]]:=Uppercase ([従業員]名前[[1]]))
Until (Records in  table ([従業員]) = 0)        ` すべてのレコードがなくなるまで
```

DELETE SELECTION

DELETE SELECTION ({テーブル})

引数	タイプ	説明
テーブル	テーブル	カレントセレクションを削除するテーブル

DELETE SELECTIONコマンドは、＜テーブル＞のカレントセレクションをすべて削除します。カレントセレクションが空の場合は、**DELETE SELECTION**コマンドは何も行いません。レコードが削除された後で、カレントセレクションは空になります。トランザクション処理中に削除されたレコードは、トランザクション処理が実行または取り消されるまで、他のユーザや他のプロセスに対してロックされます。「テーブル属性」ダイアログボックス内の「すべて消去」オプションを選択してこの**DELETE SELECTION**コマンドを使用すると、削除処理を高速に行うことができます。

警告：レコードの削除は、恒久的な操作です。削除が実行されると元に戻すことはできません。

「テーブル属性」ダイアログボックス内の「すべて消去」オプションを選択することにより、**DELETE SELECTION**コマンドを使用する際、レコード削除処理を高速にすることができます。

▼ 次の例は、[従業員]テーブルの、すべてのレコードを表示します。ユーザは、この中から削除したいものを選択します。この例には、2つのメソッドがあります。第1のメソッドは、レコードを表示するグローバルメソッドです。第2のメソッドは、“ 削除 ” というボタンのオブジェクトメソッドです。次に第1のメソッドを示します：

```
ALL RECORDS ([従業員])                  ` すべてのレコードを選択する
OUTPUT FORM ([従業員]) ; "リスト")       ` 出力フォームをセット
DISPLAY SELECTION ([従業員])            ` すべてのレコードを表示
```

FIRST RECORD

FIRST RECORD ({テーブル})

引数	タイプ	説明
テーブル	テーブル	先頭のレコードに移動するテーブル

FIRST RECORDコマンドは、＜テーブル＞のカレントセレクションの先頭のレコードをディスクからロードし、カレントレコードに設定します。検索、選択、ソートコマンドも先頭のレコードをカレントレコードに設定します。カレントセレクションが空の場合は、**FIRST RECORD**コマンドは何も行いません。このコマンドは、ループ処理に入る前に**USE SET**コマンドの次でよく使用されます。

このコマンドをデータ入力中に実行した場合、「先頭レコード」自動動作ボタンとは違う動作を行います。この場合、Beforeフェーズは発生しません。また、元のカレントレコードは、**SAVE RECORD**コマンドで保存する必要があります。

▼ 次の例は、[顧客]テーブルの最初のレコードを先頭レコードにします：

```
FIRST RECORD ([顧客])
```

LAST RECORD

LAST RECORD ({テーブル})

引数	タイプ	説明
テーブル	テーブル	最後のレコードに移動するテーブル

LAST RECORDコマンドは、＜テーブル＞のカレントセレクションの最終のレコードをディスクからロードし、カレントレコードに設定します。カレントセレクションが空の場合は、**LAST RECORD**コマンドは何も行いません。

このコマンドをデータ入力中に実行した場合には、「最終レコード」自動動作ボタンとは違う動作を行います。この場合、Beforeフェーズは発生しません。また、元のカレントレコードは、**SAVE RECORD**コマンドで保存する必要があります。

▼ 次の例は、[従業員]テーブルの最後のレコードをカレントレコードにします：

```
LAST RECORD ([従業員])
```

NEXT RECORD

NEXT RECORD ({テーブル})

引数	タイプ	説明
テーブル	テーブル	次のレコードに移動するテーブル

NEXT RECORDコマンドは、＜テーブル＞のカレントレコードの次のレコードをディスクからロードし、カレントレコードに設定します。カレントセレクションが空の場合、あるいは、**End selection**関数が“ True(真) ”の場合は、**NEXT RECORD**コマンドは何も行いません。

NEXT RECORDコマンドで、カレントセレクションの最後を超えてカレントレコードポインタを移動した場合は、**End selection**関数は“ True(真) ”を返し、カレントレコードはなくなります。この場合に、**FIRST RECORD**コマンド、 **LAST RECORD**コマンド、**GOTO SELECTED RECORD**コマンドを使用して、カレントレコードポインタをカレントセレクション内に戻します。このコマンドをデータ入力中に使用した場合には、「次レコード」自動動作ボタンとは違う動作をします。この場合に、Beforeフェーズは発生しません。また、元のカレントレコードは、**SAVE RECORD**コマンドで保存する必要があります。

PREVIOUS RECORD

PREVIOUS RECORD ({テーブル})

引数	タイプ	説明
テーブル	テーブル	前のレコードに移動するテーブル

PREVIOUS RECORDコマンドは、＜テーブル＞のカレントレコードの前のレコードをディスクからロードし、カレントレコードにします。カレントセレクションが空の場合、または**Before selection**関数が“ True(真) ”の場合は、**PREVIOUS RECORD**コマンドは何も行いません。

PREVIOUS RECORDコマンドで、カレントセレクションの前にカレントレコードポインタを移動する場合は、**Before selection**関数は“ True(真) ”を返し、カレントレコードはなくなります。この場合に、**FIRST RECORD**、 **LAST RECORD**、**GOTO SELECTED RECORD**の各コマンドを使用して、カレントレコードポインタをカレントセレクション内に戻します。

このコマンドをデータ入力中に使用した場合には、「前レコード」自動動作ボタンとは違う動作をします。この場合、Beforeフェーズは発生しません。また、元のカレントレコードは、**SAVE RECORD**コマンドで保存する必要があります。

Before selection

Before selection ({テーブル}) → ブール

引数	タイプ	説明
テーブル	テーブル	カレントセレクションの前を判定するテーブル

Before selection関数は、カレントレコードポインタが＜テーブル＞のカレントセレクションの前にある場合は“ True(真) ”を返します。**Before selection**関数は、一般に**PREVIOUS RECORD**コマンドで、カレントレコードポインタを先頭のレコードの前に移動したかどうかを調べるために使用します。カレントセレクションが空の場合には、**Before selection**関数は“ True(真) ”を返します。

PRINT SELECTIONコマンド、または「ユーザ」モードの「プリント...」メニューコマンドを選択してレポートを印刷する場合は、**Before selection**関数は最初のヘッダで“ True(真) ”を返します。最初のヘッダを判定するために次のステートメントを使用して、先頭ページに特殊なヘッダを印刷することができます：

```
If (In header & Before selection)
```

▼ 次の例は、レポートの印刷中に使用します。変数“ vタイトル ”を設定し、先頭ページのヘッダエリアに印刷します：

```
` 先頭ページの場合、タイトルをセット
If (In header & Before selection ([探傷検査])
  vタイトル="検査報告書 1996"          ` 最先頭ページのタイトルをセット
Else
  vタイトル:=""                        ` 他のページのタイトルをクリア
End if
```

End selection

End selection({テーブル}) → ブール

引数	タイプ	説明
テーブル	テーブル	テーブルのカレントセレクションの後を判定する

End selection関数は、カレントレコードポインタが＜テーブル ＞のカレントセレクションの後にある場合は “ True(真) ” を返します。一般に**End selection**関数は、**NEXT RECORD**コマンドで、カレントレコードポインタを最後のレコードの後に移動したかどうかをチェックするために使用します。カレントセレクションが空の場合には、**End selection**関数は “ True(真) ” を返します。

カレントレコードのポインタをカレントセレクション内に戻すには、**LAST RECORD**、**FIRST RECORD**、**GOTO SELECTED RECORD**の各コマンドのいずれかを使用します。**PREVIOUS RECORD**コマンドは、ポインタをカレントセレクションに戻すことができません。

PRINT SELECTIONコマンド、または「ユーザ」モードの「プリント...」メニューコマンドを選択してレポートを印刷する場合は、**End selection**関数は最後のフッタで “ True(真) ” を返します。最後のフッタを判定するために次のステートメントを使用して、最終ページに特殊なフッタを印刷することができます：

```
If (In footer & End selection)
```

▼ 次の例はレポートの印刷中に使用します。変数 “ vフッタ ” を設定し、最終ページのフッタエリアに印刷します：

```
` 最終ページの場合、タイトルをセット
If (In footer & End selection ([探傷検査]))
 vフッタ:="江田超音波試験株式会社" ` 最終ページのタイトルをセット
Else
 vフッタ:=""                       ` 他のページのタイトルをクリアする
End if
```

Selected record number

Selected record number ({テーブル}) → 数値

引数	タイプ	説明
テーブル	テーブル	レコード位置番号を求めるカレントレコードの属するテーブル

Selected record number関数は、＜テーブル＞のカレントセレクション内のカレントレコードの位置を返します。レコード位置番号は、**Record number**関数で求めたレコード番号とは違います。(**Record number**関数は、絶対レコード番号を返します。)レコード位置番号は、カレントセレクションに依存します。

カレントセレクションがない場合は、0を返します。

▼ 次の例は、カレントレコードのレコード位置番号を変数に格納します：

```
v位置番号:=Selected record number ([従業員]) ` レコード位置番号を求める
```

GOTO SELECTED RECORD

GOTO SELECTED RECORD ({テーブル} ; レコード位置番号)

引数	タイプ	説明
テーブル	テーブル	移動するレコードの属するテーブル
レコード位置番号	数値	セレクション内のレコード位置

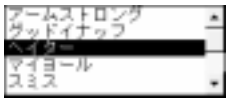
GOTO SELECTED RECORDコマンドは、＜テーブル＞のカレントセレクション内の指定されたレコードに移動し、そのレコードをカレントレコードにします。カレントセレクションは変更されません。引数＜レコード位置番号＞は、カレントセレクション内のレコードの位置を指定します。**Record number**関数で求められるレコード番号ではありません。カレントセレクション内のレコード位置は、カレントセレクション自体を変更する検索などの処理の実行で更新されます。また、カレントセレクション自体を変更しなくても、そのレコードの順番を変えるソートなどの処理を実行した場合にも変更されます。

カレントセレクションにレコードが全く存在しない場合、あるいは指定したレコード位置番号がセレクション内に存在しない場合には、**GOTO SELECTED RECORD**コマンドは、何も行いません。

▼次の例は、セレクション内のレコードのフィールドの内容を“ a名字 ”という配列に取り込み、“ a位置番号 ”という整数配列にレコード位置番号を取り込みます。両方の配列をソートし、その配列を使用してセレクション内のレコードを参照します：

```
` 名字を配列にコピーする。
SELECTION TO ARRAY ([従業員]名字 ; a名字)
` レコード位置番号を格納する配列を定義する
ARRAY LONGINT (a位置番号 ; Size of array (a名字))
For ($i ; 1 ; Size of array (a名字))  ` 配列にレコード位置番号を取り込む
  a位置番号{$i}:=$i
End for
SORT ARRAY (a名字 ; a位置番号 ; >)          ` 両方の配列をソートする。
```

配列“ a名字 ”をスクロールエリアに表示すれば、ユーザは、その中の1つの項目をクリックすることによって選択することができます。ユーザが項目をクリックすると、その項目の位置が数値で配列の名前“ a名字 ”に対して代入されます。次の図の例では、3番目の項目が選択されています。従って、“ a名字 ”に3が代入されます。



“ a名字 ”の値は、セレクションのレコードをロードするのに使用します。また、“ a名字 ”の値は、配列“ a位置番号 ”の配列要素をアクセスするのにも使用します。この場合の配列“ a位置番号 ”の配列要素は、スクロールエリアでクリックされた項目に対応するレコードのレコード位置番号です。次のメソッドは、スクロールエリア“ a名字 ”のオブジェクトメソッドです。ユーザがクリックしたレコードをロードするために**GOTO SELECTED RECORD**コマンドを使用しています。

```
GOTO SELECTED RECORD (a位置番号{a名字})
```

SCAN INDEX

SCAN INDEX (フィールド ; レコード数 ; 方向)

引数	タイプ	説明
フィールド	フィールド	インデックスを読み取るインデックスフィールド
レコード数	数値	返すレコードの数
方向		< インデックスの最後から
		> インデックスの最初から

SCAN INDEXコマンドは、テーブルから＜レコード数＞個のレコードのセレクションを返します。＜方向＞が“ < ”の場合、**SCAN INDEX**コマンドはインデックスの最後から＜レコード数＞個のレコードを返します。＜方向＞が“ > ”の場合、**SCAN INDEX**コマンドはインデックスの先頭から＜レコード数＞個のレコードをテーブルから返します。このコマンドは、インデックスを用いた処理ですので非常に効率が良くなります。

SCAN INDEXコマンドは、インデックスフィールドにのみ使用できます。このコマンドは、カレントプロセスのテーブルのカレントセレクションを変更し、新しいセレクションの先頭のレコードがカレントレコードとなります。

テーブル内のレコード数より多くのレコードを指定した場合には、**SCAN INDEX**コマンドはすべてのレコードを返します。

▼次の例は、ワースト50の顧客とベスト50の顧客に手紙を出します：

```
SCAN INDEX ([顧客]合計金額 ; 50 ; <)        ` ワースト50の顧客を取り出す
ORDER BY ([顧客]郵便番号 ; >)               ` 郵便番号順にソート
OUTPUT FORM ([顧客] ; "奮起の手紙")
PRINT SELECTION ([顧客])                    ` 手紙を印刷
` ベスト50の顧客
SCAN INDEX ([顧客]合計金額 ; 50 ; >)        ` ベスト50の顧客を取り出す
ORDER BY ([顧客]郵便番号 ; >)               ` 郵便番号順にソート
OUTPUT FORM ([顧客] ; "感謝の手紙")
PRINT SELECTION ([顧客])                    ` 手紙を印刷
```

REDUCE SELECTION

REDUCE SELECTION (テーブル ; レコード数)

引数	タイプ	説明
テーブル	テーブル	セレクションを抽出するテーブル
レコード数	数値	返すレコードの数

REDUCE SELECTIONコマンドは、＜テーブル＞から新しいレコードのセレクションを作成します。このコマンドは、＜テーブル＞のカレントセレクションから最初の＜レコード数＞個のレコードを返します。**REDUCE SELECTION**コマンドは、＜テーブル＞のカレントセレクションに適用されます。**REDUCE SELECTION**コマンドは＜テーブル＞のカレントセレクションを変更し、新しいセレクションの先頭のレコードをカレントレコードにします。

ONE RECORD SELECT

ONE RECORD SELECT ({テーブル})

引数	タイプ	説明
テーブル	テーブル	選択するレコードの属するテーブル

ONE RECORD SELECTコマンドは、＜テーブル＞のカレントセレクション中のカレントレコードでないレコードをカレントレコードに割り当てます。カレントセレクションにレコードが存在しない場合には、**ONE RECORD SELECT**コマンドは何も行いません。

第44章の**POP RECORD**コマンドの例を参照してください。

セットコマンド　48

この章では、「ルーチン」エディタの「Sets」テーマ内にあるセットコマンドについて説明します。セットを使用すれば、レコードの選択をさらに迅速かつ効率的に行えるようになります。作成したセットに対して、カレントセレクションとのリレート、保存、読み込み、消去の処理が行えるだけでなく、4th Dimensionは3つの基本機能をセットに与えています。

- 集合交差 (交わり、**INTERSECTION**)
- 集合結合 (結び、**UNION**)
- 集合差異 (違い、**DIFFERENCE**)

この章のコマンドは、セットの作成、管理、そして消去について記述します：

CREATE EMPTY SET
CREATE SET
USE SET
ADD TO SET
REMOVE FROM SET
CLEAR SET
DIFFERENCE
INTERSECTION
UNION
COPY SET
Is in set
Records in set
SAVE SET
LOAD SET

セットとカレントセレクション

セットは、セレクション内のレコードを簡潔に表現します。セットの概念は、カレントセレクションと密接な関連があります。

セットは、一般的に次のような場合に使用します：

■ 順序はともかく、セレクションの保存と復元を行う場合
■ ユーザが画面上で作成したセレクションにアクセスする場合
■ セレクション間で論理演算を行う場合

カレントセレクションは、現在選択されているレコードで構成されたリストまたは表です。リストは、メモリ上に存在します。現在選択されているレコードのみがリストに含まれます。セレクションは、実際にレコードを含んでいるわけではありません。レコードに対する参照リストを保持しているだけです。レコードに対する各々の参照リストは、1レコードに対してメモリを4バイト使用します。テーブルに対して作業を行う場合にも、カレントセレクションのレコードに対してのみ作業を行います。また、セレクションをソートした場合でも、実際にはポインタのリストがソートされるだけです。カレントセレクションは、1つのプロセス内の各テーブルに対して1つしか存在しません。

カレントセレクションと同様に、セットはセレクションを表現します。セットは、各レコードを非常に簡潔に表現します。1レコードに対してメモリを1ビット(1/8バイト)使用します。コンピュータは、ビットに対する演算を非常に高速に行いますので、セットに対して演算を行った場合には、処理が高速になります。セットは、セット内にレコードを含んでいるかどうかに関係なく、テーブル上に存在するすべてのレコードに対して1ビットずつ使用します。実際、1または 0の各ビットはレコードがセット内にあるかどうかに関係なくレコードに依存します。

セットはRAMスペースの面から見ると、非常に経済的です。テーブルのレコード件数を8で割れば、そのテーブルのセットサイズをバイト数で求めることができます。例えば、10,000件のレコードを持つテーブルに対してセットを作成すれば、セットはRAMを1,250バイト(約1.2K)使用します。

各テーブルに対してたくさんのセットを持つことができます。またデータベースとは別にディスクに保存することもできます。セットに属するレコードを変更するには、最初にセットをカレントセレクションとして使用し、それから1つまたは複数のレコードを修正します。インタープロセスセットの名前はデータベース内で重複してはいけません。

セットは、ソートした順番には決してなりません。レコードは、セットに含まれるか含まれないかだけを示します。これに対して、命名セレクションはソートの順番を保持することができますが、多くのメモリを必要とします。命名セレクションに関する詳細は、第34章を参照してください。

セットは、セットが作成された時点のカレントレコードを“記憶”しています。

次の表は、カレントセレクションとセットの概念について比較したものです：

比較項目	カレントセレクション	セット
テーブルに対して持つことのできる数	1	0～多数
ソート	○	×
ディスクへの保存	×	○
1レコード当たりのＲＡＭ消費量	4バイト	1/8バイト
論理演算による関係づけ	×	○
カレントレコードを含む	○	○(セットの作成時点)

セットは、それを作成したテーブルに属します。セットの演算は、同じテーブルに属するセット間でのみ行うことができます。

セットは、実在するデータとは別に存在します。これは、テーブルを更新を行した後では、セットが正確でなくなる可能性があることを意味します。セットが不正確になる可能性のある処理は数多くあります。例えば、すべての東京出身の人でセットを作成した後でその中の1つのレコードを大阪出身に修正しても、セットは更新されません。これは、セットがレコードのセレクションを表現しているに過ぎないためです。レコードを削除した後で新しいレコードを追加した場合には、セットは新しいレコードを含み、元のレコードを含まなくなります。セットは、その対応するセレクションが更新されていない場合にのみ正確なものであることが保証されます。

プロセスセットとインタープロセスセット

3種類のセットを持つことができます：

■**「プロセスセット」：**プロセスセットは、それを作成したプロセス内でのみアクセスすることができます。“ UserSet ”と“ LockedSet ”もプロセスセットです。プロセスセットは、プロセスが終了しだい消去されます。プロセスセットは、その名前に特別な接頭辞を必要としません。

■**「インタープロセスセット」：**インタープロセスセットを作成するには、その名前の前に“ <> ”を付けます。インタープロセスセットは、すべてのプロセスで共用されます。

```
USE SET (<>MySet)
```

■**「ローカルセット／クライアントセット」：**このセットは、バージョン6から新たに追加されました。ローカルセット／クライアントセットの名前の先頭には、ドル記号 ($)が付きます。

ローカルセット／クライアントセット

ローカルセット／クライアントセットに関する詳細は、バージョン6の4D Serverがリリースされる際にドキュメント化されます。

セットとトランザクション

セットは、トランザクショの内部で作成されます。トランザクションの内部で作成されたレコードのセット、およびトランザクションの外部で修正されたり、作成されたりしたレコードのセットを作成することができます。トランザクションを終了する際、特にトランザクション処理を取り消した場合、セットはレコードを正確に表示できないのでトランザクション中に作成したセットを消去する必要があります。

セットの例

▼ 次の例は、重複するレコードをテーブルから削除します。このテーブルは従業員に関する情報を含んでいます。**For**ループは、カレントレコードと1つ前のレコードの内容を比較する処理を、すべてのレコードに対して行います。名前、住所、郵便番号がすべて一致する場合には、そのレコードをセットに追加します。ループが終了したところでセットをカレントセレクションにし、カレントセレクションを削除します：

```
CREATE EMPTY SET ([従業員] ; "重複")
   ` 重複レコードを格納する空のセットを定義
ALL RECORDS ([従業員])
   ` すべてのレコードを選択
   ` 重複データを探すために
   ` 郵便番号,住所,名前でソート
ORDER BY ([従業員];[従業員]郵便番号 ; >; [従業員]住所 ; > ; [従業員]名前 ; >)
$名前:=[従業員]名前                    ` フィールドの内容を保管する変数を
$住所:=[従業員]住所                    ` 1番目のレコードの値で初期化する
$郵便番号:=[従業員]郵便番号
NEXT RECORD ([従業員])                 ` 2番目のレコードから比較を開始する
For ($i ; 2 ; Records in file ([従業員]))
   ` 2番目のレコードから最後のレコードまで繰り返す
   ` 郵便番号、住所、名前がすべて一つ前のレコードと一致したら重複レコード
  If (([従業員]名前=$名前) & ([従業員]住所=$住所) & ([従業員]郵便番号=$郵便番号))
    ADD TO SET ([従業員] ; "重複")
    ` セットにカレントレコード(重複レコード)を追加する
  Else
    $名前:=[従業員]名前                ` 郵便番号,住所,名前を次のレコードと
    $住所:=[従業員]住所                ` 比較するために変数に格納する
    $郵便番号:=[従業員]郵便番号
  End if
  NEXT RECORD ([従業員])               ` 次のレコードに移動する
End for
USE SET ("重複")
   ` セット"重複"をカレントセレクションにする
DELETE SELECTION ([従業員])            ` 重複レコードをすべて削除する
CLEAR SET ("重複")                     ` セットをメモリ上から消去する
```

メソッドの最後でセットを削除する代わりに、セット内のレコードを画面上に表示するか、あるいは印刷すれば、さらに詳細な比較を行うことができます。

システムセット：UserSet

4th Dimensionは、“ UserSet ”というシステムセットを持っています。UserSetはユーザによって画面上で最後に選択されたセレクションを自動的に保持します。従って、**MODIFY SELECTION**コマンドまたは**DISPLAY SELECTION**コマンドでセレクションを表示し、ユーザはそれから必要なレコードを選択します。その選択結果でセットまたはセレクションを作成します。

UserSetは1つのプロセスに対して1つしかありません。また、各テーブルに固有のUserSetを持つことはできません。UserSetを持つことができるのは、その時点でセレクションを表示しているテーブルのみです。

▼ 次のメソッドは、レコードの表示方法を示し、ユーザにレコードを選択してもらい、UserSetを使用してその選択されたレコードを表示します：

```
` すべてのレコードを表示し、ユーザにいくつかのレコードを選択してもらい、
` それをUserSetを使用してカレントセレクションにする
OUTPUT FORM ([従業員] ; "表示")          ` 出力フォーム設定
ALL RECORDS ([従業員])                   ` すべての従業員を選択
ALERT ("コマンドキーを押しながらクリックして従業員を選択してください。")
DISPLAY SELECTION ([従業員])             ` 従業員を表示する
USE SET ("UserSet")                      ` 選択された従業員を使用する。
ALERT ("次の従業員が選択されました。")
DISPLAY SELECTION ([従業員])             ` 選択された従業員を表示する
```

注：UserSetを取得するには、DISPLAY SELECTIONコマンドまたはMODIFY SELECTIONコマンドのどちらかを実行する必要があります。

システムセット：LockedSet

APPLY TO SELECTIONコマンドと**DELETE SELECTION**コマンドは、マルチプロセス環境で使用する場合に、“ LockedSet ”という名前のセットを作成します。LockedSetは、コマンド操作中にロックされたレコードを示します。

CREATE EMPTY SET

CREATE EMPTY SET ({テーブル} ; セット)

引数	タイプ	説明
テーブル	テーブル	新しい空のセットを作成するテーブル
セット	文字列	作成する空のセットの名前

CREATE EMPTY SETコマンドは、＜テーブル＞に対して新しい空のセットを、＜セット＞を作成します。**ADD TO SET**コマンドを使って、このセットにレコードを追加することができます。既に同じ名前のセットが存在している場合には、そのセットを消去して新しい空のセットに置き換えます。

▼ 次の例は、新しいセットを作成し、それをシステムセット“ UserSet ”とマージ(**UNION**コマンドを使用)し、UserSetの内容を新しいセットに格納します：

```
CREATE EMPTY SET ([従業員] ; "格納セット")      ` 新しいセットを作成
UNION ("UserSet" ; "格納セット" ; "格納セット")  ` 2つのセットをマージする
```

CREATE SET

CREATE SET ({テーブル} ; セット)

引数	タイプ	説明
テーブル	テーブル	新しいセットを作成するためのセレクションの属するテーブル
セット	文字列	作成するセットの名前

CREATE SETコマンドは、＜テーブル＞に対して新しいセット＜セット＞を作成し、＜セット＞にカレントセレクションの内容を代入します。そのテーブルのカレントレコードポインタは、＜セット＞に格納されます。＜セット＞に対して**USE SET**コマンドを使用すると、カレントセレクションとカレントレコードが復元されます。すべてのセットに対してソート順序は適用されません。＜セット＞には、デフォルトの順序が適用されるだけです。既に同じ名前のセットが存在している場合には、そのセットを消去し新しいセットに置き換えます。

▼次の例は、検索を行った後で新しいセットを作成し、それをディスクに保存します：

```
QUERY ([従業員])                              ` ユーザによる検索
CREATE SET ([従業員] ; "検索セット")           ` 新しいセットを作成
SAVE SET ("検索セット" ; "検索結果")           ` セットをディスクに保存
```

USE SET

USE SET (セット)

引数	タイプ	説明
セット	文字列	作成するセットの名前

USE SETコマンドは、セット内のレコードを＜セット＞の属するテーブルのカレントセレクションにします。

セットを作成する場合、カレントレコードはそのセットによって“記憶”されています。**USE SET**コマンドはセット上のカレントレコードの位置を復元し、そのレコードを新しいカレントレコードにします。**USE SET**コマンドを実行する前にこのレコードを削除すると、4th Dimensionはセットの先頭のレコードをカレントレコードに設定します。セットに対して**INTERSECTION**コマンド、**UNION**コマンド、**ADD TO SET**コマンドを実行した場合には、カレントレコードは再設定されます。カレントレコードの位置を含まないセットを作成した場合にも、**USE SET**コマンドはセットの先頭のレコードをカレントレコードに設定します。

警告：セットは、そのセットが作成された時点のセレクションを表現しているという点に注意してください。セットに対応するレコードが更新されると、セットは正確なものでなくなります。従って、ディスクに保存するセットは、頻繁に更新されることのないレコードで構成するべきです。そうでない場合には、ディスクに保存したセットをデータベース中のテーブルに復元すると、最新の状態のレコードを失う可能性があります。

▼ 次の例は、**LOAD SET**コマンドを使用して、所在地が“東京都千代田区の会社”のセットを復元し、その復元したセットをカレントセレクションにします：

```
` セットをメモリ上に復元する
LOAD SET ([会社] ; "東京都千代田区" ; "東京都千代田区のセット")
USE SET ("東京都千代田区")        ` カレントセレクションを東京都千代田区にする
CLEAR SET ("東京都千代田区")      ` セットをメモリから消去
```

ADD TO SET

ADD TO SET ({テーブル} ; セット)

引数	タイプ	説明
テーブル	テーブル	セットに追加するレコードの属するテーブル
セット	文字列	レコードを追加するセットの名前

ADD TO SETコマンドは、＜セット＞に＜テーブル＞のカレントレコードを追加します。ここで使用するセットは、既に作成されているものでなければなりません。存在しないセット名を指定した場合は、エラーになります。＜テーブル＞にカレントレコードが存在しない場合には、**ADD TO SET**コマンドは何も行いません。

前述の「セットの例」を参照してください。

CLEAR SET

CLEAR SET (セット)

引数	タイプ	説明
セット	文字列	メモリから消去するセットの名前

CLEAR SETコマンドは、メモリから＜セット＞を消去し、＜セット＞の占有していたメモリを解放します。**CLEAR SET**コマンドはテーブル、セレクション、レコードには影響を与えません。セットは、消去する前に**SAVE SET**コマンドを使用して保存することができます。セットはメモリを使用するので、必要のないセットは消去してください。

USE SETコマンドの例を参照してください。

Is in set

Is in set (セット) → ブール

引数	タイプ	説明
セット	文字列	調べるセット

Is in set関数は、＜セット＞の属するテーブルのカレントレコードが＜セット＞に含まれているかどうかを調べます。**Is in set**関数は、カレントレコードが＜セット＞に含まれていれば“ True(真) ”を返し、含まれていなければ“ False(偽) ”を返します。

▼ 次の例は、ボタンのオブジェクトメソッドです。これは、現在表示されているレコードが“ お得意様 ”のセットに含まれているかどうかを調べます：

```
If (is in set ("お得意様"))                    ` お得意様かどうか調べる
  ALERT ("このお客様は、お得意様です。")
Else
  ALERT ("このお客様は、お得意様ではありません。")
End if
```

Records in set

Records in set (セット) → 数値

引数	タイプ	説明
セット	文字列	調べるセット

Records in set関数は、＜セット＞に含まれるレコードの数を返します。＜セット＞が存在しない場合、または＜セット＞にレコードがない場合には0を返します。

▼次の例は、全顧客の中に占めるお得意様の割合をアラートボックスに表示します：

```
` お得意様の割合を計算
$パーセント:=(Records in set ("お得意様") / Records in file ([顧客])) * 100
` お得意様の割合をパーセント表示
ALERT (String ($パーセント ; "##0%") + "がお得意様です。")
```

SAVE SET

SAVE SET (セット ; 文書)

引数	タイプ	説明
セット	文字列	保存するセットの名前
文書	文字列	ディスクに保存する文書ファイルの名前

SAVE SETコマンドは、＜文書＞で指定した名前の文書ファイルとして＜セット＞をディスクに保存します。

＜文書＞は、セットと同じ名前である必要はありません。＜文書＞に対して空の文字列(ヌル)を指定すると、「ファイル作成」ダイアログボックスが表示されます。ユーザは、ここでファイルの名前を入力することができます。保存したセットは、**LOAD SET**コマンドを使用して復元することができます。

「ファイル作成」ダイアログボックスで「キャンセル」ボタンをクリックした場合や保存処理中にエラーが発生した場合には、システム関数OKに 0が代入されます。それ以外の場合には1がセットされます。

SAVE SETコマンドは、時間のかかる検索の結果をディスクに保存するためによく使用されます。

警告：セットは、そのセットが作成された時点のセレクションを表現しているということに注意してください。セットに対応するレコードが更新されると、セットは正確なものではなくなります。従って、ディスクに保存するセットは、頻繁に更新されることのないレコードで構成するべきです。そうでない場合には、ディスクに保存したセットをデータベース中のテーブルに復元すると、最新の状態のレコードを失う可能性があります。また、セットはフィールド値を保存しないということに注意してください。

▼ 次の例は、ユーザがセットを含んだファイル名を入力するための「ファイル作成」ダイアログボックスを表示します：

```
SAVE SET ("検索セット" ; "")
```

▼ 次の例は、**SAVE SET**コマンドを使用してシーケンシャルな検索の結果を保存します。検索条件は“ 東京の千代田区にある会社すべて ”です。結果のセットは、ユーザが指定した文書ファイルに保存されます：

```
QUERY([会社] ; [会社]市区町村="千代田区" ; *)     ` 第1の検索...
QUERY ([会社] ;&[会社]都道府県="東京都")          ` 第2の検索
CREATE SET ([会社] ; "東京都千代田区")` カレントセレクションからセットを作成
SAVE SET ("東京都千代田区" ; "")          ` ユーザの名付けたファイルに保存
CLEAR SET ("東京都千代田区")                    ` セットをメモリから消去
```

4D Server：マルチユーザデータベース上では、セットの示すレコードを他のユーザが変更してしまうと、そのセットが正しいものではなくなる可能性があるため、SAVE SETコマンドの使用はあまりお勧めできません。4D Serverを使うと、セットはローカルに保存されます。

LOAD SET

LOAD SET ({テーブル} ; セット ; 文書)

引数	タイプ	説明
テーブル	テーブル	セットの属するテーブル
セット	文字列	メモリに復元するセット
文書	文字列	ディスクに保存されたセットの文書ファイル

LOAD SETコマンドは、**SAVE SET**コマンドでディスクに保存した＜文書＞からセットをメモリに復元します。

＜文書＞に格納されたセットは、＜テーブル＞から作成されてなければいけません。メモリ内で作成されたセットは既に同じセットが存在すると上書きされます。

引数＜文書＞は、セットを保存した文書ファイルの名前です。文書はセットと同じ名前である必要はありません。＜文書＞に対して空の文字列(ヌルストリング)を指定すると、「ファイルを開く」ダイアログボックスが表示されます。ユーザは、ここで復元するセットを選択することができます。

「ファイルを開く」ダイアログボックスで「キャンセル」ボタンをクリックした場合や保存処理中にエラーが発生した場合はシステム関数OKに 0が代入されます。それ以外の場合には1が代入されます。

警告：セットは、そのセットが作成された時点のセレクションを表現しているという点に注意してください。セットに対応するレコードが更新されると、セットは正確なものでなくなります。従って、ディスクに保存するセットは、頻繁に更新されることのないレコードで構成するべきです。そうでない場合には、ディスクに保存したセットをデータベース中のテーブルに復元すると、最新の状態のレコードを失う可能性があります。

▼ 次の例は、**LOAD SET**コマンドを使用して“ 東京都千代田区 ”のセットを復元します：

```
` セットをメモリに復元
LOAD SET ([会社] ; "東京都千代田区" ; "東京都千代田保存")
USE SET ("東京都千代田区") ` セット"東京都千代田"をカレントセレクションにする
CLEAR SET ("東京都千代田区")    ` セットをメモリから消去
```

4D Server：マルチユーザデータベース上では、セットの示すレコードを他のユーザが変更してしまうと、そのセットが正しいものではなくなる可能性がありますので、LOAD SETコマンドを使用することはあまりお勧めできません。

DIFFERENCE

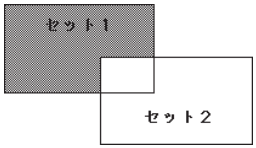
DIFFERENCE (セット1 ; セット2 ; 結果セット)

引数	タイプ	説明
セット1	文字列	オリジナルセット
セット2	文字列	排除するセット
結果セット	文字列	結果を格納するセット

DIFFERENCEコマンドは、＜セット1 ＞と＜セット2＞を比較(集合差異演算)し、＜セット2＞と一致しない＜セット1 ＞のレコードだけを＜結果セット＞に格納します。つまり、＜セット1＞にだけが存在し、＜セット2＞には存在しないレコードのみを＜結果セット＞に格納します。次の表に、集合差異演算の処理で考えられるすべての組み合わせを示します。

セット1	セット2	結果セット
○	×	○
○	○	×
×	○	×
×	×	×

次の図に、集合差異演算の処理結果を図で示します。塗りつぶした部分が結果のセットです。



＜結果セット＞は、**DIFFERENCE**コマンドで作成されます。＜結果セット＞と同じ名前のセット(セット1、セット2も含めて)が既に存在していた場合には、＜結果セット＞に置き換えられます。＜セット1＞と＜セット2＞は同じテーブルに属していなければなりません。＜結果セット＞も＜セット1＞と＜セット2＞と同じテーブルに属します。

▼ 次の例は、表示したセレクションからユーザが選択したレコードを排除します。このレコードリストは、次のステートメントで画面に表示されます：

```
DISPLAY SELECTION ([顧客])                    ` 顧客のリストを表示
```

レコードリストの下部には、オブジェクトメソッド付きのボタンがあります。このオブジェクトメソッドはユーザが選択したレコード(UserSet)を排除し、新しいセットを表示します：

```
CREATE SET ([顧客] ; "カレント")      ` カレントセレクションからセットを作成
` “ UserSet ” と一致するレコードを排除
DIFFERENCE ("カレント" ; "UserSet" ; "カレント")
USE SET ("カレント")                          ` 新しいセットを使用
CLEAR SET ("カレント")                        ` セットをメモリから消去
```

INTERSECTION

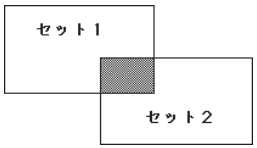
INTERSECTION (セット1 ; セット2 ; 結果セット)

引数	タイプ	説明
セット1	文字列	第1のセット
セット2	文字列	第2のセット
結果セット	文字列	結果を格納するセット

INTERSECTIONコマンドは、＜セット1＞と＜セット2＞を比較(集合交差演算)し、＜セット1＞と＜セット2＞の両方に存在するレコードだけを選択し、＜結果セット＞に格納します。下表に、集合交差演算の処理で考えられる全ての組み合わせを示します。

セット1	セット2	結果セット
○	×	×
○	○	○
×	○	×
×	×	×

次の図に、集合交差演算の処理結果を図で示します。塗りつぶした部分が結果のセットです。



＜結果セット＞は、**INTERSECTION**コマンドで作成されます。＜結果セット＞と同じ名前のセット(セット1、セット2も含めて)が既に存在する場合には、＜結果セット＞に置き換わります。＜セット1＞と＜セット2＞は同じテーブルに属していなければなりません。＜結果セット＞も＜セット1＞と＜セット2＞と同じテーブルに属します。

▼ 次の例は、“ 藤上 ” と “ 吉野 ” という2人の販売担当者が重複して担当する顧客を検索します。販売担当者は、各自の顧客を表すセット “ 藤上 ” と “ 吉野 ” を持っています：

```
INTERSECTION ("藤上" ; "吉野" ; "重複")  ` 重複して担当する顧客のセットを作成
USE SET ("重複")                         ` セットを使用
CLEAR SET ("重複")                       ` セットをメモリから消去
DISPLAY SELECTION ([顧客])               ` 2人が重複して担当する顧客のリストを表示
```

UNION

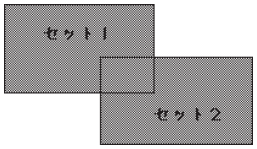
UNION (セット1 ; セット2 ; 結果セット)

引数	タイプ	説明
セット1	文字列	第1のセット
セット2	文字列	第2のセット
結果セット	文字列	結果を格納するセット

UNIONコマンドは、＜セット1＞と＜セット2＞を比較(集合結合演算)し、＜セット1＞と＜セット2＞に含まれるすべてのレコードを＜結果セット＞に格納します。下表に、集合結合演算の処理で考えられるすべての組み合わせを示します。

セット1	セット2	結果セット
○	×	○
○	○	○
×	○	○
×	×	×

次の図に、集合結合演算の処理結果を図で示します。塗りつぶした箇所が結果セットの部分です。



＜結果セット＞は、**UNION**コマンドで作成されます。＜結果セット＞と同じ名前のセット(セット1、セット2も含めて)がすでに存在する場合には、＜結果セット＞に置き換えられます。＜セット1＞と＜セット2＞は同じテーブルに属していなければなりません。＜結果セット＞も＜セット1＞、＜セット2＞と同じテーブルに属します。＜結果セット＞のカレントレコードは、＜セット1＞からのカレントレコードです。

▼ 次の例は、“ お得意様 ” のセットにレコードを追加します。すべてのレコードを表示した後で、“ お得意様 ” のセットをディスクから復元します。そして、ユーザが選択したすべてのレコード(UserSet)を “ お得意様 ” のセットに追加します。最後に新しい “ お得意様 ” のセットをディスクに保存します：

```
ALL RECORDS ([顧客])                          ` すべての顧客を選択
DISPLAY SELECTION ([顧客])                    ` すべての顧客のリストを表示
LOAD SET ("お得意様" ; "お得意様格納")          ` お得意様のセットを復元
UNION ("お得意様" ; "UserSet" ; "お得意様")     ` 新しい顧客をセットに追加
SAVE SET ("お得意様" ; "お得意様格納")          ` お得意様のセットを保存
```

REMOVE FROM SET

REMOVE FROM SET ({テーブル ;} セット)

引数	タイプ		説明
テーブル	テーブル	→	カレントレコードのテーブル 省略した場合は、デフォルトテーブルの
セット	文字列	→	カレントレコードを削除するための セットの名前

REMOVE FROM SETコマンドは、引数＜セット＞から＜テーブル＞のカレントレコードを削除します。セットは常に存在していなければならず、存在していない場合には、エラーが発生します。オプション引数＜テーブル＞に対するカレントレコードがない場合には、**REMOVE FROM SET**コマンドは無効です。

参照：ADD TO SET

COPY SET

COPY SET (コピー元 ; コピー先)

引数	タイプ		説明
コピー元	文字列	→	コピー元のソースセットの名前
コピー先	文字列	→	コピー先のセットの名前

COPY SETコマンドは、引数＜コピー先＞セットの中に＜コピー元＞セットの内容をコピーします。

両方のセットとも、プロセスセット、インタープロセスセット、またはローカルセットにすることができます。

4D Server：ローカルセットはクライアントマシン上で管理されますが、クライアント／サーバにおいて、インタープロセスセットとプロセスセットはサーバマシン上で管理されます。COPY SETコマンドは2つのマシン間でセットをコピーすることができます。4D Serverとセットに関する詳細は、『4D Serverリファレンス』マニュアルを参照してください。

例：

1. 次の例は、クライアント／サーバにおいて、クライアントマシン上で管理されるローカルセット“ $SetA ”をサーバマシン上で管理されるプロセスセット“ SetB ”にコピーします：

```
COPY SET("$SetA" ; "SetB")
```

2. 次の例は、クライアント／サーバにおいて、サーバマシン上で管理されるプロセスセット“ SetA ”をクライアント／サーバ上で管理されるローカルセット“ $SetB ”にコピーします：

```
COPY SET("SetA" ; "$SetB")
```

文字列関数　49

この章では、「ルーチン」エディタの「String」テーマ内にある文字列関数について説明します。この章では、文字列に対して機能する関数と文字参照記号について説明します。

Ascii
Change string
Char
Convert case
Delete string
Insert String
Length
Lowercase
Num
Position
Replace string
String
Substring
Uppercase
Mac to Win
Win to Mac
Mac to ISO
ISO to Mac

文字参照記号

文字列[[文字位置]] → 文字列 (1バイト)

引数	タイプ	説明
文字列	文字列	この文字列から文字を取り出す
文字位置	数値	文字を取り出す位置（バイト）

文字参照記号([[...]])は、＜文字列＞内の1文字(1バイト)を参照するために使用します。文字の位置を示す＜文字位置＞は、文字参照記号で囲んで指定します。

＜文字位置＞で指定した場所の文字を返します。

文字参照記号が代入演算子の左側にある場合は、文字列内の指定した位置に文字を代入します。次の例は、Nameの最初の文字を大文字にします。

```
Name[[1]]:=Uppercase (Name[[1]])
```

文字列の長さよりも大きな数値を指定した場合の結果は保証しません。

▼ 文字参照記号の使用例を次に示します：

```
v結果:="abcd"[[3]]                    ` v結果に"c"を代入
v結果:=Last Name[[1]]                 ` v結果にLast Nameの先頭の文字を代入
v結果:=City[[$i]]                     ` v結果にCityの$i番目の文字を代入
```

次のサブルーチンは、文字列内の各単語の先頭の文字を大文字に変換する関数です：

```
$0:=$1                                ` 返す文字列をコピー
$0[[1]]:=Uppercase ($0[[1]])          ` 先頭は必ず大文字で始まる
For ($i; 1 ; Length ($0) - 1)         `先頭文字以外のすべての文字についてループ
  If (Position ($0[[$i]] ; " !&()-{};:;<>?/,.=+*") > 0)      ` 文字でなければ…
    $0[[$i+1]]:=Uppercase($0[[$i+1]])  ` 次に現れた文字を大文字にする
  End if
End for
```

Length

Length(文字列) → 数値

引数	タイプ	説明
文字列	文字列	長さを知りたい文字列

Length関数は、＜文字列＞の長さをバイト数で返します。

▼ **Length**関数の使用例を次に示します。結果を変数 " v結果 " に代入します。コメントは、変数 " v結果 " に代入される値についての説明です：

```
v結果:=Length ("Japan")          ` v結果に5を代入
v結果:=Length ("Citizen")        ` v結果に7を代入
```

Substring

Substring (文字列 ; 先頭文字位置 ; {文字数}) → 文字列

引数	タイプ	説明
文字列	文字列	この文字列から部分文字列(substring)を得る
先頭文字位置	数値	取り出す文字列の最初の文字の位置(バイト)
文字数	数値	取り出す文字列の長さ(バイト)

Substring関数は、＜先頭文字位置＞と＜文字数＞で指定した部分文字列(Substring)を＜文字列＞から取り出して返します。＜先頭文字位置＞には文字列の中で取り出す文字列の最初の文字の位置を指定し、＜文字数＞には取り出す文字列の長さを指定します。

＜先頭文字位置＞と＜文字数＞の和が、文字列自身の文字数よりも大きい場合や＜文字数＞を指定しない場合には、先頭文字位置以降の文字列をすべて取り出します。＜先頭文字位置＞が文字列の長さより大きいと、**Substring**関数は空の文字列("")を返します。

▼ **Substring**関数の使用例を次に示します。結果を変数 " v結果 " に代入します。コメントは、変数 " v結果 " に代入される内容についての説明です：

```
v結果:=Substring ("62.04.08" ; 4 ; 2)        ` v結果に"04"を代入
v結果:=Substring ("Emergency" ; 1 ; 6)       ` v結果に"Emerge"を代入
v結果:=Substring (var ; 2)                   ` v結果に先頭の文字以外のすべての文字を代入
```

Position

Position (検索文字列 ; 文字列) → 数値

引数	タイプ	説明
検索文字列	文字列	探す文字列
文字列	文字列	元の文字列

Position関数は、＜文字列＞の中で＜検索文字列＞が最初に現われる位置(バイト)を返します。

＜検索文字列＞が見つからない場合は、**Position**関数はゼロ(0)を返します。

＜検索文字列＞が見つかると、文字列の中に検索文字列が最初に現れた文字位置を返します。

空の文字列に対して空の検索文字列を指定すると、**Position**関数はゼロ(0)を返します。

注：この“ @ ”記号は、4th Dimension全般ではワイルドカード検索用に使用される文字です。しかし、このPosition関数では、“ @ ”記号は文字列として扱われます。例えば、Position関数は“ abc@ ”を単なる文字として認識します。

▼ **Position**関数の使用例を次に示します。結果を変数“ v結果 ”に代入します。コメントは、変数“ v結果 ”に代入される値についての説明です：

```
v結果:=Position ("ll" ; "Willow") ` v結果に3を代入
v結果:=Position (変数1 ; 変数2)` v結果に変数2の中で変数1が最初に現れる位置を代入
```

Change string

Change string (元の文字列 ; 修正文字列 ; 修正位置) → 文字列

引数	タイプ	説明
元の文字列	文字列	元の文字列
修正文字列	文字列	新しい文字列
修正位置	数値	修正開始位置

Change string関数は、＜元の文字列＞の中の文字グループを修正したものを返します。＜修正位置＞で指定された位置から、＜修正文字列＞で＜元の文字列＞を上書きします。

＜修正文字列＞が空の文字列("")の場合は、**Change string**関数は＜元の文字列＞を加工しないで返します。

常に＜元の文字列＞と同じ長さの文字列を返します。＜修正位置＞が＜元の文字列＞以下の長さの場合や＜元の文字列＞以上の長さの場合には、**Change string**関数は＜元の文字列＞を返します。

Change string関数は、文字を挿入しないで上書きするという点が**Insert string**関数とは異なります。

▼ **Change string**関数の使用例を次に示します。結果を変数 “ v結果 ” に代入します：

```
v結果:=Change string ("Acme" ; "CME" ; 2)        ` v結果に"ACME"を代入
v結果:=Change string ("November" ; "Dec" ; 1)    ` v結果に"December"を代入
```

Insert string

Insert string (元の文字列 ; 挿入文字列 ; 挿入位置) → 文字列

引数	タイプ	説明
元の文字列	文字列	元の文字列
挿入文字列	文字列	新しい文字列
挿入位置	数値	挿入開始位置

Insert string関数は、＜元の文字列＞に文字列を挿入したものを返します。**Insert string**関数は、＜挿入位置＞で指定された位置の前に、＜挿入文字列＞を挿入します。

＜挿入文字列＞が空の文字列("")であれば、**Insert string**関数は＜元の文字列＞を加工しないで返します。

＜挿入位置＞が、＜元の文字列＞の長さよりも大きい場合は、＜挿入文字列＞を＜元の文字列＞の後ろに追加します。＜挿入位置＞が1よりも小さい場合には、＜挿入文字列＞を＜元の文字列＞の前に挿入します。

Insert string関数は、文字を上書きしないで挿入するという点が**Change string**関数とは異なります。

▼ **Insert string**関数の使用例を次に示します。結果を変数“ v結果 ”に代入します：

```
v結果:=Insert string ("The tree" ; "green" ; 4) ` v結果に"The green tree"を代入
v結果:=Insert string ("Shut" ; "o" ; 3)         ` v結果に"Shout"を代入
v結果:=Insert string ("Indention" ; "ta" ; 6)   ` v結果に"Indentation"を代入
```

Delete string

Delete string (元の文字列 ; 削除位置 ; 文字数) → 文字列

引数	タイプ	説明
元の文字列	文字列	元の文字列
削除位置	数値	削除を開始する文字の位置(バイト)
文字数	数値	削除する文字数(バイト)

Delete string関数は、＜削除位置＞から＜文字数＞分の文字を＜元の文字列＞から削除した文字列を返します。

Delete string関数は、次のような場合に＜元の文字列＞と同じ文字列を返します：

- ＜元の文字列＞が空の文字列の場合
- ＜削除位置＞がゼロ(0)か、ゼロより小さい場合
- ＜削除位置＞が＜元の文字列＞の長さより大きい場合
- ＜文字数＞がゼロ(0)の場合

＜削除位置＞と＜文字数＞の和が＜元の文字列＞の長さと等しいかまたは大きい場合は、文字列の最後まで文字を削除します。

▼ **Delete string**関数の使用例を次に示します。結果を変数 “ v結果 ” に代入します：

```
v結果:=Delete string ("Lamborghini" ; 6 ; 5)   ` v結果に"Lambo"を代入
v結果:=Delete string ("Indentation" ; 6 ; 2)   ` v結果に"Indention"を代入
v結果:=Delete string (var ; 3 ; 32000)         ` v結果にvarの最初の2文字を代入
```

Replace string

Replace string (元の文字列 ; 対象文字列 ; 置き換え文字列 ; {回数}) → 文字列

引数	タイプ	説明
元の文字列	文字列	元の文字列
対象文字列	文字列	置き換えの対象となる文字列
置き換え文字列	文字列	置き換える文字列
回数	数値	置き換える回数

Replace string関数は、＜元の文字列＞に存在するすべての＜対象文字列＞を＜置き換え文字列＞に置き換えます。

＜置き換え文字列＞が空の文字列("")の場合は、**Replace string**関数は＜元の文字列＞の中の＜対象文字列＞をすべて削除します。

＜回数＞を指定した場合、**Replace string**関数は＜元の文字列＞の最初の文字から探して、その回数分だけ＜対象文字列＞で置き換えます。

＜対象文字列＞が空の文字列の場合は、**Replace string**関数は変更前の文字列を返します。＜回数＞が1未満の場合、＜元の文字列＞はすべて置き換えられます。

▼ **Replace string**関数の使用例を次に示します。結果を変数 " v結果 " に代入します。コメントは、変数 " v結果 " に代入される内容についての説明です：

```
v結果:=Replace string ("Willow" ; "ll" ; "d")    ` v結果に"Widow"を代入
v結果:=Replace string ("Shout" ; "o" ; "")       ` v結果に"Shut"を代入
v結果:=Replace string (var ; Char(9) ; ",")      ` varの tabをすべてカンマに置き換える
```

Lowercase

Lowercase (文字列) → 文字列

引数	タイプ	説明
文字列	文字列	小文字に変換する文字列

Lowercase関数は、アルファベット文字列をすべて小文字に変換して返します。

▼ 次の例は、“ Caps ” という名前の関数です。与えられた文字列の最初の文字を大文字にして返すものです。例えば、「Name:=*Caps*("john")」の結果は、「Name="John"」となります。この例は、文字参照記号([[...]])を使用しています：

```
` 関数：Caps(文字列)
$0:=Lowercase ($1)
$0[[1]]:=Uppercase ($1[[1]])
```

Uppercase

Uppercase (文字列) → 文字列

引数	タイプ	説明
文字列	文字列	大文字に変換する文字列

Uppercase関数は、アルファベット文字列をすべて大文字に変換して返します。

上記の **Lowercase**関数の例を参照してください。

String

String (数値 ; {フォーマット}) → 文字列

引数	タイプ	説明
数値	数値	文字列に変換する数値
フォーマット	文字列	変換に使用するフォーマット

String (日付 ; {フォーマット}) → 文字列

引数	タイプ	説明
日付	日付	文字列に変換する日付
フォーマット	数値	フォーマット：1,2,3,4,5,6

String (時間 ; {フォーマット}) → 文字列

引数	タイプ	説明
時間	時間	文字列に変換する時間
フォーマット	数値	フォーマット：1,2,3,4,5

String関数には、3種類の形式があります。これらの形式はすべて、文字列以外のデータを文字列に変換します。

第1の形式は、＜数値＞を文字列に変換します。必要に応じて、＜フォーマット＞を使用し、書式を指定します。フォーマットは、フォームの数値フォーマットと同じです。詳細に関しては、『4th Dimension / 4D First デザインリファレンス』を参照してください。カスタムフォーマットの書式名をパスすることができます。カスタムフォーマットの名前は“|”で始めなければなりません。

第2の形式は、YY.MM.DDのフォーマットで＜日付＞を文字列に変換します。＜フォーマット＞を指定しない場合、YY.MM.DDはデフォルトフォーマットとして用いられます。＜フォーマット＞を指定した場合、次の表に示すフォーマットに従って変換します：

フォーマット	種類	例
1	Y.M.D	97.8.21
2	YYYY年M月D日（X）	1997年 8月 21日（木）
3	YYYY年M月D日X曜日	1997年 8月 21日 木曜日
4	YY.MM.DD	97.08.21
5	Month Date, Year	August 21, 1997
6	Month Date, Year（短表記）	Aug 21, 1997

String関数の第3の形式は、HH:MM:SSのフォーマットで＜時間＞を文字列に変換します。＜フォーマット＞を指定しない場合は、HH:MM:SSはデフォルトフォーマットとして用いられます。＜フォーマット＞を指定した場合には、次の表に示すフォーマットに従って変換します。

フォーマット	種類	例
1	HH:MM:SS	01:02:03
2	HH:MM	01:02
3	H 時 M 分 Y 秒	1時2分3秒
4	H 時 M 分	1時2分
5	H:MM AM / PM	1:02 AM

▼ 次の例は、v数値という数値を円記号(\)でフォーマットした文字列に変換します：

```
v結果:=String (v数値 ; "\###,##0")
```

▼ 次の例は、現在の日付をアラートボックスに表示します：

```
ALERT ("今日は" + String (Current date) + "です。")
```

▼ 次の例は、アラートボックスに現在の時間を表示します：

```
ALERT ("只今" + String (Current time) + "です。")
```

Ascii

Ascii (文字) → 数値

引数	タイプ	説明
文字	文字列	ASCIIコードに変換する文字

Ascii関数は、＜文字＞をASCIIコードに変換して返します。＜文字＞が1文字以上の場合は、**Ascii**関数は最初の文字だけをコードに変換します。**Ascii**関数の逆の変換を行う関数が**Char**関数です。ASCIIコードで示す文字を返します。ASCIIコードに関する詳細は、付録Bを参照してください。

通常、大文字と小文字は同じものとして扱われますが、**Ascii**関数を使用すれば大文字と小文字を区別します。

▼ 次のような場合の結果は " True(真) " になります：

```
("A" = "a")
```

▼ 次のような場合の結果は " False(偽)になります：

```
(Ascii ("A") = Ascii ("a"))
```

▼ 次の例は、文字列の最初の文字AのASCIIコードを返します：

```
GetAsc:=Ascii ("ABC")                    ` GetAscは65を得る
```

Char

Char (ASCIIコード) → 文字列 (1バイト)

引数	タイプ	説明
ASCIIコード	数値	0から255までのASCIIコード

Char関数は、キーボードから入力できないような<ASCIIコード>をプロシージャに書き込むために使用します。次の表に、こうした文字の一部を示します。

コード	ASCII コード	キー
3		Enter (数値キーパッド上の)
9	TAB	Tab
13	キャリッジリターン(CR)	Enter (キーボード上の)

次の文字は、Macintosh上でのみ利用することができます。(このコード16-20は、Chicagoフォントにしか存在しません。)

コード	ASCII コード	キー
16		⌘ - P
17	⌘	⌘ - Q
18	✓	⌘ - R
19	◆	⌘ - S
20		⌘ - T

▼ 次の例は、変数にキャリッジリターンを代入するために**Char**関数を使用します。そしてアラートを表示します。キャリッジリターンは、アラートボックス内で改行を行うために使用します：

```
CR:=Char (13)                    ` 変数CRにキャリッジリターンを代入
ALERT ("従業員："+ String (Records in file ([従業員]))+CR+"OKをクリックしてください。")
```

Num

Num (文字列) → 数値

引数	タイプ	説明
文字列	文字列	数値に変換する文字列

Num (ブール値) → 数値（0または1）

引数	タイプ	説明
ブール値	ブール	数値に変換するブール値

Num関数には、2種類の形式があります。

第1の形式は、＜文字列＞を数値に変換します。

＜文字列＞が数字をまったく含まない場合は、**Num**関数はゼロ(0)を返します。また、英文字と数字が混在する場合も、**Num**関数はゼロ(0)を返します。

ピリオド(.)、ハイフン(-)、e(またはE)の3種類の文字は、**Num**関数に対して特別な意味を持ち、それぞれ数値表現のフォーマット文字とみなされます。

ピリオドは、小数点を意味します。数字の文字列の間に置かなければなりません。

ハイフンは、数値や指数が負であることを意味します。数字の前や指数eの後に配置します。ハイフンが数字の間にあると、文字列は無効になり、**Num**関数はゼロ(0)を返します。例えば、**Num**("123-456")は0になります。また、**Num**(-9)は引数が文字列ではありませんのでエラーになります。

eまたはEがあると、その右側の数字をすべて指数として解釈します。eは、数字の文字列の間に置かなければなりません。**Num**("123e-2")は1.23になります。

Num関数の第2の形式は＜ブール値＞を調べ、0または1を返します。＜ブール値＞が“ False(偽) ”の場合は0を返し、“ True(真) ”の場合は1を返します。

Numの使用例を次に示します。結果を変数“ v結果 ”に代入します。コメントは、変数“ v結果 ”に代入される値についての説明です：

```
v結果:=Num ("ABCD")                    ` v結果に0を代入
v結果:=Num ("A1B2C3")                  ` v結果に0を代入
v結果:=Num ("123")                     ` v結果に123を代入
v結果:=Num ("123.4")                   ` v結果に123.4を代入
v結果:=Num ("-123")                    ` v結果に-123を代入
v結果:=Num ("-123e2")                  ` v結果に-12300を代入
```

▼ 次の例は、顧客の貸越について 0または1を返すものです。文字列を反復する演算子、アスタリスク(*)を使用して顧客に対する評価を[顧客]危険率フィールドに納めます。このフィールドは、顧客の貸越残によって“ 低い ”値または“ 高い ”値が割り当てられます：

```
 ` 顧客の貸越残が1000以下なら危険率は低い
 ` 顧客の貸越残が1000以上なら危険率は高い
[顧客] 危険率:=("低い" * Num ([顧客]貸越残<1000))+("高い" * Num ([顧客]貸越残>=1000))
```

Convert case

Convert case (文字列 ; 変換タイプ ; 対象タイプ) → 文字列

引数	タイプ	説明
文字列	文字列	変換対象となる文字列
変換タイプ	数値	文字列の変換タイプ
対象タイプ	数値	変換対象となる文字のタイプ

「文字列」中の変換対象となる文字タイプを「対象タイプ」で指定し、「変換タイプ」で指定したタイプに変換します。「対象タイプ」の値は、加算することによって、複数の対象を同時に指定することができます。例えば、“ 1バイトのアルファベットと数字と記号 ”および“ 1バイトのカタカナ ”を対象とする場合は、4+16の結果である 20を指定することができます。

以下に漢字Talk 7下で指定できる「変換タイプ」と「対象タイプ」の値を示します。

指定できる変換タイプ

2　1バイトのアルファベットと数字と記号へ変換
3　2バイトのアルファベットと数字と記号へ変換
4　1バイトのカタカナへ変換
5　2バイトのカタカナへ変換
7　2バイトのひらがなへ変換

指定できる対象タイプ

-1	テキスト全体を対象
4	1バイトのアルファベットと記号を対象
8	2バイトのアルファベットと記号を対象
16	1バイトのカタカナを対象
32	2バイトのカタカナを対象
128	2バイトのひらがなを対象

例

v文字:=Convert case ("あいうえおABCD12345+-*/?ふぉーでぃふぁーすと" ; 2 ; -1)

上記のコードを実行すると、
変数"v文字"に「aiueoABCD12345+-*/?fo-dhifa-suto」が返ります。

v文字:=Convert case ("あいうえおあBCD12345+-*/?ふぉーでぃふぁーすと" ; 4 ; -1)

上記のコードを実行すると、
変数"v文字"に「ｱｲｳｴｵｱBCD12345+-*/?ﾌｫｰﾃﾞｨﾌｧｰｽﾄ」が返ります。

v文字:=Convert case ("あいうえおABCD12345+-*/?ふぉーでぃふぁーすと" ; 5 ; -1)

上記のコードを実行すると、
変数"v文字"に「アイウエオアBCD12345+-*/?フォーディファースト」が返ります。

v文字:=Convert case ("ｱｲｳｴｵABCD12345+-*/?fo-dhifa-suto**" ; 7 ; 4)**

上記のコードを実行すると、
変数"v文字"に「ｱｲｳｴｵあBCD12345+-*/?**ふぉーでぃふぁーすと**」が返ります。

v文字:=Convert case ("ｱｲｳｴｵABCD12345+-*/?fo-dhifa-suto**" ; 7 ; 16)**

上記のコードを実行すると、
変数"v文字"に「あいうえおABCD12345+-*/?fo-dhifa-suto」が返ります。

v文字:=Convert case ("ｱｲｳｴｵABCD12345+-*/?fo-dhifa-suto**" ; 7 ; (4+16))**

上記のコードを実行すると、
変数"v文字"に「あいうえおあBCD12345+-*/?**ふぉーでぃふぁーすと**」が返ります。

Mac to Win

Mac to Win (テキスト)→ 文字列

引数	タイプ		説明
テキスト	数値	→	Macintosh ASCIIマップを使用して表されたテキスト
関数の返す値	数値	←	Windows ANSIマップを使用して表されたテキスト

Mac to Win関数は、MacOS ASCIIマップを使用して表され、引数＜テキスト＞に渡されるテキストと同等のテキストをWindows ANSIマップを使用して表して返します。

この関数には、MacOS ASCIIマップを使用して表したテキストパラメータが必要です。

通常、Windows上で実行している場合には、このコマンドを使用してASCIIコードを変換する必要はありません。4DとWindowsの間でテキストをコピーしたり貼り付けたりする場合や、データを読み込んだり書き出したりする場合には、4Dは自動的にこのような変換を実行します。ただし、**SEND PACKET**コマンドや**RECEIVE PACKET**コマンドなどの、ディスクの読み取りコマンドや書き込みコマンドを使用する場合には、ASCII変換を明示的に実行する必要があります。これが、**Mac to Win**関数の主要な目的です。

4D内では、まだ別のASCIIマップに変換していないすべてのテキスト値、フィールド、または変数はMacintosh上でもWindows上でもMacOSでコード化されます。詳細については、付録Bの「ASCIIコード」を参照してください。

例：

Windows**SEND PACKET**コマンドを使用してドキュメントに文字を書き込む時に、出力ASCIIマップを使用してMacOSからWindowsへの文字のフィルタリングを実行(**USE ASCII MAP**コマンドを参照してください)していない場合には、手動でテキストをMacOSからWindowsに変換する必要があります。この操作は、次のように実行します：

```
SEND PACKET ($vhDocRef ; Mac to Win(vtSomeText))
` ...
```

参照：SEND PACKET、USE ASCII MAP、Win to Mac

Win to Mac

Win to Mac (テキスト)→ 文字列

引数	タイプ		説明
テキスト	数値	→	Windows ANSマップを使用して表されたテキスト
関数の返す値	数値	←	Macintosh ASCIIマップを使用して表されたテキスト

Win to Mac関数は、Windows ANSIマップを使用して表され、引数＜テキスト＞に渡されるテキストと同等のテキストをMacintosh ASCIIマップを使用して表して返します。

このコマンドには、Windows ANSIマップを使用して表したテキストパラメータが必要です。

通常、Windows上で実行している場合には、このコマンドを使用してASCIIコードを変換する必要はありません。4DとWindowsの間でテキストをコピーしたり貼り付けたりする場合や、データを読み込んだり書き出したりする場合には、4Dは自動的にこのような変換を実行します。ただし、**SEND PACKET**コマンドや**RECEIVE PACKET**コマンドなどの、ディスクの読み取りコマンドや書き込みコマンドを使用する場合には、ASCII変換を明示的に実行する必要があります。これが、**Win to Mac**関数の主要な目的です。

4D内では、まだ別のASCIIマップに変換していないすべてのテキスト値、フィールド、または変数はMacintosh上でもWindows上でもMacOSでコード化されます。詳細については、付録Bの「ASCIIコード」を参照してください。

例：

Windows**SEND PACKET**コマンドを使用してドキュメントに文字を書き込む時に、入力ASCIIマップを使用してWindowsからMacOSへの文字のフィルタリングを実行(**USE ASCII MAP**コマンドを参照してください)していない場合には、手動でテキストをWindowsからMacOSに変換する必要があります。この操作は、次のように実行します：

