
CS
TEKLYNX
CODESOFT
TEKLYNX
CODESOFT
Barcode Better
powered by:
TEKLYNX
Barcode Better

チュートリアル

目次

本マニュアルについて

表記に関する原則

本マニュアルは、以下の原則を用いて、さまざまなタイプの情報を区別しています。

- コマンドなど、インターフェース自体の用語は、太字で表記します。
- キーボードのキーは細字の大文字を四角で囲んで表記します。

  例：「SHIFT キーを押してください。」
- 番号のついたリストは、実行する手順があることを示します。
- 接続詞「または」が段落の隣に表示された場合、特定のタスクを実行する別の手順があることを表します。
- メニューコマンドにサブメニューが含まれる場合、選択するメニュー名と、それに続くコマンドが太字で表記されます。

  したがって、「ファイル > 開くを選択」の場合、ファイル ニューを選び、開く コマンドを選択します。

製品について

このマニュアルは、特定の製品について記載されたものではなく、製品間で共通の概念について説明しています。従って、記載されている一部の機能は、

製品によってはご利用いただけない場合があります。

購入されたソフトウェアで利用できる機能については、製品のオンラインヘルプやカタログをご覧ください。

データベースへの接続

注意事項

本章では、ラベル（コンテナ）とデータベース（コンテンツ）の関連付けを行います。 ODBC （Open DataBase Connectivity） または OLE DB 接続を使用して、これを行います。

データベースを使用すると、データを保管することができます。すべてのデータは、リレーションシップと呼ばれる、2 次元のテーブルにまとめられます。テーブルのそれぞれの行はレコードと呼ばれます。レコードは、フィールド形式で、テーブルのさまざまな列に配置されるプロパティである、オブジェクトの管理を目的としています。

データベースには多くのテーブルが含まれます。特定のデータベース内で、さまざまなテーブルを関連付けるには、結合を使用します。本章の後で、結合を作成する方法について具体的な例をご覧いただきます。

ODBC

ODBC データソースを使用すると、幅広いデータベース管理システムに属するデータにアクセスすることができます。ODBC は、ラベルデザイン・ソフトウェアなどのアプリケーションと、特定の数のデータベースを簡単にリンクさせることができます。ソフトウェアには、いくつかのODBC ライバーが搭載されています。

これらのドライバーによって、最もよくあるタイプのデータベースにアクセスすることができます。

ドライバーの一部をご紹介しましょう。

- Microsoft Access Driver (*.mdb)
- Microsoft Excel Driver (*.xls)
- Microsoft FoxPro Driver (*.dbf)

OLE DB

OLE DB は、種類、フォーマット、場所に関わらず、すべてのデータにアクセス可能とするインターフェースのセットです。

それは、アクセス・インターフェース、クエリー・ドライバーなどのコンポーネントを提供します。これらのコンポーネントは、「プロバイダー」と呼ばれます。

以下の例では、データベースがソフトウェアに接続されていない場合の接続プロセスを説明します。

ODBCデータソースのインストール

以下に記載したプロセスでは、直接作成モードを使用します。必要に応じて、操作状況にあわせて表示されるメニューでWizard（ウィザード）を選択し、これを使用することができます。

TKTraining.mdb データベースへの接続

1. ツール > ODBCアドミニストレーターを選択します。
2. システムDSN タブ をクリックして、追加ボタンをクリックします。

注意：システム・データソース名（DSN）を用いて、データソースを定義することができます。これらのデータソースは、特定のコンピューター固有のものですが、特定のユーザーに限定されるわけではありません。必要な権利を持つすべてのユーザーが、システムDSN アクセスできます。

3. Microsoft Access Driver を選択し、完了ボタンをクリックします。
4. データソース名ボックスに「TK Training CS Level 2」を入力します。
5. 選択ボタンをクリックして、 InstallDir\Samples\Forms\Tutorial.にある TKTraining.mdbデータベースを選択します。
6. オプション ボタンをクリックします。 読み取り専用オプションにチェックをつけます。このオプションを使用すると、読み取り/書き込みの問題を生じることなく、と同時にデータベースを開くことができます。
7. ODBC Microsoft Access セットアップのダイアログボックスで、OK ボタンをクリックします。

データのインポート

データベースがソフトウェアに接続されたら、それをドキュメントに接続しなければなりません。

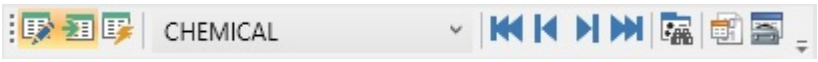
1. PRODUCT_WS3 ラベルを開きます。
2. データソース > データベース > クエリーの作成と修正を選択します。
3. データソース選択リストで、 TK Training CS Level 2 を選択します。
4. 選択テーブルリストで、「Fruits」を選択します。データベース・フィールドが、選択フィールドリストに表示されます。
5. 「ProdName」、「Origin」、「Weight」および「Reference」フィールドを選択します。

6.  ボタンをクリックします。

   選択されたレコードを、アルファベットまたは数字、昇順あるいは降順にソートすることができます。

7. ソートキーとして「Reference」、ソート順として「昇順」を選択します。
8. このクエリーをPRODUCT_WS4_ODBC.CSQ として保存します。
9. OK ボタンをクリックします。 変数が自動的に作成され、データソースビューのデータベースのブランチにリストされます。

オブジェクトがとることができる値を表示または印刷するには、ナビゲーションバーを使用します。クエリー結果ウィンドウから印刷することもできます。



変数オブジェクトの作成

1. データソースビューのデータベースのブランチにリストされた、作成された変数を選択し、ワークスペースにドラッグ＆ドロップします。
2. 表示されるコンテキストメニューにおいて、テキストを選択します。

Active Query Builder

SQL クエリービルダー

クエリービルダーは、直感的で使いやすいビジュアルなクエリー構築インターフェースを介して、複雑な SQL クエリーを構築できるビジュアルなコンポーネントです。

クエリービルダーを取り扱うには、SQL の概念に関する基本知識が必要です。クエリービルダーを使用すると、専門的で詳細な知識がなくても、正しいSQLコードを記述することができます。SQL 原則を理解していれば、望ましい結果を得ることができるでしょう。

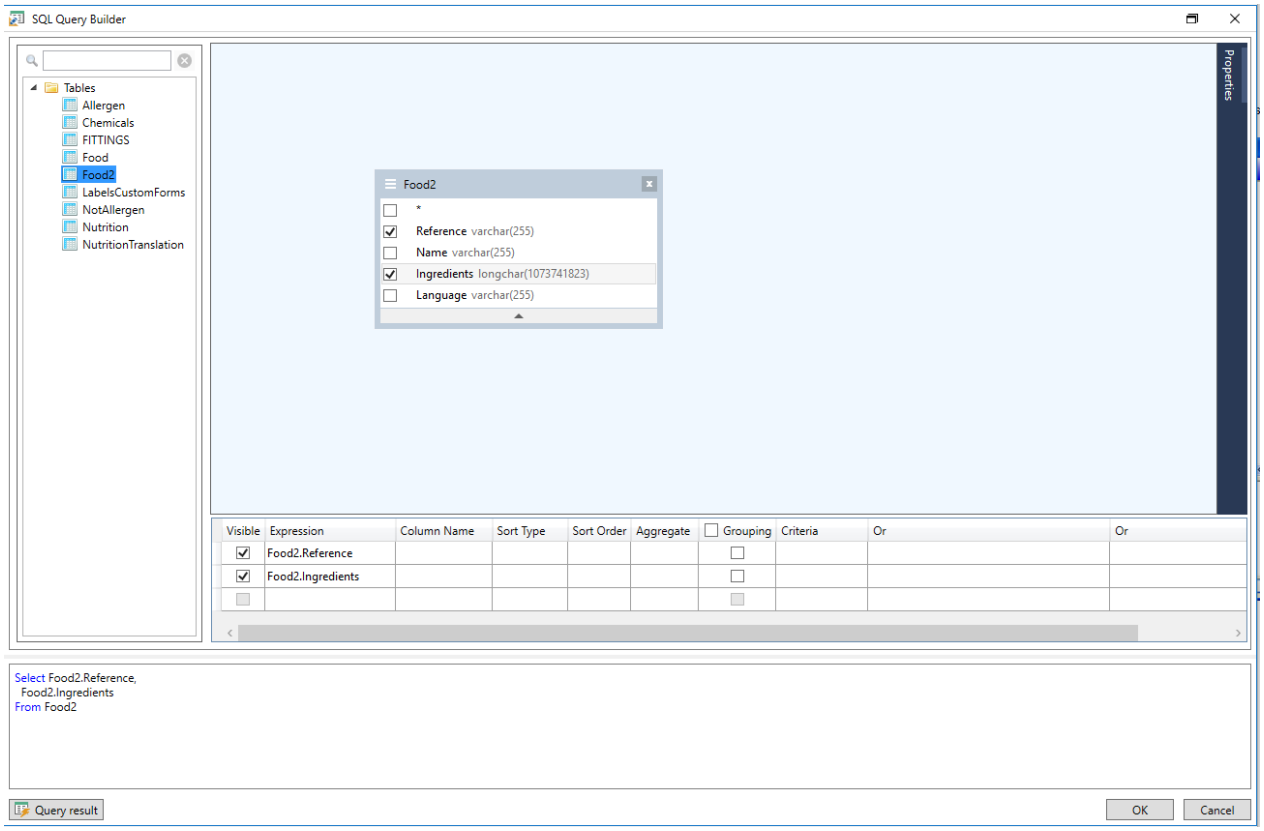
起動

クエリービルダーを起動すると、以下の要素が表示されます。

メインウィンドウは、以下の部分に分類されます。

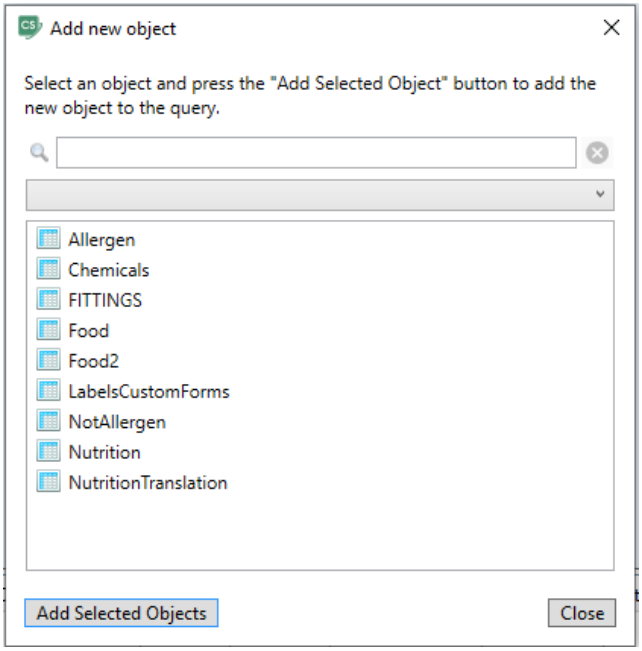
- クエリー構築エリア は、クエリーがビジュアルに表示されるメインエリアです。このエリアを使用すると、ソースデータベース オブジェクトと派生するテーブルの定義やそれらのリンクの定義、テーブルやリンクのプロパティを設定することができます。
- カラムペインは、クエリー構築エリアの下にあります。これを使用すると、クエリー出力カラムや式に関する、必要なすべてのオペレーションを行うことができます。こちらで、フィールド別名の定義、ソートやグループ化、条件の定義を行うことができます。

