XcodeML/C 仕様書

Version 0.91J (November 14, 2011)

XcalableMP/Omni Compiler Project

改版履歴

Version 0.91J

- 配列要素の参照の要素を変更。
- subArrayRef 要素を変更。
- indexRange 要素を追加。

目次

1 はじめに 1

2 XcodeProgram 要素 1

2.1 name 要素 1
2.2 value 要素 2

3 typeTable 要素 2

3.1 データ型識別名 2
3.2 basicType 要素 3
3.3 pointerType 要素 4
3.4 functionType 要素 4
3.5 arrayType 要素 4
3.6 structType 要素、unionType 要素 5
3.7 enumType 要素 6
3.8 データ型定義要素のオプション属性 7

4 シンボルリスト 7

4.1 id 要素 7
4.2 globalSymbols 要素 8
4.3 symbols 要素 8

5 globalDeclarations 要素 8

5.1 functionDefinition 要素 9
5.2 params 要素 9
5.3 varDecl 要素 9
5.4 functionDecl 要素 10

6 文の要素 10

6.1 exprStatement 要素 10
6.2 compoundStatement 要素 10
6.3 ifStatement 要素 10
6.4 whileStatment 要素 10
6.5 doStatement 要素 11
6.6 forStatement 要素 11
6.7 breakStatement 要素 11
6.8 continueStatement 要素 11
6.9 returnStatment 要素 11
6.10 gotoStatement 要素 11
6.11 statementLabel 要素 11
6.12 switchStatement 要素 12
6.13 caseLabel 要素 12
6.14 gccRangedCaseLabel 要素 12
6.15 defaultLabel 要素 12
6.16 pragma 要素 12
6.17 text 要素 12
6.18 行番号属性 12

7 文の要素 12

7.1 定数の要素 13
7.2 変数参照の要素 13

7.3 pointerRef 要素 14
7.4 配列要素の参照の要素 14
7.5 構造体メンバーの参照の要素 14
7.6 assignExpr 要素 14
7.7 2 項演算式の要素 15
7.8 単項演算式の要素 16
7.9 functionCall 要素 16
7.10 commaExpr 要素 16
7.11 postIncrExpr 要素、postDecrExpr 要素、preIncrExpr 要素、preDecrExpr 要素 16
7.12 castExpr 要素 16
7.13 condExpr 要素 17
7.14 gccCompoundExpr 要素 17
8 XcalableMP の要素 17
8.1 coArrayType 要素 17
8.2 coArrayRef 要素 18
8.3 subArrayRef 要素 18
8.4 indexRange 要素 18
9 その他の要素・属性 19
9.1 typeName 要素 19
9.2 is_gccExtension 属性 19
9.3 gccAsm 要素、gccAsmDefinition 要素、gccAsmStatement 要素 19
9.4 gccAttributes 要素 21
9.5 builtin_op 要素 23
9.6 is_gccSyntax 属性 23
9.7 is_modified 属性 24
10 コード例 24

1　はじめに

C 言語の Xcode は、C 言語プログラムに復元可能な中間コードである。この仕様書は Xcode の XML 表現を記述する。

この中間コードレベルでは、以下の特徴を持つ。

- C 言語に復元可能な情報を保持する。
- C 言語相当の型情報を表現できる。
- 様々な変形に必要な構文要素を持つ。
- human-readable なフォーマット(XML)

XcodeML ファイル へは、C プログラムに対して、C-front により変換する。XcodeML ファイル から、C プログラムに対して、デコンパイラにより変換する。解析プログラムにより、Xcode に対する解析、プログラムを作ることができる。

2　XcodeProgram 要素

Xcode によって表現されるプログラムは、Type table と global Id table, 外部定義の列からなる。Xocde ファイルのトップレベルの要素は、XcodeProgram 要素である。XcodeProgram 要素は以下の要素を含む。

- typeTable 要素　– プログラムで利用されているデータ型の情報
- globalSybmols 要素 – プログラムで利用されている大域変数の情報
- globalDeclrations 要素 – 関数、変数宣言などの情報

要素は、属性として以下の情報を持つことができる

- compiler-info　－　CtoC コンパイラの情報
- version　－　CtoC コンパイラのバージョン情報
- time　－　コンパイルされた日時
- language　－　ソース言語情報
- source　－　ソース情報

2.1　name 要素

変数名や、タイプ名などの名前を指定する要素である。名前を文字列として持つ。 属性として、type 属性を持つ。属性値はタイプ識別名である。

2.2　value 要素

初期値を表現する。次の子要素を持つ。

- 式(0-複数)　－　値を示す式。
- value　－　ネストされた値。"{ ... }" に対応する。

例:
int 型の初期値 1 に対応する表現は次のとおりになる。

```
<value>
  <intConstant type="int">1</intConstant>
</value>
```

int 型配列の初期値 { 1, 2 } に対応する表現は次のとおりになる。

```
<value>
  <value>
    <intConstant type="int">1</intConstant>
    <intConstant type="int">2</intConstant>
  </value>
</value>
```

