Interstage Information Integrator
プラグイン用アプリケーション
～作成手引書～

2012 年 4 月
富士通ミドルウェア株式会社

1. はじめに

Information Integrator では様々なデータ変換の機能をサポートしています。しかし、それらの機能では各環境に適した変換を満たせない場合があります。このため、動作環境に適したデータ変換のアプリケーションを作成した後、そのアプリケーションをプラグイン機能から呼び出すことで利用者独自のデータ変換を可能にしています。

プラグイン機能から呼び出すためにはアプリケーションの作成方法(プログラミング、コンパイル、及びビルド方法)や定義方法について考慮すべき点があります。

本書では、それらの考慮すべき点を説明します。なお、本書を参照される場合にはお手元に Information Integrator の“システム設計ガイド”、“ソフトウェア説明書”、ならびに動作環境に合った OS の開発ツールのマニュアルをご準備の上で必要に応じて参照してください。

2. 項目編集プラグイン用アプリケーションの作成方法

ここでは、項目編集プラグイン（または変換プラグイン）で利用できるアプリケーションの作成方法について説明します。

項目編集プラグイン用アプリケーションのサンプルソースは、プログラム言語としてC言語を使用しており、OS種別によって以下のコンパイラを使用します。

OS種別	コンパイラ
Windows 32bit	Microsoft® Visual Studio 2005(注)
Windows 64bit	Microsoft® Visual Studio 2005(注) ＋[x64コンパイラおよびツール]
Linux (32/64bit)	GNU C Compiler
Solaris (32/64bit)	Sun WorkShop 4.0 以降

注）Microsoft Visual Studio 2008 や Microsoft Visual Studio 2010 でコンパイルした場合、該当するVisual Studio C++の再頒布可能パッケージを Information Integrator の動作環境にインストールする必要があります。

2.1 アプリケーションのプログラミング

サンプルソースを流用せず、独自に項目編集プラグイン用アプリケーションを作成する場合に、プログラミングで考慮すべき点について説明します。

項目編集プラグインはマルチスレッド構成としてアプリケーションを実行します。そのため、マルチスレッドで動作するようにスレッドセーフな関数を使用すると共に、適切なコンパイルオプションおよびリンカオプションを設定してライブラリを作成してください。

先ず、マニュアルの“システム設計ガイド”を参照してください。

【関数インターフェースについて】

関数インターフェースについて以下の点を決定する必要があります。

- 入力とする引数、出力とする引数の決定
  関数インターフェースとする引数のうち、入力とする引数、出力とする引数はどれか。
- 入力データとする引数の決定
  入力とする引数のうち、入力データ(処理プロセス実行時に Information Integrator から受け取るデータ)とするものはどれか。
- 結果データとする引数の決定
  出力とする引数のうち、処理結果となる結果データ(処理プロセスの実行時にアプリケーションから Information Integrator に対して受け渡すデータ)はどれか。
- アプリケーション内で扱うデータの決定
  マニュアルの“システム設計ガイド”を参照して、以下について決定してください。
  －どのようなデータを扱うのか
  　テキスト形式あるいはバイナリ形式のどちらを扱うのか
  　バイナリ形式であれば、文字列型、各数値型のようにどのようなデータ型を扱うのか
  －テキスト形式またはバイナリ形式データの文字列型の場合、扱う文字コードはどれか。
- 引数の型の決定
  アニュアルの“システム設計ガイド”を参照してください。

【作成したアプリケーションについて】

作成したアプリケーションについて以下の点に注意してください。

- 作成したアプリケーションから別ライブラリに存在する関数を呼び出すことは可能です。ただし、項目編集プラグイン機能ではアプリケーションから呼び出している別ライブラリについて判断する手段がありません。したがって、アプリケーションと同じディレクトリに配置するか、アプリケーションから動的リンクでアクセスして関数を実行するようにしてください。
  なお、別ライブラリへ動的リンクでアクセスして失敗した場合、復帰値やエラーメッセージでアクセスに失敗したことがわかるように作成することをお薦めします。

- アプリケーション内でローカル変数を宣言して使用することは可能です。ただし、スタック領域にメモリを確保していきますので、数十 MB 等の大きすぎる変数を使用すると不当な動作となる場合があります。動作環境に適したサイズの変数を使用するようにしてください。

2.1.1 Windows における注意点

項目編集プラグインでは定義されたアプリケーションのライブラリを動的にリンクして関数を実行しています。

そのため、Windows の場合はアプリケーションのライブラリに関数シンボルを明示する必要があります。なお、Unix (Linux/Solaris)では、このような対処は不要です。

関数シンボルを明示するには、実行対象となる関数のプロトタイプ宣言に対して"_declspec(dllexport) _cdecl"を付与します。

例えば、以下のアプリケーションを実行する場合に、プログラム上の関数宣言および関数実装部分に対して宣言を追加します。

復帰値　　:int 型
引数　　:char *in_date, char *kind, char *outdata
関数名　　:str_convert

関数プロトタイプの例

```
int _declspec(dllexport) _cdecl str_convert( char *in_date, char *kind, char *outdata );
```

2.2 アプリケーションのコンパイル

コンパイル方法について OS 毎に説明します。

2.2.1 Windows

Windows の場合、開発ツールには Microsoft Visual Studio 2005 を使用します。ただし、プロセッサ種によって仕様が異なります。Microsoft Visual Studio 2005 を使用したコンパイル方法について、以下に順を追って説明します。

なお、開発ツールに Microsoft Visual Studio 2008 や Microsoft Visual Studio 2010 を使用することも可能です。ただし、この場合は該当する Visual Studio C++の再頒布可能パッケージを Information Integrator の動作環境にインストールする必要があります。

◆32/64bit プロセッサ共通

【Visual Studio 2005 のバージョンについて】

Information Integrator の動作環境とアプリケーション開発環境の Visual Studio 2005 のバージョンを一致させる必要があります。Information Integrator の動作環境は、Visual Studio 2005 Professional Edition Service Pack 1 (KB926602)です。このバージョンの確認方法についての詳細は Information Integrator のソフトウェア説明書を参照してください。

【コンパイルオプションおよびリンカオプションについて】

Visual Studio 2005 のプロジェクトを使用する場合

プロジェクトのプロパティでコンパイルオプションとリンカオプションを指定する必要があります。

(1) [プロジェクト]－[xxxxxのプロパティ]をクリックします(xxxxx:任意のプロジェクト名)。

(2) [xxxxxプロパティページ]画面(xxxxx:任意のプロジェクト名)では、[構成プロパティ]－[全般]－[文字セット]にはデフォルトの[Unicode 文字セットを使用する]に設定されています。
必ず[設定なし]または[マルチバイト文字セットを使用する]を設定してください。

(3) 前述したようにアプリケーションのライブラリを動的にリンクして関数を実行します。よって、Windows で動作するためには Visual Studio 2005 のライブラリ(MSCRV80.dll)をリンクする必要があります。
[xxxxxプロパティページ]画面(xxxxx:任意のプロジェクト名)で、[構成プロパティ]－[C/C++]－[コード生成]－[ランタイムライブラリ]に[マルチスレッド DLL(/MD)]を必ず設定してください。
ただし、[マルチスレッド(/MT)]を設定した場合、Visual Studio 2005 のライブラリ(MSCRV80.dll)はリンクされないため、項目編集プラグインからは呼び出すことができません。

◆64bit プロセッサのみ

【64bit 版コンパイルツールのインストール】

64bit 版としてコンパイルするには Visual Studio 2005 のオプション機能である“x64 コンパイラおよびツール”を選択してインストールしてください。

インストール方法についての詳細は、開発環境にインストールされた Visual Studio 2005 のドキュメントを参照してください。

なお、デフォルトインストールではこのオプションはインストールされませんので、注意してください。

【64bit 版コンパイルツールでのモジュール作成】

Visual Studio 2005 のプロジェクトを使用する場合

ソリューションのプラットフォームに[x64]を追加した後、ビルドを実施する必要があります。以下の手順で追加してください。

(1) プロジェクトの[ビルド]－[構成マネージャ]をクリックします。
(2) 表示された[構成マネージャ]画面で、[アクティブソリューションプラットフォーム]をクリックします。
(3) [新規作成]をクリックします。
(4) [新しいソリューションプラットフォーム]画面の[新しいプラットフォームを入力または選択してください]フィールドで直接[x64]を入力します。
(5) [新しいプロジェクトプラットフォームを作成する]チェックボックスにチェックを入れ、[OK]ボタンをクリックします。
(6) [構成マネージャ]画面を閉じます。