- テーブルビューエリアは左にあります。ここで、クエリを検索して、すぐにいずれかの部分を見つけることができます。検索の指定には、ウィンドウペインの一番上の検索フィールドを使用します。
- クエリー構築エリアの上にあるページコントロールを使用すると、メインクエリーとサブクエリーを切り替えることができます。
- クエリー構築エリアの端にある、Qの文字がついた小さなエリアは、結合サブクエリーの取り扱いをコントロールします。こちらで、新しい結合サブクエリーを追加したり、それらに関する必要なオペレーションをすべて行うことができます。



オブジェクトをクエリーに追加

オブジェクトをクエリーに追加するには、クエリー構築エリアを右クリックし、ドロップダウンメニューでオブジェクトの追加を選択します。



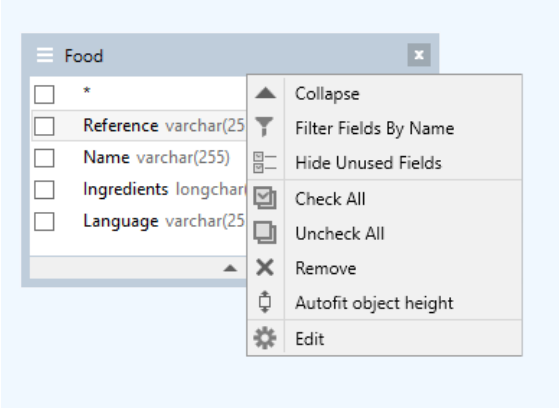
新しいオブジェクトを追加ウィンドウでは、複数のオブジェクトを一度に追加できます。テーブルの工レメントは、コンボボックスにリストされています。**オブジェクトをすべて表示**フィルタでは、選択可能なオブジェクトすべて表示します。ウィンドウの一番上の**検索フィールド**は、必要なオブジェクトを

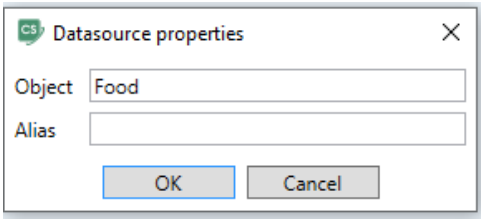
見つけることができます。1つあるいは**CTRLキー**を長押しして複数のオブジェクトを選択してから、**選択したオブジェクトを追加** ボタンを押して、クエリにこれらのオブジェクトを追加できます。この操作は数回繰り返すこともできます。オブジェクトをすべて追加したら、**閉じる**ボタンをクリックして、このウィンドウを非表示にします。

クエリからオブジェクトを削除するためには、オブジェクトヘッダーの**閉じる**ボタンをクリックします。

オブジェクト・プロパティの編集

オブジェクトを右クリックし、ドロップダウンメニューから編集を選択するか、オブジェクトのヘッダーをダブルクリックすると、クエリーに追加された各オブジェクトのプロパティを変更することができます。

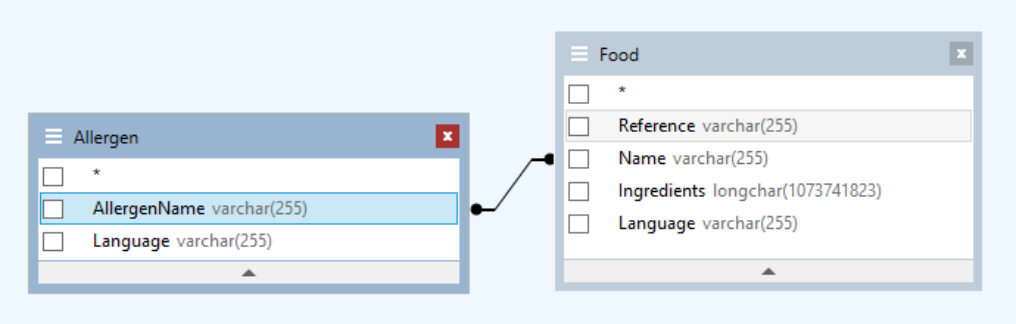




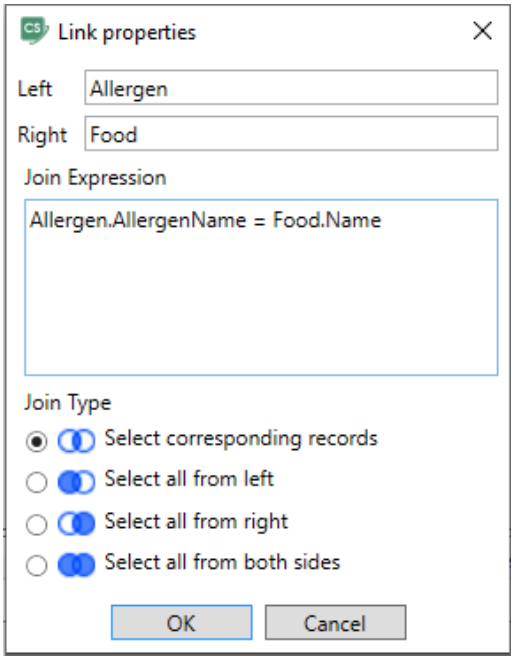
データソース· プロパティ ダイアログは、サーバーによって異なる場合がありますが、 少なくとも別名プロパティはすべてのデータベース· サーバーで同じです。

テーブルの結合

2つのオブジェクト間でリンクを作成（すなわち、結合）するには、オブジェクトのリンクしたいフィールドを選択し、それを、もうひとつのオブジェクトの対応するフィールドにドラッグします。ドラッグを終えると、リンクされたフィールド間を結ぶラインが表示されます。



デフォルトで作成される結合タイプは、INNER JOIN です。つまり、両方のテーブルで一致するレコードのみが、結果のデータセットに含まれます。他の結合タイプを定義するには、リンクを右クリックし、ドロップダウンメニューから編集を選択するか、またはそれをダブルクリックして、リンクプロパティダイアログを開きます。このダイアログを使用すると、結合タイプや他のリンクプロパティを定義することができます。



オブジェクト間のリンクを削除するためには、リンクの線を右クリックして、ドロップダウンメニューの**削除**オプションを選択する、またはリンクの線を選択してから、**削除**ボタンを押します。

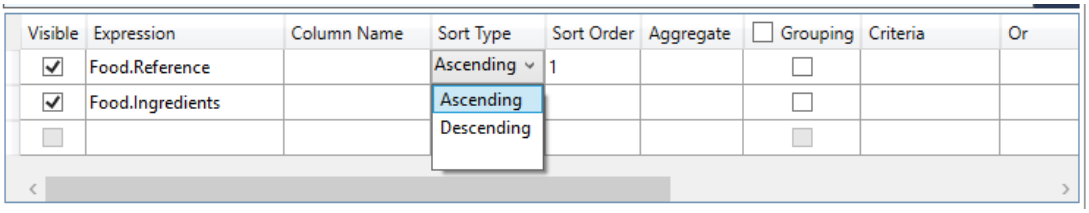
出力フィールドのソート

出力クエリーフィールドのソートを有効にするには、カラムペインのソートタイプ や ソート順を使用します。

ソートタイプのカラムを使用すると、フィールドのソート方法を、昇順または降順に指定することができます。

ソート順 のカラムを使用すると、1 つ以上のフィールドをソートする場合、フィールドのソート順を設定することができます。

特定のフィールドのソートを無効にするには、このフィールドでソートタイプをクリア (空) にします。



条件の定義

カラムペイン にリストされる式に関する条件を定義するには、条件の列を使用します。

ここで、式自体を省略する条件を記述します。クエリーで下記の条件を取得す

る場合

WHERE (field >= 10) AND (field <= 20)

条件 に次のように記述します。

>= 10 AND <= 20

基準カラムのドロップダウンリストを使用して、編集フィールドに共通式や演算子を貼りつけることもできます。高度な基準作成には、基準カラムのエクスプレッションエディタボタンを使用します。エクスプレッションエディタダイアログが表示されます。

できます。これらの条件は、OR 演算子を用いて、クエリーに関連付けられます。

パラメータ化したクエリーの規定

クエリービルダー そのパラメータの値が変わり易いままで保持される、パラメーター化されたクエリーを作成させます。

注： 前もって変数の作成が済んでいる必要があります。

1. 作成されるクエリーが実行されることになる表をドラッグおよびドロップします。
2. 判断基準が適用されることになるフィールドを選択します。
3. 判断基準 カラムまたはSQLフォーマットに編集フィールドにおいて検索判断基準のオブジェクトとして使用される変数を規定します。

例：前もって作成された変数Var0 の値を探すには、

- **SQLにおいて。**

[表]から

[表]を選択します。*

ここで、[Table].フィールド = アプリケーション.ドキュメント.Var0

- 判断基準カラム

  = アプリケーション.ドキュメント.Var0

4. 「**クエリ結果**」ボタンをクリックして、クエリの結果を表示します。

クエリー結果グリッド

クエリー結果グリッドにアクセスするには、クエリーの定義ダイアログボックスまたはデータベースブラウザへマージツールバー、あるいはデータソース >データベースメニューにより  ボタンをクリックします。

