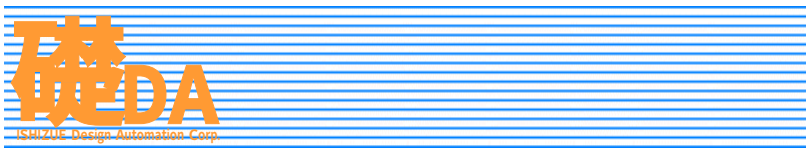

*** FP-Fixer module manual ***

FP-Fixer

- 解析モデル生成機能 マニュアル -

Ver 1.00.01

株式会社 礎デザインオートメーション

目次

1. 解析モデル生成機能マニュアルについて

本マニュアルでは、v2.08.00 より FP-Fixer に新機能として追加された解析モデル自動生成機能などについて、その特徴や内容、使用方法等を説明します。
FP-Fixer の基本的な使用方法をご理解頂いていることが前提となりますので、初めて FP-Fixer をご使用になる方は、別紙ユーザーズマニュアルをお読み頂いた上で本書をお読み下さい。

2. 解析モデル生成機能版の特徴

FP-Fixer v2.08.00 より固定小数点化対象（//fpfix module）を抽出して、その部分のみを解析するように変更しています。（解析独自のメイン関数を作成）

①プロファイル処理で、入力パターン及び期待値を観測してデータファイルを出力します。
②解析処理は、プロファイル処理で作成した入力パターン及び期待値を基に小数部ビット幅の解析を行います。
③解析の評価関数は、解析独自の評価関数を使用します。
なお、ユーザ作成の評価関数に置換え可能です。

以下、FP-Fixer v2.08.00 より追加された各機能について説明します。

2. 1 解析モデル自動生成機能

FP-Fixer は、関数単位で固定小数点化をおこないますが、従来のバージョン(v2.07.xx まで)では、ユーザがディレクティブ（//fpfix module）で指定した解析対象関数はオリジナル C ソース上で解析されるため、場合によってはユーザが独自で作成したテストベンチにユーザが解析対象関数を抽出する必要がありました。

FP-Fixer の解析モデル自動生成機能は、ユーザがディレクティブ（//fpfix module）で指定した解析対象関数をオリジナル C ソースから自動で抽出し、ツール内部でもつ解析用テストベンチで解析を行い、解析後の C ソースをオリジナル C ソースへ戻します。

本機能により、ユーザは固定小数点化したい関数をディレクティブ（//fpfix module）で指定し許容誤差指定を行うだけで、基本的な固定小数点化は行うことができるようになりました。また、オリジナル C ソース内の解析対象とならない部分が大きい場合には、解析速度の向上が期待できるようになりました。

2．2　評価関数の自動生成機能

従来のバージョン(v2.07.xx まで)では、固定小数点化する際の判断基準を評価関数という形でユーザが C 言語により記述する必要がありましたが、解析モデル生成機能版(v2.08.xx)では、評価関数を「//fpfix evFunc」という形で必ずしも記述する必要はなくなります。

ぎりぎりまで有効ビット幅を小さくしたい場合やパフォーマンスを少しでも向上させたい場合には、デザインや入力データ等にあわせて評価関数で細かく評価基準を指定できることは必要不可欠でありまた大きなメリットですが、精度等をあまり気にせずに単に固定小数点化したい場合には、評価関数の作成は手間となることもあります。

FP-Fixer の評価関数自動生成機能は、解析モデル自動抽出機能の補完機能として用意されているツール内部でもつ評価関数をユーザが利用できるようにしたものです。

本機能により、ユーザは、解析対象関数の指定（//fpfix module）と許容誤差（エラーレート）の指定をするだけで基本的な固定小数点化を行うことが可能になりました。

evfunc 属性指定をすることで、従来通りユーザ作成の評価関数を指定することが可能です。

2．3　ポート定義ディレクティブ

解析モデル自動抽出機能は解析対象関数を抽出しますので、解析対象関数の入力と出力を明確にする必要があります。
ユーザは、解析対象関数の引数に、入力ポートディレクティブ（// fpfix input）及び出力ポートディレクティブ（// fpfix output）で必ず入力及び出力変数を指定する必要があります。