```
RECEIVE PACKET ($vhDocRef ; vtSomeText ; 16*1024)
vtSomeText:=Win to Mac(vtSomeText)
` ...
```

参照：Mac to Win、RECEIVE PACKET、USE ASCII MAP

Mac to ISO

Mac to ISO (テキスト)→ 文字列

引数	タイプ		説明
テキスト	数値	→	Macintosh ASCIIマップを使用して表されたテキスト
関数の返す値	数値	←	ISO Latin-1文字マップを使用して表されたテキスト

Mac to ISO関数は、Macintosh ASCIIマップを使用して表され、引数<テキスト>に渡されるテキストと同等のテキストをISO Latin-1文字マップを使用して表して返します。

この関数には、Macintosh ASCIIマップを使用して表したテキストパラメータが必要です。

4Dは、Webブラウザとの間で送受信する文字を変換します。その結果、操作するテキスト値は、Web接続プロセスの内部では、常にMacintosh ASCIIマップを使用して表されます。

通常、Windows上で実行している場合には、この関数を使用してASCIIコードを変換する必要はありません。4DとWindowsの間でテキストをコピーしたり貼り付けたりする場合や、データを読み込んだり書き出したりする場合には、4Dは自動的にこのような変換を実行します。ただし、**SEND PACKET**コマンドや**RECEIVE PACKET**コマンドなどの、ディスクの読み取りコマンドや書き込みコマンドを使用する場合には、ASCII変換を明示的に実行する必要があります。

4D内では、まだ別のASCIIマップに変換していないすべてのテキスト値、フィールド、または変数はMacintosh上でもWindows上でもMacOSでコード化されます。詳細については、付録Bの「ASCIIコード」を参照してください。

Windowsでは、この場合には、出力フィルタASCIIマップを使用して文字をフィルタしないでおく必要があります。

従って、プラットフォームが何であっても、ISO Latin-1を使用してディスク上にHTMLドキュメントを作成したい場合には、**Mac to ISO**関数を使用して対象となるテキストを変換するだけですみます。これが、**Mac to IS**関数の主要な目的です

例：

1. 次のコードの行では、vtSomeTextに格納されている(仮想の)MacOSでコード化されたテキストを、ISO Latin-1でコード化されたテキストに変換しています：

```
vtSomeText:=Mac to ISO(vtSomeText)
```

2. 4D Web Serverアプリケーションの開発中にHTMLドキュメントを作成し、後で**SEND HTML FILE**コマンドを使用してイントラネットまたはインターネット経由で送信しようとしています。これらのHTMLドキュメントの一部は他のドキュメントに参照やリンクを持っています。次のプロジェクトメソッドでは、引数として受信したWindowsからのHTMLベースのパス名やMacintoshのパス名を求めています：

```
` 「HTML Pathname」プロジェクトメソッド
` HTML Pathname (テキスト ) → テキスト
` HTML Pathname (ネイティブなファイルマネージャパス名) → HTMLパス名
C_TEXT($0 ; $1)
C_LONGINT($vlChar ; $vlAscii)
C_STRING(31 ; $vsChar)
$0:=""
If ( On Windows )
   $1:=Replace string($1 ; "\" ; "/")
Else
   $1:=Replace string($1 ; ":" ; "/")
End if
$1:=Mac to ISO($1)
For ($vlChar ; 1 ; Length($1))
   $vlAscii:=Ascii($1[[$vlChar]])
   Case of
      \ ($vlAscii>=127)
         $vsChar:="%"+Substring(String($vlAscii;"&$");2)
      \ (Position(Char($vlAscii);":<>&%= "+Char(34))>0)
         $vsChar:="%"+Substring(String($vlAscii ; "&$") ; 2)
   Else
      $vsChar:=Char($vlAscii)
   End case
   $0:=$0+$vsChar
End for
```

注：「On Windows」プロジェクトメソッドは、第52章の「システム文書コマンド」の中で示されています。

「On Window」プロジェクトメソッドが一度データベースに表示されると、特定のディレクトリにあるドキュメントへのFTPリンクのリストを追加したい場合には、次のように記述できます：

```
` 「On Startup」データベースメソッドでインタープロセス変数を設定する
<>vsFTPURL:="ftp://123.4.56.78/Spiders/"
<>vsFTPDirectory:="APS500:Spiders:" ` MacOSファイルマネージャのパス名
` ...
` ...
ARRAY STRING(31 ; $asDocuments ; 0)
DOCUMENT LIST(...; $asDocuments)
$vlNbDocuments:=Size of array($asDocuments)
jsHandler:=...
For ($vlDocument ; 1 ; $vlNbDocuments)
   vtHTMLCode:=vtHTMLCode+"<P><A HREF="+Char(34)+<>vsFTPURL
   + HTML Pathname (Substring($1+$asDocuments{$vlDocument}
   ; Length(<>vsFTPDirectory)+1))+Char(34)+jsHandler
   +"> "+$asDocuments{$vlDocument}+"</A></P>"+Char(13)
End for
` ...
```

参照：ISO to Mac、SEND HTML FILE、SEND PACKET、USE ASCII MAP

ISO to Mac

ISO to Mac (テキスト)→ 文字列

引数	タイプ		説明
テキスト	数値	→	Macintosh ASCIIマップを使用して表されたテキスト
関数の返す値	数値	←	ISO Latin-1文字マップを使用して表されたテキスト

ISO to Mac関数は、ISO Latin-1文字マップを使用して表され、引数＜テキスト＞に渡されるテキストと同等のテキストをMacOS ASCIIマップを使用して表して返します。

通常、このコマンドを使用する必要はありません。

このコマンドには、ISO Latin-1文字マップを使用して表したテキストパラメータが必要です。

4Dは、Webブラウザとの間で送受信する文字を変換します。その結果、操作するテキスト値は、Web接続プロセスの内部では、常にMacOS ASCIIマップを使用して表されます。

通常、Windows上で実行している場合には、ASCIIコードを変換する必要はありません。4DとWindowsの間でテキストをコピーしたり貼り付けたりする場合や、データを読み込んだり書き出したりする場合には、4Dは自動的にこのような変換を実行します。ただし、**SEND PACKET**コマンドや**RECEIVE PACKET**コマンドなどの、ディスクの読み取りコマンドや書き込みコマンドを使用する場合には、4DはASCII変換を実行しません。

4D内では、まだ別のASCIIマップに変換していないすべてのテキスト値、フィールド、または変数はMacintosh上でもWindows上でもMacOSでコード化されます。詳細については、付録Bの「ASCIIコード」を参照してください。

Windowsでは、この場合には、出力フィルタASCIIマップを使用して文字をフィルタしないでおく必要があります。

従って、プラットフォームが何であっても、**RECEIVE PACKET**コマンドを使用してディスクからISO Latin-1によるHTMLドキュメントを読み取りしたい場合には、**ISO to Mac**関数を使用して対象となるテキストを変換するだけですみます。これが、**ISO to Mac**関数の主要な目的です

例：

次のコードの行では、vtSomeTextに格納されている(仮想の)ISO Latin-1でコード化されたテキストを、MacOSでコード化されたテキストに変換しています：

```
`ISO Latin-1によるHTMLドキュメントからテキストを読み取る
RECEIVE PACKET ($vhDocRef ; vtSomeText ; 16*1024)
vtSomeText:=ISO to Mac(vtSomeText)
```

参照：Mac to ISO、RECEIVE PACKET、USE ASCII MAP

ストラクチャアクセスコマンド　50

この章では、「ルーチン」エディタの「Structure Access」テーマ内にあるストラクチャアクセスコマンドについて説明します。この章のコマンドは、データベースストラクチャの情報を返します。これらのコマンドは、テーブル数、各テーブルのフィールド数、テーブルの名前、フィールドの名前、フィールドタイプを返します。

データベースストラクチャを特定すると、開発済みまたは使用中のメソッドやフォームのモジュールを他のデータベースにコピーする場合に非常に便利です。データベースストラクチャの情報を読み取ることによって、さまざまなデータベースで使用可能になり、移植性の高いコードを生成することができます。

Count fields　　**Field name**　　**Table**
Count tables　　**GET FIELD PROPERTIES**　　**Table name**
Field　　**SET INDEX**

Count tables

Count tables → 数値

Count tables関数は、データベース中のテーブルの数を返します。テーブルは作成された順番に番号が付けられます。

▼ 次の例は、配列 “ テーブル配列 ” の配列要素を初期化します。この配列はフォーム上のポップアップメニューに割り当てられ、データベース内のテーブルのリストを表示するために使用されます。For文では、テーブルの数がこのメソッドを実行中に変更することができますので、**Count tables**関数の代わりに**Size of array**関数を使用しています：

```
ARRAY STRING (15 ; テーブル配列 ; Count tables)  ` 配列を作成する
For ($i ; 1 ; Size of array (テーブル配列))      ` 配列を通してループする
   テーブル配列 {$i}:=Table name ($i)            ` 配列要素に値を代入する
End for
```

Count fields

Count fields (テーブル番号) → 数値

引数	タイプ	説明
テーブル番号	数値	テーブル番号

Count fields (テーブルポインタ) → 数値

引数	タイプ	説明
テーブルポインタ	ポインタ	ポインタテーブルに対するポインタ

Count fields関数には、2つの形式があります。

Count fields関数は、＜テーブル番号＞または＜テーブルポインタ＞で指定したテーブルのフィールドの数を返します。フィールドは作成された順に番号が付けられています。

▼ 次の例は、引数$iに渡されるテーブルポインタで、ドロップダウン（ポップアップ）メニューにフィールドのリストが表示されます。まず、配列 “ フィールド配列 ” が作成され、**For**ループを通して配列要素に値を代入します。**For**ループでは、**Count fields**関数の代わりに**Size of array**関数を使用します。これは、次のコードが実行されている間にテーブル内のフィールド数が変更されるためです：

```
C_LONGINT ($TableNumber ; $HowMany ; $i)
$TableNumber :=Table ($1)
` 配列を作成する
ARRAY STRING (15 ; フィールド配列 ; Count fields ($TableNumber))
$HowMany:=Size of array (フィールド配列)
For ($i ; 1 ; $HowMany)                                  ` 配列を通してループする
   ﾌｨｰﾙﾄﾞ配列{$i}:=Field name ($TableNumber ; $i)   ` 配列要素に値を代入する
End for
```

Table name

Table name (テーブル番号) → 文字列

引数	タイプ	説明
テーブル番号	数値	テーブル番号

Table name (テーブルポインタ) → 文字列

引数	タイプ	説明
テーブルポインタ	ポインタ	テーブルへのポインタ

Table name関数には、2つの形式があります。

Table name関数は、＜テーブル番号＞または＜テーブルポインタ＞で指定したテーブルの名前を返します。

▼ 次の例は、あるテーブルのレコードを表示します。テーブルへの参照は、テーブルに対するポインタとして渡されます。**Table name**関数は、ウインドウのタイトルバーにテーブルの名前を表示するために使用されます：

```
` $1はテーブルに対するポインタ
SET WINDOW TITLE (Table name ($1) + " のレコード")
` ウインドウタイトルを設定する
DISPLAY SELECTION ($1->)    ` セレクションを表示する
```

Field name

Field name (テーブル番号 ; フィールド番号) → 文字列

引数	タイプ	説明
テーブル番号	数値	テーブル番号
フィールド番号	数値	フィールド番号

Field name (フィールドポインタ) → 文字列

引数	タイプ	説明
フィールドポインタ	ポインタ	フィールドへのポインタ

Field name関数には、2つの形式があります。

Field name関数は、＜テーブル番号＞と＜フィールド番号＞または＜フィールドポインタ＞で指定したフィールドの名前を返します。

▼ 次の例は、配列 “ フィールド{1} ” の2番目の要素にテーブル番号1に含まれるフィールド番号2のフィールドの名前を代入します：

フィールド{1}{2}:=**Field name** (1 ; 2)

▼ 次の例は、配列 “ フィールド{1} ” の2番目の要素にフィールド “ [私のテーブル]私のフィールド ” の名前を代入します。この方法は、“ [私のテーブル]私のフィールド ” のフィールド名が変更された場合でも、新しいフィールド名が返されますので便利です：

フィールド{1}{2}:=**Field name** (->[私のテーブル]私のフィールド)

▼ 次の例は、フィールドに対してポインタを渡して、警告を表示します：

ALERT (**Table name** (**Table** ($1)) + "テーブルの" + **Field name** ($1) + "フィールドのID番号は5文字以上でなければいけません")

Table

Table (テーブル番号) → ポインタ

引数	タイプ	説明
テーブル番号	数値	テーブル番号

Table (テーブルポインタ) → 数値

引数	タイプ	説明
テーブルポインタ	ポインタ	テーブルへのポインタ

Table (フィールドポインタ) → 数値

引数	タイプ	説明
フィールドポインタ	ポインタ	フィールドへのポインタ

Table関数には、3つの形式があります。

＜テーブル番号＞を指定した場合は、**Table**関数はテーブルへのポインタを返します。＜テーブルポインタ＞を指定した場合は、**Table**関数はテーブル番号を返します。＜フィールドポインタ＞を指定した場合には、**Table**関数はそのフィールドの属するテーブルのテーブル番号を返します。この形式は、フィールドポインタだけを指定してテーブル番号が求められます。

▼次の例は変数“テーブルポインタ”に3番目のテーブルに対するポインタを代入します：

テーブルポインタ:=Table (3)

変数“テーブルポインタ”を第2の形式に使用すると、**Table**関数は数値の3を返します。次の例を実行すると変数“テーブル番号”に3を代入します。

テーブル番号:=Table (テーブルポインタ)

▼次の例は変数“ﾌｧｲﾙ番号”にテーブル[テーブル3]のテーブル番号を代入します：

テーブル番号:=Table (->[テーブル3])

▼次の例は変数“テーブル番号”にフィールド“[テーブル3]フィールド1”の属するテーブルのテーブル番号を代入します：

テーブル番号:=Table (->[テーブル3]フィールド1)

Field

Field (テーブル番号 ; フィールド番号) → ポインタ

引数	タイプ	説明
テーブル番号	数値	テーブル番号
フィールド番号	数値	フィールド番号

Field (フィールドポインタ) → 数値

引数	タイプ	説明
フィールドポインタ	ポインタ	フィールドへのポインタ

Field関数には、2つの形式があります。

＜テーブル番号＞と＜フィールド番号＞を指定した場合は、**Field**関数はフィールドへのポインタを返します。

＜フィールドポインタ＞を指定した場合には、**Field**関数はフィールド番号を返します。

▼次の例は、変数“ フィールドポインタ ”に3番目のテーブルに含まれる2番目のフィールドへのポインタを代入します：

フィールドポインタ:=Field (3 ; 2)

変数“ フィールドポインタ ”を第2の形式に使用すると、**Field**関数は数値の2を返します。

▼ 次の例を実行すると変数“ フィールド番号 ”に2を代入します：

フィールド番号:=Field (フィールドポインタ)

▼ 次の例は、変数“ フィールド番号 ”にフィールド“ [テーブル3]フィールド2 ”のフィールド番号を代入します：

フィールド番号:=Field (->[テーブル3]フィールド2)

GET FIELD PROPERTIES

GET FIELD PROPERTIES (テーブル番号 ; フィールド番号 ; タイプ ; {長さ {;インデックス}})

引数	タイプ		説明
テーブル番号	数値	→	テーブル番号
フィールド番号	数値	→	フィールド番号
タイプ	数値	←	フィールドタイプ
長さ	数値	←	文字フィールドの長さ(バイト)
インデックス	ブールタイプ	←	False(偽)：インデックスなし、 True(真)：インデックスあり

GET FIELD PROPERTIES (フィールドポインタ ; タイプ ; {長さ {;インデックス}})

引数	タイプ		説明
フィールドポインタ	ポインタ	→	フィールドポインタ
タイプ	数値	←	フィールドタイプ
長さ	数値	←	文字フィールドの長さ(バイト)
インデックス	ブール	←	False(偽)：インデックスなし、

GET FIELD PROPERTIESコマンドには、2つの形式があります。

GET FIELD PROPERTIESコマンドは、＜テーブル番号＞と＜フィールド番号＞または＜フィールドポインタ＞で指定したフィールドの情報を引数＜タイプ＞、＜長さ＞、＜インデックス＞に返します。

GET FIELD PROPERTIESコマンドでは、次のような4th Dimensionであらかじめ定義された定数の値を引数＜タイプ＞に設定します。

定数	タイプ	値
Is Alpha Field	倍長整数	0
Is Text	倍長整数	2
Is Real	倍長整数	1
Is Integer	倍長整数	8
Is LongInt	倍長整数	9
Is Date	倍長整数	4
Is Time	倍長整数	11
Is Boolean	倍長整数	6
Is Picture	倍長整数	3
Is Subtable	倍長整数	7
Is BLOB	倍長整数	30

GET FIELD PROPERTIESコマンドが引数＜長さ＞に設定している情報は、フィールドタイプが文字の場合（つまり、「**＜タイプ=Is Alpha Field**」にだけ意味を持ちます。フィールドに設定された長さをバイト数で設定します。

GET FIELD PROPERTIESコマンドが引数＜インデックス＞に設定する情報は、フィールドタイプが文字、整数、倍長整数、実数、日付、時間、ブールの場合にだけ意味を持ちます。フィールドに＜インデックス＞が設定されていない場合は、“ False(偽) ” を設定します。フィールドにインデックスが設定されている場合は、“ True(真) ” を設定します。

▼ 次の例は、変数 “ vタイプ ” , “ v長さ ” , “ vインデックス ” に1番目のテーブルに含まれる3番目の属性を設定します：

GET FIELD PROPERTIES (1 ; 3 ; vタイプ ; v長さ ; vインデックス)

▼ 次の例は、変数 “ vタイプ ”、“ v長さ ”、“ vインデックス ” にフィールド “ [テーブル3]フィールド2 ” の属性を設定します：

GET FIELD PROPERTIES (->[テーブル3]フィールド2 ; vタイプ ; v長さ ; vインデックス)

参照：Field name、SET INDEX

SET INDEX

SET INDEX (フィールド ; インデックス ; { *})

引数	タイプ	説明
フィールド	フィールド	変更するフィールド
インデックス	ブール	False(偽)：インデックスなし、 True(真)：インデックスあり
*		インデックス付けの非同期化

SET INDEXコマンドは、＜フィールド＞に対してインデックスを作成または取り除きます。インデックスの作成は別プロセスで行われますので、その間データベースを使用することができます。インデックスを作成中にインデックスを使用する処理をしてもインデックスは用いられません。＜フィールド＞にインデックスが付いているかどうかを調べるには**FIELD ATTRIBUTES**コマンドを使用します。

SET INDEXコマンドはロックされたレコードのインデックス付けは行いません：レコードがロック解除されるまで待機します。

＜インデックス＞が“ True(真) ”の場合は、＜フィールド＞のインデックスが作成されます。＜フィールド＞にインデックスが付いている場合は、このコマンドは何もしません。

＜インデックス＞が“ False(偽) ”の場合は、＜フィールド＞にインデックスが付いているとそのインデックスを取り除きます。

オプション引数＜*＞を用いて、非同期に（同時に）インデックスが付けられるようになりました。非同期なインデックス付けにより、インデックス付けが終了したかどうかに関係なく、呼び出し側のメソッドの実行をそのまま続行できます。ただし、インデックスが必要なコマンドを実行すると失敗します。

▼次の例は、フィールド“ 顧客ID ”にインデックスを付けます：

```
UNLOAD RECORD ([顧客])
SET INDEX ([顧客]顧客ID ; True)
```

サブレコードコマンド　51

この章では、「ルーチン」エディタの「Subrecords」テーマ内にあるサブレコードコマンドについて説明します。この章のコマンドは、レコードを操作するのと同じようにサブレコードを処理するためのものです。これらのコマンドを使用すれば、サブレコードを追加、修正、削除したり、フォーミュラで更新することができます。

ALL SUBRECORDS
APPLY TO SUBSELECTION
Before subselection
CREATE SUBRECORD
DELETE SUBRECORD
End subselection
FIRST SUBRECORD
LAST SUBRECORD
NEXT SUBRECORD
ORDER SUBSELECTION BY
PREVIOUS SUBRECORD
QUERY SUBSELECTION
Records in subselection

FIRST SUBRECORD、**LAST SUBRECORD**、**NEXT SUBRECORD**、**PREVIOUS SUBRECORD**の各コマンドを使用してサブセレクション内のサブレコードを移動することもできます。これらのコマンドを使用する前に正しいサブセレクションを作成することが重要です。カレントサブセレクションにサブレコードが存在しない場合には、これらのコマンドは何も行いません。

プログラミングをしないで、これらのサブレコードを移動するためのコマンドと同じ機能をフォーム上のボタンに対して割り当てることができます。

CREATE SUBRECORD

CREATE SUBRECORD (サブテーブル)

引数	タイプ	説明
サブテーブル	サブテーブル	新しいサブレコードを作成するサブテーブル

CREATE SUBRECORDコマンドは、＜サブテーブル＞に対して新しいサブレコードを作成し、それをカレントサブレコードとします。新しいサブレコードは、親レコードが**SAVE RECORD**コマンドのようなコマンドで実行されたり、ユーザによる保存指示が行われて保存された場合にのみディスクに保存されます。カレント親レコードが存在しない場合には、**CREATE SUBRECORD**コマンドは何も行いません。入力フォームを通してサブレコードを追加する場合には**ADD SUBRECORD**コマンドを使用します。

▼ 次の例は、ボタンに対するオブジェクトメソッドです。ボタンがクリックされると、このオブジェクトメソッドを実行し複数の子供の新しいサブレコードを作成します。一度に複数の子供を追加できるように、**Repeat**ループで「キャンセル」ボタンがクリックされるまで繰り返すように制御しています。フォームに、サブフォームを使用して複数の子供を表示、入力することができますが「入力可」オプションをオフに設定した場合には、サブテーブルに直接データを入力することはできません：

```
` 「キャンセル」ボタンがクリックされるまで繰り返す
Repeat
 v子供:=Request ("子供の名前を入力してください：")
  ` 子供の名前を得る
 If (OK=1)                              ` 実行？
   CREATE SUBRECORD (子供)              ` 子供の新しいサブレコード作成
   子供'名前:=v子供                     ` 子供の名前をサブフィールドへ代入
 End if
Until (OK=0)
```

DELETE SUBRECORD

DELETE SUBRECORD (サブテーブル)

引数	タイプ	説明
サブテーブル	サブテーブル	サブレコードを削除するサブテーブル

DELETE SUBRECORDコマンドは、＜サブテーブル＞のカレントサブレコードを削除します。カレントサブレコードやカレント親レコードが存在しない場合には、**DELETE SUBRECORD**コマンドは何も行いません。サブレコードが削除されるとサブテーブルのカレントサブセレクションは空になります。そのため、**DELETE SUBRECORD**コマンドは、サブセレクション内のサブレコードに対して削除処理を繰り返し行うことはできません。

サブレコードの削除は、親レコードが保存されるまでは恒久的なものではありません。親レコードを削除した場合は、そのレコードに属するサブレコードはすべて削除されます。

▼ 次の例は、サブテーブルのすべてのサブフィールドを削除します：

```
ALL SUBRECORDS ([従業員]子供)
While (Records in subselection ([従業員]子供) > 0)
   DELETE SUBRECORD ([従業員]子供)
   ALL SUBRECORDS ([従業員]子供)
End while
```

▼ 次の例は、[子供]サブテーブルから年齢が12歳以上の子供のサブレコードをすべて削除します：

```
ALL RECORDS ([従業員]) ` すべてのレコードを選択する
For ($i ; 1 ; Records in selection ([従業員])) ` テーブルの全レコードに対して
      ` 検索条件に沿ってサブレコードを検索する
   QUERY SUBRECORDS ([従業員]子供 ; [従業員]子供'年齢 >= 12)
    ` レコード間をループする
   While (Records in subselection ([従業員]子供) > 0)
      DELETE SUBRECORD ([従業員]子供)    ` サブレコードを削除する
    ` 次のレコードを検索する
      QUERY SUBRECORDS ([従業員]子供 ; [従業員]子供'年齢 >= 12)
   End while
   SAVE RECORD ([従業員])                    ` 親レコードを保存する
   NEXT RECORD ([従業員])
End for
```

ALL SUBRECORDS

ALL SUBRECORDS (サブテーブル)

引数	タイプ	説明
サブテーブル	サブテーブル	すべてのサブレコードを選択するサブテーブル

ALL SUBRECORDSコマンドは、＜サブテーブル＞のすべてのサブレコードをカレントサブセレクションにします。カレント親レコードが存在しない場合には、**ALL SUBRECORDS**コマンドは何も行いません。親レコードがロードされた時点では、サブセレクションはサブテーブル上のすべてのサブレコードを含みますが、**ADD SUBRECORDS**、**QUERY SUBRECORDS**、**DELETE SUBRECORD**などの各コマンドを実行した後では、サブセレクションがサブテーブル上のすべてのサブレコードを含むとは限りません。

▼ 次の例は、すべてのサブレコードを選択し、その金額の合計を算出します：

```
ALL SUBRECORDS ([地域]売上)
売上合計:=Sum ([地域]売上'金額)
```

Records in subselection

Records in subselection (サブテーブル) → 数値

引数	タイプ	説明
サブテーブル	サブテーブル	サブレコードの数を得るサブテーブル

Records in subselection関数は、＜サブテーブル＞のカレントサブセレクションに含まれるサブレコードの数を返します。**Records in subselection**関数は、カレントレコードのカレントサブセレクションに対してのみ適用されます。この関数は、**Records in selection**関数のサブテーブル版と言えます。カレント親レコードが存在しない場合には、値として未定義(Undefined)を返します。

▼ 次の例は、すべてのサブレコードを選択してから、親レコードに対する子供の数を表示します：

```
ALL SUBRECORDS ([従業員]子供)                ` すべてのサブレコードを選択
ALERT ("子供の数：" + String (Records in subselection ([従業員]子供)))
```

APPLY TO SUBSELECTION

APPLY TO SUBSELECTION (サブテーブル ; ステートメント)

引数	タイプ	説明
サブテーブル	サブテーブル	フォーミュラで更新するサブテーブル
ステートメント	ステートメント	1行のステートメントまたはグローバルメソッド

APPLY TO SUBSELECTIONコマンドは、＜サブテーブル＞のカレントサブセレクションのすべてのサブレコードに対して＜ステートメント＞を適用します。この＜ステートメント＞は、1行のステートメントまたはプロジェクトメソッドを指定します。ステートメントでサブレコードを修正しても、その結果はディスクに保存されません。親レコードが保存された時点でディスクに保存されます。サブセレクションクションが空の場合は、**APPLY TO SUBSELECTION**コマンドは何も行いません。

APPLY TO SUBSELECTIONコマンドは、サブセレクションから情報を収集したりサブセレクションを修正するために使用します。

▼ 次の例は、すべてのサブレコードに対し単価と数量から売上金額を算出します：

```
ALL SUBRECORDS ([請求書]明細)                ` すべてのサブレコードを選択
APPLY TO SUBSELECTION ([請求書]明細 ;
[請求書]明細'売上金額 := [請求書]明細'単価 * [請求書]明細'数量)
```

FIRST SUBRECORD

FIRST SUBRECORD (サブテーブル)

引数	タイプ	説明
サブテーブル	サブテーブル	先頭サブレコードに移動するサブテーブル

FIRST SUBRECORDコマンドは、＜サブテーブル＞のカレントサブセレクションの先頭のサブレコードをカレントサブレコードにします。またすべての検索コマンド、選択コマンド、ソートコマンドも先頭のサブレコードをカレントサブレコードに設定します。カレントサブセレクションが空の場合は、**FIRST SUBRECORD**コマンドは何も行いません。

▼ 次の例は、サブテーブルに格納されている子供の名字と名前を連結します。連結した名前を“ 子供の名前 ”という配列にコピーします：

```
` 名前を保存する配列を宣言する
ARRAY TEXT (子供の名前 ; Records in subselection ([従業員]子供))
FIRST SUBRECORD ([従業員]子供)          ` 先頭のサブレコードへ移動
` 子供の数だけ繰り返す
For ($i ; 1 ; Records in subselection ([従業員]子供))
  子供の名前{$i}:=[従業員]子供'名字 + "  " + [従業員]子供'名前
  NEXT SUBRECORD ([従業員]子供)
End for
```

LAST SUBRECORD

LAST SUBRECORD (サブテーブル)

引数	タイプ	説明
サブテーブル	サブテーブル	最終サブレコードに移動するサブテーブル

LAST SUBRECORDコマンドは、＜サブテーブル＞のカレントサブセレクションの最終のサブレコードをカレントサブレコードにします。カレントサブセレクションが空の場合は、**LAST SUBRECORD**コマンドは何も行いません。

▼ 次の例は、サブテーブルに格納されている子供の名字と名前を連結します。連結した名前を“ 子供の名前 ”という配列にコピーします。最終のサブレコードから先頭のサブレコードへ処理していく以外は、**FIRST SUBRECORD**コマンドの例と同じです：

```
` 名前を保存する配列を宣言する
ARRAY TEXT (子供の名前 ; Records in subselection ([従業員]子供))
LAST SUBRECORD ([従業員]子供)                ` 最終のサブレコードへ移動
` 子供の数だけ繰り返す
For ($i ; 1 ; Records in subselection ([従業員]子供))
  子供の名前{$i}:=[従業員]子供'名字 + "  " + [従業員]子供'名前
  PREVIOUS SUBRECORD ([従業員]子供)
End for
```

NEXT SUBRECORD

NEXT SUBRECORD (サブテーブル)

引数	タイプ	説明
サブテーブル	サブテーブル	次のサブレコードに移動するサブテーブル

NEXT SUBRECORDコマンドは、＜サブテーブル＞のカレントサブセレクションのカレントサブレコードポインタを1つ次のサブレコードに移動します。**NEXT SUBRECORD**コマンドが、最終のサブレコードを過ぎてポインタを移動した場合には、**End subselection**関数に“ True(真) ”を設定し、カレントサブレコードはなくなります。**End subselection**関数“ True(真) ”を返した場合には、ポインタをカレントサブレコードに戻すためには、**FIRST SUBRECORD**コマンドまたは**LAST SUBRECORD**コマンドを使用します。カレントサブセレクションが空の場合や**Before subselection**関数が“ True(真) ”を返した場合には、**NEXT SUBRECORD**コマンドは何も行いません。

FIRST SUBRECORDコマンドについては、前節の例を参照してください。

PREVIOUS SUBRECORD

PREVIOUS SUBRECORD (サブテーブル)

引数	タイプ	説明
サブテーブル	サブテーブル	前のサブレコードに移動するサブテーブル

PREVIOUS SUBRECORDコマンドは、＜サブテーブル＞のカレントサブセレクションのカレントサブレコードポインタを1つ前のサブレコードに移動します。**PREVIOUS SUBRECORD**コマンドが先頭のサブレコードを過ぎてポインタを移動した場合には、**Before subselection**関数に“ True(真) ”を設定し、カレントサブレコードはなくなります。**Before subselection**関数が“ True(真) ”を返した場合は、ポインタをカレントサブレコードに戻すためには、**FIRST SUBRECORD**コマンドや**LAST SUBRECORD**コマンドを使用します。カレントサブセレクションが空の場合や**End subselection**関数が“ True(真) ”を返した場合には、**PREVIOUS SUBRECORD**コマンドは何も行いません。

LAST SUBRECORDコマンドについては、前節の例を参照してください。

Before subselection

Before subselection (サブテーブル) → ブール

引数	タイプ	説明
サブテーブル	サブテーブル	ポインタがセレクションの前にあるか判定するサブテーブル

Before subselection関数は、＜サブテーブル＞のカレントサブレコードポインタが先頭のサブレコードよりも前にあるときに“ True(真) ”を返します。**Before subselection**関数は、**PREVIOUS SUBRECORD**コマンドでポインタを移動するときに、ポインタが先頭のサブレコードを過ぎたかどうかを調べるために使用します。カレントサブセレクションが空の場合にも、**Before subselection**関数は“ True(真) ”を返します。

▼ 次の例は、ボタンに対するオブジェクトメソッドです。ボタンがクリックされるとポインタが前のサブレコードへ移動します。ポインタが先頭のサブレコードの前を指した場合には、最終のサブレコードへ移動します：

```
PREVIOUS SUBRECORD ([従業員]子供)      ` 前のサブレコードへ移動
If (Before subselection ([従業員]子供)) ` サブレコードの終わり？
  LAST SUBRECORD ([従業員]子供)        ` 最終のサブレコードへ移動
End if
```

End subselection

End subselection (サブテーブル) → ブール

引数	タイプ	説明
サブテーブル	サブテーブル	ポインタがセレクションの後にあるかを判定するサブテーブル

End subselection関数は、サブテーブルのカレントサブレコードポインタが最終のサブレコードよりも後にあるときに“ True(真) ”を返します。**End subselection**関数は、**NEXT SUBRECORD**コマンドでポインタを移動するときに、ポインタが最終のサブレコードを過ぎたかどうかを調べるに使用します。カレントサブセレクションが空の場合にも、**End subselection**関数は“ True(真) ”を返します。

▼ 次の例は、ボタンに対するオブジェクトメソッドです。ボタンがクリックされるとポインタが次のサブレコードへ移動します。ポインタが最終のサブレコードの後を指した場合には、先頭のサブレコードへ移動します：

```
NEXT SUBRECORD ([従業員]子供)          ` 次のサブレコードへ移動
If (End subselection ([従業員]子供))   ` サブレコードの終わり？
  FIRST SUBRECORD ([従業員]子供)       ` 先頭のサブレコードへ移動
End if
```

ORDER SUBRECORD BY

ORDER SUBRECORD BY (サブテーブル;サブフィールド1;{ソート種別1} {; ... ;サブフィールドN;{ソート種別N}})

引数	タイプ	説明
サブテーブル	サブテーブル	ソート対象となるサブテーブル
サブフィールド	サブフィールド	ソートキーとなるサブフィールド
ソート種別	>または<	> 昇順 、 < 降順

ORDER SUBRECORD BYコマンドは、＜サブテーブル＞のカレントサブセレクションをソートします。このコマンドは、カレント親レコードのサブセレクションのみをソートします。

＜ソート種別＞で昇順または降順にソートするのかを指定します。＜ソート種別＞が(>)の場合は、ソートを昇順に行います。＜ソート種別＞が(<)の場合には、ソートを降順に行います。＜ソート種別＞を省略した場合には、ソートを昇順に行います。

1つのステートメント内に複数のソートキーとなるサブフィールドとソート種別が使用できます。

ソート処理が完了した時点で、ソートしたサブセレクションの先頭のレコードをカレントサブレコードとします。サブレコードのソートは、一時的な処理です。サブレコードはソートした状態で保存されることはありません。カレントレコードが存在しない場合やサブテーブルが定義されていない場合は、**ORDER SUBRECORD BY** コマンドは何も行いません。

フォームが固定フレームで印刷される組み込みエリアを含んでいる場合、このコマンドは親フォームのフォームメソッドのBeforeフェーズで印刷前に1度呼ばれる必要があります。

▼ 次の例は、金額サブフィールドをソートキーにして[地域]売上サブテーブルを昇順にソートします：

```
ORDER  SUBRECORD BY ([地域]売上 ; [地域]売上'金額 ; >)
```

QUERY SUBRECORDS

QUERY SUBRECORDS (サブテーブル ; 検索条件式)

引数	タイプ	説明
サブテーブル	サブテーブル	検索対象となるサブテーブル
検索条件式	ブール	検索条件式

QUERY SUBRECORDSコマンドは、＜サブテーブル＞を検索し、新しいサブセレクションを生成します。このコマンドは、サブレコードを検索する機能を持つ唯一のコマンドです。＜検索条件式＞は、サブテーブルの各サブレコードに適用します。＜検索条件式＞は、“ True(真) ”または“ False(偽) ”のいずれかの状態に評価されるブール式です。＜検索条件式＞で“ True(真) ”に評価されたサブレコードは、新しいサブセレクションに追加されます。**QUERY SUBRECORDS**コマンドは、検索処理が完了した時点で新しいサブセレクションを生成し、その先頭のサブレコードがカレントサブレコードとします。

QUERY SUBRECORDSコマンドは、カレントレコードのサブレコードのみを検索し、親テーブルの複数のレコードに対しては検索できません。**QUERY SUBRECORDS**コマンドには、親レコードに対して操作する機能はありません。

＜検索条件式＞は、サブフィールドに対し比較演算子を使用して変数または定数との比較を行います。＜検索条件式＞同士は、論理演算子“ かつ(&) ”または論理演算子“ または(|) ”を使用して結合することができます。また、＜検索条件式＞中に関数を含むこともできます。ワイルドカード記号(@)は、文字列を検索する場合にのみ使用できます。

カレントレコードが存在しない場合またはサブテーブルが定義されていない場合は、**QUERY SUBRECORDS**コマンドは何も行いません。

▼ 次の例は、10歳以上の子供を検索します：

```
QUERY SUBRECORDS ([従業員]子供 ; [従業員]子供'年齢 > 10)
```

システム文書コマンド　52

この章では、「ルーチン」エディタの「System Documents」テーマ内にあるシステム文書コマンドについて説明します。この章のコマンドは、文書ファイルに関する作業を行います。

Append document
CLOSE DOCUMENT
COPY DOCUMENT
Create document
CREATE FOLDER
DELETE DOCUMENT
Document Creater
DOCUMENT LIST
Document type
FOLDER LIST
Get document position
GET DOCUMENT PROPERTIES
Get document size
MAP FILE TYPES
MOVE DOCUMENT
Open document
SET DOCUMENT CREATOR
SET DOCUMENT POSITION
SET DOCUMENT PROPERTIES
SET DOCUMENT SIZE
SET DOCUMENT TYPE
Test path name
VOLUME ATTRIBUTES
VOLUME LIST

システム文書の概要

コンピュータで使用するすべての文書（ドキュメント）やアプリケーションは、コンピュータに接続されたりマウントされているハードディスク、フロッピーディスク、あるいはその他の記憶装置にファイルとして格納されています。4th Dimension内では、このような文書を区別せずにファイルまたは文書と呼ぶことができます。ただし、この章のほとんどのコマンドは(アプリケーションやシステムファイルではなく)ディスク上にある文書にアクセスする目的で使用するため、ここでは、“ 文書 ” という用語を使用しています。

1つのハードディスクは、1つまたは複数のパーティションとしてフォーマットすることができます。個々のパーティションは、ボリュームと呼ばれます。2つのボリュームが同一のハードディスクに物理的に存在するかどうかに関係なく、4Dのレベルでは、通常、これら2つのボリュームを2つの同等な存在物として扱います。

ボリュームは、コンピュータに物理的に接続されたり、NetBEUI(Windows)やAFP(Macintosh)のようなファイル共有プロトコルを使用してネットワーク経由でマウントされているハードディスク上にあります。どのような場合でも、4Dのレベルでは、システム

文書コマンドを使用する場合には、これらのボリュームを同等に扱います(方法について熟知しており、4Dのプラグインソフトウェアを使用してそのドメインのアプリケーションの機能を拡張する場合は例外です)。

個々のボリュームには、ボリューム名があります。Windowsでは、ボリュームは文字の後にコロンを続けて表します。通常、A:およびB:が、5.25インチまたは3.5インチのフロッピーディスクドライブを表します。通常、C:は、システムをブートするために使用するボリュームを表します(コンピュータを他の方法で構成している場合は例外です)。従って、D:～Z:までの文字を使用して、コンピュータに接続またはマウントされるボリューム(CD-ROMドライブ、追加ドライブ、ネットワークドライブなど)を表します。Macintoshでは、ボリュームには任意の名前を付けることができ、最長で31文字になります(ファインダのレベルでデスクトップに表示される名前です)。

通常、文書はフォルダに分類します。フォルダには、任意のフォルダを含むこともできます。何百個あるいは何千個ものファイルをボリュームの同一のレベルに蓄積するのはあまりよい方法ではありません。まず、見栄えが悪く、次にシステムの速度を低下させてしまいます。Windowsでは、フォルダはディレクトリと呼ばれており、今でもそのように呼ばれ(フォルダという用語のほうが、Windows 95の登場以来、より頻繁に使用されています)、Macintoshでは通常、フォルダと呼ばれます。

文書をユニークに識別するには、文書自体の名前の他に、その文書があるボリュームの名前およびフォルダの名前も必然的に知っている必要があります。これらの名前をすべて連結すれば、その文書へのパス名になります。このパス名の中では、フォルダ名はディレクトリ(区切り)記号と呼ばれる特殊文字で区切られています。Windowsでは、この文字は円記号(\)であり、Macintoshではコロン(:)です。

例で検討してみましょう。「Memo」というフォルダに「Important Memo」というド文書が入っており、Memo自体も「Document」というフォルダに、さらにDocumentは「Current Work」というフォルダに入っています。

Windowsでは、これらすべてが「C:」というドライブ(ボリューム)にある場合には、この文書へのパス名は、次のようになります。

C:\Current Work\Documents\Memos\Important Memo.TXT

Macintoshでは、これらすべてが「Internal Drive」というディスク(ボリューム)にある場合には、 この文書へのパス名は、次のようになります。

Internal Drive:Current Work:Documents:Memos:Important Memo

Windowsでは、この例の場合、文書の名前にファイル拡張子「.TXT」がついていることに注意してください。理由はこの後で説明します。

プラットフォームが何であっても、文書へのパス名は、正式には次のように表現できます。

VolName DirSep { DirName DirSep { DirName DirSep { ... } } } DocName

ボリュームにあるすべての文書(ファイル)には、通常、属性またはプロパティと呼ばれる複数の特性があります。

文書自体の名前(Macintoshでは31文字以内、Windows 3.1では8文字以内、Windows 95およびWindows NTでは255文字以内)

文書ファイルのタイプとクリエータ

Windowsでは、文書にはタイプがあります。Macintoshでは、文書にはタイプとクリエータがあります。文書のタイプは、通常、文書の種類または内容を表します。例えば、テキスト文書にはテキストが含まれます(スタイルなどは含まれません)。Windowsでは、文書のタイプは、その文書の名前に追加されたファイル拡張子と呼ばれる接尾辞によって判断されます。例えば、「.TXT」はテキスト文書に対するWindowsでのファイル拡張子です。Macintoshでは、文書のタイプは、4文字の識別子(ファインダのレベルでは表示されません)である文書のファイルタイププロパティによって判断されます。例えば、テキスト文書のファイルタイプは「TEXT」です。さらにMacintoshでは、文書には、その文書を作成したアプリケーションを表すクリエータがあります。この概念は、Windowsにはありません。文書のクリエータは、4文字の識別子(ファインダのレベルでは表示されません)である文書のファイルタイププロパティによって判断されます。例えば、4D V6で作成された文書のファイルクリエータは「4D06」です。

DocRef：文書参照番号

文書は開いたり、閉じたりできます。内蔵の4Dコマンドを使用すると、文書は一度に1つのプロセスによってのみ開くことができます。1つのプロセスで複数の文書を開くことができ、複数のプロセスではより多くの文書を開くことができますが、同一の文書を一度に2回開くことはできません。

Open document関数、**Create document**関数、**Append document**関数を使用すると、文書が開きます。文書が開くと、その文書から文字を読み取ったり、文字を書き込むことができます(**RECEIVE PACKET**コマンドおよび**SEND PACKET**コマンドを参照してください)。文書での作業を終了したら、通常、**CLOSE DOCUMENT**コマンドを使用して閉じます。

開いているすべての文書は、**Open document**関数、**Create document**関数、**Append document**関数で返されるDocRef式を使用して参照されます。DocRefは、開いている文書をユニークに識別します。これは、正式には時間タイプの式です。開いている文書を操作するすべてのコマンドでは、DocRefが引数（パラメータ）として必要です。このようなコマンドに間違ったDocRefを渡すと、ファイルマネージャエラーが発生します。

I/Oエラーの処理

文書にアクセスしたり(開く、閉じる、削除する、名前を変更する、コピーする)、文書のプロパティを変更したり、文書で文字を読み取ったり書き込んだりすると、I/O エラーが発生する場合があります。文書が見つからない場合には、それがロックされていたり、すでに開かれている場合があります。このようなエラーは、**ON ERR CALL**コマンドでインストールされているエラー処理メソッドを使用して検出することができます。システム文書を使用している時に最も発生しやすいエラーについては、「OSファイルマネージャエラー」の節で説明しています。

Documentシステム変数

Open document関数、**Create document**関数、**Append document**関数の3つを使用すると、標準の「ファイルを開く」または「ファイル保存」ダイアログボックスを使用して文書にアクセスできます。

標準のダイアログを使用して文書にアクセスすると、4Dは Document**システム変数にその文書への完全なパス名を返します。このシステム変数は、コマンドの引数リストに表示される引数<文書>とは区別する必要があります。**

文書名や文書パス名の指定

この節にある、文書名が必要なほとんどのルーチンでは、文書の名前および文書へのパス名の両方を使用できます(*)。名前を渡すと、コマンドはデータベースのフォルダ内にある文書を検索します。パス名を渡す場合には、必ず有効なパス名でなければなりません。間違った名前や間違ったパス名を渡すと、コマンドはファイルマネージャエラーを発生しますが、これは**ON ERR CALL**メソッドを使用して阻止することができます。

(*) その他の方法で指定された場合は、例外です。

警告：引数<文書>の最大長は255文字です。これより長い名前を渡すと、引数名は途中で切り取られ、ファイルマネージャエラーが発生します。

ディスク上にある文書を操作する場合に便利なプロジェクトメソッド

■ 実行しているプラットフォームの検出

4th DimensionにMAP FILE TYPESコマンドのような、プラットフォームの特殊性によるコーディングの変形を排除するためのコマンドがあっても、プログラムによりパス名を取得するようなディスク上の文書を操作する時の下位レベルを対象に開始すると、Macintoshプラットフォームまたは Windowsプラットフォームのどちらを実行しているのか知っている必要があります。

次の「On Windows」プロジェクトメソッドでは、データベースがWindows上で実行されているかどうかを示しています：

```
` 「On windows」プロジェクトメソッド
` On windows → ブール
` On windows → Windowsの場合はTrue
C_BOOLEAN($0)
C_LONGINT($vlPlatform ; $vlSystem ; $vlMachine)
PLATFORM PROPERTIES($vlPlatform ; $vlSystem ; $vlMachine)
$0:=($vlPlatform=Windows)
```

■ 正しいディレクトリ区切り記号の使用

Windowsでは、ディレクトリレベルは円記号(\)で表します。Macintoshでは、フォルダレベルはコロン(:)で表します。実行しているプラットフォームにしたがって、次の「Directory symbol」プロジェクトメソッドでは、正しいディレクトリシンボル(文字)のASCIIコードを返します：

```
` 「Directory symbol」プロジェクトメソッド
` Directory symbol → 整数
` Directory symbol → "/" (Windows) または ":" (MacOS)文字のASCIIコード
C_INTEGER($0)
If (On Windows)
   $0:=Ascii("/")
Else
   $0:=Ascii(":")
End if
```

■ ロングネームからのファイル名の抽出

文書へのロングネーム(パス名＋ファイル名)を取得した後では、例えば、ウィンドウのタイトルに文書のファイル名を表示するために、そのロングネームからファイル名を抽出する必要がある場合があります。「Long name to file name」プロジェクトメソッドは、この処理をWindowsおよびMacintoshの両方で実行します：

```
` 「Long name to file name 」プロジェクトメソッド
` Long name to file name (文字列) → 文字列
` Long name to file name (長いファイル名) → ファイル名
C_STRING(255 ; $1 ; $0)
C_INTEGER($viLen ; $viPos ; $viChar ; $viDirSymbol)
$viDirSymbol:= Directory symbol
$viLen:=Length($1)
$viPos:=0
For ($viChar ; $viLen ; 1 ; -1)
   If (Ascii($1[[$viChar]])=$viDirSymbol)
      $viPos:=$viChar
      $viChar:=0
   End if
End for
If ($viPos>0)
   $0:=Substring($1 ; $viPos+1)
Else
   $0:=$1
End if
If (<>vbDebugOn) `On Startupデータベースメソッドではこの変数をTrueまたはFalseに設定
   If ($0="")
      TRACE
   End if
End if
```

■ ロングネームからのパス名の抽出

文書へのロングネーム(パス名＋ファイル名)を取得した後では、例えば、文書を追加して同じ位置に保存するために、そのロングネームから文書が保存されているディレクトリへのパス名を抽出する必要がある場合があります。「Long name to path name」プロジェクトメソッドは、この処理をWindowsおよびMacintoshの両方で実行します：

```
` 「Long name to path name 」プロジェクトメソッド
` Long name to path name (文字列) → 文字列
` Long name to path name (長い名前) → パス名
C_STRING(255 ; $1 ; $0)
C_STRING(1 ; $vsDirSymbol)
C_INTEGER($viLen ; $viPos ; $viChar ; $viDirSymbol)
$viDirSymbol:= Directory symbol
$viLen:=Length($1)
$viPos:=0
For ($viChar ; $viLen ; 1 ; -1)
   If (Ascii($1[[$viChar]])=$viDirSymbol)
      $viPos:=$viChar
      $viChar:=0
   End if
End for
If ($viPos>0)
   $0:=Substring($1 ; 1 ; $viPos)
Else
   $0:=$1
End if
If (<>vbDebugOn) `On Startupデータベースメソッドではこの変数をTrueまたはFalseに設定
   If ($0="")
      TRACE
   End if
End if
```

参照：Append document、CLOSE DOCUMENT、COPY DOCUMENT、 Create documen、CREATE FOLDER、DELETE DOCUMENT、Document creator、DOCUMENT LIST、Document type、FOLDER LIST、Get document position,、GET DOCUMENT PROPERTIES、Get document size、MAP FILE TYPES、MOVE DOCUMENT、Open document、SET DOCUMENT CREATOR、SET DOCUMENT POSITION、SET DOCUMENT PROPERTIES、SET DOCUMENT SIZE、SET DOCUMENT TYPE、Test path name、VOLUM ATTRIBUTES、 VOLUME LIST

Create document

Create document (文書 ; {形式}) → 文書ファイル参照番号

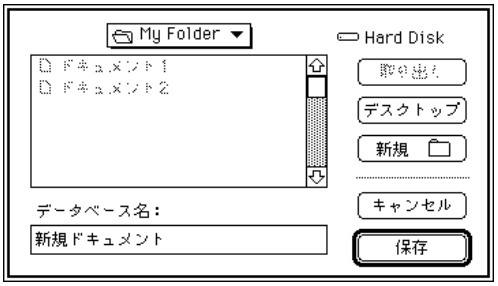
引数	タイプ	説明
文書	文字列	文書ファイル名
形式	文字列	文書形式

Create document関数は、＜文書＞で指定した名前の新しい文書ファイルを作成し、その文書ファイルの文書ファイル参照番号を返します。文書ファイルが既にディスク上に存在する場合には、その文書ファイルは上書きします。この文書ファイルは書き込むために開きます。

＜文書＞が空の文字列(ヌル"")の場合には、「ファイル作成」ダイアログボックスを開きます。ユーザは、ここで新しい文書ファイルの名前を入力することができます。

Windows版

Macintosh版



省略可能な引数＜タイプ＞には、Windows版では、1 から 3文字の Windows のファイル拡張子（つまり、テキスト文書には TXTなど）を指定します。Macintosh版では、ファイルタイプである 4文字の文字列（文書には TEXT、Macintosh のピクチャ文書には PICT 等）を指定します。

このコマンドはさまざまな方法で呼び出すことが可能です。一般的なファイルタイプの例を以下に示します。

テキストファイル：
$DocRef:=Create document ("" ; ".TXT")

により、Windows 上の .TXTファイルとMacintosh のテキスト文書が作られます。
4th Dimension が Macintosh のファイルタイプ TEXT と Windows のファイル拡張子 .TXT を自動的にマッピングしているからです。

例えば、文書名を “ Information ” とした場合、Windows 上では “ Information.TXT ” が作成され、Macintosh 上では “ Information ” が作成されます。

ピクチャファイル：
Macintosh と Windows の両方で目的のファイルタイプのファイルを作成するためには、**MAP FILE TYPES**コマンドが利用できます。「On Startup」データベースメソッドには、次のように書いておきます。

```
MAP FILE TYPES ("PICT" ; "PCT" ; "Macintoshピクチャ文書")
```

この命令文は、Windows の .PCTファイルとMacintosh の PICTファイルのマッピングを設定します。そこで、次のような命令文を使用すると、

```
$DocRef:=Create document ("" ; "PCT")
```

Windows上では .PCT ファイルが作られ、Macintosh 上では PICT ファイルが作られます。

ユーザ定義の拡張子

Macintosh 上でファイル名の一部として拡張子を使用している場合、Windows のファイル拡張子と重複しないように、プラットフォームごとに異なる **Create Document** 文を書く必要があるかもしれません。例えば、Macintosh上で拡張子に「.Prefs」を付けた独自の環境設定文書を使用しているとします。例えば、次のように利用しているとします。

```
$DocRef:=Create document ("Taro.Prefs" ; "TEXT")
```

このような場合、Windowsのファイル拡張子とMacintosh のファイル名の拡張子（これらは 2 種の別個なものです）の間で混同を避けようとすることでしょう。さらに、Windows 上では「.TXT」ファイルではなく「.PRF」ファイルを作るかもしれません。データベースが実行されているプラットフォームを返す **PLATFORM INFO**コマンドを利用し、その後、プラットフォームに応じた **Create Document**関数をコールすればよいのです。次のコードは、実際にこれを行っています。

```
 ` この例では、インタープロセス変数<>Platform は、あらかじめ
 ` データベースの「On Startup」データベースメソッドで GET PLATFORM INFO
 ` コマンドで初期化されているものとして使用します
If(<>Platform=3)
 ` Windows 上で .PRFファイル を作る
 $DocRef:=Create document ("Taro" ; "PRF")
Else
 ` Macintosh 上でテキストファイルを作る
 $DocRef:=Create document ("Taro.Prefs" ; "TEXT")
End if
```

Windows 上

■ 省略可能な＜タイプ＞引数を使用しない場合、4th Dimension では .TXT 文書を想定します。

■ TEXT を指定すると、4th Dimension では自動的に Macintosh の TEXT ファイルタイプと Windows の .TXT 拡張子とを結び付け、.TXT 文書に対するアクセスと見なします。

■ あらかじめ **MAP FILE TYPES**コマンドでマッピングされた Macintosh のファイルタイプを指定すると、4th Dimension では自動的に Macintosh のファイルタイプと Windows の拡張子とを結び付け、そのタイプの Windows 文書を作成するものと見なします。

■ 指定された 4 文字の値がマッピングされていない場合、4th Dimension は値を切り捨てて 3 文字にし、以下で説明している引数として処理します。

■ 1 から 3 文字の値を指定すると、4th Dimension では＜タイプ＞引数を Windows のファイル拡張子として使用し、名前の後ろに付け加えられます。

Macintosh 上

■ 省略可能な＜タイプ＞引数を使用しない場合、4th Dimension では TEXT 文書を想定します。

■ 省略可能な＜タイプ＞引数を使用する場合、4文字の Macintosh ファイルタイプを渡します。

警告：Windows 上で、Create Document関数を使用し、＜文書＞に空の文字列を渡すと、「ファイル保存」ダイアログボックスの「ファイル名」入力エリアで、ファイル拡張子が変更される可能性があります。その場合、省略可能な＜タイプ＞引数に指定したファイル拡張子とはまったく違うファイル拡張子のファイルが作られることになります。そこで、ファイル拡張子を確実に使おうとする場合、作成された文書名を確認することをお勧めします。

```
$DocRef:=Create document ("" ; "TXT")
If (OK=1)
   ` この例では、インタープロセス変数<>Platform は、あらかじめ
   ` データベースの「On Startup」データベースメソッドで GET PLATFORM INFO
   ` コマンドで初期化されているものとして使用します
  If (<>Platform=2)
    If (Substring (Document ; Length (Document) - 3) # ".TEXT")
      ALERT ("テキスト文書を作成してください。")
      CLOSE DOCUMENT ($DocRef)
      DELETE DOCUMENT (Document)
      OK:=0
    End if
  End if
  If (OK=1)
     ` 文書の内容について処理を行う
    CLOSE DOCUMENT ($DocRef)
  End if
End if
```

ユーザが文書ファイルを作成した場合には、システム変数OKに1がセットされます。システム変数Documentには、開いた文書ファイルの名前が代入されます。

それ以外の場合には、システム変数OKに 0が代入されます。文書ファイルが**Create document**関数で作成されない場合は、文書ファイルの参照番号 0を返します。

▼ 次の例は、新しい文書ファイル“ ノート ”を作成し、それに“ こんにちは ”という文字列を書き込み、文書ファイルを閉じます：

```
C_TIME (vDoc)
v文書:=Create document ("ノート")        ` 新しい文書ファイル"ノート"を作成
If (OK=1)
   SEND PACKET (v文書 ; "こんにちは")  ` 文書ファイルに書き込む
   CLOSE DOCUMENT (v文書)              ` 文書ファイルを閉じる
End if
```

文書ファイル“ ノート ”をワープロソフトウェアで開いてみると、“ こんにちは ”という文字列が含まれています。

Open document
Append document

Open document (文書 ; {形式}) → 文書ファイル参照番号
Append document (文書 ; {形式}) → 文書ファイル参照番号

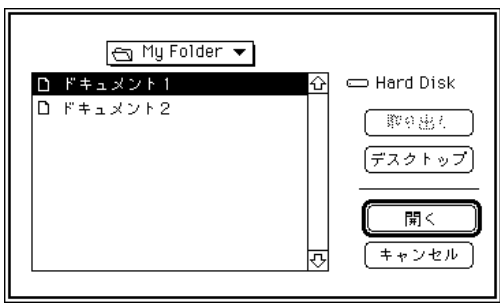
Open document関数は、文書の先頭から読み書きを行うために＜文書＞で指定した既存の文書ファイルを開き、その文書ファイル参照番号を返します。文書に書き出されたデータはその文書の先頭に書き出され、既存データを上書きします。

Append document関数は、＜文書＞で指定した名前の文書ファイルを開き、文書ファイル参照番号を返します。開いた文書ファイルの最後に、データを付け加えます。

どちらの関数でも、文書が空の文字列(ヌル"")の場合には、Windows版では「開く」(Macintosh版では、「ファイルオープン」）ダイアログボックスを開きます。ユーザは、ここで文書ファイルを選択することができます。

Windows版

Macintosh版



省略可能な引数＜タイプ＞には、Windows版では、1 から 3文字の Windows のファイル拡張子（つまり、テキスト文書には TXTなど）を指定します。Macintosh版では、ファイルタイプである 4文字の文字列（文書には TEXT、Macintosh のピクチャ文書には PICT 等）を指定します。

このコマンドはさまざまな方法でコールすることが可能です。以下にその例を示します。

テキストファイル：
$DocRef:=Open document ("" ; "TXT")
または
$DocRef:=Append document ("" ; "TXT")

により、Windows 上の .TXT ファイルとMacintosh のテキスト文書がオープンされます。4th Dimension が Windows のファイル拡張子「.TXT」とMacintosh のファイルタイプ「TEXT」を自動的にマッピングしているからです。

ピクチャファイル：
Macintosh と Windows の両方で目的のファイルタイプのファイルを作成するためには、**MAP FILE TYPES**コマンドが利用できます。「On startup」データベースメソッドには、次のように書いておきます。

MAP FILE TYPES ("PICT" ; "PCT" ; "Macintosh ピクチャ文書")

この命令文は、Macintosh の PICT ファイルと Windows の .PCT ファイルのマッピングを設定します。そこで、次のような命令文を使用すると、

$DocRef:=Open document ("" ; "PCT")
または
$DocRef:=Append document ("" ; "PCT")

Windows 上では .PCT ファイルがオープンされ、Macintosh 上では PICT 文書がオープンされます。4th Dimension がWindows のファイル拡張子「.PCT」と Macintosh のファイルタイプ「PICT」を自動的にマッピングしているからです。

ユーザ定義の拡張子

拡張子を Macintosh 上でファイル名の一部に使用している場合、Windows のファイル拡張子と重複しないように、プラットフォームごとに異なる **Open Document**関数と**Append Document**関数を書く必要があるかもしれません。例えば、Macintosh上で拡張子に「.Prefs」を付けた独自の環境設定文書を使用しているとします。例えば、次のように利用しているとします。

$DocRef:=Create document ("Taro.Prefs" ; "TEXT")
または

$DocRef:=Append document ("Taro DOE.Prefs" ; "TEXT")

このような場合、Windows のファイル拡張子とMacintosh のファイル名の拡張子（これらは 2 種の別個なものです）の間で混同を避けようとすることでしょう。さらに、Windows 上では 「.TXT」ファイルではなく「.PRF」ファイルを作るかもしれません。データベースが実行されているプラットフォームを返す **PLATFORM INFO**コマンドを利用し、その後、プラットフォームに応じた **Create Document**関数をコールすればよいのです。次のコードは、実際にこれを行っています：

```
` この例では、インタープロセス変数<>Platform は、あらかじめ
` データベースの「On Startup」データベースメソッドで GET PLATFORM INFO
` コマンドで初期化されているものとして使用します
If (<>Platform=3)
 ` Windows 上で .PRFファイルをオープンする
 $DocRef:=Open document ("Taro DOE" ; "PRF")
Else
 ` Macintosh上でテキストファイルをオープンする
 $DocRef:=Open document ("Taro DOE.Prefs" ; "TEXT")
End if
```

Windows 上

＜文書＞引数に空の文字列を指定した場合：

■ 省略可能な＜タイプ＞引数を使用しない場合、4th Dimension ではすべてのタイプの文書をオープンすることができます。

■ TEXT を指定すると、4th Dimension では自動的に Macintosh の TEXT ファイルタイプと Windows の .TXT 拡張子とを結び付け、.TXT 文書にアクセスできます。

■ あらかじめ **MAP FILE TYPES**コマンドでマッピングされた Macintosh のファイルタイプを指定すると、4th Dimension では自動的にMacintosh のファイルタイプと Windows の拡張子とを結び付け、そのタイプの Windows 文書にアクセスできます。

■ 指定された 4 文字の値がマッピングされていない場合、4th Dimension は値を切り捨てて 3 文字にし、以下で説明している引数として処理します。

■ 1 から 3 文字の値を指定すると、4th Dimension では＜タイプ＞引数を Windows のファイル拡張子として使用し、名前の後に付け加えます。

＜文書＞引数に名前を指定する場合、拡張子を名前に含まないようにします。ファイル拡張子は＜タイプ＞引数で渡します。＜タイプ＞引数を渡さない場合、4th Dimension では、ファイル拡張子のないファイルへのアクセスか、そのファイルが存在しなければ TXT 文書であると見なします。

Windows 上の例を以下に示します：

$DocRef:=Open document ("Info" ; "TXT")

これは、INFO.TXT ファイルをオープンしようとします。

$DocRef:=Open document ("Info" ; "TXT")

これは、INFOファイルをオープンしようとするか、あるいはファイルが存在しない場合は INFO.TXT をオープンしようとします。

$DocRef:=Open document ("Info.BAT" ; "TXT")

INFO.TXT（拡張子 .BAT は無視され、ファイル名から取り除かれます）をオープンしようとします。

警告：Windows 上で、Create Document関数を使用し、＜文書＞に空の文字列を渡すと、「ファイル保存」ダイアログボックスの「ファイル名」入力エリアで、ファイル拡張子が変更される可能性があります。その場合、省略可能な＜タイプ＞引数に指定したファイル拡張子とはまったく違うファイル拡張子のファイルが作られることになります。そこで、ファイル拡張子を確実に使おうとする場合、オープンされた文書名を確認することをお勧めします。

```
$DocRef:=Open document ("" ; "TXT")
If (OK=1)
   ` この例では、インタープロセス変数<>Platform は、あらかじめ
   ` データベースの 「On Startup」データベースメソッドで GET PLATFORM INFO
   ` コマンドで初期化されているものとして使用します
  If (<>Platform=2)
   If (Substring (Document ; Length (Document) - 3) # ".TEXT")
    ALERT (".テキスト文書をオープンしてください。")
    CLOSE DOCUMENT ($DocRef)
    DELETE DOCUMENT (Document)
    OK:=0
   End if
  End if
  If (OK=1)
     ` 文書の内容について処理を行う
   CLOSE DOCUMENT ($DocRef)
  End if
End if
```

Macintosh 上

＜文書＞引数に空の文字列を指定した場合：

■ 省略可能な＜タイプ＞引数を使用しない場合、4th Dimension ではすべてのタイプの文書をオープンすることができます。

■ 省略可能な＜タイプ＞引数を使用する場合、4文字の Macintosh ファイルタイプを渡します。

＜文書＞引数に名前を指定すると、＜タイプ＞引数は無効になります。

ユーザが文書ファイルを開いた場合は、システム変数OKに1がセットされます。システム変数Documentには、開いた文書ファイルの名前が代入されます。それ以外の場合には、システム変数OKに 0がセットされます。文書が**Open document**関数や**Append document**関数で開くと、文書ファイルの参照番号 0を返します。

▼ 次の例は、文書ファイル“ ノート ”を開き、それに“ さようなら ”という文字列を書き込み、文書ファイルを閉じます。これで、前の内容の“ こんにちは ”は上書きされますので残りません：

```
C_TIME (vDoc)
v文書:=Open document ("ノート")          ` 新しい文書ファイル"ノート"を開く
If (OK=1)
   SEND PACKET (v文書 ; "さようなら")     ` 文書ファイルに書き込む
   CLOSE DOCUMENT (v文書)                ` 文書ファイルを閉じる
End if
```

文書ファイル“ ノート ”をワープロソフトウェアで開いてみると、“ さようなら ”という文字列が含まれています。

▼ 次の例は、文書ファイル“ ノート ”を開き、それに“　また、会いましょう ”という文字列とキャリッジリターンを書き加え、文書ファイルを閉じます。この場合、前の内容“ こんにちは ”が上書きされますので残りません。前の内容“ さようなら ”は、そのまま残ります。従って、文書ファイルの内容は、“ さようなら　また、会いましょう ”とキャリッジリターンになります：

```
C_TIME (vDoc)
文書:=Append document ("ノート")          ` 新しい文書ファイル"ノート"を開く
SEND PACKET (文書;"　また、会いましょう" + Char (13))` 文書ファイルに書き加える
CLOSE DOCUMENT (文書)                     ` 文書ファイルを閉じる
```

CLOSE DOCUMENT

CLOSE DOCUMENT (文書ファイル参照番号)

引数	タイプ	説明
文書ファイル参照番号	文書ファイル参照番号	文書ファイルの参照番号

CLOSE DOCUMENTコマンドは、＜文書ファイル参照番号＞で指定した文書ファイルを閉じます。

文書ファイルを閉じることは、ファイルに書き込んだデータを確実に保存する唯一の方法です。文書ファイルを確実に保存するためには、必ずこのコマンドを使用して閉じなければなりません。

▼ 次の例は、新しい文書ファイル“ ノート ”を作成し、それに“ こんにちは ”という文字列を書き込み、文書ファイルを閉じます：