3　typeTable 要素

typeTable 要素は、このファイル全体で使われているデータ型についての情報を定義する。typeTable 要素は、データ型を定義するデータ型定義要素の列からなる。 データ型定義要素には以下の要素がある。

- pointerType 要素
- functionType 要素
- arrayType 要素
- structType 要素
- unitionType 要素
- enumType 要素
- basicType 要素

3.1　データ型識別名

プログラム内において、データ型はデータ名で区別される。その名前は、次の２つのいずれかであ

る。

- 基本データ型名

C 言語の基本データ型

'void', 'char', 'short', 'int' , 'long', 'long_long', 'unsigned_char', 'unsigned_short', 'unsigned', 'unsigned_long', 'unsigned_long_long', 'float', 'double', 'long_double', 'wchar_t', 'bool' (_Bool 型)

_Complex、_Imaginary に対応する型

'float_complex', 'double_complex', 'long_double_complex', 'float_imaginary', 'double_imaginary', 'long_double_imaginary'

GCC の組み込み型

'__builtin_va_arg'

- 派生データ型名 – 上記、基本データ型名以外の任意の英数字の並び。

派生データ型の名前は、プログラム内でユニークなものでなくてはならない。

3.2 basicType 要素

basicType 要素は C,C99 の基本データ型を定義する。 属性として以下のものをもつ。

- type
- name

例:

```
struct {int x; int y;} s;
struct s const * volatile p;
```

は次の XcodeML に変換される。

```
  <structType type="S0">
    ...
  </structType>
  <basicType type="B0" is_const="1" name="S0"/>
  <pointerType type="P0" is_volatile="1" ref="B0"/>
```

3.3 pointerType 要素

pointerType 要素はポインターのデータ型を定義する。以下の属性を持つ。

- type　－　このポインター型の派生データ型の名前。
- ref　－　このポインタ・データ型が参照するデータ型の名前

pointType 要素は、他の要素を持たない。

例:

"int *" に対応するデータ型定義は以下のようになる。

```
<pointerType type="P0123" ref="int" />
```

3.4 functionType 要素

funtionType 要素は、関数データ型を定義する。

- type　－　この関数型の派生データ型の名前
- return_type　－　この関数型が返すデータ型の名前
- is_inline　－　この関数型が inline 型であるかどうかの情報、0 または 1、false または true

プロトタイプ宣言がある場合には、引数要素に対応する param 要素を含む。

例:

"double foo(int a,int b)" の foo に対するデータ型は以下のようになる。

```
<functionType type="F0457" return_type="double">
<params>
    <name type="int">a</name>
    <name type="int">b</name>
  </params>
</fucntionType>
```

3.5 arrayType 要素

arrayType 要素は、配列データ型を定義する。arrayType 要素は以下の属性を持つ。

- type　－　この配列型の派生データ型の名前
- element_type　－　配列要素のデータ型の識別子を指定する
- array_size　－　配列のサイズ（要素数）。array_size と子要素の arraySize を省略した場合は、サイズ未指定を意味する。array_size 属性は子要素の arraySize と同時に指定することはできない。
- is_const, is_volatile, is_restrict, is_static　－　これらの属性はそれぞれ配列サイズの const, volatile, restrict, static 修飾子を指定するかどうかの情報。値は 0 または 1、false または true。

以下の要素を持つ。

- arraySize　－　配列のサイズ（要素数）を表す式。式要素ひとつを子要素に持つ。
  サイズを数値で表現できない場合や、可変長配列の場合に指定する。arrayType 要素が arraySize 要素を持つ場合、array_size 属性の値は"*"とする。

例:

"int a[10]"の a に対する type_entry は以下のようになる。

```
<arrayType type="A011" element_type="int" array_size="10"/>
```

3.6　structType 要素、unionType 要素

struct(構造体)データ型は、structType 要素で定義する。structType 要素は以下の属性を持つ。

- type　－　この配列型の派生データ型の名前

Union(共用体)データ型は、unionType 要素で定義する。untionType 要素は、structType 要素と同じ、属性、要素を持つ。

StructType・unionType 要素は、メンバーに対する識別子の情報である symbols 要素を持つ。 構造体・共用体のタグ名がある場合には、スコープに対応するシンボルテーブルに定義されている。
メンバーのビットフィールドは、id 要素の bit_field 属性または id 要素の子要素 bitField タグに記述する。bitField タグは式を子要素に持ち、bitField タグを持つ id 要素の bit_field 属性の値は、"*"とする。