このポート定義ディレクティブには、ダイナミックレンジ指定機能（drang 属性）などいくつか有効な機能が追加されています。

2．4　期待値観測機能

FP-Fixer では、期待値と解析中の出力を評価関数で許容誤差範囲内か否かを判断しながら解析を行いますが、従来のバージョン（v2.07.xx まで）では、期待値は必ずユーザが用意する必要がありました。

新規に追加された期待値観測機能は、プロファイル処理（-i）で入力パターン及び期待値を観測してデータファイルを出力しますので、ユーザは期待値を用意する必要がなくな

りました。

2.5 制約条件ファイル

FP-Fixer の従来のバージョン(v2.07.xx まで)では、解析対象関数指定(//fpfix module)、評価関数指定（//fpfix evFunc）、変数毎のビット精度等の強制指定（//fpfix fixedP）などのディレクティブは、ソースコード中に直接記述する必要がありました。

新機能として追加された制約条件ファイルは、上記のような指定記述をテキストファイルなどに記述することで可読性の向上、運用のし易さを実現しています。

3．解析モデル生成機能概要図

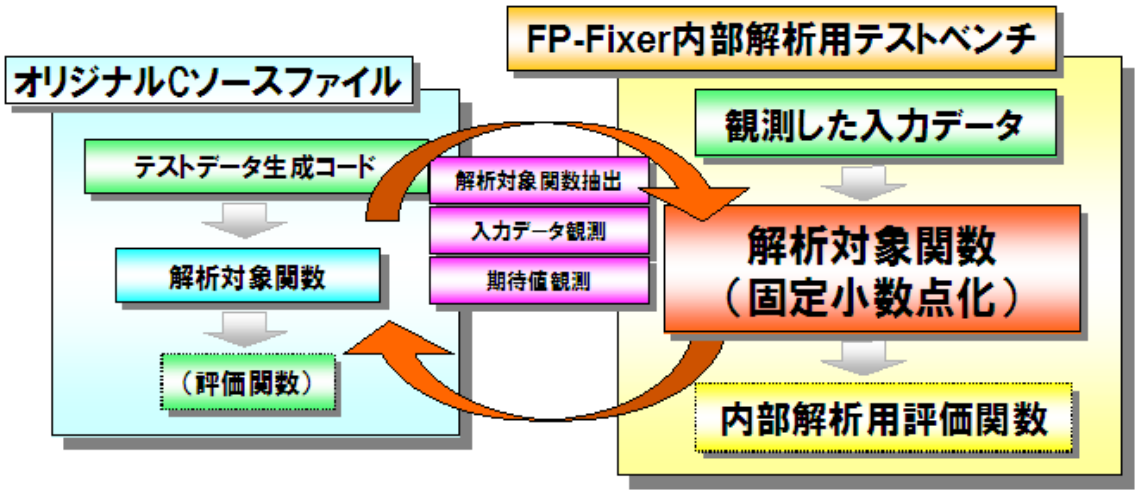

図 1 解析モデル生成機能概要図

4. 解析モデル生成機能における C 言語記述方法

4. 1 評価関数を自動生成する場合

4. 1. 1 module ディレクティブの記述

固定小数点化対象（以下、解析対象）とする関数をコールしている箇所に「module」ディレクティブを記述して下さい。

例）
```
int main( int argc, char *argv[] )
    {
        .....
        fftfp(inbuf,outbuf_sfftfp); // fpfix module
        .....
    }
```

【参照】example/fft_1/main.c の main 関数内参照

4. 1. 2 入出力ディレクティブの記述

解析対象関数の引数及びグローバル変数に、入力及び出力のディレクティブを記述して下さい。入出力ディレクティブの詳細は、「5 章ポート定義ディレクティブ」を参照してください。

入出力のディレクティブは、
①入力 →　　// fpfix input ..
②出力 →　　// fpfix output ..
③入出力 →　// fpfix inout ..
となります。

なお、ディレクティブを記述しない場合、
①実態（値）受け　→　input
　但し、構造体でポインタメンバがある場合は、inout
②ポインタ受け　→ inout
③グローバル変数 → inout
と、なります。

例１）
```
    ...
    target( A, B, &Y ); // fpfix module
```

```
....

void target(
      double A;  // fpfix input
      double B;  // fpfix input
      double *Y; // fpfix output rate:5:m
){
    ...
}
```