```
C_TIME (vDoc)
文書:=Create document ("ノート")          ` 新しい文書ファイル"ノート"を作成
If (OK=1)
   SEND PACKET (文書 ; "こんにちは")      ` 文書ファイルに書き込む
   CLOSE DOCUMENT (文書)                  ` 文書ファイルを閉じる
End if
```

DELETE DOCUMENT

DELETE DOCUMENT (文書)

引数	タイプ	説明
文書	文字列	削除する文書ファイル

DELETE DOCUMENTコマンドは、＜文書＞を削除します。文書ファイルの削除に成功するとシステム変数OKに１が代入されます。削除に失敗した場合には、システム変数OKに０が代入されます。**DELETE DOCUMENT**コマンドは、開いている文書ファイルは削除しません。

＜文書＞に空の文字列(ヌル"")を指定することはできません。＜文書＞に空の文字列を指定すると、ダイアログボックスは表示されず、エラーが発生します。

警告：DELETE DOCUMENTコマンドは、他のアプリケーションで作成された文書ファイルも含めたあらゆるファイルを削除することができます。一度削除したファイルは復元できません。

▼次の例は、新しい文書ファイル“ ノート ”を削除します：

```
DELETE DOCUMENT ("ノート")          ` 文書ファイルを削除
```

MAP FILE TYPES

MAP FILE TYPES (MacOS ; Windows ; コンテキスト)

引数	タイプ	説明
MacOS	文字列	Macintosh OS のファイルタイプ（常に 4 文字）
Windows	文字列	Windowsのファイル拡張子（1 から 3 文字）
コンテキスト	文字列	Windowsの「ファイルを開く」ダイアログの「ファイルの種類」ドロップダウンリストに表示される文字列

MAP FILE TYPESコマンドによって、Windows のファイル拡張子と Macintosh のファイルタイプを関連づけることができます。

このルーチンを 1 回コールするだけで、データベースによるセッションの作業全体についてマッピングを設定することができます。コールが行われると、Windows 上のシステム文書に関する 4th Dimension のコマンドのコールで、実際に引数としてルーチンに渡された Macintosh のファイルタイプは、自動的に Windows の拡張子で置き換えられます。

引数＜MacOS＞には、4文字の Macintosh のファイルタイプを渡します。4 文字の文字列を渡さなければ、コマンドは動作せず、エラーが発生します。

引数＜Windows の＞には、1から 3文字の Windows のファイル拡張子を渡します。1 から 3 文字の文字列を渡さなければ、コマンドは動作せず、エラーが発生します。

引数＜コンテキスト＞には、Windowsの「ファイルを開く」ダイアログの「ファイルの種類」ドロップダウンリストに表示される文字列を渡します。コンテキストの文字列は 32 文字以内に制限されており、それ以上の文字は無視されます。

重要：Windows のファイル拡張子と Macintosh のファイルタイプとのマッピングを行うと、その作業セッション中はマッピングを変更したり削除できません。4D アプリケーションの開発とデバッグの際に変更する必要がある場合は、データベースを開き直して、再度ファイル拡張子のマッピングを行います。

▼ 次に示す 4th Dimension の簡単なコード（「On startup」データベースメソッドなどで使用します）は、Macintosh の MS-Word ファイルタイプ “ WDBN ” を Windows のファイル拡張子 “ DOC ” にマッピングします：

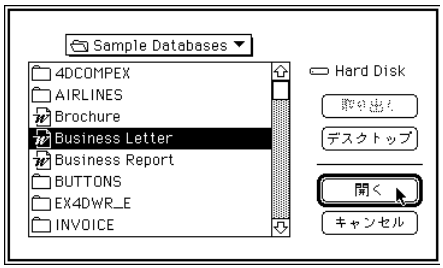
```
MAP FILE TYPES ("WDBN" ; "DOC" ; "Word documents")
```

前記のコールが実行された後、次に示すコードを実行すると、Macintosh と Windowsの「ファイルを開く」ダイアログには Word の文書が表示されます：

```
$DocRef:=Open document ("" ; "WDBN")
If (OK=1)
   ...
End if
```

Windows版 (Windows NTの画面)

Macintosh版



COPY DOCUMENT

COPY DOCUMENT (コピー元 ; コピー先 {; *})

引数	タイプ		説明
コピー元	文字列	→	コピーする文書の名前
コピー先	文字列	→	コピーされる文書の名前
*		→	既存の文書を上書き

COPY DOCUMENTコマンドは、引数＜コピー元＞で指定された文書を＜コピー先＞で指定された位置にコピーします。

＜コピー元＞および＜コピー先＞は共に、データベースフォルダにある文書を表す名前、または文書をボリュームのルートレベルに相対的に表すパス名です。

＜コピー先＞という名前の文書がすでに存在する場合には、**COPY DOCUMENT**コマンドがコピー先文書を削除するか上書きすることを指示するオプション引数＜ * ＞を指定しなければ、エラーが発生します。

例：

1. 次の例では、文書をその文書のあるフォルダ内で複製します：

```
COPY DOCUMENT("C:\FOLDER\DocName" ; "C:\FOLDER\DocName2")
```

2. 次の例では、文書をデータベースフォルダにコピーします(C:\FOLDERがデータベースフォルダではない場合に限ります)：

```
COPY DOCUMENT("C:\FOLDER\DocName" ; "DocName")
```

3. 次の例では、ある文書をあるボリュームから別のボリュームへコピーします：

```
COPY DOCUMENT("C:\FOLDER\DocName" ; "F:\Archives\DocName.OLD")
```

4. 次の例では、文書をその文書のあるフォルダ内で、すでに存在する別のコピーを上書きして複製します：

```
COPY DOCUMENT("C:\FOLDER\DocName" ; "C:\FOLDER\DocName2" ; *)
```

参照：MOVE DOCUMENT

Test path name

Test path name (パス名) → 数値

引数	タイプ		説明
パス名	文字列	→	ディレクトリ、フォルダ、文書へのパス名
関数の返す値	数値	←	1=パス名は既存の文書を表す 0=パス名は既存のディレクトリまたはフォルダを表す <0=無効なパス名、OSファイルマネージャエラー

Test path name関数は、引数＜パス名＞に渡された名前またはパス名を持つ文書またはフォルダが、ディスク上に存在するかどうかをチェックします。

文書が見つかれば、**Test path name**関数は 1を返します。フォルダが見つかれば、**Test path name**関数は 0を返します。

4Dには、次のようなあらかじめ定義された定数があります。

定数	タイプ	値
Is a documen	倍長整数	1
Is a directory	倍長整数	0

文書もフォルダも見つからない場合には、**Test path name**関数は負の値を返します (つまり、ファイルが見つからない場合には -43になります)。

▼ 次の例では、「Journal」という文書がデータベースのフォルダにあるかどうかをテストし、見つからない場合にはこれを作成します：

```
If (Test path name("Journal")  #  Is a document)
   $vhDocRef:=Create document("Journal")
   If (OK=1)
      CLOSE DOCUMENT($vhDocRef)
   End if
End if
```

参照：Create document、CREATE FOLDER

CREATE FOLDER

CREATE FOLDER (パス名)

引数	タイプ		説明
パス名	文字列	→	作成する新しいフォルダへのパス名

CREATE FOLDERコマンドは、引数＜パス名＞に渡すパス名にしたがって、フォルダを作成します。

名前を渡すと、データベースのフォルダにフォルダが作成されます。パス名を渡す場合には、作成したいフォルダの名前までの、ボリュームのルートレベルまたはデータベースフォルダのレベルから始まる有効なパス名を表していなければなりません。

例：

1. 次の例では、「Archives」というフォルダをデータベースのフォルダ内に作成します：

```
CREATE FOLDER("Archives")
```

2. 次の例では、「Archives」というフォルダをデータベースのフォルダ内に作成し、「January」および「February」というサブフォルダを作成します：

```
CREATE FOLDER("Archives")
CREATE FOLDER("Archives\January")
CREATE FOLDER("Archives\February")
```

3. 次の例では、「Archives」というフォルダをCボリュームのルートレベルに作成します：

```
CREATE FOLDER("C:\Archives")
```

4. Cボリュームのルートレベルに「NewStuff」というフォルダがない場合には、次の例は失敗します：

```
CREATE FOLDER("C:\NewStuff\Pictures")`1回の呼び出しで2つのフォルダは作成できない
```

参照：FOLDER LIST、Test path name

VOLUME LIST

VOLUME LIST (ボリューム)

引数	タイプ		説明
ボリューム	配列	→	現在マウントされているボリュームの名前

VOLUME LISTコマンドは、テキスト配列または文字列配列の＜ボリューム＞に、現在コンピュータに定義されているボリュームの名前(Windowsの場合)またはマウントされているボリュームの名前(Macintoshの場合)を割り当てます。

Macintoshでは、ファインダレベルで見ることのできるボリュームのリストが返ります。

それ以外の場合、つまりWindowsでは、現在定義されているボリュームが、それぞれのボリュームが物理的に存在するかどうかに関係なく返ります(つまり、フロッピーディスクドライブに実際にディスクが入っているかどうかに関係なく、ボリュームAが返ります)。

例：

「asVolumes」というスクロール可能エリアを使用して、コンピュータに定義またはマウントされているボリュームのリストを表示するには、次のように記述します：

```
Case of
   \ (Form event=On Load)
      ARRAY STRING(31 ; asVolumes ; 0)
      VOLUME LIST(asVolumes)
      ` ...
End case
```

参照：DOCUMENT LIST、FOLDER LIST、VOLUME ATTRIBUTES

VOLUME ATTRIBUTES

VOLUME ATTRIBUTES (ボリューム ; サイズ ; 使用サイズ ; 空きサイズ)

引数	タイプ		説明
ボリューム	文字列	→	ボリュームの名前
サイズ	数値	←	ボリュームのサイズ（バイト単位）
使用サイズ	数値	←	使用サイズ（バイト単位）
空きサイズ	数値	←	空きサイズ（バイト単位）

VOLUME ATTRIBUTESコマンドは、引数＜ボリューム＞に渡した名前を持つボリュームの使用サイズおよび空きサイズを、バイト単位で表して返します。

例：

このアプリケーションには、夜間や週末に実行するディスク上に大規模な中間ファイルを格納するバッチ処理のいくつかが含まれています。このプロセスをできるだけ自動的にまた柔軟にするには、空きサイズが中間ファイルを格納するために十分である最初のボリュームを自動的に見つけるルーチンを作成します。例えば、次のようなプロジェクトメソッドを作成します：

```
` 「Find volume for space」プロジェクトメソッド
` Find volume for space (倍長整数) → 文字列
` Find volume for space (必要なサイズ ) → ボリューム名または空の文字列
C_STRING(31 ; $0)
C_STRING(255 ; $vsDocName)
C_LONGINT($1 ; $vlNbVolumes ; $vlVolume ; $vlSize ; $vlUsed ; $vlFree)
C_TIME($vhDocRef)
`関数の返す値を初期化する
$0:=""
`エラー阻止メソッドを使用して、すべてのI/O処理を保護する
ON ERR CALL("エラーメソッド")
` ボリュームのリストを取得する
ARRAY STRING(31 ; $asVolumes ; 0)
gError:=0
VOLUME LIST($asVolumes)
If (gError=0)
    `Windowsで実行する場合には(通常の)2つのフロッピーディスクドライブは省略する
    If (On Windows)
        $vlVolume:=Find in array($asVolumes ; "A:")
        If ($vlVolume>0)
            DELETE ELEMENT($asVolumes ; $vlVolume)
```

```
        End if
        $vlVolume:=Find in array($asVolumes ; "B:")
        If ($vlVolume>0)
            DELETE ELEMENT($asVolumes ; $vlVolume)
        End if
    End if
    $vlNbVolumes:=Size of array($asVolumes)
    ` For each volume
    For ($vlVolume ; 1 ; $vlNbVolumes)
        `使用サイズおよび空きサイズを取得する
        gError:=0
        VOLUME ATTRIBUTES($asVolumes{$vlVolume} ; $vlSize ; $vlUsed ; $vlFree)
        If (gError=0)
            `空きサイズは十分ありますか(必要なサイズに32KBを加えたサイズ)?
            If ($vlFree>=($1+32768))
                ` 十分であれば、ボリュームがアンロックされているかどうかをチェック...
                $vsDocName:=$asVolumes{$vlVolume}+Char( Directory symbol )
                    +"XYZ"+String(Random)+".TXT"
                $vhDocRef:=Create document($vsDocName)
                If (OK=1)
                    CLOSE DOCUMENT($vhDocRef)
                    DELETE DOCUMENT($vsDocName)
                    `すべて問題がなければ、ボリュームの名前を返す
                    $0:=$asVolumes{$vlVolume}
                    $vlVolume:=$vlNbVolumes+1
                End if
            End if
        End if
    End for
End if
ON ERR CALL("")
```

このプロジェクトメソッドがアプリケーションに追加されると、例えば、次のように記述することができます：

```
$vsVolume:= Find volume for space (100*1024*1024)
If($vsVolume#"")
   ` Continue
Else
   ALERT("少なくとも100 MBの空き空間を持つボリュームが必要です!")
End if
```

参照：VOLUME LIST

FOLDER LIST

FOLDER LIST (パス名 ; ディレクトリ)

引数	タイプ		説明
パス名	文字列	→	ボリューム、ディレクトリまたはフォルダへのパス名
ディレクトリ	配列	←	この位置にあるディレクトリの名前

FOLDER LISTコマンドは、テキスト配列または文字列配列のディレクトリを、引数<パス名>に渡すパス名に示されているフォルダに配置します。

指定した位置にフォルダがない場合には、このコマンドは空の配列を返します。<パス名>に渡すパス名が無効な場合には、**FOLDER LIST**コマンドはファイルマネージャエラーを生成します。このエラーの生成は、**ON ERR CALL**メソッドを使用すると中止することができます。

警告：引数<パス名>の最大長は 255文字です。これより長いパス名を渡すと、パス名は途中でとぎれ、ファイルマネージャエラーが発生します。

参照：DOCUMENT LIST、VOLUME LIST

DOCUMENT LIST

DOCUMENT LIST (パス名 ; 文書)

引数	タイプ		説明
パス名	文字列	→	ボリューム、ディレクトリまたはフォルダへのパス名
サイズ	配列	←	この位置にある文書の名前

DOCUMENT LISTコマンドは、テキスト配列または文字列配列のディレクトリを、引数<パス名>に渡すパス名に示されている文書に配置します。

指定した位置に文書がない場合には、このコマンドは空の配列を戻します。<パス名>に渡すパス名が無効な場合には、**DOCUMENT LIST**コマンドはファイルマネージャエラーを生成します。このエラーの生成は、**ON ERR CALL**メソッドを使用すると中止することができます。

警告：引数<パス名>の最大長は 255文字です。これより長いパス名を渡すと、パス名は途中でとぎれ、ファイルマネージャエラーが発生します。

参照：DIRECTORY LIST、FOLDER LIST、VOLUME LIST

SET DOCUMENT TYPE

SET DOCUMENT TYPE (文書 ; ファイルタイプ)

引数	タイプ		説明
文書	文字列	→	文書の名前
ファイルタイプ	文字列	→	Windowsファイル拡張子 (1~3バイト) MacOSファイルタイプ (4バイト)

SET DOCUMENT TYPEコマンドは、文書の名前またはパス名を引数<文書>に渡す文書ファイルのタイプを設定します。

文書の新しいタイプは、引数<ファイルタイプ>に渡します。

前述の「文書ファイルのタイプとクリエータ」の節を参照してください。

参照：Document type、MAP FILE TYPES、SET DOCUMENT CREATOR、SET DOCUMENT PROPERTIES

SET DOCUMENT CREATOR

SET DOCUMENT CREATOR (文書 ; ファイルクリエータ)

引数	タイプ		説明
文書	文字列	→	文書の名前
ファイルクリエータ	文字列	→	MacOSファイルクリエータ (4バイト) または、空の文字列（Windows）

SET DOCUMENT CREATORコマンドは、文書の名前またはパス名を引数＜文書＞に渡す文書ファイルのクリエータを設定します。

文書の新しいクリエータは、引数＜ファイルクリエータ＞に渡します。

このコマンドは、Windowsでは無効です。

前述の「文書ファイルのタイプとクリエータ」の節を参照してください。

参照：Document creator、SET DOCUMENT PROPERTIES、SET DOCUMENT TYPE

GET DOCUMENT PROPERTIES

GET DOCUMENT PROPERTIES (文書 ; ロック ; 非表示 ; 作成日 ; 作成時間 ; 更新日 ; 更新時間)

引数	タイプ		説明
文書	文字列	→	文書の名前
ロック	ブール	←	ロックの場合はTrue、アンロックの場合はFalse
非表示	ブール	←	非表示の場合はTrue、表示の場合はFalse
作成日	日付	←	作成日
作成時間	時間	←	作成時間
更新日	日付	←	更新日
更新時間	時間	←	更新時間

GET DOCUMENT PROPERTIESコマンドは、文書の名前またはパス名を引数＜文書＞に渡す文書についての情報を返します。

呼び出し後：

■ 引数＜ロック＞ は、文書がロックされていれば True を返します。ロックされている文書を開いたり削除することはできません。

■ 引数＜非表示＞は、文書が隠されていれば True を戻します。

■ 引数＜作成日＞および ＜作成時間＞により、文書が作成された日付と時間が返ります。

■ 引数＜更新日＞および ＜更新時間＞により、文書が更新された日付と時間が返ります。

例：

文書データベースを作成し、そのデータベースに作成したすべてのレコードをディスク上の文書にデータを書き出す場合を想定します。データベースは定期的にアップデートされたため、文書が存在しなかったり、文書が最後に保存された後でレコードが更新されていた場合には、各文書を作成したり、再作成するような、データ書き出しのアルゴリズムを記述しようとしています。このため、文書(存在する場合には) の更新日付と時間を対応するレコードと比較する必要があります。この例を図解するために、次のような表を使用して定義を表しています。

各レコードに日付と時間の両方を保存するのではなく、任意の以前の日付および時間と、レコードが保存された日付および時間との間の経過秒数を示す「time stamp」の値を保存することができます (この例では、Sep, 1st 1997 at 00:00:00、つまり 1997 年 9月 1日 午前0時を使用しています)。

この例では、[文書]作成日 フィールドにレコードが最初に作成された時のタイムスタンプがあり、[文書]更新日フィールドにレコードが最後に更新された時のタイムスタンプがあります。

この後に示されている「Time stamp」プロジェクトメソッドでは、パラメータが渡されない場合には、現在の日付と時間を特定の日付と時間としてタイムスタンプを計算します：

```
` 「Time stamp」プロジェクトメソッド
` Time stamp { (日付 ; 時間) } → 倍長整数
` Time stamp { (日付 ; 時間) } → 1997 年 9月 1日から経過した秒数
C_DATE($1 ; $vdDate)
C_TIME($2 ; $vhTime)
C_LONGINT($0)
If (Count parameters=0)
   $vdDate:=Current date
   $vhTime:=Current time
Else
   $vdDate:=$1
   $vhTime:=$2
End if
$0:=(($vdDate-!97/09/01!)*86400)+$vhTime
```

注：このメソッドを使用すると、日付と時間を 97/09/01 at 00:00:00 から 2063/01/19at 03:14:07 の間でコード化することができるため、倍長整数の 0~2~31 - 1 の範囲に対応できます。

一方、次に示されている「Time stamp to date」および「Time stamp to time」のプロジェクトメソッドでは、タイムスタンプに保存されている日付および時間を抽出することができます：

```
` 「Time stamp to date」プロジェクトメソッド
` Time stamp to date (倍長整数) → 日付
` Time stamp to date (Time stamp) → 抽出する日付
C_DATE($0)
C_LONGINT($1)
$0:=!97/09/01!+($1//86400)
```

```
` 「Time stamp to time」プロジェクトメソッド
` Time stamp to time (倍長整数) → 日付
` Time stamp to time (Time stamp ) → 抽出する日付
C_TIME($0)
C_LONGINT($1)
$0:=Time(Time string(?00:00:00?+($1%86400)))
```

レコードの作成や更新の方法に関係なく、レコードのタイムスタンプが、正しく更新されるようにするには、[文書]テーブルのトリガを使用して、その規則を強制すればよいだけです：

```
` [文書]テーブルのトリガ
Case of
   \ (Database event=Save New Record Event)
      [文書]作成日:= Time stamp
      [文書]更新日:= Time stamp
   \ (Database event=Save Existing Record Event)
      [文書]更新日:= Time stamp
End case
```

この内容がデータベースに実現されれば、次に示されている「CREATE DOCUMENTATION」プロジェクトメソッドを作成するために必要なすべての準備が整います。文書の作成および更新の日付および時間の処理には、**GET DOCUMENT PROPERTIES**コマンドおよび **SET DOCUMENT PROPERTIES** コマンドを使用します：

```
` 「CREATE DOCUMENTATION 」プロジェクトメソッド
C_STRING(255 ; $vsPath ; $vsDocPathName ; $vsDocName)
C_LONGINT($vlDoc)
C_BOOLEAN($vbOnWindows ; $vbDoIt ; $vbLocked ; $vbInvisible)
C_TIME($vhDocRef ; $vhCreatedAt ; $vhModifiedAt)
C_DATE($vdCreatedOn ; $vdModifiedOn)
If (Application type=4D Client)
    ` 4D Clientを実行している場合には、4D Clientが存在する クライアントマシンに
    `文書をローカルに保存します。
    $vsPath:= Long name to path name (Application file)
Else
    `それ以外の場合には、データファイルが存在する位置に文書を保存する
    $vsPath:= Long name to path name (Data file)
End if
    `任意で "Documentation" と命名したディレクトリにある文書を保存する
$vsPath:=$vsPath+"Documentation"+Char(Directory symbol )
    `このディレクトリが存在しない場合には、作成する
```

```
If (Test path name($vsPath)   #   Is a directory)
   CREATE FOLDER($vsPath)
End if
   `古い文書、つまり、対応するレコードが削除されている文書は削除しなければ
   `ならないため、すでにある文書のリストを作成する
ARRAY STRING(255 ; $asDocument ; 0)
DOCUMENT LIST($vsPath ; $asDocument)
   ` 部門テーブルのレコードを選択する
ALL RECORDS([文書])
   ` 各レコードに対して
$vlNbRecords:=Records in selection([文書])
$vlNbDocs:=0
$vbOnWindows:= On Windows
For ($vlDoc ; 1 ; $vlNbRecords)
   `ディスクにドキュメントを (再) 作成しなければならないと想定する
   $vbDoIt:=True
   `文書の名前およびパス名を求める
   $vsDocName:="DOC"+String([文書]番号 ; "00000")
   $vsDocPathName:=$vsPath+$vsDocName
   `この文書はすでに存在するか？
   If (Test path name($vsDocPathName+".HTM")=Is a document)
   ` その場合には、文書のリストから文書を除去します。
   ` このようにすると、文書が削除される場合があります。
      $vlElem:=Find in array($asDocument ; $vsDocName+".HTM")
      If ($vlElem>0)
         DELETE ELEMENT($asDocument ; $vlElem)
      End if
      ` レコードが最後に更新された後で、文書が保存されたか?
      GET DOCUMENT PROPERTIES($vsDocPathName+".HTM" ; $vbLocked ;
      $vbInvisible ; $vdCreatedOn ; $vhCreatedAt ; $vdModifiedOn ; $vhModifiedAt)
      If (Time stamp ($vdModifiedOn ; $vhModifiedAt) >= [文書]更新日)
         `その場合には、ドキュメントを再作成する必要はない
         $vbDoIt:=False
      End if
   Else
      `文書は存在しないため、これら 2つの変数をリセットし、文書の最終的な
      `プロパティを設定する前にこれらを計算する必要があることを確認できる
      `ようにする
      $vdModifiedOn:=!00/00/00!
      $vhModifiedAt:=?00:00:00?
   End if
   ` ドキュメントを (再) 作成する必要があるか?
```

```
    If ($vbDoIt)
        `その場合には、更新された文書の数を増分する
        $vlNbDocs:=$vlNbDocs+1
        `ドキュメントがすでに存在する場合には、削除する
        DELETE DOCUMENT($vsDocPathName+".HTM")
        `再度、作成する
        If ($vbOnWindows)
            $vhDocRef:=Create document($vsDocPathName ; "HTM")
        Else
            $vhDocRef:=Create document($vsDocPathName+".HTM")
        End if
        If (OK=1)
            ` ...
            ` ここで、文書の内容を記述する
            ` ...
            CLOSE DOCUMENT($vhDocRef)
            If ($vdModifiedOn=!00/00/00!)
                文書は存在しなかったため、更新の日付および時間を正しい値に設定する
                $vdModifiedOn:=Current date
                $vhModifiedAt:=Current time
            End if
            `文書のプロパティを変更し、作成の日付および時間が対応するレコードと
            `等しくなるようにする
            SET DOCUMENT PROPERTIES($vsDocPathName+".HTM" ; $vbLocked
            ; $vbInvisible ; Time stamp to date ([文書]作成日) ;
            Time stamp to time ([文書]作成日) ; $vdModifiedOn ; $vhModifiedAt)
        End if
    End if
    ` 現在実行中の処理を確認するために
    SET WINDOW TITLE("文書処理中："+String($vlDoc)+"/ "+String($vlNbRecords))
    NEXT RECORD([文書])
End for
    ` 古い文書、つまりまだ $asDocument 配列内にある文書を削除する
For ($vlDoc ; 1 ; Size of array($asDocument))
    DELETE DOCUMENT($vsPath+$asDocument{$vlDoc})
    SET WINDOW TITLE("文書削除中： "+Char(34)+$asDocument{$vlDoc}+Char(34))
End for
    ` We're done
ALERT("処理された文書の数："+String($vlNbRecords)+Char(13)+
    "更新された文書の数： "+String($vlNbDocs)+Char(13)+
    "削除された文書の数： "+String(Size of array($asDocument)))
```

参照：Document creator、Document type、SET DOCUMENT PROPERTIES

SET DOCUMENT PROPERTIES

SET DOCUMENT PROPERTIES (文書 ; ロック ; 非表示 ; 作成日 ; 作成時間 ; 更新日 ; 更新時間)

引数	タイプ		説明
文書	文字列	→	文書の名前
ロック	ブール	→	ロックの場合はTrue、アンロックの場合はFalse
非表示	ブール	→	非表示の場合はTrue、表示の場合はFalse
作成日	日付	→	作成日
作成時間	時間	→	作成時間
更新日	日付	→	更新日
更新時間	時間	→	更新時間

SET DOCUMENT PROPERTIESコマンドは、文書の名前またはパス名を引数＜文書＞に渡す文書についての情報を変更します。

呼び出しの前に：

■ 引数＜ロック＞に Trueを渡し、文書をロックします。ロックされた文書を、開いたり削除することはできません。文書のロックを解除するには＜ロック＞に False を渡します。

■ 引数＜非表示＞に Trueを渡し、文書を隠します。デスクトップウィンドウで文書が見えるようにするには、＜非表示＞に Falseを渡します。

■ 引数＜作成日＞および＜作成時間＞に、文書の作成日および作成時難を渡します。

■ 引数＜更新日＞および＜更新時間＞に、最新の文書更新日および更新時間を渡します。

作成および最新の更新の日付および時間は、文書を作成したり、これにアクセスするつど、システムのファイルマネージャによって管理されます。このコマンドを使用すると、特別な用途のためにこれらのプロパティを変更することができます。**GET DOCUMENT PROPERTIES**コマンドの例を参照してください。

参照：GET DOCUMENT PROPERTIES、SET DOCUMENT CREATOR、SET DOCUMENT TYPE

Get document size

Get document size (文書 {; *}) → 数値

引数	タイプ		説明
文書	文書参照番号 または、文字列	→	文書参照番号 または、文書の名前
*			MacOSのみ： 省略した場合、データフォークのサイズ 指定した場合、リソースフォークの場合
関数の返す値	数値	←	文書のサイズ（バイト単位）

Get document size関数は、文書のサイズをバイト単位で表示して返します。

文書が開かれている場合には、引数＜文書＞に文書参照番号を渡します。文書が開かれていない場合には、＜文書＞に文書の名前またはパス名を渡します。

Macintoshでは、オプション引数＜ * ＞を渡さないと、データフォークのサイズが返ります。＜ * ＞を渡すと、リソースフォークのサイズが返ります。

参照：Get document position、SET DOCUMENT POSITION、SET DOCUMENT SIZE

SET DOCUMENT SIZE

SET DOCUMENT SIZE (文書 ; サイズ)

引数	タイプ		説明
文書	文書参照番号 または、文字列	→	文書参照番号 または、文書の名前
サイズ	数値	→	新しいサイズ（バイト単位）

SET DOCUMENT SIZEコマンドは、引数＜サイズ＞に渡すド文書のサイズをバイト単位で設定します。

文書が開かれている場合には引数＜文書＞に文書参照番号を渡します。文書が開かれていない場合には、＜文書＞に文書の名前またはパス名を渡します。

Macintosh では、文書データフォークのサイズが変わります。

参照：Get document position、Get document size、SET DOCUMENT POSITION

Get document position

Get document position (文書参照番号) → 数値

引数	タイプ		説明
文書参照番号	文書参照番号	→	文書参照番号
関数の返す値	数値	←	文書開始位置からのファイル位置(バイト単位)

この関数は、文書参照番号を引数＜文書参照番号＞に渡す現在開かれている文書でのみ有効です。

Get document position関数は、文書の最初から見て、次の読み込み(**RECEIVE PACKET**コマンド)または書き込み(**SEND PACKET**コマンド)が発生する位置を返します。

参照：RECEIVE PACKET、SEND PACKET、SET DOCUMENT POSTION

SET DOCUMENT POSITION

SET DOCUMENT POSITION (文書参照番号 ; オフセット {; アンカー})

引数	タイプ		説明
文書参照番号	文書参照番号	→	文書参照番号
オフセット	数値	→	ファイルの位置（バイト単位）
アンカー		→	1=ファイルの最初に相対的 2=ファイルの最後に相対的 3=現在の位置に相対的

このコマンドは、文書参照番号を引数＜文書参照番号＞に渡す現在開かれている文書でのみ有効です。

SET DOCUMENT POSITIONコマンドは、引数＜オフセット＞に渡す、次の読み込み(**RECEIVE PACKET**コマンド) または書き込み (**SEND PACKET**コマンド) が発生する位置を設定します。

オプション引数＜アンカー＞を省略すると、位置は文書の最初に相対的になります。＜アンカー＞を指定すると、ここにリストされている値のうちいずれかを渡します。

アンカーによっては、引数＜オフセット＞に正の値または負の値を渡すことができます。

参照：Get document position、RECEIVE PACKET、SEND PACKET

MOVE DOCUMENT

MOVE DOCUMENT (移動元パス名 ; 移動先パス名)

引数	タイプ		説明
移動元パス名	文字列	→	既存の文書への完全なパス名
移動先パス名	文字列	→	移動先のパス名

MOVE DOCUMENTコマンドは、文書を移動したり、名前を変更します。

引数＜移動元パス名＞には文書への完全なパス名を指定し、＜移動先パス名＞には文書の新しい名前または新しい位置を指定します。.

警告：MOVE DOCUMENTコマンドを使用すると、文書を同じボリューム上のどのディレクトリにも移動することができます。文書を 2つの異なるボリュームの間で移動するには、COPY DOCUMENTコマンドを使用して文書を「移動」してからDELETE DOCUMENTコマンドを使用して、オリジナルの文書を削除します。

例：

1. 次の例では、文書「DocName」の名前を変更します：

```
MOVE DOCUMENT("C:\FOLDER\DocName" ; "C:\FOLDER\NewDocName")
```

2. 次の例では、文書「DocName」を移動し、名前を変更します：

```
MOVE DOCUMENT("C:\FOLDER1\DocName" ; "C:\FOLDER2\NewDocName")
```

3. 次の例では、文書「DocName」を移動しています：

```
MOVE DOCUMENT("C:\FOLDER1\DocName" ; "C:\FOLDER2\DocName")
```

注：最後の 2つの例では、移動先のフォルダ "C:\FOLDER2" が存在しなければなりません。MOVE DOCUMENTコマンドは文書を移動するだけで、フォルダは作成しません。

参照：COPY DOCUMENT

Document type

Document type (文書) → 文字列

引数	タイプ		説明
文書	文字列	→	文書の名前
関数の返す値	文字列	←	Windowsファイル拡張子（1~3バイト）、または MacOSファイルタイプ（4バイト）

Document type関数は、文書の名前またはパス名を引数＜文書＞に渡す文書のタイプを返します。

Windows では、**Document type**関数は文書のファイル拡張子 (つまり Microsoft Word の場合には "DOC"、実行可能ファイルの場合には "EXE" など) を返し、4th Dimension または**MAP FILE TYPES**コマンドへの呼び出しによって、Windows ファイル拡張子に相当する MacOS ベースの 4文字からなるファイルタイプ (つまり "TXT" ファイル拡張子には "TEXT") がマップされていればこのようなファイルタイプを返します。

Macintosh では、**Document type**関数は、4文字からなる文書のファイルタイプ (つまりテキスト文書の場合には "TEXT"、ダブルクリック可能なアプリケーションの場合には "APPL" など) を返します。

参照：Document creator、GET DOCUMENT PROPERTIES、MAP FILE TYPES、SET DOCUMENT TYPE

Document creator

Document creator (文書) → 文字列

引数	タイプ		説明
文書	文字列	→	文書の名前、または文書の完全なパス名
関数の返す値	文字列	←	空の文字列（Windows）、または ファイルクリエータ（MacOS）

Document creator関数は、文書の名前またはパス名を引数＜文書＞に渡す文書のクリエータを返します。

Windows では、**Document creator**関数は空文字列を返します。

参照：Document type、SET DOCUMENT CREATOR

システム環境コマンド　53

この章では、「ルーチン」エディタの「System Environment」テーマ内にあるシステム環境コマンドについて説明します。

Count screen
Corrent machine
Current machine owner
FONT LIST
Font name
Font number
Gestalt
Menu bar height
Menu bar screen
SCREEN COORFINATES
SCREEN DEPTH
Screen Height
Screen Width
SET SCREEN DEPTH
System folder
Temporary folder

Screen height
Screen width

Screen height → 数値
Screen width → 数値

引数	タイプ		説明
*	数値	→	Windows：<*>が指定されている場合はアプリケーションウインドウの高さまたは画面の高さ Macintosh：メイン画面の高さ
関数の返す値	数値	←	ピクセル数で表される高さ

Screen height関数と**Screen width**関数は、画面の高さと幅をピクセル数で返します。

注：4th Dimensionバージョン6から Macintosh と Windows の両方において、Screen width関数と Screen Height関数はそれぞれ、メインのスクリーン装置の幅と高さ（ピクセル単位）を返します。4th Dimension 6.0には、省略可能な * 引数が導入されています。Macintosh 上で * 引数を渡しても、2つの関数からは同じ値が返されます。Windows 上では、* 引数を渡すと、2つの関数はそれぞれ、4th Dimensionアプリケーションの MDI ウインドウの幅と高さ（ピクセル単位）を返します。

参照：SCREEN COORDINATES

Count screen

Count screen → 数値

引数	タイプ		説明
この関数には、引数はありません。			
関数の返す値	数値	←	画面の数

Count screens関数は、マシンに接続されている画面モニタの数を返します。

Windowsでの注意： Windowsでは、**Count screens**関数は常に 1を返します。

参照：Menu bar screen、SCREEN COORDINATES、SCREEN DEPTH、Screen height、Screen width

SCREEN COORDINATES

SCREEN COORDINATES (左 ; 上 ; 右 ; 下 { ; 画面})

引数	タイプ		説明
左	数値	←	画面エリアの左端のグローバル座標
上	数値	←	画面エリアの上端のグローバル座標
右	数値	←	画面エリアの右端のグローバル座標
下	数値	←	画面エリアの下端のグローバル座標
画面	数値	→	画面番号、または省略された場合には主画面(メインスクリーン)

SCREEN COORDINATESコマンドは、＜画面＞に指定した画面のグローバル座標を引数＜左＞、＜上＞、＜右＞、および＜下＞に返します。

Windowsの場合

通常、引数＜画面＞は指定しません。

Macintoshの場合

引数＜画面＞を省略した場合、このコマンドは主画面(メインスクリーン)、つまりメニューバーが表示されている画面の座標を返します。

参照：Count screen、Menu bar screen、SCREEN DEPTH

SCREEN DEPTH

SCREEN DEPTH (深さ ; カラー {; 画面})

引数	タイプ		説明
深さ	数値	←	画面の深さ(カラーの数 = 2 ^ 深さ)
カラー	数値	←	1、カラー画面 0、白黒またはグレイスケール
画面	数値	→	画面番号、または省略された場合には主画面(メインスクリーン)

SCREEN DEPTHコマンドは、モニタについての情報を引数＜深さ＞と＜カラー＞に返します。

呼び出しの後：

■ 画面の深さが引数＜深さ＞に返されます。画面の深さは、モニタ上で表示されるカラーの数を表す2のべき乗の指数です。例えば、モニタが256色(2^8)に設定されている場合、画面の深さは 8になります。

4th Dimensionでは、次の表のように前もって定義された定数が用意されています：

定数	タイプ	値
Black and white	倍長整数	0
Four colors	倍長整数	2
Sixteen colors	倍長整数	4
Two fifth six colors	倍長整数	8
Thousands of colors	倍長整数	16
Millions of colors 24 bit	倍長整数	24
Millions of colors 32 bit	倍長整数	32

モニタがカラーを表示するよう設定されている場合、＜カラー＞には1が返されます。 モニタがグレイスケールを表示するよう設定されている場合、＜カラー＞には 0(ゼロ)が返されます。この値は、Macintoshプラットフォーム上で重要であることに注意してください。

4th Dimensionでは、次の表のように前もって定義された定数が用意されています：

定数	タイプ	値
Is gray scale	倍長整数	0
Is color	倍長整数	1

■ オプションの引数＜画面＞には、情報を得たいモニタを指定します。Windowsでは、通常、引数＜画面＞は使用しません。Macintoshでは、引数＜画面＞を省略すると、このコマンドは、主画面(メインスクリーン)、つまり、メニューバーが表示されている画面の深さを返します。

例：
アプリケーションが多くのカラーグラフィックスを表示するとします。その場合には、データベースのどこかに次のように記述することができます：

```
SCREEN DEPTH ($vlDepth ; $vlColor)
If ($vlDepth<8)
   ALERT("モニタのカラー表示を256色以上に設定すれば、フォームの表示がよくなります。")
End if
```

参照：Count screens、SET SCREEN DEPTH

SET SCREEN DEPTH

SET SCREEN DEPTH (深さ ; カラー {; 画面})

引数	タイプ		説明
深さ	数値	←	画面の深さ(カラーの数 = 2 ^ 深さ)
カラー	数値	←	1、カラー画面 0、グレイスケール
画面	数値	→	画面番号、または省略された場合には主画面(メインスクリーン)

このコマンドは、Windows上では何もしません。

Macintosh上では、**SET SCREEN DEPTH**コマンドはユーザが引数＜画面＞に渡した番号を持つ画面の深さとカラー／グレイスケールの設定を変更します。引数＜画面＞を省略した場合には、このコマンドは主画面(メインスクリーン)に対して適用されます。

引数＜カラー＞と＜深さ＞に渡す値についての詳細は、**SCREEN DEPTH**コマンドの説明を参照してください。

参照：SCREEN DEPTH

Menu bar screen

Menu bar screen → 数値

引数	タイプ		説明
この関数には、引数はありません。			
関数の返す値	数値	←	メニューバーが表示されている画面の番号

Menu bar screen関数は、メニューバーが表示されている画面の番号を返します。

Windowsでの注意： Windowsでは、Menu bar screen関数は常に1を返します。

参照：Count screen、Menu bar height

Menu bar height

Menu bar height → 数値

引数	タイプ		説明
この関数には、引数はありません。			
関数の返す値	数値	←	メニューバーの高さ(ピクセル単位) (メニューバーが表示されていない場合にはゼロを返します。)

Menu bar height関数は、メニューバーが表示されている画面の番号を返します。

参照：HIDE MENU BAR、Menu bar height、SHOW MENU BAR

FONT LIST

FONT LIST (フォント)

引数	タイプ		説明
フォント	配列	←	フォント名の配列

FONT LISTコマンドは、文字列またはテキスト配列の引数＜フォント＞に、システム上で使用可能なフォントの名前を格納します。

例：

フォーム上に、システム上で使用可能なフォントリストを表示するドロップダウンリストを作成したいとします。その場合、次のようなドロップダウンリストのメソッドを記述します：

```
Case of
   \ (Form event=On Load)
      ARRAY STRING(63 ; asFont ; 0)
      FONT LIST(asFont)
       ...
End case
```

参照：Font name、Font number

Font name

Font name (フォント番号) → 文字列

引数	タイプ		説明
フォント番号	数値	←	フォント名を返すフォント番号
関数の返す値	文字列	←	フォント名

Font name関数は、＜フォント番号＞に指定した番号を持つフォントの名前を返します。指定された番号を持つ使用可能なフォントがない場合には、このコマンドは空の文字列を返します。

例：

1. デフォルト(省略時)のシステムフォントを使用してフォームオブジェクトを表示したい場合には、次のように記述します：

```
FONT(myObject ; Font name(0)) `0はデフォルトのシステムフォントの番号です。
```

2. デフォルトのアプリケーションフォントを使用してフォームオブジェクトを表示したい場合には、次のように記述します：

```
FONT(myObject ; Font name(1)) `1はデフォルトのアプリケーションフォントの番号です。
```

参照：FONT LIST、Font number

Font number

Font number(フォント名) → 数値

引数	タイプ		説明
フォント名	文字列	←	フォント番号を返すフォント名
関数の返す値	数値	←	フォント番号

Font number関数は、＜フォント名＞に指定したフォント名を持つフォント番号を返します。指定された名前を持つフォントがない場合には、このコマンドは0を返します。

例：

データベースのフォームのいくつかが「特殊」という名前のフォントを使用するとします。その場合、データベースのどこかに、次のように記述することができます：

```
If (Font number("特殊")=0)
   ALERT("特殊フォントをインストールすれば、このフォームの表示がよくなります。")
End if
```

参照：FONT LIST、Font name

Current machine

Current machine → 文字列

引数	タイプ		説明
この関数には、引数はありません。			
関数の返す値	文字列	←	マシンのネットワーク名

Current machine関数は、「ネットワーク」コントロールパネルで設定されたマシンのネットワーク名を返します。

例：

クライアント／サーバ版の4D環境でアプリケーションを実行していない場合でも、アプリケーションに含まれているいくつかのネットワークサービスが、マシンが正しく構成されていることを必要とする可能性があります。アプリケーションの On Startupデータベースメソッドの中で、次のように記述します：

```
If ( (Current machine="") | (Current machine owner=""))
   `マシンのネットワークIDを設定するようユーザに要求するダイアログボックスを表示する。
End if
```

参照：Current machine owner

Current machine owner

Current machine owner→ 文字列

引数	タイプ		説明
この関数には、引数はありません。			
関数の返す値	文字列	←	マシンオーナーのネットワーク名

Current machine owner関数は、「ネットワーク」コントロールパネルで設定されたマシンのオーナー名を返します。

例：

Current machine関数の例を参照してください。

参照：Current machine

System folder

System folder→ 文字列

引数	タイプ		説明
この関数には、引数はありません。			
関数の返す値	文字列	←	稼働しているシステムディレクトリまたはフォルダへのパス名

System folder関数は、 WindowsまたはMacintoshの稼働しているシステムフォルダへのパス名を返します。

参照：Aci folder、Temporary folder

Temporary folder

Temporary folder→ 文字列

引数	タイプ		説明
この関数には、引数はありません。			
関数の返す値	文字列	←	テンポラリ（中間）フォルダへのパス名

Temporary folder関数は、システムによって設定される現在(カレント)のテンポラリ（中間）フォルダへのパス名を返します。

▼ **APPEND TO CLIPBOARD**コマンドの例を参照してください。

参照：System folder

Gestalt

Gestalt (セレクタ ; 値)→ 文字列

引数	タイプ		説明
セレクタ	文字列	→	4文字のGestaltセレクタ
値	数値	←	gestalt値
関数の返す値	数値	←	エラーコード

Gestalt関数は、ユーザが＜セレクタ＞に渡すセレクタに基づいて、システムハードウェアおよびソフトウェアの特性を示す数値を＜値＞に返します。

必要な情報が得られると、**Gestalt**関数の結果として0を、取得できなかった場合には、エラーコード -5550を返します。セレクタがわからなければ、**Gestalt**関数はエラーコード -5551を返します。

重要：GestaltマネージャはMacOSの一部です。セレクタのいくつかはWindows上でも実現されていますが、このコマンドの有効性はWindows上では減少してしまいます。

Gestalt関数に渡すことができるセレクタについての詳細は、Gestaltマネージャに関するApple社の開発者向け(Developer)ドキュメントを参照してください。

例：

Machintosh上で、MacOsのバージョン7.6を使用している場合、次のコードは、「システムバージョン0x0760を実行しています。」という警告(アラート)を表示します。

```
$vlErrCode:=Gestalt("sysv" ; $vlInfo)
If ($vlErrCode=0)
   ALERT("システムバージョン："+String($vlInfo ; "&x") + "を実行しています。")
End if
```

テーブルコマンド　54

この章では、テーブルとフォームのデフォルトを設定するコマンドについて説明します。これらのコマンドは、「ユーザ」モードで“ テーブル／フォーム選択 ”を選択した場合と同じです。

Current form table　　**DEFAULT TABLE**　　**OUTPUT FORM**
Current default table　　**INPUT FORM**

4th Dimensionのプログラミング言語の多くのコマンドは、テーブルの指定が必要です。コマンドの最初の引数としてテーブルを指定することができます。また、**DEFAULT TABLE**コマンドでデフォルトテーブルを設定することもできます。各々のプロセスは独自のデフォルトテーブルを持っています。

DEFAULT TABLE

DEFAULT TABLE (テーブル)

引数	タイプ	説明
テーブル	テーブル	デフォルトとして設定するテーブル

DEFAULT TABLEコマンドは、＜テーブル＞をカレントレコードに対するデフォルトテーブルとして設定します。

DEFAULT TABLEコマンドが実行されるまで、デフォルトテーブルは存在しません。デフォルトテーブルを設定した後で、＜テーブル＞を省略したコマンドがデフォルトテーブルに対して実行されます。例えば、次のコマンドを見てください：

INPUT FORM ([テーブル] ; "フォーム")

デフォルトテーブルで[テーブル]を設定した場合に、次のような同じコマンドの別の記述が可能です：

INPUT FORM ("フォーム")

デフォルトテーブルの設定のもう1つの目的は、テーブルに特定されないステートメントを作成することです。これによって、同じステートメントで異なるテーブルを操作することができます。

DEFAULT TABLEコマンドではフィールドを参照する場合に、テーブル名を省略することはできません。例えば、次のように記述します：

```
[私のテーブル]私のフィールド:="設定"
```

次のように記述することはできません：

```
DEFAULT TABLE ([私のテーブル])
私のフィールド:="設定"
```

これは、単にデフォルトテーブルが設定されるだけです。しかし、フォームメソッド、オブジェクトメソッドにおいてそれに属するテーブルのフィールドを参照する場合は、テーブル名を省略することができます。

4th Dimensionでは、すべてのテーブルを“開いて”、使用する準備ができています。しかし、**DEFAULT TABLE**コマンドはテーブルを“開いたり”、カレントテーブルを設定したり、入出力のためにテーブルを準備することはありません。**DEFAULT TABLE**コマンドはプログラミングの労力の節約とステートメントを読みやすくするための便宜を図るだけです。

TIP：DEFAULT TABLEコマンドの使用やテーブル名の省略により、ステートメントを読みやすくすることができるかもしれませんが、多くのプログラマーはこのコマンドが実際の価値よりは多くの問題と混乱を引き起こすものであることに気付くでしょう。

▼ 次の例は、最初に**DEFAULT TABLE**コマンドを使用しないステートメントを示しています。このステートメントは、新しいレコードをデータベースに追加するのによく使用されるループです。またこの後で、**DEFAULT TABLE**コマンドを使用した同じステートメントを示します。**INPUT FORM**コマンドと**ADD RECORD**コマンドは、1番目の引数としてテーブルを必要とします：

```
INPUT FORM ([顧客] ; "住所")
Repeat
  ADD RECORD ([顧客])
Until (OK=0)
```

デフォルトテーブルの指定により、次のメソッドが導かれます：

```
DEFAULT TABLE ([顧客])
INPUT FORM ("住所")
Repeat
  ADD RECORD
Until (OK=0)
```

Current default table

Current default table → ポインタ

Current default table関数は、**DEFAULT TABLE**コマンドで最後に指定されたテーブルのポインタを返します。

フォームを指定する

この節は、入出力のフォームを指定するコマンドについて説明します。フォームは、データ入力、印刷、データ読み込み、データ書き出し、ユーザインタフェースの作成など4th Dimension内で広範囲に使用されます。

入力フォームは、一般的にデータ入力などの1度に1つのレコードしか表示しないコマンドに付随します。出力フォームは通常、リスト形式で画面またはプリンタに複数レコードを表示するコマンドに付随します。フォームは、本質的には入力フォームでも出力フォームでもありません。フォームのいくつかは入力フォームとして、または出力フォームとして使用されます。

INPUT FORMコマンドと**OUTPUT FORM**コマンドは、テーブルに対して使用するフォームを指定します。各テーブルは、カレント入力フォームとカレントイ出力フォームを持っています。これらのコマンドは、その後のフォームを必要とするコマンドで使用されます。カレントフォームは、「エクスプローラ」の「フォーム」ページ上でカレント入力フォームを“Ｉ”およびカレント出力フォームを“Ｏ”で指定します。**INPUT FORM**コマンドまたは**OUTPUT FORM**コマンドで、入出力のフォームを指定しない場合には、すべての操作に対して、「デザイン」モードで指定されたフォームを使用します。

INPUT FORMコマンドと**OUTPUT FORM**コマンドは、単に使用するフォームを指定するだけで、フォームを表示するわけではありません。

INPUT FORM

INPUT FORM ({テーブル} ; フォーム {; *})

引数	タイプ		説明
テーブル	テーブル	→	入力フォームを指定するテーブル 省略した場合は、デフォルトテーブル
フォーム	文字列	→	フォーム名
*		→	自動ウインドウサイズ

INPUT FORMコマンドは、＜テーブル ＞のカレント入力フォームに＜フォーム＞を指定します。各テーブルは、個々の入力フォームを持っています。そのフォームは、＜テーブル＞に属するものでなければなりません。**INPUT FORM**コマンドは他のコマンドで表示したり、使用するフォームを指定するだけです。フォームを表示することはできません。(フォームの作成に関する詳細は、『4th Dimension / 4D First デザインリファレンス』を参照してください。)

デフォルト入力フォームは、「デザイン」モードで定義します。「エクスプローラ」の「フォーム」ページのフォーム名の隣にある文字 “ I ” で確認します。**INPUT FORM**コマンドで入力フォームを指定しない場合には、デフォルト入力フォームが使用されます。

入力フォームは一般に、データの入力や修正を行うコマンドにより表示されます。

次のコマンドは、すべて入力フォームを表示します：

ADD RECORD **MODIFY RECORD**
DISPLAY RECORD **QUERY BY EXAMPLE**

次のコマンドは、出力フォームを通してレコードのリストを表示します。そして、ユーザがレコードをダブルクリックすると、入力フォームを表示します：

DISPLAY SELECTION **MODIFY SELECTION**

入力フォームは、ユーザがサブフォームをダブルクリックした場合にも表示されます。この場合、「デザイン」モードでサブフォームを作成するときに、入力フォーム(フルページフォーム)を割り当てなければなりません。

次のデータ読み込みコマンドは、入力フォームを使用します：

IMPORT DIF **IMPORT SYLK** **IMPORT TEXT**

オプション引数＜*＞は、「デザイン」モードの「フォームプロパティ」ウインドウおよび**Open window**関数で設定するフォームプロパティと一緒に使用されます。＜*＞を指定することにより、次回、(ダイアログボックスや入力フォームとして) フォームを使用する際に自動的にウインドウサイズを変更をフォームプロパティで行います。

注：オプション引数<*>を使用するしないに関係なく、INPUT FORMコマンドはテーブルの入力フォームを変更します。

▼ 次の例は、**INPUT FORM**コマンドの一般的な使用方法です。この例では、**INPUT FORM**コマンドを入力フォームが使用される直前に記述していますが、実際には、これは必要はありません。他のメソッドで**INPUT FORM**コマンドを実行しても構いません：

```
INPUT FORM ([会社] ; "会社追加")    ` 新しい会社を追加するためのフォームを指定する
ADD RECORD ([会社])                 ` 新しい会社を追加する
```

参照：ADD RECORD、DISPLAY RECORD、DISPLAY SELECTION、IMPORT DIF、IMPORT SYLK、IMPORT TEXT、MODIFY RECORD、MODIFY SELECTION、Open window、OUTPUT FORM、QUERY BY EXAMPLE

OUTPUT FORM

OUTPUT FORM ({テーブル} ; フォーム)

引数	タイプ	説明
テーブル	テーブル	出力フォームを指定するテーブル
フォーム	文字列	フォーム名

OUTPUT FORMコマンドは、<テーブル> のカレント出力フォームに<フォーム>を指定します。各テーブルは、個々の出力フォームを持っています。フォームは、<テーブル>に属するものでなければなりません。**OUTPUT FORM**コマンドは、他のコマンドで使用するフォームを指定するだけです。フォームを表示することはできません。(フォームの作成に関する詳細は、『4th Dimension / 4D First デザインリファレンス』を参照してください。)

デフォルト出力フォームは、「デザイン」モードで定義します。「エクスプローラ」の「フォーム」ページ内のフォーム名の隣にある文字 " O " で確認します。**OUTPUT FORM**コマンドで出力フォームを指定しない場合には、デフォルト出力フォームが使用されます。

出力フォームは 3つのコマンドグループ(画面上にレコードをリスト表示するグループ、レポートを作成するグループ、データを書き出すグループ)で使用します。

下記のコマンドは、出力フォームを介して、レコードを表示します：

DISPLAY SELECTION　　MODIFY SELECTION

出力フォームは、サブフォーム内の表示に使用することもできます。この場合、「デザイン」モードでサブフォームを作成する時に、出力フォーム(マルチラインフォーム)を割り当てなければなりません。

次のコマンドは、レポートを印刷する場合の出力フォームを使用します：

PRINT LABEL **PRINT SELECTION**

次のデータ書き出しコマンドは、出力フォームを使用します：

EXPORT DIF **EXPORT SYLK** **EXPORT TEXT**

▼ 次の例は、**OUTPUT FORM**コマンドの一般的な使用方法です。この例では、**OUTPUT FORM**コマンドを出力フォームが使用される直前に記述していますが、実際にはこれは必要はありません。他のメソッドで**OUTPUT FORM**コマンドを実行しても構いません：

```
INPUT FORM ([部品] ; "部品入力")          ` 入力フォームの選択
OUTPUT FORM ([部品 ; "部品リスト")        ` 出力フォームの選択
MODIFY SELECTION ([部品])          ` このコマンドは両方のフォームを使用する
```

Current form table

Current form table → ポインタ

引数	タイプ		説明
この関数には、引数はありません。			
関数の返す値	ポインタ	←	現在表示されているフォームが属すテーブルポインタ

Current form table関数は、カレントプロセスでフォームが表示または印刷されているテーブルのポインタを返します。

カレントプロセスに表示または印刷されているフォームがない場合には、このコマンドはNilを返します。

カレントプロセスで複数のウィンドウが開いている（最後に開かれたウィンドウがカレントアクティブウィンドウになる）場合には、このコマンドは、アクティブウィンドウにフォームが表示されているテーブルへのポインタを返します。

現在表示されているフォームがサブフォームエリア用の「詳細」フォームである場合には、ユーザがデータ入力中であり、ダブルクリック可能なサブフォームエリアのレコードまたはサブレコードをダブルクリックしたことを意味します。この場合には、コマンドは以下を返します：

■ サブフォームがテーブルを表示している場合には、サブフォームエリアに表示されたそのテーブルへのポインタ

■ サブフォームエリアがサブテーブルを表示している場合には、重要な意味を持たないポインタ

例：

アプリケーション全体を通じて、レコードを表示する際には、次の表示方法に従います。フォーム内に変数「vsCurrentRecord」がある場合、ユーザが新しいレコードを処理していれば、「新規レコード」と表示します。ユーザが5200レコードから成るセレクションの56番目のレコードを処理していれば、「56 / 5200」と表示します。

そのためには、オブジェクトメソッドを使用して変数「vsCurrentRecord」を作成し、その後、それをコピーして、すべてのフォームに貼り付けます：

```
`「vsCurrentRecord」入力不可変数のオブジェクトメソッド
Case of
   \ (Form event =On Load)
      C_STRING (31 ; vsCurrentRecord)
      C_POINTER ($vpParentTable)
      C_LONGINT ($vlRecordNum)
      $vpParentTable:=Current form table
      $vlRecordNum:=Record number ($vpParentTable->)
   Case of
      \ ($vlRecordNum=-3)
         vsCurrentRecord:="新規レコード"
      \ ($vlRecordNum=-1)
         vsCurrentRecord:="レコードなし"
      \ ($vlRecordNum>=0)
         vsCurrentRecord:=String (Selected record number ($vpParentTable->))+" of
                  "+String (Records in selection ($vpParentTable->))
   End case
End case
```

参照：DIALOG、INPUT FORM、OUTPUT FORM、PRINT SELECTION

この章では、「ルーチン」エディタの「Tool Bar」テーマ内にあるツールバーコマンドについて説明します。

HIDE TOOL BAR **SHOW TOOL BAR**

HIDE TOOL BAR

HIDE TOOL BAR

引数 **タイプ** **説明**

このコマンドには、引数はありません。

HIDE TOOL BARコマンドは、ツールバーを隠します。

ツールバーがすでに隠れている場合は、**HIDE TOOL BAR**コマンドは何も行いません。

参照：HIDE MENU BAR、SHOW MENU BAR、SHOW TOOL BAR

SHOW TOOL BAR

SHOW TOOL BAR

引数 **タイプ** **説明**

このコマンドには、引数はありません。

SHOW TOOL BARコマンドは、ツールバーを表示します。

ツールバーがすでに表示さている場合は、**SHOW TOOL BAR**コマンドは何も行いません。

参照：HIDE MENU BAR、**HIDE TOOL BAR**、SHOW TOOL BAR

トランザクションコマンド 56

この章では、「ルーチン」エディタの「Transactions」テーマ内にあるトランザクションコマンドについて説明します。

START TRANSACTION
VALIDATE TRANSACTION
CANCEL TRANSACTION
In Transaction

トランザクションを使用する

トランザクションは、データベース上の一連のリレートされたデータに対する更新です。トランザクションは、そのトランザクションの正当性が確認されるまで、ディスクに保存されません。キャンセルされたり、他の外部的な原因でトランザクションが完了できなかった場合には、更新処理の結果は保存されません。

トランザクション処理で更新されたデータベースのデータは、一時的なバッファにローカルに保存されます。トランザクションが**VALIDATE TRANSACTION**コマンドで受け入れられた時点で、更新されたデータがデータベース上に保存されます。トランザクションが**CANCEL TRANSACTION**コマンドでキャンセルされた場合には、更新されたデータは保存されません。

トランザクションが受け入れられたりキャンセルされると、カレントプロセスのテーブルのセレクションは、トランザクションが一時的なレコードアドレスで処理されますので、空になります。これと同じ理由から、トランザクション内での命名セレクションの使用には十分な注意が必要です。トランザクションが受け入れられたりキャンセルされると、命名セレクションは、正しくないレコードアドレスを含んでいるかもしれないトランザクションの前または途中で作成されてしまいます。そのため、例えば、命名セレクションが削除されたレコードのアドレスを含んでいたり、トランザクション中の追加レコードの一時的アドレスを含んでいる可能性があります。

また、それらがレコードアドレスを使ったビットテーブルを基としてるのでセットにも適用します。

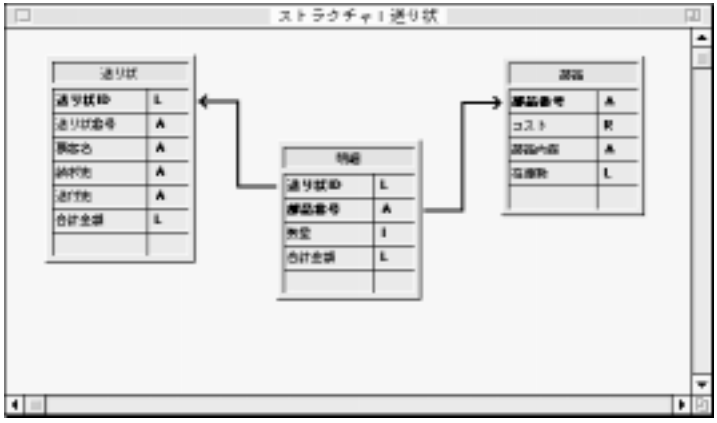
下記のコマンドは、トランザクションの中で使用することはできません：

GOTO RECORD
RELATE ONE SELECTION
RELATE MANY SELECTION

トランザクションの例

この節の例は、次のデータベースストラクチャを基にしています。データベースは簡単な送り状システムです。送り状の明細は、[明細]テーブルに格納されています。このテーブルは、[明細]テーブルの“送り状ID”フィールドと[送り状]テーブルの“送り状ID”フィールドで[送り状]テーブルにリレートしています。送り状が追加されると、重複不可のIDは**Sequence number**関数を使用して計算されます。[送り状]テーブルと[明細]テーブルのリレートは自動リレートになっており、「リレート値自動代入」チェックボックスがチェックされています。

[明細]テーブルと[部品]テーブルのリレートは、マニュアルリレートです。



ユーザは送り状を入力する場合に、次のような動作を実行しなければなりません。

■ [送り状]テーブルにレコードを追加する

■ [明細]テーブルにレコードをいくつか追加する

■ 送り状にリストされた各部品の[部品]在庫数フィールドを更新する

つまり、リレートされたデータを保存する必要があります。これは、トランザクションの使用が必要になる典型的な場面の1つです。

レコードが追加されない場合やレコードが更新されない場合は、処理中にこれらのレコードをすべて保存するか、またはトランザクションでキャンセルするかを確認する必要があります。

トランザクションを使用しない場合には、データベースのデータを保証することはできません。例えば、[部品]テーブルのレコードがロックされていると、[部品]在庫数フィー

ルドに格納されている数量を更新することはできません。従って、このフィールドは論理上、正しいものではなくなります。

つまり、販売した部品の合計と倉庫内に残っている在庫数が、レコードに入力したオリジナルの数量と等しくならなくなります。こういう場面を防ぐために、トランザクションを使用します。

トランザクションを使って、データ入力を実行する方法にはいくつかあります：

1. 「デザイン」モードの「データベースプロパティ」ダイアログボックス内にある「データ入力時に自動トランザクションを使用する」チェックボックスを選択して4th Dimensionにハンドルする。この場合、4th Dimensionはトランザクションを開始して、データ入力を受け付けるかどうかによってそのトランザクションを有効にしたり取り消したりします。サブフォームにリレートしたテーブルを持つフォームでデータ入力作業を行う場合は、トランザクションは必要です。このオプションはデータベース全体に適用されます。

トランザクション自身をハンドルしたい場合は、**START TRANSACTION**、**VALIDATE TRANSACTION**、**CANCEL TRANSACTION**のトランザクションコマンドを使用する必要があります。

2. 次のように記述することができます：

```
READ WRITE ([明細])          ` [明細]テーブルをリードライト状態にセットする
READ WRITE ([部品])          ` [部品]テーブルをリードライト状態にセットする
INPUT FORM ([送り状] ; "入力")     ` 入力レイアウトを選択する
Repeat
   START TRANSACTION
   ADD RECORD ([送り状]) ` 必要なだけレコードを追加する
   If (OK=1)
      VALIDATE TRANSACTION
   Else
      CANCEL TRANSACTION
   End if
Until (OK=0)
READ ONLY (*)                ` [明細]、{部品]テーブルをリードオンリー状態にセットする
```

3. データ入力実行中のレコードロックを減らすには、フォームメソッド内からトランザクションを管理したり、それが必要になった時にだけリードライト状態のテーブルにアクセスすることができます。

サブフォームにリレートテーブルの[明細]を持つ[送り状]テーブルの入力フォームを使ってデータ入力を行います。このフォームには、「動作なし」ボタン属性の「bキャンセル」と「b登録」の2つのボタンがあります。

メソッドは次のようになります：

```
READ WRITE ([明細])          ` [明細]テーブルをリードライト状態にセットする
READ ONLY ([部品])           ` [部品]テーブルをリードライト状態にセットする
INPUT FORM ([送り状] ; "入力")     ` 入力レイアウトを選択する
Repeat
   ADD RECORD ([送り状]) ` 必要なだけレコードを追加する
Until (b登録=0)
READ ONLY ([明細])           ` [明細]テーブルをリードオンリー状態にセットする
```

[部品]テーブルがデータ入力中はリードオンリー状態になっていることに注目してください。リードライト状態は、データ入力が有効な場合にのみ利用できます。

[送り状]入力フォームから開始されるトランザクションを下記に示します：

```
Case of
   \ (Form event=On Load)
         START TRANSACTION
         [送り状]送り状ID := Sequence number ([送り状]送り状ID)
   Else
         [送り状]合計金額 := Sum([明細]合計金額)
End case
```

「bキャンセル」ボタンをクリックすると、トランザクションはもちろんのことデータ入力も取り消す必要があります。下記に「bキャンセル」ボタンのオブジェクトメソッドを示します：

```
Case of
   \ (Form event=On Clicked)
         CANCEL TRANSACTION
         CANCEL
End case
```

「b登録」ボタンをクリックすると、トランザクションはもちろんのことデータ入力も受け付ける必要があります。下記に「b登録」ボタンのオブジェクトメソッドを示します：