なお、64bit 版モジュールの作成方法についての詳細は、開発環境にインストールされた Visual Studio 2005 のドキュメントを参照してください。

【作成した 64bit 版モジュールの確認方法】

モジュール作成先に配置される以下のマニフェストファイルを確認してください。

マニフェストファイル名：任意のモジュール名.xxx.manifest (※)

※xxx：Visual Studio の環境により異なります。

64bit プロセッサ向けには“processorArchitecture”の指定値が’amd64’になっていることを必ず確認してください。

プロセッサ種	processorArchitecture の値	記載例
32bit	″x86″	processorArchitecture=″x86″
64bit	’amd64’	processorArchitecture=’amd64’

2.2.2 UNIX(Linux/Solaris)共通

Unix の場合のライブラリの作成方法を以下に簡単に説明します。

(1) プログラムの作成
任意のテキストエディタを使用してプログラムを作成します。C 言語の基本的な作法(ANSI 準拠)を厳守してください。
(2) プログラムのコンパイルおよびリンクによるライブラリの作成
ライブラリ名は任意に作成できますが、UNIX 系の作法として**″libxxxxx.so(xxxxx：任意)″**としてください。

2.2.3 Linux

Linux の場合、開発ツールには GNU C Compiler を使用します。Linux OS の標準インストールではインストールされませんので、別途入手してください。

Information Integrator からはマルチスレッド構成として実行されるため、以下のオプションを指定する必要があります。
これらのオプションの詳細は各開発ツールのマニュアルを参照してください。

オプション種	オプション名	備考
コンパイルオプション	-pthread	POSIX スレッドを扱うためのオプション。
	-fPIC	共有ライブラリ(.so)を作成するためのオプション。
	-DLINUX	OS 固有。
	-D_LINUX	
	-DLinux	
リンカオプション	-pthread	POSIX スレッドを扱うためのオプション。
	-lpthread	
	-ldl	動的ライブラリをロードするためのライブラリをリンクするためのオプション。
	-lcrypt	-
	-shared	共有ライブラリ(.so) を生成するためのオプション。
	-Bdynamic	動的ライブラリをロードするためのライブラリをリンクするためのオプション。

なお、アプリケーションの作成環境と Information Integrator の動作環境のリリース種(RHEL 5/6)、ならびにプロセッサ種(32/64bit)は必ず一致させてください。一致していない場合、正しく動作しません。

2.2.4 Solaris

Solaris の場合、開発ツールには Sun WorkShop 4.0 以降を使用します。Solaris OS の標準インストールではインストールされませんので、別途 Sun WorkShop を入手してください。

Information Integrator からはマルチスレッド構成として実行されるため、以下のオプションを指定する必要があります。これらのオプションの詳細は各開発ツールのマニュアルを参照してください。

オプション種	オプション名	備考
コンパイルオプション	-mt	マルチスレッド形式には必須。 -D_REENTRANT をプリプロセッサに渡し、-lthread を付加するためのオプション。
	-KPIC	共有ライブラリ(.so)を作成するためのオプション。
	-DSolaris	OS 固有。
	-Xa	ANSI C に従って意味処理を変更するためのオプション。
リンカオプション	-mt	マルチスレッド形式には必須。 -D_REENTRANT をプリプロセッサに渡し、-lthread を付加するためのオプション。
	-G	共有ライブラリ(.so) を生成するためのオプション。
	-dy	

2.3 Information Integrator の定義作成

Information Integrator からアプリケーションを実行するために定義を作成します。定義を作成する際のポイントを次に説明します。

2.3.1 アプリケーションの動作確認

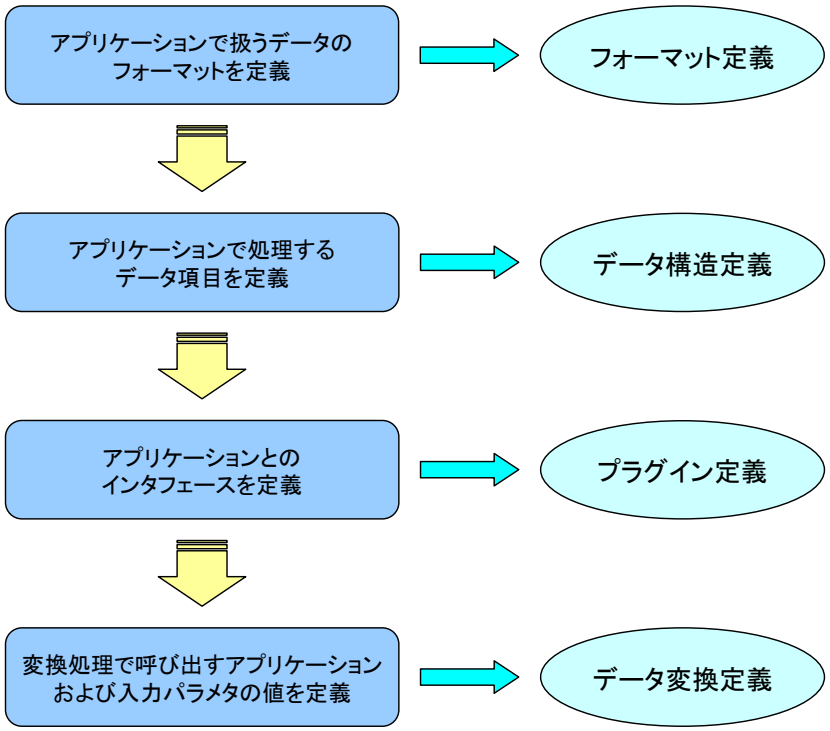
項目編集プラグイン機能と連携する前に、作成されたアプリケーションの動作確認を実施する必要があります。これにより、処理プロセスの実行時に異常が発生した場合においても、アプリケーション内での異常であるのか、または定義方法に誤りがあるのかの切り分けが可能となります。

アプリケーションのライブラリを動的にリンクする実行形式ファイルを作成して予め動作確認を実施してください。
なお、動作確認時には以下の点に注意してください。
(問題が無い場合は□にチェックをいれてください。)

- □ Information Integrator からの入力データとして想定している引数に正しくデータが渡っているか
- □ Information Integrator への出力データとして想定している引数に正しくデータが渡っているか
- □ Information Integrator への出力データのうち、結果データとして想定している引数に正しくデータが渡っているか
- □ Information Integrator への入力/出力データ以外の関数のインターフェースを構成する各引数に正しくデータが渡っているか
- □ Information Integrator への出力データとして想定している引数に対して、アプリケーション内でデータを設定する際にデータの扱いが正しいか
  - データの操作時に不当な領域にアクセスしないか
  - データを設定した後のデータ内容に誤りがないか
  - 出力するデータと結果データの領域のデータ型が一致しているか
- □ Information Integrator からの入力/出力データにアクセスする際に使用するローカル変数のデータ型と合っているか
- □ アプリケーション内で扱うデータがテキスト形式の場合、テキストとして扱う処理になっているか
- □ アプリケーション内で扱うデータがバイナリ形式の場合、バイナリとして扱う処理になっているか
  - バイトオーダーが正しいか
  - 数値データに文字列データを設定していないか
  - 文字列データに数値データを設定する際のデータ型が正しいか
  - sprintf 等で使用する書式化文字列と設定する変数のデータ型が正しいか
- □ アプリケーション内で扱うデータがテキスト形式、あるいはバイナリ形式の文字列の場合、文字コードが正しいか
- □ アプリケーション内でスレッドアンセーフである関数を使用していないか

2.3.2 定義の作成手順

定義の作成手順について説明します。以下の順で定義を作成してください。



2.3.2.1 フォーマット定義

アプリケーションで扱うデータの文字コード、レコードタイプ、およびバイトオーダーのフォーマットを決定します。これらをフォーマット定義として定義します。

定義パラメタ	指定値		備考
文字コード	SJMS/S90/U90/UTF8/UTF84		
レコードタイプ	CSV	アプリケーションで扱うデータがテキスト形式である場合に指定します。	
	バイナリ	アプリケーションで扱うデータがバイナリ形式である場合に指定します。	
バイトオーダー	BIG/LITTLE		レコードタイプにバイナリを指定した場合、指定します。