例２）構造体の場合
構造体の場合、全てのメンバーが同じ入出力の場合（①）と
メンバー毎の指定（②）が可能です。

```
struct _tag1 {
      double width;
      double height;
      double pixels[128];
}

...
target( &A, &Y ); // fpfix module
....

void target(
      struct _tag1 *A;  // fpfix input varname=A           ①
      struct _tag2 *Y;  // fpfix input varname=Y.width     ②
                        // fpfix input varname=Y.height
                        // fpfix output varname=Y.pixels rate=5:m
){
    ...
}
```

【参照】example/fft_1/fft_1_fp.c
【参照】example/rgbycc/rgbycc.c の rgbycc、yccrgb 関数参照

※解析対象の入力となる変数の入力不定
出力変数の出力不定がある場合、解析でストップする場合があります。
特に配列の一部不定等には気を付けて下さい。
変数の初期化をお勧めします。

4.1.3 評価関数の作成

評価関数は、プロファイル処理で作成された期待値データと解析中の出力データを比較

して指定のエラーレートに収まるかどうか評価します。
次のように、入出力ディレクティブの rate 属性によりコールする評価関数が異なります。

①// fpfix output rate=5:m
評価関数 ：evFunc_$rate((期待値), (解析データ), 0.05); // %で比較

②// fpfix output rate=5:a
評価関数 ：evFunc_$add((期待値), (解析データ), 5); // +-で比較

(1)自動評価関数
「$FPFIX/evfunc/fpf_evfunc.tmpl」をカレントディレクトリへコピーして下さい。
カレントに fpf_evfunc.tmpl が無い場合は、ツールが自動でコピーします。

output 及び inout ディレクティブの rate 属性により、デフォルトでカレントディレクトリの「fpf_evfunc.tmpl」内の「evFunc_$rate()」または「evFunc_$add()」関数をコールします。

(2)評価関数の変更
「fpf_evfunc.tmpl」内の evFunc_$rate()または evFunc_$add()関数内を変更してください。

(3)評価関数の追加
「fpf_evfunc.tmpl」内の evFunc_$rate()または evFunc_$add()関数の引数と引数を同じにして、独自の関数を「fpf_evfunc.tmpl」に追加して下さい。
追加後、output または inout ディレクティブに、evfunc 属性を追加して下さい。

// fpfix output rate=5:m evfunc=<追加した評価関数名>

【参照】example/rgbbycc/rgbycc.c の rgbycc、yccrgb 関数参照

※出力に rate 属性が指定されていない場合
「rate=0.5:m」とみなし評価関数をコールします。
fpfix 起動時のオプション「-erate」で指定されている場合は、
その内容で評価関数をコールします。

※DSP、Algoc、Ansic 出力時、評価関数コール用のテンプレート「fpf_evfunc.h」を「c」ディレクトリに作成します。
このテンプレートに従って、出力後の C コードを編集して評価することが出来ます。

4. 2 評価関数を独自に作成する場合

4. 2. 1 module ディレクティブの記述

解析対象とする関数をコールしている箇所に「module」ディレクティブに加え
例のように evfunc 属性で評価関数を指定して下さい。

```
例) int main( int argc, char *argv[] )
    {
        .....
        fftfp(inbuf,outbuf_sfftfp); // fpfix module evfunc=cmpBuf
        .....
    }
```

【参照】example/fft_1_uev/main.c の main 関数内参照

4. 2. 2 入出力ディレクティブの記述

解析対象関数の引数及びグローバル変数に、入力及び出力のディレクティブを記述して下さい。記述の仕方は、「１． ２ 入出力ディレクティブの記述」と同様です。
異なる点は、rate 属性及び evfunc は読み飛ばします。

【参照】example/fft_1_uev/fft_1_fp.c
【参照】example/rgbycc_uev/rgbycc.c

4. 2. 3 評価関数の作成

module ディレクティブの evfunc 属性で指定した関数名で評価関数を作成します。
評価関数は、ソースコード内に記述するか、カレントディレクトリに「<指定関数名>.h」で作成してください。
ソースコード内の記述をお勧めします。

【参照】example/fft_1_uev/main.c
【参照】example/rgbycc_uev/rgbycc.c
example では、C 出力後の評価を同じ関数で行う為評価関数をコールする
記述がありますが、コールする記述は無くてもかまいません。