```
Case of
  \ (Form event=On Clicked)          ` 登録ボタンをクリックした場合
    $明細数:=Records in selection ([明細])
    READ WRITE ([部品])              ` 部品を更新する
    FIRST RECORD ([明細])            ` 最初の明細で開始する
    $受け入れ:=True                  ` すべてにOKであることように仮定する
    For ($Line ; 1 ; $明細数)        ` 全明細をループする
      RELATE ONE ([明細]部品番号)
      OK:=1                          ` 続行したいと仮定する
      While (Locked ([部品]) & (OK=1))
        CONFIRM ("部品番号："+[明細]部品番号+" が使用されています。待ちますか？")
        If (OK=1)
          DELAY PROCESS (Current process ; 60)
          LOAD RECORD ([部品])       ` レコードをロードしてくる
        End if
      End while
      If (OK=1)
        [部品]在庫数:=[部品]在庫数 - [明細]数量
          ` 倉庫の数量を更新する
        SAVE RECORD ([部品])         ` レコードを保存する
      Else
        $Line:=$明細数 + 1           ` ループを抜ける
        $受け入れ:=False
      End if
      NEXT RECORD ([明細])           ` 次の明細に移動する
    End for
    READ ONLY ([部品])               ` 部品テーブルをリードオンリー状態にする
    I f ($受け入れ)
      SAVE RECORD ([送り状])         ` 送り状レコードを保存する
      VALIDATE TRANSACTION           ` すべてのデータベースの修正を受け入れる
    Else
      CANCEL TRANSACTION             ` すべてをキャンセルする
    End if
    CANCEL                           ` レイアウトを抜ける
End case
```

このコードでは、任意ボタンのクリックに関係なく、**CANCEL**コマンドを実行します。新しいレコードは**ACCEPT**コマンドを呼び出しても受け入れられませんが、**SAVE RECORD**コマンドならば受け入れられます。**SAVE RECORD**コマンドが**VALIDATE TRANSACTION**コマンドの前で呼び出されているからです。

つまり、[送り状]テーブルのレコードを保存するということは、トランザクションの一部であるということです。**ACCEPT**コマンドを呼び出してレコードを受け入れることもできますが、それでは送り状が保存される前にトランザクションが受け入れられてしまいます。つまり、レコードはトランザクションの外で保存されるということです。

参照：CANCEL TRANSACTION、In transaction、START TRANSACTION、VALIDATE TRANSACTION

START TRANSACTION

START TRANSACTION

START TRANSACTIONコマンドは、トランザクションを開始します。データベースに対するすべての更新は、そのトランザクションを受け入れる(**VALIDATE TRANSACTION**コマンドを実行)か、キャンセルされる(**CANCEL TRANSACTION**コマンドを実行)までは、一時的なバッファにローカルに保存されます。

この節の前述の例を参照してください。

CANCEL TRANSACTION

CANCEL TRANSACTION

CANCEL TRANSACTIONコマンドは、**START TRANSACTION**コマンドで開始したトランザクションをキャンセルします。**CANCEL TRANSACTION**コマンドは、データベースのデータをトランザクション開始前の状態に戻します。

この節の前述の例を参照してください。

VALIDATE TRANSACTION

VALIDATE TRANSACTION

VALIDATE TRANSACTIONコマンドは、**START TRANSACTION**コマンドで開始したトランザクションを受け入れます。**VALIDATE TRANSACTION**コマンドは、トランザクションによる更新結果をデータベースに保存します。

この節の前述の例を参照してください。

In transaction

In transaction → ブール

引数	タイプ		説明
この関数には、引数はありません。			
関数の返す値	ブール	←	現在のプロセスがトランザクション内の場合、Trueを返す

In transaction関数は、現在のプロセスがトランザクション内にあればTRUE(真)、なければFALSE(偽) を返します。

例：

複数のレコードに対する操作(レコードの追加、修正、または削除)を実行すると、ロックされたレコードに出くわす可能性があります。このような場合、データの整合性を維持するためには、失敗したときに操作全体をロールバックして、データベースを元の状態に戻せるように、トランザクションの中で操作を行わなければなりません。

トリガ内から、あるいはサブルーチン(トランザクションの中でも、外でも呼び出せる)から操作を実行する場合には、**In transaction**関数を使用して、現在のプロセスのメソッドまたは呼び出し側のメソッドがトランザクションを開始したかどうかをチェックすることができます。トランザクションが開始されていなければ、操作を開始してはいけません。失敗した場合に、操作をロールバックすることができなくなってしまいます。

参照：CANCEL TRANSACTION、START TRANSACTION、VALIDATE TRANSACTION

この章では、「ルーチン」エディタの「Triggers」テーマ内にあるトリガコマンドについて説明します。

Database event　　**Trigger level**　　**TRIGGER PROPERTIES**

トリガについて

トリガは、テーブルに付属するメソッドであり、テーブルのプロパティです。トリガを呼び出す必要はありません。ユーザがテーブルのレコードを操作する（追加、削除、修正、ロードする）たびに、4Dデータベースエンジンによって、自動的に起動されます。まず、簡単なトリガを記述し、その後でより洗練されたものにすることができます。

トリガを使用すれば、データベースのレコードに対して「不正な」操作が行われるのを防ぐことができます。トリガは、偶発的にデータが失われたり、変更されたりするのを防ぐだけでなく、テーブルに対する操作を制限するための非常に強力なツールです。例えば、送り状のシステムにおいて、誰かが、送り状の送付先である顧客を指定せずに、送り状を追加するのを防止することができます。

4Dの旧バージョンとの互換性

トリガは、バージョン6で導入された新しいタイプのメソッドです。4th Dimensionの旧バージョンでは、テーブルメソッド（ファイルプロシージャと呼ばれていました）は、データの入力、表示、あるいは印刷のために、テーブル用のフォームが使用される場合にのみ4Dによって実行されていましたが、あまり使われていませんでした。トリガは旧バージョンのファイルプロシージャよりもずっと低いレベルで実行されることに注意していください。ユーザアクションによって（データ入力など）、あるいはプログラムによって（**SAVE RECORD**コマンドへの呼び出しなど）、レコードに対してどのような操作を行った場合でも、4Dによってテーブルのトリガが起動されます。バージョン3のデータベースからバージョン6に変換して、この新しいトリガの機能を利用したい場合には、以下に示した「データベースプロパティ」ダイアログボックスにおいて、「旧バージョンのファイルプロシージャ方式を使用する」プロパティの選択を解除する必要があります。

トリガのアクティブ化と作成

デフォルトでは、「デザイン」モードでテーブルを作成したときには、テーブルにはトリガはありません。

テーブルのトリガを使用するには、以下を行う必要があります：

■ トリガをアクティブにし、4Dに対してトリガをいつ起動すべきかを知らせる。

■ トリガ用のコードを記述する。

まだ記述されていないトリガをアクティブにしたり、トリガをアクティブにしないで記述しても、テーブルに対して実行される操作に影響を与えることはありません。

1. トリガをアクティブにする

テーブルのトリガをアクティブにするには、「テーブルプロパティ」ウィンドウでテーブルの「トリガ」オプション（データベースイベント）のいずれかを選択しなければなりません：

新規レコード保存時：

このオプションを選択すると、レコードがテーブルに追加されるたびに、トリガが起動されます。

次のような場合に、トリガが起動されます：

■ （「ユーザ」モードまたは**ADD RECORD**コマンドを使って)。データ入力でレコードを追加する

■ **CREATE RECORD**コマンドと**SAVE RECORD**コマンドでレコードを作成し、保存する。トリガは、レコードの作成時ではなく、**SAVE RECORD**コマンドを呼び出した時にも起動されることに注意してください。

■ (「ユーザ」モードまたは「データ読み込み...」コマンドを使って)レコードをインポートする（読み込む)。

■ (**ARRAY TO SELECTION**コマンド、**SAVE RELATED ONE**コマンドを使って)新規レコードを作成または保存する。

■ **CREATE RECORD**コマンドと**SAVE RECORD**コマンドを呼び出すプラグインを使用する。

既存レコード保存時：

このオプションを選択すると、テーブルのレコードが修正されるたびに、トリガが起動されます。

次のような場合に、トリガが起動されます：

■ (「ユーザ」モードまたは**MODIFY RECORD**コマンドを使って)データ入力でレコードを修正する。

■ **SAVE RECORD**コマンドですでに存在しているレコードを保存する。

■ (**ARRAY TO SELECTION**コマンド、**APPLY TO SELECTION**コマンド、**MODIFY SELECTION**コマンドを使って)既存レコードを保存する。

■ **SAVE RECORD**コマンドを呼び出すプラグインを使用する。

レコード削除時：

このオプションを選択すると、テーブルのレコードが削除されるたびに、トリガが起動されます。

次のような場合に、トリガが起動されます：

■ (「ユーザ」モードまたは**DELETE RECORD**コマンド、あるいは**DELETE SELECTION**コマンドを使って)レコードを削除する。

■ リレートの削除制御オプションによって、リレートレコードの削除を引き起こす何らかの操作を行う。

■ **DELETE RECORD**コマンドを呼び出すプラグインを使用する。

レコード読込時：

このオプションを選択すると、テーブルのレコードがロードされるたびに、トリガが起動されます。これは、カレントレコードをデータファイルからロードするあらゆる状況を含みます。このオプションは、前述の3つのオプションほど使用しません。

注：このオプションは、新規レコードの作成をカバーしません。既存レコードのロードに対してのみ、適用されます。

重要：複数のレコードに影響する操作を実行したり、コマンドを呼び出すと、トリガは各レコードに対して一度ずつ呼び出されます。例えば、100個のレコードから構成されるカレントセレクションを持ったテーブルに対してAPPLY TO SELECTIONコマンドを呼び出すと、トリガは100回起動されます。

2. トリガを作成する

テーブルのトリガを作成するには、「エクスプローラ」ウィンドウを使用するか、Alt (Windowsの場合)またはOption (Macintosh)キーを押して、「ストラクチャ」ウィンドウのテーブルタイトルをダブルクリックしてください。詳細については、「4th Dimension / 4D First デザインリファレンス」を参照してください。

データベースイベント

トリガは、前述の4つのデータベースイベントのいずれかに対して起動することができます。トリガ内で、**Database event**関数を呼び出すことによって、どのイベントが発生しているかを検出します。この関数は、データベースイベントを示す数値を返します。

一般には、**Database event**関数によって返される結果に関して、**Case of**文を用いて、トリガを記述します：

```
` 任意テーブルのトリガ
C_LONGINT($0)
$0:=0 `データベースリクエストが許可されると仮定する
Case of
   \ (Database event=Save New Record Event)
     `新規に作成されたレコードの保存のために適切な動作(アクション)を実行する
   \ (Database eventt=Save Existing Record Event)
     `既存のレコードの保存のために適切な動作を実行する
   \ (Database event=Delete Record Event)
     `レコードの削除のために適切な動作を実行する
   \ (Database event=Load Record Event)
     `レコードのメモリへのロードのために適切な動作を実行する
End case
```

トリガと関数

トリガには、2つの目的があります：

■ レコードが保存、削除される前に、あるいはロードされた直後に、レコードに対して動作(アクション)を実行する。

■ データベース操作を許可または拒絶する。

1. 動作を実行する

[文書] テーブルにレコードが保存（追加または修正）されるたびに、作成時を示すタイムスタンプと最新の修正時を示す「タイムスタンプ」メソッドでレコードを「マーク」したいとします。この場合、次のようなトリガを記述できます：

```
` [文書]テーブルのトリガ
Case of
   \ (Database event=Save New Record Event)
      [文書]作成日:= Time stamp
      [文書]修正日:= Time stamp
   \ (Database event=Save Existing Record Event)
      [文書]修正日:= Time stamp
End case
```

注：この例で使用されている「Time stamp」ユーザ定義関数は、固定日付が任意に選択された時点からの経過秒数を返す小さなプロジェクトメソッドです。

いったんこのトリガを記述し、アクティブにすると、ユーザがどのような方法で[文書]テーブルにレコードを追加または修正しても(データ入力、データ読込み、プロジェクトメソッド、4Dプラグイン)、レコードが最終的にディスクに書き込まれる前に、トリガによって、[文書]作成日と[文書]修正日のフィールドに自動的に日付が割り当てられます。

注：この例の詳細については、GET DOCUMENT PROPERTIESコマンドの例を参照してください。

2. データベース操作を許可または拒絶する

データベース操作を許可または拒絶するには、トリガは、$0関数結果にトリガエラーコードを返さなければなりません。

例：

[従業員]テーブルの場合を取り上げてみましょう。データ入力に際して、[従業員]保険証番号に関する規則を強制します。「確認」ボタンをクリックする際に、ボタンのオブジェクトメソッドを使用して、そのフィールドをチェックします：

```
` 「b確認」ボタンのオブジェクトメソッド
If (Good SS number ([従業員]保険証番号))
   ACCEPT
Else
   BEEP
   ALERT ("正しい保険証番号を入力してください。")
End if
```

フィールド値が正当であれば、データ入力を受け入れます。フィールド値が正当でなければ、警告（アラート）を表示して、データ入力の状態になります。

またプログラムによって[従業員]レコードを作成した場合、次のコードはプログラムとしては正当ですが、前述のオブジェクトメソッドで示された規則に違反します：

```
` 任意プロジェクトメソッドの抜粋
` ...
CREATE RECORD ([従業員])
[従業員]名前 :="DOE"
SAVE RECORD ([従業員]) `←DB規則に違反しているので。保険証番号は保存されない
```

[従業員]テーブルのトリガを使用すれば、データベースのあらゆるレベルで[従業員]保険証番号の規則を強制できます。トリガは次のようになります。

```
`[従業員]テーブルのトリガ
$0:=0
$dbEvent:=Get database event
Case of
   \ (($dbEvent=Save New Record Event) | ($dbEvent=Save Existing Record Event))
   If (Not(Good SS number ([従業員]保険証番号)))
      $0:=-15050
   Else
   ` ...
   End if
End case
```

いったんトリガを記述し、アクティブにすると、「**SAVE RECORD ([従業員])**」はデータベースエンジンエラーコード-15050が生じ、レコードは保存されません。

同様に、4Dプラグインが不正な保険証番号を持つ[従業員]レコードを保存しようとしても、トリガが同様のエラーを生じ、レコードは保存されません。

トリガを使用すれば、意図的であれ、偶発的であれ、誰も(ユーザ、データベース設計者、プラグイン、4Dサーバを使用する4Dオープンクライアント)保険証番号の規則に違反できないことが保証されます。

テーブルのトリガを使用しない場合でも、レコードを保存または削除しようとした際に、データベースエンジンエラーが生じる場合があることに注意してください。たとえば、一意なインデックス(索引)フィールドに重複する値を持つレコードを保存しようとすると、エラー－9998が返されます。

従って、エラーを返すトリガは、アプリケーションに新しいデータベースエンジンエラーを追加することになります：

■ 4Dは“ 通常 ”のエラーを管理する：一意な(重複不可の)インデックス、リレーショナル(関係)データ制御など。
■ トリガを使用して、アプリケーションに特有のカスタムエラーを管理する。

重要：エラーコード値は、任意のものを返すことができます。ただし、4Dデータベースエンジンによってすでに予約されているエラーコードは使用しないでください。－32000から－15000までの範囲のエラーコードを使用することを強く推奨します。－15000より大きなエラーコードは、データベースエンジン用に予約されています。

プロセスレベルでは、データベースエンジンエラーを処理するのと同様に、トリガエラーを処理します：

■ 4Dに標準のエラーダイアログボックスを表示させ、その後メソッドが停止されるようにすることができる。
■ **ON ERR CALL**コマンドを使用してインストールされるエラー処理メソッドを使用して、適切な方法でエラーから回復することができる。

注：データ入力中に、レコードの正当性確認または削除を行おうとして、トリガエラーが返されると、エラーは一意なインデックスエラーと同様に処理されます。エラーダイアログが表示され、データ入力の状態になります。(「カスタム」モードではなく)「ユーザ」モードでしかデータベースを使用しない場合でも、トリガの使用には利点があります。

トリガがエラーを返さない($0:=0)場合でも、これは、必ずしもデータベース操作が成功することを意味するわけではありません。一意なインデックス違反が生じることもあります。また、操作がレコードの更新である場合には、レコードがロックされており、I/Oエラーが生じたりする可能性もあります。そうしたチェックはトリガの実行後に行われます。しかし、実行プロセスのより高度なレベルでは、データベースエンジンによって返されるエラーもトリガによって返されるエラーも同じです。

トリガと4Dアーキテクチャ

トリガはデータベースエンジンレベルで実行されます。次の図にその様子をまとめています：

トリガは、データベースエンジンが実際に配置されたマシン上で実行されます。これは、シングルユーザ版の4Dでは明白です。4D Serverでは、トリガはクライアントマシンではなく、サーバマシン上の動作プロセス内で実行されます。

トリガが起動される場合、トリガはデータベース操作を実行しようとするプロセスのコンテキスト内で実行されます。トリガの実行を引き起こすこのプロセスは、起動プロセスと呼ばれます。

トリガは特に、起動プロセスのカレントセレクション、カレントレコード、テーブルの読み／書き(read/write)状態、およびレコードロック操作を用いて動作します。

警告：トリガは、そのトリガが属しているテーブルのカレントレコードを変更することはできませんし、またしてはいけません。トリガ内で、複数のフィールドに関して一意値をチェックする必要がある場合には、SET QUERY DESTINATIONコマンドを使用してください。このコマンドを使用すれば、テーブルのカレントセレクションやカレントレコードを変更することなく、テーブルを照会することができます。

4D環境のほかのデータベースオブジェクトや言語オブジェクトの使用には注意してください。これは、トリガが起動プロセスのマシンとは別のマシン上で実行される可能性があるためです。4D Serverは、これに当てはまります。

■ **インタープロセス変数：**トリガは、トリガが実行されるマシンのインタープロセス変数にアクセスします。4D Serverでは、トリガが起動プロセスのマシンとは別のマシンにアクセスする可能性があります。

■ **プロセス変数：**独立したプロセス変数テーブルがすべてのトリガによって共有されます。トリガは、起動プロセスのプロセス変数にはアクセスしません。

■ **ローカル変数：**トリガ内でローカル変数を使用できます。その有効範囲は、トリガの実行中です。ローカル変数は、トリガの実行のたびに作成され、削除されます。

■ **セマフォ：**トリガは、(トリガが実行されるマシン上の)ローカルセマフォだけでなく、グローバルセマフォもテストしたり、設定したりできます。ただし、トリガは即座に実行されなければならないため、トリガ内からセマフォをテストしたり、設定する場合には、十分な注意が必要です。

■ **セットと命名セレクション：**トリガ内からセットや命名セレクションを使用する場合、トリガが実行されるマシン上で作業することになります。

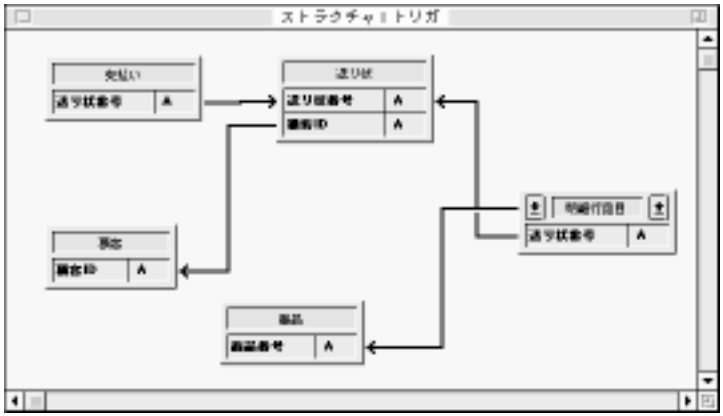
■ **ユーザインタフェース：**トリガ内でユーザインタフェース要素を使用しないでください(警告(アラート)、メッセージ、ダイアログボックスを使用しない)。従って、トリガのトレースはデバッガウィンドウに限定する必要があります。クライアント／サーバでは、トリガは4D Server上で実行されることを覚えておいてください。サーバマシン上で警告(アラート)メッセージを表示しても、クライアント上のユーザの助けにはなりません。起動プロセスにユーザインタフェースの処理を行わせるようにしてください。

トリガとトランザクション

トランザクションは、起動プロセスレベルで処理されなければなりません。トリガレベルでトランザクションを管理してはいけません。1つのトリガの実行中に、複数のレコードを追加、修正、あるいは削除しなければならない場合(下記の例を参照)には、最初にトリガ内から**In transaction**コマンドを使用して、起動プロセスが現在トランザクション内にあるかをテストしなければなりません。そうでない場合には、トリガがロックされたレコードに出くわす可能性があります。従って、起動プロセスがトランザクション内になければ、レコードに対する操作を開始しないでください。起動プロセスに、そのプロセスが実行しようとしているデータベース操作はトランザクション内で実行されなければならないことを知らせるために、$0にエラーを返すだけにしてください。そうしないと、ロックされたレコードに出くわした場合、起動プロセスにはトリガの動作(アクション)をロールバックする方法がなくなります。

波及的（カスケード）トリガ

次の例のようなストラクチャがあるとします：



注：上記のテーブルは、簡略化されています。実際には、テーブルには、ここに示したよりも多くのフィールドがあります。

データベースが、ある送り状の削除を“ 許可 ”するとしましょう。そのような操作がトリガレベルでどのように処理されるかを検討してみます（プロセスレベルで削除を実行することも可能です）。

データの関係上の整合性を維持するために、送り状を削除する場合、[送り状]のトリガで、次の動作が実行される必要があります：

■ [顧客]レコードにおいて、送り状の金額分だけ、総売上げフィールドの額を減らす。

■ その送り状に関連したすべての[明細行品目]レコードを削除する。

■ これはまた、[明細行品目]トリガが、削除された明細品目に関連した[商品]レコードの売上げ数量フィールドの数量を減らすことを意味する。

■ 削除された送り状に関連したすべての[支払い]レコードを削除する。

第1に、[送り状]のトリガは上記の動作を、起動プロセスがトランザクション内にある場合にだけ実行しなければなりません。そうすれば、ロックされたレコードに出くわした場合にロールバックを行うことができます。

第2に、[明細行品目]のトリガは[送り状]のトリガから波及しています。明細品目の削除は、[送り状]のトリガ内から**DELETE SELECTION**コマンドを呼び出した結果であるため、[明細行品目]のトリガは[送り状]のトリガの実行の「範囲内で」実行されます。

この例のすべてのテーブルでトリガがアクティブになっていると考えてください。このとき、トリガは次のように波及します：

■ **起動プロセスが送り状を削除するため、[送り状]のトリガが起動される。**

■ **[送り状]のトリガが総売上げフィールドを更新するため、[顧客]のトリガが起動される。**

■ **[送り状]のトリガが明細品目を削除するため（繰り返し）、[明細行品目]のトリガが起動される。**

■ **[明細行品目]のトリガが売上げ数量フィールドを更新するため、[商品]のトリガが起動される。**

■ **[送り状]のトリガが支払いを削除するため（繰り返し）、[支払い]のトリガが起動される。**

この波及関係において、[送り状]のトリガはレベル1で、[顧客]、[明細行品目]、および[支払い]のトリガはレベル2で、そして[商品]のトリガはレベル3で実行されていると言えます。

トリガ内から、**Trigger level**関数を使用して、トリガが実行されるレベルを検出することができます。さらに、**TRIGGER PROPERTIES**コマンドを使用して、他のレベルについての情報を得ることができます。

例えば、プロセスレベルで[商品]レコードが削除されている場合、[商品]のトリガはレベル3ではなく、レベル1で実行されます。

Trigger level関数と**TRIGGER PROPERTIES**コマンドを使用すれば、動作(アクション)の原因を検出できます。前述の例では、送り状はプロセスレベルで削除されます。仮に、プロセスレベルで[顧客]レコードを削除すると、その結果、[顧客]のトリガはその顧客に関連したすべての送り状を削除しようとします。これにより、前述の例と同じように、[送り状]のトリガが起動されることになりますが、起動される理由は異なります。[送り状]のトリガ内から、そのトリガがレベル1で実行されたか、レベル2で実行されたかを、検出することができます。トリガがレベル2で実行された場合には、次に、それが[顧客]レコードが削除されたためであるかどうかをチェックできます。そうであれば、総売上げフィールドの更新にわずらわされる必要はありません。

トリガ内での通し(シーケンス)番号の使用

「On saving new record」データベースイベントを処理する際に、テーブルのレコードに対して一意なID番号を維持するために、**Sequence number**関数を呼び出すことができます。

例：

```
`[送り状]テーブルのトリガ
Case of
   \ (Database event=On Saving new record)
   ` ...
   [送り状]送り状番号:=Sequence number ([送り状])
   ` ...
End case
```

ただし、トランザクション内でこれを行う場合には、誤って通し番号を増やすのを避けるために、フィールドに関するテストで、この呼び出しを囲まなければなりません。

典型的な例としては、データ入力トランザクション中にサブフォームにレコードを入れる場合が挙げられます。

例：

```
` [明細行品目]テーブルのトリガ
Case of
   : (Database event=On Saving new record)
      ` ...
      If ([明細行品目]商品番号=0)
         [明細行品目]商品番号:=Sequence number ([送り状])
      End if
      ` ...
End case
```

参照：Database event、Record numbers、Trigger level、TRIGGER PROPERTIES

Database event

Database event → 数値

引数	タイプ		説明
この関数には、引数はありません。			
関数の返す値	文字列	←	0=トリガの実行サイクル外 1=新規レコード保存時 2=既存レコード保存時 3=レコード削除時 4=レコード読込時

トリガ内から呼び出されると、**Database event**関数はデータベースイベントの種類、つまりそのトリガが起動された理由を示す数値を返します。

データベースイベントには、次のようなあらかじめ定義された定数があります：

定数	タイプ	値
Save New Record Event	倍長整数	1
Save Existing Record Event	倍長整数	2
Delete Record Event	倍長整数	3
Load Record Event	倍長整数	4

データベースイベントが既存のレコードの保存イベント(Save Existing Record Event)である場合、**Modified**関数を使用して、レコードの特定のフィールドが修正されたかどうかを検出することができます。修正された場合には、**Old**関数を使用して、現在ディスクに格納されている修正前のフィールドの値を検索することもできます。

注：この2つの関数は、英数字や実数といった単純なフィールドに適用される場合にのみ重要です。ピクチャやBOLB、サブテーブルフィールドでは、これらのコマンドの結果は重要ではありません。

トリガ内で、複数のレコードに対するデータベース操作を実行すると、行うべき操作をトリガが正確に実行するのを妨げる条件(通常はロックされたレコード)に出くわす可能性があります。(このような状況の例としては、[送り状]テーブルにレコードを追加中に、[商品]テーブルの複数のレコードを更新する場合が挙げられます。)この時点で、データベース操作を試みるのを中止し、データベースエラーを返して、そのデータベースリクエストが実行できないことを起動プロセスに知らせなければなりません。このとき、起動プロセスは、トランザクションの中で、トリガによって実行された不完全なデータベース操作をキャンセルできなければなりません。この種の状況が発生した場合には、トリガ内から、何らかの操作を試みる前からそのプロセスがトランザクション内にあることを確かめる必要があります。これを行うには、**In transaction**関数を使用します。

波及的(カスケーディング)トリガが呼び出す場合には、4th Dimensionには、使用可能なメモリの容量以外に制限はありません。トリガの実行を最適化するために、データベースイベントだけでなく、トリガが波及的に起動される際の呼び出しのレベルに基づいて、トリガのコードを記述することもできます。例えば、[送り状]テーブルに対する削除データベースイベントの中で、[送り状]レコードの削除が、削除された[顧客]レコードに関連したすべての送り状の削除にともなうものである場合には、 [顧客]総売上げフィールドの更新をスキップすることもできます。

これを行うには、**Trigger level**関数と**TRIGGER PROPERTIES**コマンドを使用します。

例：

Database event関数を使用して、次のようにトリガを作成します：

```
` 任意テーブル用のトリガ
C_LONGINT($0)
$0:=0 `データベースリクエストが許可されると仮定する
Case of
   \ (Database event=Save New Record Event)
      `新規に作成されたレコードの保存のために適切な動作を実行する
   \ (Database event=Save Existing Record Event)
      `既存のレコードの保存のために適切な動作を実行する
   \ (Database event=Delete Record Event)
      `レコードの削除のために適切な動作を実行する
   \ (Database event=Load Record Event)
      `レコードのメモリへのロードのために適切な動作を実行する
End case
```

参照：In transaction、Modified、Old、Trigger level、TRIGGER PROPERTIES

Trigger level

Trigger level → 数値

引数	タイプ		説明
この関数には、引数はありません。			
関数の返す値	文字列	←	トリガの実行レベル トリガの実行サイクル外であれば、0

Trigger level関数は、トリガの実行レベルを返します。

実行レベルについての詳細は、前述の「波及的トリガ」の節を参照してください。

参照：Database event、 TRIGGER PROPERTIES

TRIGGER PROPERTIES

TRIGGER PROPERTIES (トリガレベル ; イベント ; テーブル番号 ; レコード番号)

引数	タイプ		説明
トリガレベル	数値	→	トリガ実行レベル
イベント	数値	←	データベースイベント
テーブル番号	数値	←	影響を受けるテーブル番号
フィールド番号	数値	←	影響を受けるフィールド番号

TRIGGER PROPERTIESコマンドは、ユーザが＜トリガレベル＞に渡すトリガの実行レベルについての情報を返します。トリガ実行レベルの波及に基づいて異なる動作を実行するには、**TRIGGER PROPERTIES**コマンドを**Trigger level**関数と組み合わせて使用します。詳細については、前述の「波及的トリガ」の節を参照してください。

存在しないトリガレベルを渡すと、コマンドはすべての引数に 0を返します。

渡されたトリガ実行レベルのデータベースイベントの種類が、引数＜イベント＞に返されます。

データベースイベントには、次のようなあらかじめ定義された定数があります：

定数	タイプ	値
Save New Record Event	倍長整数	1
Save Existing Record Event	倍長整数	2
Delete Record Event	倍長整数	3
Load Record Event	倍長整数	4

渡されたトリガ実行レベルのデータベースイベントに関係するレコードのテーブル番号とレコード番号が引数＜テーブル番号＞と＜レコード番号＞に返されます。

注：トランザクション中、新規に作成されたレコードには一時的なレコード番号が与えられることを覚えておいてください。

参照：Database event、Trigger level

ユーザインタフェースコマンド　58

この章では、「ルーチン」エディタの「User Interface」テーマ内にあるユーザインタフェースコマンドについて説明します。この章のコマンドは、フィールドや変数などの入力フォームエリアに影響を与えます。例えば、これらのコマンドはフォームが表示や印刷される場合にのみ有効なフォームオブジェクトを変更します。このオブジェクトは新しいフォームやレコードが表示されると、「デザイン」モードで割り当てられた属性に戻ります。これらのコマンドはフォームメソッドやオブジェクトメソッドで使用されます。

BEEP
Caps lock down
GET HIGHLIGHT
GET MOUSE
HIGHLIGHT TEXT
INVERT BACKGROUND
Get platform interface
Macintosh command down
Macintosh control down
Macintosh option down
Last object
PLAY
Pop up menu
POST CLICK
POST EVENT
POST KET
REDRAW
SET CURSOR
SET FIELD TITLES
SET PLATFORM INTERFACE
SET TABLE TITLES
Shift down
Windows Ctrl down
Windows Alt down

HIGHLIGHT TEXT

HIGHLIGHT TEXT (テキストオブジェクト; 先頭 ; 最終)

引数	タイプ	説明
テキストオブジェクト	フィールド	反転表示するテキストオブジェクトまたは変数
先頭	数値	反転表示の先頭位置
最終	数値	反転表示の最終位置

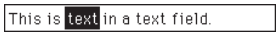
HIGHLIGHT TEXTコマンドは、＜テキストオブジェクト＞の中のテキストの一部を反転表示します。**HIGHLIGHT TEXT**コマンドは、挿入ポインタ(カーソル)が＜テキストオブジェクト＞上にない場合には、指定した＜テキストオブジェクト＞に進みます。**HIGHLIGHT TEXT**コマンドを組み込みエリアのフィールドで使用することはできません。

＜先頭＞は、反転表示する先頭文字の位置です。＜最終＞は、反転表示する最終文字の位置です。＜先頭＞と＜最終＞が同じ場合は、挿入ポインタ(カーソル)が＜先頭＞で指定された文字の前に置かれ、文字は全く反転表示されません。

＜最終＞が＜テキストオブジェクト＞の文字数より大きい場合は、＜先頭＞からテキストの最終までのすべての文字を反転表示します。

▼ 次の例は、フィールド“ コメント ”のテキストを反転表示します。結果を次の図に示します：

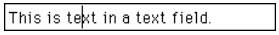
```
HIGHLIGHT TEXT (コメント ; 9 ; 13)` テキストの反転表示
```



フィールド上のテキストの反転表示

▼ 次の例は、フィールド“ コメント ”上に挿入ポインタ(カーソル)を表示します。結果を次の図に示します：

```
HIGHLIGHT TEXT(コメント ; 11 ; 11)` コメントフィールドの挿入ポイントの位置
```



フィールド上の挿入ポインタ(カーソル)の位置

GET HIGHLIGHT

GET HIGHLIGHT (テキストオブジェクト; 先頭 ; 最終)

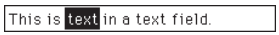
引数	タイプ	説明
テキストオブジェクト	フィールド	テキストのフィールドまたは変数
先頭	変数	反転表示された先頭位置
最終	変数	反転表示された最終位置

GET HIGHLIGHTコマンドは、現在反転表示されているテキストを検出するために使用します。テキストの反転は、ユーザによる指定や**HIGHLIGHT TEXT**コマンドの実行で行うことができます。このコマンドは組み込みエリアのフィールドで使用することはできません。

変数＜先頭＞に、反転表示された文字の先頭位置を代入します。変数＜最終＞に、反転表示された最終の文字位置に1を加えた値を代入します。＜先頭＞と＜最終＞が同じ場合は、テキストは選択されていない状態です。この場合の挿入ポインタ(カーソル)は、＜先頭＞の1文字前にあります。

▼ 次の例は、フィールド“ コメント ”の反転表示された部分を検出します。**GET HIGHLIGHT**コマンドは、“ v先頭 ”と“ v最終 ”の2つの変数に値を代入します。次の図の場合、“ v先頭 ”に9が代入され、“ v最終 ”に13が代入されます：

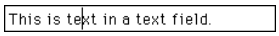
```
GET HIGHLIGHT (コメント ;v先頭 ; v最終) ` コメントの反転表示部分を求める
```



フィールド上で反転表示されたテキスト

▼ 次の例は、次の図のように、フィールド“ コメント ”に反転表示されたテキストがない場合です。この場合、変数“ v先頭 ”と“ v最終 ”の両方に11が代入されます：

```
GET HIGHLIGHT (コメント ; v先頭 ; v最終) ` コメントの挿入ポインタを求める
```



フィールド上のテキスト挿入ポイント(カーソル)

▼ 次の例は、**Substring**関数を使用して反転表示されたテキストを取り出します：

```
GET HIGHLIGHT (コメント ; v先頭 ; v最終) ` コメントの反転表示部分を求める
` Substring関数を使用して変数“ vテキスト ”に反転表示部分を代入する
vテキスト:=Substring (コメント ; v先頭 ; v最終 - v先頭)
```

INVERT BACKGROUND

INVERT BACKGROUND (テキスト変数)

引数	タイプ	説明
テキスト変数	変数	バックグラウンドに配置される変数

INVERT BACKGROUNDコマンドは、フォーム中の＜テキスト変数＞を背景に配置するために使用します。このコマンドは、現在表示中または印刷中のフォームやレコードでのみ機能します。また、画面に表示する場合にのみ使用することができます。LaserWriterプリンタを使用してバックグラウンド印刷をすることはできません。出力フォーム内の変数を背景に配置することはできません。**INVERT BACKGROUND**コマンドは、入力可能な変数に対しては使用することができません。文字入力は部分的に背景を消すだけです。

▼ 次の例は、フィールドの値を判定する出力フォーム内の変数のオブジェクトメソッドです。このオブジェクトメソッドは、フィールドの値が正の場合は何も実行せず、負の場合はフォーム内にある変数の表示を背景に配置します：

```
vAmount:=[Accounts]Amount          ` 変数にフィールドの値を代入する
If (vAmount<0)                     ` 変数の値が負の場合
  INVERT BACKGROUND (vAmount)      ` 変数を背景にする
End if
```

音の発生

この節のコマンドは、PCまたはMacintoshのスピーカを通じて音を発生します。

BEEP

BEEP

BEEPコマンドは、ビープ音を発生します。発生するビープ音は、コントロールパネルで変更することができます。

▼ 次の例は、コントロールパネルで設定されたビープ音を発生します：

```
BEEP
```

PLAY

PLAY (サウンド名 ; {チャネル})

引数	タイプ	説明
サウンド名	文字列	発生する音の名前
チャネル	数値	シンセサイザチャネル

Macintosh上：
PLAYコマンドは、＜サウンド名＞で指定したサウンドリソースの音を発生します。

引数＜チャネル＞は、Macintoshのシンセサイザチャネルを指定します。＜チャネル＞を指定しない場合には、単純デジタル音の同期指定とみなします。同期指定の場合には、発生した音が終了するまで以降の処理をすべて中断します。＜チャネル＞に1を指定した場合には、単純デジタル音の非同期指定とみなします。非同期指定の場合には、音をバックグラウンドで発生し、以降の処理を通常どおり実行します。

非同期指定の音の発生を終了するためには、次のようにします：

```
PLAY ("" ; 0)
```

▼ 次の例は、On Startupデータベースメソッドに含まれるものです。“ ウエルカム ” の音でユーザを歓迎します：

```
PLAY ("ウエルカム")                    ` "ウエルカム"の音を発生する
```

REDRAW

REDRAW (サブテーブル)

引数	タイプ	説明
サブテーブル	テーブル	書き直す領域またはサブテーブル

メソッドを使用して、サブフォームで表示されるフィールドやサブフィールドの値を変更する場合に、フォームを確実に変更するために**REDRAW**コマンドを使用します。

SET PLATFORM INTERFACE

SET PLATFORM INTERFACE (インタフェース)

引数	タイプ	説明
インタフェース	数値	新しい プラットフォームインタフェース設定 -1 自動 0 MacOS（漢字Talk7） 1 Windows3.1 2 Windows 95 3 Copland

SET PLATFORM INTERFACEコマンドは、フォームの表示用にプラットフォームインタフェースを設定します。

4th Dimensionでは、次の表のように前もって定義された定数の1つを引数＜インタフェース＞に渡します：

定数	タイプ	値
Automatic interface	倍長整数	-1
Macintosh interface	倍長整数	0
Windows 3.1 interface	倍長整数	1
Windows 95 interface	倍長整数	2
Copland interface	倍長整数	3

コマンドに渡した値が現在のものと同じなら、何も起こりません。

注：プラットフォームインタフェースの設定は、「デザイン」モードの「データベースプロパティ」ダイアログボックスで変更することもできます。

例：

4D のクライアント/サーバ アーキテクチャでは、Macintosh と Windows ステーションが同時に異なるプラットフォームインタフェースを使用することが可能です。**SET PLATFORM INTERFACE**コマンドをこの目的のためにデータベースの「On startup」データベースメソッドで利用することができます。

```
` この例題では、ユーザの環境設定は [環境設定]テーブルに納められています。
` Current User に対応するレコードを探します
QUERY ([環境設定] ; [環境設定]ユーザ名=Current User)
If (Records in selection ([環境設定]) = 0)
` 見つからない場合、デフォルトの設定を探します
 QUERY ([環境設定] ; [環境設定]ユーザ名 = "デフォルト")
End if
` ユーザの環境設定に対応するプラットフォームインタフェースを設定します
SET PLATFORM INTERFACE ([環境設定]プラットフォーム)
```

参照：Get platform interface

Get platform interface

Get platform interface → 数値

引数	タイプ		説明
この関数には、引数はありません。			
関数の返す値	数値	←	現在使用中のプラットフォームインタフェース

Get platform interface関数は、現在使用しているプラットフォームインタフェースを表す数値を返します。

Get platform interface関数は、下記の値の1つを返します：

定数	タイプ	値
Automatic interface	倍長整数	-1
Macintosh interface	倍長整数	0
Windows 3.1 interface	倍長整数	1
Windows 95 interface	倍長整数	2
Copland interface	倍長整数	3

SET PLATFORM INTERFACEコマンドまたは「デザイン」モードの「データベースプロパティ」ダイアログボックスでプラットフォームインタフェースを変更することができます。

参照：SET PLATFORM INTERFACE

Shift down

Shift down → ブール

引数	タイプ		説明
この関数には、引数はありません。			
関数の返す値	ブール	←	Shiftキーのステータス

Shift down関数は、Shiftキーが押されているとTrue（真）を返します。

例：

ボタン「bAnyButton」用の次のオブジェクトメソッドは、ボタンがクリックされたときにどのモディファイアキーが押されているかによって、異なる動作（アクション）を実行します：

```
` 「bAnyButton」ボタンのオブジェクトメソッド
Case of
      `このほかの複数のキーの組み合わせをここでテストすることも可能
      ...
   \ (Shift down & Windows Ctrl down)
      `ShiftおよびWindows Ctrl (または Macintosh Command) キーが押されてた場合
      DO ACTION1
      ...
   \ (Shift down)
      `Shiftキーだけが押されてた場合
      DO ACTION2
      ...
   \ (Windows Ctrl down)
      `Windows Ctrl（またはMacintosh Command）キーだけが押された場合
      DO ACTION3
      ...
      `このほかの個々のキーをここでテストすることも可能
      ...
End case
```

参照：Caps lock down、Macintosh command down、Macintosh control down、Macintosh option down、Windows Alt down、Windows Ctrl down

Caps lock down

Caps lock down → ブール

引数	タイプ		説明
この関数には、引数はありません。			
関数の返す値	ブール	←	Caps Lockキーのステータス

Caps lock down関数は、Caps LockキーがTrue（真）を返します。

例：

Shift down関数の例を参照してください。

参照：Macintosh command down、Macintosh control down、Macintosh option down、Shift down、Windows Alt down、Windows Ctrl down

Windows Ctrl down

Windows Ctrl down → ブール

引数	タイプ		説明
この関数には、引数はありません。			
関数の返す値	ブール	←	WindowsのCtrlキーのステータス (Macintoshの場合は、Commandキー)

Windows Ctrl down関数は、WindowsのCtrlキーが押されているとTrue（真）を返します。

注： Macintoshプラットフォーム上で呼び出された場合は、Windows Ctrl down関数はMacintoshの Commandキーが押されているとTrue（真）を返します。

例：

Shift down関数の例を参照してください。

参照：Caps lock down、Macintosh command down、Macintosh control down、Macintosh option down、Shift down、Windows Alt down

Windows Alt down

Windows Alt down → ブール

引数	タイプ		説明
この関数には、引数はありません。			
関数の返す値	ブール	←	WindowsのAltキーのステータス (Macintoshの場合は、optionキー)

Windows Alt down関数は、WindowsのAltキーが押されているとTrue（真）を返します。

注： Macintoshプラットフォーム上で呼び出された場合は、Windows Alt down関数はMacintoshの optionキーが押されているとTrue（真）を返します。

例：

Shift down関数の例を参照してください。

参照：Caps lock down、Macintosh command down、Macintosh control down、Macintosh option down、Shift down、Windows Ctrl down

Macintosh command down

Macintosh command down → ブール

引数	タイプ		説明
この関数には、引数はありません。			
関数の返す値	ブール	←	Macintoshの commandキーのステータス (Windowsの場合は、Ctrlキー)

Macintosh command down関数は、Macintoshの commandキーが押されているとTrue（真）を返します。

注：Windowsプラットフォーム上で呼び出された場合は、Macintosh command down関数はWindowsの Ctrlキーが押されているとTrue（真）を返します。

例：

Shift down関数の例を参照してください。

参照：Caps lock down、Macintosh control down、Macintosh option down、Shift down、Windows Alt down、Windows Ctrl down

Macintosh option down

Macintosh option down → ブール

引数	タイプ		説明
この関数には、引数はありません。			
関数の返す値	ブール	←	Macintoshの optionキーのステータス (Windowsの場合は、Altキー)

Macintosh command down関数は、Macintoshの optionキーが押されているとTrue（真）を返します。

注：Windowsプラットフォーム上で呼び出された場合は、Macintosh option down関数はWindowsの Altキーが押されているとTrue（真）を返します。

例：

Shift down関数の例を参照してください。

参照：Caps lock down、Macintosh command down、Macintosh control down、Shift down、Windows Alt down、Windows Ctrl down

Macintosh control down

Macintosh control down → ブール

引数	タイプ		説明
この関数には、引数はありません。			
関数の返す値	ブール	←	Macintoshの controlキーのステータス

Macintosh control down関数は、Macintoshの controlキーが押されているとTrue（真）を返します。

注：Windowsプラットフォーム上で呼び出された場合は、Macintosh control down関数はFalse（偽）を返します。このMacintoshと同等のキーは、Windows上にはありません。

例：

Shift down関数の例を参照してください。

参照：Caps lock down、Macintosh command down、Macintosh option down、Shift down、Windows Alt down、Windows Ctrl down

GET MOUSE

GET MOUSE (水平 ; 垂直 : マウスボタン { ; *})

引数	タイプ		説明
水平	数値	←	**マウスの水平座標**
垂直	数値	←	**マウスの垂直座標**
マウスボタン	数値	←	**マウスボタンのステータス** **0 = 何もしていない** **1 = ボタンの押下** **2 = 右マウスボタンの押下（Windowsのみ）** **3 = 両方のボタンの押下（Windowsのみ）**
*		→	**指定した場合、ローカルの座標システムが使用される。** **省略した場合、グローバルの座標システムが使用される。**

GET MOUSEコマンドは、マウスの現在の状態を返します。

水平座標と垂直座標が、引数＜水平＞と＜垂直＞に返されます。オプション引数＜*＞を省略すると、これらの座標は画面（スクリーン）に対して相対的に表されます。引数 * を渡すと、座標はカレントプロセスの最前面のウィンドウに対して相対的に表されます。

引数＜マウスボタン＞は、上記のようにボタンの状態を返します。

例：

Pop up menu関数を参照してください。

参照：Caps lock down、Macintosh command down、Macintosh control down、Macintosh option down、ON EVENT CALL、Shift down、Windows Alt down、Windows Ctrl down

Pop up menu

Pop up menu (項目テキスト { ;デフォルト})

引数	タイプ		説明
項目テキスト	テキスト	→	**定義された項目テキスト**
デフォルト	数値	→	**デフォルトで選択された項目番号**
関数の返す値	数値	←	**選択された項目番号**

Pop up menu関数は、マウスの現在の位置にポップアップメニューを表示します。

ユーザインタフェースの規則に従うために、通常このコマンドは、マウスクリックに応じて、またマウスボタンが押されたままの状態である場合に呼び出します。

ポップアップメニューの項目（アイテム）は、次のように引数＜項目＞を使用して定義します：

■ 各項目の間をセミコロン「 ; 」で区切ります。例えば、“ 項目テキスト1 ; 項目テキスト2 ; 項目テキスト3 ”。

■ 項目を選択不可にするには、項目テキスト内に開いた括弧 「 (」を指定します。

■ 区切り線を指定するには、項目テキストとして 「(-」を渡します。

■ 行のフォントスタイルを指定するには、項目テキスト内に記号「<」に続けて以下の文字のいずれかを指定します：

<B	**太字（ボールド）**
<I	**斜体（イタリック）**
<U	**アンダーライン**
<O	**アウトライン**
<S	**シャドウ**

■ 項目にチェックマークを付けるには、項目テキスト中に疑問符「 ? 」に続けてチェックマークとして表示したい文字（キャラクタ）を指定します。 Macintoshの場合、その文字が表示されます。Windowsの場合は、どのような文字を渡したかにかかわらず、チェックマークが表示されます。

■ 項目にアイコンを追加するには、項目テキスト内にアクセント (^) に続けて、そのASCIIコードから48を引いたMacOSベースのアイコンリソースのリソースIDである文字を指定します。

■ 項目にショートカットを追加するには、項目テキスト中にスラッシュ「 / 」に続けてその項目のためのショートカット文字を指定します。ただし、この最後のオプションは

純粋に情報提供のためのものであることに注意してください。ショートカットがポップアップメニューをアクティブにすることはありません。ただし、アプリケーションのメインメニューバーにそのポップアップメニュー項目に相当するものがある場合には、ショートカットを含めることも可能です。

オプション引数の＜デフォルト＞によって、ポップアップメニューが表示される際に選択される省略時（デフォルト）のメニュー項目を指定することができます。1からメニュー項目の数までの値を渡してください。この引数を省略すると、コマンドは最初のメニュー項目を省略時の設定として選択します。

メニュー項目を選択した場合には、コマンドはその番号を返します。選択しなければ、ゼロ (0)を返します。

注：適当な項目数のポップアップメニューを使用してください。50以上もの項目を表示したい場合は、ポップアップメニューではなく、フォーム内のスクロール可能エリアの使用を検討するほうが賢明です。

例：

プロジェクトメソッド「MY SPEED MENU」は、ナビゲーションスピードメニューをプルダウンします：

```
`「MY SPEED MENU」プロジェクトメソッド
GET MOUSE($vlMouseX ; $vlMouseY ; $vlButton)
If (Macintosh control down |  ($vlButton=2))
    $vtItems:="このデータベースについて...<l;(-;!-その他のオプション;(-"
    For ($vlTable ; 1 ; Count tables)
        $vtItems:=$vtItems+";"+Table name($vlTable)
    End for
    $vlUserChoice:=Pop up menu($vtItems)
    Case of
        \ ($vlUserChoice=1)
            `情報を表示する
        \ ($vlUserChoice=2)
            `オプションを表示する
    Else
        If ($vlUserChoice>0)
            ` 番号が$vlUserChoice-4のテーブルに移動する。
        End if
    End case
End if
```

このプロジェクトメソッドは、以下から呼び出せます：

■ マウスボタンが解除されるのを待たずにマウスクリックに反応するフォームオブジェクトのメソッド（透明ボタン）

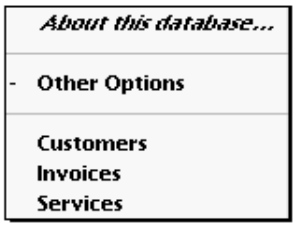
■ イベントを「スパイ」し、他のプロセスと通信するプロセス

■ **ON EVENT CALL**コマンドを使用してインストールされるイベント処理メソッド

最後の2つのケースでは、フォームオブジェクトにおいてクリックが行われる必要はありません。これは、**Pop up menu**関数の利点の1つです。ポップアップメニューを表示するにはフォームオブジェクトを使用します。ポップアップメニューを使えば、どこにでもメニューを表示できます。

Windowsでは、ポップアップメニューはマウスの右ボタンを押すことによって表示されます。 Macintoshでは、「Control-クリック」を押すことによって表示されます。ただし、このメソッドは、実際にはマウスがクリックされたかどうかをチェックしないことに注意してください。このテストは呼び出し側のメソッドが行います。

次の図は、Windows（左）上とMacintosh（右）上で表示されるポップアップメニューの例です。Windows版では、標準のチェックマークが使われていることに注意してください。



参照：GET MOUSE

SET CURSOR

SET CURSOR ({カーソル})

引数	タイプ		説明
カーソル	数値	←	**MacOSベースのカーソルリソース番号**

SET CURSORコマンドは、マウスカーソルを引数＜カーソル＞に渡したID番号を持つMacOSベースの‘CURS’リソースに格納されたカーソルに変更します。

引数を省略すると、マウスカーソルは標準の矢印に設定されます。

使用可能なカーソルのリストを得るには、**RESOURCE LIST**コマンドを使用してください。

参照：RESOURCE LIST

Last object

Last object → ポインタ

Last object関数は、最後に操作されたオブジェクトまたはカレント入力可能エリアのポインタを返します。**Last area**関数を使用すれば、現在選択されているオブジェクトを知らなくても、そのフォームエリアに対して動作を実行することができます。オブジェクトに対して動作を行う前に**Type**関数を使用して、オブジェクトが正しいデータタイプかどうかを確認してください。**Last object**関数は、組み込みエリアのフィールドで使用することはできません。

▼ 次の例は、ボタンのオブジェクトメソッドです。このメソッドは、カレントオブジェクトのデータを大文字に変換します。この場合、オブジェクトのデータタイプは、テキストまたは文字列でなければなりません(データタイプが0または24)：

```
$p:=Last object                          ` ポインタを保存する
` データタイプが文字列またはテキストエリアの場合
If ((Type ($p->)=0) | (Type ($p->)=24))
  $p->:=Uppercase ($p->)                 ` エリアのテキストを大文字に変換する
End if
```

注：この関数はバージョン3の4th DimensionのLast area関数とまったく同じです。

POST KEY

POST KEY (コード {; モディファイア {; プロセス}})

引数	タイプ		説明
テキスト	数値	→	文字のASCIIコード、 またはファンクションキーコード
モディファイア	数値	→	モディファイアキーのステータス
プロセス	数値	→	送信先のプロセス参照番号、 省略または0の場合、アプリケーションイベントキュー

POST KEYコマンドは、キーストロークをシミュレート（模倣）します。これは、ユーザが実際にキーボード上で文字を入力した場合と同様の結果を生じます。

引数＜コード＞には、その文字のASCIIコードを渡します。

オプション引数＜モディファイア＞を渡す場合には、イベント（モディファイア）定数を1つあるいは組み合わせて渡します。例えば、Shiftキーをシミュレートするには、Shiftキービットを渡します。引数＜モディファイア＞を渡さなかった場合には、モディファイアはシミュレートされません。

オプション引数＜プロセス＞を指定すると、キーストロークは、＜プロセス＞に渡したプロセス番号を持つプロセスに送られます。0（ゼロ）を渡すか、引数を省略した場合には、キーストロークはアプリケーションレベルに送られ、4Dスケジューラーがそれをアプリケーションプロセスにディスパッチします。

▼ **Process number**コマンドの例を参照してください。

参照：POST CLICK、POST EVENT

POST CLICK

POST CLICK (マウスX ; マウスY {; プロセス}{; *})

引数	タイプ		説明
マウスX	数値	→	水平座標
マウスY	数値	→	垂直座標
プロセス	数値	→	送信先のプロセス参照番号、 省略または0の場合、アプリケーションイベントキュー
*		→	指定された場合はローカルな座標システムが使用される 省略された場合はグローバルな座標システムが使用される

POST CLICKコマンドはマウスクリックをシミュレート（模倣）します。これは、ユーザが実際にマウスボタンをクリックした場合と同様の結果を生じます。

引数＜マウスX＞と＜マウスY＞には、クリックの水平座標と垂直座標を渡します。オプション引数 ＜ * ＞を渡した場合、これらの座標は、＜プロセス＞に渡したプロセス番号を持つプロセスの最前面のウィンドウに対して相対的に表されます。引数＜ * ＞を省略すると、これらの座標は画面（スクリーン）に対して相対的に表されます。

オプション引数＜プロセス＞を指定すると、クリックは、＜プロセス＞に渡したプロセス番号を持つプロセスに送られます。0（ゼロ）を渡すか、引数を省略した場合には、クリックはアプリケーションレベルに送られ、4Dスケジューラーがそれをアプリケーションプロセスにディスパッチします。

参照：POST EVENT、POST KEY

POST EVENT

POST EVENT (タイプ ; メッセージ ; 時間 ; マウスX ; マウスY ; モディファイア {; プロセス})

引数	タイプ		説明
タイプ	数値	→	イベントの種類
メッセージ	数値	→	イベントメッセージ
時間	数値	→	イベント時間（チック単位）
マウスX	数値	→	水平座標
マウスY	数値	→	垂直座標
モディファイア	数値	→	モディファイアキーのステータス
プロセス	数値	→	送信先のプロセス参照番号、 省略または0の場合、アプリケーションイベントキュー
*		→	指定された場合はローカルな座標システムが使用される 省略された場合はグローバルな座標システムが使用される

POST EVENTコマンドは、キーボードまたはマウスイベントをシミュレート（模倣）します。これは、ユーザが実際にキーボードやマウス上で動作を行った場合と同様の結果を生じます。

引数＜タイプ＞には、次のようなあらかじめ定義された定数のいずれかを渡します：

定数	タイプ	値
Mouse down event	倍長整数	1
Mouse up event	倍長整数	2
Key down event	倍長整数	3
Key up event	倍長整数	4
Auto key event	倍長整数	5

イベントがマウス関連のイベントであれば、引数＜メッセージ＞に0（ゼロ）を渡します。イベントがキーボード関連のイベントであれば、シミュレートされる文字のASCIIコードを＜メッセージ＞に渡します。

引数＜時間＞には、通常、**Tickcount**関数によって返される値を渡します。

イベントがマウス関連のイベントであれば、クリックの水平座標と垂直座標を＜マウスX＞と＜マウスY＞に渡します。オプション引数＜*＞を省略すると、これらの座標は画面（スクリーン）に対して相対的に表されます。引数＜*＞を渡した場合、これらの座標は、＜プロセス＞に渡したプロセス番号を持つプロセスの最前面のウィンドウに対して相対的に表されます。

引数＜モディファイア＞には、イベント（モディファイア）定数を1つあるいは組み合わせて渡します。例えば、Shiftキーをシミュレートするには、 Shiftキービットを渡します。

オプション引数＜プロセス＞を指定すると、イベントは、＜プロセス＞に渡したプロセス番号を持つプロセスに送られます。0（ゼロ）を渡すか、引数を省略した場合には、イベントはアプリケーションレベルに送られ、4Dスケジューラーがそれをアプリケーションプロセスにディスパッチします。

参照：POST CLICK、POST KEY

SET TABLE TITLES

SET TABLE TITLES (テーブルタイトル ; テーブル番号)

引数	タイプ		説明
テーブルタイトル	文字列配列	→	ダイアログボックスに現れるテーブル名
テーブル番号	数値配列	→	実際のテーブル番号

SET TABLE TITLESコマンドを使用すれば、「ユーザ」モードまたは「カスタム」モードにおいて、「クエリ」エディタのような標準の4thDimensionダイアログボックスにデータベースのテーブルを表示する際に、それらのテーブルのマスク、名前の変更、並べ替えが行えます。

引数＜テーブルタイトル＞と＜テーブル番号＞の配列は同期させる必要があります。配列＜テーブルタイトル＞には、テーブルの名前を、表示したい名前で渡します。ある特定のテーブルを表示したくない場合には、そのテーブル名または新しく付けたタイトルを、配列に指定しないでください。テーブルは、この配列に指定した順序で表示されます。配列＜テーブル番号＞の各要素には、配列＜テーブルタイトル＞の同じ番号の要素に渡されるテーブル名または新規タイトルに対応する実際のテーブル番号を渡します。

例えば、テーブルA、B、Cで構成されるデータベースがあり、テーブルはこの順序で作成されたとします。表示の際にはこれらのテーブルをX、Y、Z という名前にし、さらにテーブルBは表示したくないとします。最終的に、ZとXを、この順序で表示することにします。この場合、配列＜テーブルタイトル＞2つの要素としてZとXを渡し、配列＜テーブル番号＞の2つの要素として3と1を渡します。

SET TABLE TITLESコマンドはデータベースの実際の構造（ストラクチャ）を変更するわけではありません。これは、「ユーザ」モードまたは「カスタム」モードにおいて、「クエリ」エディタのような標準の4th Dimensionダイアログボックスを次回使用する際にのみ影響します。**SET TABLE TITLES**コマンドの有効範囲は、そのワークセッションです。クライアント／サーバにおける利点の1つは、複数の4Dクライアントステーションがそれぞれ異なる見方でサーバのデータベースを同時に「見る」ことができる点です。

SET TABLE TITLESコマンドは、何度でも呼び出すことができます。ただし、これは標準の4th Dimensionダイアログボックスの次回の表示にしか影響しないことに注意してください。

SET TABLE TITLESコマンドは、次のよう場合に使用します：

■ データベースを動的にローカライズする。
■ データベースの実際の定義に関係なく、自由にテーブルを表示する。
■ 固有のユーザまたは任意ユーザのカスタム特権に応じてテーブルの表示を変える。

警告：SET TABLE TITLESコマンドはテーブルの非表示属性を上書きしません。データベースの設計（デザイン）レベルでテーブルが非表示として設定されている場合、SET TABLE TITLESコマンドへの呼び出しにそのテーブルを指定しても、テーブルは表示されません。

例：

■ 各国で販売する予定の4Dアプリケーションを構築しているとします。この場合、ローカライズの問題を慎重に考慮する必要があります。「ユーザ」モードまたは「カスタム」モードで表示される標準の4th Dimensionダイアログボックスに注意すれば、 [翻訳]テーブルといくつかのプロジェクトメソッドを使用して、必要なだけ各国向けにローカライズされたフィールドを作成し、使用することによって、ローカライズのニーズに対応できます。

■ データベースに次のテーブルを追加します：

翻訳	
実際の名前	A
言語	A
翻訳された名前	A

■ 次に、以下に示した「TRANSLATE TABLES AND FIELDS」プロジェクトメソッドを作成します。このメソッドはデータベースの実際のストラクチャをブラウズし、引数として渡される言語に対応するローカライズ版の作成に必要なすべての[翻訳]レコードを作成します。

```
`「TRANSLATE TABLES AND FIELDS」プロジェクトメソッド
   ` TRANSLATE TABLES AND FIELDS (文字列)
   ` TRANSLATE TABLES AND FIELDS (言語 )
C_STRING(31 ; $1)
C_LONGINT($vlTable ; $vlField)
For ($vlTable ; 1 ; Count tables) `テーブルをループする
   `指定された言語用のテーブル名の翻訳があるかどうかをチェックする
   QUERY([翻訳];[翻訳]実際の名前=Table name($vlTable) ; *)
   QUERY([翻訳] & [翻訳]言語=$1)
   If (Records in selection([翻訳])=0)
      ` なければ、レコードを作成する
      CREATE RECORD([翻訳])
      [翻訳]実際の名前:=Table name($vlTable)
      [翻訳]言語:=$1
      `翻訳したテーブル名を入力する必要がある
      SAVE RECORD([翻訳])
   End if
   For($vlField ; 1 ; Count fields($vlTable))
      `指定された言語用のフィールド名の翻訳があるかどうかをチェックする
      QUERY([翻訳];[翻訳]実際の名前=Field name($vlTable ; $vlField) ; *)
      QUERY([翻訳] & [翻訳]言語=$1)
      If (Records in selection([翻訳])=0)
         `なければ、レコードを作成する
         CREATE RECORD([翻訳])
         [翻訳]実際の名前:=Field name($vlTable ; $vlField)
         [翻訳]言語:=$1
         `翻訳したフィールド名を入力する必要がある
         SAVE RECORD([翻訳])
      End if
   End for
End for
```

■ この時点で次の行を実行すれば、テーブルタイトルとフィールドタイトルの日本語のローカライズ版に必要なすべてのレコードが作成されます。

```
TRANSLATE TABLES AND FIELDS ("Japanese")
```

■ この呼び出しの実行後、新しく作成されたレコードのそれぞれに対して“ [翻訳]翻訳された名前 ”を入力できます。

■ 最後に、日本語のローカライズ版を使用してデータベースの標準の4Dダイアログボックスを表示したい場合はいつでも、「LOCALIZED TABLES AND FIELDS」プロジェクトメソッドを使ってに次の行を実行します：