【注意事項】

- アプリケーション内で扱うデータの内容とフォーマット定義に指定する内容が必ず一致するようにしてください。例えば、アプリケーション内で扱うデータのレコードタイプがバイナリ形式の場合、フォーマット定義のレコードタイプに CSV を指定すると、正しく動作しない場合があります。
- アプリケーション内で扱うデータが数値データの場合、アプリケーション内で扱うレコードタイプおよびバイトオーダーと、フォーマット定義の定義値が必ず一致するようにしてください。
- アプリケーション内で扱うデータのレコードタイプが CSV であり、テキスト形式データである場合、アプリケーション内で扱う文字コードとフォーマット定義の文字コードにおいても必ず一致するようにしてください。また、アプリケーション内で扱うデータのレコードタイプがバイナリであり、文字列データである場合も同様です。

なお、項目編集プラグインではフォーマット定義の上記以外のパラメタは参照しません。

2.3.2.2 データ構造定義

データ構造のどのデータからアプリケーションを呼び出すのかを決定します。また、アプリケーションが処理した結果データは呼び出し元のデータ型と一致している必要があります。これらをデータ構造定義として定義します。

アプリケーションで扱うデータのレコードタイプにより、データ構造のデータ型に指定可能な型は異なります。詳細は“システム設計ガイド”を参照してください。

2.3.2.3 プラグイン定義

項目編集プラグインで実行するアプリケーションのインターフェースについて定義します。

1. 作成したアプリケーションの引数全てをプラグイン定義の入力パラメタ情報に定義します。
2. 予めアプリケーションで扱うデータについて定義したフォーマット ID を指定します。Information Integrator の動作環境と同じ文字コードであるテキスト形式データを扱う場合は、フォーマット ID を省略することも可能です。
3. アプリケーションの復帰値や終了確認についてのパラメタを指定します。
4. プラグイン定義の入力パラメタ情報についてのパラメタを指定します。

【注意事項】

- 作成したアプリケーションの引数全てがプラグイン定義の入力パラメタ情報に定義されていることを確認してください。
- アプリケーションで扱うデータとフォーマット ID の定義内容が一致していることを確認してください。
- プラグイン定義に指定した[復帰値の属性]がアプリケーションの復帰値と一致していることを確認してください。
- プラグイン定義の[終了コードの正常異常判断基準]の[範囲指定]と[値指定]の指定内容が、アプリケーションの正常/異常時の動作と矛盾がないことを確認してください。
- プラグイン定義の入力パラメタ情報にある以下のパラメタは、必ず入力してください。省略すると正しく動作できません。
  ・名前
  ・種別
  ・引数型
  ・データ型
  上記以外のパラメタについては各入力パラメタ情報に指定した[種別]に依存しますので、指定した[種別]に従って入力してください。
- プラグイン定義の全ての入力パラメタ情報のうち、[種別]の“結果”、“入力”が必ず1つずつ含まれるようにしてください。
- プラグイン定義の入力パラメタ情報の[種別]が“入力データサイズ”の場合、以下を確認してください。
  －[参照先項目]に指定したパラメタが同じプラグイン定義に存在するか
  －[参照先項目]に指定したパラメタの[種別]が“入力”または“デフォルト”であるか
- プラグイン定義の入力パラメタ情報の[種別]が“出力データサイズ”の場合、以下を確認してください。
  －[参照先項目]に指定したパラメタが同じプラグイン定義に存在するか
  －[参照先項目]に指定したパラメタの[種別]が“出力”または“結果”であるかを確認してください。
- プラグイン定義の[復帰値の属性]に“CharPTR”を指定した場合、[終了コードの正常異常判断基準]の[範囲指定]には“!=”、[値指定]には“0”となっているかを確認してください。
- プラグイン定義の入力パラメタ情報の[種別]が“結果”の[データ長]の値とアプリケーションで出力する結果データの長さが一致しているかどうかを確認してください。

2.3.2.4 データ変換定義

データ変換ファンクションからアプリケーションを実行するための定義方法について以下に説明します。

1. 予め定義済みのデータ構造定義を変換前データ/変換後データのいずれかにおいて読み込みます。
2. 読み込んだデータ構造定義が表示されるので、アプリケーションを呼び出すデータを選択します。
3. 2. で選択したデータのセルにおいて、演算式フィールドの[演算種別]に[プラグイン]を選択した後、[引数]に以下の形式で

指定します。
()内に指定した引数は、指定した順序でアプリケーションに入力データとして受け渡します。したがって、プラグイン定義の入力パラメタ情報の[種別]が[入力]であるパラメタの順序と[演算式]の[引数]が必ず一致するように指定してください。
形式:plugin1(param1,param2,…)

Plugin1	定義済みのプラグイン ID
Param1,param2	プラグイン定義の入力パラメタ情報のうち、[種別]が[入力]のもの

詳細については“デザインシートヘルプ”の“演算式書式”を参照してください。

【確認事項】

デザインシートでは演算式に指定したインターフェースについてチェックされませんので、以下の点を確認してください。

- 読み込んだデータ構造定義のデータ構造のうち、アプリケーションを呼び出すデータの選択が正しいか。
- 手順3で指定したプラグイン ID の[入力パラメタ情報]のうち、[種別]が“結果”であるパラメタの[データ型]と指定元のデータ構造の[データ型]が一致しているか。
- プラグイン定義の[入力パラメタ情報]の[種別]が“入力”である数と手順3に指定した引数の数が一致しているか。
- 手順3に指定した引数の順序が正しいか。
- 手順3に指定した引数に想定したデータが指定されているかを次の観点で確認してください。
  －プラグイン定義の入力パラメタ情報に指定されたデータ型と一致しているか。
  －引数にラベル(@n,¥n)で指定していないか。
  －引数を“$項目名”で指定した場合、“$項目名”のデータ型とプラグイン定義の入力パラメタ情報に指定されたデータ型と一致しているか。
  －引数に固定文字列を指定した場合、ダブルクォーテーションで囲まれているか。
  －引数に固定文字列を指定した場合、対象となる入力パラメタ情報のデータ型が文字列型であるか。
- [変換後データ]の[演算式]にプラグインを組み込む場合、[変換後データ]の[項番(from)]の指定値が正しいか。
- プラグイン定義の入力パラメタ情報の[種別]が“デフォルト”がある場合、[演算式]の[引数]に指定していないか。

2.4 トラブルシューティング

これまでに説明した方法でアプリケーションおよび Information Integrator の定義を作成していても、何らかのエラーが発生して正しく動作できない場合があります。

ここではエラーが発生した場合の原因の特定方法と対処方法について説明します。

2.4.1 エラー原因の特定方法

エラーが発生した場合のエラーメッセージの内容によってエラー原因の特定方法が異なります。なお、エラーメッセージは Windows の場合はイベントビューア、UNIX の場合は Information Integrator の起動コンソール画面またはシステムログに出力されます。

2.4.1.1 DE10047 のメッセージが出力された場合

DE10047 のメッセージが出力された場合、呼出元の Information Integrator または呼出先のアプリケーション内でエラーが発生しています。マニュアルの“メッセージ集”を参照の上、以下の手順で確認してください。

1. DE10047 のメッセージと共に出力されたパラメタ(値が“*(アスタリスク)”でないもの)を確認します。
2. プラグイン ID とアプリケーション名／メソッド名を確認して問題が発生したプラグインを確認します。
3. 復帰値とユーザーアプリケーションからのエラーメッセージを確認します。
4. 実行回数を確認して何回目の実行でエラーが発生したのかを判断します。
   例えば、複数レコードが存在する場合、出力された実行回数によって何番目のデータがエラーが発生した入力データであるかを特定することができます。
5. “エラーが発生した処理を示す番号(D*)”を確認します。
6. “エラーが発生した処理を示す番号(D*)”の内容により、以下の対処を実施します。