このグリッドを使用すると、クエリーの結果を表示したり、特定の条件とすべての結果を検索し、対応するラベルを印刷したりすることができます。

ご注意: “<Binary data>”タグは、BLOBデータ型 (例えば画像ファイル) を含むデータベースフィールドのクエリ結果ダイアログで表示されます。

クエリーの結果グリッドには、以下の機能が含まれます。

- **検索機能** 

  検索を実行するフィールドを選択できる、検索フィールド

  検索する値を入力できる、検索データ

  フィールドの任意の場所または最初にある値を検索 

- クエリー結果レコードをブラウズする、ナビゲーション機能

  最初のレコード 

  前のレコード 

  次のレコード 

  最後のレコード 

- **結果グリッド**

  クエリーの実行結果を表示します。

- **再クエリー**

  リクエストを再度照会し、グリッドを更新します。

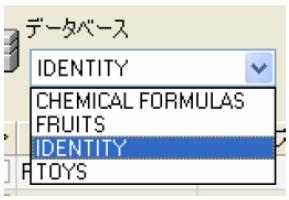
データベースマネージャー

データベース構造ウィンドウ

データベース構造ウィンドウを使用すると、テーブル/フィールドの追加、変更または削除など、データベースのファイル構造を管理することができます。

接続のリストからデータベースを選択

1. データベースのドロップダウンリストをクリックします。
2. 目的のデータベースをクリックします。

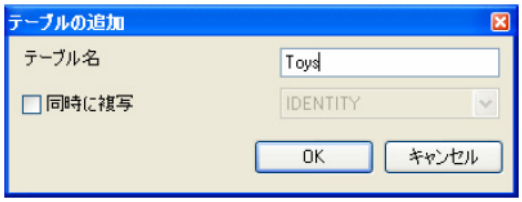


データベースのテーブルを選択

1. テーブルのドロップダウンリストをクリックします。
2. 目的のテーブルをクリックします。

現在のデータベースにテーブルを追加

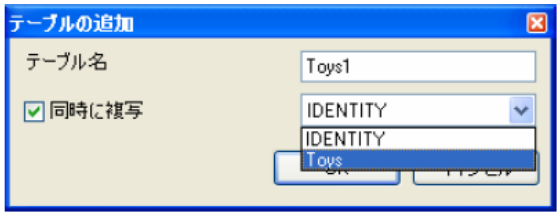
1. テーブルの追加ボタンをクリックします。
2. 新しいテーブルの名前を入力します。
3. OK ボタンをクリックします。



選択されたデータベースにすでに存在するテーブルから、テーブルの構造をコピーすることもできます。

この操作方法は以下のとおりです。

1. 「同時に複写」の隣にあるボックスにチェックをつけます。
2. ドロップダウンリストをクリックします
3. 必要なデータをクリックします。
4. OK ボタンをクリックします。



現在のデータベースからテーブルを削除

1. テーブルのドロップダウンリストをクリックします。
2. 目的のテーブルをクリックします。
3. テーブルの削除ボタンをクリックします。

現在のテーブルのデータを表示/非表示

1. データ表示ボタンをクリックします。

キーフィールドの定義

1. 必要なフィールドの隣にあるボックスにチェックをつけます。

2. 適用ボタンをクリックします。

フィールドの内容タイプを定義

1. 対象フィールドのタイプカラムをクリックします。
2. ドロップダウンリストのボタンをクリックします。
3. 希望するデータタイプをクリックします。

4. 適用ボタンをクリックします。

フィールドの最大サイズを定義

1. 対象フィールドの長さカラムをクリックします。
2. 希望する長さを入力します。
3. 適用ボタンをクリックします。

空フィールドを許可

1. 対象フィールドの「Null を認める」カラムにチェックをつけます。
2. 適用ボタンをクリックします。

データベース修正ウィンドウ

データベース修正ウィンドウを使用すると、データの追加、変更または削除など、データベースファイルの内容を管理することができます。

データベース修正ウィンドウを使用すると、データの追加、変更または削除など、データベースファイルの内容を管理することができます。

内容によりレコードを選択

フィールドの内容を使用して、レコードを検索します。

1. 検索フィールドの選択ドロップダウンリストのボタンをクリックします。
2. 目的のフィールドをクリックします。
3. データ入力フィールドをクリックします。

4. データ入力フィールドに検索する値を入力します。

一致するレコードをすべて選択

1 つ以上のレコードが表示されていなければなりません。

1. 検索フィールドの選択ドロップダウンリストのボタンをクリックします。
2. 目的のフィールドをクリックします。
3. データ入力フィールドをクリックします。
4. データ入力フィールドに検索する値を入力します。
5. 全てを選択 ()ボタンをクリックします。

一致するレコードを選択

1 つ以上のレコードが表示されていなければなりません。また、検索フィールドにいくつかの一致する内容があります。

レコードを選択するには、検索ツールを使用します。 1 (先頭レコード) 、2 (前レコード) 、3 (次レコード) または 4 (最終レコード) をクリックします。

新しいレコードの作成

1. アスタリスク (*) が付いた行のフィールドをクリックします。
2. 対応するフィールドに必要な値を入力します。
3. 適用ボタンをクリックします。

レコードの変更

1. 変更したいデータをクリックします。
2. 必要なデータを入力します。
3. 適用ボタンをクリックします

レコードの削除

1. 目的のレコードのデータベースのカーソルをクリックします。
2. 右クリックします。
3. コンテキストメニューで、選択レコードの削除をクリックします。

データベースクエリーウィンドウ

データベースクエリーウィンドウを使用すると、さまざまなフィルターを作成し、適用することができます。

クエリーの追加

1. クエリーの追加ボタンをクリックします。
2. クエリーの名前を入力します。
3. OK ボタンをクリックします。

１つまたは複数のフィールドを選択/選択解除

1. ナビゲーションツール  を使用します。
2. クエリーボタンをクリックし、データベースプレビューを更新します。

選択したフィールドの順序を変更

1. 並べ替えフィールドリストで目的のフィールドをクリックします。
2. 上下の矢印ボタンをクリックして、フィールドの順序を調整します。
3. クエリーボタンをクリックし、データベースプレビューを更新します。

事前に定義されたデータを使用し、フィルターを作成

1. フィルター タブを選択します。
2. レコードの追加ボタン()をクリックします。
3. フィールドにおいて、ドロップダウンリストから希望するフィールドを選択します。
4. 演算子 フィールドにおいて、ドロップダウンリストから希望する演算子を選択します。
5. 値 フィールドにおいて、条件となる値を入力します。
6. クエリー ボタンをクリックし、結果を表示します。

論理演算子を複数のフィルターに適用

1. レコードの追加ボタン ()をクリックします。
2. 論理式 フィールドにおいて、ドロップダウンリストからAND または OR を選択します。
3. フィルターを作成します (上記に記載)。
4. クエリー ボタンをクリックし、結果を表示します。

フィルターの削除

注意：1 つ以上のフィルターが必要です。

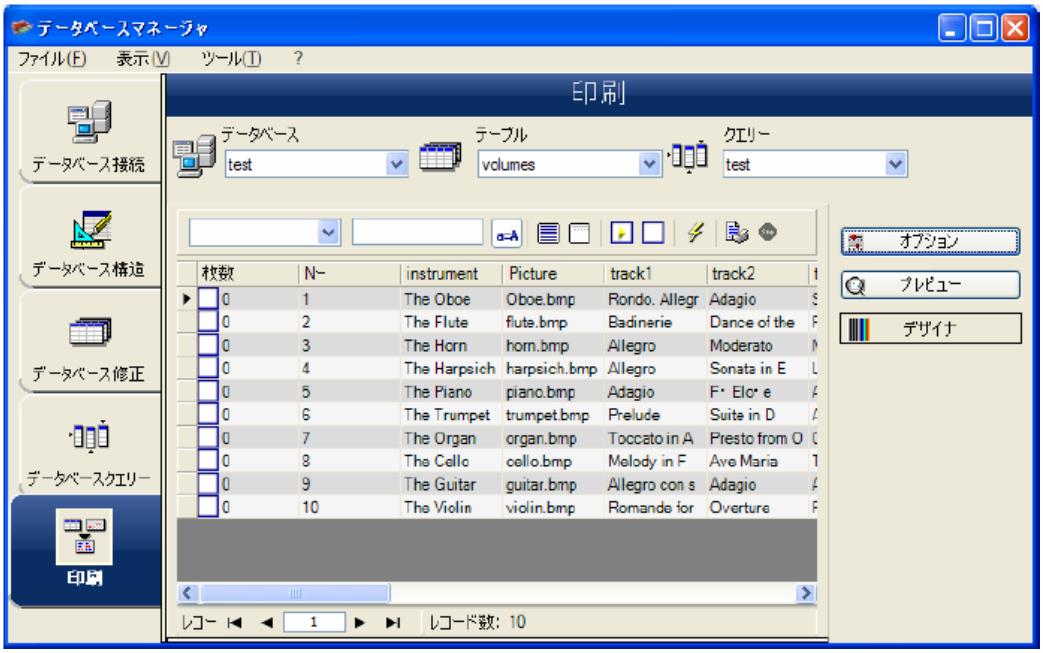
1. 目的のフィルターのデータベースのカーソルをクリックします。
2. レコードの削除ボタン()をクリックします。

SQL でフィルターを変更

注意：1 つ以上のフィルターが必要です。

1. SQL クエリータブを選択します。
2. 「SQL 文で編集」にチェックをつけ、SQL クエリーを有効にして、手動で変更を行います。
3. クエリーボタンをクリックし、結果を表示します。

印刷ウィンドウ



印刷ウィンドウを使用すると、印刷を開始する前に、印刷するファイルの選択、プリンターの指定、さまざまなパラメーターの定義を行うことができます。

印刷するドキュメントを選択

1. オプションボタンをクリックします。
2. ラベル名グループのファイルを選択します。
3. ファイルからドキュメントを選択します。

または

1. ラベル作成アシスタントボタン ( ) をクリックします。
2. ウィザードの説明にしたがって、進めます。

注意：データベースに関連するラベルを作成すると、データベースの各フィールドの位置づけに必要な要素を、正確に定義することができます。

既存のラベルを選択

1. 既存のラベルドキュメントを開くボタン ()をクリックします。
2. .lab ファイルを選択します。
3. 開くボタンをクリックします。

注意： オプションの「ラベル名」や「プリンター名」グループにある「フィールド」ラジオボタンを使用すると、 現在のデータベースのフィールドのひとつにそれらの文字が定義されている場合に、必要なラベル、プリンターが選択されます。