```
LOCALIZED TABLES AND FIELDS ("Japanese")
```

```
` 「LOCALIZED TABLES AND FIELDS 」プロジェクトメソッド
` LOCALIZED TABLES AND FIELDS (文字列)
` LOCALIZED TABLES AND FIELDS (言語)
C_STRING(63 ; $1)
C_LONGINT($vlTable ; $vlNbTable ; $vlField ; $vlNbField)
$vlNbTable:=Count tables `データベースに存在するテーブル数を数える
` SET TABLE TITLESコマンドに渡される配列を初期化する
ARRAY STRING(31 ; $asTableName ; $vlNbTable)
ARRAY INTEGER($aiTableNumber ; $vlNbTable)
For ($vlTable ; 1 ; $vlNbTable) ` テーブルをループする
    $asTableName{$vlTable}:=Table name($vlTable) `テーブル名を取得する
    $aiTableNumber{$vlTable}:=$vlTable `実際のテーブル番号を格納する
    QUERY([翻訳];[翻訳]実際の名前=$asTableName{$vlTable} ; *)`翻訳テーブルを検索する
    QUERY([翻訳] & [翻訳]言語=$1)
    If (Records in table([翻訳])>0)
        `可能なら、ローカライズされたテーブル名を使用する
        $asTableName{$vlTable}:=[翻訳]翻訳された名前
    End if
    $vlNbField:=Count fields($vlTable) `そのテーブルのフィールド数を取得する
    ` SET TABLE TITLESコマンドに渡される配列を初期化する
    ARRAY STRING(31 ; $asFieldName ; $vlNbTable)
    ARRAY INTEGER($aiFieldNumber ; $vlNbTable)
    For ($vlField ; 1) `フィールドをループする
        $asFieldName{$vlField}:=Field name($vlTable ; $vlField)`フィールド名を取得する
        $aiFieldNumber{$vlField}:=$vlField `実際のフィールド番号を格納する
        QUERY([翻訳];[翻訳]実際の名前=$asFieldName{$vlField} ; *)
        `翻訳テーブルを検索する
        QUERY([翻訳] & [翻訳]言語=$1)
        If (Records in table([翻訳])>0)
            `可能なら、ローカライズされたフィールド名を使用する
            $asFieldName{$vlField}:=[翻訳]翻訳された名前 `
        End if
    End for
    SORT ARRAY($asFieldName ; $aiFieldNumber ; >)
    SET FIELD TITLES(Table($vlTable)-> ; $asFieldName ; $aiFieldNumber)
End for
SORT ARRAY($asTableName ; $aiTableNumber ; >)
SET TABLE TITLES($asTableName ; $aiTableNumber)
```

■ 新しいローカライズ版は、コードの修正や再コンパイルを行うことなく、データベースに追加できることに注意してください。

参照：Count tables、SET FIELD TITLES、Table name

SET FIELD TITLES

SET FIELD TITLES (テーブル|サブテーブル ; フィールドタイトル ; フィールド番号)

引数	タイプ		説明
テーブル、または サブテーブル	テーブル、または サブテーブル	→	フィールドタイトルを設定するテーブル またはサブテーブル
フィールドタイトル	文字列配列	→	ダイアログボックスに現れるフィールド名
フィールド番号	数値配列	→	実際のフィールド番号

SET FIELD TITLESコマンドを使用すれば、「ユーザ」モードまたは「カスタム」モードにおいて、「クエリ」のような標準の4th Dimensionダイアログボックスに表示する際に、テーブルやサブテーブルに渡される、そのテーブルやサブテーブルのフィールドのマスク、名前の変更、並べ替えが行えます。

引数＜フィールドタイトル＞と＜フィールド番号＞配列は同期させる必要があります。配列＜フィールドタイトル＞には、フィールドの名前を、表示したい名前で渡します。ある特定のフィールドを表示したくない場合には、そのフィールド名または新しく付けたタイトルを、配列に指定しないでください。フィールドは、この配列に指定した順序で表示されます。配列＜フィールド番号＞の各要素には、配列＜フィールドタイトル＞の同じ番号の要素で渡されるフィールド名または新規タイトルに対応する実際のフィールド番号を渡します。

例えば、フィールドF、G、Hで構成されるテーブルまたはサブテーブルがあり、フィールドはこの順序で作成されたとします。表示の際には、これらのフィールドをM、N、Oという名前にし、さらにフィールドNは表示したくないとします。最終的に、 OとMを、この順序で表示することにします。この場合、配列＜フィールドタイトル＞の2つの要素としてOとMを渡し、配列fieldNumbersの2つの要素として3と1を渡します。

SET FIELD TITLESコマンドは、テーブルの実際の構造（ストラクチャ）を変更するわけではありません。これは、「ユーザ」モードまたは「カスタム」モードにおいて、「クエリ」エディタのような標準の4th Dimensionダイアログボックスを次回使用する際にのみ影響します。**SET FIELD TITLES**コマンドの有効範囲は、そのワークセッションです。クライアント／サーバにおける利点の1つは、複数の4Dクライアントステーションがそれぞれ異なる見方でサーバのテーブルを同時に「見る」ことができる点です。**SET FIELD TITLES**コマンドは、何度でも呼び出せます。ただし、これは標準の4th Dimensionダイアログボックスの次回の表示にしか影響しないことに注意してください。

SET FIELD TITLESコマンドは、次のような場合に使用します：

- データベースを動的にローカライズする。
- データベースの実際の定義に関係なく、自由にテーブルを表示する。
- 固有のユーザまたは任意ユーザのカスタム特権に応じてテーブルの表示を変える。

警告：SSET FIELD TITLESコマンド**はテーブルの非表示属性を上書きしません。データベースの設計（デザイン）レベルでテーブルが非表示として設定されている場合、SET TABLE TITLES**コマンド**への呼び出しにそのテーブルを指定しても、テーブルは表示されません。**

▼ **SET FIELD TITLES**コマンドの例を参照してください。

参照：Count fields、**Field name**、SET TABLE TITLES

ユーザ&グループコマンド　59

この章では、「ルーチン」エディタの「User and Groups」テーマ内にあるユーザ&グループコマンドについて説明します。この章のコマンドは、パスワードやアクセス権を変更するために使用します。パスワードアクセスシステムに関する詳細は、『4th Dimension / 4D First デザインリファレンス』を参照してください。

CHANGE ACCESS
CHANGE PASSWORD
Current user
DELETE USER
EDIT ACCESS
GET GROUP LIST
GET GROUP PROPERTIES
GET USER LIST
GET USER PROPERTIES
Is user deleted
SET GROUP PROPERTIES
SET USER PROPERTIES
User in group
Validate password

EDIT ACCESS

EDIT ACCESS

EDIT ACCESSコマンドは、ユーザにパスワードシステムの編集環境を提供します。このコマンドは、「デザイン」モードの「パスワード」エディタを表示します。

グループは、デザイナ、管理者、グループオーナーによって編集されます。デザイナと管理者は、すべてのグループを編集することができます。グループオーナーは、所有するグループだけを編集することができます。また、グループオーナーはユーザをグループに割り当てたり、削除することができます。**EDIT ACCESS**コマンドは、グループが定義されていない場合には効果がありません。

デザイナと管理者は、ユーザをグループに割り当てたり、削除するだけでなく、新しいユーザを登録することもできます。

▼ 次の例は、ユーザに対して「パスワード」エディタを表示します：

EDIT ACCESS

CHANGE ACCESS

CHANGE ACCESS

CHANGE ACCESSコマンドは、データベースを終了しないでアクセスレベルを変更します。そして、ユーザがデータベース起動時に表示した同じ「パスワード」ダイアログボックスを表示します。ここで、ユーザは別のユーザ名として入力することができます。

「パスワード」ダイアログボックスの表示方法は、「データベースプロパティ」ダイアログボックス内の設定状況に依存します。

▼ 次の例は、ユーザに対して「パスワード」ダイアログボックスを表示します：

CHANGE ACCESS

CHANGE PASSWORD

CHANGE PASSWORD (パスワード)

引数	タイプ	説明
パスワード	文字列	新しいパスワード

CHANGE PASSWORDコマンドは、カレントユーザのパスワードを変更します。このコマンドは、カレントパスワードを新しいパスワード、＜パスワード＞に置き換えます。パスワードの区別はとても正確です。

▼ 次の**Current user**関数の例を参照してください。

Current user

Current user → 文字列

Current user関数は、カレントユーザのユーザ名を返します。

▼ 次の例は、ユーザがユーザ自身のパスワードを変更するためのプロシージャです。「デザイン」モードの「データベースプロパティ」ダイアログボックスで、「パスワード」ダイアログボックスの表示方法を設定することができます。また、 **CHANGE ACCESS**コマンドを使って、2種類ある「パスワード」ダイアログボックスのうちの1つを表示することができます：

■ ユーザは、「ユーザ名」リストからユーザ名を選択してパスワードを入力します。

■ ユーザは、ユーザ名とそれに対応するパスワードを入力します。

```
CHANGE ACCESS  ` 「パスワード」ダイアログボックスを表示
If (OK=1)
 $pw1:=Request ("新しいパスワード：" + Current user)
 ` パスワードは少なくとも 6文字以上にする
 If (((OK=1) & ($pw1 # "")) & (Length ($pw1) > 5))
 ` パスワードが正しく入力されたことを確認する
  $pw2:=Request ("もう1度、パスワード：")
  If ((OK=1) & ($pw1=$pw2))
   CHANGE PASSWORD ($pw2)          ` パスワードの変更
   CHANGE ACCESS
  End if
 End if
End if
```

User in group

User in group (ユーザ ; グループ) → ブール

引数	タイプ	説明
ユーザ	文字列	ユーザ名
グループ	文字列	グループ名

User in group関数は、＜ユーザ＞が＜グループ＞に属していれば “ True(真) ” を返します。

▼ 次の例は、特定の送り状を探しています。カレントユーザが “ Manager ” グループに属していれば、そのユーザは機密情報を表示するフォームをアクセスすることができます。そのユーザが “ Manager ” グループに属さなければ、別のフォームを表示します：

```
QUERY([送り状] ; [送り状]小売り > 10000)
If(User in Group (Current user ; "Manager"))
   OUTPUT FORM ([送り状] ; "機密出力")
   INPUT FORM ([送り状] ; "機密入力")
Else
   OUTPUT FORM ([送り状] ; "出力")
   INPUT FORM ([送り状] ; "入力")
End if
MODIFY SELECTION ([送り状] ; *)
```

DELETE USER

DELETE USER (ユーザID)

引数	タイプ		説明
ユーザID	数値	→	削除するユーザのID番号

DELETE USERコマンドは、引数＜ユーザID＞に渡したユーザID番号を持つユーザを削除します。この場合、**GET USER LIST**コマンドで返された有効なユーザID番号を渡さなければなりません。

ユーザアカウントが存在しなかったり、またはすでに削除されている場合は、エラーコード -9979が発生します。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

削除されたユーザ名は、データベースが開かれたり、**CHANGE ACCESS**コマンドを呼び出した際に表示される「パスワード」ウインドウにはもはや現れません。しかし、一意のユーザID番号を管理するために、そのユーザアカウントはパスワードシステム内には保持されています。削除されたユーザ名「デザイン」モードの「パスワード」ウインドウにおいて緑（グリーン）で表示されます。

参照：GET USER LIST、GET USER PROPERTIES、Is user deleted、SET USER PROPERTIES

エラー処理

DELETE USERコマンドを呼び出すための特定のアクセス権を持っていない場合やパスワードシステムが他のユーザによってアクセスされている場合は、アクセス権エラーが生成されます。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

Is user deleted

Is user deleted (ユーザID) → ブール

引数	タイプ		説明
ユーザID	数値	→	ユーザID番号
関数の返す値	ブール	←	True = ユーザアカウントが削除されたり、存在しない場合 False = ユーザアカウントがアクティブの場合

Is user deleted関数は、引数＜ユーザID＞に渡された一意のユーザID番号を持つユーザアカウントを調べます。

ユーザアカウントが存在しなかったり、すでに削除されてしまっている場合、**Is user deleted**関数はTrueを返します。ユーザアカウントがアクティブの場合は Falseを返します。

参照：DELETE USER、GET USER PROPERTIES、SET USER PROPERTIES

エラー処理

Is user deleted関数を呼び出すための特定のアクセス権を持っていない場合やパスワードシステムが他のユーザによってアクセスされている場合は、アクセス権エラーが生成されます。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

GET USER LIST

GET USER LIST (ユーザ名 ; ユーザID)

引数	タイプ		説明
ユーザ名	文字列配列	←	「パスワードエディタ」ウインドウ内に現れるユーザの名前
ユーザID	数値配列	←	対応する一意のユーザID番号

GET USER LISTコマンドは、引数＜ユーザ名＞と＜ユーザID＞の配列に「パスワード」ウインドウ内に現れるユーザの名前と一意のID番号割り当てます。

引数＜ユーザ名＞配列には、「パスワード」ウインドウ内に表示されるユーザ名が埋め込まれます。このウインドウには、使用できないユーザアカウントが緑（グリーン）で表示されるユーザ名も含まれます。

注：削除されたユーザを見つけるにはIs user deleted関数を使用します。

＜ユーザ名＞と同期される引数＜ユーザID＞配列は、対応している一意のユーザID番号で埋め込まれます。このID番号は、下記の値または範囲を持っています。

ユーザID番号	ユーザの種類
1	デザイナ
2	管理者
3~15000	デザイナによって作成されたユーザ (つまり、ユーザID番号3がデザイナによって最初に作成されたユーザ ユーザID番号4が2番目に作成されたユーザになります。)
-11~-15000	管理者によって作成されたユーザ (つまり、ユーザID番号-11が管理者によって最初に作成されたユーザ ユーザID番号-12が2番目に作成されたユーザになります。)

参照：GET GROUP LIST、GET USER PROPERTIES、SET USER PROPERTIES

エラー処理

GET USER LISTコマンドを呼び出すための特定のアクセス権を持っていない場合やパスワードシステムが他のユーザによってアクセスされている場合は、アクセス権エラーが生成されます。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

GET USER PROPERTIES

GET USER PROPERTIES (ユーザID ; ユーザ名 ; 起動メソッド ; パスワード ; ログイン回数 ; 最終日付 {; グループID})

引数	タイプ		説明
ユーザID	数値	→	一意のユーザID番号
ユーザ名	文字列	←	ユーザの名前
起動メソッド	文字列	←	スタートアップメソッドの名前
パスワード	文字列	←	常に空の文字列
ログイン回数	数値	←	データベースにログインした回数
最終日付	日付	←	データベースに最後にログインした日付
グループID	数値配列	←	データベースに最後にログインした日付

GET USER PROPERTIESコマンドは、引数＜ユーザID＞に渡された一意のユーザID番号を持つユーザに関する情報を返します。この場合、**GET USER LIST**コマンドで返された有効なユーザID番号を渡さなければなりません。

ユーザアカウントが存在しなかったり、またはすでに削除されている場合は、エラーコード -9979が発生します。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。それ以外は、**Is user deleted**関数を呼び出して、**GET USER PROPERTIES**コマンドを呼び出す前にユーザアカウントを調べることができます。

＜ユーザ＞と同期される引数＜ユーザID＞配列は、対応している一意のユーザID番号で埋め込まれます。このID番号は、下記の値または範囲の番号を持っています。

ユーザID番号	ユーザの種類
1	デザイナ
2	管理者
3~15000	デザイナによって作成されたユーザ (つまり、ユーザID番号3がデザイナによって最初に作成されたユーザ ユーザID番号4が2番目に作成されたユーザになります。)
-11~-15000	管理者によって作成されたユーザ (つまり、ユーザID番号-11が管理者によって最初に作成されたユーザ ユーザID番号-12が2番目に作成されたユーザになります。)

このコマンドを呼び出すと、引数＜ユーザ名＞、＜起動メソッド＞、＜パスワード＞、＜ログイン回数＞、＜最終日付＞にユーザの名前、Startupメソッド、空の文字列、およびデータベースにログインした回数および最後にログインした日付を返します。

オプション引数＜グループID＞を指定すると、ユーザが属すグループの一意のID番号を返します。グループID番号は、次の範囲の番号を持つことができます。

グループID番号	グループの種類
15000~32767	デザイナまたは関連したグループオーナーによって作成されたグループ(つまりグループID番号15001がデザイナによって最初に作成されたグループ、グループID番号15002が2番目に作成されたグループになります。)
-150011~-32768	管理者または関連したグループオーナーによって作成されたグループ(つまりグループID番号-15001が管理者によって最初に作成されたグループ、グループID番号15002が 2番目に作成されたグループになります。)

参照：GET GROUP LIST、GET USER LIST、SET USER PROPERTIES

エラー処理

GET USER PROPERTIESコマンドを呼び出すための特定のアクセス権を持っていない場合やパスワードシステムが他のユーザによってアクセスされている場合は、アクセス権エラーが生成されます。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

SET USER PROPERTIES

SET USER PROPERTIES (ユーザID ; ユーザ名 ; 起動メソッド ; パスワード ; ログイン回数 ; 最終日付 {; グループID})

引数	タイプ		説明
ユーザID	数値	→	ユーザアカウントの一意のID番号、または -1=デザイナに関連したユーザの追加 -2=管理者に関連したユーザの追加
		←	新しいユーザの一意のID番号
ユーザ名	文字列	→	ユーザの名前
起動メソッド	文字列	→	スタートアップメソッドの名前
パスワード	文字列	→	新しい（暗号化された）パスワード、または *を指定すると、パスワードは以前のまま
ログイン回数	数値	→	データベースにログインした回数
最終日付	日付	→	データベースに最後にログインした日付
グループID	数値配列	→	ユーザが属すグループのID番号

SET USER PROPERTIESコマンドは、引数＜ユーザID＞に渡された一意のユーザID番号を持つユーザアカウントのプロパティを変更および更新することができます。また、デザイナまたは管理者に関連したユーザを追加することができます。

既存のユーザアカウントのプロパティを修正している場合、**GET USER LIST**コマンドで返された有効なユーザID番号を渡さなければなりません。

ユーザアカウントが存在しなかったり、またはすでに削除されている場合は、エラーコード -9979が発生します。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。それ以外は、**Is user deleted**関数を呼び出して、**SET USER PROPERTIES**コマンドを呼び出す前にユーザアカウントを調べることができます。

ユーザID番号は、下記の値または範囲の番号を持っています。

ユーザID番号	ユーザの種類
1	デザイナ
2	管理者
3~15000	デザイナによって作成されたユーザ (つまり、ユーザID番号3がデザイナによって最初に作成されたユーザ ユーザID番号4が2番目に作成されたユーザになります。)
-11~-15000	管理者によって作成されたユーザ (つまり、ユーザID番号-11が管理者によって最初に作成されたユーザ ユーザID番号-12が2番目に作成されたユーザになります。)

デザイナに関連した新しいユーザを追加するには、引数＜ユーザID＞に -1を渡します。管理者に関連した新しいユーザを追加するには、引数＜ユーザID＞に -2を渡します。この両方のケースでは、新しいユーザを追加している際、4th Dimensionは利用可能であるけれども現在使用不可になっているユーザアカウントの1番目を再利用しようと試みます。使用不可になっているアカウントが利用できない場合にのみ新しいアカウントを作成します。このコマンドを呼び出すと、そのユーザが正常に追加された場合は、その追加されたユーザのID番号が＜ユーザID＞に返されます。

＜ユーザID＞に-1、-2または有効なユーザID番号が渡されない場合、**SET USER PROPERTIES**コマンドは何も行わない。

このコマンドを呼び出す前に、引数＜ユーザ名＞、＜起動メソッド＞、＜パスワード＞、＜ログイン回数＞、＜最終日付＞にユーザの名前、Startupメソッド、暗号化されたパスワード、およびデータベースにログインした回数および最後にログインした日付を設定します。引数＜パスワード＞に“*”を設定すると、このアカウントのパスワードを変更せずにこのユーザアカウントの他のプロパティを変更することができます。（グループは別として）そのユーザのすべてのプロパティを変更したくない場合は、まず**GET USER PROPERTIES**コマンドを呼び出して、それから変更したくないプロパティ用に返される値を渡します。

オプション引数＜グループID＞を指定しなかった場合、そのユーザのカレントグループは変更されません。ユーザを追加する際に＜グループID＞を指定しなかった場合は、そのユーザは任意のグループに属しません。

オプション引数＜グループID＞を指定すると、そのユーザのすべてのグループを変更します。このコマンドを呼び出す前に、引数＜グループID＞配列にそのユーザが属すグループの一意のID番号を割り当てなければなりません。グループID番号は、次の範囲の番号を持つことができます。

グループID番号	グループの種類
15000~32767	デザイナまたは関連したグループオーナーによって作成されたグループ(つまりグループID番号15001がデザイナによって最初に作成されたグループ、グループID番号15002が2番目に作成されたグループになります。)
-150011~-32768	管理者または関連したグループオーナーによって作成されたグループ(つまりグループID番号-15001が管理者によって最初に作成されたグループ、グループID番号15002が 2番目に作成されたグループになります。)

あるユーザのすべてのグループを無効にするには、＜グループID＞に空の文字列を渡します。

参照：GET GROUP LIST、GET USER LIST、GET USER PROPERTIES

エラー処理

SET USER PROPERTIESコマンドを呼び出すための特定のアクセス権を持っていない場合やパスワードシステムが他のユーザによってアクセスされている場合は、アクセス権エラーが生成されます。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

GET GROUP LIST

GET GROUP LIST (グループ名 ; グループID)

引数	タイプ		説明
グループ名	文字列配列	←	「パスワードエディタ」ウインドウ内に現れるグループの名前
グループID	数値配列	←	対応する一意のグループID番号

GET GROUP LISTコマンドは、引数＜グループ名＞と＜グループID＞の配列に「パスワード」ウインドウ内に現れるグループの名前と一意のID番号割り当てます。

引数＜グループ名＞配列には、「パスワード」ウインドウ内に表示されるすべてのグループ名が埋め込まれます。このウインドウには、使用できないグループが赤（レッド）で表示されるグループ名も含まれます。

注：データベースに入ることができないグループを見つけるには、GET GROUP PROPERTIESコマンドを使用します。

＜グループ名＞と同期される引数＜グループID＞配列は、対応している一意のグループID番号で埋め込まれます。グループID番号は、次の範囲の値を持つことができます。

グループID番号	グループの種類
15000~32767	デザイナまたは関連したグループオーナーによって作成されたグループ(つまりグループID番号15001がデザイナによって最初に作成されたグループ、グループID番号15002が2番目に作成されたグループになります。)
-150011~-32768	管理者または関連したグループオーナーによって作成されたグループ(つまりグループID番号-15001が管理者によって最初に作成されたグループ、グループID番号15002が 2番目に作成されたグループになります。)

参照：GET GROUP PROPERTIES、GET USER LIST、SET USER PROPERTIES

エラー処理

GET GROUP LISTコマンドを呼び出すための特定のアクセス権を持っていない場合やパスワードシステムが他のユーザによってアクセスされている場合は、アクセス権エラーが生成されます。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

GET GROUP PROPERTIES

GET GROUP PROPERTIES (グループID ; グループ名 ; オーナー {; メンバーID})

引数	タイプ		説明
グループID	数値	→	一意のグループID番号
グループ名	文字列	←	グループの名前
オーナー	文字列	←	グループオーナーのユーザID番号
メンバーID	数値配列	←	グループメンバーのID番号

GET GROUP PROPERTIESコマンドは、引数＜グループID＞に渡された一意のグループID番号を持つグループのプロパティを返します。この場合、**GET USER LIST**コマンドで返された有効なグループID番号を渡さなければなりません。グループID番号は、次の範囲の番号を持つことができます。

グループID番号	グループの種類
15000~32767	デザイナまたは関連したグループオーナーによって作成されたグループ (つまりグループID番号15001がデザイナによって最初に作成されたグループ、グループID番号15002が2番目に作成されたグループになります。)
-150011~-32768	管理者または関連したグループオーナーによって作成されたグループ (つまりグループID番号-15001が管理者によって最初に作成されたグループ、グループID番号15002が 2番目に作成されたグループになります。)

有効なグループID番号を指定しなかった場合、**GET GROUP PROPERTIES**コマンドは空の文字列を返します。

このコマンドを呼び出すと、引数＜グループ名＞と＜オーナー＞にそのグループの名前とオーナー名を返します。

オプション引数＜メンバーID＞を指定すると、グループに属すユーザとグループの一意のID番号が返されます。メンバーID番号は、次の範囲の値を持つことができます。

メンバーID番号	メンバーの種類
1	デザイナ
2	管理者
3~15000	デザイナによって作成されたユーザ (つまり、ユーザID番号3がデザイナによって最初に作成されたユーザ ユーザID番号4が2番目に作成されたユーザになります。)
-11~-15000	管理者によって作成されたユーザ (つまり、ユーザID番号-11が管理者によって最初に作成されたユーザ ユーザID番号-12が2番目に作成されたユーザになります。)

15000~32767	デザイナまたは関連したグループオーナーによって作成されたグループ(つまりグループID番号15001がデザイナによって最初に作成されたグループ、グループID番号15002が2番目に作成されたグループになります。)
-150011~-32768	管理者または関連したグループオーナーによって作成されたグループ(つまりグループID番号-15001が管理者によって最初に作成されたグループ、グループID番号15002が 2番目に作成されたグループになります。)

参照：GET GROUP LIST、GET USER LIST、SET GROUP PROPERTIES

エラー処理

GET GROUP PROPERTIESコマンドを呼び出すための特定のアクセス権を持っていない場合やパスワードシステムが他のユーザによってアクセスされている場合は、アクセス権エラーが生成されます。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

SET GROUP PROPERTIES

SET GROUP PROPERTIES (グループID ; グループ名 ; オーナー {; メンバーID})

引数	タイプ		説明
グループID	数値	→	グループの一意のID番号、または -1=デザイナに関連したグループの追加 -2=管理者に関連したグループの追加
		←	新しいグループの一意のID番号
グループ名	文字列	→	グループの名前
オーナー	文字列	→	グループオーナーのユーザID番号
メンバーID	数値配列	→	新しいグループメンバーのID番号

SET GROUP PROPERTIESコマンドは、引数＜グループID＞に渡された一意のグループID番号を持つ既存グループのプロパティを変更および更新することができます。また、デザイナまたは管理者に関連したグループを追加することができます。

既存グループのプロパティを修正している場合、**GET USER LIST**コマンドで返された有効なグループID番号を渡さなければなりません。グループID番号は、次の範囲の番号を持つことができます。

グループID番号	グループの種類
15000~32767	デザイナまたは関連したグループオーナーによって作成されたグループ (つまりグループID番号15001がデザイナによって最初に作成されたグループ、グループID番号15002が2番目に作成されたグループになります。)
-150011~-32768	管理者または関連したグループオーナーによって作成されたグループ (つまりグループID番号-15001が管理者によって最初に作成されたグループ、グループID番号15002が 2番目に作成されたグループになります。)

デザイナに関連した新しいユーザを追加するには、引数＜グループID＞に -1を渡します。管理者に関連した新しいユーザを追加するには、引数＜グループID＞に -2を渡します。この両方のケースでは、新しいユーザを追加している際、4th Dimensionは利用可能であるけれども現在使用不可になっているグループの1番目を再利用しようと試みます。使用不可になっているグループが利用できない場合にのみ新しいグループを作成します。このコマンドを呼び出すと、そのグループが正常に追加された場合は、その追加されたグループのID番号が＜グループID＞に返されます。

＜グループID＞に-1、-2または有効なグループID番号が渡されない場合、**SET GROUP PROPERTIES**コマンドは何も行わない。

このコマンドを呼び出す前に、引数＜グループ名＞と＜オーナー＞にグループの名前とオーナーを設定します。（下記のメンバーは別として）そのグループのすべてのプロパテ

ィを変更したくない場合は、まず**GET GROUP PROPERTIES**コマンドを呼び出して、それから変更したくないプロパティ用に返される値を渡します。

オプション引数＜メンバーID＞を指定しなかった場合、そのグループのカレントメンバーリストは変更されません。グループを追加する際に＜メンバー＞を指定しなかった場合は、そのグループはメンバーを持ちません。

注：グループオーナーは、あるユーザが所有するグループのメンバーとして自動的に設定されるわけではありません。引数＜メンバー＞を使って、そのグループのグループオーナーを設定するのはあなたの義務です。

オプション引数＜メンバーID＞を指定すると、そのグループのメンバーリスト全体を変更します。このコマンドを呼び出す前に、引数＜メンバーID＞配列にそのグループがメンバーとして取得するユーザとグループの一意のID番号を割り当てなければなりません。メンバーID番号は、次の範囲の値を持つことができます。

メンバーID番号	**メンバーの種類**
1	デザイナ
2	管理者
3~15000	デザイナによって作成されたユーザ (つまり、ユーザID番号3がデザイナによって最初に作成されたユーザ ユーザID番号4が2番目に作成されたユーザになります。)
-11~-15000	管理者によって作成されたユーザ (つまり、ユーザID番号-11が管理者によって最初に作成されたユーザ ユーザID番号-12が2番目に作成されたユーザになります。)
15000~32767	デザイナまたは関連したグループオーナーによって作成されたグループ (つまりグループID番号15001がデザイナによって最初に作成されたグループ、グループID番号15002が2番目に作成されたグループになります。)
-150011~-32768	管理者または関連したグループオーナーによって作成されたグループ (つまりグループID番号-15001が管理者によって最初に作成されたグループ、グループID番号15002が 2番目に作成されたグループになります。)

あるグループのすべてのメンバーを無効にするには、＜メンバーID＞に空の文字列を渡します。

参照：GET GROUP LIST、GET USER LIST、SET GROUP PROPERTIES

エラー処理

SET GROUP PROPERTIESコマンドを呼び出すための特定のアクセス権を持っていない場合やパスワードシステムが他のユーザによってアクセスされている場合は、アクセス権エラーが生成されます。**ON ERR CALL**コマンドを使ってインストールされたエラー処理メソッドでこのエラーを受け取ることができます。

Validate password

Validate password (ユーザID ; パスワード) → ブール

引数	タイプ		説明
ユーザID	数値	→	ユーザID番号
パスワード	文字列	→	暗号化されていないパスワード
関数の返す値	ブール	←	True = ユーザアカウント用のパスワードの場合 False = ユーザアカウントのパスワードでない場合

Validate password関数は、引数＜パスワード＞に渡した文字列が引数＜ユーザID＞に渡されたID番号を持つユーザアカウント用のパスワードの場合、True（真）を返します。

例：

次の例は、ユーザ“山田”のパスワードが“Laurel”であるかどうかを調べます。

```
GET USER LIST(atUserName ; alUserID)
$vlElem:=Find in array(atUserName ; "山田")
If ($vlElem>0)
   If (Validate password(alUserID{$vlElem} ; "Laurel")>0)
      ALERT("正解です。")
   Else
      ALERT("間違っています！")
   End if
Else
   ALERT("ユーザ名が見つかりません。")
End if
```

参照：GET USER PROPERTIES、SET USER PROPERTIES

この章では、「ルーチン」エディタの「Variables」テーマ内にある変数コマンドについて説明します。この章では、変数を操作するためのコマンドについて説明します。

SAVE VARIABLES
CLEAR VARIABLE
LOAD VARIABLES
Undefined

SAVE VARIABLES

SAVE VARIABLES (文書 ; 変数1{ ;...; 変数N})

引数	タイプ	説明
文書	文字列	変数を保存する文書ファイル
変数	変数	保存する変数

SAVE VARIABLESコマンドは、ディスク上の＜文書＞ファイルに＜変数＞を保存します。

変数は同じタイプ(テキスト、数値、日付、時間、ブール、またはピクチャ)である必要はありません。**SAVE VARIABLES**コマンドは、配列を保存しません。

文書ファイル名は、変数と同じ名前である必要はありません。＜文書＞に空の文字列("")を指定した場合には標準の「ファイル作成」ダイアログボックスを表示します。ここで、ユーザは作成する文書ファイルの名前を入力することができます。

変数を正常に保存した場合は、システム変数OKに1が代入され、システム変数Documentに文書名が入ります。それ以外の場合には、システム変数OKに 0が代入され、システム変数Documentは空の文字列になります。

SAVE VARIABLESコマンドで変数を文書ファイルに保存する場合は、4D専用の内部フォーマットで保存します。従って、保存した変数は**LOAD VARIABLES**コマンド以外では読み込むことができません。

▼ 次の例は、文書ファイル“ 保存 ”に3つの変数を保存します：

```
SAVE VARIABLES ("保存" ; v文字列 ; v数値 ; vピクチャ)
```

LOAD VARIABLES

LOAD VARIABLES (文書 ; 変数1{ ;...; 変数N})

引数	タイプ	説明
文書	文字列	変数を保存した文書ファイル
変数	変数	読み込む変数

LOAD VARIABLESコマンドは、**SAVE VARIABLES**コマンドで作成したディスク上の＜文書＞ファイルから＜変数＞を読み込みます。

読み込む変数が存在しない場合には作成し、存在する場合には上書きします。

文書ファイル名は、変数と同じ名前である必要はありません。＜文書＞に空の文字列("")を指定した場合には、標準の「ファイルを開く」ダイアログボックスを表示します。ここで、ユーザは読み込む文書ファイルを選択することができます。

変数を正常に読み込んだ場合は、システム変数OKに1が代入され、システム変数Documentに文書名が入ります。それ以外の場合には、システム変数OKに 0が代入され、システム変数Documentは空の文字列を含みます。

コンパイルされたデータベースでは、変数はディスクからロードされた時と同じ型でなくてはなりません。

▼ 次の例は、文書ファイル“ 保存 ”から3つの変数を読み込みます：

```
LOAD VARIABLES ("保存" ; v文字列 ; v数値 ; vピクチャ)
```

CLEAR VARIABLE

CLEAR VARIABLE (変数)

引数	タイプ	説明
変数	変数	消去する変数

CLEAR VARIABLEコマンドは、メモリから＜変数＞を消去します。**CLEAR VARIABLE**コマンドは、変数を未定義に設定します。

CLEAR VARIABLEコマンドは、主にピクチャなどの大きな変数をメモリから消去するために使用します。

ローカル変数、つまり名前の先頭にドル記号($)のついた変数は、**CLEAR VARIABLE**コマンドでは消去できません。ローカル変数は、その変数の属するメソッドの実行が終了した時点で、自動的に消去されます。

▼ 次の例は、変数 “ v私の変数 ” を消去します：

```
CLEAR VARIABLE (v私の変数)                `私の変数を消去
```

Undefined

Undefined (変数) → ブール

引数	タイプ	説明
変数	変数	判定する変数

Undefined関数は、＜変数＞が定義されていない場合は “ True(真) ” を返し、＜変数＞が定義されている場合には “ False(偽) ” を返します。変数が定義されるのは、変数に値を代入した時点です。まだ値を代入していない変数や**CLEAR VARIABLE**コマンドで消去した変数は、未定義になります。

4D Compilerを使用してデータベースをコンパイルする場合は、**Undefined**関数はすべての変数に対して “ True(真) ” を返します。

▼ 次の例は、変数 “ 実行 ” が未定義かどうかを判定し、未定義の場合には定義します：

```
If (Undefined (実行))
  実行:=""
End if
```

Webサービス：概要

4th Dimensionと4D Serverには、両方ともWeb上に4Dデータベースを公開できるWebサーバエンジンがあります。4D Webサーバエンジンには、次のような非常に独創的な機能があります。

■ ダイレクトWebサービス

データベースは、直接Web上に公開されます。ユーザは、データベースシステム、Webサイト、またはその間のCGIインタフェースを開発する必要はありません。データベースがWebサイトになります。

■ オンラインの透過的なHTML変換

4Dはオンラインで透過的かつ動的にフォームと設計コンポーネントをHTMLページに変換します。変換された新しいHTMLページは、データベースにすでに接続されている場合でもその場でWebブラウザから利用可能になります。現在では、ほとんどのWebデータベースシステムは、CGIベースのシステムまたは静的なHTMLベースのシステムです。

CGIシステムでは、データベース、Webサイト、CGIを開発する必要があります。静的HTMLベースのシステムでは、データベースの設計コンポーネントを変更するたびにその変更をHTMLに再変換するユーティリティプログラムを実行する必要があります。どちらの場合でも、Webコンポーネントはオフラインで作成され、データベースやWeb開発者からの手動での介入が必要です。これに対して、4th Dimensionでは、必要なだけ必要な時に設計コンポーネントを変更することができます。そして、「デザイン」モードで変更を保存すると、変更は透過的にWebブラウザですぐに利用可能になります。このため、アプリケーションの開発とテストを行っているときに、すでに接続されているWebブラウザでその結果をすぐにテストすることができます。

■ レコードおよびデータへの動的アクセス

4Dは、4Dデータベースエンジンの標準クライアントとしてWebブラウザを扱います。例えば、ローカルな4th Dimensionデータベース上または4D Serverのクライアントワークステーションからレコードを変更する場合には、それらのレコードはWebブラウザで即座に利用可能になります。他のシステムのように、HTMLの発行のためにレコードを再処理する必要はありません。

■ セッション保守とデータベースコンテキスト

Webブラウザは、その名前が示すとおり、ランダムにWebページをブラウズすることができます。ユーザは、あるページから別のページに、1つのWebサイトから別のサイトへとジャンプできます。クライアント／サーバデータベース関連でWebブラウザを使用するときには、データベーストランザクションのロジックにブラウザを合わせる必要があります。例えば、レコードを追加しているのであれば、レコードデータ入力における妥当性検査および取り消し処理の規則に従う必要があります。ユーザはブラウザナビゲーション制御経由では、レコードを終了させて、レコードを不確かな状態のままにすることはできません。4D Webサーバエンジンには、内蔵のセッション保守とデータベースコンテキスト保守が含まれています。WebページのURL全体には、独自のコンテキストID番号とサブコンテキストID番号が保守されます。これらの番号でブラウザで現在表示されているWebページと4D側のデータベース接続のコンテキストとの完全な同期が保証されます。

■ 透過的なマルチユーザ保守

Webブラウザは、4Dクライアントになると完全なデータベースクライアントとして扱われます。例えば、Webブラウザでレコードの変更を開始すると、4Dはレコードを自動的にロックして他のクライアントがレコードを同時に変更しないようにします。ユーザがデータ入力の妥当性検査やキャンセルを行った後に、4Dはレコードのロックを自動的に解除します。さらに、4th Dimensionまたは4D Clientを使用して実行する権限を与えられるように、4Dではトランザクションでデータ入力を実行することができます。

■ Webプロセスの保守

4Dには、4DのWebクライアント／サーバアーキテクチャを処理する複数のプロセスがあります。内蔵のメインWebサーバプロセスは、Webへの接続処理を扱います。Webブラウザがデータベースへのアクセス権を付与されると、Web接続は接続目的のために自動的に作成された独立プロセスへの実行を開始します。完全に統合された4Dのマルチタスクアーキテクチャの結果として、通常のクライアントやWeb 4Dクライアントは、データベースエンジンで平行して処理されるデータベースリクエスト(照会など)を同時に実行することができます。また、クライアントも、特定のWebクライアントから発信されたリクエストは他のプロセスのコンテキストに介入しないことが保証されています。

■ 最適化されたWeb Serverアーキテクチャ

4D Webサーバエンジンには、4Dデータベースエンジンと同じ機能があります。例えば、配列に一連のレコード値をロードする場合に、そのロード処理はWebサーバマシンでローカルに実行されます。その後、リクエストの結果が全体としてリクエストを出したWebクライアントに送信されます。

さらに、Web接続は、Webサーバ側の単純かつ完全な機能を備えた4Dプロセスなので、ユーザは自分の好きな4Dアルゴリズムを実行できます。そのアルゴリズムはWebサーバ側でローカルに実行され、リクエストがある場合には、そのリクエストだけがWebブラウザに送られます。例えば、リレート設定、セット、合計結果の計算を含むクエリ（照会）

を実行すると、結果だけがWebブラウザに返ります。4Dアプリケーションはコンパイルできるので、Webのエキスパートにならなくても複雑なWebデータベースシステムを構築することができます。

■ HTMLとJavaScriptのカプセル化

4DはWeb上にデータベースを公開するために必要なほとんどすべてのことを実行するのですが、HTMLとJavaScriptのコードのカプセル化によって4D開発をカスタマイズすることもできます。例えば、自分のデータベース／Webサイトを目立つHTMLページにしてホームページをインパクトのあるものにすることもできます。

ユーザはカスタムHTMLページ全体を構築してそれを**SEND HTML FILE**コマンドでWebに入れることができます。また、Webブラウザ上での表示がフォームにある4DとHTMLオブジェクトが混在した組み合せになる4DフォームにHTMLをカプセル化することもできます。カプセル化されたHTMLの内部では、リクエストをサーバに返すことなくクライアントのWebブラウザ側でアクションやデータ制御を実行するJavaScriptコードを実装できます。

■ HTMLと4Dオブジェクトでのバインド

4D開発環境内部にカプセル化したHTMLコードを使用する場合には、HTMLオブジェクトに入力された値とデータを4D側で検索できることが必要になります。複雑なHTML解析ルーチンをユーザが作成するのではなく、4Dは、HTMLオブジェクトが同じ名前を持つだけでいいという、HTMLオブジェクトを4D変数にバインドするための簡単な組み込みシステムを提供します。結果として、HTMLリクエストの分析や応答を実装するのが非常に簡単になります。つまり、Webブラウザから戻ってきたデータが自動的に入る4D変数を扱う4Dコードを作成する必要があるだけです。

次は何をするのか？

■ Web公開用にマシンとデータベースを設定するには、「Webサービス：システム設定」の節を参照してください。

■ Web上にデータベースを公開する方法を習得するには、「Web Services：入門編(パートI)」と「Webサービス：入門編(パートII)」の節を参照してください。

■ HTMLカプセル化の詳細については、「Web サービス：HTMLとJavaScriptのカプセル化」の節を参照してください。

■ Webとプロセスのやり取りの詳細については、「Webサービス：Web 接続プロセス」の節を参照してください。

参照：SEND HTML FILE、SET HTML ROOT、SET HTML ROOT、SET WEB DISPLAY LIMIT、SET WEB TIMEOUT、STOP WEB SERVER

Webサービス：システム設定

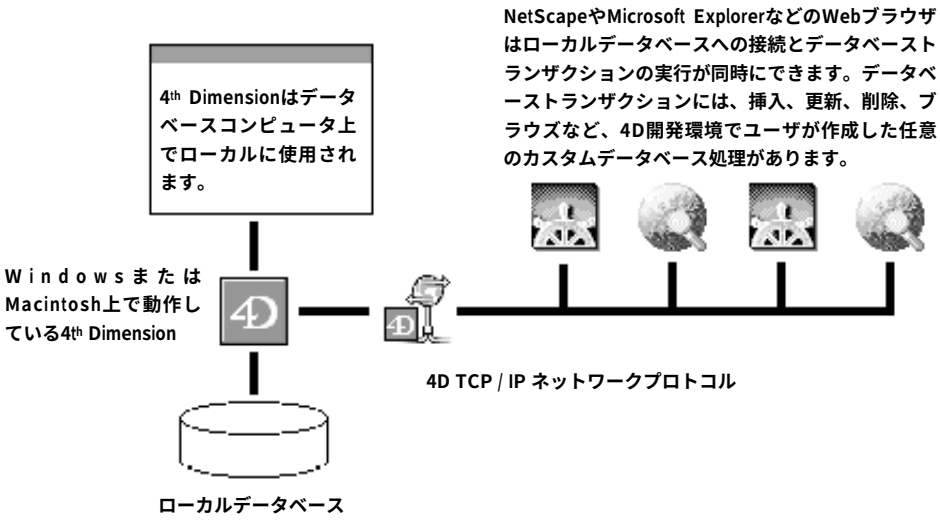
4th Dimension 6.0と4D Server 6.0には、Web上でデータベースに透過的かつ動的にサービスを行うWebサービス機能があります。

4th DimensionとWeb

4th Dimensionを使用してWeb上に4Dデータベースを公開する場合、同時に次のことができます。

- 4Dでローカルにデータベースを使用する
- Webブラウザを使用してデータベースに接続する

これを以下の図にまとめます。

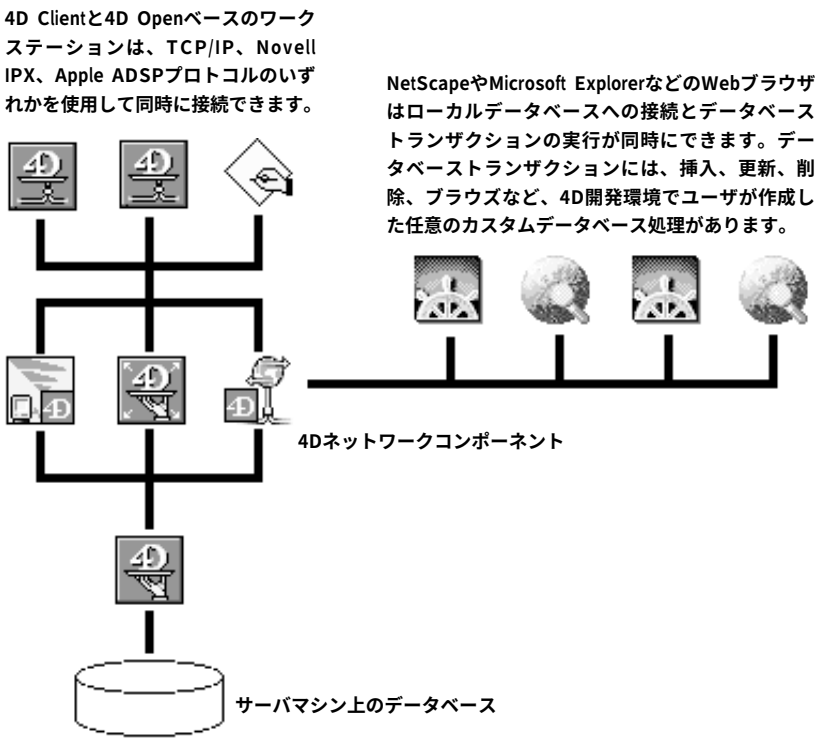


4D ServerとWeb

4D Serverを使用してWeb上に4Dデータベースを発行する場合には、以下を使用して4Dデータベースへの接続とその処理を同時に行えます。

- 4D Clientワークステーション
- 4D Openをベースにしたアプリケーション
- Webブラウザ

これを以下の図にまとめます。



Web上での4Dデータベースのサービス

4th Dimensionや4D Serverを使用してWeb上に4Dデータベースを提供するには、以下に示す適切な接続ライセンスとネットワークコンポーネントが必要です。

- 必要なWeb接続ライセンスがアプリケーションにインストールされている必要があります。詳細については『4D インストールガイド』を参照してください。

■ Web接続は、TCP/IPプロトコルを使用してネットワーク上に構築されます。
従って、

■ マシンにTCP/IPをインストールしておき、正しく設定しておく必要があります。詳細についてはコンピュータまたはオペレーティングシステムのマニュアルを参照してください。

■ 4D TCP/IP Networkコンポーネントをインストールしておく必要があります。Macintoshでは、このネットワークコンポーネントを4th Dimension、4D Server、または4Dマージ済カスタムアプリケーションに直接インストールする必要があります。Windowsでは、このネットワークコンポーネントをアクティブなWINDOWSディレクトリのACI\NETWORKディレクトリにインストールする必要があります。

注：どちらの場合も、詳細についてはネットワークコンポーネントのマニュアルを参照してください。

■ Web接続ライセンスとTCP/IPのインストールまたはチェックを行った後は、4D内部からWebサービスを起動する必要があります。これについては、次の節で説明します。

TCP/IPがまだ稼働していないとき、または4D TCP/IPネットワークコンポーネントがまだ存在していないときに4D Webサービスを起動しようとすると、4th Dimensionは次の警告を表示します。

このメッセージが表示された場合には、説明されているとおりにインストレーションを実行するか、TCP/IP設定のトラブルシューティングを行ってください。

4D Webサービスの開始

4D Webサービスは、次の3つの異なる方法で起動することができます。

■ 4D Serverまたは4th Dimension「ユーザ」モードのメインメニューバーから「Web サーバ」メニューを使用する方法。図のように、「Web サーバ」メニューでは都合のよいときにWeb サービスの開始と終了ができます。

■ データベースがオープンされるたびに自動的にデータベースを公開する方法。Web上にデータベースを自動的に公開するには、4D Serverまたは4th Dimensionの「デザイン」モードの「ファイル」メニューから「データベースプロパティ...」メニューコマンドを選択します。次のような「データベースプロパティ...」ダイアログボックスが表示されます。

「Web サーバ起動時オプション」エリアで、「起動時にデータベースを公開する」チェックボックスを選択してから「OK」ボタンをクリックします。これを一度行えば、ユーザが4th Dimensionや4D Serverでオープンするたびにデータベースはweb上に自動的に公開されます。

■ プログラムで**START WEB SERVER**コマンドを呼び出す方法。

Tips：Web上にデータベースの公開を開始または終了する際に、4Dを停止させて、データベースを再オープンする必要はありません。Webサービスの中断と再起動は「Web サーバ」メニューを使用したり、START WEB SERVERコマンドとSTOP WEB SERVERコマンドを呼び出すことで何回でも必要なだけ行えます。

Web上に公開された4Dデータベースへの接続

Webへの4Dデータベースの公開を開始したら、ユーザはWebブラウザを使用してそのデータベースに接続することができます。これを行うには、次のように行います：

■ Webサイトに登録名(例："Flowers International")がある場合には、その名前をブラウザのOpen、Address、Locationエリアのいずれかに指定します。その後、Enterキーを押して接続します。
■ Webサイトに登録名がない場合には、マシンのIPアドレス(例：123.4.567.89)をブラウザのOpen、Address、Locationエリアのいずれかに指定します。その後、Enterキーを押して接続します。

この時に、すべてがうまく行っていれば接続できます。接続できない場合には、次の状況のいずれかになっているはずです。

1. "...the server may not be accepting connections or may be busy..."(サーバは接続を受け付けないか、またはビジー状態です)といったメッセージを受け取った場合。

この場合、以下を確認します。

■ 入力した名前またはIPアドレスが正確なことを確認します。

■ 4th Dimensionまたは4D Serverが立ち上がって稼働しており、そのWeb サービスを開始していることを確認します。

■ デフォルトのWeb TCPポート以外のTCPポートでサービスを受けるようにデータベースが設定されているかどうかを確認します(下記の3を参照)。

■ TCP/IPがサーバマシンとブラウザマシンの両方で正しく設定されているかどうかを確認します。両方のマシンは同一のネットとサブネット上にあるか、あるいはルータが正しく設定されている必要があります。

■ ハードウェアが正しく接続されていることを確認します。

■ サイトのテストをローカルでしているのではなく、インターネットやイントラネット上で他者がサービスを提供しているWebデータベースに接続しようとしているのであれば、表示されたメッセージが正しいこともあります。つまり、サーバがオフになっているかビジー状態であるということです。この場合には、ログオンできるまで少し待って再試行するか、Webのプロバイダに連絡してください。

2. 接続はできたが、Webページに"This database has not been setup for the Web yet"(このデータベースはWeb用に設定されていません)というメッセージが表示される場合。

これは、4D Webサービスが立ち上がって稼働しており、データベースに正しく接続できたことを示しています。ただし、データベースがWeb経由で操作可能な状態になるために、いくつかの最低限のコンポーネントを含む必要があります。詳細については、「Webサービス：入門編(パートI)」の節を参照してください。

注：また、このメッセージは、現時点でデータベースが利用可能なWebライセンスを備えていないということも意味します。

(3) 接続はしたが、予想してたようなWebページを取得できない場合。

これは、1つのマシン上で同時に複数のWebサーバが稼働しているときに発生する可能性があります。次の例を参照してください。

■ すでに独自のWebサービスを稼働しているWindows NT 4.0システム上で1つだけの4D Webデータベースを稼働している場合。

■ 1つのマシン上で複数の4D Webデータベースを稼働している場合。

このような状況のもとでは、4D Webデータベースが発行されるTCPポート番号を変更する必要があります。これを行うには、次の節を参照してください。

TCPポート番号を特定の値に設定する

デフォルトでは、4Dは通常のWeb TCPポートにWebデータベースを公開します。このポートはポート80です。ポート80が他のWebサービスですでに使用されている場合には、このデータベース用に4Dが使用するTCPポートを変更する必要があります。変更するには、4D Serverまたは4th Dimensionの「デザイン」モードの「ファイル」メニューから「データベースプロパティ...」メニューコマンドを選択します(前述の図を参照)。「TCPポート入力」エリアに移動し、適切な値(同一のマシン上で稼働している別のTCP/IPサービスで使用されていないTCPポート番号)を指定します。

注：0を指定すると、4Dはデフォルトの TCPポート番号 80を使用します。

Webブラウザでは、Webデータベースへの接続用に入力するアドレスにデフォルト以外のTCPポート番号を指定する必要があります。そのアドレスにはコロンとポート番号で構成される接尾辞を指定します。例えば、TCPポート番号700を使用している場合には、「123.4.567.89:700」と指定します。

警告：デフォルトの80以外のTCPポート番号を使用する場合には、同時に使用する予定の他のサービス用のデフォルトのポート番号は使用しないように注意してください。例えば、WebサーバマシンでFTPプロトコルも使用する予定の場合には(自分がしていることの影響がわからない限り)、TCPポートの20と21は使用しないでください。これらのポートはFTPプロトコルのデフォルトポートです。デフォルトのTCPポート番号とプロトコルの詳細については、TCP/IPプロトコルの解説書でRFC 1700標準の割り当て番号表を参照してください。256未満のポート番号は、既知のサービス用に予約されており、256から1024までのポート番号はUNIXプラットフォームから提供される特定のサービス用に予約されています。何千番目かのポート番号を使用すると他に影響しないでしょう。

参照：SEND HTML FILE、SET HTML ROOT、SET WEB DISPLAY LIMIT、SET WEB TIMEOUT、STOP WEB SERVER

Webサービス：入門編(パートI)

例：

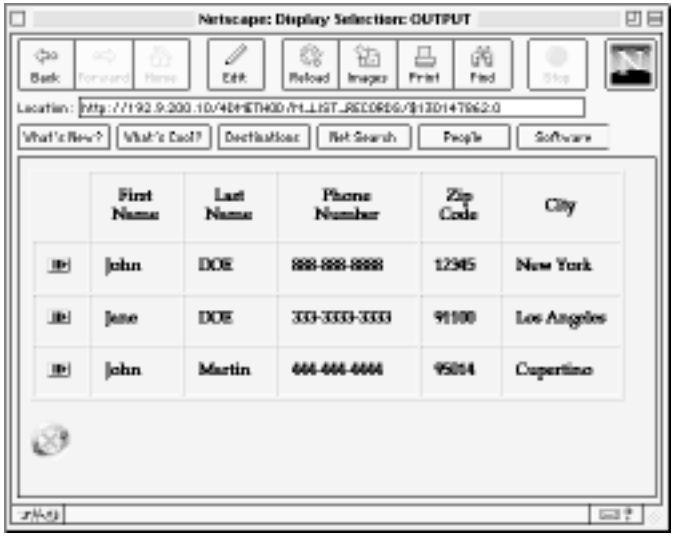
4th Dimensionで提供される例の中に「WebDemo1」というデータベースがあります。WindowsやMacintoshでこのデータベースをオープンすると、このデータベースは自身をWebサーバとしてネットワーク上に自動的に公開します。データベースのオープン中にエラーメッセージが表示された場合には、トラブルシューティングのために「Webサービス：システム設定」の節を参照してください。

1. Webサーバデータベースへの接続

今起動したばかりのWebサーバに接続するには、2番目のマシン上で稼働しているWebブラウザ上でデータベースをオープンします。次の画面と似たWebページが表示されるはずです。この画面はMacintosh上で稼働するNetscapeのものです。

2. レコードの表示とブラウズ

リンクされている「List Existing Records」テキストをクリックします。これによって、4Dの**DISPLAY SELECTION**コマンドで表示した画面と同等のWebの画面が次のように表示されます。



ここでは、用途に応じてレコードをブラウズできます。「終了」ボタン(画面左下のXが付いているボタン)をクリックすると、Webサイトのホームページに戻ります。

3. レコードの追加

Webサイトのホームページで、リンクされた「Add Some Records」テキストをクリックすると4Dの**ADD RECORD**コマンドで表示した画面と同等のWebの画面が次のように表示されます。

ユーザは必要なだけの数のレコードを追加できます。追加が終ったら、「キャンセル」ボタン(Xが付いているボタン)をクリックして、Webサイトのホームページに戻ります。

4. メインメニューでのレコードのリストまたは追加

ホームページで、「Go to Main Menu Bar」ボタンをクリックします。これでホームページを終了し、4D の「カスタム」メニューバーと同等のWebの画面が次のように表示されます。

ここで、各メニュー項目をクリックすると、レコードの追加と一覧表示を行うことができます。ホームページから使用したのと同じ4Dメソッドがメニュー項目に対応しています。

5. 接続の終了

作業を終了したら、ブラウザを終了してください。これで、タイムアウト遅延時間が経過すると、4Dは「Web接続プロセス」を終了します。

Web接続の初期化

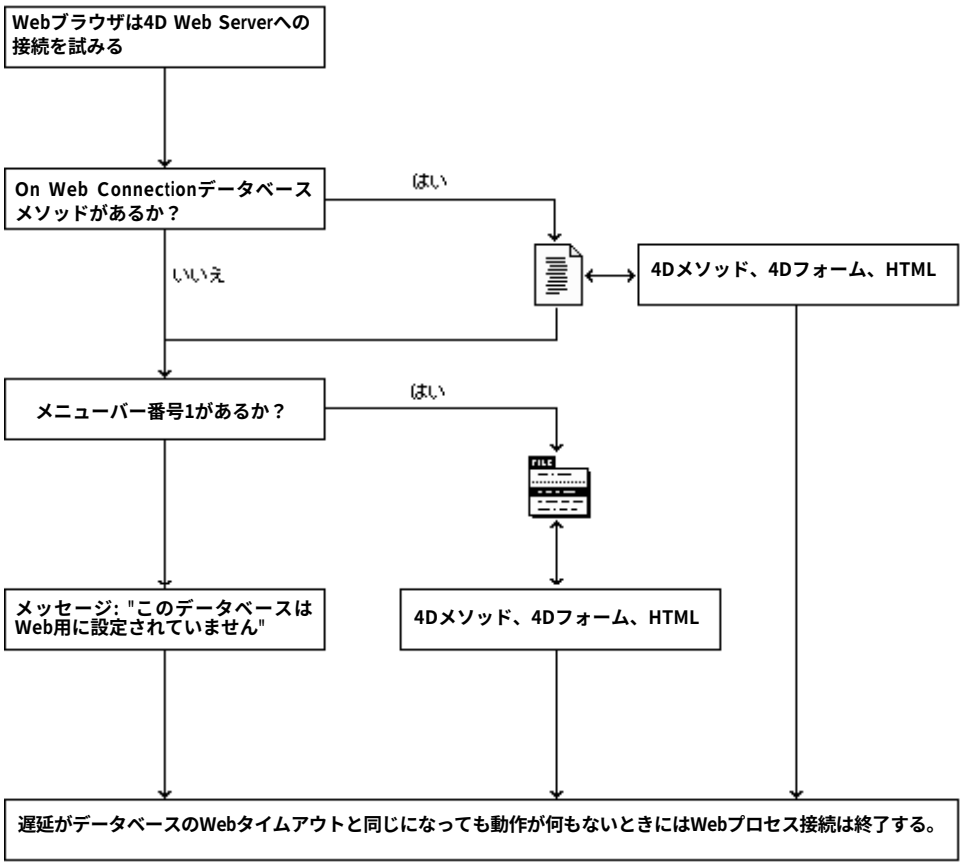
Webブラウザが Webサーバとして公開された4Dデータベースに接続するときには、いつでも4Dは次のアクションを実行します。

■ 存在する場合には「On Web Connection」データベースメソッドを実行します。

■ そのようなデータベースメソッドが存在しない、あるいはメソッドがすでに目的を達成している場合、4Dはメニューバー番号1が存在する場合にはそれを表示します。

■ メニューバーが存在しない場合には、4th Dimensionは以下を示すデフォルトのWebページを表示します。
"This database has not been setup for the Web yet"(このデータベースはWeb用に設定されていません)

これらのアクションを次の図にまとめます。



「On Web Connection」データベースメソッドは、HTMLページと同様、データベース内に定義された任意のプロジェクトメソッドやフォームを呼び出すことができます。このデータベースメソッドは、実際にセッション全体を処理することができます。

4Dや4D ServerへのWeb接続は、クライアント／サーバ接続と同じではありません。HTMLとWebをサポートするHTTPプロトコルは、「セッションをベースにした」プロトコルではありません。HTTPプロトコルはむしろ「リクエストをベースにした」プロトコルです。クライアント／サーバでは、ユーザはサーバに接続し、セッションで作業を行ってからサーバからの接続を解除します。HTTPでは、Webサーバの割り込み（アテンション）またはWeb サーバからのアクションを必要とするアクションをユーザが実行するたびに、リクエストはサーバに送信されます。つまり、HTTPリクエストは「接続+リクエスト+応答の待機+接続解除」のシーケンスとして理解されます。

HTTP上でクライアント／サーバセッションを実行するために、4Dは、URLの透過的な符号化（エンコード）によって、Web接続を一意に特定するコンテキストを保守し、それと同時に接続を処理する4Dプロセスに接続を対応させます。

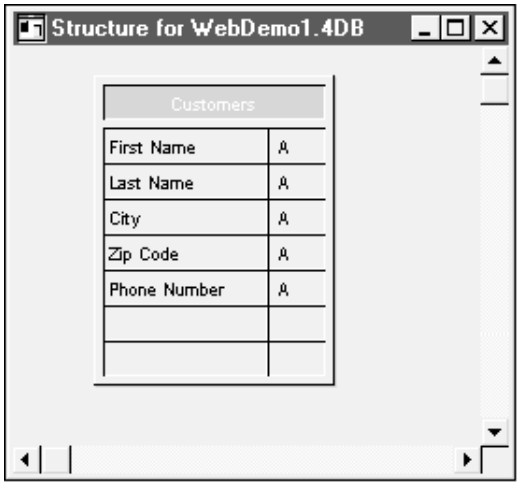
ただし、4Dにはクライアント／サーバの接続解除アクションと同等のセッションを終了させるものを提供する方法がありません。このため、クライアント／サーバセッションの終了はタイムアウト方式によって処理されます。Web接続を処理する4Dプロセスは、データベースのWebタイムアウト設定と同じだけ遅延時間が経過しても何の動作も検出されないと終了します。

1つになったデータベースとWebサーバ

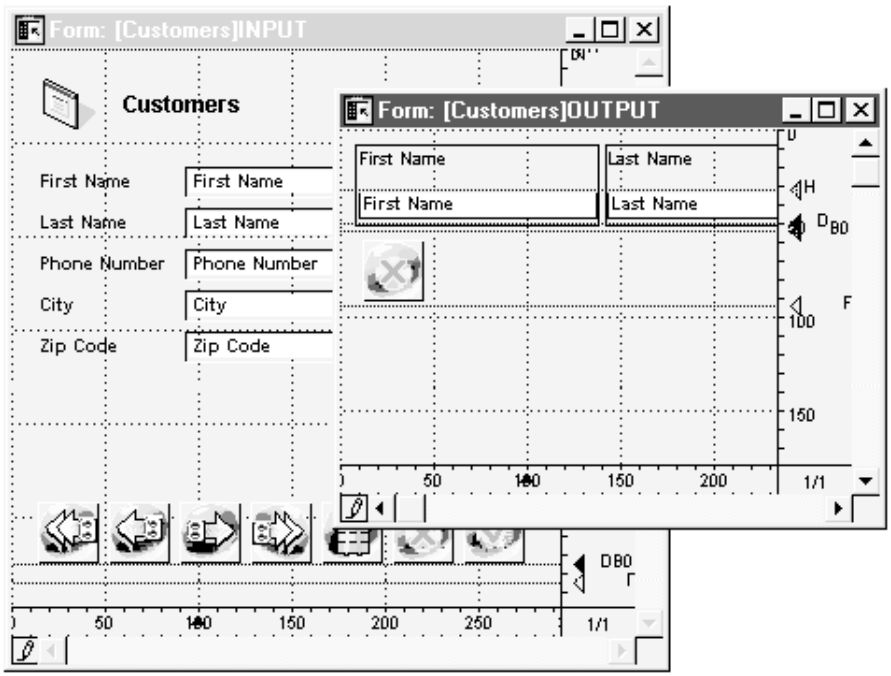
4Dメニューバー、フォーム、メソッドを使用すると、4D Webサーバセッションを完全に管理することができます。前述の例では、レコードの一覧表示と追加処理が簡単な4Dメソッドとフォームで実行されました。HTMLホームページがなかった場合には、Webブラウザには接続上でメニューバー番号1が表示されたでしょう。

HTMLホームページを省略した場合には、データベースのクライアント／サーバトランザクションをサポートするWebサーバの構築は、1ユーザまたは複数のユーザに対してWindowsまたはMacintosh上に4Dデータベースを構築することで構成されます。次の手順は、この方法でサンプルデータベースを作成する工程を説明しています。

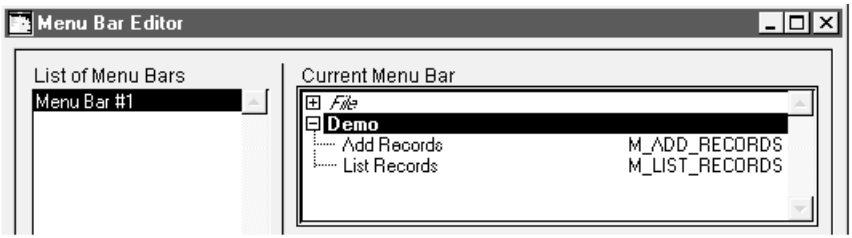
1. 次の図はサンプルデータベースのストラクチャを示しています。



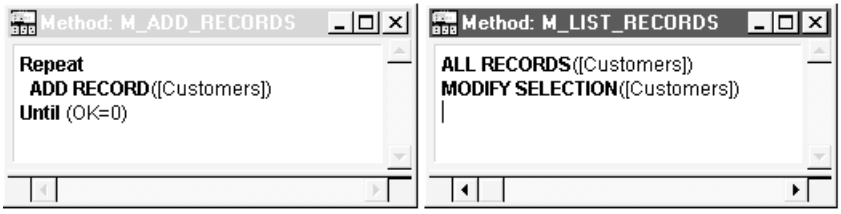
2. レコードの操作ができるように入力フォームと出力フォームを追加します。



3.「カスタム」メニューで作業を行い、Web接続をサポートするようにメニューバー番号1を追加します。



4. 2つのプロジェクトメソッドを作成します。



これで終りです。

5分もあれば、ローカルで操作できるデータベースであり、しかもイントラネットネットワークやインターネットに公開できるWebサーバでもある4Dデータベースの作成が完了します。

次の節の「Web Services：入門編(パートII)」を参照してください。

参照：SEND HTML FILE, SET WEB DISPLAY LIMIT, SET WEB DISPLAY LIMIT, SET WEB TIMEOUT, START WEB SERVER, STOP WEB SERVER

Webサービス：入門編(パートII)

データベースにHTML的部分を追加する

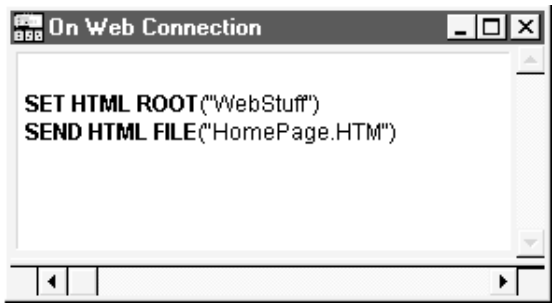
Webユーザに対してデータベースのメニューバー番号1以外にもやり取りをさせたいと考えているのであれば、4DフォームもHTMLページも表示するように「On Web Connection」データベースメソッドに機能を追加することができます。任意のHTMLツールで作成した既存のサイトにあるHTMLページを再利用できます。

4DベースのWebサイトは、完全な4Dベースシステムか、または4DフォームとHTMLページの組み合せにすることができます。4Dデータベースの中からHTMLページを使用する際に注目すべき点は、4DとHTMLの両方の開発環境から長所を得るということです。必要がなければ、HTMLページを使用する必要はありません。

例：

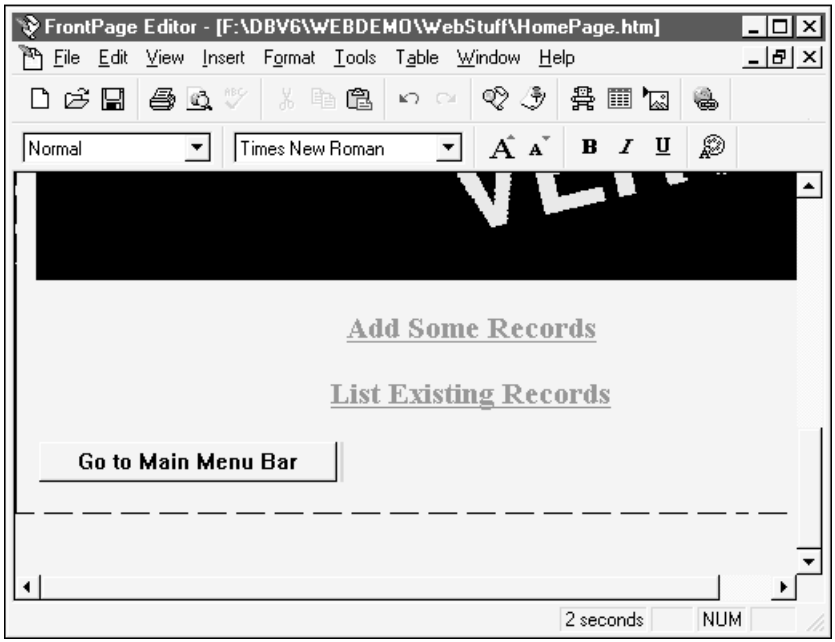
この例は、既存のHTMLページをデータベースに追加します。次の図は、データベースのディレクトリを示しています。

データベース用のホームページとしてHTMLドキュメント「HomePage.HTM」を使用します。ドキュメント「4DV6Logo.GIF」はこのHTMLドキュメントで使用されているピクチャです。このHTMLドキュメントは、以下に示す「On Web Connection」データベースメソッドで4Dデータベースに追加されます。

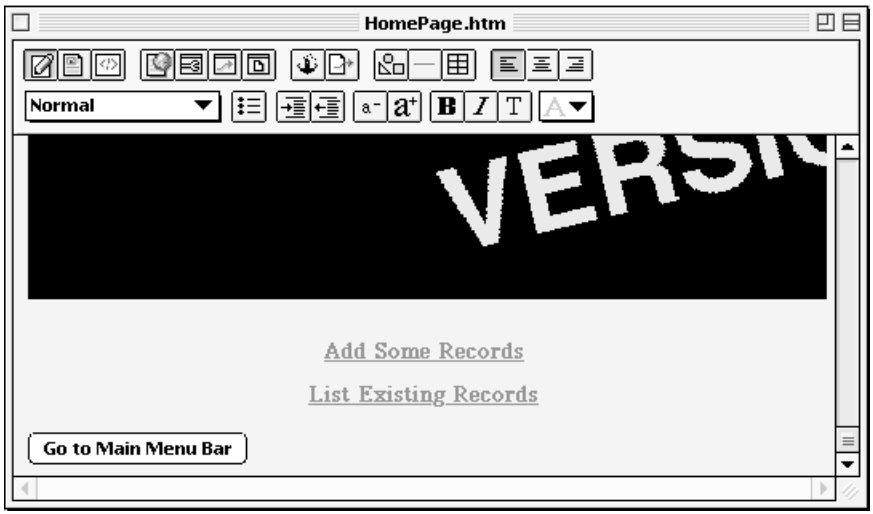


上記の**SET HTML ROOT**コマンドは、4th Dimensionに対してデフォルトでどこにあるHTMLドキュメントを検索するのかを通知します。**SEND HTML FILE**コマンドは、接続したブラウザにカレントWebページとしてHTMLドキュメントを送信します。

次の図は、Microsoft Front Pageで表示したHomePage.HTMドキュメントです。



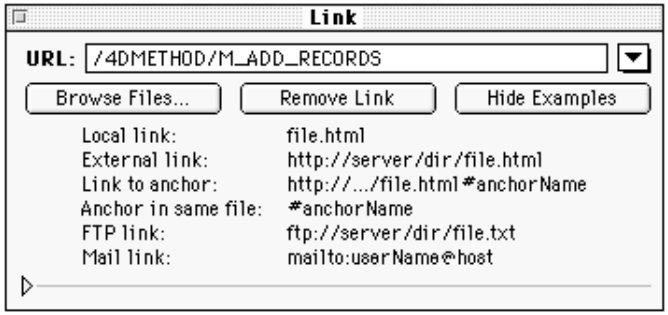
次の図は、同じHTMLドキュメントをClaris Home Pageで表示したものです。



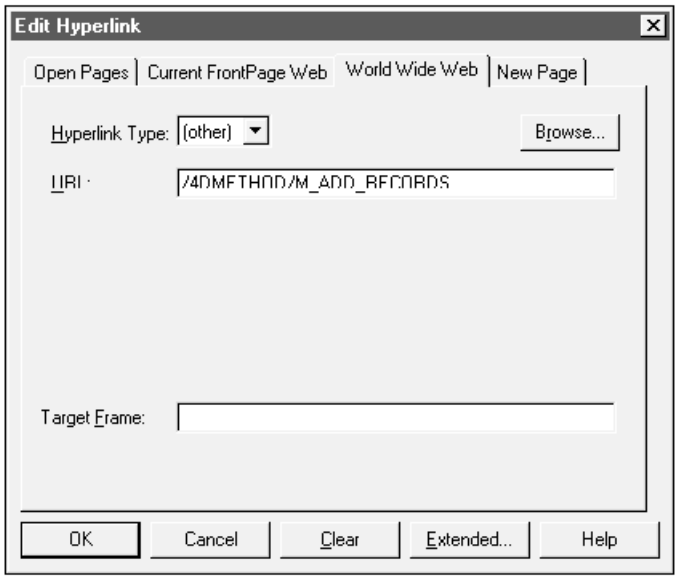
URLのリンク

「Add Some Records」と「List Existing Records」という2つのリンクされたテキスト項目は、4Dプロジェクトメソッドの「M_ADD_RECORDS」と「M_LIST_RECORDS」の実行をそれぞれのURL経由でトリガします。その表記方法は非常に簡単で、HTMLオブジェクトはURL "/4DMETHOD/Name_of_your_Method"で自分のデータベースのプロジェクトメソッドにリンクできます。

次の図は、Claris Home Pageでのテキスト"Add Some Records"へのURLです。



次の図は、前述と同じURLをMicrosoft Front Pageで指定する場合です。



これらのリンクが定義された後、WebブラウザがURLを送り返すときに、4Dは「/4DMETHOD/」キーワードの後に指定されているプロジェクトメソッドを実行します。実行後、プロジェクトメソッドが完了すると、実行をトリガしたHTMLページに戻ります。プロジェクトメソッド自体が4Dフォームや他のHTMLページなどを表示できることに注目してください。

ボタン

次の例のHTMLドキュメントには、レコードのサブミットに使用するボタンが含まれています。HTMLのボタンには、ノーマル、サブミット、リセットという3種類があります。

■ **「ノーマル」ボタン：**ノーマルボタンは、「/4DMETHOD/」キーワードを使用して4Dメソッドを参照するURLの属性になることができます。ノーマルボタンはナビゲーションのために使用されます。

■ **「サブミット」ボタン：**サブミットボタンは、ユーザが入力した値があればその値でフォームをWebサーバにサブミットします。このボタンは、普通の4Dフォームよりも HTMLページ経由でデータ入力を実行する方がよいときに便利です。

■ **「リセット」ボタン：**リセットボタンは、4D開発環境の中ではそれほど便利ではありません。リセットボタンはユーザがフォームに入力した値があればそれをクリアし、サーバに対してはリクエストは送信しません。

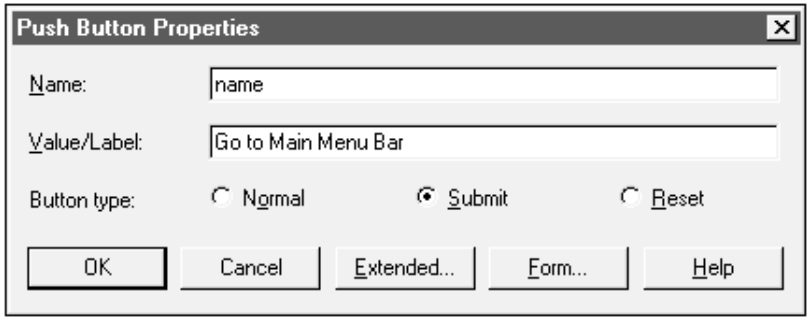
HTMLページを4Dに統合している間に、最も多く使用するのは「ノーマル」ボタンと「サブミット」ボタンです。

実行する4Dメソッドの指定

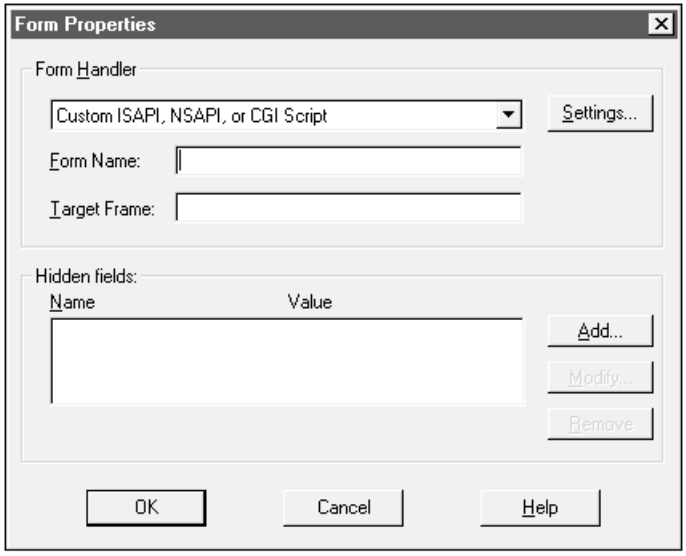
4D側からHTMLフォームをサブミットするには、フォームがサブミットされた後で4Dで実行されるPOSTアクション4Dメソッドを指定する必要があります。

Microsoft Front Pageを使用して、POSTアクション4Dメソッドを指定するには次のように行います：

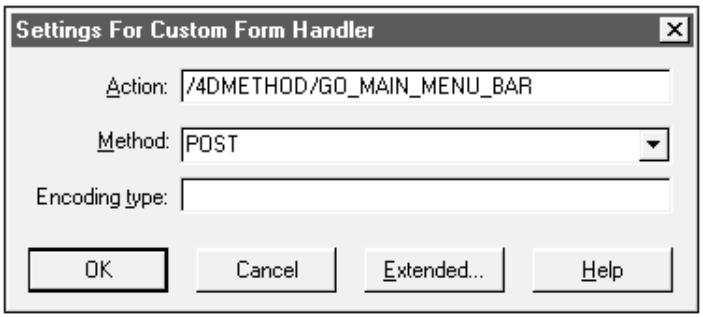
1.「サブミット」ボタンのプロパティの内容を確認します。次のダイアログボックスが表示されます。



2.「Form...」ボタンをクリックします。「フォームプロパティ」ダイアログボックスが表示されます。



3.「Settings」ボタンをクリックします。次のダイアログボックスが表示されます。



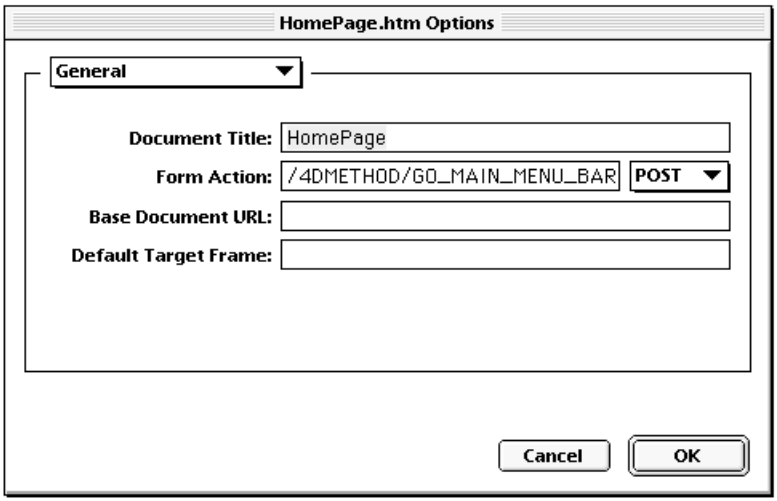
4. リンクされたURLに、アクションとして「/4DMETHOD/Name_of_your_Method」を指定します。

ここで、「Go Main Menu Bar」サブミットボタンがクリックされたときに実行される4Dメソッドとして「GO_MAIN_MENU_BAR」を入力します。

5.「Method」ドロップダウンリストから「POST」を選択します。

Claris Home Pageを使用してPOSTアクション4Dメソッドを指定するには、次のように行います：

1.「Editメニュー」から「Document Options...」を選択します。次のダイアログボックスが表示されます。

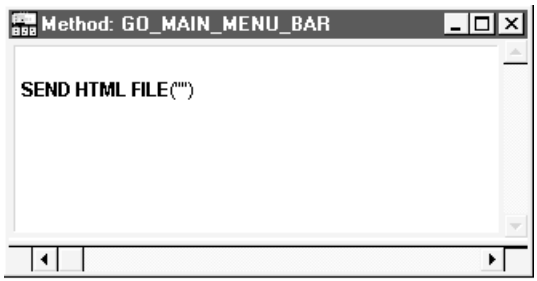


2. ポップアップメニューから「POST」を選択します。

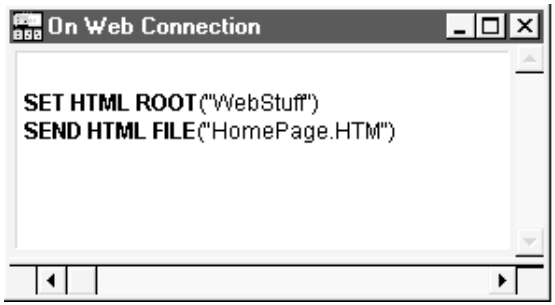
3. Form Actionとして「/4DMETHOD/Name_of_your_Method」を指定します。

プロジェクトメソッド

次は、4Dプロジェクトメソッド「GO_MAIN_MENU_BAR」です。



この例で、このメソッドには1つの目的だけがあります。その目的とは、Webブラウザで表示されているカレントHTMLページから出るということです。これを行うためには、単に空の文字列を渡して**SEND HTML FILE**コマンドを呼び出します。これは4Dに対してカレントHTMLページを終了し、**SEND HTML FILE**コマンドを発行した4Dプロジェクトメソッドのコードの行に戻るように命令しています。この行で「HTMLモード」が始まります。この例の場合は、「On Web Connection」データベースメソッドを実行しているところに戻ることを意味します。



SEND HTML FILEコマンドの呼び出しはこのメソッドの最後の行だったので、「On Web Connection」データベースメソッドはそれに従って終了し、4Dはデータベースのメニューバー番号1に切り替わります。

これで終りです。5分以内に、Webページを設計し、「GO_MAIN_MENU_BAR」4Dプロジェクトメソッドを追加することで、クライアント／サーバ機能をHTML開発環境に組み合せたデータベースとWebサーバを持つことができます。

次は何をするのか？

■ 4DにHTMLフォームとコードを統合することの詳細については、「Webサービス：HTMLとJavascriptのカプセル化」の節を参照してください。

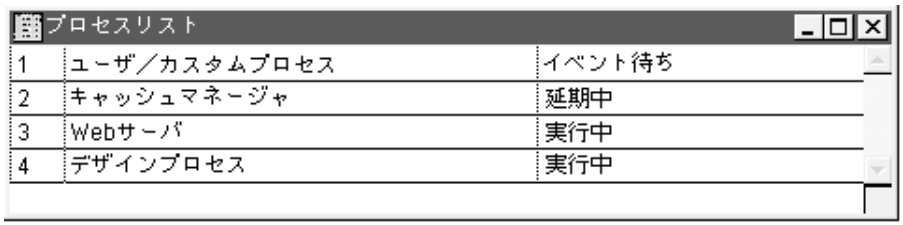
■ 最初の4D Webサーバを設定しているときに問題が発生した場合には、「Webサービス：システム設定」の節を参照してください。

参照：SEND HTML FILE、SET HTML ROOT、SET WEB TIMEOUT、START WEB SERVER、STOP WEB SERVER

Webサービス：「Web接続」プロセス

「Webサーバ」プロセス

Webサーバプロセスは、データベースがWebサイトとして公開されているときに稼働し、実行します。次に示す「デザイン」モードの「プロセスリスト」ウィンドウでは、「Webサーバ」プロセスは稼働中で実行中の3番目のプロセスです。



これは、4Dカーネルプロセスです。つまり、「デザイン」モードの「プロセス」メニューの「アボート」メニューコマンドを使用してこのプロセスをアボートすることはできません。また、**CALL PROCESS**などのコマンドを使用してプロセス間通信を行うこともできません。「Webサーバ」プロセスはユーザインタフェースコンポーネント(ウィンドウ、メニューなど)を持たないことに注意してください。

「Web サーバ」プロセスは、次の方法で起動できます。

■ 「ユーザ」モードの「Webサーバ」メニューから「Web サーバ開始」を選択する。

■ 4Dコマンドの**START WEB SERVER**コマンドを呼び出す。

■ 「データベースプロパティ」の「起動時にデータベースを公開する」チェックボックスが選択されているデータベースをオープンする。

「Web サーバ」プロセスは、次の方法で停止できます。

■ 「ユーザ」モードの「Webサーバ」メニューから「Web サーバ停止」を選択する。
■ 4Dコマンドの**STOP WEB SERVER**コマンドを呼び出す。
■ 現在発行されているデータベースを停止する。

「Webサーバ」プロセスの目的は、Web接続を処理するだけです。「Webサーバ」プロセスの起動は、実際のWeb接続のオープンは意味せず、Webユーザに対してWeb接続の初期化を許可する意味しかありません。「Webサーバ」プロセスの停止は現在稼働中の「Web接続」プロセスがあればそれをクローズするという意味はなく、Webユーザに対して新しいWeb接続の初期化を許可しなくなるという意味しかありません。

「Webサーバ」プロセスを停止するときに、オープンされている「Web接続」プロセスが存在する場合には、Webユーザが（「データベースプロパティ」ウインドウの「接続」ページにある「Webサーバ接続タイムアウト」での設定または**SET WEB TIMEOUT**コマンドをプログラムで使用して設定した）Webサーバ接続タイムアウトと同じかそれを超える経過時間のためにデータベースの照会を停止するまで、これらのプロセスは実行を継続します。

「Web接続」プロセス

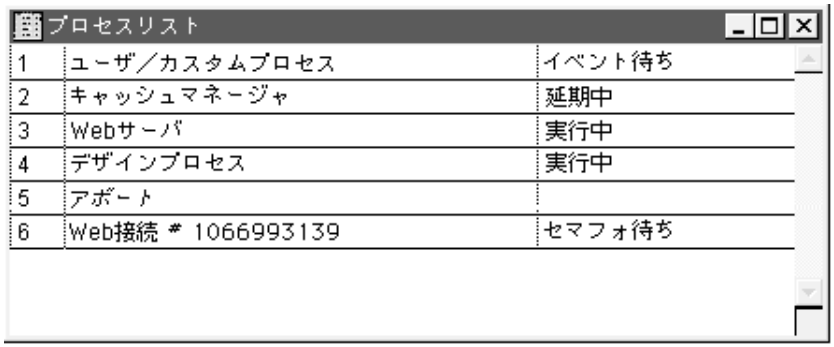
Webブラウザがデータベースへの接続を試行するたびに、接続リクエストは「Webサーバ」プロセスによって処理され、以下の手順が実行されます。

■ まず最初に、「Webサーバ」プロセスは新規接続で利用できるWebライセンスが存在するかどうかを確認します。すべてのライセンスがすでに使用されている場合には、Webブラウザに以下のメッセージを送信します。

"This database has not been setup for the Web yet"(このデータベースはWeb用に設定されていません)

■ 利用可能なライセンスがある場合には、「Webサーバ」プロセスは一時的な4Dプロセスを作成して、Webブラウザとの接続を初期化します。この一時的なプロセスは非常に高速に実行した後アボートします。

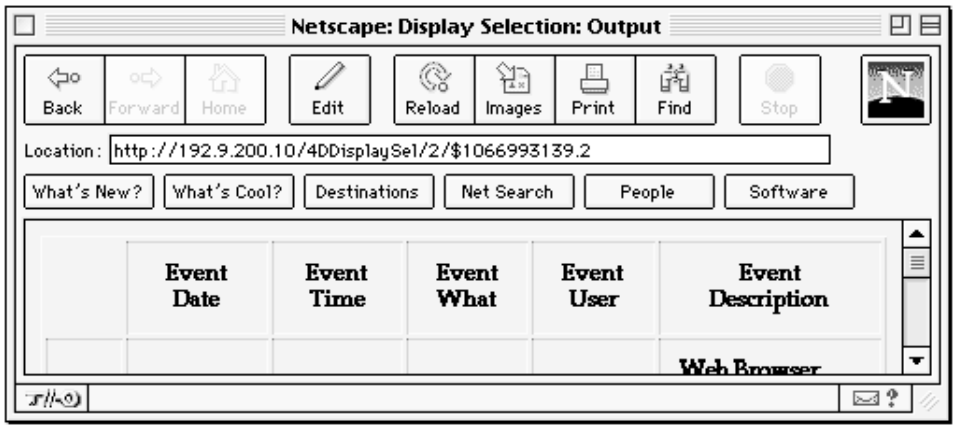
■ Web接続が正常に初期化された場合には、「Web 接続」プロセスが起動します。このプロセスが、この接続の全Webセッションを扱います。以下に示す「デザイン」モードの「プロセスリスト」ウィンドウは、Webブラウザ接続が初期化された後に起動された「Web 接続」プロセス “ Web 接続 # 1066993139 ” を表示しています。



また、上の図の中で、5番目のアボートしたプロセスにも注目してください。このプロセスは「Web サーバ」プロセスによって起動され、停止されたものです。このプロセスがWeb接続の初期化を処理したのです。

Web接続コンテキストID

「Web接続」プロセスの名前に示されている番号を“ コンテキストID ”といいます。これはランダムに生成され、各Web接続を一意に特定しています。コンテキストIDは、Web接続中、4D側とブラウザ側の両方で保守されます。この例では、コンテキストIDは“ 1066993139 ”です。次のWebブラウザウィンドウでは、ブラウザのLocationエリアに表示されたURLでこの番号を確認することができます。



URLはWebセッション中には4Dによって自動的に保守されます。HTTPリクエストが4DのTCP/IP ネットワークコンポーネントで受信されるたびに、4DはURLからコンテキストIDを抽出することで正しい「Web接続」プロセスへのリクエストをリダイレクトすることができます。

コンテキストIDでは、次のことを行うことができます。

■ 4Dによって各Web接続経由でWebとデータベースの両方を保守できます。

■ 複数の同時実行Web接続を透過的に処理できる。

■ 異なるコンテキストIDが各接続で生成されるので、ブックマークを使用するときに将来発生する可能性のある、希望とは違う接続を行わない。

Webとデータベースセッションの同期化: Web接続サブコンテキストID

上記のウィンドウでは、コンテキストIDの後にドットと数値の2（.2）のサブコンテキストIDがあります。4Dは、ブラウザに新しい4DベースのHTMLページが送信されるたびにこの番号を自動的に管理し、1つずつ増分します。サブコンテキストIDは、データベースセッションの保守において基本的なものです。

通常、Webブラウザには、Backボタン、Forwardボタン、Historyボタンなどのナビゲーション制御が含まれています。これらの制御は、ドキュメント、ニュース、掲示板などをブラウズするときに便利です。これらはデータベーストランザクションを実行するときよりは興味をそそられないでしょう。

例えば、Webユーザがテーブルにレコードを追加している場合には、データ入力の妥当性検査がされているのか、つまり、Webユーザが4Dフォームの「登録」ボタンまたは「キャンセル」ボタンをクリックしたかどうかを開発者は知る必要があります。この時点で、Webユーザが他のページをナビゲートする場合には、データ入力は不確定な状態のままになっています。これを防ぐために、4Dはブラウザ側のWebセッションと4D側のデータベースセッションをサブコンテキストIDを使って同期させています。

フォームがサブミットされたり、HTTPリクエストがブラウザから4Dに送信されるたびに、Webとデータベースセッションの同期化解除が検出された場合には、4Dは"Using browser navigation controls, you left a form requiring data validation. 4th Dimension will now return to that form so you can accept or cancel it"(ブラウザナビゲーション制御を使用しているので、データ妥当性検査が必要なフォームをそのままにしています。このフォームの受け付けまたはキャンセルができるように4th Dimensionはこのフォームに返ります)というメッセージを送信します。そして、4DはサブコンテキストIDを使ってWebデータ入力ページに戻ります。

この同期化は、「Web接続」プロセスにとっても基本的なものです。例えば、「**ADD RECORD ([...])**」などを使用して4Dコードの実行を促進するために正確にフォームから出る必要があります。

同期化は選択的なものです。ブラウザ側に表示されているカレントWebページが4Dフォームである場合(**ADD RECORD**、**DISPLAY SELECTION**、**DIALOG**など)、最後には同期化が発生します。

カレントWebページが別のWebページからのリンクでアクセスされたHTMLページである(**SEND HTML FILE**コマンドで送信された)場合、ページを自由にナビゲートすることができます。

次の4Dコードがあるとします。

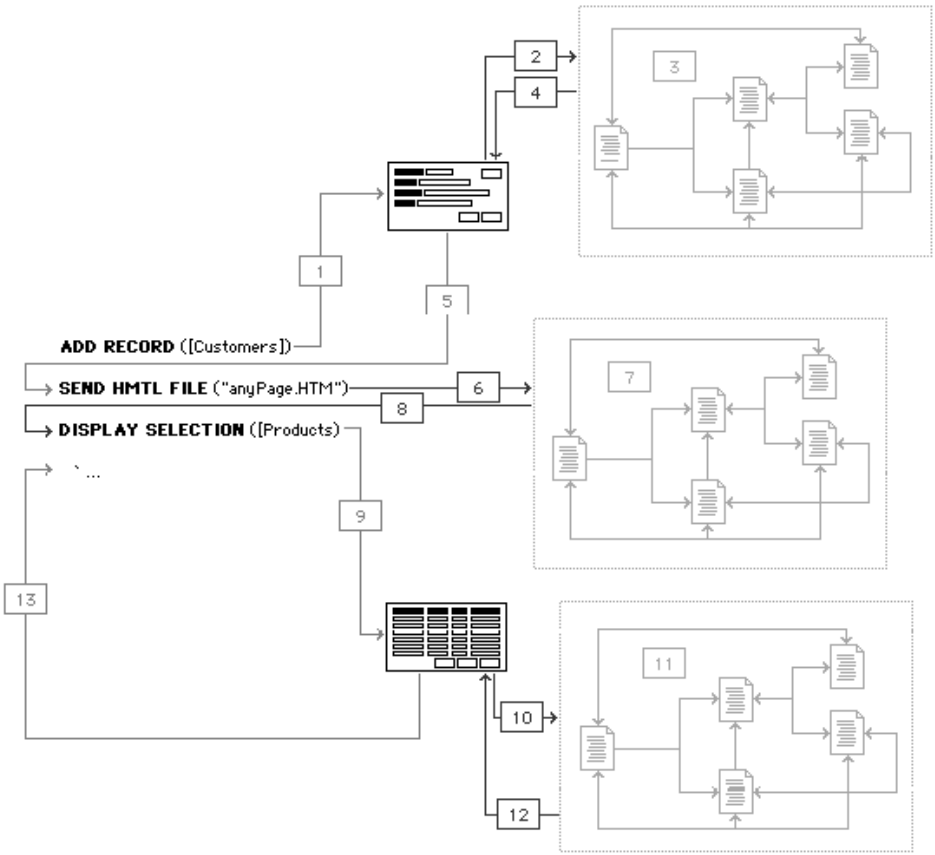
```
ADD RECORD ([顧客])
SEND HTML FILE ("anyPage.HTM")
DISPLAY SELECTION ([商品])
```

次の図は、実行中に4DとWebブラウザの両方で発生することを詳細に示しています。

■ 赤の線は、4Dフォーム変換とサブミッションを示しています。

■ 青の線は、4DベースのHTMLページと4Dベースではないページとの切り換えを示しています。

■ 緑のエリアは、4Dベースではないページを示しています。



手順の説明

(1) **ADD RECORD**コマンドが発行されます。4Dはテーブルのカレント入力フォームをHTMLページに変換して、それをWebブラウザに送信します。入力フォームがマルチページフォームの場合には、標準4Dページナビゲーションボタンによって、フォームのページ間をナビゲートできます。この4Dベースのナビゲーションが実装され、4Dによって透過的に実行されます(Webフォームサブミッション経由)。

(2) データ入力中(つまり、**ADD RECORD**コール内)に、ボタンがクリックされ、そのオブジェクトメソッドが**SEND HTML FILE**コールを発行します。

(3) **SEND HTML FILE**コール内部で、HTMLページにリンクがある場合には、複数のページをナビゲートすることが可能です。そして、「**SEND HTML FILE ("")**」が発行されたときにHTMLモードは終了します。

(4) クリックされたボタンのオブジェクトメソッドと、**ADD RECORD**コマンドで初期化されたデータ入力が実行されます。手順(2)と(3)はデータ入力内で数回繰り返すことができます。

(5) 最終的に、データ入力が受け付けられるか、キャンセルされ、「Web接続」プロセスが実行されます。

(6) 次のコールは、**SEND HTML FILE**コマンドです。

(7) この手順は手順3と似ています。HTMLページにリンクがある場合には、複数のページをナビゲートすることが可能です。そして、「**SEND HTML FILE ("")**」が発行されたときにHTMLモードが終了します。

(8)「Web接続」プロセスが実行されます。

(9) **DISPLAY SELECTION**コマンドが発行されます。4Dはテーブルの現在の出力フォームをHTMLページに変換して、それをWebブラウザに送信します。**DISPLAY SELECTION**コマンド中に、4Dは選択ページと個別のレコード表示の間を透過的にナビゲートします。また、4Dは**MODIFY SELECTION**コマンドを使用してデータ入力の管理とレコードロックをWebフォームサブミッションから行うことができます。

(10) 選択経由でのナビゲーション中に、フォームのフッタエリアにあるボタンがクリックされ、そのオブジェクトメソッドがSEND HTML FILEコールを発行します。

(11) この手順は手順3および手順7と似ています。HTMLページにリンクがある場合には、複数のページをナビゲートすることが可能です。そして、「**SEND HTML FILE ("")**」が発行されたときに、HTMLモードが終了します。

(12) クリックされたボタンのオブジェクトメソッドと、**DISPLAY SELECTION**コマンドで初期化された選択表示が実行されます。手順(10)と(11)は選択のナビゲーション中に数回繰り返すことができます。

(13) 最終的に、選択表示が終了し、「Web 接続」プロセスが実行されます。

以下同様。

自由なWebナビゲーション(例えば、BackボタンやForwardボタンをクリックすること)は、任意の**SEND HTML FILE**(コマンド前述の図の緑のエリア)内で可能です。その一方で、任意の4DベースのHTMLページ(データ入力、選択表示など、**CONFIRM**コマンドや**Request**関数によって表示されたものの標準ダイアログボックスを含む)はブラウザナビゲーション制御の1つを使用して終了され、4Dは最後には「Web接続」プロセス側で現在実行されている発行済の4Dコマンドに対応するサブコンテキストIDを持つWebページに戻ることによって、Webセッションとデータベースセッションを同期化します。

「Web接続」プロセスとWebセッション

ユーザの立場からすると、Webブラウザ側でのユーザのアクションがWebセッションを誘導します。

プログラム作成の立場からすると、「Web接続」プロセスがWebセッションを誘導するのであって、Webセッションが「Web接続」プロセスを誘導する訳ではありません。Webブラウザは「Web接続」プロセスが送信したページを表示します。それは、次のいずれかを行います。

■ 4Dコードを実行する、または

■ カレントWebページのブラウザからのサブミッションを待機する。

設計者の立場からすると、「Web接続」プロセスは実行のドメインが4th Dimensionまたは4D Serverである4Dプロセスとして見えるべきですが、そのユーザインタフェースは接続されたWebブラウザ上でエコーされます。

このようなことから、Webデータベースアプリケーションの設計を行うときには、この「Web接続」プロセスの二重性を常に考慮してください。次の例を参照してください。

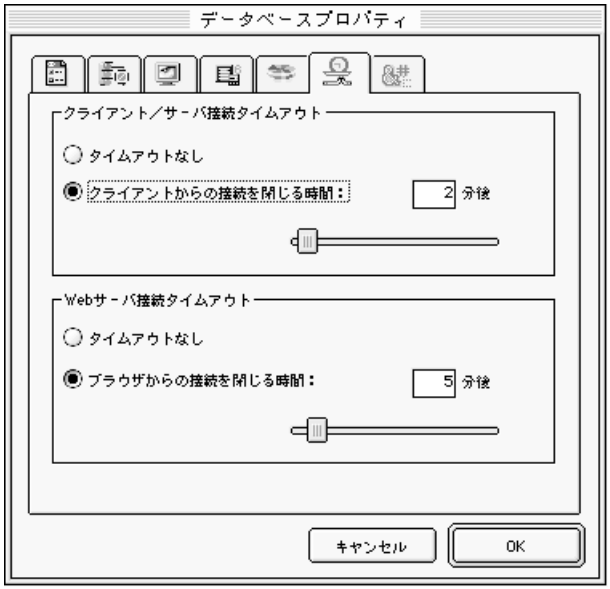
■ どんな種類のデータ入力でも、その間、メインメニューバーはブラウザのものであり、4Dのメニューバーではありません。フォーム内では、4Dメニューバーを考慮に入れないでください。4DメニューバーはWebサーバマシン上にあり、Webブラウザマシン上にはありません。

■ Webブラウザで使用されるフォームを設計するときには、機能の4DフォームセットはHTMLのフォームセットでは制限されていることを考慮してください(ただし、時には4Dの追加機能もあります)。4Dフォーム機能のセット(例：オブジェクトタイプやフォームイベント)を使用できるとは思わないでください。詳細については、「Webサービス：HTMLとJavascriptのカプセル化」の節を参照してください。

■ プロセス間通信によって、**CALL PROCESS**コマンドは「Web 接続」プロセスに適用されたときには、現在アクティブなフォームがWebブラウザに表示されているので、何の影響もありません。一方、「Web接続」プロセスは別の4Dプロセスに対して**CALL PROCESS**コマンドを発行できます。さらに、プロセス間通信は**GET PROCESS VARIABLE**コマンドおよび**SET PROCESS VARIABLE**コマンドを使用することで双方向に異なった形で実行することができます。これには、プロセスでユーザインタフェースを持つ必要はありません。

Web接続タイムアウト

前述したように、「Web接続」プロセスは4Dコードを実行しているか、またはブラウザ側で現在表示されているWebページのサブミッション用に待機しているかのいずれかです。後者の場合には、「Web接続」プロセスは、「データベースプロパティ」ウィンドウ(次の図参照)で設定するか、**SET WEB TIMEOUT**コマンドでプログラムによって設定されるWebサーバ接続タイムアウトと同じかそれを超える遅延を待ちます。



「Webサーバ接続タイムアウト」設定のスコープは、データベースセッションです。

すべての「Web接続」プロセスは、タイムアウト時間の値に従います。値の設定が変更されると、プロセスは即座に影響を受けます。デフォルト値は、5分間です。

必要に応じてこのタイムアウトを長くたり、短くしたりすることができます。例えば、Webユーザがデータベースから提供されるページ内のHTMLリンクを経由して他のWebサイトにサーフィンすることをアプリケーションが許可している場合には、タイムアウトを長くことができます。タイムアウトを長くすことで、ユーザはデータベースへの接続をクローズすることなく、他のWebサイトにさらに長い時間ナビゲートできます。

警告：バージョン6では、「Web接続」プロセスをプログラムで停止する方法がありません。長いタイムアウトを指定した場合には、Webユーザがかなり前にWeb接続との作業を終っていてもプロセスはその遅延を待つことになります。「タイムアウトなし」オプションを指定すると、「Web接続」プロセスはデータベースが終了したときにだけ停止します。

Tips：Webデータベースの開発とテストを行うときには、4th Dimensionまたは4D Serverで提示されているWebライセンスが許可するWeb接続の最大値に達することがあります。例えば、これは「On Web Connection」データベースメソッドをデバッグしている間に発生することがあります。このメソッドでは接続を複数回再試行しているためです。Webサーバをオフにすることはタイムアウト遅延の完了を待っている「Web接続」プロセスをアボートすることにはなりません。ただし、「Webサーバ」プロセスとは違って、「Web接続」プロセスは「プロセス」メニューの「アボート」コマンドを使用してアボートすることができます(これは、「プロセスリスト」ウィンドウが最前面のウィンドウであるときに「デザイン」モードで利用できます)。

「On Web Connection」データベースメソッド

「On Web Connection」データベースメソッドは、Webブラウザがデータベースへの接続を初期化するたびに4th Dimensionまたは4D Serverから自動的に呼び出されます。これは、データベースがWebサーバとして公開されたときにだけ発生します。

「On Web Connection」データベースメソッドは、4Dから渡される2つのテキスト引数を受け取ります。これらの2つの引数は、次のように宣言できます。