“ユーザーアプリケーションのアクセスに関するエラー”の場合

番号	内容	対処
1	ユーザーアプリケーションのライブラリが存在していません。	プラグイン定義の[プラグインパス]に指定したライブラリのパスが正しいかを確認し、誤っている場合はプラグイン定義を修正して再登録してください。
2	ユーザーアプリケーションのライブラリのアクセスでエラーが発生しました。	プラグイン定義の[プラグインパス]に指定したライブラリのアクセス権が正しいか、またライブラリの形式が正しいかを確認してください。 “2.アプリケーションの作成方法”を参照して確認してください。
3	ユーザーアプリケーションのライブラリがディレクトリで指定されています。	プラグイン定義の[プラグインパス]に指定したライブラリのパスにディレクトリで指定されています。プラグイン定義を修正して再登録してください。
4	ユーザーアプリケーションのライブラリのオープンでエラーが発生しました。	プラグイン定義の[プラグインパス]に指定したライブラリのアクセス権が正しいか、またライブラリの形式が正しいかを確認してください。 “2.アプリケーションの作成方法”を参照して確認してください。
5	ユーザーアプリケーションの関数シンボル獲得処理でエラーが発生しました。	プラグイン定義の[メソッド名/関数名]に指定した名称が正しいかを確認し、誤っている場合はプラグイン定義を修正して再登録してください。

“メモリに関するエラー”の場合

番号	内容	対処
10	メモリ不足エラーが発生しました。	一時的なメモリ不足が発生しており、データ変換を行うことができません。 メモリの拡張を行うか、一定時間の経過後に再度、処理プロセスを実行してください。 またアプリケーション内で使用するメモリ量(ローカル変数宣言も含みます)とシステムのメモリ量が妥当であるかも確認してください。

“定義に関するエラー”の場合

番号	内容	対処
11	プラグイン定義の定義内容と入力データ数が合っていません。	プラグイン定義の[入力パラメタ情報]の[種別]が“入力”である数とデータ変換定義の[演算式]の[引数]に指定したデータの数が合っていません。 プラグイン定義の定義内容を確認の上、データ変換定義の[演算式]の[引数]に指定したデータを修正して再登録してください。
12	プラグイン定義のパラメタ情報のパラメタ種別と参照先項目の組み合わせに誤りがあります。	プラグイン定義の[入力パラメタ情報]の[種別]によって以下を確認の上、プラグイン定義を修正して再登録してください。 ・[入力データサイズ]の場合 [参照先項目]に指定したパラメタの[種別]が“入力”または“デフォルト”以外で指定されています。 ・[出力データサイズ]の場合 [参照先項目]に指定したパラメタの[種別]が“出力”または“結果”以外で指定されています。
13	プラグイン定義のパラメタ情報の指定された参照先項目の参照先がありません。	プラグイン定義の[入力パラメタ情報]の[参照先項目]に指定したパラメタがプラグイン定義内に存在していません。 プラグイン定義内のパラメタを指定するように修正して再登録してください。
40	項目編集プラグインで扱えない文字コードで指定されています。	プラグイン定義の[フォーマット ID]で指定したフォーマット定義に以下の文字コード以外のものが指定されています。 フォーマット定義の文字コードには以下のいずれかを指定するように修正して再登録してください。 SJMS S90 U90 UTF8 UTF84

“入力データに関するエラー”の場合

番号	内容	対処
15	CSV からバイナリへの変換に失敗しました。	データ変換機能内ではデータは全て CSV データ(テキストデータ)として扱われます。そのため、アプ

		リケーションで扱うデータがバイナリである場合、アプリケーション呼出前に CSV からバイナリへデータ変換を行います。 データ変換定義の[演算式]の[引数]に指定されたデータが、プラグイン定義の[入力パラメタ情報]の[種別]が“入力”の[データ型]として正しいことを確認してください。 複数レコードが存在する場合、手順 4 で確認した実行回数によってどの入力データがエラーであったのかを特定してください。 例えば、プラグイン定義の[入力パラメタ情報]の[種別]が“入力”の[データ型]が“SQL_INTEGER”であるにも関わらず、データ変換定義の[演算式]の[引数]では数値以外の文字列で指定されていると正しく変換できません。
16	バイナリから CSV への変換に失敗しました。	データ変換機能内ではデータは全て CSV データ(テキストデータ)として扱われます。そのため、アプリケーションで扱うデータがバイナリである場合、プラグイン定義の[入力パラメタ情報]の[種別]が[結果]の[データ型]に応じてアプリケーションから出力したバイナリデータを CSV データへ変換します。 プラグイン定義の[入力パラメタ情報]の[種別]が“結果”の[データ型]としてアプリケーションで出力した結果データが正しいかどうか確認してください。 複数レコードが存在する場合、手順 4 で確認した実行回数によってどの結果データがエラーであったのかを特定してください。 例えば、プラグイン定義の[入力パラメタ情報]の[種別]が“結果”の[データ型]が“SQL_INTEGER”であるにも関わらず、アプリケーションで出力した結果データが文字列であると正しく変換できません。

“実行結果に関するエラー”の場合

番号	内容	対処
20	復帰値に誤りがあります。	プラグイン定義の[復帰値の属性]に“CharPTR”を指定しているにも関わらず、復帰値が NULL となっています。 複数レコードが存在する場合、手順 4 で確認した実行回数によってどのプラグイン実行がエラーであったのかを特定してください。 プラグイン定義の[復帰値の属性]に“CharPTR”を指定すると、[終了コードの正常異常判断基準]の[範

		囲指定]には“!=”、[値指定]には“0”しか指定できません。 復帰値が NULL にならないようにアプリケーションを修正してください。
21	終了コードの正常異常判断基準の範囲外で指定されています。	プラグイン定義の[終了コードの正常異常判断基準]の[範囲指定]と[値指定]の指定内容によって復帰値の値を異常終了と判断しました。 複数レコードが存在する場合、手順 4 で確認した実行回数によってどのプラグイン実行がエラーであったのかを特定してください。 復帰値の値がプラグイン定義の[終了コードの正常異常判断基準]の[範囲指定]と[値指定]の指定内容と合っている場合はアプリケーション内で異常となっているため、アプリケーションの処理を確認してください。また、エラーメッセージが出力されている場合はその内容からアプリケーション内でエラーとなった処理を判断してください。
22	処理結果データにデータが格納されていません。	現状、“エラーが発生した処理を示す番号(D*)”には当番号は出力されません。

“その他のエラー”の場合

番号	内容	対処
30	内部エラー	現状、“エラーが発生した処理を示す番号(D*)”には当番号は出力されません。
32	内部エラー	内部ライブラリのオープンに失敗しました。
33	入力データの文字コード変換でエラーが発生しました。	入力データは以下のように内部で文字コード変換しますので、入力データとして正しいかどうかを確認してください。 ・入力データ (データ変換定義の[変換前/後データ]に指定した[フォーマット ID]に指定された文字コードに準拠) ↓ ・内部入力データ (データ変換定義シートの[変換時の文字コードの扱い]に指定された文字コードに準拠) ↓ ・アプリケーションで扱う入力データ (プラグイン定義に指定した[フォーマット ID]に指定された文字コードに準拠) アプリケーション呼出時に内部入力データの文字コードからプラグイン定義のフォーマット ID に指定した文字コードに対して文字コード変換します。

		複数レコードが存在する場合、手順 4 で確認した実行回数によってどの入力データがエラーであったのかを特定してください。
34	処理結果データの文字コード変換でエラーが発生しました。	処理結果データは以下のように内部で文字コード変換しますので、処理結果データとして正しいかどうかを確認してください。 ・アプリケーションで出力した処理結果データ (プラグイン定義に指定した[フォーマット ID]に指定された文字コードに準拠) ↓ ・内部処理結果データ (データ変換定義シートの[変換時の文字コードの扱い]に指定された文字コードに準拠) ↓ ・処理結果データ (データ変換定義の[変換前/後データ]に指定した[フォーマット ID]に指定された文字コードに準拠) アプリケーションから復帰時に処理結果データをプラグイン定義のフォーマットIDに指定した文字コードから内部処理結果データの文字コードに対して文字コード変換します。 複数レコードが存在する場合、手順 4 で確認した実行回数によってどの結果データがエラーであったのかを特定してください。
38	内部エラー	プラグイン定義の[入力パラメタ情報]の[データ型]にプラグイン定義で扱えないデータ型が指定されています。 プラグイン定義の[入力パラメタ情報]の[データ型]を確認の上、正しい値に修正して再登録してください。

2.4.1.2 DE10047 以外のメッセージが出力された場合

通常は DE10047 以外のメッセージが出力されることはありません。万が一、DE10047 以外のメッセージが出力された場合、呼出先のアプリケーションの構成に誤りがあるか、アプリケーション内でエラーが発生したためにアプリケーション呼出プロセスが異常終了したことが考えられます。

