

EnSight の計算機エディタを利用すると、各種組み込み関数や数学関数を使って、データの変換など、計算を行うことができます。

この組み込み関数をユーザーが追加する機能をサポートしています。ここでは、その関数組み込みについて紹介します。

UDMF : User Defined Math Functions

1. できること

現在サポートしている関数組み込みに関する機能、制限を以下に示します。

- ・データがノード上にある場合には、そのノード数分、ノード毎に計算を行う。
- ・データが要素中心にある場合には、その要素数分、要素毎に計算を行う。
- ・ノードと要素データが混ざっている場合には、ノードデータを要素データに変換され、要素データとして扱い、要素毎に計算を行う。

- ・戻り値は、スカラー、ベクトル、定数の3つを指定できる。
- ・スカラー、ベクトルの場合には、各ノード、もしくは要素毎の計算でそのノード、もしくは要素上のスカラー、ベクトル値を作成する。
- ・定数の場合は、各ノード、要素毎に計算した結果を最終的に足し合わせた値を作成する。

ノード後、要素毎に計算を行うため、ある特定のノードや要素を指定したり、隣り合うノードや要素の値を用いて計算するようなことはできません。

注：なお、Windows 版で行うには、cygwin 環境が必要です。

(詳細は検索エンジン等で cygwin に関する情報を検索してみてください)

2. サンプルと仕組み

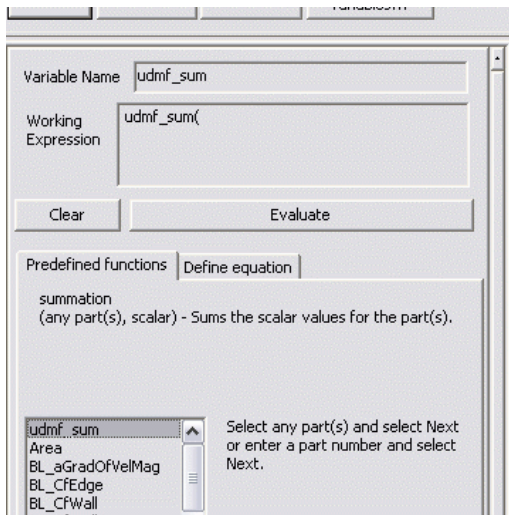
このユーザー定義関数は、UNIX/Linux では shared object(.so)として、Windows では DLL ファイルとして作成します。

以下のフォルダを見てください。

[CEI_HOME]/ensight82/machines/[machine]/lib_udmf/

このフォルダに、libudmf-sum.so (Windows では libudmf-sum.dll) ファイルがあります。

このファイルは、ユーザー定義関数である udmf_sum 関数を登録しています。



計算機アイコンを見ると、組み込み関数一覧に udmf_sum という関数を見ることができます。

この関数は選択されたパートの指定されたスカラーデータを単に足し合わせることができる関数です。

この関数のソースコードの他、いくつかのサンプルソースが以下のフォルダに提供されています。

[CEI_HOME]/ensight82/src/math_functions/

sum ディレクトリの下にある libudmf-sum.c ファイルが、この udmf_sum 関数を定義しているソースコードです。

このソースをコンパイルしたものが libudmf-sum.so (もしくは .dll) です。

3 . ソースコード

実際に libudmf-sum.c ファイルを開いてみてください。

いくつかの関数が定義されています。ソース内のコメントをあわせてご参照ください。

```
int
USERD_get_name_of_mf(char mf_name[UDMFSNAME]) {
    memset(mf_name, '\0', UDMFSNAME);
    strcpy(mf_name, "sum");
    return(Z_OK);
}
```

- EnSight がこの関数を呼び出すための名前を登録します。
このサンプルでは、"sum" という名前で登録しています。

```
int
USERD_get_mf_version(char version_number[UDMFSNAME]) {
    strcpy(version_number, "1.000");
    return(Z_OK);
}
```

- ユーザーが書き換える必要はありません。
内部で利用しているバージョン番号です。

```
int
USERD_get_nargs(int *nArgs) {
    *nArgs = 1;
    return(Z_OK);
}
```

- スカラーデータやベクトルデータなど、関数に与える引数の数を指定します。
この関数は、udmf_sum(part, scalar) のように、スカラーデータ成分をひとつ、パラメーターとして渡します。
もし、ベクトルデータを渡す場合でも、成分をひとつ渡すのであれば (例えば、udmf_sum(part, vector) のように) 1 となります。

```

int
USERD_get_meta_data(char listDescription[UDMFLNAME],
                    char funcDescription[UDMFLNAME],
                    int *argTypes, int *returnType) {

strcpy(listDescription, "udmf_sum(part, scalar)");
strcpy(funcDescription,
        "summation(any part(s), scalar) - "
        "Sums the scalar values for the part(s).");
        注：実際のソースでは機種毎の場合分け ifdef 文があります。
argTypes[0] = UDMFSCL;
*returnType = UDMFCON;
return(Z_OK);
}

```

- listDescription には、関数の定義を記述します。
この関数は、udmf_sum() という関数名で表示され、第 1 引数にパート（必須）を指定します。
次に第 2 引数にスカラーデータ成分を指定します。
- funcDescription は、計算機エディタに表示される説明文です。
- argTypes[0] は、引数のタイプです。
この関数は、先に定義したように引数成分はひとつ持ちます。
その引数成分のタイプはスカラー（UDMFSCL）であることを定義しています。
- *returnType は、戻り値のタイプです。
この例では、コンスタンツ（定数）である UDMFCON を指定しています。