(1)評価関数の引数
評価関数の引数は、必ず、期待値、解析対象からの出力の順で記述する必要があります。
複数の出力がある場合は、
(期待値１),(出力１),(期待値２),(出力２)の順
で作成してください。

例１）

```
target(in,&out); //fpfix module evfunc=myEvfunc

// 評価関数
int myEvfunc( double exp_out, double tg_out )
{
    ..
}
```

例２）出力が複数ある場合

```
target(in, &out1, &out2 ); //fpfix module evfunc=myEvfunc

// 評価関数
int myEvfunc(
      double exp_out1, double tg_out1,
      double exp_out2, double tg_out2,
){
    ..
}
```

例３）出力が構造体の場合

```
typedef struct _str {
   ..
} T_STR;

T_STR in, out;
....
target(in, &out ); //fpfix module evfunc=myEvfunc

// 評価関数
int myEvfunc( T_STR exp_out1, T_STR tg_out1 ){
    ..
}
```

例４）出力が配列の場合

```
T_STR in, out[16];
....
target(in, out); //fpfix module evfunc=myEvfunc

// 評価関数
int myEvfunc( T_STR exp_out1[], T_STR tg_out1[] ){
    ..
}
```

5．ポート定義ディレクティブ

解析対象関数の引数やグローバル変数に対して、入出力等を定義するディレクティブです。
ポート定義ディレクティブが付加されていない変数は、
・実態（値）渡し → 入力ポート
　　但し、構造体でその構造体メンバーにポインタが含まれている場合は入出力
・アドレス渡し　→ 入出力ポート
・グローバル変数 → 入出力
と見なします。

また、出力でエラーレートが指定されていない場合、デフォルトで 0.5%として処理します。
デフォルトのエラーレートは、fpfix 起動時の「-erate」オプションで変更可能です。

5．1　入力ポートディレクティブ

解析対象関数の入力となる引数やグローバル変数に対して以下のようにポートを定義します。

```
// fpfix input varname=<varName> fixedP=< <S|U> <bitW>:<fsize> <ovf>.<rnd>>
            drang=<pmaxValue, pminValue, nmaxValue, nminValue>
```

(1)fixedP 属性
・この属性は、浮動小数点型の double、float に指定できます。
・double、float 型で、fixedP が省略されるときはビット精度解析対象となります。

(2)drang 属性
・この属性は入力データとして取り得るプラス方向の最大値、最小値、またマイナス方向の最大値、最小値を指定します。
・この属性が省略された場合、ダイナミックレンジは入力データに依存します。
・符号なし（unsigned の U 指定）変数の場合、nmax 及び nmin は省略できます。

(3)varname 属性
・この属性は、引数またはグローバル変数が構造体の場合、メンバーの名称を記述します。
・通常変数（構造体でない）の場合は省略できます。

例）
```
struct _sampl {
        double member1;
        double member2;
    };
```

```
void func_fix(
       struct _sampl in; // fpfix input varname=in.member1 ...
                         // fpfix input varname=in.member2 ...
       ....
{ .. }
```

5. 2 出力ポートディレクティブ

解析対象関数の出力となる引数やグローバル変数に対して以下のようにポートを定義します。

```
// fpfix output varname=<varName>
          fixedP=< <S|U> <bitW>:<fsize> <ovf>.<rnd>> rate=<value> <m|a>
          evfunc=<nameOfFunc>
```

(1)fixedP、varname 属性は、入力ポートディレクティブと同様です。

(2)rate 属性は、評価の為の出力誤差を指定します。

```
<value>   : <m|a>が m の時、パーセントで指定する
                  a の時、誤差値を指定する。
<m|a>     : m の時、次の計算により誤差範囲を確認する。
              up = ext * (1+(<value>/100);
              lo = ext * (1-(<value>/100);
              if( up >= ret && lo <= ret ) OK ;

             a の時
              up = ext + <value>;
              lo = ext - <value>;
              if( up >= ret && lo <= ret ) OK ;
```

(3)evfunc 属性は、評価関数を追加した場合にその関数名を指定します。

5. 3 入出力ポートディレクティブ

解析対象関数の入出力となる引数やグローバル変数に対して以下のようにポートを定義します。

```
// fpfix inout varname=<varName> fixedP=< <S|U> <bitW>:<fsize> <ovf>.<rnd>>
          rate=<value> <m|a>
```

drang=<pmaxValue pminValue nmaxValue nminValue>
r-turn

(1)各属性は、入力及び出力ディレクティブと同様です。

(2)r-turn 属性
解析対象からの出力をそのまま入力として扱う場合指定します。
1 回目のデータは、プロファイル取得したデータを使用します。
2 回目以降は、解析対象からの出力を入力データとして解析します。