以下を確認してください。

- エラーが発生する直前に当該アプリケーションを呼び出すプラグインが正しく実行されているか
- プラグイン定義の入力パラメタ情報の[種別]が“結果”の[データ長]の値とアプリケーションで出力する結果データの長さが一致しているか
- Information Integrator への出力データとして想定している引数に対して、アプリケーション内でデータを設定する際にデータの扱いが正しいか
  －データの操作時に不当な領域にアクセスしないか
  －データを設定した後のデータ内容に誤りがないか
  －出力するデータと結果データの領域のデータ型が一致しているか

3. 実行形式プラグイン用アプリケーションの作成方法

ここでは、実行形式プラグイン(または処理プラグイン)で利用できるアプリケーションの作成方法について説明します。

3.1 アプリケーションのプログラミング

サンプルソースを流用せず、独自に実行形式プラグイン用アプリケーションを作成する場合に、プログラミングで考慮すべき点について説明します。

・モジュール名は任意です。
・プロセス/スレッド構成は任意です。
・指定可能なパラメタ数は 31 個までです。
・プラグインの終了コードを判定する場合、アプリケーションの復帰コードは、以下の範囲内に収まるようにしてください。
　Windows: -128 から 128 まで
　UNIX　: 0 から 128 まで
・アプリケーション実行時の環境変数は、プラグイン定義に指定した値と、以下の環境変数が有効となります。
　Windows: システムの環境変数
　UNIX　: III サーバ起動時の環境変数
・プラグイン実行中にファンクションの取り消しを実施した場合のアプリケーション終了方法は以下のとおりです。
　Windows: TerminateProcess()による強制終了
　UNIX　: SIGTERM シグナルの送信

3.1.1 Windows における注意点

・Windows Server 2008 以降では、アプリケーションはプラグイン定義で指定した利用者ではなく、SYSTEM で起動されます。
・MFC 等を使ったウィンドウ資源をもつアプリは起動できません(プロセスが起動できてもウィンドウは表示されません)。
・ユーザセッションが起動することで有効となる資源(ネットワークドライブやプリンタなど)はプラグインからアクセスできません。

3.2 アプリケーションのコンパイル

・Information Integrator サーバ環境で実行可能なモジュールを作成できるなら、コンパイラ製品の種類/バージョンは任意です。
・コンパイル/リンケージ方法は任意です。

3.3 Information Integrator の定義作成

Information Integrator からアプリケーションを実行するために定義を作成します。定義を作成する際のポイントを次に説明します。

3.3.1 アプリケーションの動作確認

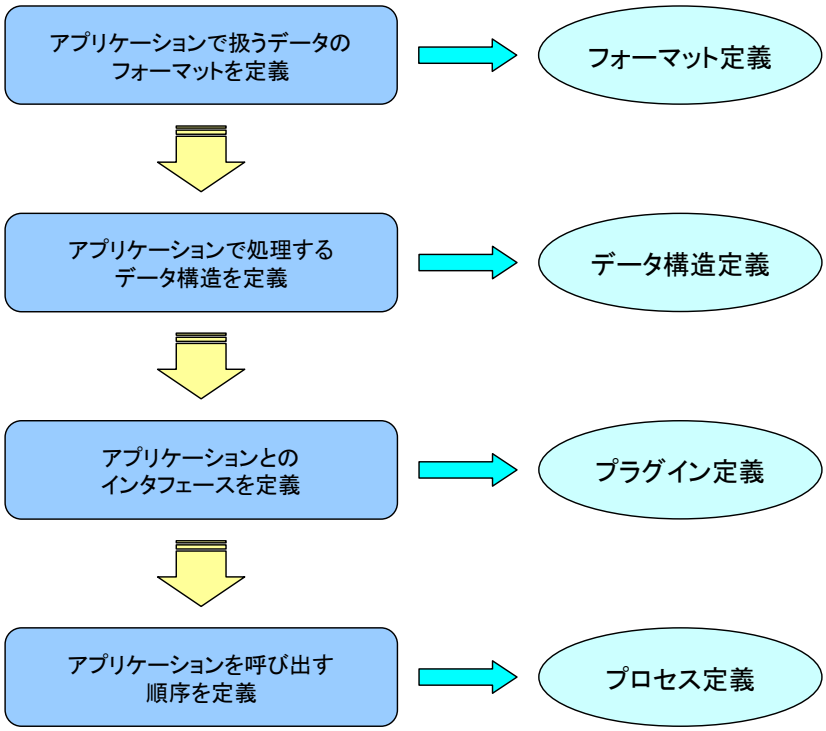
実行形式プラグイン機能と連携する前に、作成されたアプリケーションの動作確認を実施する必要があります。これにより、処理プロセスの実行時に異常が発生した場合においても、アプリケーション内での異常であるのか、または定義方法に誤りがあるのかの切り分けが可能となります。

実行形式ファイルについて予め動作確認を実施してください。
なお、動作確認時には以下の点に注意してください。
(問題が無い場合は□にチェックをいれてください。)

- □ Information Integrator からの入力データとして想定している引数に正しくデータが渡っているか
- □ Information Integrator への出力データとして想定している引数に正しくデータが渡っているか
- □ Information Integrator への入力/出力データ以外の関数のインターフェースを構成する各引数に正しくデータが渡っているか
- □ Information Integrator への出力データとして想定している引数に対して、アプリケーション内でデータを設定する際にデータの扱いが正しいか

3.3.2 定義の作成手順

定義の作成手順について説明します。以下の順で定義を作成してください。



3.3.2.1 フォーマット定義

データをアプリケーションに受け渡す前に文字コード、レコードタイプ、およびバイトオーダーを変更する必要があれば、プラグインのファンクションの前にデータ変換のファンクションを用意し、そこでフォーマット定義を指定します。

定義パラメタ	指定値		備考
文字コード	SJMS/S90/U90/UTF8/UTF84		
レコードタイプ	CSV	アプリケーションで扱うデータがテキスト形式である場合に指定します。	
	バイナリ	アプリケーションで扱うデータがバイナリ形式である場合に指定します。	
バイトオーダー	BIG/LITTLE		レコードタイプにバイナリを指定した場合、指定します。

【注意事項】

- アプリケーション内で扱うデータの内容とフォーマット定義に指定する内容が必ず一致するようにしてください。例えば、アプリケーション内で扱うデータのレコードタイプがバイナリ形式の場合、フォーマット定義のレコードタイプに CSV を指定すると、正しく動作しない場合があります。
- アプリケーション内で扱うデータが数値データの場合、アプリケーション内で扱うレコードタイプおよびバイトオーダーと、フォーマット定義の定義値が必ず一致するようにしてください。
- アプリケーション内で扱うデータのレコードタイプが CSV であり、テキスト形式データである場合、アプリケーション内で扱う文字コードとフォーマット定義の文字コードにおいても必ず一致するようにしてください。また、アプリケーション内で扱うデータのレコードタイプがバイナリであり、文字列データである場合も同様です。

3.3.2.2 データ構造定義

データをアプリケーションに受け渡す前にデータ構造を変更する必要があれば、プラグインのファンクションの前にデータ変換のファンクションを用意し、そこでデータ構造定義を指定します。

3.3.2.3 プラグイン定義

実行形式プラグインで実行するアプリケーションのインターフェースについて定義します。

1. 作成したアプリケーションの引数全てをプラグイン定義の入力パラメタ情報に定義します。
2. アプリケーションの終了確認についてのパラメタを指定します。

【注意事項】

- 作成したアプリケーションの引数全てがプラグイン定義の入力パラメタ情報に定義されていることを確認してください。
- プラグイン定義の[終了コードの正常異常判断基準]の[範囲指定]と[値指定]の指定内容が、アプリケーションの正常/異常時の動作と矛盾がないことを確認してください。

3.3.2.4 プロセス定義

プロセス定義で各ファンクションの種別および実行順序を指定する際に、プラグインの実行順序についても指定します。

4. 項目編集プラグイン用アプリケーションのサンプルについて

サンプルプログラムは Windows 用です。Linux/Solaris で使用する場合はサンプルソースを修正・コンパイルしてください。
サンプルを使用する場合、プラグイン定義は以下のように指定してください。

- 「プラグイン種別」に DLL を指定
- 「フォーマット ID」に文字コードが SJIS、レコードタイプがバイナリ、バイトオーダーが LITTLE のフォーマット定義を指定