```
` 「On Web Connection」データベースメソッド
C_TEXT($1 ; $2)
  ` メソッドのコード
```

URLエクストラデータ

最初の引数($1)は、ブラウザのロケーションエリアでユーザが入力したURLであり、ホストアドレスが削除されたものです。

イントラネット接続を例に説明しましょう。4D WebサーバマシンのIPアドレスが「123.4.567.89」であると仮定します。次の表は、Webブラウザに入力されたURLに従った$1の値を示しています。

Web ブラウザロケーションエリアに入力されたURL	引数 $1の値
123.4.567.89	/
http://123.4.567.89	/
123.4.567.89/Customers	/Customers
http://123.4.567.89/Customers	/Customers
http://123.4.567.89/Customers/Add	/Customers/Add
123.4.567.89/Do_This/If_OK/Do_That	/Do_This/If_OK/Do_That

この引数は、必要に応じて自由に使用してください。4DはURLのホスト部分以外に渡される値は単に無視します。

例えば、値"/Customers/Add"で「[顧客]テーブルに新しいレコードを追加するために直接移動せよ」という意味の表記を設定することができます。データベースのWebユーザに対して利用可能な値のリストやデフォルトのブックマークを指定することで、アプリケーションのさまざまな部分へのショートカットを提供できます。この方法で、Webユーザは、データベースに新しく接続するたびにナビゲーションパス全体を通らずに、即座にWebサイトのリソースにアクセスできます。

警告：前回のセッションで作成されたブックマークでデータベースにユーザが再度入るのを避けるために、4Dは標準4D URLのいずれかに対応するURLがあればそれを途中で処理します。

HTTPリクエストのヘッダ

2番目の引数($2)は、Webブラウザから送信されたHTTPリクエストのヘッダです。このヘッダが「On Web Connection」データベースメソッドにそのまま渡されていることに注意してください。そのコンテンツは接続を試行するWebブラウザの特徴によって変わります。

Windows NT上で稼働するNetscape 3.0では、次のようなヘッダを受け取ります。

```
GET HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/3.01 (WinNT; I)
Host: 192.9.200.11
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
```

Windows NT上で稼働するMicrosoft Internet Explorerでは、次のようなヘッダを受け取ります。

```
GET / HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, */*
Accept-Language: en
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0d; Windows NT)
Connection: Keep-Alive
If-Modified-Since: Sunday, 10-Dec-96 01:51:37 GMT
```

アプリケーションがこの情報を扱う場合には、ヘッダの解析はユーザの責任になります。

例：クライアントのローカルホームページの実装

次の例では、「On Web Connection」データベースメソッドに送られる引数$1は、ある組織の中でクライアントホームページを実装するために使用されます。

データベースには、[顧客]と[商品]という2つのテーブルがあります。次に示す「On Startup」データベースメソッドは、後で「On Web Connection」データベースメソッドで使用されるインタープロセス配列を初期化します。

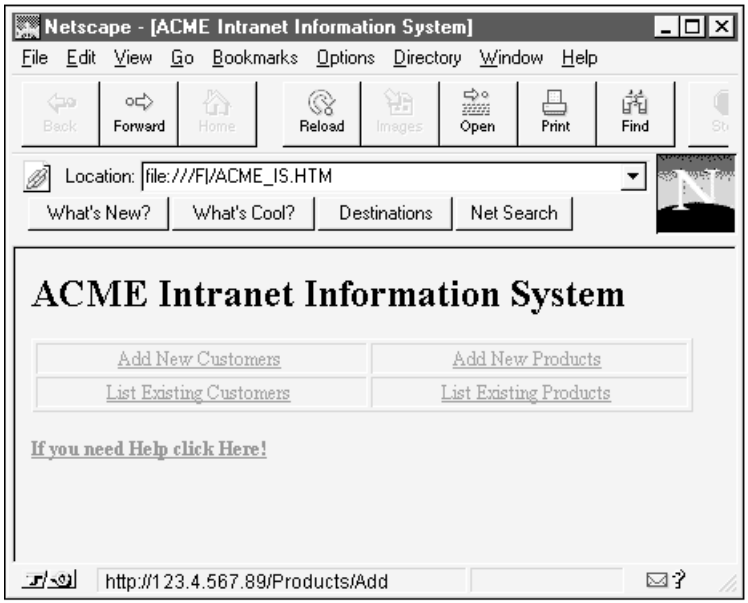
```
` 「On Startup」データベースメソッド
` テーブルリスト
ARRAY STRING(31 ; <>asTables ; Count tables)
For ($vlTable ; 1 ; Size of array(<>asTables))
   <>asTables{$vlTable}:=Table name($vlTable)
End for

   `ログイン時の標準Webアクション
ARRAY STRING(31 ; <>asActions ; 2)
<>asActions{1}:="Add"
<>asActions{2}:="List"
```

「On Web Connection」データベースメソッドの主な仕事は、アドレスのホスト部分の後ろにあるURLに渡されたエクストラデータを解読して、それに従ってアクションを行うことです。このメソッドは、以下のとおりです。

```
` 「On Web Connection」データベースメソッド
C_TEXT($1 ; $2)
C_TEXT($vtURL)
    `万一に備えて$1が"/"または"/..."と同じかどうかをチェックする
If ($1="/@")
        `URLから最初の"/"を引いたものをローカル変数にコピーする
    $vtURL:=Substring($1 ; 2)
        `URLを解析し、URLのトークンでローカル配列を登録する
        `例えば、、URLのエクストラデータが"aaa/bbb/ccc"の場合には、
        `結果の配列は"aaa"、"bbb"、"ccc"の順の3つの要素になる
    $vlElem:=0
    ARRAY TEXT($atTokens ; $vlElem)
    While ($vtURL  #  "")
        $vlElem:=$vlElem+1
        INSERT ELEMENT($atTokens ; $vlElem)
        $vlPos:=Position("/" ; $vtURL)
        If ($vlPos>0)
            $atTokens{$vlElem}:=Substring($vtURL ; 1;$vlPos-1)
            $vtURL:=Substring($vtURL ; $vlPos+1)
```

```
        Else
            $atTokens{$vlElem}:=$vtURL
            $vtURL:=""
        End if
    End while
    `URLのHOST部分の後にエクストラデータが渡された場合
    If ($vlElem>0)
        `On Startupデータベースメソッドで初期化されたインタープロセス配列を
        `使用して、1番目のトークンがテーブルの名前であるかどうかをチェックする
        $vlTableNumber:=Find in array(<>asTables ; $atTokens{1})
        If ($vlTableNumber>0)
            `その場合には、このテーブルに対するポインタを取得する
            $vpTable:=Table($vlTableNumber)
            `入力フォームと出力フォームを設定する
            INPUT FORM($vpTable-> ; "Web入力")
            OUTPUT FORM($vpTable-> ; "Web出力")
            `On Startupデータベースメソッドで初期化されたインタープロセス配列を使用
            `して2番目のトークンが既知の標準アクションであるかどうかをチェックする
            $vlAction:=Find in array(<>asActions ; $atTokens{2})
            Case of
                `レコードの追加
                \ ($vlAction=1)
                    Repeat
                        ADD RECORD($vpTable-> ; *)
                    Until (OK=0)
                    `レコードの一覧表示
                \ ($vlAction=2)
                    READ ONLY($vpTable->)
                    ALL RECORDS($vpTable->)
                    DISPLAY SELECTION($vpTable-> ; *)
                    READ WRITE($vpTable->)
            Else
            `ここで追加の標準テーブルアクションを実装できる
            End case
        Else
        `ここでその他の標準アクションを実装できる
        End if
    End if
End if
`これまでのコードで何が起こっても、通常のLog Onプロセスで追跡する
WWW NORMAL LOG ON
```

この時点で、この組織の人々は、データベースに接続し、リストされたメソッドで設定された表記に従ったURLを入力できます。また、ユーザは毎回URLを再入力したくない場合にはブックマークを作成することもできます。最終的なソルーションは、組織のメンバ全員に対してデータベースをアクセスするためにローカルで使用するHTMLページを提供することです。このHTMLページは、次のようになります。



つまり、HTMLページ「ACME_IS.HTM」は、組織内の4Dベースの情報システムに対するクライアントのローカルホームページなのです。ユーザが「Add New Products」リンクをクリックすると、Webブラウザはurlが「http://123.4.567.89/Products/Add」であるホストに接続します。データベースコンピュータのIPアドレスが「123.4.567.89」である場合、「On Web Connection」データベースメソッドは$1にURLのエクストラデータ"/Products/Add"を受け取り、このため[商品]テーブルへのレコード追加処理に進みます。

最後には、ユーザはデスクトップ上のそのページからのリンクをドラッグ&ドロップして、次に示す「Add New Customers」アイコンのようなInternet Shortcutアイコンを作成できます。これらのアイコンをダブルクリックするだけで、4D Webデータベースの任意の場所に直接移動できます。

次は、このHTMLページのソースコードです。

```
<HTML>
<HEAD>
 <TITLE>ACME Intranet Information System</TITLE>
</HEAD>
<BODY>
<H1><B>ACME Intranet Information System</B></H1>
<P ALIGN=CENTER><TABLE BORDER=1 CELLPADDING=1 WIDTH="100%">
 <TR>
 <TD>
  <P ALIGN=CENTER><A HREF="http://123.4.567.89/Customers/Add">Add New Customers</A>
 </TD>
 <TD>
  <P ALIGN=CENTER><A HREF="http://123.4.567.89/Products/Add">Add New Products</A>
 </TD>
 </TR>
 <TR>
 <TD>
  <P ALIGN=CENTER><A HREF="http://123.4.567.89/Customers/List">List Existing Customers</A>
 </TD>
 <TD>
  <P ALIGN=CENTER><A HREF="http://123.4.567.89/Products/List">List Existing Products</A>
 </TD>
 </TR>
</TABLE></P>
<P><B><A HREF="Help.HTM">If you need Help click Here!</A></B></P>
</BODY>
</HTML>
```

Webサービス：HTMLサポート

HTMLとJavaScriptコードを4Dアプリケーションに実装する前に、4th Dimensionがすでに提供しているものは何かを確認しましょう。

メニューバー：

■ 各メニューバーは、1つのHTMLページに変換されます。各メニュータイトルはテキストとしてだけ表示され、メニュー項目は4Dメソッドへのリンクとして表示されます。

■ Webブラウザ側のメニュー項目をクリックすると、「Web 接続」プロセス側にある対応する4Dメソッドの実行が開始されます。

フォーム：

■ オブジェクトは上から下へ、左から右への順で変換されます。ただし、そのHTMLは文書処理指向のアプリケーションであることに注意してください。HTMLでは横方向のオブジェクト位置が違い、また、ラップ処理が発生します。

■ マルチページフォームは、透過的にサポートされます。ただし、0ページ目はサポートされません。その結果、ページナビゲーションボタンは各ページ上にある必要があります。

■ 適切な場合には、自動アクションが透過的にサポートされます。

■ フォームイベント(On Load、On Unload、On Clicked)がサポートされます。それ以外のイベントはサポートされません。

■ ヘッダ、ディテイル（詳細）、ブレーク、フッタのタグは、**DISPLAY SELECTION**コマンドと**MODIFY SELECTION**コマンドへの呼び出し中に考慮されます。フォームのヘッダは、HTMLページの先頭に一度現れ、ディテイル（詳細）エリアは必要な数だけ繰り返され、また、自動選択ページナビゲーションリンクのすぐ下にある変数(ボタンなど)はHTMLページの一番下にあるフッタエリアに配置されます。

フィールドオブジェクト：

4DフォームがHTMLページに変換されるときに、フィールドオブジェクトは次のように変換されます。

4Dフィールドタイプ	HTMLオブジェクト	HTML Markup
文字	テキストフィールド (*)	<INPUT Type="text" ...>
テキスト	テキストフィールド (*)	<TEXTAREA ...> (**)
		<INPUT Type="text" ...> (***)
実数	テキストフィールド (*)	<INPUT Type="text" ...>
整数	テキストフィールド (*)	<INPUT Type="text" ...>
倍長整数	テキストフィールド (*)	<INPUT Type="text" ...>

日付	**テキストフィールド (*)**	**<INPUT Type="text" ...>**
時間	**テキストフィールド (*)**	**<INPUT Type="text" ...>**
ブール	**ラジオ、または**	**<INPUT Type="radio" ...>**
	チェックボックス(*)	**<INPUT Type="checkbox" ...>**
ピクチャ	**イメージ(常に入力不可)**	**<IMG SRC="..." ...>**
サブテーブル	**HTMLではサポートしてない**	**なし**
BLOB	**HTMLではサポートしてない**	**なし**

(*) または入力不可のときにはテキストのみ
(**) テキスト値が複数の行で構成されている場合
(***) テキスト値が1行だけまたは空行の場合

注：入力可能な変数は、変数と同じタイプのフィールドと似た動作になります。

フォームオブジェクト：

4DフォームがHTMLページに変換されるときに、フォームオブジェクトは次のように変換されます。

4Dフォームオブジェクト	**同等のHTMLオブジェクト**	**HTML Markup**
線	**水平ライン (1)**	**<HR>**
矩形	**HTMLではサポートしてない**	**なし**
楕円	**HTMLではサポートしてない**	**なし**
角の丸い矩形	**HTMLではサポートしてない**	**なし**
静止ピクチャ	**イメージ / イメージマップ(2)**	**<IMG SRC="...">**
		<INPUT Type="image" ...>
グループボックス	**テキスト**	**Text with font markups if any**
静的テキスト	**テキスト**	**Text with font markups if any**
ボタン	**サブミットボタン**	**<INPUT Type="submit" ...>**
デフォルトボタン	**サブミットボタン**	**<INPUT Type="submit" ...>**
ラジオボタン	**ラジオボタン (3)**	**<INPUT Type="radio" ...>**
チェックボックス	**チェックボックス**	**<INPUT Type="checkbox" ...>**
ポップアップメニュー	**ドロップダウンリスト**	**<SELECT ...>...</SELECT>**
ドロップダウンリスト	**ドロップダウンリスト**	**<SELECT ...>...</SELECT>**
メニュー/ドロップダウン	**ドロップダウンリスト**	**<SELECT ...>...</SELECT>**
コンボボックス	**ドロップダウンリスト**	**<SELECT ...>...</SELECT>**
スクロールエリア	**スクロールリスト (4)**	**<SELECT ...>...</SELECT>**
透明ボタン	**下記の備考2参照**	
ハイライトボタン	**下記の備考2参照**	
3Dボタン	**下記の備考2参照**	
ボタングリッド	**下記の備考2参照**	
グラフ	**イメージ(入力不可)**	**<IMG SRC="..." ...>**
プラグイン	**イメージ(入力不可)**	**<IMG SRC="..." ...>**

次のオブジェクトは、HTMLではサポートされていません。そのため、それらはトランザクション中は無視されます。

階層ポップアップメニュー、階層リスト、サブフォーム、タブコントロール、ラジオピクチャ、サーモメータ、ルーラ、ダイアル、ピクチャメニュー、ピクチャボタン、3Dチェックボックス、3Dラジオボタン

備考：

1. 水平方向以外の行は、HTMLではサポートされません。そのため、それらの行は無効になります。

2. 非表示的ボタンは、透明ボタン、ハイライトボタン、3Dボタン、ボタングリッドタイプのオブジェクトです。非表示的ボタンが静止ピクチャの上に重ならない場合には、静止ピクチャは固定イメージとして変換されます。非表示的ボタンが静止ピクチャの上に1つでも重なる場合は、静止ピクチャは「Server-Side Image Map」として変換されます。Webブラウザ側では、このイメージは「Server-Side Image Map」として扱われます。4D側では、サブミッションが受け付けられたときに、ボタンが実際にクリックされたかのように適切なボタン用に「On Clicked」イベントを生成するために、クリックの場所を4Dが再計算します。このようなことから、非表示的ボタンを管理することは、静止ピクチャと重なっている場合には非常に簡単です。非表示的ボタンは、通常の4Dインタフェースで行うように、「フォーム」メソッドまたはそれらのオブジェクトメソッドによって管理します。また、非表示的ボタンではWeb Image Mappingを処理するのが非常に簡単になります。非表示的ボタンは、静止ピクチャオブジェクトとまったく重ならない場合には、変換中は無視されます。

3. ラジオボタンのグループ化は、変換中は保持されます。

4. グループ化されているスクロールエリアは、HTMLではサポートされません。4Dは、同じ行にある独立したスクロールリストとしてグループ化されたスクロールエリアを変換します。

DISPLAY SELECTION／MODIFY SELECTIONコマンドによるレコード選択：

■ UserSetメカニズムは、サポートされません。

■ 自動選択ページングメカニズムが4Dから提供されます。詳細については、**SET WEB DISPLAY LIMIT**コマンドの説明を参照してください。

4Dコマンド：

4D Webデータベースを開発しているときに、コマンドが呼び出されたときに何が起こるのかを尋ねられることがあります。コマンドは、Webサーバマシン上で有効になるのか、あるいはWebブラウザマシン上で有効になるのでしょうか？「Web接続」プロセスはWebサーバマシン上で稼働していますが、そのユーザインタフェースは接続されているWebブラウザ上にリモートでエコーされます。従って、Webデータベースの開発では、4Dコマンドは次のように規定できます。

「Web 接続」プロセス内からの実行には影響を受けないコマンド：

CREATE RECORDのようなコマンドは、稼働中のプロセスの中で作動します。この場合、**CREATE RECORD**コマンドは「Web接続」プロセス内でレコードを作成します。同じことが**Screen width**などの関数にも当てはまります。この関数は、Webサーバマシン(プロセスが稼働しているマシン)の画面の幅を返します。

透過的なWebサポートのための追加組み込み機能を含むコマンド：

コマンド名	コメント
ADD RECORD	フォームの自動変換。マルチページフォームをサポートする
ALERT	ダイアログボックスの自動変換
CONFIRM	ダイアログボックスの自動変換
DIALOG	フォームの自動変換、マルチページフォームをサポートする
DISPLAY SELECTION	フォームの自動変換 内蔵ののWebページングメカニズム。 UserSetメカニズムはサポートしない。
MODIFY RECORD	フォームの自動変換、マルチページフォームをサポートする
MODIFY SELECTION	フォームの自動変換 内蔵のWebページングメカニズム UserSetメカニズムはサポートしない。
QUERY	標準の「クエリ」ダイアログボックスをサポートする
QUERY BY EXAMPLE	フォームの自動変換、マルチページフォームをサポートする
Request	ダイアログボックスの自動変換

行いたいことがわかっているときに使用するコマンド：

次のコマンドは、Webサーバマシン上でローカルに実行されます。

例えば、Webブラウザから選択したものの印刷処理を起動できます。ただし、印刷はWebサーバマシン上で行われます。

さらに、ユーザインタフェースコンポーネントが含まれると、そのコンポーネントはWebサーバマシンに表示されます。つまり、「**Open document("")**」と「**Open documen("文書")**」の違いです。Web サーバマシン上でダイアログボックスがクローズされるまでWebブラウザが応答を待つので、このような呼び出しは避けるべきです。それに対して、ダイアログボックスが含まれないときにこれらのルーチンを呼び出すことはまったく問題がありません。

コマンド名	コメント
Append document	ファイルダイアログボックスが起動されなければ、問題なし
BEEP	Web サーバマシン上で警告音を出す
Create document	ファイルダイアログボックスが起動されなければ、問題なし
DISPLAY RECORD	何も行わない
EXPORT DIF	ファイルダイアログボックスが起動されなければ、問題なし
EXPORT SYLK	ファイルダイアログボックスが起動されなければ、問題なし
EXPORT TEXT	ファイルダイアログボックスが起動されなければ、問題なし
IMPORT DIF	ファイルダイアログボックスが起動されなければ、問題なし
IMPORT SYLK	ファイルダイアログボックスが起動されなければ、問題なし
IMPORT TEXT	ファイルダイアログボックスが起動されなければ、問題なし
LOAD SET	ファイルダイアログボックスが起動されなければ、問題なし
LOAD VARIABLES	ファイルダイアログボックスが起動されなければ、問題なし
MESSAGE	メッセージは、Web サーバマシン上に表示される
Open document	ファイルダイアログボックスが起動されなければ、問題なし
Open external window	ウィンドウは、Web サーバマシン上でオープンする
Open resource file	ファイルダイアログボックスが起動されなければ、問題なし
Open window	ファイルダイアログボックスが起動されなければ、問題なし
PLAY	音は、4Dマシン上で発声される
PRINT FORM	ファイルダイアログボックスが起動されなければ、問題なし
PRINT LABELS	ァイルダイアログボックスが起動されなければ、問題なし
PRINT RECORD	ファイルダイアログボックスが起動されなければ、問題なし
PRINT SELECTION	ファイルダイアログボックスが起動されなければ、問題なし
QUIT 4D	リモートでWebサーバをシャットダウンできる。
SAVE SET	ファイルダイアログボックスが起動されなければ、問題なし
SAVE VARIABLES	ファイルダイアログボックスが起動されなければ、問題なし
SELECT LOG FILE	ファイルダイアログボックスが起動されなければ、問題なし
SET CHANNEL	ファイルダイアログボックスが起動されなければ、問題なし
TRACE	デバッガウィンドウは、Web サーバマシン上に表示される

「Web 接続」プロセスではサポートされないコマンド；

コマンド名	コメント
ADD DATA SEGMENT	「Web接続」プロセス内部からはこのコマンドを呼び出してはならない。このコマンドはWeb上で使用するようには設計されていない。
ADD SUBRECORD	「Web接続」プロセス内部からはこのコマンドを呼び出してはならない。このコマンドはWeb上で使用するようには設計されていない。
CHANGE ACCESS	「Web接続」プロセス内部からはこのコマンドを呼び出してはならない。このコマンドはWeb上で使用するようには設計されていない。
EDIT ACCESS	「Web接続」プロセス内部からはこのコマンドを呼び出してはならない。「パスワード」ウィンドウは4Dマシン上に表示される。ブラウザはウィンドウがクローズされるまで待つ。
GRAPH TABLE	「Web接続」プロセス内部からはこのコマンドを呼び出してはならない。このコマンドはWeb上で使用するようには設計されていない。
MODIFY SUBRECORD	「Web接続」プロセス内部からはこのコマンドを呼び出してはならない。このコマンドはWeb上で使用するようには設計されていない。
ORDER BY	プログラムによるサポートのみ。標準の「並べ替え」ダイアログボックスはWebではサポートされていない。
PRINT SETTINGS	「Web接続」プロセス内部からはこのコマンドを呼び出してはならない。「プリント」ダイアログボックスは4Dマシン上に表示される。 ブラウザはウィンドウがクローズされるまで待つ。
REPORT	「Web接続」プロセス内部からはこのコマンドを呼び出してはならない。「クイックレポート」ウィンドウは4Dマシン上に表示される。 ブラウザはウィンドウがクローズされるまで待つ。

Web サービス：HTML と JavaScript のカプセル化

はじめに：

この節を参照する前に、Webサービスに関する次の節を読んでください。

- Webサービス：概要
- Webサービス：システム設定
- Webサービス：入門編 (パート I)
- Webサービス：入門編 (パート II)
- Webサービス：「Web接続」プロセス
- Webサービス：HTMLサポート

4DアプリケーションにHTMLコードをカプセル化する方法には、次の3種類があります。

1. **SEND HTML FILE**コマンドを使用すると、ディスク上に格納されたWebページを送信できます。

2. "{anyPage.HTM}"などの4Dフォーム静的テキストオブジェクトは、HTMLドキュメント"anyPage.HTM"を4Dフォームの静的テキストオブジェクトの位置に挿入します。

3. フォーム内の任意の4Dテキスト変数は、最初の文字がASCIIコードの1の場合には、HTMLコードを4Dフォームにカプセル化できます(例：vtHTML:=Char(1)+"...HTML code...")。

2と3では、Webブラウザ側で結果として表示されるフォームは4DとHTMLオブジェクトの組み合せです。静的テキストオブジェクトを使用しているときには、ドキュメントは全体を挿入することに注意してください。テキスト変数を使用しているときには、いくつかのコードを挿入します。

SEND HTML FILEコマンドまたはフォームの静的テキストオブジェクトを使用すると、既存のHTMLドキュメントを使用するか、または、以前にプログラムで構築してからディスクに保存していたドキュメントを参照できます。フォームの中にあるテキスト変数を使用すると、メモリ上にHTMLコードを構築することができます。

4Dメソッドでの HTMLオブジェクトの構築

4Dアプリケーションでどのようにカプセル化したかには関係なく、そのオブジェクトに対するリンクを作成することで、4DメソッドにHTMLオブジェクトをリンクできます。オブジェクトのURLは、「/4DMETHOD/Method_Name」である必要があり、ここの“ Method_Name ”は、HTMLオブジェクトがクリックされたときに実行される4Dプロジェクトメソッドの名前です。HTMLオブジェクトにバインドされる4Dメソッドの例については、「Web サービス：入門編(パートII)」の節を参照してください。

重要：
■ HTMLファイルを送信する場合には、任意のリンク可能なHTMLオブジェクトに4Dメソッドをバインドできます。HTMLモードを停止する**SEND HTML FILE**("")コールを発行するPOSTアクション4D Methodを持つには、HTMLページ内にSubmitボタンを作成して、HTMLページ(4D Web Serverにサブミットされる予定のページ)のPOSTアクションを実行する必要があることに留意してください。

■ 一方、静的テキストまたはテキスト変数としてHTMLコードを4Dフォームの中にカプセル化する場合には、サブミットボタンは使用できません。4Dメソッドを参照するリンクを持つことはできますが、4Dがページを構築してくれるので、POSTアクションを指定する機会がありません。

4D変数によるHTMLオブジェクトのバインド：パート1

4D変数を使用すると、HTMLオブジェクトをバインドすることができます。

注：作業はプロセス変数で行います。

まず最初に、HTMLオブジェクトは4D変数の値を使用して初期化した値を持つことができます。

次に、Webページがサブミットして戻された後に、HTMLオブジェクトの値は4D変数の中に返ります。これを行うには、ページのHTMLソースの中で、バインドしたい4Dプロセス変数の名前と同じ名前のHTMLオブジェクトを作成します。この詳細については、後述の「4D変数によるHTMLオブジェクトのバインド：パート2」を参照してください。

HTMLオブジェクト値は4D変数値で初期化できるものなので、プログラムでHTMLオブジェクトのvalueフィールドに[VarName]を含めることでデフォルト値を提供できます。ここで、VarNameは、「Web接続」プロセスで定義された4Dプロセス変数の名前です。この名前を大カッコで囲みます。

実際、[VarName]構文を使用すると、HTMLページの任意の場所に4Dデータを挿入できます。例えば、以下のように書いた場合に、

```
<p>Welcome to [vtSiteName]! </p>
```

4D変数「vtSiteName」の値が、HTMLページ内に挿入されます。

Tips：[VarName]構文を使用したバインドは、再帰的に扱われます。変数VarNameの中にあるテキスト値に他の有効な[...]参照が含まれる場合には、4Dは、参照が見つからなくなるまでその参照を変数値に置き換えます。

以下に例を示します。

```
` 次の4Dコードは、"4D4D"をプロセス変数「vs4D」に割り当てる
vs4D:="4D4D"
` 次に、HTMLページ"AnyPage.HTM"を送信する
SEND HTML FILE("AnyPage.HTM")
```

HTMLページAnyPage.HTMのソースは、次のとおりです。

```
<HTML>
<HEAD>
   <TITLE>AnyPage</TITLE>
   <SCRIPT LANGUAGE="JavaScript"><!--

        function Is4DWebServer(){
           return (document.frm.vs4D.value=="4D4D")
        }

        function HandleButton(){
           if (Is4DWebServer()){
              alert("You are connected to 4D Web Server!")
           } else {
              alert("You are NOT connected to 4D Web Server!")
           }
        }


   //--></SCRIPT>
</HEAD>
<BODY>
<FORM action="/4DMETHOD/WWW_STD_FORM_POST" method="POST" name="frm">

<P><INPUT TYPE="hidden" NAME="vs4D" VALUE="[vs4D]"></P>

<P><A HREF="JavaScript:HandleButton()"><IMG SRC="AnyGIF.GIF" BORDER=0
ALIGN=bottom></A></P>

<P><INPUT TYPE="submit" NAME="bOK" VALUE="OK"></P>

</FORM>
</BODY>
</HTML>
```

HTMLページ(HTMLドキュメントまたは変換された4Dフォーム)を送信する前に、4Dは4D変数を参照するオブジェクトを検索するために、また、リンクのURLを再マップするためにHTMLソースコードを常に解析します(後述)。

前述のHTMLソースコードのvs4Dという名前のhidden入力オブジェクトを見てください。このオブジェクトの値はテキスト値"[vs4D]"に設定されています。HTMLファイルを送信しているプロジェクトメソッドには以前に定義された4Dプロセス変数vs4Dがあるので、4DはこのHTMLオブジェクトの値を置き換えて4D変数の値である"4D4D"に設定します。

埋め込みJavaScript関数「Is4DWebServer」は、vs4D HTMLオブジェクトの値をテストします。以下にその手順を示します。HTMLページが4Dから提供される場合には、オブジェクトの値は"4D4D"に変更されます。ただし、HTMLページが別のアプリケーション(例: MacintoshのWeb Star)から提供される場合には、オブジェクトはHTMLページ内で定義された値"[vs4D]"のままになります。つまり、Webブラウザ側にあるページ内からJavaScriptを使ってそのオブジェクトの値をテストすることで、ページが4Dから提供されるものかどうかが検出できるのです。

この最初の例は、4Dで提供されるときに他のWebサーバとの互換性を維持しながら追加の機能を提供する「インテリジェントな」HTMLページを構築する方法を示しています。

重要：ユーザがバインドできるのはプロセス変数だけです。また、バージョン6.0の初期リリースでは、4D配列でHTML SELECTオブジェクトをバインドすることはできません。一方、SELECTオブジェクトの各要素は独立した4D変数を参照できます(例: 最初の要素をV1へ、2番目の要素をV2へなど)。

4DからWebブラウザに対する方向でバインドすることは、任意のカプセル化メソッド(**SEND HTML FILE**コマンド、静的テキスト、4Dフォームのテキスト変数)で作動します。

JavaScriptカプセル化

4Dは、HTMLドキュメントに埋め込まれたJavaScriptソースコードをサポートします。ただし、6.0の初期リリースはJavaScriptの「.js」ファイルのHTMLドキュメントへの挿入(例：<SCRIPT SRC="...")はサポートしていません。

SEND HTML FILEコマンドを使用すると、HTMLソースエディタで準備したページや、4Dを使用してプログラムで構築し、ディスク上に保存していたページを送信できます。どちらの場合でも、開発者がページを全面的に制御できます。FORMマークアップでスクリプトを使用するのと同様に、ドキュメントのHEADセクションにJavaScriptスクリプトを挿入できます。前の例では、開発者がフォームに名前を付けることができたので、スクリプトはフォーム"frm"を参照します。また、FORMマークアップレベルで、フォームのサブミッションのトリガ、受け付け、拒否もできます。

4DフォームにHTMLをカプセル化する場合には、HEADセクションやFORM宣言に対する制御権を開発者は持っていません。このため、スクリプトのスコープが異なります。例えば、HTMLフォームに対してはフォームの名前でアクセスすることはできません。ただし、下記のものと上記の例とで、JavaScript関数「Is4DWebServer」を比べてみると、