通常は、プロファイルで取得したデータを入力としますが、解析対象の出力とプロファイルの入力データが一致しない場合があります。この場合、解析結果と、DSP や Algoc、Ansic の出力後の結果が合わない場合が発生してしまいます。r-turn 属性を指定することによりこの問題を解決します。

6. 制約条件ファイル

制約条件ファイルは、次の指定が可能です。

①起動時のオプション
②ソースに記述するディレクティブ

fpfix 起動時に「-ctf <nameOfFile>」の指定で、制約条件ファイルを読込み、設定内容を有効とします。

優先順位
1.ソースに記述(最優先)
2.起動オプション
3.制約条件ファイルの内容

6.1 制約条件定義フォーマット

```
// 起動オプション
$def_start_option ;
  -max  <maxBitSize> ;    // 最大許容ビット幅指定
  -maxt <maxBitSize> ;    // 乗算演算結果最大許容ビット幅指定
  -p    <prefix> ;        // 出力ファイル名のプレフィックス指定
  -rnd  <t|r|e> ;         // 丸め指定
  -ovf  <i|s> ;           // オーバーフロー指定
  -02 | -01 | -00 ;       // 解析等のコンパイルオプティマイズオプション
                          //   gcc に準ずる
  -simmode ;              // 解析順番を負荷の高い順に実行
  -mul_func ;             // 左詰め乗算を使用する
  -div_func ;             // 割算引き放し法関数使用（DSP 出力のみ有効）
  -simdbg ;               // デバッグ用（isz_work/simdbg.txt を出力）
  -simdbgf <funcName> ;   // デバッグ用（指定関数内の情報のみ出力）
  -D <string> ;           // プリプロセッサマクロを指定
  -stdmath ;              // math 関数を変換しない
  -lp <userlibPath> ;      // ユーザライブラリパス指定
  -ip <userIncludePath> ; // ユーザライブラリインクルードパス指定
  -erate ;            // デフォルトエラーレート指定
  -gwl ;              // 解析中ループオーバフロー警告抑制
  -gm0 ;              // 解析中ジョブメッセージ抑制
  -gm1 ;              // 解析中変数名表示抑制
```

```
  -gm2 ;              // 解析中ビット幅表示抑制
$end_start_option ;

$def_directive ;
  # 関数コールに付加
  module <fileName> <funcName> [prefix=<string>] [evfunc=<funcName>];
  roll   <fileName> <funcName> ;

  # 変数関連
  fixedP <fileName> <funcName> <varName> <fixedP ﾃﾞｨﾚｸﾃｨﾌﾞﾌｫｰﾏｯﾄ> ;
  int    <fileName> <funcName> <varName> <int ﾃﾞｨﾚｸﾃｨﾌﾞﾌｫｰﾏｯﾄ> ;
  input  <fileName> <<funcName> <varName> <input ﾃﾞｨﾚｸﾃｨﾌﾞﾌｫｰﾏｯﾄ> ;
  output <fileName> <<funcName> <varName> <output ﾃﾞｨﾚｸﾃｨﾌﾞﾌｫｰﾏｯﾄ> ;
  inout  <fileName> <<funcName> <varName> <inout ﾃﾞｨﾚｸﾃｨﾌﾌﾞｫｰﾏｯﾄ> ;
$end_directive ;
```

※グローバル変数の場合、<funcName>は、$記述

7. 制限及び注意事項

解析モデル生成機能版（v2.08）の制限及び注意事項を記述します。

7. 1 従来のバージョン(v2.07.xx)との互換性

v2.07 の評価関数との互換がありませんので、v2.08 用の評価関数に書き換える必要があります。また、//fpfix evFunc ディレクティブは読み飛ばします。

v2.07 のデータをそのまま実行したい場合は、「-mev」オプションを指定して下さい。(v2.07 互換モード)

7. 2 多項式記述の注意

期待値は、コードエラボレーション後のプロファイル収集処理（-i）で作成します。
多項式は、コードエラボレーション（-s）で２項式に変換されますので、多項式の時と値が少し異なる場合があります。