サンプルの関数名、入力パラメタ、および処理内容について、次に説明します。

4.1 システム日付／時刻の取得処理（GetSystemDate）

呼び出された際のシステム日付／時刻を取得し、項目値として返却します。
なお、サマータイムは考慮していません。

【関数名】

FIfiJobGetSystemDate

【プラグイン定義の入力パラメタ】

	パラメタの意味	名前	種別	引数型	ﾃﾞｰﾀ長	ﾃﾞｰﾀ型
1	出力バッファへのポインタ	Out	結果	CharPTR	15	SQL_VARCHAR
2	出力バッファサイズ	outsize	入力	Long	—	SQL_INTEGER
3	設定種別 1:日付+時刻 2:日付 3:時刻	type	入力	Long	—	SQL_INTEGER
4	動作フラグ 0:継続 1:新規	flg	入力	Long	—	SQL_INTEGER

【データ変換定義の指定例】

演算種別	引数
プラグイン	PLUGIN_GetSystemDate("9", "2", "1")

【処理内容】

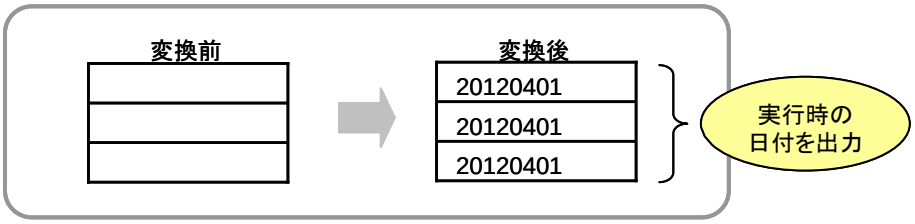
１．日時取得

動作フラグが０の場合は、システム日時格納用変数が空ならシステム日時を取得し、システム日時格納用変数に設定します。システム日時格納用変数が空でなければ何もしません。

動作フラグが１の場合は、呼び出される度にシステム日時を取得し、システム日時格納用変数に設定します。

２．日時設定

設定種別が１の場合は、ＹＹＹＹMMDD　HHMMＳＳの形式で返却用変数に設定します。
設定種別が２の場合は、ＹＹＹＹMMDDの形式で返却用変数に設定します。
設定種別が３の場合は、HHMMＳＳの形式で返却用変数に設定します。



【終了コード】

０：正常
－１：異常

4.2 数値まるめ処理(Round)

数値データに対して、四捨五入／切上げ／切捨てを行います。

【関数名】

FIfiJobRound

【プラグイン定義の入力パラメタ】

	パラメタの意味	名前	種別	引数型	ﾃﾞｰﾀ長	ﾃﾞｰﾀ型
1	出力バッファへのポインタ	out	結果	CharPTR	20	SQL_VARCHAR
2	出力バッファサイズ	outsize	入力	Long	—	SQL_INTEGER
3	変換対象となる入力データ	in	入力	CharPTR	—	SQL_VARCHAR
4	処理種別 1:四捨五入 2:切上げ 3:切捨て	type	入力	Long	—	SQL_INTEGER
5	桁数 -17 ≦ 桁数 ≦ 17	digit	入力	Long	—	SQL_INTEGER

【データ変換定義の指定例】

演算種別	引数
プラグイン	PLUGIN_Round("8", $利率, "1", "1")

【処理内容】

１．数値まるめ

求める桁数となるようまるめ処理を行います。

例）桁数が－１：一の位をまるめて十の位の表示にします。

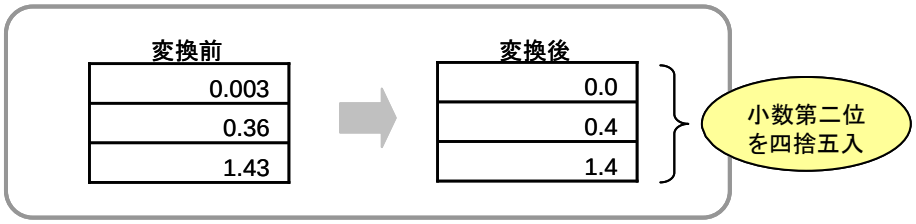
桁数が０：小数第一位をまるめて一の位まで求めます。

桁数が１：小数第二位をまるめて小数第一位まで求めます。

処理種別が１の場合は、四捨五入した値を返却用変数に設定します。

処理種別が２の場合は、切り上げた値を返却用変数に設定します。

処理種別が３の場合は、切り捨てた値を返却用変数に設定します。



【終了コード】

０：正常

－１：異常

4.3 文字列置換処理(ChangeChar)

項目値に含まれる特定の文字列を、別の文字列に置換します。

扱う文字コードは SJIS です。

文字列は 16 進数表記(0x～)も可です。

【関数名】

FIfiJobChangeChar

【プラグイン定義の入力パラメタ】

	パラメタの意味	名前	種別	引数型	ﾃﾞｰﾀ長	ﾃﾞｰﾀ型
1	出力バッファへのポインタ	out	結果	CharPTR	20	SQL_VARCHAR
2	出力バッファサイズ	outsize	入力	Long	—	SQL_INTEGER
3	変換対象となる入力データ	in	入力	CharPTR	—	SQL_VARCHAR
4	置換対象文字列	target	入力	CharPTR	—	SQL_VARCHAR
5	置換後文字列	replacement	入力	CharPTR	—	SQL_VARCHAR

【データ変換定義の指定例】

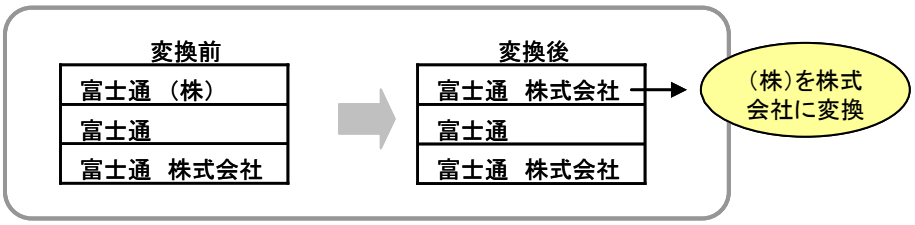
演算種別	引数
プラグイン	PLUGIN_ChangeChar("20", $会社名, "(株)", "株式会社")

【処理内容】

１．文字列置換

項目の先頭から最終まで置換対象文字列かどうかを判定します。

置換対象文字列の場合は、置換後文字列に置き換えます。



【終了コード】

０：正常

－１：異常

4.4 文字列削除処理(DeleteChar)

項目値に含まれる特定の文字列を、項目値から削除します。

扱う文字コードはSJISです。

文字列は16進数表記(0x～)も可です。

【関数名】

FIfiJobDeleteChar

【プラグイン定義の入力パラメタ】

	パラメタの意味	名前	種別	引数型	ﾃﾞｰﾀ長	ﾃﾞｰﾀ型
1	出力バッファへのポインタ	out	結果	CharPTR	20	SQL_VARCHAR
2	出力バッファサイズ	outsize	入力	Long	—	SQL_INTEGER
3	変換対象となる入力データ	in	入力	CharPTR	—	SQL_VARCHAR
4	削除対象文字列	target	入力	CharPTR	—	SQL_VARCHAR

【データ変換定義の指定例】

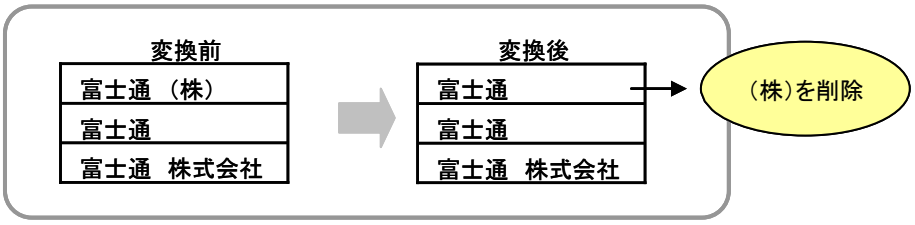
演算種別	引数
プラグイン	PLUGIN_DeleteChar("20",$会社名,"(株)")