```
function Is4DWebServer( ) {
   return (document.form[0].vs4D.value=="4D4D")
}
```

どちらの関数も同じことをしていますが、2番目の例はHTMLドキュメントオブジェクトのフォームプロパティを使用してforms[0]要素からオブジェクトにアクセスしています。結果として、4Dが変換されたHTMLページ(フォーム)を提供してもしなくても、開発者がその名前を知らないときでも作動します。

注：6.0の初期リリースはJavaアプレットに対して組み込みサポートを提供しません。ただし、このマニュアル作成中に、ACIのパートナーが4D環境でJavaアプレットを統合する4Dプラグインを現在開発中であるという情報を得ています。

4D変数によるHTMLオブジェクトのバインド：パート2

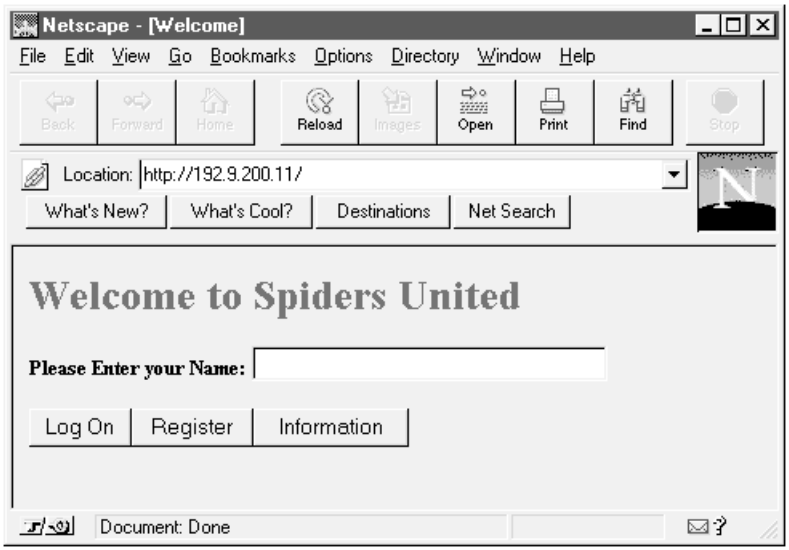
SEND HTML FILEコマンドを使用してHTMLページを送信するときに、「Webブラウザから4Dへ」の方向でHTMLオブジェクトに4D変数をバインドすることもできます。このバインドは一度HTMLページがサブミットされると、4DはHTMLオブジェクトの値を4Dプロセス変数にコピーして戻し、双方向に作動します。

警告：値を4Dプロセス変数に返すことは、SEND HTML FILEコマンドを使って送信されたHTMLページだけで可能なことです。4Dフォームでカプセル化されたHTMLでは、値を返すことはフォームに存在する実際の4Dオブジェクトだけに限られています。

次のHTMLページソースコードを参照してください。

```
<HTML>
<HEAD>
   <TITLE>Welcome</TITLE>
   <SCRIPT LANGUAGE="JavaScript"><!--

        function GetBrowserInformation(formObj){
           formObj.vtNav_appName.value = navigator.appName
           formObj.vtNav_appVersion.value = navigator.appVersion
           formObj.vtNav_appCodeName.value = navigator.appCodeName
           formObj.vtNav_userAgent.value = navigator.userAgent
           return true
        }

        function LogOn(formObj){
           if(formObj.vtUserName.value!=""){
              return true
           } else {
              alert("Enter your name, then try again.")
              return false
           }
        }
    //--></SCRIPT>
</HEAD>
<BODY>
<FORM action="/4DMETHOD/WWW_STD_FORM_POST" method="POST" name="frmWelcome"
                        onsubmit="return GetBrowserInformation(frmWelcome)">

<H1>Welcome to Spiders United</H1>

<P><B>Please Enter your Name:</B>
   <INPUT TYPE="text" NAME="vtUserName" VALUE="[vtUserName]" SIZE=30></P>
<P>
   <INPUT TYPE="submit' NAME="vsbLogOn" VALUE="Log On" onclick="return LogOn(frmWelcome)">
   <INPUT TYPE="submit' NAME="vsbRegister" VALUE="Register">
   <INPUT TYPE="submit' NAME="vsbInformation" VALUE="Information">
</P>

<P>
   <INPUT TYPE="hidden' NAME="vtNav_appName" VALUE="">
   <INPUT TYPE="hidden' NAME="vtNav_appVersion" VALUE="">
   <INPUT TYPE="hidden' NAME="vtNav_appCodeName" VALUE="">
   <INPUT TYPE="hidden' NAME="vtNav_userAgent" VALUE="">
</P>
</FORM>
</BODY>
</HTML>
```

4DがこのページをWebブラウザに送ると、次のように表示されます。



このページの主な機能は、次のようになります。

■ vsbLogOn、vsbRegister、vsbInformationという3つのサブミットボタンがあります。

■ ユーザが「Log On」ボタンをクリックすると、フォームのサブミッションは最初にJavaScript関数「LogOn」で処理されます。名前が入力されていないと、フォームは4Dへのサブミッションを行わないで、JavaScriptの警告を表示します。

■ フォームにはSubmitスクリプトの他にPOST 4D Method(GetBrowserInformation)があり、名前が「vtNav_App」で始まる4つの隠しオブジェクトに対してNavigatorプロパティをコピーします。

■ オブジェクト「vtUserName」の初期値は、[vtUserName]です。

このHTMLページを**SEND HTML FILE**コマンドを使用して送信する4Dメソッド「WWW Welcome」の内容を見てみましょう。このメソッドは、「On Web Connection」データベースメソッドから呼び出されます。

```
` 「WWW Welcome 」プロジェクトメソッド
` WWW Welcome → Boolean
` WWW Welcome → Yes = セッションを開始できるか？
C_BOOLEAN($0)
$0:=False
`ブラウザ情報を返す隠しINPUT HTMLオブジェクト
C_TEXT(vtNav_appName ; vtNav_appVersion ; vtNav_appCodeName ;vtNav_userAgent)
vtNav_appName:=""
vtNav_appVersion:=""
vtNav_appCodeName:=""
vtNav_userAgent:=""
```

```
`ユーザ名が入力されるテキストINPUT HTMLオブジェクト
C_TEXT(vtUserName)
vtUserName:=""
` HTMLサブミットボタンの値
C_STRING(31 ; vsbLogOn ; vsbRegister ; vsbInformation)
Repeat
    `サブミットボタンの値をリセットするのを忘れずに！
    vsbLogOn:=""
    vsbRegister:=""
    vsbInformation:=""
    `Webページを送信する
    SEND HTML FILE("Welcome.HTM")
    `どのサブミットボタンがクリックされたのかを検出するためにボタンの値をテストする
    Case of
        `「Log On 」ボタンがクリックされた場合
        \(vsbLogOn  #  "")
            QUERY([WWWユーザ];[WWWユーザ]ユーザ名=vtUserName)
            $0:=(Records in selection([WWWユーザ])>0)
            If ($0)
                WWW POST EVENT ("Log On" ; WWWログ情報)
                `WWW POST EVENTメソッドはデータベーステーブルに情報を保存する
            Else
                CONFIRM("このユーザ名は登録されていません。登録しますか?")
                $0:=(OK=1)
                If ($0)
                    $0:= WWW Register
                    `WWW Registerメソッドで新しいWebユーザの登録ができる
                End if
            End if
        `「Register」ボタンがクリックされた場合
        \(vsbRegister  #  "")
            $0:= WWW Register
            `「Information  」ボタンがクリックされた場合
        \(vsbInformation  #  "")
            DIALOG([ユーザインタフェース] ; "WWW情報")
    End case
Until (Not(<>vbWebServicesOn)  |  $0)
```

このメソッドの機能は、次のとおりです。

■ 4D変数「vtNav_appName」、「vtNav_appVersion」、「vtNav_appCodeName」、「vtNav_userAgent」(同じ名前のHTMLオブジェクトにバインドされている)は、JavaScriptスクリプト「GetBrowserInformation」を使用してHTMLオブジェクトに割り当てられた値を取り戻します。単純かつ直接的に、メソッドは変数を文字列として初期化してから、Webページがサブミットされた後で値を取り戻します。

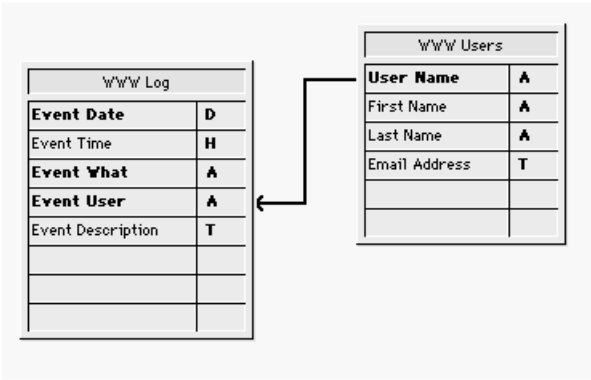
■ 4D変数「vsbLogOn」、「vsbRegister」、「vsbInformation」は、3つのサブミットボタンにバインドされます。これらの変数は、ページがブラウザに送られるたびにリセットされることに注意してください。これらのボタンのどれかでサブミットが実行されると、ブラウザはクリックされたボタンの値を4Dに返します。4D変数はそのたびにリセットされるので、空の文字列ではなくなった変数によってどのボタンがクリックされたのかがわかります。それ以外の2つの変数は空の文字列であり、これは、ブラウザが空の文字列を返したためではなく、ブラウザがそれらの変数についてなにも「言わなかった」ためです。その結果、4Dは変数をそのままにしていたのです。このため、ページがブラウザに送信されるたびに、これらの変数を空の文字列でリセットする必要があるのです。

これが、複数のサブミットボタンがWebページに存在するときにどのサブミットボタンがクリックされたのかを見分ける方法です。4Dフォーム内の4Dボタンは、数値変数です。ただし、HTMLのときは、すべてのオブジェクトはテキストオブジェクトになります。4Dに返されるときに、ボタンがクリックされたかどうかのテストはボタンにバインドされる4D変数のテキスト値のテストで構成されます。

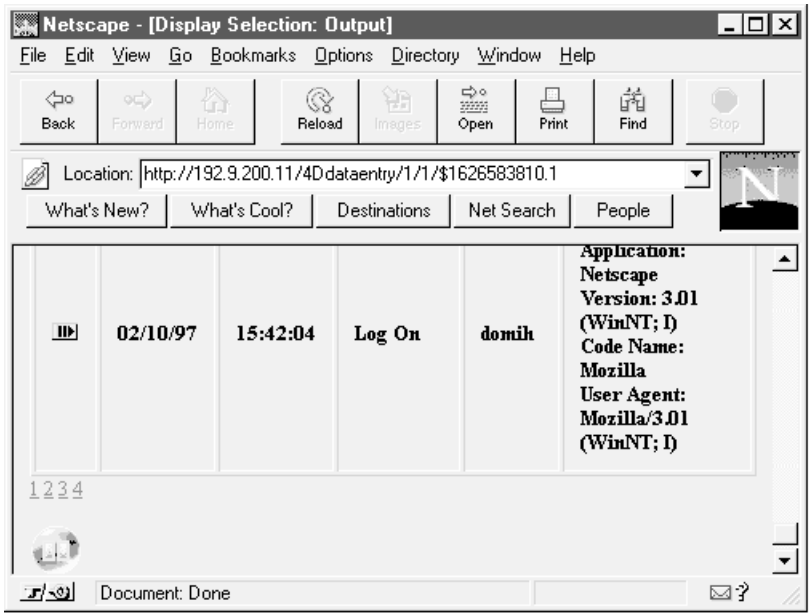
4D変数をSELECTオブジェクトでバインドする場合には、テキスト変数もバインドします。4Dでは、ドロップダウンリストのどの要素が選択されたのかをテストするには、4D配列の数値をテストします。HTMLでは、これはHTMLオブジェクトにバインドされた4D変数に返される選択された項目の値になります。

4D変数でどのオブジェクトをバインドするかに関係なく、返される値はテキストタイプであり、文字列またはテキストの4Dプロセス変数をバインドすることになります。

この例で注目すべき点は、ブラウザについての情報を取得した後、Webとデータベースの機能をもう一度組み合せることで、これらの値を4Dテーブルに格納できるということです。これが、(リストのない)「WWW POST EVENT」プロジェクトメソッドが行うことです。このメソッドは「イベントをポストする」わけではなく、Webセッション情報を次に示すテーブルに保存するのです。



情報をテーブルに保存した後、他のプロジェクトメソッドを使用してその情報をWebユーザに送り戻すことができます。これを行うには、単に**QUERY**コマンドを使用して適切な情報を検索し、**DISPLAY SELECTION**コマンドを使用して[WWW Log]レコードを表示します。次の図は、Webサイトに登録されたユーザが利用できるログ情報を示しています。



この例で示すバインド機能を使用し、開発者が提示できる情報すべてを組み合せるか、またはHTMLダイアログや4Dフォーム経由でユーザから情報を収集すると、データベースのWebサイトに非常に役に立つ管理機能を追加することができます。

4D変数によるHTMLオブジェクトのバインド：パート3

Webサービス、HTMLサポートセクションにあるように、Webページとして4Dフォームを使用すると、4Dは静止ピクチャに重なる非表示的ボタンを使ってサーバ側のイメージマッピングを提供します。

SEND HTML FILEコマンドを使用してHTMLドキュメントを送信する場合には、4D変数をイメージマップのHTMLオブジェクト(INPUT TYPE="IMAGE")でバインドすることで情報を取り出すことができます。例えば、「bImageMap」という名前のイメージマップのHTMLオブジェクトを作成することができます。ブラウザ側にあるイメージをクリックするたびに、クリック位置でのサブミットが4D Webサーバに送り返されます。クリックの座標(イメージの左上端から相対的に表される)を取り出すには、4Dプロセス変数bImageMapと、bImageMap_X変数およびbImageMap_Y変数をバインドする必要があるだけです。ここで、この変数はクリックの縦方向の座標と横方向の座標を(テキストとして)返します。

HTMLページでは、次のように作成できます。

```
<p> <INPUT TYPE="image" SRC="myImage.GIF" NAME="bWorldMap" BORDER= >< /P>
```

HTMLページを送信する4Dメソッドでは、次のように作成できます。

```
bImageMap:=""
bImageMap_X:=""
bImageMap_Y:=""
SEND HTML FILE("ThisPage.HTM")
```

次に、POSTアクション4Dメソッドまたはカレントメソッドで、POSTアクションメソッドが**SEND HTML FILE**("")呼び出しを発行した後に、次のようにクリックの座標位置を取り出します。

```
$vlX:=Num(bImageMap_X) ` 数値形式で横方向の座標を得る
$vlY:=Num(bImageMap_Y) ` 数値形式で縦方向の座標を得る
If (($vlX#0)&($vlY#0))
   `取得した座標に従って処理をする
End if
```

ファイル参照とURL

データベースのコンテキストIDとサブコンテキストIDの保守を確実にするために、4Dはファイル参照とURLを自動的に再マップします。例えば、4DはすべてのIMG参照とHREF参照をローカルファイルに再マップします。

テキスト変数を使用して独自のHTMLコードを4Dフォームに挿入する場合には、4D再マッピング構文に従う必要があります。

ローカルのGIFファイルは、"/4DPict/_/GIF_file_pathname/$-2"として再マップされ、ここで「GIF_file_pathname」はファイルが存在するボリュームのルートに相対的なGIFファイルの完全なHTMLパス名になります。

例：

次の4Dメソッドは、引数として受け取ったパス名に対する再マップされた参照を返します。

```
` 「WWW Local GIF URL」プロジェクトメソッド
` WWW Local GIF URL Project (テキスト)
` WWW Local GIF URL (本来のパス名) → ローカルGIFファイルのURL
C_TEXT($0 ; $1)
$0:="/4DPict/_/"+ HTML Pathname ($1)+"/$-2"
```

注：「HTML Pathname」メソッドの詳細については、Mac to ISO関数の例を参照してください。

次に、テキスト変数を使ってHTMLコードを4Dフォームに挿入するときに、次のように指定できます。

```
vtHTML:=Char(1)+"<P><IMG SRC="+Char(34)+ WWW Local GIF
URL("F:\ThisImage.HTM"+Char(34)+" ALIGN=MIDDLE></P>"+Char(13)
```

これによって、4D変数vtHTMLの位置でGIFドキュメントが4Dフォームに挿入されます。

重要：カスタムHTMLコードを4Dフォームに挿入するには、単にこの種のコードを作成します。SEND HTML FILEコマンドを使用してHTMLページを送信するだけ、またはADD RECORDコマンドなどのコマンドを使用する場合には、4Dが透過的にHTMLを変換し、再マップすることに留意してください。

再マッピングは、以下のプロトコルを持つリンクを変更しません。

- http:
- ftp:
- mailto:
- news:
- gopher:
- javascript:
- telnet:

重要：4th Dimensonのバージョン6.0の初期のものはFrameをサポートしません。FRAMEマークアップには、4Dがまだサポートしていない再帰的なHTMLファイル参照があります。

次の節では、「ルーチン」エディタの「Web Server」テーマ内にあるWebサーバコマンドについて説明します。

START WEB SERVER **STOP WEB SERVER** **SEND HTML FILE**
SET WEB DISPLAY LIMITS **SET WEB TIMEOUT** **SET HTML ROOT**
CHANGE WEB LICENSE

START WEB SERVER

START WEB SERVER

引数 **タイプ** **説明**

このコマンドには、引数はありません。

START WEB SERVERコマンドは、内蔵の4th Dimension Webサーバ機能を使用してイントラネットネットワークまたはインターネット上でデータベースのサービスを開始します。

Webサーバが正常に起動された場合には、システム変数OKに1が設定され、そうでなければOKは0(ゼロ)が設定されます。例えば、TCP/IPネットワークコンポーネントが指定されていない場合には、システム変数OKは 0に設定されます。

参照：STOP WEB SERVER

システム変数とシステムセット

Webサーバが正常に開始された場合はOKに1、そうでない場合はOKに 0が設定されます。

STOP WEB SERVER

STOP WEB SERVER

引数	タイプ	説明

このコマンドには、引数はありません。

STOP WEB SERVERコマンドは、データベースのWebサーバとしてのサービスを停止します。データベースがWebサイトとしてサービスされていた場合には、すべてのWeb接続は停止され、すべてのWebプロセスが終了します。

データベースがWebサイトとしてサービスされていなかった場合には、このコマンドは何も行いません。

参照：START WEB SERVER

SET WEB TIMEOUT

SET WEB TIMEOUT (タイムアウト)

引数	タイプ		説明
タイムアウト	数値	→	秒単位で表されるWeb接続タイムアウト

SET WEB TIMEOUTコマンドは、「Web接続」プロセスのタイムアウトを設定します。デフォルトのタイムアウトは 5分です。

秒単位の新しいタイムアウトである＜タイムアウト＞引数を渡すことで、この遅延を短くしたり、長くすることができます。

このコマンドは即座に有効になり、そのスコープは作業中のセッションになります。すべての「Web 接続」プロセスが影響を受けます。

SET HTML ROOT

SET HTML ROOT (パス名HTML)

引数	タイプ		説明
パス名HTML	数値	→	HTMLファイル用のデフォルトディレクトリに対するHTMLパス名

SET HTML ROOTコマンドは、**SEND HTML FILE**コマンドの引数として渡すHTMLファイルを4Dが検索するデフォルトのディレクトリやフォルダを変更します。

デフォルトでは、4Dはデータベースのストラクチャファイルを含むディレクトリでHTMLドキュメントを検索します。

指定するパス名はHTMLパス名である必要があり、そのパス名はプラットフォームに関係なく、スラッシュ("/")文字で区切られたディレクトリまたはフォルダ名です。HTMLパス名の詳細については、書店にあるHTMLに関する書籍のランゲージリファレンス部分を参照してください。

無効なパス名を指定した場合には、OSのファイルマネージャエラーが生成されます。**ON ERR CALL**メソッドでこのエラーを途中で処理することができます。エラーメソッド内部から警告やメッセージを表示する場合には、ブラウザ側に表示されます。

▼ **SEND HTML FILE**コマンドの例を参照してください。

参照：ON ERR CALL

エラー処理

無効なパス名を指定すると、OSのファイルマネージャエラーが生成されます。**ON ERR CALL**メソッドでこのエラーを途中で処理することができます。

SET WEB DISPLAY LIMIT

SET WEB DISPLAY LIMIT (レコード数 { ; ページ数 { ; ピクチャ参照番号)

引数	タイプ		説明
レコード数	数値	→	各HTMLページに表示する最大レコード数
ページ数	数値	→	各HTMLページの下部にあるページ参照の最大数
ピクチャ参照番号	数値	→	フルページレコードボタン用のピクチャ参照番号

SET WEB DISPLAY LIMITコマンドは、ユーザが**DISPLAY SELECTION**コマンドまたは**MODIFY SELECTION**コマンドを呼び出すと、Webブラウザ側にあるレコードの選択を4th Dimensionが表示する方法を変更します。

4th Dimensionまたは4D Clientを使用してレコードの選択を表示するときに、プログラムは選択するすべてのレコードをロードするわけではなく、一度にウィンドウに表示できるレコードを(ディスクから)ロードするだけです。そうすることによって、ユーザが作成したのが数千レコードの選択であったとしても、それらを表示することは非常に高速になります。そして、ウィンドウのスクロールやサイズ変更をすると、4Dはそれに対応してレコードをロードします。

Webでは、4Dはページに表示されたレコードの選択を分割します。ページング体系のない数千レコードの選択は、結果としてインターネット上またはイントラネット上で1つのWebページだけに数千のレコードを表示することになります。また、これらのレコードのダウンロードにはかなりの時間を要し、Webブラウザのメモリ不足につながります。

デフォルトでは、4th Dimensionは選択の最初の20レコードを表示し、また、各HTMLページの最後の20リンクを最初の20ページの選択に含みます。これは、デフォルトで、選択ページごとの最後にあるページリンク上をクリックすることで、最初の400レコードの選択をブラウズできることを意味します。このページングシステムはコーディングに対して透過的であることに注意してください。すべては、**DISPLAY SELECTION**コマンドや**MODIFY SELECTION**コマンドへの呼び出しの内部で発生します。

SET WEB DISPLAY LIMITコマンドでこれらの設定を変更できます。引数<レコード数>には、選択ページごとに表示したいレコードの最大数を指定します。引数<ページ数>には、各選択ページの最後に必要な選択ページリンクの最大数を指定します。

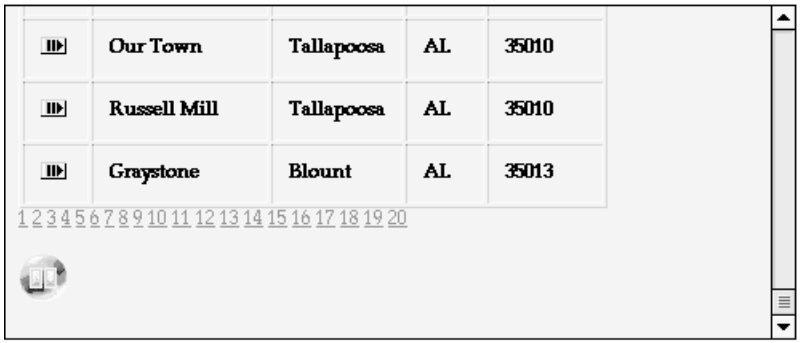
例えば、10,000レコードの選択があり、1つの表示選択でそれらすべてをブラウズしたい場合には、<レコード数>=100、<ページ数>=100を渡すことができます。ただし、このデータがネットワークまたはインターネットの上を通っていくことに注意してください。インターネットの場合は、表示選択設定を変更するときに、スピード要因を考慮する必要があります。

さらに、**SET WEB DISPLAY LIMIT**コマンドはフルページレコードボタンのデフォルトアイコンをオプションで変更できます。引数<ピクチャ参照番号>に、データベースのピクチャライブラリに格納されている、新しいアイコンとして使用したいピクチャのピクチャ参照番号を指定します。

SET WEB DISPLAY LIMITコマンドは、その後の**DISPLAY SELECTION**コマンドまたは**MODIFY SELECTION**コマンドへの呼び出しだけに影響を与え、そのスコープは現在のプロセスでローカルです。

例：

次の例では、**DISPLAY SELECTION**コマンドまたは**MODIFY SELECTION**コマンドが[郵便番号テーブル用に発行されます。デフォルトで、4Dは以下に示すようにWebブラウザ側にレコードを表示します。



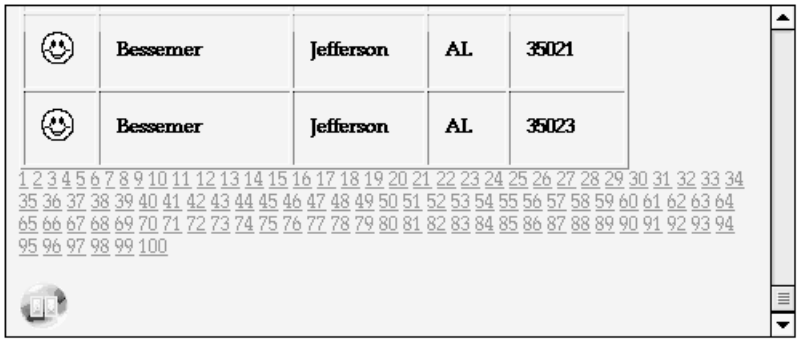
最初の400レコードがブラウズできることに注意してください。

次のピクチャがデータベースのピクチャライブラリに追加された場合、

および、選択を表示するプロジェクトメソッドがここで示す**SET WEB DISPLAY LIMIT**コマンドの呼び出しを実行する場合、**DISPLAY SELECTION**コマンドまたは**MODIFY SELECTION**コマンドへの呼び出しの前に以下を行います。

SET WEB DISPLAY LIMIT (50 ; 100 ; 17877)

次に、Webブラウザ側の選択が次のように表示されて終ります。



これで選択の最初の50,000レコードをブラウズできるようになります。

参照：DISPLAY SELECTION, MODIFY SELECTION

SEND HTML FILE

SEND HTML FILE (HTMLファイル)

引数	タイプ		説明
HTMLファイル	文字列	→	HTMLファイルへのHTMLのパス名、または **SEND HTML FILE**を終了させるための空の文字列

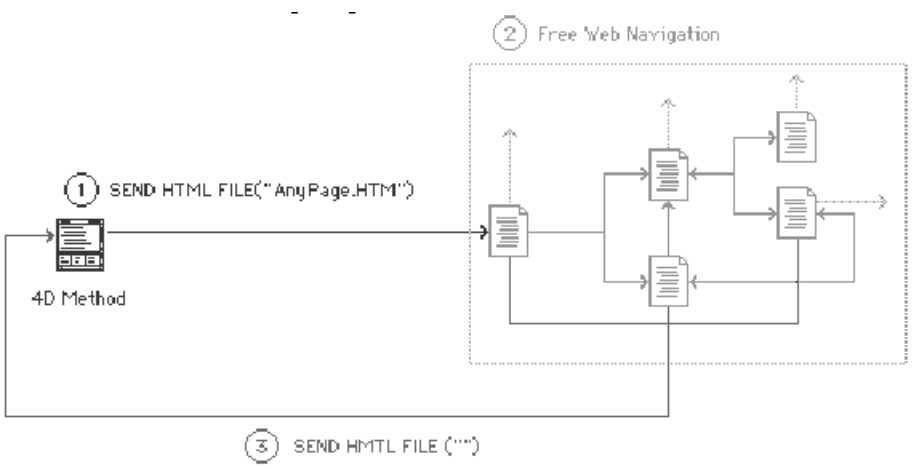
SEND HTML FILEコマンドは、引数<HTMLファイル>で渡されたパス名のHTMLドキュメントに格納されているWebページをWebブラウザに送信します。

重要：4th Dimensionは、HTMLドキュメントが「ISO Latin-1」で符号化（エンコード）されていることを想定しています。

4th Dimensionは、デフォルトで、**SET HTML ROOT**コマンドを使用して別のディレクトリやフォルダにHTMLドキュメントのデフォルト位置を設定しない限り、データベースのストラクチャファイルがあるディレクトリやフォルダの内部で対象のHTMLドキュメントを検索します。

無効なHTMLパス名を指定すると、4Dはメッセージ“ リクエストされたHTMLページが見つかりません ”をWebブラウザに送信します。

もう1つの構文**SEND HTML FILE**("")は、引数<HTMLファイル>で空の文字列を渡すものですが、**SEND HTML FILE**コマンドへの呼び出しを終了させます。**SEND HTML FILE**コマンドは、HTMLモードで初期化されたものです。これを次の図で示します。



1. 4Dメソッド(プロジェクト、オブジェクトまたはデータベース)は、ドキュメントをブラウザに送信することで**SEND HTML FILE**コマンドへの呼び出しを発行します。

2. ブラウザに送られた最初のWebページには、他のWebページへのHTMLリンクがあるか、またはそれ自体で他のWebページに送信するために**SEND HTML FILE**コマンドを呼び出す4Dメソッドを参照することができます。これらの他のページには、他のページへのアクセスをするための4Dメソッドを参照するなどのリンクがあるはずです。Webページをナビゲートする間に、Backボタンなどのブラウザのナビゲーション制御を使用することもできます。

3. 任意のWebページは、**SEND HTML FILE**("")呼び出しを発行する4Dメソッドへの参照を含めることができます。この呼び出しはすべてのものを初期化した**SEND HTML FILE**呼び出しを終了し、自由なWebナビゲーションを本来起動した4Dメソッドの実行をさらに行って、元に戻ります。

注：「Web接続」プロセスではないプロセスからSEND HTML FILEコマンドを呼び出しても、コマンドは何も行わず、エラーも返りません。呼び出しが無効になるだけです。

例：

1. データベースストラクチャファイルを含むディレクトリには、"HomePage.HTM"というHTMLドキュメントがあります。これは、データベースに接続するときにデータベースのデフォルトのメニューバー番号1の代わりに表示したいWebページです。このWebページを表示するには、アプリケーションの「On Web Connection」データベースメソッドで次のように指定します：

```
` 「On Web Connection」データベースメソッド
SEND HTML FILE ("HomePage.HTM")
```

2. フォルダデータベースフォルダは、次のように構成されています：

```
..\Documents\CurrentWork\Databases\MyDB.4DB
..\Documents\CurrentWork\Databases\MyDB.RSR
..\Documents\CurrentWork\Databases\MyDB.4DD
..\Documents\CurrentWork\Databases\WebStuff\HTM\HomePage.HTM
```

Webページ"HomePage.HTM"は、次の方法で送信できます：

```
SEND HTML FILE ("WebStuff/HTM/HomePage.HTM")
```

または、次の方法でも送信できます：

```
SET HTML ROOT ("WebStuff/HTM/")
SEND HTML FILE ("HomePage.HTM")
```

3. 4D Webセッションの間、4Dフォームを使用してレコードを追加しています。このフォームには「bヘルプ」ボタンがあり、そのオブジェクトメソッドは次のとおりです：

```
` 「bヘルプ」ボタンのオブジェクトメソッド
SEND HTML FILE ("Help.HTM")
```

Help.HTMドキュメントから開始して、ユーザはWebサイト用のデータベースHelpシステムを実装する多数のHTMLページを自由にナビゲートすることができます。各ページで、「Done」というサブミットボタンがあり、これによってデータ入力に戻ることができます。これを行うには、HTMLドキュメントそれぞれにこのサブミットボタンの定義が含まれている必要があります。

```
<!--「bDone」サブミットボタン→
<P><INPUT TYPE="submit" NAME="bDone" VALUE="Done"></P>
```

上記と同じ処理をできるフォームポストアクションの設定は、次のようになります。

```
<!--サブミットボタンがクリックされたら、「4D htm_Help_Done」メソッドを実行する→
<FORM action="/4DMETHOD/htm_Help_Done" method="POST">
```

4D側では、プロジェクトメソッド「htm_Help_Done」は「bヘルプ」ボタンで起動された**SEND HTML FILE**コマンドを終了させます：

```
` 「htm_Help_Done」プロジェクトメソッド
SEND HTML FILE ("")
```

「bヘルプ」ボタンのオブジェクトメソッドでの**SEND HTML FILE**コマンドへの呼び出しは、メソッドの最後の行です。メソッドが完了したときには、データ入力に戻ります。

CHANGE WEB LICENSE

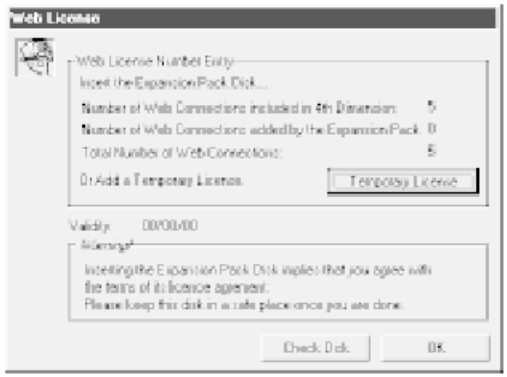
CHANGE WEB LICENSE

引数　　タイプ　　説明

このコマンドには、引数はありません。

CHANGE WEB LICENSEコマンドは、「Webライセンス」ダイアログボックスを表示します。このダイアログボックスを使って、ユーザは内蔵の4th Dimension Webサーバにライセンスを追加したり、または4th Dimension Webサーバからライセンスを削除することができます。

Windows上の「Webライセンス」ダイアログボックス：



Macintosh上の「Webライセンス」ダイアログボックス：

このダイアログボックスは、元々「デザイン」モード内でしか利用することができませんでした。しかし、この**CHANGE WEB LICENSE**コマンドを使用することにより、「ユーザ」モードまたは「カスタム」モードで「Webライセンス」ダイアログボックスを表示できるようになりました。

注：「デザイン」モードの「データベースプロパティ」ダイアログボックス内にある「ライセンス」ボタンをクリックすることにより、「Webライセンス」ダイアログボックスを表示することができます。

Tips：CHANGE WEB LICENSEコマンドは、顧客に配布されるコンパイルして組み込んだ4Dアプリケーションにwebライセンスを拡張することができる便利な方法です。4Dのデベロッパまたは情報システム部の管理者が、4Dアプリケーションを配付するためにこのコマンドを使用します。そして、そのアプリケーションのアップデート版を毎回送付することなく、ユーザにそのWebライセンスを拡張させます。

例：

独自のシステム定義または「環境設定」ダイアログボックス内に、下記のメソッドを持つボタンを含みます：

```
`「bWebライセンス」ボタンのオブジェクトメソッド
CHANGE WEB LICENSE
```

これを実行すると、データベース自身を変更することなく、ユーザは4D Webサーバに同時に接続できるユーザ数をインクリメント（増分）することができます。

ウインドウコマンド　62

この章では、「ルーチン」エディタの「Windows」テーマ内にあるストラクチャアクセスコマンドについて説明します。この章のコマンドは、ウインドウの管理を行なうためのものです。ウインドウの管理には、カスタムウインドウのオープンとクローズ、および画面サイズの確保とウインドウタイトルの変更が含まれます。

CLOSE WINDOW
DRAG WINDOW
ERASE WINDOW
Find window
Frontmost window
GET WINDOW RECT
Get window title
HIDE WINDOW
MAXIMIZE WINDOW
MINIMIZE WINDOW
Next window
Open window
Open external window
REDRAW WINDOW
SET WINDOW RECT
SET WINDOW TITLE
SHOW WINDOW
WINDOW LIST
Window kind
Window process

ウインドウについて

ウインドウは、ユーザに対して情報を表示するのに使用します。ウインドウには、データの入力、データの表示、メッセージやダイアログでのユーザに対する告知の3つの機能があります。

最低でも1つのウインドウが開かれています。これは、タイトルバーとサイズボックスを持った標準ウインドウです。ユーザにウインドウよりも大きいフォームでスクロールさせる必要のある場合は、スクロールバーがそれに追加されます。また、ウインドウにコントロールメニュー（Macintosh版では、クローズ）ボックスを持たせることもできます。

「ユーザ」モードにおいて、ウインドウはレコードリスト(出力フォーム)、またはデータ入力画面(入力フォーム)のいずれかを表示します。「カスタム」モードにおいてウインドウは、スプラッシュ画面(カスタムグラフィック)を表示します。スプラッシュ画面はフォームを表示するコマンドでデータ表示画面に置き換えられます。コマンドの処理が終了すると、再びスプラッシュ画面が表示されます。

カスタムウインドウは、**Open window**関数で開くことができます。カスタムウインドウには、Windows（Macintosh）のさまざまな形式のウインドウを利用することができます。不要になったウインドウは、**CLOSE WINDOW**コマンドで閉じます。

コマンドの中には、自らウインドウを開くものがあります。**GRAPH TABLE**コマンド、**REPORT**コマンド、**PRINT LABEL**コマンドなどは、自ら最前面にウインドウを開きます。

新規プロセスを開始したり、プロセスメソッドの冒頭で任意のウインドウを開かないと、4th Dimensionは任意フォームが表示されるとすぐにデフォルトのウインドウを自動的に開きます。

ウインドウのタイプ

下記のあらかじめ定義された定数の1つを使って、**Open window**関数で開くウインドウのタイプを指定することができます。

定数	タイプ	値	フローティングウインドウ
Plain window	倍長整数	8	不可
Plain no zoom box window	倍長整数	0	不可
Plain fixed size window	倍長整数	4	不可
Modal dialog box	倍長整数	1	不可
Alternate dialog box	倍長整数	3	可
Movable dialog box	倍長整数	5	可
Plain dialog box	倍長整数	2	可
Palette window	倍長整数	720	可
Round corner window	倍長整数	16	不可

フローティングウインドウ：

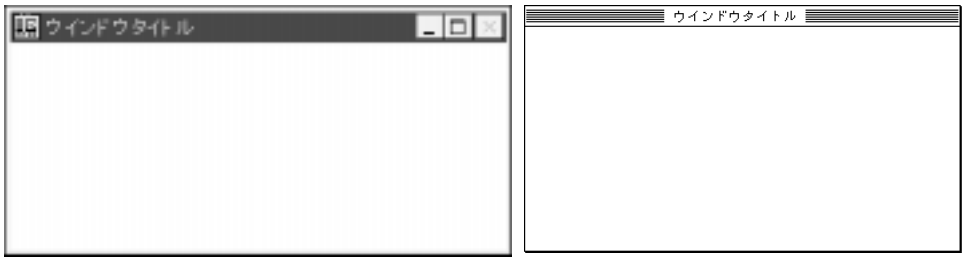
Open window関数に上記の定数の1つを渡すと、一般的なウインドウを開きます。フローティングウインドウを開くには、**Open window**関数に負数を渡します。

次の図は、各ウインドウタイプを示しています。左側がWindows、右側がMacintoshのウインドウです。

タイプ 8のウインドウ（標準）

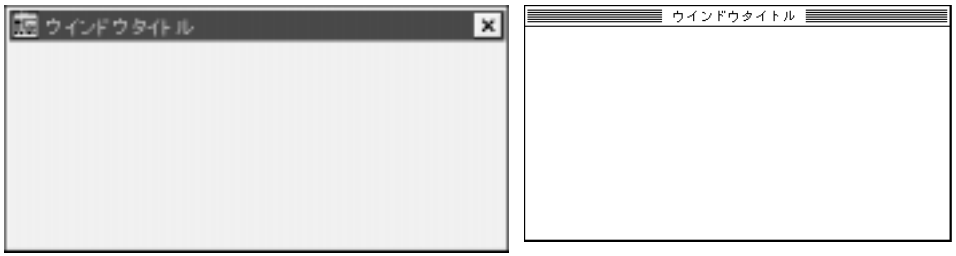
- **タイトル付加：**可
- **「閉じる」ボタンまたはクローズボックス付加：**可
- **サイズ変更：**可
- **「最大化」/「最小化」ボタンまたは「ズーム」ボックス付加：**可
- **スクロールバーの使用：**可
- **使用用途：DISPLAY SELECTION**コマンド、**MODIFY SELECTION**コマンド等

タイプ 0のウインドウ（「ズーム」ボックスなし）



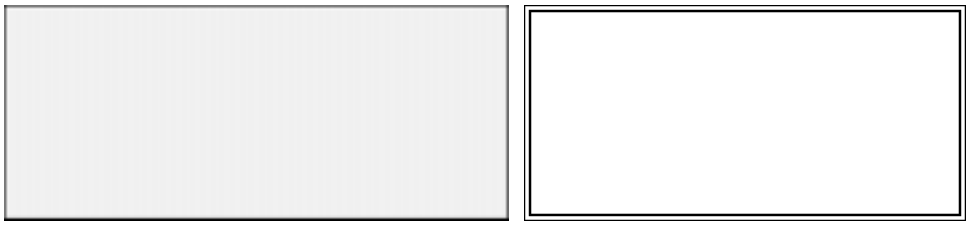
- **タイトル付加：**可
- **「閉じる」ボタンまたはクローズボックス付加：**可
- **サイズ変更：**可
- **「最大化」/「最小化」ボタンまたは「ズーム」ボックス付加：**Macintosh上では、不可
- **スクロールバーの使用：**可
- **使用用途：DISPLAY SELECTION**コマンド、**MODIFY SELECTION**コマンド等

タイプ 4のウインドウ（固定サイズ）



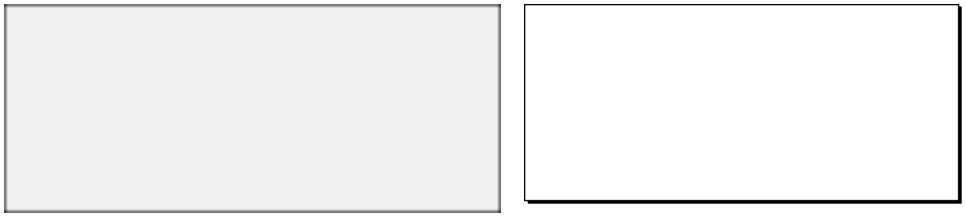
- **タイトル付加：**可
- **「閉じる」ボタンまたはクローズボックス付加：**可
- **サイズ変更：**Macintosh上では、不可
- **「最大化」/「最小化」ボタンまたは「ズーム」ボックス付加：**不可
- **スクロールバーの使用：**可／不可
- **使用用途：ADD RECORD** (... ; ...*)または同等の機能

タイプ 1のウインドウ（モーダルダイアログボックス）



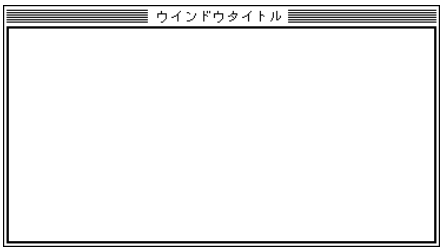
■ **タイトル付加：**不可
■ **「閉じる」ボタンまたはクローズボックス付加：**不可
■ **サイズ変更：**不可
■ **「最大化」/「最小化」ボタンまたは「ズーム」ボックス付加：**不可
■ **スクロールバーの使用：**不可
■ **使用用途：DIALOG**コマンド、**ADD RECORD** (... ; ...*)または同等の機能
■ **このタイプのウインドウは、モーダル（形式）ウインドウです**

タイプ 3のウインドウ（予備のダイアログボックス）



■ **タイトル付加：**不可
■ **「閉じる」ボタンまたはクローズボックス付加：**不可
■ **サイズ変更：**不可
■ **「最大化」/「最小化」ボタンまたは「ズーム」ボックス付加：**不可
■ **スクロールバーの使用：**不可
■ **使用用途：DIALOG**コマンド、**ADD RECORD** (... ; ...*)または同等の機能
■ **フローティングウインドウとして使用されない場合、このタイプのウインドウはモーダル（形式）ウインドウ**

タイプ 5のウインドウ（移動可能なダイアログボックス）



■ **タイトル付加：**可

■ **「閉じる」ボタンまたはクローズボックス付加：**不可

■ **サイズ変更：**不可

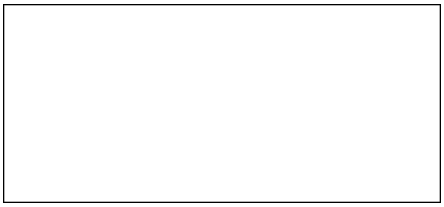
■ **「最大化」/「最小化」ボタンまたは「ズーム」ボックス付加：**不可

■ **スクロールバーの使用：**不可

■ **使用用途：DIALOG**コマンド、**ADD RECORD** (... ; ...*)または同等の機能

■ **このタイプのウインドウはモーダル（形式）ウインドウですが、フローティングウインドウとして移動したり使用することが可能**

タイプ 2のウインドウ（標準のダイアログボックス）



■ **タイトル付加：**不可

■ **「閉じる」ボタンまたはクローズボックス付加：**不可

■ **サイズ変更：**不可

■ **「最大化」/「最小化」ボタンまたは「ズーム」ボックス付加：**不可

■ **スクロールバーの使用：**不可

■ **使用用途：DIALOG**コマンド、**ADD RECORD** (... ; ...*)または同等の機能、スプラッシュスクリーン

■ **フローティングウインドウとして使用されない場合、このタイプのウインドウはモーダル（形式）ウインドウです**

タイプ （720 {+1} [+2} {+4} {+8}）のパレットウインドウ

Open window関数を呼び出すと、パレットウインドウの下記の定数を1つまたは複数を追加して、そのウインドウのさまざまな機能を取得することができます。

定数	タイプ	値
Has zoom box	倍長整数	8
Has grow box	倍長整数	4
Has window title	倍長整数	2
Has highlight	倍長整数	1

■ **タイトル付加：**可、「**Has window title**」定数が指定された場合

■ **「閉じる」ボタンまたはクローズボックス付加：**可

■ **サイズ変更：**可、「**Has grow box**」定数が指定された場合

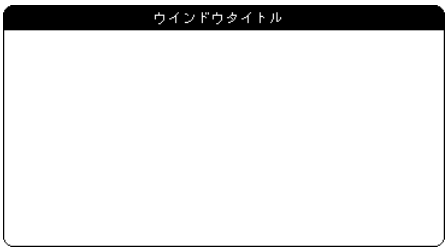
■ **「最大化」/「最小化」ボタンまたは「ズーム」ボックス付加：**可、「**Has zoom box**」定数が指定された場合

■ **スクロールバーの使用：**可、「**Has grow box**」定数が指定された場合

■ **使用用途：DIALOG**コマンド、**DISPLAY SELECTION**コマンド（データ入力ではない）

タイプ 16のウインドウ（角の丸いウインドウ）

Open window



- **タイトル付加：**可
- **「閉じる」ボタンまたはクローズボックス付加：**可
- **サイズ変更：**Macintosh上では、不可
- **「最大化」/「最小化」ボタンまたは「ズーム」ボックス付加：**不可
- **スクロールバーの使用：**Macintosh上では、不可
- **使用用途：**滅多に使用しない

参照：Open exteenal window、Open window

Open window

Open window (左;上;右;下;{タイプ};{タイトル};{コントロールメニュー（Macintosh版では、クローズ）ボックス}) → {ウインドウ参照番号}

引数	タイプ		説明
左	数値	→	ウインドウ左端の画面上の位置(ピクセル)
上	数値	→	ウインドウ上端の画面上の位置(ピクセル)
右	数値	→	ウインドウ右端の画面上の位置(ピクセル)
下	数値	→	ウインドウ下端の画面上の位置(ピクセル)
タイプ	数値	→	ウインドウのタイプ
タイトル	文字列	→	ウインドウのタイトル
コントロールメニューボックス(Macintosh版では(クローズボックス)	文字列	→	ボックスをダブルクリックしたときに呼び出されるメソッド
関数の返す値	参照番号	←	ウインドウ参照番号

Open window関数は、最初の4つの引数で指定した大きさの新しいウインドウを開きます。

■ ＜左＞は、画面の左端からウインドウの内側の左端までの距離をピクセル数で示します。

■ ＜上＞は、画面の上端からウインドウの内側の上端までの距離をピクセル数で示します。

■ ＜右＞は、画面の左端からウインドウの内側の右端までの距離をピクセル数で示します。

■ ＜下＞は、画面の上端からウインドウの内側の下端までの距離をピクセル数で示します。

＜右＞と＜下＞の両方に -1を渡す場合、下記の状態のもとではウインドウを自動的にサイズ変更するように4th Dimensionに指示します。

■ 任意のフォームをデザインして、「デザイン」モードの「フォームプロパティ」ウインドウでそのフォームの「サイズオプション」を設定する。

■ **Open window**関数を呼び出す前に、オプション引数＜*＞を持つ**INPUT FORM**コマンドを使ってそのフォームを選択する。

重要：このウインドウの自動サイズオプションは、表示されるフォーム用のINPUT FORMコマンドを呼び出し、そしてそのINPUT FORMコマンドにオプション引数＜*＞が受け渡された場合にのみ機能します。

■ 引数＜タイプ＞はオプションで、各種の表示するウインドウのタイプを指定します。ウインドウのタイプは、前節に示したウインドウに対応します。ウインドウのタイプが拒否された場合は、作成されるウインドウはフローティングウインドウです。ウインドウのタイプが指定されない場合には、タイプ1がデフォルトで用いられます。

■ 引数＜タイトル＞は、オプションです。これで、表示するウインドウのタイトルを設定します。

＜タイトル＞に 空の文字列（""）を渡す場合は、「デザイン」モードの「フォームプロパティ」ウインドウ内にある「ウインドウタイトル」を表示されるフォームに設定するように4th Dimensionに指示します。

重要：デフォルトのフォームタイトルは、表示されるフォーム用のINPUT FORMコマンドを呼び出し、そしてそのINPUT FORMコマンドにオプション引数＜*＞が受け渡された場合にのみウインドウにセットしされます。

■ 引数＜コントロールメニュー(クローズ)ボックス＞はオプションで、ウインドウにコントロールメニュー(クローズ)ボックス用のメソッドを設定します。＜コントロールメニュー(クローズ)ボックス＞を指定すると、コントロールメニュー(クローズ)ボックスがウインドウに追加されます。ユーザがウインドウ上のクローズボックスをクリックすると、＜コントロールメニュー(クローズ)ボックス＞に受け渡されるメソッドを呼び出します。

Tips：On Close Box**イベントが生じる際にウインドウに表示されるフォームのフォームオブジェクトの中からウインドウのクローズ処理を管理することもできます。これに関する詳細は、Form event関数の節を参照してください。**

1つのプロセス内で複数のウインドウが開いている場合は、最後のウインドウ(最前面のウインドウ)がそのプロセスにおけるアクティブウインドウになります。アクティブウインドウの情報のみを修正することができます。ユーザのキー入力によって、アクティブウインドウは常に最前面に配置することができます。

フォームは、ウインドウ内部に表示されます。**MESSAGE**コマンドのテキストもウインドウ内部に表示することができます。

ウインドウの大きさを画面の大きさに依存いないようにするには、**Screen height**関数と**Screen width**関数を使用してウインドウの左上端と右下端の位置を計算することができます。この技法の関する詳細は、次ページの例を参照してください。

例：

次のコーディングは“ *ウインドウ中央表示* ”というプロジェクトメソッドで、画面の中央にウインドウを開きます。2個、3個または4個の引数を受け取り、その数によって処理を振り分ける点に注目してください：

```
` プロジェクトメソッド：ウインドウ中央表示
` $1－ウインドウの幅
` $2－ウインドウの高さ
` $3－ウインドウのタイプ (オプション)
` $4－ウインドウのタイトル (オプション)
$S幅:=Screen width / 2                    ` 画面の中心を求める (幅)
$S高さ:=(Screen height / 2) -10           ` 画面の中心を求める (高さ)
$W高さ:=$1 / 2                            ` ウインドウの高さの半分を求める
$W幅:=$2 / 2                              ` ウインドウの幅の半分を求める
Case of
  \ (Count parameters=2)
    OPEN WINDOW ($S幅-$W幅;$S高さ-$W高さ;$S幅+$W幅;$S高さ+$W高さ)
  \ (Count parameters=3)
    OPEN WINDOW ($S幅-$W幅;$S高さ-$W高さ;$S幅+$W幅;$S高さ+$W高さ;$3)
  \ (Count parameters=4)
    OPEN WINDOW ($S幅-$W幅;$S高さ-$W高さ;$S幅+$W幅;$S高さ+$W高さ;$3;$4)
End case
```

プロジェクトメソッドを作成したら、次のようにこのメソッドを使用することができます：

```
ウインドウ中央表示 (400 ; 250 ; Movable dialog box ; "更新")
DIALOG ([テーブル1];"更新オプション")
CLOSE WINDOW
If (OK=1)
   …
End if
```

2. 次の例は、コントロールメニューボックス（クローズボックス）のメソッドを持ったフローティングウインドウを開きます。このウインドウは、画面の右上端に開きます。このメソッドが画面の大きさを気にしなくてよいように**Screen width**関数を使用している点に注目してください：

```
Open window (Screen width-149;33;Screen width-4;178;-Palette window;""; "CloseBox")
DIALOG ([ダイアログ] ; "カラーパレット")
```

“ CloseBox ”メソッドは、**CANCEL**コマンドを呼び出します：

```
CANCEL
```

3. 次の例は、ウインドウに表示されるフォームのプロパティから得たサイズとタイトルを持つウインドウを開きます：

```
INPUT FORM ([顧客] ; "レコード追加" ; *)
Open window (10 ; 80 ; -1 ; -1 ; Plain window ; "")
Repeat
   ADD RECORD ([顧客])
Until (OK=1)
```

注：Open window関数でフォームプロパティを自動的に使用できるようにするには、オプション引数の＜*＞を持ったINPUT FORMコマンドを呼び出し、さらに「デザイン」モードでそのフォームのプロパティがそれに応じて設定されている必要があります。

参照：CLOSE WINDOW、Open external window

Open external window

Open external window (左;上;右;下;タイプ;タイトル;プラグインエリア)→整数

引数	タイプ		説明
左	数値	→	ウインドウ左端の画面上の位置(ピクセル)
上	数値	→	ウインドウ上端の画面上の位置(ピクセル)
右	数値	→	ウインドウ右端の画面上の位置(ピクセル)
下	数値	→	ウインドウ下端の画面上の位置(ピクセル)
タイプ	数値	→	ウインドウのタイプ
タイトル	文字列	→	ウインドウのタイトル
プラグインエリア	文字列	→	外部ルーチンエリアコマンド
関数の返す値	数値	←	ウインドウ参照番号

Open external window関数は、新しいウインドウを開きます。4Dプラグインで提供される引数＜プラグインエリア＞のコマンドでサポートされる外部ルーチンエリアを表示します。

Open external window関数は、倍長整数の値を返します。この値は、（他のウインドウコマンドで使用される）ウインドウ参照番号および（4Dプラグインで提供される他のルーチンで使用される）ウインドウに表示される外部ルーチンへの参照番号として使用されます。

最初の6個の引数は**Open window**関数と同じです。しかし、いずれの引数もオプションではありません。すべて引数は指定します。

Open external window関数は、モードレスウインドウを作成します。このコマンドはユーザの入力を待ちません。従って、複数のアクティブウインドウを一度に開くことができます。各ウインドウは、クリックすることによって最前面に移動して操作することができます。タイトルバーを持っているウインドウタイプの場合、コントロール-メニューボックス（Windows）またはクローズボックス（Macintosh）を使ってウインドウを閉じることができます。

▼ 次の例は、外部ウインドウを開き、外部ルーチンエリアに“ 4D Draw ”を表示します：

```
X:=Open external window (50 ; 50 ; 350 ; 450 ; 8 ; "4D Draw" ; "_4D Draw")
```

▼ 次の例は、上記の例で開いた外部ウインドウを閉じます：

```
CLOSE WINDOW (X)
```

参照：CLOSE WINDOW、Open window

CLOSE WINDOW

CLOSE WINDOW ({外部エリア参照番号})

引数	タイプ	説明
外部エリア参照番号	数値	外部エリア参照番号

CLOSE WINDOWコマンドは、**Open window**関数で開いたウインドウを閉じます。**CLOSE WINDOW**コマンドは、カスタムウインドウが開かれていない場合には、何も行いません。また、標準ウインドウに対しても何も行いません。**CLOSE WINDOW**コマンドをレイアウトがアクティブ状態の時に使用した場合には、何も行いません。**Open window**関数で開いたウインドウは、必ず**CLOSE WINDOW**コマンドで閉じてください。

オプション引数＜外部エリア参照番号＞は、外部エリア参照番号で指定された外部ウインドウを閉じます。

▼ 次の例は、ウインドウを開き**ADD RECORD**コマンドを使用して新しいレコードを追加します。レコードを追加した時点で、**CLOSE WINDOW**コマンドによってウインドウを閉じます：

```
Open window (5 ; 40 ; 250 ; 300 ; 0 ; "新規採用")
Repeat                                  ` 取り消すまでループする
  ADD RECORD ([従業員])                 ` 新しいレコードを追加する
Until (OK=0)
CLOSE WINDOW                            ` カスタムウインドウを閉じる
```

ERASE WINDOW

ERASE WINDOW ({ウインドウ})

引数	タイプ		説明
ウインドウ	ウインドウ参照番号	→	ウインドウ参照番号、または省略した場合はカレントプロセスの最前面ウインドウ

ERASE WINDOWコマンドは、引数＜ウインドウ＞に渡した参照番号を持つウインドウを内容を消去します。

＜ウインドウ＞を省略すると、**ERASE WINDOW**コマンドはカレントプロセスの最前面ウインドウの内容を消去します。

通常、**ERASE WINDOW**コマンドは、**MESSAGE**コマンドと**GOTO XY**コマンドと一緒に組み合わせて使用します。この場合、**ERASE WINDOW**コマンドはウインドウ内容を消去し、カーソルをウインドウの左上端の位置を「**GOTO XY** (0 ; 0)」に移動します。

ウインドウ内容を消去する**ERASE WINDOW**コマンドと画面からウインドウ自体を消去する**CLOSE WINDOW**コマンドを混同しないように注意してください。

参照：GOTO XY、MESSAGE

REDRAW WINDOW

REDRAW WINDOW ({ウインドウ})

引数	タイプ		説明
ウインドウ	ウインドウ参照番号	→	ウインドウ参照番号、または省略した場合はカレントプロセスの最前面ウインドウ

REDRAW WINDOWコマンドは、引数＜ウインドウ＞に渡した参照番号のウインドウをグラフィカルに更新することができます。

＜ウインドウ＞を省略すると、**REDRAW WINDOW**コマンドはカレントプロセスの最前面ウインドウに適用します。

注：4th Dimensionでは、ウインドウの移動、サイズ変更、前面への移動等を行う度にグラフィカル更新をハンドルします。また、ウインドウ内のフォームやウインドウ内に表示されている値を変更する場合もグラフィカル更新をハンドルします。

参照：ERASE WINDOW

GET WINDOW RECT

GET WINDOW RECT (左 ; 上 ; 右 ; 下 {; ウインドウ})

引数	タイプ		説明
左	数値	→	ウインドウの内容エリアの左座標
上	数値	→	ウインドウの内容エリアの上座標
右	数値	→	ウインドウの内容エリアの右座標
下	数値	→	ウインドウの内容エリアの下座標
ウインドウ	ウインドウ参照番号	→	ウインドウ参照番号、または省略した場合はカレントプロセスの最前面ウインドウ

GET WINDOW RECTコマンドは、引数＜ウインドウ＞に渡した参照番号を持つウインドウのグローバル座標を返します。

＜ウインドウ＞を省略すると、**GET WINDOW RECT**コマンドはカレントプロセスの最前面ウインドウに適用します。

座標は、Windows上ではアプリケーションウインドウ、Macintosh上ではメイン画面の内容エリアの左隅との相対位置で表されます。座標は、（タイトルバーと境界線を除いた）ウインドウの内容エリアと一致する矩形を返します。

参照：SET WINDOW RECT

SET WINDOW RECT

SET WINDOW RECT (左 ; 上 ; 右 ; 下 {; ウインドウ})

引数	タイプ		説明
左	数値	→	ウインドウの内容エリアの左座標
上	数値	→	ウインドウの内容エリアの上座標
右	数値	→	ウインドウの内容エリアの右座標
下	数値	→	ウインドウの内容エリアの下座標
ウインドウ	ウインドウ参照番号	→	ウインドウ参照番号、または省略した場合はカレントプロセスの最前面ウインドウ

SET WINDOW RECTコマンドは、引数＜ウインドウ＞に渡した参照番号を持つウインドウのグローバル座標を返します。

＜ウインドウ＞を省略すると、**SET WINDOW RECT**コマンドはカレントプロセスの最前面ウインドウに適用します。

このコマンドは、引数で受け渡された新しい座標でウインドウサイズを変更したり、ウインドウを移動することができます。

座標は、Windows上ではアプリケーションウインドウ、Macintosh上ではメイン画面の内容エリアの左隅との相対位置で表されます。座標は、（タイトルバーと境界線を除いた）ウインドウの内容エリアと一致する矩形を返します。

警告：このコマンドを使用することにより、Windows上ではメインウインドウ、Macintosh上ではメイン画面の表示範囲を越えてウインドウを移動することができてしまいます。これを防ぐには、Screen width関数やScreen height関数を使って、ウインドウの新しい座標を2重チェックします。

参照：DRAG WINDOW、GET WINDOW RECT

MINIMIZE WINDOW

MINIMIZE WINDOW ({ウインドウ})

引数	タイプ		説明
ウインドウ	ウインドウ参照番号	→	ウインドウ参照番号

このコマンドに関する詳細は、次回のバージョンアップ時に記述する予定です。

MAXIMIZE WINDOW

MAXIMIZE WINDOW ({ウインドウ})

引数	タイプ		説明
ウインドウ	ウインドウ参照番号	→	ウインドウ参照番号

このコマンドに関する詳細は、次回のバージョンアップ時に記述する予定です。

SET WINDOW TITLE

SET WINDOW TITLE (タイトル {; ウインドウ})

引数	タイプ		説明
タイトル	文字列	→	ウインドウタイトル
ウインドウ	ウインドウ参照番号	→	ウインドウ参照番号

SET WINDOW TITLEコマンドは、引数＜ウインドウ＞に渡した参照番号を持つウインドウのタイトルを引数＜タイトル＞（最大80バイト）で指定したテキストに変更します。

ウインドウが存在しない場合、**SET WINDOW TITLE**コマンドは何も行いません。

＜ウインドウ＞を省略すると、**SET WINDOW TITLE**コマンドはカレントプロセスの最前面ウインドウに適用します。

注：「ユーザ」モードでは、4th Dimensionは自動的にウインドウタイトルを変更します。例えば、データ入力を行う場合、ウインドウタイトルは “ 更新：テーブル名 ” となります。ウインドウタイトルを変更すると、4th Dimensionはそれを上書きします。これに対して、「カスタム」モードでは、4th Dimensionはウインドウタイトルを変更しません。

例：

次の例は、任意のフォームでデータ入力を行っている最中に、（プログラムでサブフォームに表示されるリレートレコードをブラウズするような）長い処理を実行するボタンをクリックして、現在のウインドウタイトルを使って、その処理の進行状況を確認し続けることができます：

```
` 「bAnalysis」ボタンのオブジェクトメソッド
Case of
   \ (Form event=On Clicked)
      `ローカル変数に現在のウインドタイトルを保存する
      $vsCurTitle:=Get window title
      `長い処理を開始する
      FIRST RECORD([送り状明細])
      For ($vlRecord ; 1 ; Records in selection([送り状明細]))
         DO SOMETHING
         `処理経過を表示する
         SET WINDOW TITLE( "処理数 #"+String($vlRecord))
      End for
         `元のウインドウタイトルに戻す
      SET WINDOW TITLE ($vsCurTitle)
End case
```

参照：Get window title

DRAG WINDOW

DRAG WINDOW

引数　　タイプ　　説明

このコマンドには、引数はありません。

DRAG WINDOWコマンドは、マウス移動にしたがって、現在の最前面ウインドウをドラッグします。通常、このコマンドは、マウスクリックに瞬時に反応できるオブジェクト（例えば、透明ボタン）のオブジェクトメソッドから呼び出します。

例：

次のフォームは、固定のピクチャで作成された枠を持っています。このピクチャの枠の各コーナーには4つの透明ボタンが設定されています。

各ボタンは、次のメソッドを持っています。

DRAG WINDOW`クリックすると、ウインドウのドラッグを開始する

「ユーザ」モードまたは「カスタム」モードでは、次のプロジェクトメソッドを実行すると、次ページのような図を取得します：

```
Open window(50 ; 50 ; 50+400 ; 50+300 ; 2)
DIALOG([Table1] ; "Custom Drag")
CLOSE WINDOW
```

各コーナーの任意の場所をクリックすると、ウインドウをドラッグすることができます。

参照：GET WINDOW RECT, SET WINDOW RECT

Get window title

Get window title ({ウインドウ}) → 文字列

引数	タイプ		説明
ウインドウ	ウインドウ参照番号	→	ウインドウ参照番号、省略した場合は、カレントプロセスの最前面ウインドウ
関数の返す値	文字列	←	ウインドウのタイトル

Get window title関数は、引数＜ウインドウ＞に渡される参照番号を持つウインドウのタイトルを返します。ウインドウが存在しない場合は、空の文字列が返されます。

引数＜ウインドウ＞を省略した場合は、**Get window title**関数はカレントプロセスの最前面ウインドウのタイトルを返します。

▼ **SET WINDOW TITLE**コマンドの例を参照してください。

参照：SET WINDOW TITLE

HIDE WINDOW

HIDE WINDOW ({ウインドウ})

引数	タイプ		説明
ウインドウ	ウインドウ参照番号	→	ウインドウ参照番号、省略した場合は、 カレントプロセスの最前面ウインドウ

このコマンドに関する詳細は、次のアップデート時に行う予定です。

SHOW WINDOW

SHOW WINDOW ({ウインドウ})

引数	タイプ		説明
ウインドウ	ウインドウ参照番号	→	ウインドウ参照番号、省略した場合は、 カレントプロセスの最前面ウインドウ

このコマンドに関する詳細は、次のアップデート時に行う予定です。

Frontmost window

Frontmost window ({*}) → ウインドウ参照番号

引数	タイプ		説明
*		→	指定した場合は、フローティングウインドウを考慮 省略した場合は、フローティングウインドウを無視

関数の返す値	参照番号	←	ウインドウ参照番号

Frontmost window関数は、最前面ウインドウのウインドウ参照番号を返します。

参照：Frontmost process、Next window

WINDOW LIST

WINDOW LIST (ウインドウ {; *})

引数	タイプ		説明
ウインドウ	配列	→	ウインドウ参照番号の配列
*		→	指定した場合は、フローティングウインドウを考慮 省略した場合は、フローティングウインドウを無視

WINDOW LISTコマンドは、引数＜ウインドウ＞の配列に動作しているすべてのプロセスの中で現在開いているウインドウのウインドウ参照番号を配置します。

オプション引数＜ * ＞を指定しなかった場合は、フローティングウインドウは無視されます。

例：

次のプロジェクトメソッドは、フローティングウインドウとダイアログボックスを除いて、現在開いているウインドウ全部を並べます：