フィールドからドキュメントを選択

データベースにおいて、フィールドのひとつに印刷ジョブに使用されるラベル名が含まれる場合、このフィールドを、 データベースマネージャーが.lab ファイルを取得する場所として定義することができます。

Ref	Designation	Qt	Code	Labname
6574	Ref1	1	9876546321	Label1

6354	Ref2	2	1236478855	Label2
6987	Ref3	3	6987456321	Label1
3684	Ref4	4	3698745632	Label3

この例では、フィールド「Labname」は、 Labname フィールドとして使用できます。

1. ラベル名グループのフィールドを選択します。
2. 必要なフィールドを選択します。

プリンターの選択

プリンターの追加と削除ボタンをクリックします。

数式

数式データソース

コマンド： データソース > 数式 > 追加

数式データソースには、作成したデータソースのリストが含まれます。

これらのデータソースには、演算子、定数、データソース、予約変数、式および関数の組み合わせが存在しています。データは数字または英数字となります。

ドキュメント内で演算を行う場合、まず、数式データソースを作成しなければなりません。

このデータソースには、特定の式に関して必要な関数を定義できるダイアログボックスがあります。

関数について

関数とは、事前に定義された数式のことです。引数と呼ばれる特定の値を、シンタックスと呼ばれる特定の順番で使用して、演算を実行します。

関数は演算またはオペレーションの結果として、数字、文字列または論理値を返します。

数式の定義では、以下の6 つの関数グループがあります。

型と暗黙の型変換

このソフトウェアの変数には、一般のプログラミング言語にあるような型 (文字型、数値型、およびポイ

ンター型など) の明確な区別はなく、任意の時点で任意の型を代入できます。ソフトウェアは、代入した値を可能な限り適切な型に変換して処理を実行します。

命名規則

変数名は、英字で始まる半角英字と数字だけで構成される必要があり、少なくとも先頭から20文字または内部メモリーの許す範囲で識別されます。マルチバイト文字列、空白、記号文字などを使うことはできません。命名規約から逸脱した場合の動作は未定義です。

大文字と小文字は区別しません。しかし、プログラミングの定石として混在することは当然避けるべきです。

英文のドキュメントでは、より緩やかな命名規約が書かれていますが、このドキュメントの規約が優先します。ミドルウエア製品など他のアプリケーションとの連携、日本語を含めたマルチバイト環境で正しく動作させるには、このドキュメントの規約に従わなければなりません。

式の評価順序と副作用完了点

もう一つの重要な注意点は、式と変数の評価順序が必ずしも表記した通りとは限らないという点です。一般のプログラミング言語では、文は先頭から順次実行されますが、このソフトウェアでは式の実行順序を決定するコードは存在しません。従って、実行順序を仮定することはできません。

先ず全ての式評価に先立って、外部インターフェイスとなる変数 (フォーム、およびデータベース変数) を同順位で評価します。次に自由変数を評価します。従って、式の評価結果をこれらの変数へ返すことはできません。そうしないと、式が不定な値を評価する可能性があるからです。

最後に各式を評価します。各式の副作用完了点 (全ての式評価が完了し、値が確定する点) は、式末にあります。式中では、他の式の値、変数値、および関数の戻り値を仮定することはできません。

以上の点を要約すると、関数オブジェクトには以下の注意点があります：

- 関数中の式は常に左端から解析されますが、個々の関数オブジェクトがどのような順序で呼ばれるかは不定です。 個々の演算子の優先順位は、演算子の項を参照してください。
- 各式の副作用完了点は、常に式末にあります。
- 関数は再帰呼び出しをおこなうことができません。試みた場合の結果は未定義です。
- 自由変数を除く、変数間の相互参照結果は不定です。変数間で値を参照する時は、必ず自由変数を経由しなければなりません。

このプログラムには、6 種類の関数グループがあります:

- チェックデジッド関数
- 変換関数
    - ATA 2000変換機能
- 日時関数
- 倫理関数
- 数学関数
- 文字列関数

演算子

このプログラムでは、算術、比較、連結、及び論理演算子を使用できます。

演算子は、実行するオペレーションを示す数学記号です。算術、比較、連結、論理など、さまざまなタイプの演算子があります。

算術演算子

演算子	用途
*	乗算：2 つの数を掛け合わせます。
+	加算：2 つの数を加算します。
-	減算：ある数から別の数を引くか、あるいはオペランドにマイナス値を指定します。
/	除算：ある数を別の数で割ります。
^	べき乗：ある数を指数で累乗します。
%	剰余：ある数を別の数で割った余りです。

比較演算子

演算子	意味
<	~より小さい
<=	~より小さいまたは等しい
>	~より大きい
>=	~より大きいまたは等しい
=	~と等しい
<>	~とは異なる

連結演算子

2 つの文字列を組み合わせる場合に使用します。

演算子	意味
&	2 つの文字列の連結

論理演算子

（論理関数も参照してください）

演算子	意味
!	論理否定

チェックデジッド関数

addmodulo10 **(string)**: string に modulo 10 形式のチェックデジッドを付けます。

addmodulo10_212 **(string)**: string に modulo 10_212 チェックデジッドを付けます。

addmodulo43 **(string)**: string に modulo 43 形式のチェックデジッドにチェックサムを付けます。

bimodulo 11 **(string)**: 2 文字の modulo 11 形式チェックデジッドを返します。Code 11 のチェックデジッドです。

canadacustomscd **(string)**: Canada Customs Standard 形式のチェックデジッドです。

Check103 **(string)**: この関数は Mod103 を計算します。

check 128 **(string)**: Code 128、EAN-128、または JAN-128 のチェックデジッドを返します。

checkPZN **(string)**: PZN バーコードのチェックデジットを計算します。PZN バーコードは、ドイツの医薬製品で使用される Code 39 に基づきます。

CheckSumMod10 **(string)**: この関数は Mod10 を計算します。

CheckSumMod10_Codabar **(string)**: この関数は Codabar の特定の Mod10 チェックサムを計算します。

CheckSumMod10_MSI **(string)**: この関数は MSI バーコードの Mod10 チェックサムを計算します。

CheckSumMod11_3Suisse **(string)**: この関数は 3Suisses の Mod11 チェックサムを計算します。

CheckSumMod34 **(string)**: この関数は Mod34 を計算します。

CheckSum2Mod47 **(string)**: この関数は Mod47 を計算します。

checkupce **(string)**: UPC-E 形式のチェックデジッドです。6 桁の文字を与えなければいけません。

modulo 10 **(value)**: modulo 10 形式チェックデジッドを返します。Code 2 of 5 interlaced、EAN-13、EAN-8、JAN-13、JAN-8、EAN-128 K-Mart、または VPCA のチェックデジッドです。

modulo10_212 **(string)**: modulo 10 チェックデジッドの別形式です。

modulo10IBM **(string)**: IBM 形式の modulo 10 チェックデジッドです。

modulo10UPS **(string)**: UPS 形式の modulo 10 チェックデジッドです。

modulo 11 **(string)**: 1 文字の modulo 11 形式チェックデジッドを返します。Code 11 のチェックデジッドです。

modulo11IBM **(string)**: IBM 形式の modulo 11 チェックデジッドです。

modulo16 **(string)**: 1 文字の modulo 16 形式チェックデジッドを返します。

modulo 24 **(string)**: modulo 24 形式のチェックデジッドです。Code PSA のチェックデジッドです。

modulo 32 (value): modulo 32 形式のチェックデジッドです。主にイタリアの製薬業界で使用されています。

modulo 43 (string): modulo 43 形式のチェックデジッドです。code 39 のチェックデジッドです。

modulo 47 (string): 2 文字の modulo 47 形式チェックデジッドです。Code 93 のチェックデジッドです。

Plessey (string): 2 文字の Plessey 形式チェックデジッドです。

pricecd (string): 4 種類の UPC Random Price チェックデジッドを返します。

pricecd5 (string): 5 種類の UPC Random Price チェックデジッドを返します。

StringToExt39 (string): この関数は、Extended 39 バーコード で使用される文字列を準備します。

UPSCheckDigit (string): UPSチェック文字に戻ります。このメソッドの入力は文字列です。このメソッドは、トラッキング番号の残りをユーザーに任せます。

これは15文字シーケンスを取り、このシーケンスを使用してチェックデジットを計算します。

1 - 最初の2文字は"1Z"でなければなりません。

2 - 次の6文字は弊社のUPSアカウント番号"XXXXXX"を入力します。

3- 次の2文字はサービスタイプを意味します:

- "01"は翌日配達の航空便。

- "02"は翌々日配達の航空便。

- "03"は陸送便。

4- 次の5文字は弊社の請求書番号です（弊社の請求書は6桁で、最初の桁を省略する場合があります。 例えば、123456の請求書は23456の文字になります）。

5- 次の2桁は、パッケージ番号で、ゼロで埋められます。例えば、パッケージ1は"01"、2は"02"となります。

6- 最後の文字はチェックデジットです。

注:上記のシーケンスは17文字で、15文字だけがチェックデジットを計算するために必要です。これを行うには、"1Z"の部分を省略し、メソッドで最後の15文字だけを使用します。

変換関数

AI253 **(string):** アプリケーション識別子 253 の文字列を準備するための固有の関数。

AI8003 **(string)**: アプリケーション識別子 8003 の文字列を準備するための固有の関数。

ASCII **(string)**: ASCII コードを返します。文字列が渡されても、先頭の 1 バイトだけを評価します。

例:

ascii("A") = 65

char **(integer)**:

- 128 ~ 255 の値について:ASCII テーブルの整数の引数に対応する文字を指定します。
- 0 ~ 127 および 255 を超えるすべての負の数について:整数の引数に対応する Unicode 文字を返します。

例:

char (65) = "A"

CodabarData **(data, start, stop)**: CodabarData のデータを調整します。

Code128CData **(string)**: コード 128 C のデータを調整します。

Currencytoeuro **(value)**: 変換レートで除算して、ユーロに変換します。

DatabarData **(data, min, max, hasComposite)**: DatabarData のデータを調整します。

DBCSToUnicode **(string, codepage)**: 数式では変換されるデータ «string» と使用される Codepage «codepage» の 2 つのパラメータが使用されます。Codepage パラメータは、言語名 (Thai、Japanese、ChineseGBK、Korean、ChineseBig5、European、Eastern、Cyrillic、Greek、Turkish、Hebrew、Arabic、Baltic、Vietnamese、UTF-8、ACP) または数値 (インターネットの検索コードページ識別文書) で表されます。