したがって、DSP、Algoc、Ansic 出力後の C を評価する場合、多項式での結果を期待値として実行した場合、解析結果と合わない場合が発生します。ご注意下さい。

7. 3 解析対象の入出力不定

解析対象の入力変数の入力不定及び出力変数の出力不定がある場合、プロファイル収集処理で、不定データからオーバフローとみなしてしまい必要ビット幅を大きく見積もり、解析でストップする場合があります。
特に配列の一部不定等に注意してください。

8．トラブルシューティング

FP-Fixer を利用中、ルールに従っているのにうまく動作しない場合は以下の点を確認してください。また「７．制限及び注意事項」や別紙ユーザーズマニュアル「３．制限事項」の内容もご確認ください。

◆ 自分で作成した評価関数で許容誤差レートをいくら広げても解析でビット幅が足りなくなる

評価関数の evfunc 属性は正しく設定されていますでしょうか。evfunc 属性を設定しない場合、ツール内部の評価関数(デフォルトのエラーレートは 5%)が有効となってしまいユーザ指定の評価関数は無効となってしまいます。

評価関数の評価基準は適切でしょうか。例えば、限りなく 0 に近い入力データがある場合などでビット幅が多く必要となっている場合、その値についてレートをいくら広げても出力全体として大きいビット幅が必要となってしまう場合があります。そのような値がくる場合には、0 とみなしたり、評価したい範囲（上限と下限）を設定しその範囲内でエラーレートで評価するなど評価方法を再度検討して下さい。

評価関数の作成方法についてのお問合せはサポート窓口までご連絡下さい。

◆ 出力した結果（C ソースをコンパイル実行）すると期待値と結果があわない

「７． 2 多項式記述の注意」にありますように、解析モデル生成機能では期待値をコードエラボレーション（多項式を二公式に分解）後に観測するため、出力後の C を評価する場合、多項式での結果を期待値として実行した場合、解析結果と合わない場合が発生します。

◆ 解析モデル生成機能版（v2.08 系）で実行した場合には正常に解析が完了したが、v2.07 系互換モード（-mev）で実行すると解析でビット幅が足りなくなる

従来のバージョン（v2.07 系まで）には、「５． 3 入出力ポートディレクティブ (2)」にあるような r-turn 属性はサポートしていません。r-turn 属性は、解析対象からの出力をそのまま入力として扱う場合に指定します。

（1 回目のデータは、プロファイル取得したデータを使用し、2 回目以降は、解析対象からの出力を入力データとして解析します。）

出力データを次の入力として使用するデザインについては、v2.07 系互換モードではビット幅が大きくなる傾向がありますのでご注意下さい。

（スタイルチェックでははじけない項目です）

◆ 解析は正常に完了したが有効ビット幅をもっと小さくしたい。まずはどういった方法をとればよいか。

有効ビット幅を大きくする要因は、主に

① 除算
② 乗算
③ 積和演算
④ 入力データ
⑤ 丸め
⑥ アルゴリズム

などがあります。

①除算については、固定小数点化を行う場合には出来る限り使用しないように推奨していますが、FP-Fixer には引き戻し法除算（-div_func オプション）がありますので、オプションをつけて実行してみる方法があります。但し、このオプションは実行速度を低下させるため使用する際にはご注意下さい。

②乗算については、もし DSP をお使いの場合、ご使用の DSP の乗算の機能についてご確認下さい。もし 32 ビット以上の乗算を扱える機能を搭載している場合には、-maxt オプションを使用して、32 ビット以上の乗算を DSP にマッピングすることが可能です。

また、速度を落としてでもビット幅を小さくしたい場合には、FP-Fixer には左詰乗算オプション（-mul_func）がありますので、オプションをつけて実行してみる方法があります。但し、このオプションは実行速度を低下させるため使用する際にはご注意下さい。

④大きなビット幅が必要となるデータがきている変数を特定し、固定小数点定義ディレクティブ（//fpfix fixedP など）で強制的にビット幅を指定する方法があります。また、評価関数で評価基準を変数毎に細かく指定することでビット幅を削減できることがあります。

■ 変更履歴

2010/01/13 ver1.00.01 誤字修正
2010/01/11 ver1.00.00 FP-Fixer 解析モデル生成マニュアル新規作成