```
` 「TILE WINDOWS」プロジェクトメソッド
WINDOW LIST($alWnd)
$vlLeft:=10
$vlTop:=80 `ツールバー用の空間を確保する
For ($vlWnd ; 1 ; Size of array($alWnd))
   If (Window kind($alWnd{$vlWnd} # Modal Dialog)
      GET WINDOW RECT($vlWL ; $vlWT ; $vlWR ; $vlWB ; $alWnd{$vlWnd})
      $vlWR:=$vlLeft+($vlWR-$vlWL)
      $vlWB:=$vlTop+($vlWB-$vlWT)
      $vlWL:=$vlLeft
      $vlWT:=$vlTop
      SET WINDOW RECT($vlWL ; $vlWT ; $vlWR ; $vlWB ; $alWnd{$vlWnd})
      $vlLeft:=$vlLeft+10
      $vlTop:=$vlTop+25
   End if
End for
```

注：このメソッドは、メインウインドウのサイズ（Windows上）または画面サイズと場所（Macintosh上）を調べるために利用することができます。

参照：Window kind、Window process

Window kind

Window kind ({ウインドウ}) → 数値

引数	タイプ		説明
ウインドウ	ウインドウ参照番号	→	ウインドウ参照番号、省略した場合は、カレントプロセスの最前面ウインドウ
関数の返す値	数値	←	ウインドウのタイプ

Window kind関数は、引数＜ウインドウ＞に渡された参照番号を持つウインドウの4th Dimensionタイプを返します。もし、ウインドウが存在しない場合は、**Window kind**関数は0を返します。

それ以外は、**Window kind**関数は次の値の1つを返します。

定数	タイプ	値
Regular window	倍長整数	8
Modal dialog	倍長整数	9
External window	倍長整数	5
Floating window	倍長整数	14

引数＜ウインドウ＞を省略した場合は、**Window kind**関数はカレントプロセスの最前面ウインドウのタイプを返します。

▼ **WINDOW LIST**コマンドの例を参照してください。

参照：GET WINDOW RECT、Get window title、Window process

Window process

Window process ({ウインドウ}) → 数値

引数	タイプ		説明
ウインドウ	ウインドウ参照番号	→	ウインドウ参照番号、省略した場合は、カレントプロセスの最前面ウインドウ
関数の返す値	数値	←	プロセス参照番号

Window process関数は、引数＜ウインドウ＞に渡された参照番号を持つウインドウを動作するプロセス番号を返します。もし、ウインドウが存在しない場合は、**Window process**関数は 0を返します。

引数＜ウインドウ＞を省略した場合は、**Window process**関数はカレント最前面ウインドウのプロセスを返します。

参照：Current process

Next window

Next window ({ウインドウ}) → 数値

引数	タイプ		説明
ウインドウ	ウインドウ参照番号	→	ウインドウ参照番号、省略した場合は、カレントプロセスの最前面ウインドウ
関数の返す値	数値	←	ウインドウ参照番号

Next window関数は、（ウインドウの前後の順番を基にした）引数＜ウインドウ＞に渡された参照番号を持つウインドウの“ 背面 ”にあるウインドウのウインドウ参照番号を返します。もし、ウインドウが存在しない場合は、**Window process**関数は 0を返します。

参照：Frontmost window

Find window

Finf window (左 ; 上 ;{ ; ウインドウ部分}) → ウインドウ参照番号

引数	タイプ		説明
左	数値	→	ウインドウ左端の画面上の位置
上	数値	→	ウインドウ上端の画面上の位置
ウインドウ部分	数値	←	ウインドウ部分のID番号 カレントプロセスの最前面ウインドウ
関数の返す値　数値		←	ウインドウ参照番号

Find window関数は、引数＜左＞と＜上＞で渡された画面上の位置に触れている最初のウインドウの参照番号を返します。

画面上の位置は、アプリケーションウインドウ（Windows）の内容エリアの左上隅またはメインスクリーン（Macintosh）の相対位置で表されなければなりません。

オプション引数＜ウインドウ部分＞を指定した場合、任意ウインドウの有無に関係なく、この引数は次の値の1つを返します。

定数	タイプ	値	プラットフォーム
In menu bar	倍長整数	1	Macintoshのみ
In system window	倍長整数	2	Macintoshのみ
In contents	倍長整数	3	WindowsまたはMacintosh
In drag	倍長整数	4	Macintoshのみ
In grow	倍長整数	5	Macintosh のみ
In go away	倍長整数	6	Macintosh のみ
In zoom box	倍長整数	7	Macintoshのみ

参照：Frontmost window、Next window

システム変数　A

この付録では、4th Dimensionのシステム変数についての概要を示します。4th Dimensionのシステム変数はプロセス変数です。1プロセスに各システム変数が1つあります。

OK

システム変数OKは、システム変数の中で最も頻繁に使用されます。一般的には、処理が成功すると1が代入され、成功しなかったときに 0が代入されます。次のコマンドは、システム変数OKに値を代入します：

ADD RECORD	**ADD SUBRECORD**	**Append document**
APPLY TO SELECTION	**ARRAY TO LIST**	**ARRAY TO SELECTION**
CHANGE ACCESS	**CONFIRM**	**Create document**
DELETE DOCUMENT	**DIALOG**	**DISTINCT VALUES**
EXPORT DIF	**EXPORT SYLK**	**EXPORT TEXT**
IMPORT DIF	**IMPORT SYLK**	**IMPORT TEXT**
JOIN	**LOAD SET**	**LOAD VARIABLES**
MODIFY RECORD	**MODIFY SUBRECORD**	**ORDER BY**
ORDER BY FORMULA	**Open document**	**PLAY**
PRINT LABEL	**PRINT SELECTION**	**PRINT SETTINGS**
PROJECT SELECTION	**QUERY**	**QUERY BY EXAMPLE**
QUERY BY FORMULA	**QUERY SELECTION**	**QUERY SELECTION**
RECEIVE PACKET	**RECEIVE RECORD**	**BY FORMULA**
RECEIVE VARIABLE	**REPORT**	**Request**
SAVE SET	**SAVE VARIABLES**	**SELECT LOG FILE**
SEND PACKET	**SEND RECORD**	**SEND VARIABLE**
SET CHANNEL	**USE ASCII MAP**	**VALIDATE TRANSACTION**

注：バージョン6で新規に追加されたコマンドについては、各コマンドの説明の中で示しています。

Document

システム変数Documentには、次のコマンドで開かれたり、作成された最後のMacintoshのディスクファイルの名前が格納されます：

Append document	**Create document**	**EXPORT DIF**
EXPORT SYLK	**EXPORT TEXT**	**IMPORT DIF**
IMPORT SYLK	**IMPORT TEXT**	**LOAD SET**
LOAD VARIABLES	**Open document**	**PRINT LABEL**
REPORT	**SAVE SET**	**SAVE VARIABLES**
SELECT LOG FILE	**SET CHANNEL**	**USE ASCII MAP**

注：バージョン6で新規に追加されたコマンドについては、各コマンドの説明の中で示しています。

FldDelimit

システム変数FldDelimitは、テキストを読み込んだり書き出す際に、フィールドの区切り文字として使用する文字のASCIIコードが格納されています。デフォルトの値は、ASCIIコードでタブに相当する9です。区切り文字を変更する場合は、この値を変更してください。

RecDelimit

システム変数RecDelimitは、テキストを読み込んだり書き出す際に、レコードの区切り文字として使用する文字のASCIIコードが格納されています。デフォルトの値はASCIIコードでキャリッジリターン(CR)に相当する13です。区切り文字を変更する場合は、この値を変更してください。

Error

システム変数Errorは、**ON ERR CALL**コマンドによってインストールされたエラー処理メソッドの中でのみ有効です。この変数にはエラーコードが格納されています。付録Cに4th DimensionとWindows（Macintosh）のエラーコードのリストを示します。

MouseDown、MouseX、MouseY、KeyCode、Modifiers、MouseProc

これらのシステム変数は、**ON EVENT CALL**コマンドによってインストールされたイベントメソッドの中でのみ有効です。

システム変数MouseDownは、マウスのボタンが押されたときに1が、それ以外の場合は0に代入されます。

イベントがMouseDownの場合は、 システム変数MouseXとMouseYはマウスがクリックされた場所の水平と垂直の座標を含みます。両方の値ともピクセル単位で表わされ、ウインドウのローカルな座標システム内に存在します。

システム変数KeyCodeは、押されたキーのASCIIコードが代入されます。

システム変数Modifiersは、Windows（Macintosh）のキーボードのモディファイアキーの値を含んでいます。システム変数Modifiersは、イベントの両方のタイプに有効です。詳細は、**ON EVENT CALL**コマンドに関する記述を参照してください。

システム変数MouseProcは、イベントが発生したプロセス番号を含みます。システム変数MouseProcは、イベントの両方のタイプに有効です。

この付録には、3つの表があります：

■ 第1の表は、Windows と Macintosh に共通の標準 ASCII コード(0 から 127)です。

■ 第2の表は、Macintosh に関する ASCII コード 128 から 255 と、それが 4th Dimension によって Windows 上でどのように表示されるかを示したものです。

■ 第3の表は、ファンクションキーに関する特殊キーコード値です。これらの値は **ON EVENT CALL**コマンド のメソッド内でファンクションキーを調べる際に使用できます。

ASCII コード 0 から 127

次のコードは、Macintosh と Windows に共通な ASCII コードです。

ASCII		MacintoshまたはWindows
10進	16進	結果
0	0	NUL
1	1	SOH
2	2	STX
3	3	ETX
4	4	EOT
5	5	ENQ
6	6	ACK
7	7	BEL
8	8	BS

ASCII		MacintoshまたはWindows
10進	16進	結果
9	9	HT
10	A	LF
11	B	VT
12	C	FF
13	D	CR
14	E	SO
15	F	SI
16	10	DLE
17	11	DC1

ASCII		MacintoshまたはWindows
10進	16進	結果
18	12	DC2
19	13	DC3
20	14	DC4
21	15	NAK
22	16	SYN
23	17	ETB
24	18	CAN
25	19	EM
26	1A	SUB
27	1B	ESC
28	1C	FS
29	1D	GS
30	1E	RS
31	1F	US
32	20	sp
33	21	!
34	22	"
35	F	SI
36	10	DLE
37	11	DC1
38	12	DC2
39	13	DC3
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-

ASCII		MacintoshまたはWindows
10進	16進	結果
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	F	SI
70	10	DLE
71	11	DC1
72	12	DC2
73	13	DC3

ASCII		MacintoshまたはWindows
10進	16進	結果
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d

ASCII		MacintoshまたはWindows
10進	16進	結果
101	55	e
102	56	f
103	57	g
104	58	h
105	59	i
106	5A	j
107	5B	k
108	5C	l
109	5D	m
110	5E	n
111	5F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	¦
125	7D	}
126	7E	~
127	7F	DEL

拡張ASCIIコード（128 から 255）

次の表は、各ASCIIコードについて、Macintosh と Windows 上で 4th Dimension によって表示される文字を示したものです。

表にはさらに、個々のプラットフォーム上で US キーボードを使用した場合に文字を得るためのキーボード操作も含まれています。

注：Windows 欄の黒く塗りつぶされている箇所は、Windows 上では使用できない文字、または Macintoshとは異なる文字を示します。それから、Macintosh側の「結果」欄のTimesフォントの後にカッコ「( )」で表されている文字は、Osakaフォントで表示された場合のものです。

ASCII		Macintosh		WIndows	
10進	16進	結果 (Timesﾌｫﾝﾄ)	キーボード操作	結果 (Arialﾌｫﾝﾄ)	キーボード操作
128	80	Ä (\)	Option-u, Shift-a	Ä	Alt+0196
129	81	Å	Option-Shift-a	Å	Alt+0197
130	82	Ç	Option-Shift-c	Ç	Alt+0199
131	83	É	Option-e, Shift-e	É	Alt+0201
132	84	Ñ	Option-n, Shift-n	Ñ	Alt+0209
133	85	Ö	Option-u, Shift-o	Ö	Alt+0214
134	86	Ü	Option-u, Shift-u	Ü	Alt+0220
135	87	á	Option-e, a	á	Alt+0225
136	88	à	Option-`, a	à	Alt+0224
137	89	â	Option-i, a	â	Alt+0226
138	8A	ä	Option-u, a	ä	Alt+0228
139	8B	ã	Option-n, a	ã	Alt+0227
140	8C	å	Option-a	å	Alt+0229
141	8D	ç	Option-c	ç	Alt+0231
142	8E	é	Option-e, e	é	Alt+0233
143	8F	è	Option-`, e	è	Alt+0232
144	90	ê	Option-i, e	ê	Alt+0234
145	91	ë	Option-u, e	ë	Alt+0235
146	92	í	Option-e, i	í	Alt+0237

ASCII		Macintosh		WIndows	
10進	16進	結果 (Timesﾌｫﾝﾄ)	キーボード操作	結果 (Arialﾌｫﾝﾄ)	キーボード操作
147	93	ì	Option-`, i	ì	Alt+0236
148	94	î	Option-i, i	î	Alt+0238
149	95	ï	Option-u, i	ï	Alt+0239
150	96	ñ	Option-n, n	ñ	Alt+0241
151	97	ó	Option-e, o	ó	Alt+0243
152	98	ò	Option-`, o	ò	Alt+0242
153	99	ô	Option-i, o	ô	Alt+0244
154	9A	ö	Option-u, o	ö	Alt+0246
155	9B	õ	Option-n, o	õ	Alt+0245
156	9C	ú	Option-e, u	ú	Alt+0250
157	9D	ù	Option-`, u	ù	Alt+0249
158	9E	û	Option-i, u	û	Alt+0251
159	9F	ü	Option-u, u	ü	Alt+0252
160	A0	†	Option-t		
161	A1	° (｡)	Shift-Option-8	°	Alt+0176
162	A2	¢ (｢)	Option-4	¢	Alt+0162
163	A3	£ (｣)	Option-3	£	Alt+0163
164	A4	§ (･)	Option-6	§	Alt+0167
165	A5	• (.)	Option-8		
166	A6	¶ (ｦ)	Option-7	¶	Alt+0182
167	A7	ß (ｧ)	Option-s	ß	Alt+0223
168	A8	® (ｨ)	Option-r	®	Alt+0174
169	A9	© (ｩ)	Option-g	©	Alt+0169
170	AA	™ (ｪ)	Option-2	™	Alt-0153
171	AB	´ (ｫ)	Shift-Option-e	´	Alt+0145

ASCII		Macintosh		WIndows	
10進	16進	結果 (Timesﾌｫﾝﾄ)	キーボード操作	結果 (Arialﾌｫﾝﾄ)	キーボード操作
172	AC	¨ (ｬ)	Shift-Option-u	¨	Alt+0168
173	AD	≠ (ｭ)	Option-=		
174	AE	Æ (ｮ)	Shift-Option-"	Æ	Alt+0198
175	AF	Ø (ｯ)	Shift-Option-o	Ø	Alt+0216
176	B0	∞ (ｰ)	Option-5		
177	B1	± (ｱ)	Shift-Option-=	±	Alt+0177
178	B2	≤ (ｲ)	Option-,		
179	B3	≥ (ｳ)	Option-. (period)		
180	B4	¥ (ｴ)	Option-y	¥	Alt+0165
181	B5	µ (ｵ)	Option-m	µ	Alt+0181
182	B6	∂ (ｶ)	Option-d		
183	B7	∑ (ｷ)	Option-w		
184	B8	∏ (ｸ)	Shift-Option-p		
185	B9	π (ｹ)	Option-p		
186	BA	∫ (ｺ)	Option-b		
187	BB	ª (ｻ)	Option-9	ª	Alt+0170
188	BC	º (ｼ)	Option-0 (zero)	º	Alt+0186
189	BD	Ω (ｽ)	Option-z		
190	BE	æ (ｾ)	Option-:	æ	Alt+0230
191	BF	ø (ｿ)	Option-o	ø	Alt+0248
192	C0	¿ (ﾀ)	Shift-Option-?	¿	Alt+0191
193	C1	¡ (ﾁ)	Option-1	¡	Alt+0161
194	C2	¬ (ﾂ)	Option-l (L)	¬	Alt+0172
195	C3	√ (ﾃ)	Option-v		
196	C4	ƒ (ﾄ)	Option-f		

ASCII		Macintosh		WIndows	
10進	16進	結果 (Timesﾌｫﾝﾄ)	キーボード操作	結果 (Arialﾌｫﾝﾄ)	キーボード操作
197	C5	≈ (ｶ)	Option-x		
198	C6	Δ (ﾆ)	Option-j		
199	C7	« (ﾇ)	Option-]	«	Alt+0171
200	C8	» (ﾈ)	Shift-Option-]	»	Alt+0187
201	C9	… (ﾉ)	Option-;	…	Alt+0133
202	CA	(space)	Spacebar	(space)	Alt+0160
203	CB	À (ﾋ)	Option-`, Shift-a	À	Alt+0192
204	CC	Å (ﾌ)	Option-n, Shift-a	Å	Alt+0195
205	CD	Õ (ﾍ)	Option-n, Shift-o	Õ	Alt+0213
206	CE	Œ (ﾎ)	Shift-Option-q		
207	CF	œ (ﾏ)	Option-q		
208	D0	– (ﾐ)	Option--(dash)		
209	D1	— (ﾑ)	Shift-Option-(dash)		
210	D2	“ (ﾒ)	Option-@	“	Alt+0147
211	D3	” (ﾓ)	Shift-Option-@	”	Alt+0148
212	D4	‘ (ﾔ)	Option-[	‘	Alt+0145
213	D5	’ (ﾕ)	Shift-Option-[	’	Alt+0146
214	D6	÷ (ﾖ)	Option-/	÷	Alt+0247
215	D7	◊ (ﾗ)	Shift-Option-v		
216	D8	ÿ (ﾘ)	Option-u, y	ÿ	Alt+0255
217	D9	Ÿ (ﾙ)	Option-u, Shift-y		
218	DA	⁄ (ﾚ)	Shift-Option-1		
219	DB	¤ (ﾛ)	Shift-Option-2	¤	Alt+0164
220	DC	‹ (ﾜ)	Shift-Option-3		
221	DD	› (ﾝ)	Shift-Option-4		

ASCII		Macintosh		WIndows	
10進	16進	結果 (Timesﾌｫﾝﾄ)	キーボード操作	結果 (Arialﾌｫﾝﾄ)	キーボード操作
222	DE	fi	Shift-Option-5		
223	DF	fl	Shift-Option-6		
224	E0	‡	Shift-Option-7		
225	E1	·	Shift-Option-9	·	Alt+0183
226	E2	,	Shift-Option-0		
227	E3	„	Shift-Option-w		
228	E4	‰	Shift-Option-r		
229	E5	Â	Option-i, Shift-a	Â	Alt+0194
230	E6	Ê	Option-i, Shift-e	Ê	Alt+0202
231	E7	Á	Option-e, Shift-a	Á	Alt+0193
232	E8	Ë	Option-u, Shift-e	Ë	Alt+0203
233	E9	È	Option-`, Shift-e	È	Alt+0200
234	EA	Í	Shift-Option-s	Í	Alt+0205
235	EB	Î	Shift-Option-d	Î	Alt+0206
236	EC	Ï	Shift-Option-f	Ï	Alt+0207
237	ED	Ì	Option-`, Shift-i	Ì	Alt+0204
238	EE	Ó	Shift-Option-h	Ó	Alt+0211
239	EF	Ô	Shift-Option-j	Ô	Alt+0212
240	F0		Shift-Option-k		
241	F1	Ò	Shift-Option-l (L)	Ò	Alt+0210
242	F2	Ú	Shift-Option-;	Ú	Alt+0218
243	F3	Û	Option-i, Shift-u	Û	Alt+0219
244	F4	Ù	Option-`, Shift-u	Ù	Alt+0217
245	F5	ı	Shift-Option-b		
246	F6	ˆ	Shift-Option-i		

ASCII		Macintosh		WIndows	
10進	16進	結果 (Timesﾌｫﾝﾄ)	キーボード操作	結果 (Arialﾌｫﾝﾄ)	キーボード操作
247	F7	˜	Shift-Option-n		
248	F8	¯	Shift-Option-,	¯	Alt+0175
249	F9	˘	Shift-Option-.		
250	FA	˙	Option-h		
251	FB	°	Option-k		
252	FC	¸	Shift-Option-z	¸	Alt+0184
253	FD	˝	Shift-Option-g		
254	FE	˛	Shift-Option-x		
255	FF	ˇ	Shift-Option-t		

ASCIIコードと 4th Dimensionの理解

内部のデータベースエンジンと Macintosh 上の 4th Dimensionランゲージは、Macintosh と Windows の両方とも拡張ASCIIセットを処理します。

Windows上でキーボードから情報を入力すると（レコードの追加、メソッドの編集等を行う）、4- Dimensionでは国際的なAltura ASCII 変換方式を使用して、キーボード入力（つまり、Windows 文字セットによる表現）を文字セットに変換します。

例えば “ é “ を入力する場合、ALT+0233 とタイプしますが、4th Dimensionではレコードにコード 142 を保存します。これは、エンドユーザには見えません。例えば、検索を行う場合、検索対象の値を単に入力します。つまり、タイプした値（ALT+0233）は同時にASCIIコード 142 に翻訳され、目的のレコードを見つけることができるわけです。

「メソッド」エディタに ALT+0233 とタイプする場合も同じことが行われます。しかし、ASCIIコードを使用して文字を探す場合、どちらのプラットーフォームでも「Macintosh」の ASCIIコードを使用します。

例えば、

QUERY (...; [MyFile]MyField="é") **`é は ASCII 142**

は次の書き方でも同じことです。

QUERY (...; [MyFile]MyField=Char (142)) **`é は ASCII 142**

従って ASCIIコードを含むコードも、両方のプラットフォームで同じように動作します。

ファンクションキー

4th Dimensionは、システム変数 KeyCode にファンクションキーの値を返します。ファンクションキーの値は ASCIIコードには属していません。

ファンクションキー	KeyCode
F1	-122
F2	-120
F3	-99
F4	-118
F5	-96
F6	-97
F7	-98
F8	-100
F9	-101
F10	-109
F11	-103
F12	-111
F13	-105
F14	-107
F15	-113

注：システム変数 KeyCode は ON EVENT CALL コマンドによって設定されるイベントプロジェクトメソッドで使用します。ただし、4th Dimensionには ON EVENT CALLコマンドは用意されていません。

ファンクションキーに加え、Return、Enter などの共通なキーを押した場合に KeyCode に返される値の表を以下に示します。

コード	キー
3	Enter
13	Return
8	Backspace または Delete
9	Tab
27	Escape または Clear
127	Del
5	Help
1	Home
4	End
11	Page Up
12	Page Down
28	左矢印
29	右矢印
30	上矢印
31	下矢印

この付録では、4- Dimensionの使用中に発生するエラーについて説明します。エラーは、メソッド内のシンタックスエラーまたは入出力ディスクエラーなどによって発生します。

これらのほとんどのエラーは、**ON ERR CALL**コマンドでエラーメソッドとしてインストールされた4th Dimensionメソッドによって中断させることができます。この場合、エラーコードは、4th Dimensionのシステム変数Errorに含まれます。

シンタックスエラー

次の表に、メソッドの中で発生するシンタックス（構文）エラーに対するコードを示します。これらは、メソッド作成時に検出されます。次の表は、「ユーザ」モードまたは「カスタム」モードでコード実行中に発生する可能性のあるエラーに対するシンタックスエラーコードを示しています。これらのエラーの中には、インタープリタモードだけで発生する可能性のあるもの、コンパイルモードだけで発生する可能性のあるもの、また、両方のモードで発生する可能性のあるものがあります。**ON ERR CALL**コマンドを使用すると、インストールされたエラー割り込みメソッドを使用してこれらのエラーに介入できます。

コード	エラーの起きた理由
1	“ (” が必要です。
2	フィールド名が必要です。
3	このコマンドはサブテーブルのフィールドに対してのみ実行可能です。
4	引数はすべて同じタイプでなければなりません。
5	このコマンドの対象となるテーブルがありません。
6	このコマンドはサブテーブル(フィールド)に対してのみ実行できます。
7	数値の引数が必要です。
8	文字(列)の引数が必要です。
9	条件判断に対する結果が必要です。
10	このコマンドはこのタイプのフィールドに対しては実行できません。
11	このコマンドは条件判断には使えません。
12	このコマンドには数値引数が使えません。
13	このコマンドには文字(列)引数が使えません。
14	このコマンドには日付引数が使えません。

コード	エラーの起きた理由
15	引数／変数のタイプが対応していません。
16	このフィールドにはリレートが設定されていません。
17	テーブル名が必要です。
18	フィールドのタイプが対応していません。
19	このフィールドにはインデックスの属性が設定されていません。
20	"="が必要です。
21	メソッドが見つかりません。
22	ソートとグラフに用いるフィールドは同一のテーブル／サブテーブルのものでなければなりません。
23	"<" または ">" が必要です。
24	";" が必要です。
25	ソートするフィールドが多すぎます。
26	このフィールドのタイプは数値、日付、文字(列)のいずれかでなければなりません。
27	このフィールド名はテーブル名を伴わなければなりません。
28	フィールドのタイプは数値でなければなりません。
29	数値は０または１でなければなりません。
30	変数が必要です。
31	この番号のメニューバーが見つかりません。
32	日付が必要です。
33	コマンドまたは関数が見つかりません。
35	このセットは他のテーブルに属しています。
36	ファイル名が違います。
37	":="が必要です。
39	セットが見つかりません。
40	このルーチンは関数です(メソッドではありません)。
41	サブテーブルのフィールドあるいは変数が必要です。
42	レコードをスタックにプッシュできません。
43	関数が見つかりません。
44	メソッドが見つかりません。
45	フィールドあるいは変数が必要です。
46	数値あるいは文字(列)の引数が必要です。
47	フィールドのタイプは"文字"でなければなりません。
48	シンタックス・エラーです。
49	この演算子はここでは使えません。
50	この演算子は一緒には使えません。
52	配列変数が必要です。
53	インデックスの範囲を超えています。
54	引数のタイプが違います。
55	ブール型の引数が必要です。

コード	エラーの起きた理由
56	フィールド、変数またはテーブル名が必要です。
57	演算子が必要です。
58	")"が必要です。
59	このタイプの引数はここでは使えません。
60	この引数またはローカル変数は、コンパイルしたデータベース上でEXECUTEステートメントでは使えません。
61	配列タイプはコンパイルしたデータベース上で修正できません。
62	このコマンドはサブテーブル（フィールド）に対しては使えません。
63	このフィールドにはインデックスの属性が設定されていません。

Tips：

これらのエラーコードの中には、タイプミスのための単純なシンタックスエラーを示しているものもあります。例えば、本当は「v:=0」なのにステートメント「v=0」を実行した場合には、エラー番号37が返されます。このエラーは、「デザイン」モードの「メソッド」エディタでコードを修正することによってなくすことができます。

これらのエラーコードの中には、単なるプログラミングエラーが原因であるものもあります。例えば、(**DEFAULT TABLE**コマンドを使用して)最初にデフォルトテーブルを設定しておらず、引数＜テーブル＞を渡していないときに**ADD RECORD**コマンドを実行すると、エラー番号5が返されます。この場合には、このコマンドに適用されるテーブルがありません。デフォルトテーブルの設定を忘れていたのかどうかや、コマンドに引数＜テーブル＞を渡すのを忘れていたのかどうかを確認することで、このエラーはなくなります。

これらのエラーコードの中には、設計上の欠点を示すものもあります。例えば、他のフィールドにリレートしていないフィールドに**RELATE ONE**コマンドを適用すると、エラー番号16が返されます。自分が作成したコードが本当に間違っているのか、あるいはそのフィールドから始まるリレーションを作成するのを忘れただけなのかを確認することで、このエラーはなくなります。

エラーの中には、発生すると、コードのどこで止まるのかを必ずしも正確に示さないものもあります。例えば、サブルーチンの中で「vpFld:=**Field(**$1;$2)」という行でエラー番号53(範囲外を示す)が返された場合に、そのエラーは引数としてサブルーチンに渡されたテーブルやフィールド番号が誤っていたことが原因です。このため、エラーはコール側のメソッド内で示され、エラーが実際に発生した場所は示しません。この場合、「デバッグ」ウィンドウでコードのどの行が本当の犯人であるかを突き止めてから「デザイン」モードの「メソッド」エディタで訂正してください。

内部のエラーコード

次の表は、データベースのカーネルによって生成されたエラーコードのリストです。これらは、ユーザによる中断、アクセス権エラー、およびダメージを受けたオブジェクトのようなデータベースエンジンの内部で発生したエラーを含んでいます。

コード	エラーの起きた理由
1006	ユーザによりプログラムが中断されました。ユーザが"オプション-クリックしました。
-9939	外部ルーチンが見つかりません。 例：4D のメソッド内で 4D Draw をコールするデータベースがあるとします。インストールされている 4D Draw を削除し、そのデータベースをオープンし直すと 4D Draw をコールしようとしたところで、このエラーが発生します。
-9940	4D Extension の初期化に失敗しました。 例：データベースのオープン時には、外部パッケージを見つけるためにストラクチャファイルや Proc.Ext ファイル、 Mac4DX フォルダや Win4DX フォルダが調べられます。コードインスタンスが見つからないパッケージがあると、このエラーが発生します。例えば、パッケージは存在しているが、68K の Macintosh 上でデータベースが起動中に 68Kのコードインスタンスが見つからない場合です。
-9941	EX_GESTALT セレクタが見つかりません。
-9942	4D Client のライセンスが 4D Server のバージョンと一致していません。
-9943	4D Passport のバージョンに誤りがあります。
-9944	ユーザが 4D Open アクセスのアクセスグループに属していません。
-9945	CD-ROM 4D Runtime エラー、書き込み処理は許可されていません。
-9946	その命名セレクションは存在しないのでクリアすることはできません。
-9947	「4D Client のみの接続を許可する」チェックボックスが選択されています。
-9950	データセグメント番号が無効です。
-9951	フィールドがリレートされていません。
-9952	データセグメントが無効です。
-9953	ログファイルがありません。
-9954	カレントレコードがありません。
-9955	QuickTime がインストールされていません。
-9956	この 4D Open のバージョンは 4D Server と互換性がありません。 あるいは、バージョン1.1.1 の 4D Client の「環境設定」ダイアログボックス内にある「4D Client のみの接続を許可する」チェックボックスが選択されています。
-9957	項目選択リストはロックされています。
-9958	プロセスが起動できませんでした。

コード	エラーの起きた理由
-9959	バックアッププロセスが他のユーザまたはプロセスによって開始されています。
-9960	サーバ上に4D Backupがインストールされてません。
-9961	バックアッププロセスが現在動いていません。
-9962	サーバがシャットダウンしているため、バックアップを起動することができません。
-9963	ワークステーションによって要求されたレコード番号が無効です。
-9964	ワークステーションによって送られたソートの定義ファイルが正しくありません。
-9965	ワークステーションによって送られた検索の定義ファイルが正しくありません。
-9966	ワークステーションによって要求されたタイプが無効です。
-9967	レコードがロードされていませんので、修正することができません。
-9968	ワークステーションによって要求されたレコード選択番号が無効です。
-9969	ワークステーションによって要求されたフィールドタイプが無効です。
-9970	フィールドにインデックスがありません。
-9971	フィールド番号がワークステーションによって要求された範囲外です。
-9972	テーブル番号がワークステーションによって要求された範囲外です。
-9973	TRICリソースが同一のものではありません。
-9974	レコードはすでに削除されています。
-9975	トランザクションインデックスページがロードされてません。
-9976	バックアップ処理中のため、修正することができません。
-9977	セレクションがありません。
-9978	ユーザのパスワードが間違っています。
-9979	ユーザが見つかりません。
-9980	ストラクチャがロックされているため、テーブルを作成することはできません。
-9981	ワークステーションによって送られたフィールド名／フィールド番号の定義ファイルが無効です。
-9982	レコードがワークステーション上で選択されていないため、ロードできませんでした。
-9983	同じ外部パッケージが２つインストールされています。
-9984	トランザクションは複製されたインデックスキーエラーのため取り消されました。
-9985	無限ループです。
-9986	自動削除処理中にレコードがロックされました。
-9987	他のレコードがすでにこのレコードにリレートしています。
-9989	ストラクチャの構造が正しくありません。（データベースを修復する必要があります。）
-9990	シリアルポートでタイムアウトが起きました。
-9991	特権エラーです。
-9992	パスワードが間違っています。

コード	エラーの起きた理由
-9993	メニューバーに障害があります。(データベースを修復する必要があります。)
-9994	シリアル通信がユーザの操作(option-space)によって割り込まれました。
-9995	デモバージョンの限界です。
-9996	スタックがいっぱいです。(再帰呼び出しまたはネストが深すぎます。)
-9997	レコードの最大の番号を越えています。
-9998	インデックスキーがすでに存在しています。エントリが複数存在します。
-9999	レコードを保存するディスクの領域が足りません。
-10500	データのアドレスが正しくありません。(データベースを修復する必要があります。)
-10501	１インデックスのブロックが正しくありません。(インデックスを修復する必要があります。)
-10502	レコードの構造が正しくありません。(データファイルを修復する必要があります。)
-10503	レコードの番号が範囲を越えています。(**GO TO RECORD**コマンドの実行中またはデータファイルを修復する必要があります。)
-10504	インデックスのブロック番号が範囲を越えています。(インデックスを修復する必要があります。)
-1	プラグインで要求された入力ポイントが見つかりません。
4001	プラグインで要求されたテーブル番号が正しくありません。
4002	プラグインで要求されたレコード番号が正しくありません。
4003	プラグインで要求されたフィールド番号が正しくありません。
4004	カレントレコードがないのにプラグインで要求されたテーブルのカレントレコードにアクセスしました。

備考：

1. リストされているエラーのどれかが重大な問題を反映している場合、例えば、-10502「レコードの構造が正しくありません。(データファイルを修復する必要があります。)」などの場合、それ以外のエラーは標準どおり発生する可能性があり、**ON ERR CALL**プロジェクトメソッドを使用して管理できます。例えば、エラー -9998「インデックスキーがすでに存在しています。エントリが複数存在します。」は、「重複不可」プロパティが設定されているインデックス付きフィールドを含むテーブルに複写した値を作成する機会をアプリケーションで提供すれば、普通に扱うことができます。

2. リストされているエラーの中には4D言語レベルでは決して発生しないものもあります。これらのエラーはデータベースエンジンルーチンによってローレベルで発生し、処理されるか、あるいは、4D Backupや4DOpenを使用したときに発生し、処理されます。

3. エラー-10503「レコードの番号が範囲を越えています。」は、データベースの修復が必要であることを必ずしも意味している訳ではありません。このエラーは、ユーザがトランザクションで新しく作成されたレコード番号を使用しようとしたとき(例：**GOTO**

RECORDコマンド)に発生する可能性があります。この原因は、新しく作成されたレコードが、トランザクションの間には、トランザクションの妥当性が検査されるまで一時的なレコード番号を割り当てられるためです。このエラーがその状況で発生した場合には、データベースには何の障害もなく、障害があるのはアルゴリズムです。

4. エラー-9999「レコードを保存するディスクの領域が足りません。」は、データベースのすべてのセグメントに空き領域がなかったり、空き領域のないボリュームに配置されたときに発生します。また、データファイルがロックされていたり、ロックされたボリュームに配置されたときにも、このエラーは発生します。これによって、例えば、アプリケーションの「On Startup」データベースメソッド内からロックされたデータファイルを検出することができます。詳細については、後述の「データファイルのロックステータスのテスト」の節を参照してください。

ネットワークコンポーネントエラーコード

次の表は、ネットワークコンポーネントで発生するエラーです：

コード	エラーの起きた理由
-10001	データベースへの接続が中断されました。
-10002	このプロセスへの接続が中断されました。
-10003	接続パラメータが正しくありません。
-10020	OP Select 4D Server使用中に選択されたサーバがありません。
-10021	OP Find 4D Server使用中に選択されたサーバがありません。
-10050	Get/SetOption内のオプションが不明です。
-10051	Get/SetOption内の値が正しくありません。
-10130	この接続ステータスでは、このセッションに継続することはできません。。
-10131	接続が中止されました。
-10132	接続パラメータが無効です。
2	OP Select 4D Server使用中に「他...」ボタンがクリックされました。

システムエラーコード

この節は、それぞれの管理者のためのエラーコードを示します。

注：次のコードは、Macintoshのオペレーションシステムによって返されるエラーを基にしてます。このコードはWindows上でも同じように返されるので、いずれかのプラットフォーム上で同じエラーコードを検査することができます。

File Managerのエラー

次の表は、File Managerによって返されるエラーコードです。デスクトップファイル関係のコマンドを使用したときに発生します。

コード	エラーの起きた理由
-33	ファイルのディレクトリがいっぱいです。新しいファイルを作成することができません。
-34	ディスクがいっぱいです。ディスクには使用可能な空き要領がありません。
-35	ボリュームが存在しません。
-36	Ｉ／Ｏエラーです。ブロックに障害があります。
-37	ファイル名またはボリューム名が間違っています。
-38	オープンしてないファイルを読み書きしようとしました。
-39	読み込み中に論理的EOF(End Of File)がありました。
-40	ファイルの始め以前に移動しようとしました。
-41	新しいファイルをオープンするにはメモリが足りません。
-42	同時にオープンしたファイルが多すぎます。
-43	ファイルが見つかりません。
-44	ハードウェアによってボリュームが書き込み禁止になっています。
-45	ファイルがロックされています。
-46	ソフトウェアによってボリュームが書き込み禁止になっています。
-47	削除されてしまったファイルにアクセスしようとしました。
-48	すでに削除されてしまったファイルの名前をファイルに付けようとしました。
-49	すでにオープンされているファイルをオープンしようとしました。
-51	無効な文書参照番号で文書にアクセスしようとしました。
-52	ファイルマネージャ内部エラーです(ファイルマーカーの位置が見つかりません)。
-53	ボリュームがオンラインではありません。
-54	ロックされたファイルをオープンしようとしました。
-57	Macintosh用のディスクでないもので作業しようとしました。

コード	エラーの起きた理由
-58	Macintosh用のディスクでないもので作業しようとしました。
-60	ディスクが壊れています。
-61	書き込み禁止です。
-64	ディスクにハードウェア上の問題があります。(インストールやフォーマットが正しくない等)
-84	ディスクにハードウェア上の問題があります。(インストールやフォーマットが正しくない等)
-120	パス名に存在しないフォルダを指定して、ファイルをアクセスしようとしました。
-121	アクセスパスを作成することができませんでした。
-124	接続されていない共有ボリュームをアクセスしようとしました。

Memory Managerのエラー

次の表は、Memory Managerによって返されるエラーコードです。メモリが少ない時に印刷すると発生します。

コード	エラーの起きた理由
-108	ヒープ領域に空がありません。
-109	メモリ内部に問題があります。メモリが論理的に正しくありません。すぐに終了してください。マシンをリスタートさせ、データベースを再起動してください。
-117	メモリ内部に問題があります。メモリが論理的に正しくありません。すぐに終了してください。マシンをリスタートさせ、データベースを再起動してください。

プリントエラーコード

次の表は、Print Managerによって返されるエラーコードです。プリント中に発生します。

コード	エラーの起きた理由
-1	プリントファイルの保存中にエラーが起こりました。
-27	プリンタの接続時または接続解除時にエラーが起こりました。
-128	ユーザがプリンタを中断させました。
-193	リソースファイルが見つかりません。
-4100	プリンタ接続に割り込みを受けました。
-4101	プリンタがビジー(busy)であるか、あるいは接続されていません。

コード	エラーの起きた理由
-8150	Laser Writerが選択されていません。
-8151	このプリンタは、異なるバージョンのドライバで初期化されています。
-8192	Laser Writerが時間切れです。

Resource Managerのエラー

次の表は、Resource Managerによって返されるエラーコードです。

コード	エラーの起きた理由
-192	リソースがありません。
-193	リソースマップに障害があります。(ファイルを修復する必要があります。)

Macintosh SANE NaN Errorのコード

次の表に、AppleのSANE(Standard Apple Numeric Environment)によるNaN(Not a Number)コードを示します。NaNメッセージはSANEが扱える範囲を、演算結果が越えた場合に発生します。

NaNコード	NaNの起きた理由
1	不当な平方根(－１の平方根など)です。
2	不当な加減算(無限大+無限大など)です。
4	不当な割り算(０/０など)です。
8	不当な掛け算(０*無限大など)です。
9	不当な割り算の余り(０で割ったときの余りなど)です。
17	内容が間違ったASCII文字(列)を数値などに変換しようとしました。
20	Compタイプの数を浮動小数点に変換しようとしました。
21	NaNコードを０コードで作ろうとしました。
33	三角関数の引数が不当です。
34	逆三角関数の引数が不当です。
36	対数関数の引数が不当です。
37	指数関数の引数が不当です。
38	財務関数の引数が不当です。
255	記憶領域が初期化されていません。

Sound Managerのエラー

次の表は、Sound Managerによって返されるエラーコードです：

コード	エラーの起きた理由
-203	サウンドコマンドが多すぎます。
-204	サウンドリソースがロードできませんでした。
-205	サウンドチャンネルが論理的に間違ってます。
-206	サウンドリソースのフォーマットが間違っています。
-207	サウンドを実行するにはメモリが不十分です。
-209	サウンドチャンネルがビジーです。

シリアルポートのエラーコード

次の表は、シリアルポートアクセスによって返されるエラーコードです：

コード	エラーの起きた理由
-28	シリアルポートが開いていません。

システムエラー

次の表は、システムエラーのリストです。このエラーは一般に復旧不可能です：

コード	エラーの起きた理由
4	ゼロによる割り算が行なわれました。
15	セグメントロードエラーです。 4th Dimensionはコードセグメントをロードすることができませんでした。 4th Dimensionに十分なメモリを割り当ててください。
17～24	システムパッケージが見つかりません。 システムフォルダがインストールされているかどうかチェックしてください。
28	スタックがアプリケーションヒープの中に移動しました。 4th Dimensionに十分なメモリを確保してください。

データファイルのロック状態のテスト

データファイルまたはデータファイルが配置されているボリュームのロック状態をテストするには、オペレーションシステム（OS）のファイルマネージャをコールする必要があります。データファイルを変更する各データベース処理に対してこれを行うと、データベースエンジンのパフォーマンスに重大な影響を与えることでしょう。

あなた自身の責任のもとでワークセッションの開始時にこのロック状態をテストします。通常、あなたのデータベースの「On Startup」データベースメソッドの中でこれを行います。4D 3.2.5と4D Server1.2.5までのバージョンでは、データファイルのロック状態をテストするには、データファイルおよびそのデータファイルが配置されているボリュームのファイルマネージャ属性を返した外部ルーチンの使用は必要でした。しかし、今度の4Dと4D Server新規レコードを作成しようとする度にロックされたデータファイルに合図を送ることにより簡単にこのテストを行うことができます。データファイルまたはそれのボリュームがロックされている場合、データベースエンジンはエラーコード -9999（レコードを保存するディスクの領域が足りません。）を生成します。

次のコードは、データファイルのロック状態をテストする方法を示した例です：

```
` 「Is data locked」プロジェクトメソッド
` Is data locked → ブール
` Is data locked -> データファイルがロックまたはいっぱいの場合はTrue
gError:=0
ON ERR CALL("エラー処理")
CREATE RECORD([任意テーブル])
SAVE RECORD([任意テーブル])
ON ERR CALL("")
If (gError=0)
   DELETE RECORD([任意テーブル])
End if
$0:=(gError=-9999)
```

“ エラー処理 ” と名付けられた**ON ERR CALL**メソッドを次に示します：

```
`「エラー処理」プロジェクトメソッド
gError:=Error
```

すると、あなたのデータベースの「On Startup」データベースメソッドで次のように記述できます：

```
` 「On Startup」データベースメソッド
` ...
If (Is data locked)
   `データファイルがロックされていたり、データファイルがあるボリュームがロック
   `されていたりいっぱいの場合は、その状態を示すダイアログボックスを表示する。
   `どちらのケースでも、リードオンリー状態でデータファイルを使用したい場合がある
   `かもしれません。つまり、そのデータファイルがCD-ROMボリューム上に配置されて
   `いるかもしれません。この場合、そのダイアログボックスは「リードオンリーで使用
   `中」ボタンと「終了」ボタンを持っているかもしれません。
   If (bQuit=1)
      QUIT 4D `データベースを抜けて、ファインダレベルに戻ったかチェックする
   Else
      `データファイルがロックされていることを示すインタープロセスフラグをセットする
      <>gREADONLY:=True
      `起動時の実行処理を続ける
   End if
End if
```

ロックされたデータファイルを使ってデータベースの使用を許可する場合、そのデータベースの内容を変更するかもしれない操作へのアクセス権を制限するようにしてください。これを行うには、あなたのテスト関数を再度コールするか、または上記の例のインタープロセスフラグを管理します。例えば、あなたが顧客レコードを追加するプロセスを開始する「M_ADD_CUST」というプロジェクトメソッドを持っている場合、このメソッドが実行可能かどうかあなたは前もって知っています。もし、それを知らないと、そのプロセスを開始するのに役に立ちません。例えば、

```
` 「M_ADD_CUST」プロジェクトメソッド
If (Can write data)
   `先に進む
   <>vlPID_CUST:=New process(...;...;...)
   ` ...
End if
```

「Can write data」プロジェクトメソッドを次に示します：

```
` 「Can write data」プロジェクトメソッド
` Can write data -> ブール
` Can write data -> データファイルがロックまたはいっぱいの場合はTrue
$0:=True
If (<>gREADONLY) `または If (Is data locked)
   ALERT("この処理はリードオンリー状態のデータファイルでは実行されません。")
   $0:=False
End if
```

重要：4D 3.2.5と4D Server 1.2.5のデータベースエンジンのパフォーマンスを保護するには、データファイルの内容を修正するかもしれない各データベース操作中にデータファイルのロック状態をチェックしてはいけません。例えば、任意のトランザクションの中でレコードを追加している場合、そのトランザクションが有効になるまではデータファイルに手を加えられないことを覚えておいてください。任意のトランザクション内での変更処理中におけるデータのロック状態をテストすることにより、不要なオーバーヘッドのテストを追加することになります。従って、データファイルのロック状態は、任意のトランザクション内でない時の新規レコードを作成する場合にしかテストされません。これは、「ユーザ」モードにおける新規レコードの追加および読み込み処理を含みます。また、（新規レコードに使用される）ADD RECORDコマンドやSAVE RECORDコマンド、および（新規レコード作成する）ARRAY TO SELECTIONコマンドも含みます。しかし、これはトランザクション内における新規レコード作成処理、既存レコードの修正または削除処理は含みません。

注：データファイルのロック状態のテストは、データファイルへの書き込み中に発生するその他のI / Oエラーのテスト処理の続行を中止しません。

定義済み定数　D

この付録では、バージョン6の4th Dimensionであらかじめ定義されている定数を各項目別に示します。この定数は、「デザイン」モードの「エクスプローラ」ウインドウ上にある「定数」ページにあるものと同じです。

4D Environment

定数	タイプ	値
4D Client	倍長整数	4
4D Engine	倍長整数	1
4D First	倍長整数	6
4D Runtime	倍長整数	2
4D Runtime Classic	倍長整数	3
4D Server	倍長整数	5
4th Dimension	倍長整数	0
Demo Version	倍長整数	1
Full Version	倍長整数	0

ASCII Codes

定数	タイプ	値
ACK ASCII code	倍長整数	6
At sign	倍長整数	64
Backspace	倍長整数	8
BEL ASCII code	倍長整数	7
BS ASCII code	倍長整数	8
CAN ASCII code	倍長整数	24
Carriage return	倍長整数	13
CR ASCII code	倍長整数	13
DC1 ASCII code	倍長整数	17
DC2 ASCII code	倍長整数	18
DC3 ASCII code	倍長整数	19
DC4 ASCII code	倍長整数	20

ASCII Codes（続き）

定数	タイプ	値
DEL ASCII code	倍長整数	127
DLE ASCII code	倍長整数	16
Double quote	倍長整数	34
EM ASCII code	倍長整数	25
ENQ ASCII code	倍長整数	5
Enter	倍長整数	3
EOT ASCII code	倍長整数	4
ESC ASCII code	倍長整数	27
Escape	倍長整数	27
ETB ASCII code	倍長整数	23
ETX ASCII code	倍長整数	3
FF ASCII code	倍長整数	12
FS ASCII code	倍長整数	28
GS ASCII code	倍長整数	29
HT ASCII code	倍長整数	9
LF ASCII code	倍長整数	10
Line feed	倍長整数	10
NAK ASCII code	倍長整数	21
NBSP	倍長整数	202
NUL ASCII code	倍長整数	0
Period	倍長整数	46
Quote	倍長整数	39
RS ASCII code	倍長整数	30
SI ASCII code	倍長整数	15
SO ASCII code	倍長整数	14
SOH ASCII code	倍長整数	1
SP ASCII code	倍長整数	32
Space	倍長整数	32
STX ASCII code	倍長整数	2
SUB ASCII code	倍長整数	26
SYN ASCII code	倍長整数	22
Tab	倍長整数	9
US ASCII code	倍長整数	31
VT ASCII code	倍長整数	11

BLOB

定数	タイプ	値
C string	倍長整数	0
Compact compression mode	倍長整数	1
Extended real format	倍長整数	1
Fast compression mode	倍長整数	2
Is not compressed	倍長整数	0
Macintosh byte ordering	倍長整数	1
Macintosh double real format	倍長整数	2
Native byte ordering	倍長整数	0
Native real format	倍長整数	0
Pascal string	倍長整数	1
PC byte ordering	倍長整数	2
PC double real format	倍長整数	3
Text with length	倍長整数	2
Text without length	倍長整数	3

Clipboard

定数	タイプ	値
No such data in clipboard	倍長整数	-102
Picture data	文字列	PICT
Text data	文字列	TEXT

Colors

定数	タイプ	値
Black	倍長整数	15
Blue	倍長整数	6
Brown	倍長整数	13
Dark Blue	倍長整数	5
Dark Brown	倍長整数	10
Dark Green	倍長整数	9
Dark Grey	倍長整数	11
Green	倍長整数	8
Grey	倍長整数	14
Light Blue	倍長整数	7
Light Grey	倍長整数	12
Orange	倍長整数	2
Purple	倍長整数	4
Red	倍長整数	3
White	倍長整数	0
Yellow	倍長整数	1

Communications

定数	タイプ	値
Data bits 5	倍長整数	0
Data bits 6	倍長整数	2048
Data bits 7	倍長整数	1024
Data bits 8	倍長整数	3072
MacOS Printer Port	倍長整数	0
MacOS Serial Port	倍長整数	1
Parity Even	倍長整数	12288
Parity None	倍長整数	0
Parity Odd	倍長整数	4096
Protocol DTR	倍長整数	30
Protocol None	倍長整数	0
Protocol XONXOFF	倍長整数	20
Speed 115200	倍長整数	1022
Speed 1200	倍長整数	94
Speed 1800	倍長整数	62
Speed 19200	倍長整数	4
Speed 230400	倍長整数	1021
Speed 2400	倍長整数	46
Speed 300	倍長整数	380
Speed 3600	倍長整数	30
Speed 4800	倍長整数	22
Speed 57600	倍長整数	0
Speed 600	倍長整数	189
Speed 7200	倍長整数	14
Speed 9600	倍長整数	10
Stop bits One	倍長整数	16384
Stop bits One and a half	倍長整数	-32768
Stop bits Two	倍長整数	-16384

Database Engine

定数	タイプ	値
Is new record	倍長整数	-3
No current record	倍長整数	-1

Database Events

定数	タイプ	値
Delete Record Event	倍長整数	3
Load Record Event	倍長整数	4
Save Existing Record Event	倍長整数	2
Save New Record Event	倍長整数	1

Date Display Formats

定数	タイプ	値
Abbr Month Date	倍長整数	6
Abbreviated	倍長整数	2
Long	倍長整数	3
MM DD YYYY	倍長整数	4
MM DD YYYY Forced	倍長整数	7
Month Date Year	倍長整数	5
Short	倍長整数	1

Days and Months

定数	タイプ	値
April	倍長整数	4
August	倍長整数	8
December	倍長整数	12
February	倍長整数	2
Friday	倍長整数	6
January	倍長整数	1
July	倍長整数	7
June	倍長整数	6
March	倍長整数	3
May	倍長整数	5
Monday	倍長整数	2
November	倍長整数	11
October	倍長整数	10
Saturday	倍長整数	7
September	倍長整数	9
Sunday	倍長整数	1
Thursday	倍長整数	5
Tuesday	倍長整数	3
Wednesday	倍長整数	4

Events (Modifiers)

定数	タイプ	値
Activate window bit	倍長整数	0
Activate window mask	倍長整数	1
Caps Lock key bit	倍長整数	10
Caps Lock key mask	倍長整数	1024
Command key bit	倍長整数	8
Command key mask	倍長整数	256
Control key bit	倍長整数	12
Control key mask	倍長整数	4096
Mouse button bit	倍長整数	7
Mouse button mask	倍長整数	128
Option key bit	倍長整数	11
Option key mask	倍長整数	2048
Right control key bit	倍長整数	15
Right control key mask	倍長整数	32768
Right option key bit	倍長整数	14
Right option key mask	倍長整数	16384
Right shift key bit	倍長整数	13
Right shift key mask	倍長整数	8192
Shift key bit	倍長整数	9
Shift key mask	倍長整数	512

Events (What)

定数	タイプ	値
Activate event	倍長整数	8
Auto key event	倍長整数	5
Disk event	倍長整数	7
Key down event	倍長整数	3
Key up event	倍長整数	4
Mouse down event	倍長整数	1
Mouse up event	倍長整数	2
Null event	倍長整数	0
Operating system event	倍長整数	15
Update event	倍長整数	6

Expressions

定数	タイプ	値
MAXINT	倍長整数	32767
MAXLONG	倍長整数	2147483647
MAXTEXTLEN	倍長整数	32000

Field and Variable Types

定数	タイプ	値
Array 2D	倍長整数	13
Boolean array	倍長整数	22
Date array	倍長整数	17
Integer array	倍長整数	15
Is Alpha Field	倍長整数	0
Is BLOB	倍長整数	30
Is Boolean	倍長整数	6
Is Date	倍長整数	4
Is Integer	倍長整数	8
Is LongInt	倍長整数	9
Is Picture	倍長整数	3
Is Pointer	倍長整数	23
Is Real	倍長整数	1
Is String Var	倍長整数	24
Is Subtable	倍長整数	7
Is Text	倍長整数	2
Is Time	倍長整数	11
Is Undefined	倍長整数	5
LongInt array	倍長整数	16
Picture array	倍長整数	19
Pointer array	倍長整数	20
Real array	倍長整数	14
String array	倍長整数	21
Text array	倍長整数	18

Find window

定数	タイプ	値
In contents	倍長整数	3
In drag	倍長整数	4
In go away	倍長整数	6
In grow	倍長整数	5
In menu bar	倍長整数	1
In system window	倍長整数	2
In zoom box	倍長整数	8

Font Styles

定数	タイプ	値
Bold	倍長整数	1
Condensed	倍長整数	32
Extended	倍長整数	64
Italic	倍長整数	2
Outline	倍長整数	8
Plain	倍長整数	0
Shadow	倍長整数	16
Underline	倍長整数	4

Form Events

定数	タイプ	値
On Activate	倍長整数	11
On Clicked	倍長整数	4
On Close Box	倍長整数	22
On Close Detail	倍長整数	26
On Data Change	倍長整数	20
On Deactivate	倍長整数	12
On Display Detail	倍長整数	8
On Double Clicked	倍長整数	13
On Drag Over	倍長整数	21
On Drop	倍長整数	16
On External Area	倍長整数	19
On Getting Focus	倍長整数	15
On Keystroke	倍長整数	17
On Load	倍長整数	1
On Losing Focus	倍長整数	14
On Menu Selected	倍長整数	18
On Open Detail	倍長整数	25
On Outside Call	倍長整数	10
On Printing Break	倍長整数	6
On Printing Detail	倍長整数	23
On Printing Footer	倍長整数	7
On Printing Header	倍長整数	5
On Unload	倍長整数	24
On Validate	倍長整数	3

Function Keys

定数	タイプ	値
Backspace Key	倍長整数	8
Down Arrow Key	倍長整数	31
End Key	倍長整数	4
Enter Key	倍長整数	3
Escape Key	倍長整数	27
F1 Key	倍長整数	-122
F10 Key	倍長整数	-109
F11 Key	倍長整数	-103
F12 Key	倍長整数	-111
F13 Key	倍長整数	-105
F14 Key	倍長整数	-107
F15 Key	倍長整数	-113
F2 Key	倍長整数	-120
F3 Key	倍長整数	-99
F4 Key	倍長整数	-118
F5 Key	倍長整数	-96
F6 Key	倍長整数	-97
F7 Key	倍長整数	-98
F8 Key	倍長整数	-100
F9 Key	倍長整数	-101
Help Key	倍長整数	5
Home Key	倍長整数	1
Left Arrow Key	倍長整数	28
Page Down Key	倍長整数	12
Page Up Key	倍長整数	11
Return Key	倍長整数	13
Right Arrow Key	倍長整数	29
Tab Key	倍長整数	9
Up Arrow Key	倍長整数	30

Hierarchical Lists

定数	タイプ	値
Ala Macintosh	倍長整数	1
Ala Windows	倍長整数	2
Macintosh node	倍長整数	860
Use PicRef	倍長整数	131072
Use PICT resource	倍長整数	65536
Windows node	倍長整数	138

ISO Latin Character Entities

定数	タイプ	値
ISO L1 a acute	文字列	á
ISO L1 a circumflex	文字列	â
ISO L1 a grave	文字列	à
ISO L1 a ring	文字列	å
ISO L1 a tilde	文字列	ã
ISO L1 a umlaut	文字列	ä
ISO L1 ae ligature	文字列	æ
ISO L1 Ampersand	文字列	&
ISO L1 c cedilla	文字列	ç
ISO L1 Cap A acute	文字列	Á
ISO L1 Cap A circumflex	文字列	Â
ISO L1 Cap A grave	文字列	À
ISO L1 Cap A ring	文字列	Å
ISO L1 Cap A tilde	文字列	Ã
ISO L1 Cap A umlaut	文字列	Ä
ISO L1 Cap AE ligature	文字列	&AELig;
ISO L1 Cap C cedilla	文字列	Ç
ISO L1 Cap E acute	文字列	É
ISO L1 Cap E circumflex	文字列	Ê
ISO L1 Cap E grave	文字列	È
ISO L1 Cap E umlaut	文字列	Ë
ISO L1 Cap Eth Icelandic	文字列	Ð
ISO L1 Cap I acute	文字列	Í
ISO L1 Cap I circumflex	文字列	Î
ISO L1 Cap I grave	文字列	Ì
ISO L1 Cap I umlaut	文字列	Ï
ISO L1 Cap N tilde	文字列	Ñ
ISO L1 Cap O acute	文字列	Ó
ISO L1 Cap O circumflex	文字列	Ô
ISO L1 Cap O grave	文字列	Ò
ISO L1 Cap O slash	文字列	Ø
ISO L1 Cap O tilde	文字列	Õ
ISO L1 Cap O umlaut	文字列	Ö
ISO L1 Cap THORN Icelandic	文字列	Þ
ISO L1 Cap U acute	文字列	Ú

ISO Latin Character Entities（続き）

定数	タイプ	値
ISO L1 Cap U circumflex	文字列	Û
ISO L1 Cap U grave	文字列	Ù
ISO L1 Cap U umlaut	文字列	Ü
ISO L1 Cap Y acute	文字列	Ý
ISO L1 Copyright	文字列	©
ISO L1 e acute	文字列	é
ISO L1 e circumflex	文字列	ê
ISO L1 e grave	文字列	è
ISO L1 e umlaut	文字列	ë
ISO L1 eth Icelandic	文字列	ð
ISO L1 Greater than	文字列	>
ISO L1 i acute	文字列	í
ISO L1 i circumflex	文字列	î
ISO L1 i grave	文字列	ì
ISO L1 i umlaut	文字列	ï
ISO L1 Less than	文字列	<
ISO L1 n tilde	文字列	ñ
ISO L1 o acute	文字列	ó
ISO L1 o circumflex	文字列	ô
ISO L1 o grave	文字列	ò
ISO L1 o slash	文字列	ø
ISO L1 o tilde	文字列	õ
ISO L1 o umlaut	文字列	ö
ISO L1 Quotation mark	文字列	"
ISO L1 Registered	文字列	®
ISO L1 sharp s German	文字列	ß
ISO L1 thorn Icelandic	文字列	þ
ISO L1 u acute	文字列	ú
ISO L1 u circumflex	文字列	û
ISO L1 u grave	文字列	ù
ISO L1 u umlaut	文字列	ü
ISO L1 y acute	文字列	ý
ISO L1 y umlaut	文字列	ÿ

Math

定数	タイプ	値
Degree	実数	0.0174532925199432958
e number	実数	2.71828182845904524
Pi	実数	3.141592653589793239
Radian	実数	57.29577951308232088

Open window

定数	タイプ	値
Alternate dialog box	倍長整数	3
Has grow box	倍長整数	4
Has highlight	倍長整数	1
Has window title	倍長整数	2
Has zoom box	倍長整数	8
Modal dialog box	倍長整数	1
Movable dialog box	倍長整数	5
Palette window	倍長整数	720
Plain dialog box	倍長整数	2
Plain fixed size window	倍長整数	4
Plain no zoom box window	倍長整数	0
Plain window	倍長整数	8
Round corner window	倍長整数	16

Picture Display Formats

定数	タイプ	値
On Background	倍長整数	3
Scaled to Fit	倍長整数	2
Scaled to fit prop centered	倍長整数	6
Scaled to fit proportional	倍長整数	5
Truncated Centered	倍長整数	1
Truncated non Centered	倍長整数	4

Platform Interfaces

定数	タイプ	値
Automatic interface	倍長整数	-1
Copland interface	倍長整数	3
Macintosh interface	倍長整数	0
Windows 3.1 interface	倍長整数	1
Windows 95 interface	倍長整数	2

Platform Properties

定数	タイプ	値
INTEL 386	倍長整数	386
INTEL 486	倍長整数	486
Macintosh 68K	倍長整数	1
Pentium	倍長整数	586
Power Macintosh	倍長整数	2
PowerPC 601	倍長整数	601
PowerPC 603	倍長整数	603
PowerPC 604	倍長整数	604
Windows	倍長整数	3

Process state

定数	タイプ	値
Aborted	倍長整数	-1
Delayed	倍長整数	1
Does not exist	倍長整数	-100
Executing	倍長整数	0
Hidden modal dialog	倍長整数	6
Paused	倍長整数	5
Waiting for input output	倍長整数	3
Waiting for internal flag	倍長整数	4
Waiting for user event	倍長整数	2

Query Destinations

定数	タイプ	値
Into current selection	倍長整数	0
Into named selection	倍長整数	2
Into set	倍長整数	1
Into variable	倍長整数	3

Resources Properties

定数	タイプ	値
Changed resource bit	倍長整数	1
Changed resource mask	倍長整数	2
Locked resource bit	倍長整数	4
Locked resource mask	倍長整数	16
Preloaded resource bit	倍長整数	2
Preloaded resource mask	倍長整数	4
Protected resource bit	倍長整数	3
Protected resource mask	倍長整数	8
Purgeable resource bit	倍長整数	5
Purgeable resource mask	倍長整数	32
System heap resource bit	倍長整数	6
System heap resource mask	倍長整数	64

SCREEN DEPTH

定数	タイプ	値
Black and white	倍長整数	0
Four colors	倍長整数	2
Is color	倍長整数	1
Is gray scale	倍長整数	0
Millions of colors 24 bit	倍長整数	24
Millions of colors 32 bit	倍長整数	32
Sixteen colors	倍長整数	4
Thousands of colors	倍長整数	16
Two fifty six colors	倍長整数	8

SET RGB COLOR

定数	タイプ	値
Default background color	倍長整数	-2
Default dark shadow color	倍長整数	-3
Default foreground color	倍長整数	-1
Default light shadow color	倍長整数	-4

Standard System Signatures

定数	タイプ	値
Picture Document	文字列	PICT
QT Animation compressor	文字列	rle
QT Compact video compressor	文字列	cdvc
QT Graphics compressor	文字列	smc
QT Photo compressor	文字列	jpeg
QT Raw compressor	文字列	raw
QT Video compressor	文字列	rpza
Text Document	文字列	TEXT
Windows MIDI Document	文字列	MI D
Windows Sound Document	文字列	WAV
Windows Video Document	文字列	AVI

TCP Port Numbers

定数	タイプ	値
TCP Authentication	倍長整数	113
TCP DNS	倍長整数	53
TCP Finger	倍長整数	79
TCP FTP Control	倍長整数	21
TCP FTP Data	倍長整数	20
TCP Gopher	倍長整数	70
TCP HTTP WWW	倍長整数	80
TCP IMAP3	倍長整数	220
TCP Kerberos	倍長整数	88
TCP KLogin	倍長整数	543
TCP Nickname	倍長整数	43
TCP NNTP	倍長整数	119
TCP NTalk	倍長整数	518
TCP NTP	倍長整数	123
TCP PMCP	倍長整数	1643
TCP PMD	倍長整数	1642
TCP POP3	倍長整数	110
TCP Printer	倍長整数	515
TCP RADACCT	倍長整数	1646
TCP RADIUS	倍長整数	1645
TCP Remote Cmd	倍長整数	514
TCP Remote Exec	倍長整数	512
TCP Remote Login	倍長整数	513
TCP Router	倍長整数	520
TCP SMTP	倍長整数	25
TCP SNMP	倍長整数	161
TCP SNMPTRAP	倍長整数	162
TCP SUN RPC	倍長整数	111
TCP Talk	倍長整数	517
TCP Telnet	倍長整数	23
TCP TFTP	倍長整数	69
TCP UUCP	倍長整数	540
TCP UUCP RLOGIN	倍長整数	541

Test path name

定数	タイプ	値
Is a directory	倍長整数	0
Is a document	倍長整数	1

Time Display Formats

定数	タイプ	値
HH MM	倍長整数	2
HH MM AM PM	倍長整数	5
HH MM SS	倍長整数	1
Hour Min	倍長整数	4
Hour Min Sec	倍長整数	3

Window kind

定数	タイプ	値
External window	倍長整数	5
Floating window	倍長整数	14
Modal dialog	倍長整数	9
Regular window	倍長整数	8

テーマ別索引

コマンド索引

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

その他