例:

DBCSToUnicode(unicodetoDBCS("傍傍傍傍", "UTF-8"), "UTF-8") =傍傍傍傍

この関数は、Unicode 以外のファイルまたはデータソース (例、データベース) からのデータで必要です。

ート: この関数は、UnicodeToDBCS メソッドに対する予約されたアクションを持ちます。

dollar **()**: ドル通貨形式で返します。

例:

変数 PRICE が 199 であるとき:

dollar(PRICE) は $199.00 を返します。

Ean128Data **(string_1, string_2)**: Adjust "string_1" data for EAN 128 barcode according to template "string_2"

Eurotocurrency **(value)**: 変換レートで乗算し、ユーロから変換します。

ート: 変換定数は、オプション ダイアログボックスの その他 タブで設定します。

FileToData **(fileName, errorData, maxSize)**: ファイルから読み取られたデータを返します。

fixed **(value, num_decimals, non_sep)**: 数値文字列をフォーマットし、四捨五入して返します。 num_decimals は小数点以下の桁数、non_sep は 1000 単位の区切り文字の付加を指定するブール値です。

例:

fixed (1234.5678, 3, TRUE) = "1234.568"

fixed (1234.5678, 3, FALSE) = "1,234.568"

ノート: 数式 変数の 出力 タブ、10 進数表示 で、桁数や区切り文字を適切に設定しておかなければいけません。

FormatDate **(date, dateFormat, localeID)**: «dateFormat» に従ってフォーマットされた文字列に «date» を変換します。

localeID パラメータは、データ値を表示するために使用されるローケルを指定します。localeID はオプシ

ョンです (デフォルト: 0、システム ローケルが使用されます)。可能な値 (数) については

http://msdn.microsoft.com/en-us/goglobal/bb964664.aspx を参照してください。

FormatMoney (amount, use separation characters(T/F), price characters, force leading zaro (T/F), location for price character, # of decimal places)

この関数によって通貨記号、ドットの追加、先行ゼロの強制、区切り文字の使用ができます。

例:

FormatMoney("123456.234", "T","$","F",0,2) = $123,456.23

FormatMoney("1234","F","$","T",8,2) = $1234.00

FormatMoney("0.234","F","$","F",0,2) = $.23

FormatMoney("0.234","F","$","T",0,2) = $0.23

注：価格文字の位置は価格文字位置に関係し、価格文字と量の間にスペースが加われます。その数値が結果文字列の長さを下回る場合、無視されます。

FormatNumber (number): この関数は数値フィールドを書式化できます。ポンド記号 (#) は値が存在する場合だけ表示されることを意味し、ゼロ (0) は常に表示されることを意味します。

例：

FormatNumber(123.45, "US$ #,###,###.00") = US$ 123.45
FormatNumber(123.45, "US$ 0,000,000.00") = US$ 0,000,123.45
FormatNumber(.45, "#,##0.00") = 0.45
FormatNumber(.45, "#,###.00") = 45
FormatNumber(7188302335, "(###) ###-####") = (718) 830-2335
FormatNumber(123.45, "00.00") = 23.45
FormatNumber(123.567, "###,##0.00") = 123.57

GetEnv (名前): 環境変数の値 «名前» (文字列) を返します。

例:

GetEnv("OS") = " Windows_NT"

ート:この関数には、ユーザーが定義した変数を使用します。

GS1AIData **(AIName, AIData, isLastAI)**: GS1 AI のデータを調整します。

GS1HRData **(AIName», AIData, calcCheckDigit): Adjusts data for GS1 Human readable.**

GS1Norm **(string):** Adjusts data for GS1 Norm.

int **(value)**: 引数と等しいか、それ以下の最も大きな整数値を返します。

例:

int (-5.863) = -6 (-5 > -5.863 > -6 なので、-5 ではありません)

int (5.863) = 5

LmChar **(整数)** : ANSI表の整数因数に対応する文字に戻る。本表はローケルシステムに依存しません。その数値は 0 と255の間です。

MaxiCodeData **(Mode, PostalCode, CountryCode, ClassOfService, TrackingNumber, UpsShipperNumber, JulianDate, ShipmentID, PackageNumber, TotalPackages, PackageWeight, AdressValidation, ShipToAdress ,ShipToCity ,ShipToState)**: 入力 «string» から maxicode データを作成します。

text **(value, format)**: value を format フォーマット指定子でフォーマットして返します。

例:

text (012345678,"###-##-####") = 012-24-5678

text (2125551212,"(###)###-####") = (212)555-1212

trunc **(value)**: value を小数点で切り捨て、正数部分を返します。int との違いに注意してください。

例:

trunc (-5.863) = -5

trunc (5.863) = 5

UnicodetoDBCS **(Data, Codepage)**: (データ、コードページ):方式には2つのパラメータが入り、データは変換され、コードペ - ジが使用されます。 コードページパラメータは値として以下のもののどれでも使用できます。

Thai

Japanese

Chinese GBK

Korean

Chinese Big5

European eastern

Greek

Turkish

Hebrew

Arabic

Baltic

Vietnamese

value **(string)**: value を数値化します。

例:

value("123") = 123

ノート: このソフトウエアでは暗黙の型変換が強力なので、通常 value を使う必要はありません。 しかし、暗黙の型変換で予期しない型に変換されるときに有効です。

ValueEx **(string)**: «string» を数値に変換します。

VoiceCode **(string, string, string)**: VoiceCodeは4桁の数値で、GTIN、Lot、およびPTIからのオプションのDateを使用して計算されます。

Return value	string	VoiceCode値を計算します。
GTIN	string	14桁のGTIN数値。数値だけを使用できます。数値でない場合はエラーが返されます。 左側の14桁は、データ長が14以上の場合に受け付けられます。 データ長が14桁未満の場合、'0'が左側に追加されます。
LotNumber	string	最大20の英数字シンボルデータ。左側の20のシンボルは、データ長が20桁以上の場合に受け付けられます。
Date	string	これはオプションのパラメータです。YYMMDDフォーマットで日付を表す6桁のデータ。 不正な長さ、非数値、または不正な日付の場合、数式はエラーを返します。

この計算は以下のように実行されます:

1. PlainTextを計算。

    a. PlainTextは14桁のGTINで、Lot CodeおよびDate (存在する場合) によって付加されます。

    b. アプリケーションを識別する接頭文字または括弧を含めないでください。

    c. GTIN、LotおよびDateフィールドの間にはスペースはありません。

d. ゼロパックで'/'文字がないYYMMDDとして表される日付。

2. 標準のANSI CRC-16を使用してPlainText ASCIIバイトのANSI CRC-16ハッシュを計算します。

   X16 + X15 + X2 + 1の多項式のハッシュ

   CRC16 関数を参照してください。

3. 少なくとも4桁の有意な数値を10進数（ハッシュモード10000）で、ハッシュからVoiceCodeを計算します。

例:

GTIN = (01) 10850510002011

Lot = (10) 46587443HG234

PlainText = 10850510002011146587443HG234

CRC-16 Hash = 26359

VoiceCode = 6359

Large Digits = 59

Small Digits = 63

例:

VoiceCode("10850510002011", "46587443HG234") = 6359

VoiceCode("65457886676767", "2", "100126") = 5836

ATA 2000変換機能

Bin2Hex(«値», «パッド»)

二進法値を十六進法出力値に変換します。'パッド'パラメーターは、結果の左に文字を追加せずに出力全体の長さを定義するために使用されます。

例:

Bin2Hex("0010100111101001101",0)　は、14F4Dを返す

Bin2Hex("0010100111101001101",0)　は、00014F4Dを返す

CRC16_CCITT(«文字列»)

CRC16 CCITTチェックサムを計算します。

これはCRC計算のTOC向けATA2000規定に使用されます。

例:

CRC16_CCITT ("A")は、B915を返す

CRC16_CCITT ("AB")は、4B74を返す

CRC16_CCITT ("ABC")は、F508を返す

CRC32_CCITT(«文字列»)

CRC32 CCITTチェックサムを計算します。

これはCRC計算のTOC向けATA2000規定に使用されます。

例:

CRC32_CCITT ("A")は、D3D99E8Bを返す

CRC32_CCITT ("AB")は、30694C07を返す

CRC32_CCITT ("ABC")は、CBF43926を返す

Dec2Bin(«値», «パッド»)

十進法値を二進法出力値へ変換します。'パッド'パラメーターは、結果の左に文字を追加せずに出力全体の長さを定義するために使用されます。

例:

Dec2Bin("5", 0)は、101を返す

Dec2Bin("5", 4)は、0101を返す

Dec2Hex(«値», «パッド»)

十進法値を十六進法出力値に変換します。'パッド'パラメーターは、結果の左に文字を追加せずに出力全体の長さを定義するために使用されます。

例:

Dec2Hex(510, 0)は、1FEを返す

Dec2Hex(510, 4)は、01FEを返す

String2Hex(«値», «パッド»)

文字列値を十六進法に変換します。'パッド'パラメーターは、結果の右に文字を追加せずに出力のためのバイトでの配置を定義するために使用されます。

RFTAGでは、データは単語配置（2バイト）で十六進法によくエンコード化します。

例:

String2Hex("A", 0)は、41を返す（**A** は、十六進法で**41**である十進法のAINSIコード**65**)

String2Hex("ABC", 2)は、414243を返す

String2Hex("ABC", 4)は、41424300を返す

こちらは、RFID ATA2000出生記録のサンプルです:

String2Hex("MFR S0671*SEQ M37GXB92*PNO PQ7VZ4*PDT CONTROLLER*ICC 456789*UIC 2*DMF 20160701*UNT KG*HAZ UN0003*ESD 1*LOT 123456*CNT FR*PMLPML123456789", 2) が返すのは

4D46522053303637312A534551204D333747584239322A504E4F20505137565A342A50445420434F4E54524F4C4C45522A49

String6BitsEncoding(«値», «パッド»)