【処理内容】

1．文字列削除

項目の先頭から最終まで削除対象文字列かどうかを判定します。

削除対象文字列の場合は削除し、次の文字を詰めます。



【終了コード】

0：正常

－1：異常

4.5 符号付加／除去処理（SignEdit）

文字属性の項目値に対して、符号を付加／除去します。

【関数名】

FIfiJobSignEdit

【プラグイン定義の入力パラメタ】

	パラメタの意味	名前	種別	引数型	ﾃﾞｰﾀ長	ﾃﾞｰﾀ型
1	出力バッファへのポインタ	out	結果	CharPTR	20	SQL_VARCHAR
2	出力バッファサイズ	outsize	入力	Long	—	SQL_INTEGER
3	変換対象となる入力データ	in	入力	CharPTR	—	SQL_VARCHAR
4	処理種別 +:+を付加 -:-を付加 1:符号を除去	type	入力	CharPTR	—	SQL_VARCHAR
5	設定位置 1:前 2:後ろ	position	入力	Long	—	SQL_INTEGER

【データ変換定義の指定例】

演算種別	引数
プラグイン	PLUGIN_SignEdit("8", $売上額, "+", "1")

【処理内容】

１．符号付加（処理種別が＋／－）

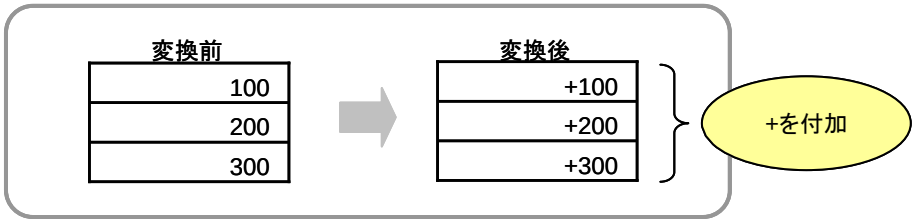
処理種別に指定された符号を、設定位置に指定された箇所に付加します。

２．符号除去（処理種別が１）

設定位置にある＋符号を除去します。

設定位置にある符号が－の場合はエラーとします。

設定位置に符号がない場合は何もしません。



【終了コード】

０：正常

－１：異常

4.6 日付計算処理(CalcDate)

日付データ（ＹＹＹＹMMＤＤ）に対して、加算／減算します。営業日計算のために、ソース上に 2012 年の祝日を設定しています。また、ソースに指定した任意の休日を考慮できます。

【関数名】

FIfiJobCalcDate

【プラグイン定義の入力パラメタ】

	パラメタの意味	名前	種別	引数型	ﾃﾞｰﾀ長	ﾃﾞｰﾀ型
1	出力バッファへのポインタ	out	結果	CharPTR	15	SQL_VARCHAR
2	出力バッファサイズ	outsize	入力	Long	—	SQL_INTEGER
3	入力バッファへのポインタ	in	入力	CharPTR	—	SQL_VARCHAR
4	入力バッファサイズ	insize	入力	Long	—	SQL_INTEGER
5	演算種別 +:加算 -:減算	sign	入力	CharPTR	—	SQL_VARCHAR
6	単位 Y:年 M:月 D:日	order	入力	CharPTR	—	SQL_VARCHAR
7	加算/減算する数値	val	入力	Long	—	SQL_INTEGER
8	営業日計算（演算の単位が日の場合のみ有効） 0:なし 1:あり	biz	入力	Long	—	SQL_INTEGER

【データ変換定義の指定例】

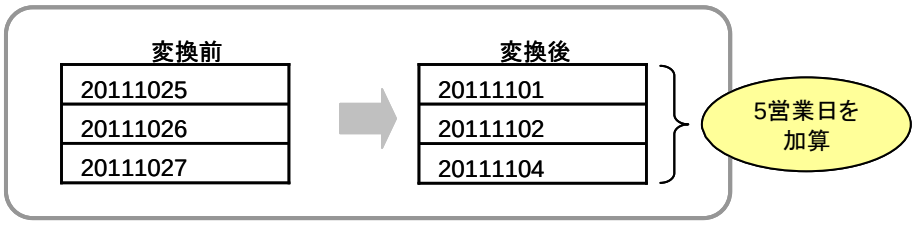
演算種別	引数
プラグイン	PLUGIN_CalcDate("9", $日付, "9", "+", "D", "5", "1")

【処理内容】

１．日付計算

項目値に対して数値を加算／減算します。

営業日計算が「あり」の場合は、営業日を考慮して加算／減算します。



【終了コード】

０：正常

－１：異常

4.7 和暦／西暦変換処理（ConvJcAd）

日付データに対して、西暦から和暦、または和暦から西暦に変換します。

西暦の形式：ＹＹＹＹMMＤＤ

和暦の形式：ｘＹＹMMＤＤ（ｘの値は、Ｈ：平成、Ｓ：昭和、Ｔ：大正、Ｍ：明治）

年号は以下のように取り扱います。

明治：1868/09/08～1912/07/29

大正：1912/07/30～1926/12/24

昭和：1926/12/25～1989/01/07

平成：1989/01/08～

【関数名】

FIfiJobConvJcAd

【プラグイン定義の入力パラメタ】

	パラメタの意味	名前	種別	引数型	ﾃﾞｰﾀ長	ﾃﾞｰﾀ型
1	出力バッファへのポインタ	out	結果	CharPTR	15	SQL_VARCHAR
2	出力バッファサイズ	outsize	入力	Long	—	SQL_INTEGER
3	入力バッファへのポインタ	in	入力	CharPTR	—	SQL_VARCHAR
4	入力バッファサイズ	insize	入力	Long	—	SQL_INTEGER
5	処理種別 1:西暦⇒和暦 2:和暦⇒西暦	type	入力	Long	—	SQL_INTEGER

【データ変換定義の指定例】

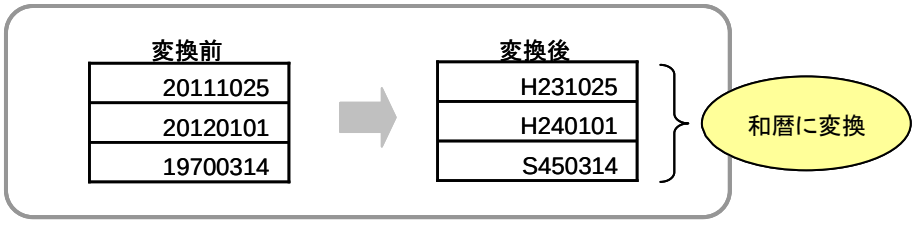
演算種別	引数
プラグイン	PLUGIN_ConvJcAd(″15″, $日付, ″8″, ″1″)

【処理内容】

１．変換処理

西暦⇒和暦、または和暦⇒西暦に変換します。

変換できない場合は、項目値に空白を設定してエラーとします。



【終了コード】

０：正常

－１：異常

4.8 GS日付形式変換処理(ConvGSDate)

ジュリアン日付形式から一般の日付形式、または一般の日付形式からジュリアン日付形式に変換します。

ジュリアン日付形式：ＹＹＤＤＤ（ＤＤＤは通し番号）

一般の日付形式　　：ＹＹＹＹMMＤＤ

【関数名】

FIfiJobConvGSDate

【プラグイン定義の入力パラメタ】

	パラメタの意味	名前	種別	引数型	ﾃﾞｰﾀ長	ﾃﾞｰﾀ型
1	出力バッファへのポインタ	out	結果	CharPTR	15	SQL_VARCHAR
2	出力バッファサイズ	outsize	入力	Long	—	SQL_INTEGER
3	入力バッファへのポインタ	in	入力	CharPTR	—	SQL_VARCHAR
4	入力バッファサイズ	insize	入力	Long	—	SQL_INTEGER
5	処理種別 1:ジュリアン⇒一般 2:一般⇒ジュリアン	type	入力	Long	—	SQL_INTEGER

【データ変換定義の指定例】

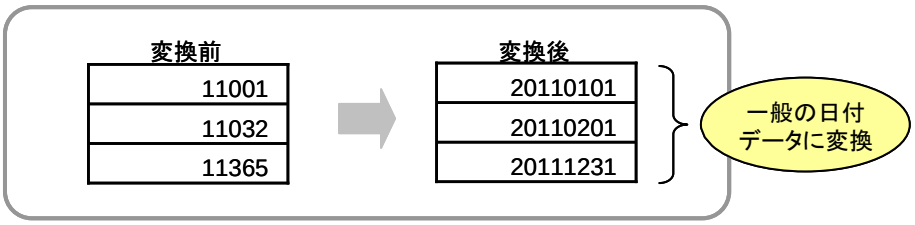
演算種別	引数
プラグイン	PLUGIN_ConvGSDate("15", $日付, "7", "1")