引数や戻り値の定義は、以下の 3 つのタイプで指定します。

UDMFSCL : スカラーデータ成分
UDMFVCT : ベクトルデータ成分
UDMFCON : 定数値

```

int
USERD_evaluate(void *args[], void *value, int *undefined) {
    float *result;
    float *arg1;
    result = value;
    arg1 = args[0];
    *result = *arg1;
    *undefined = 0;
    return(Z_OK);
}

```

- 実際の計算ルーチンは、この関数で定義します。
この関数がノード毎（もしくは要素毎）に呼び出されます。
args[0]で、あるノード、もしくは要素（順番に処理される中の）の値が
取得できます。
この例では、指定したスカラーデータ成分の値が入ります。

*value は戻り値です。

この例では、単に、入力データをそのままコピーしている点に注目してください。
この関数では、戻り値を定数（UDMFCON）に指定しています。
定数を戻り値にした場合、EnSight の組み込み関数は、各ノード（要素）毎に
計算された値を最終的に足し合わせて、定数値を作成します。

この udmf_sum 関数は指定したパートの指定したスカラー成分を足し合わせる
関数ですので、単に値を受け渡すだけで足し算結果を得ることができます。

例えば、このルーチンで 100 倍した値を渡すと、最終結果は、各スカラー値を
100 倍した値を最終的に足し合わせた結果となります。

undefined は、ある条件下でそのノード（要素）の値を計算しない場合などに
利用でき、1 に設定して return することができます。

その他、このソースディレクトリには、ベクトルデータを使った例（example1, 2, 3
など）などもありますので、あわせてご参照ください。

4 . コンパイルとライブラリの作成

ソースのコンパイル方法について、以下に示します。

Windows 版で関数を組み込むには、Cygwin 環境が必要です。
別途 cygwin 環境を Windows にインストールしてから行ってください。

1) 機種毎の定義ファイルの作成

まず、機種毎のコンフィグファイルを作成します。

```
# cd $CEI_HOME/ensight82/src/config3
```

src/config3 ディレクトリに移動します。

* 適宜、src ディレクトリのバックアップ等を取ってから、
作業してください。
また、root でインストールしている場合には、root で実行してください。

* Windows の場合は、コンパイラパスが通った DOS 窓で、
sh を起動し、/cygdrive/c/ などに移動します。

```
# ./init -prod linux_2.4  
$ ./init -prod win32_mt /cygdrive/c/Progra~1 /CEI/ensight82/src
```

* 機種毎の定義ファイルを指定し、init ファイルを実行します。
init ファイルに実行権が立っていない場合は、chmod +x init で
実行権を付加してから実行してください。

Initialization complete

If you have previously built this tree for a different architecture
or with different options it is recommended that you make clean
before continuing

エラーなく、上記のメッセージが出力されれば、次に進みます。
同フォルダに config ファイルが作成されます。

Windows 版の場合、Program Files の空白スペースの関係で、コンパイル時にエラーとなる場合があります。

以下のファイルを開き、Program Files を Progra~1 に変更してください。

```
config3/config
```

```
例：CEIDEVROOT='/cygdrive/c/Progra~1/CEI/ensight82/src'
```

* チルダ (~) で指定しています。

2) sum 関数の定義の変更

実際に関数を組み込む前に、まずは先に述べた sum 関数を変更してみましょう。

例えば、もっとも簡単な、関数の定義文字を適当に修正してみてください。

```
#ifdef WIN32
    strcpy(funcDescription,
           "TEST summation(any part(s), scalar) - "
           "Sums the scalar values for the part(s).");
#else
    strcpy(funcDescription,
           "TEST summation(any part(s), scalar)¥n¥n"
           "Sums the scalar values for the part(s).¥n");
#endif
```

上記例では、単に、TEST という文字列を最初に追加しています。

3) コンパイル

コンパイルは、同フォルダにある Makefile を利用して行います。

```
# mkdir ../lib      (Linux/UNIX の場合、ひとつ上に lib ディレクトリを
                    作成してから行ってください)

# make
libudmf-sum.c
Loading libudmf-sum.so ...
gcc -shared -L'/usr/local/CEI/ensight82/src'/lib -L/home/dev/arch/linux_2.4/lib
-o libudmf-sum.so libudmf-sum.o
-lm
```

```
cp libudmf-sum.so ../lib
```

コンパイルが終了すると、.so ファイル、もしくは .dll ファイルが、ひとつ上の lib ディレクトリに作成されます。

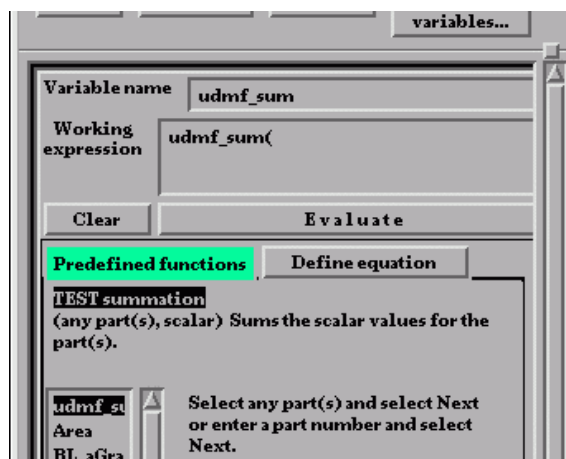
3) ライブラリの配置と EnSight の実行

先に述べたように、この sum 関数は、もともとユーザー定義関数として、以下のフォルダに登録されています。

```
$CEI_HOME/ensight82/machines/linux_2.4/lib_udmf/libudmf-sum.so
```

このファイルの名前を変更（例えば、libudmf-sum.org）し、作成した新しいライブラリファイルをこのフォルダにコピーします。

EnSight を起動し、計算機エディタを開いてみてください。



図に示すように関数の説明が書き換わっていることを見ることができるはずです。

以上