文字列値を 6-ビット ASCIIエンコーディング 出力値に変更します。

'パッド'パラメーターは、結果の右に文字を追加せずに出力のためのバイトでの配置を定義するために使用されます。

ATA2000データは、圧縮目的で6ビット（8ビットの代わりに）にエンコードされることができます。

例:

String6BitsEncoding("11", 0) は、C710 を返します。

VDAString6BitsEncoding(«string»)

ドイツ自動車工業会（VDA）要件を使用して文字列値を6ビットASCII値に変換します。

ATA2000ではデータを6ビット（8ビットではなく）に圧縮してコード化することができます。

変換に使用したテーブルはここで利用可能です。より詳しい内容はVDA5500およびVDA4994の仕様をご参照ください。

例:

VDAString6BitsEncoding("11") は、C71860820820 を返します。

日時関数

日関数は、コンピュータのローカルタイムを取得して返します。引数に渡す日時の書式は、now、または today 関数が返す書式で記述します。

BestBefore (date , dateFormat , offset , offsetUnit , changeMonth , outputFormat, localeID)

キーボードから入力した値に基づいて、有効期限の日付を計算できます (現在の日付に基づいて計算されるのではありません)。

戻り値	文字列	基準日とオフセットによって計算された、文字列形式の日付値です。
date	文字列	文字列形式の基準日です。
dateFormat	文字列/数値	基準日の値の形式です。 - 事前定義されているエンコード形式の番号を指定します。(例: 1 は dd/mm、2 は dd mmmm のようになります) - 0 または負の値を指定すると、現在のシステムの日付形式が使用されます。 - 文字列で値を指定すると、その形式が直接エンコードされます。(例: "yyyy/mm/dd")
offset	数値	基準日に加算する日付の単位の数です。
offsetUnit	文字列/数値	offset パラメータの内容を指定します。 - 日の場合は 1、「d」、または「D」を指定します - 月の場合は 2、「m」、または「M」を指定します - 年の場合は 3、「y」、または「Y」を指定します
changeMonth	数値	オプション (デフォルト: True) - 0 は False を示します - 1 は True を示します

		算出された日付がその月に存在しない場合 (例: 2 月の 30 日)、設定に応じて以下のように処理します。 True に設定されている場合は翌月の最初の日を返します (例: 3 月 1 日)。 False に設定されている場合は現在の月の最終日を返します (例: 2 月 28 日)。
outputFormat	文字列/数値	オプション (デフォルト: dateFormat と同じ) このパラメータは、算出された日の出力形式を指定します。指定可能な値は dateFormat と同じです。
localeID	数値	オプション (デフォルト: 0、システム ロケールが使用されます) このパラメータは、データ値を表示するために使用されるロケールを指定します。

例:

BestBefore("14/06/2012" ,"dd/mm/yyyy","18","m", 1, "mmmm dd, yyyy", 1033) will return December 14, 2013.

BestBefore("14/06/2012" ,"dd/mm/yyyy","18","m", 1, "mmmm dd, yyyy", 1036) will return Décembre 14, 2013.

例: Formula with a date data source.

Create a date variable name date0 with the date of the day, for example 26/06/2012 (dd/mm/yy format).

BestBefore(date0 ,"dd/mm/yy","18","m", 1, "dd/mm/yyyy") will return 14/12/2013.

CFIA_Month (date):

Returns month name used by Canadian Food Inspection Agency (CFIA): JA, FE, MR, AL, MA, JN, JL, AU, SE, OC, NO, DE.

Example:

CFIA_Month ("03/01/2021") returns MR

CFIA_Month (today()) returns month name based on today's date, if today is 01/01/2021 - the return value is JA

CFIA_Month ("06/01") returns JN

Note: «date» argument format depends on the environment settings applied for the system.

DateOffset **(date , offset, offsetUnit , changeMonth)**

戻り値	日付	指定された基準日に、日付の間隔を加算します。
date	日付	基準日の値です。
offset	数値	基準日に加算する日付の単位の数です。
offsetUnit	文字列/数値	offset パラメータの内容を指定します。 - 日の場合は 1、「d」、または「D」を指定します - 月の場合は 2、「m」、または「M」を指定します - 年の場合は 3、「y」、または「Y」を指定します
changeMonth	数値	オプション (デフォルト: True) - 0 は False を示します - 1 は True を示します 算出された日付がその月に存在しない場合 (例: 2 月の 30 日)、設定に応じて以下のように処理します。 True に設定されている場合は翌月の最初の日を返します (例: 3 月 1 日)。 False に設定されている場合は現在の月の最終日を返します (例: 2 月 28 日)。

例:

DateOffset("26/06/2012",1,"d",1) will return 27/06/2012

-OR-

DateOffset(today(),1,"D", 0)

例: Formula with a date data source.

Create a date variable name Date0 with the date of the day, for example 26/06/2012. You should ensure that date variable is in default system format (otherwise you can use DateValue() function to get the date value from the date variable).

DateOffset(Date0,1,"D", 0) will return 27/06/2012

DateValue **(formattedDate , format)**

戻り値	日付	formattedDate パラメータの日付値を戻します。
formattedDate	文字列	日付値に変換される、テキスト形式の日付です。
format	文字列	オプション (デフォルト: システムの日付形式) 定した場合、そのままの形式の文字列が表示されます (例: "dd/mm/yy")。

例:

DateValue(22062012,"ddmmyyyy") will return 22/06/2012.

day **(date)**: date 引数から、日を取り出して返します。

FiscalDate**(fiscalStartDate, outputFormat)**

Allows you to calculate the date for the start of your fiscal year (yyyy.mm.dd).

Examples (if current date is 2009.11.24):

FiscalDate("2009.01.01", 1) = 09

FiscalDate("2009.01.01", "yyyy") = 2009

FiscalDate("2009.01.01", 3) = 47

FiscalDate("2009.01.01", "day") = 328

hour **(date)**: date 引数から、時を取り出して返します。

minute **(date)**: date 引数から、分を取り出して返します。

month **(date)**: date 引数から、月を取り出して返します。

now **()**: 現在のシステム日時を返します。

second **()**: date 引数から、秒を取り出して返します。

shiftcode **(items)**

Shift codeは、現在の時刻に応じて変更が必要となるラベル上のフィールドのデータ・ソースとして使用されます。シフトは、定義され名前が付けられる時間帯を示します。

値を戻します	文字列	現在の shift code 値を戻します。
アイテム	文字列	次の形式の Shift code アイテム:開始時間:開始分-停止時間:停止分-値|… |開始時間:開始分-停止時間:停止分-値

たとえば、シフトは次のように定義される場合があります。

- 7:00 から 15:00 = 日中
- 15:00 から 23:00 = 晩

- 23:00 から 7:00 = 夜間

注:

1. 時間の値の定義では、24 時間制が使用されます。0:00 が真夜中で、12:00 は正午になります。

2. 入力に重複する時間がある場合には、最初に受け入れられる値が戻されます。

3. 現在の時間のシフトがない場合には、空の文字列が戻されます。

例 (現在の時間が 16:00 の場合):

shiftcode("7:00-15:00-日中|15:00-23:00-晩|23:00-7:00-夜間") ="晩"

SpecificDateFormat(date format, +/[offset data][date interval])

SpecificDateFormat関数によって、一定の間隔で日付をオフセットして日付スタンプをカスタマイズし、このカスタマイズされた日付を数式で使用できます。SpecificDateFormat 関数は、デフォルトの日付フォーマットまたはユーザーが選択している別のフォーマットを使用して、システム時計に基づいて現在の日付をオフセットします。

式で使用する場合、SpecificDateFormat 関数は、以下のパラメータを使用して記述する必要があります:

SpecificDateFormat ("[date format]", "+/ [offset data][date interval]")

例：SpecificDateFormat("mmmm","+2M")

- [date format] 使用する日付フォーマットを指定します。このパラメータは括弧で囲む必要があります。有効な日付フォーマットの表示テーブル:

日付設定	出力	説明
D と d	1	ヵ月（先行ゼロなし）
DD	01	2 文字のヵ月（先行ゼロ）
dd	1	2 文字のヵ月（先行スペース）

DDD	224	年の開始からの日数（先行ゼロ）
ddd	224	年の開始からの日数（先行スペース）
DDDDD と ddddd	33482	1/1/1900 からの日数 （5～9 文字を使用可能）
W と w	1	曜日 (日曜=1、土曜=7)
WW	34	年の開始からの週 （先行ゼロ）
ww	34	年の開始からの週 （先行スペース）
WWW と www	783	1/1/1900 からの週 （最後の 3 桁）
WWWW と wwww	4783	1/1/1900 からの週 （4～9 文字を使用可能）
WWWWWWWWW	000004783	1/1/1900 からの週（先行ゼロ）
wwwwwwwww	4783	1/1/1900 からの週（先行スペース）
M と m	9	月数（先行ゼロなし）
MM	09	2 桁の月数（先行ゼロ）
mm	9	2 桁の月数（先行スペース）
MMM	009	3 桁の月数（先行ゼロ）
mmm	9	3 桁の月数（先行スペース）
MMMM と mmmm	1101	1/1/1900 からの月
MMMMM	01101	1900 年からの 5 桁の月数（先行ゼロ）
mmmmm	1101	1900 年からの 5 桁の月数 (先行スペース)
MMMMMMMMM	000001101	1900 年からの 9 桁の月数（先行ゼロ）
mmmmmmmmm	1101	1900 年からの 9 桁の月数（先行スペース）
YY と yy	91	2 桁の年
Y と y	1991	通年
YYY と yyy	991	年の最後の 3 桁 （3～9 文字を使用可能）
YYYYYYYYY	000001991	通年（先行ゼロ）
yyyyyyyyy	1991	通年（先行スペース）

- +/[offset data] この値は、日付をオフセットするための量を指定します。現在の日付から加算または減算するかどうかによって、プラス（+）またはマイナス（-）記号を間隔の前に付ける必要があります。
- [date interval] 日付間隔には、日では d、週では w、月では m、または年では y のいずれかが必要です。

以下の表は、異なるフォーマットの式で使用される SpecificDateFormat 関数の例を示しています (

注: これらの例では、現在の日付は 2013 年 7 月 29 日と想定します) 。

SpecificDateFormat("mm dd yy" , "+1y") = 7 29 14 （ 現在の日付から 1 年後 ）

SpecificDateFormat("ww/m/yyyy" , "30w) = 1/12/2012 （ 現在の日付の 30 週前 ）

SpecificDateFormat("ddddd / wwww /mmmm" , "") = 41484/5928/1363 （ 1990 年からの日、週、および月 ）