【処理内容】

１．変換処理

ジュリアン日付形式⇒一般の日付形式、または一般の日付形式⇒ジュリアン日付形式に変換します。

変換できない場合は、項目値に空白を設定してエラーとします。



【終了コード】

０：正常

－１：異常

4.9 有効範囲チェック処理（CheckRange）

数値データが有効範囲かどうかをチェックします。

【関数名】

FIfiJobCheckRange

【プラグイン定義の入力パラメタ】

	パラメタの意味	名前	種別	引数型	ﾃﾞｰﾀ長	ﾃﾞｰﾀ型
1	出力バッファへのポインタ	out	結果	IntPTR	4	SQL_INTEGER
2	入力データ	in	入力	Long	—	SQL_INTEGER
3	有効範囲（最小値） ※下限は long 型で表現できる最小値	min	入力	Long	—	SQL_INTEGER
4	有効範囲（最大値） ※上限は long 型で表現できる最大値	max	入力	Long	—	SQL_INTEGER
5	エラー動作 0:中断 1:継続	flg	入力	Long	—	SQL_INTEGER
6	継続時の設定値 ※long 型で表現できる値	val	入力	Long	—	SQL_INTEGER

【データ変換定義の指定例】

演算種別	引数
プラグイン	PLUGIN_CheckRange($予算値, "0", "1000", "1", "0")

【処理内容】

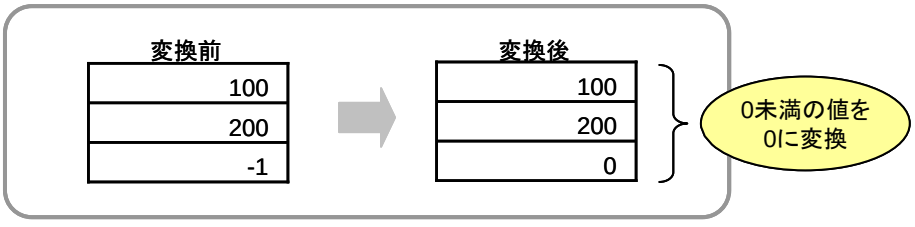
１．チェック処理

項目値が有効範囲（最小値～最大値）かどうかをチェックします。

有効範囲でない場合、以下の処理を行います。

・エラー動作が「中断」ならエラーにします。

・エラー動作が「継続」なら継続時の設定値を返却し、エラーにしません。



【終了コード】

０：正常

－１：異常

5. 実行形式プラグイン用アプリケーションのサンプルについて

サンプルプログラムは Windows 用です。Linux/Solaris で使用する場合はサンプルソースを修正・コンパイルしてください。
サンプルの入力パラメタ、および処理内容について、次に説明します。

5.1 集計処理（Aggregate）

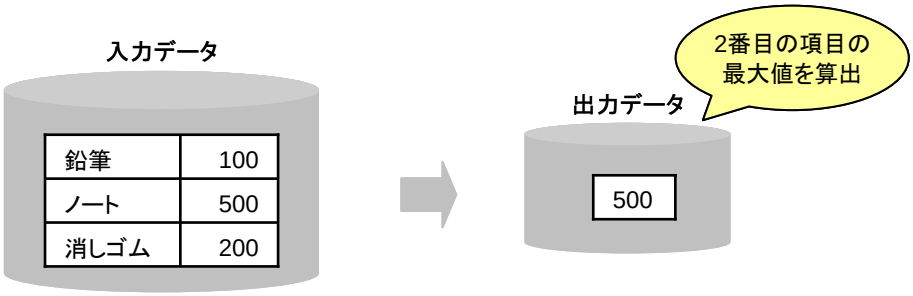
CSV 形式のファイル内にある項目の最大値／最小値／平均値／件数を集計します。

【プラグイン定義の入力パラメタ】

１．入力データファイル名
２．出力データファイル名
３．項目番号
４．集計種別
　　１：最大値（ＭＡＸ）
　　２：最小値（ＭＩＮ）
　　３：平均値（ＡＶＧ）
　　４：件数（ＣＯＵＮＴ）

【処理内容】

１．集計処理
　　集計種別の指定値に従って集計処理を行い、出力ファイルに集計結果を出力します。



【終了コード】

０：正常
－１：異常

5.2 レコード重複チェック処理（CheckDup）

CSV 形式、または固定長テキスト形式のファイル内にあるレコードが重複しているかどうかをチェックします。

【プラグイン定義の入力パラメタ】

１．入力データファイル名

２．出力データファイル名

３．エラー動作

０：中断

１：継続

【処理内容】

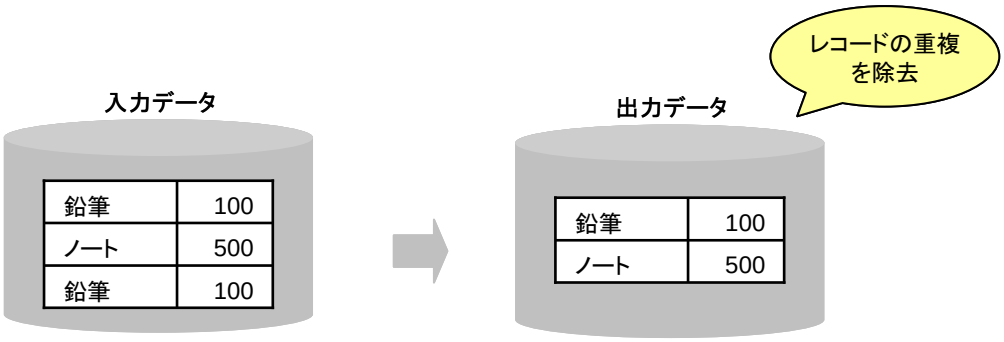
１．チェック処理

レコードの重複チェックを行います。

重複する場合、以下の処理を行います。

・エラー動作が「中断」ならエラーにします。

・エラー動作が「継続」なら、出力ファイルに重複レコードを取り除いたデータを出力します。



【終了コード】

０：正常

－１：異常

5.3 エラーデータ除去処理（RemoveErrorData）

入力データファイルに含まれているエラーデータを除去し、出力データファイルに出力します。

【プラグイン定義の入力パラメタ】

１．入力データファイル名

２．エラーデータファイル名（※）

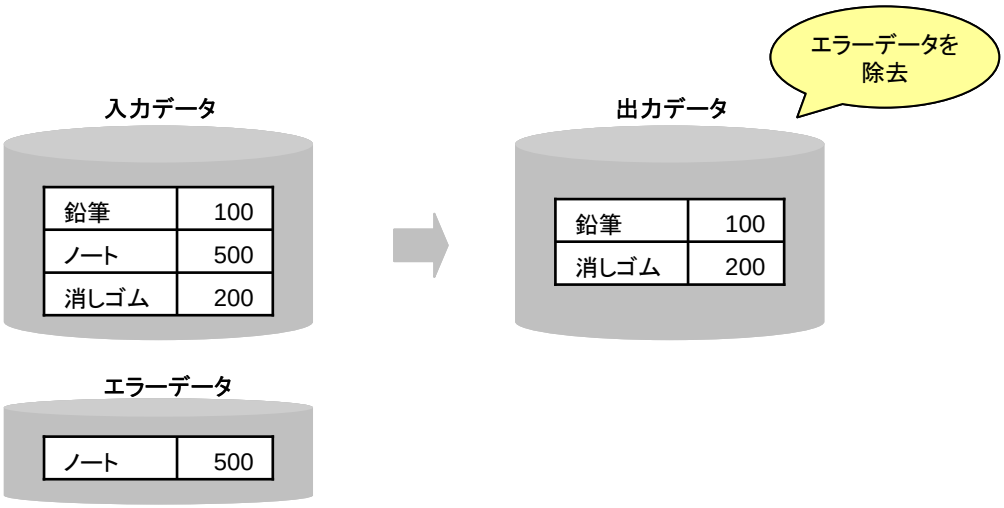
３．出力データファイル名

※）データ変換定義の「エラーデータ出力ディレクトリ」に指定したディレクトリに作成されているファイルの名前です。

【処理内容】

１．チェック処理

入力データファイルから１レコードずつ取り出し、エラーデータファイルに存在しなければ、出力データファイルにそのレコードを出力します。



【終了コード】

０：正常

－１：異常