TimeOffset (FormatDate(Now(), "mm/dd/yyyy hh:nn:ss"),"time interval +/offset time")

TimeOffset関数によって、一定の間隔で時間をオフセットして時間スタンプをカスタマイズし、このカスタマイズされた時間を数式で使用できます。 TimeOffset 関数は、デフォルトの時間フォーマット使用して、システム時計に基づいて現在の時間をオフセットします。

式で使用する場合、TimeOffset 関数は、以下のパラメータを使用して記述する必要があります:

TimeOffset(FormatDate(Now(), "mm/dd/yyyy hh:nn:ss"),"time interval +/ offset time")

例:TimeOffset(FormatDate(Now(), "mm/dd/yyyy hh:nn:ss"),"h+5")

- "mm/dd/yyyy hh:nn:ss"のデフォルト フォーマットだけを使用できます。DateValue 関数に TimeOffset を組み込む必要がある別のフォーマットを入力するには、別の時間フォーマットを指定できる FormatDate 関数に組み込んでください。
- "+/offset time" この値は、時間をオフセットするための量を指定します。現在の時間から加算または減算するかどうかによって、プラス (+) またはマイナス (-) 記号をオフセットの前に付ける必要があります。
- "time interval" 時間間隔は、秒では s、分では m、時間では h のいずれかである必要があります。

以下の表は、異なるフォーマットの式で使用される TimeOffset 関数の例を示しています (注: これらの例では、現在の時間は 2013 年 7 月 30 日 4:12:10 PM と想定します) 。

TimeOffset(FormatDate(Now(), "mm/dd/yyyy hh:nn:ss"),"h+5") = 07/30/2013 21:12:10 （ 現在の時間から 5 時間後 ）

FormatDate(DateValue(TimeOffset(FormatDate(Now(), "mm/dd/yyyy hh:nn:ss"), "h+10"), "mm/dd/yyyy hh:nn:ss"), "hhnn") = 16-22 （ 現在の時間から 10 分後 ）

today (): 現在のシステム日付を返します。

week (date): date 引数から、週番号を取り出して返します。その年の最初の木曜日を含む週が起算週で、1 から始まります (ISO 8601形式)。

weekday (date): date 引数から、日番号を取り出して返します。

ノート: 週の起算日は日曜日で、1 から始まります。

例:

今日が 2006/02/24 (金曜日) であるとき weekday(now()) = 6

WeekISO8601 (Date, DateFormat)

ISO8601 に対応する式を作成できます。ISO 8601 は、日付と時間に 関連するデータのやりとりを規定する国際標準です。

戻り値	日付	指定した日付の週を戻します
Date	文字列	テキスト形式の日付、日付値に変換されます。
DateFormat	文字列	基本の日付値の形式。 - 数値エンコードの定義済みの形式。 (例、1 dd/mm/yy、2 dd/mm/yyyy など - 0 または負の値は、現在のシステムの日付形式を使用することを

		意味します - 文字列値が形式を直接エンコードします。(例、"yyyy/mm/dd")

例:

WeekISO8601(" 03/01/2010 ",0) = 53

WeekISO8601(" 02/01/2011 "," dd/mm/yyyy ") = 52

WeekISO8601(" 01/01/2011 ", " dd/mm/yyyy ") = 52

year **(date)**: date 引数から、年を取り出して返します。

例:

minute (now()) 現在の分を返す。

year (today()) 今年の年を返す。

論理関数

論理関数は、値をテストして真、または偽を返すか、処理を実行する関数です。

ノート: このソフトウエアでは、TRUE (真) は 1、FALSE (偽) は 0 と定義します。真偽値が要求されるところで 0 以外の数値が渡されると、TRUE として扱います。

and **(expr_1, expr_2)** ：両方の引数が正しい場合、TRUE を返します。少なくとも1 つが正しくない場合、FALSE となります。引数は、論理値から計算しなければなりません。

exact **(string_1, string_2)** ：2 つの文字列が同じ場合TRUE、異なる場合はFALSE を返します。この関数は大文字・小文字を区別します。

例：

exact("String","String") = 1

exact("String","string") = 0

if (expr, Val_if_true, Val_if_false): expr が正しい場合はVal_if_true 値を返し、expr が正しくない場合は Val_if_false 値を返します。

例：

if(exact("String", "string"), "true", "false") = false

if(exact("String", "String"), "true", "false") = true

not (logical): 論理式logical の否定を返します。

例:

not(exact("String", "string")) = 1

not(exact("String", "String")) = 0

not(False) = 1 または not(0) = 1

not(True) = 0 または not(1) = 0

not(1+1=2) = 0

or (expr_1, expr_2): 2 つの引数の内1 つが正しい場合TRUE を返し、引数が両方とも正しくない場合 FALSE を返します。この引数は論理値から計算しなければなりません。

例:

or(exact("String", "string"),exact("string", "String")) = 0

or(exact("String", "String"),exact("string", "String")) = 1

or(true,true) = 1 または or(1,1) = 1

or(true,false) = 1 または or(1,0) = 1

or(false,false)= 0 または or(0,0) = 0

数学関数

Abs **(data):** この関数はデータの絶対値（正の数）を表示します。 数字の後に文字を使用できます。

例

Abs(-5) = 5

Abs(5) = 5

base10tobaseX **(string_1,string_2)** ：string_2 を10 進数 から string_1進数へ変換します。

例

Base 16 という名前のフィールドに文字列「0123456789ABCDEF」が含まれる場合、

base10tobaseX(Base16, 12) はC を生成します。

base10tobaseX(Base16,10) は A を生成します。

base10tobaseX("012345","9")は13 を生成します。

注意：この式は、「string_2」引数に負の値をとることができません。

baseXtobase10 **(string_1, string_2)**: string_1 文字列で指定した基数の文字列 string_2 を 10 進数へ変換して返します。

例：

Base 16 という名前のフィールドに文字列「0123456789ABCDEF」が含まれる場合

baseXtobase10(Base16, "E") は14 を生成します。

baseXtobase10(Base16,10) は 16 を生成します。

baseXtobase10("012345",10)は6 を生成します。

Ceil **(data)**: この関数は、データを次の整数に丸めます。 数字の後に文字を使用できます。

例：

Ceil(3.234) = 4

Ceil(7.328) = 8

Decimals **(data1, data2):** この関数は data1 に data2 小数位を使用します。 数字の後に文字を使用できます。

例：

Decimals(4, 2) = 4.00

Decimals(3.524, 1) = 3.5

eval_add **(string, string)**: パラメーターの加算を返します。

例：

eval_add(5,5)=10

eval_div **(string, string)**:パラメーターの除算を返します。

例：

eval_div(20,2)=10

eval_mult **(string, string)**: パラメーターの乗算を返します。

例：

eval_mult(5,2)=10

eval_sub **(string,string)**:パラメーターの減算を返します。

例：

eval_sub(20,10)=10

Floor **(data)**: この関数は、データを次の整数に丸めます。 数字の後に文字を使用できます。

例：

Floor(3.234)= 3

Floor(7.328)= 7

hex **(val_1, val_2)**: 10 進数のval_1 をval_2 桁の16 進数に変換します。

例：

hex(2,8) = 00000002

int **（値）**：値 引数より小さいまたは等しい、最大整数を返します。

例:

int (-5.863) = -6

int (5.863) = 5

max **(data1, data2)**: この関数は最低値を表示します。 数字の後に文字を使用できます。

例:

Max(5, 12) = 12

min (data1, data2): この関数は最低値を表示します。 数字の後に文字を使用できます。

例:

Min(5, 12) = 5

mod (val_1, val_2) ： val_1 引数を val_2 引数で割った余りを返します。符号は被除数と同じになります。

例:

mod (7,2) = 1

mod (-7,2) = -1

mod (7,-2) = 1

mod (-7,-2) = -1

quotient (val_1, val_2) ：val_1 引数を val_2 引数で割った商を整数で返します。

例：

quotient(10,2) = 5

round (val_1, val_2) ：val_1 引数 を、val_2 の桁数で四捨五入した値を返します。

- val_2 が 0 より大きい場合、val_1 は、小数点以下の桁数で丸めます。
- val_2 が0 に等しい場合、 val_1 は最も近い整数で丸めます。
- val_2 が 0 より小さい場合、val_1 は、小数点の左側の桁数で丸めます。

例:

round (4.25,1) = 4.3

round (1.449, 1) = 1.4

round (42.6,-1) = 40

文字関数

各ボックスに文字が含まれている場合、文字列をテーブルに含めることができます。それは長さで定義します（スペースを含む、文字列の総文字数）。文字列の文字の位置は、テーブルの場所に対応します。例えば、1 字目は位置 1 になります。

例：位置 3 は文字列の3 番目の文字に対応します。

cyclebasex **(baseX_string, start_value, increment, copies)** ： あらゆるデータベース・カウントシステムにおいて、カウントを利用可能とします。ナンバリングシステムは、リンクされた式内で定義しなければいけません。開始値、それぞれの増分の値、コピー数もそれぞれの数について指定します。これらの値はすべて、ラベルの他のフィールドにリンクさせることができますが、フィールド名を引用符に含めてはいけません。

例：

Base 16 という名前のフィールドに文字列 0123456789ABCDEF が含まれる場合、以下のとおりです。

cyclebasex(base16, "8", 1 ,1) = 8,9,A,B,C…

cyclebasex(base16, "F", -1,1) = F,E,D,C,B,A 9,8,7…

cyclebasex(base16, "B0 ", 1,1) = B0, B1, B2…

cyclebasex("012345", "4",1,2) = 4,4,5,5,10,10,11,11…

cyclechar **(first_char, last_char, increment, copies)** ： 完全なサイクルについて、ユーザー定義の文字セットを作成します。

例：

cyclechar("A", "C") = A B C A B C A B C …

cyclechar("A", "C", 1,2) = A A B B C C A A B B …

cyclenumber **(first_char, last_char, increment, copies)** ： 通常の数や文字のシーケンスを使用するのではなく、独自の数字のシーケンスを設定することができます(0,1,2… r A,B,C…)。

例:

cyclenumber(1,3) は次のシーケンスのラベルを作成します： 1 2 3 1 2 31 2 3...

cyclenumber(1,3,1,2) は次のシーケンスのラベルを作成します： 1 1 2 2 3 3 1 1 2 2 3 3...

cyclestring **(string)** ： 増分フィールドとして、完全なサイクルを使用し、単語または文字のグループを作成することができます。この完全な文字列は引用符(" ") 含めなければならなく、各単語または文字グループはセミコロン(;)で区別しなければなりません。

例:

cyclestring("Mon ; Tue ; Wed ; Thu ; Fri ; Sat ; Sun") = Mon TueWed Thu Fri Sat Sun以下の例は、O とI 除く、すべてのアルファベット文字を使用するラベルを示します。

cyclestring("A;B;C;D;E;F;G;H;J;K;L;M;N;P;Q;R;ST;U;V;W;X;Y;Z")

exact **(string_1, string_2)** ： 2 つの文字列が同じ場合TRUE、異なる場合FALSE を返します。

例:

exact("software","software") = 1

exact("sftware","software") = 0

extract **(string, sep, pos)** ： 文字列sep で区別されるデータを含む文字列string から位置pos の文字列を返します。

例：

Extract("AB;CD;EFG;HIJ", ";", 3) = ”r;EFG”

find (string, key, start) ：string 引数で最初に現れるkey 引数の位置を返します。string 引数の検索は、start 引数 (start >= 1)によって返される位置から始まります。key 数が見つからない場合は、この関数はゼロを返します。この関数は、大文字・小文字を区別します。

例：

find("Peter McPeepert","P",1) = 1

find("Peter McPeepert","p",1) = 12

left (string, num_char) ： 文字列string の先頭からnum_char 引数で与えられた長さの文字列を返します。

例:

left("Peter McPeepert",1) = P

left("Peter McPeepert ",5) = Peter

len (string) : : string 引数の長さを返します。スペースも文字としてカウントします。

例:

len("Paris, New York") = 15

len("") = 0

len(" ") = 1

lower (string): 文字列string をすべて小文字に変換します。

例:

lower("Paris, New York") = paris, new york

LTrim **(string)**: この関数は自動的に左データの終了スペースまたは先頭スペースを切り取ります。

例:

LTrim(" No."): No

mid **(string, start, num_char)**: 文字列string の start 引数 (start >=1)で与えられた位置からnum_char 引数で与えられた長さの文字列を返します。。

例:

mid("Paris, New York",8,8) = New York

pad **(string, num_char, fill_char)**: 指定された長さになるように、フィールドの左側に指定した文字を追加します。すべての文字をパディング文字として選択できます。

例:

GREETING という名前のフィールドに HELLO の値を表示する場合、pad(GREETING,8,0) = 000HELLO

pad(5,3,0) = 005

pad("Nine",6,"a") = aaNine

replace **(string, start, num_char, new_string)**: 変換された string 引数を返します。start 引数で与えられた位置からnum_char 引数で与えられた文字を new_string 数で与えられた文字列に置換します。

例:

replace("Paris, New York", 8, 8, "Singapore") = Paris, Singapore

ReplaceString **(string, old_string, new_string)** 文字列 string 内のすべてのold_string をnew_string に置換します。

例:

ReplaceString("abc12def12", "12", "") = abcdef

rept (string, num_char): string 引数を num_char 回繰り返した文字列を返します

例:

rept("Ah Paris! ", 2) = Ah Paris! Ah Paris!

right (string, num_char): 文字列string の最後の文字からnum_char で与えられた長さの文字列を返します。

例:

right("Purchase order", 8) = order

RTrim (string): この関数は自動的に右データの終了スペースまたは先頭スペースを切り取ります。

例:

RTrim("Part ") :Part

search (string, key, start): 文字列string 内でstart (start >= 1)位置以降に現れるkey の位置を返します。 検索は、start 引数 (start >= 1)により定 義される位置から始まります。この関数は、key 引数が見つからない場合、ゼロを返します。

例:

search("Purchase order","order”,1) = 10

search("Purchase order","c",1) = 4

StrAfter (data, start after, length): この関数は long after を指定した start after 文字と全く同じ長さの文字列にします。

例:

StrAfter("1234-5678", '-', 3)= ダッシュの後に 3 文字をとります (567)

StrAfter("1234-5678", '-')= ダッシュの後にすべての文字をとります(5678)

StrBefore **(data,start before, length)**: この関数は long before を指定した start before 文字と全く同じ長さの文字列にします。

例:

StrBefore("1234-5678", '-', 2)= ダッシュの直前に 2 文字をとります (34)

StrBefore("1234-5678", '-')= ダッシュの前にすべての文字をとります (1234)

SuppressBlankRows **(«string»):** スキップした空行がある文字列を戻します。これでオブジェクトを作成し、フィールドを置き、空白のものを抑制します。

例:

SuppressBlankRows ({Var0} & char(10) & {Var1} & char(10) & {Var2}) （ Var0、Var1、Var2は変数。char(10)は"\n"シンボルで改行という意味です ）

Variables/Values	Formula	Result
Var0 = "Doris" Var1 = "Bull Run Ranch" Var2 = "Aurora"	*SuppressBlankRows({Var0} & char(10) & {Var1} & char(10) & {Var2})*	Doris Bull Run Ranch Aurora
Var0 = "Craig" Var1 = "" Var2 = "Chicago"	*SuppressBlankRows({Var0} & char(10) & {Var1} & char(10) & {Var2})*	Craig Chicago
Var0 = "Craig" Var1 = "" Var2 = "Chicago"	*{Var0} & char(10) & {Var1} & char(10) & {Var2}*	Craig Chicago

trim **(string)**: 文字列 string の始めと終わりのすべてのスペースを削除します。また単語間に複数のスペースが存在する場合は、1 つに削減します。

例:

trim(" Purchase order") = Purchase order

trimall **(string)**: 文字列 string からすべてのスペースを削除します。

例:

trimall("Paris / New York / Rome") = Paris/NewYork/Rome

upper **(string)**: 文字列string をすべて大文字に変換します。

例:

upper("Purchase order") = PURCHASE ORDER

ztrim **(value)**: 完全な数字フィールドのすべてのゼロを左から削除します。

例:

WEIGHT という名前のフィールドが000200 の値を表す場合

ztrim(weight) = 200

数式データソースのプロパティを定義

コマンド： データソース > 数式 > 追加

ドキュメントブラウザ の データソース タブで、変数のプロパティを設定できます。

1. 編集ボックスに直接、数式を入力します

   または

   必要な要素を選択し、挿入ボタンをクリックします。

2. テストボタンをクリックすると、シンタックスが正しいかどうかを確認できます。エラーが発生した場合、画面の指示に従って、必要な変更をすべて行ってください。
3. OK ボタンをクリックします。

ヒント：ダブルクリックにより要素を挿入することができます。

注意：数式に使用される変数に、以下の文字の 1 つを含む名前がある場合、それを括弧 {}に含めなければいけません。&+-*/<>=^%,!\"

注意：テストボタンをクリックすると、数式をチェックすることができます。メッセージに式の値が表示される場合、式が正しいことになります。値が正しくない場合は、画面の指示に従って必要な変更を行ってください。取得した値が切れている場合、出力タブに指定される最大長を変更してください。。

実践ワークショップ: 特別なチェックデジットの作成

この例では2/5 interleaved バーコード用のチェックデジットを計算します。

重みの計算

チェックデジットはデータの最初の文字に1 を掛け、2 番目の文字は 2、3 番目の文字は 1、4 番目の文字は 2...を掛けて算出します。

チェックデジット文字はCustomer_Code 変数の値から計算されます。予めCustomer_Code 変数を作成し、値として26053 を入力しておきます。

Formula_1_Weighted 式を作成し次の式を入力します。

Formula_1_Weighted :

mid(Customer_Code, 1, 1) * 1 &

mid(Customer_Code, 2, 1) * 2 &

mid(Customer_Code, 3, 1) * 1 &

mid(Customer_Code, 4, 1) * 2 &

mid(Customer_Code, 5, 1) * 1

Customer_Code の値が26053 であれば

2 * 1 = 2

6 * 2 = 12

0 * 1 = 0

5 * 2 = 10

3 * 1 = 3

結果を連結して2120103 が得られます。

ウエイトの計算結果を加算

次のステップは、前の式から導き出された数値を加算することです。この文字列の最大許容長さは 2 であることにご留意ください。式は以下のとおりです。

Formula_2_Sum：

mid(Formula_1_Weighted, 1, 1) +

mid(Formula_1_Weighted, 2, 1) +

mid(Formula_1_Weighted, 3, 1) +

mid(Formula_1_Weighted, 4, 1) +

mid(Formula_1_Weighted, 5, 1) +

mid(Formula_1_Weighted, 6, 1) +

mid(Formula_1_Weighted, 7, 1)

結果として

2 + 1 + 2 + 0 + 1 + 0 + 3 = 9

が得られます。

チェックデジットの計算

前の結果を用いて、チェック数字の値を計算します。3 番目の式を作成し、Formula_3_CheckDigit と名前を付けます。式は以下のとおりです。

Formula_3_CheckDigit：

if ((Formula_2_Sum % 10) > 0, 10 - Formula_2_Sum % 10, 0)

この計算は、先に計算した重み値加算値9 を10 で割り、その余りが1 以上なら10 から余りを引きます。この例では余りが9 なので、

10 - 9 = 1

この操作はチェックデジット文字が2 桁になった場合に、1 桁に戻すためのトリックです。

エンコードするデータの計算

バーコードを作成する場合、エンコードするデータを含まなければなりません。例えば、チェックデジットの値(Formula_3_CheckDigit)に連結されるCustomer_Code 数の値などです。

4 番目の式を作成し、 Formula_4_NewCustCode と名前を付けます。この式は、Customer_Code と Formula_3_CheckDigit を連結した結果となります。式は以下のとおりです。

Formula_4_NewCustCode：

Customer_Code & Formula_3_CheckDigit

結果として

26053 & 1 =「260531」

が得られます。

バーコードの作成

1. Formula_4_NewCustCode 式を選択し、Customer_Code バーコードのラベルにドラッグ & ドロップします。
2. バーコードのプロパティを定義します。

France
+33 (0) 562 601 080

Germany
+49 (0) 2103 2526 0

Singapore
+65 6908 0960

United States
+1 (414) 837 4800

www.teklynx.com

Microsoft Partner
Gold Independent Software Vendor (ISV)