例:
"struct {int x; int y : 8; int z : sizeof(int); } S;"の S に対する structType 要素は以下のようになる。

```
<stuctType type="S6e89">
  <symbols>
```

```
        <id type="int">
          <name>x</name>
        </id>
        <id type="int" bit_field="8">
          <name>y</name>
        </id>
        <id type="int" bit_field="*">
          <name>z</name>
          <bitField>
            <sizeOfExpr>
              <typeName ref="int"/>
            </sizeOfExpr>
          </bitField>
        </id>
      </symbols>
    </structType>
```

3.7 enumType 要素

enum 型は、enumType 要素で定義する。type 要素で、メンバの識別子を指定する。
次の子要素を持つ。

- symbols － メンバの識別子を定義する。メンバの初期値は id － value 要素で表す。

メンバの識別子は、スコープに対応するシンボルテーブルにクラス moe として定義されている。enum のタグ名がある場合には、スコープに対応するシンボルテーブルに定義されている。

例:
"enum { e1, e2, e3 = 10 } ee; "の ee に対する enumType 要素は以下のようになる。

```
    <enumType name="E0">
      <symbols>
        <id>
          <name>e1</name>
        </id>
        <id>
          <name>e2</name>
        </id>
        <id>
```

```
            <name>e3</name>
            <value><intConstant>10</intConstant></value>
          </id>
        </symbols>
      </enumType>
```

3.8　データ型定義要素のオプション属性

データ型定義要素の属性として以下の属性を持つ（省略可)。

- is_const　－　このデータ型で定義される変数が定数であるかどうかの情報、 0 または 1、false または true
- is_volatile　－　このデータ型で定義される変数が volatile であるかどうかの情報、 0 または 1、false または true
- is_restrict　－　このデータ型で定義される変数が restrict であるかどうかの情報、0 または 1、false または true

4　シンボルリスト

4.1　id 要素

id 要素は、変数名や配列名、関数名、struct/union のメンバー名、 関数の引数、compound statement の局所変数名を定義する。id 要素は次の属性を持つ。

- sclass　－　storage class をあらわし、 'auto', 'param', 'extern', 'extern_def', 'static', 'register', 'label', 'tagname', 'moe', 'typedef_name' のいずれか。
- type　－　識別子のデータ型
- bit_field 属性　－　structType・unionType においてメンバーのビットフィールドを指定する。
- is_gccThread　－　GCC の__thread キーワードが指定されているかどうかの情報、0 または 1、false または true。
- is_gccExtension 属性

以下の要素を持つ。

- name 要素　－　識別子の名前は name 要素で指定する。
- value 要素　－　識別子に対応した値は value 要素で指定する。
- bitField 要素　－　structType・unionType においてメンバーのビットフィールドを指定する。

識別子が変数などの場合、そのアドレスを要素として持つ。コンパイラで生成される変数などの場合は、なくてもよい。

例:

"int xyz;"の変数 xyz に対するシンボルテーブルエントリは以下のようになる。なお、P6e7e0 は "int *"に対する type_id。

```
<id sclass="extern_def" type="int">
 <name>xyz</name>
 <value>
   <VarAddr type="P6e7e0">xyz</varAddr>
 </value>
</id>
```

"int foo()"の関数 foo に対するシンボルテーブルエントリは以下のようになる。なお、F6f168 は、foo のデータ型に対する type_id。P6f1a8 は、F6f168 へのポインタの type_id。識別子 foo は関数へのポインタになることに注意。

```
<id sclass="extern_def" type="0x6f168">
 <name>foo</name>
 <value>
  <funcAddr type="0xfla8">foo</funcAddr>
 </value>
</id>
```

4.2 globalSymbols 要素

大域のスコープを持つ識別子を定義する。要素として、大域のスコープを持つ識別子の id 要素を持つ。

4.3 symbols 要素

symbols 要素は、局所スコープを持つ識別子を定義する。要素として、定義する識別子に対する id 要素を持つ。

5 globalDeclarations 要素

globalDeclarations 要素は、プログラム中の大域変数の宣言や関数定義をするための要素である。以下の要素を持つ。

- functionDefinition 要素 － 関数の定義
- varDecl 要素 － 変数の定義
- functionDecl 要素 － 関数の宣言
- text 要素 － ディレクティブなど任意のテキストを表す

5.1 functionDefinition 要素

functionDefinition 要素は、関数定義を行う。
以下の要素を持つ

- name 要素　－　関数名を指定する
- symbols 要素　－　パラメータのシンボルリスト
  symbols 要素において、パラメータに対するシンボルテーブルを指定する。
- params 要素　－　パラメータの定義
- body 要素　－　関数本体, body 要素は要素として文または functionDefinition を含む。body 要素内の functionDefinition は、GCC のネストされた関数を表す。

以下の属性を持つ

- is_gccExtension 属性

5.2 params 要素

params 要素は、関数の引数の並びを指定する。

- name 要素　－　引数に対応する name 要素を持つ。
- ellipsis　－　可変長引数を表す。params の最後の子要素に指定可能。
- params 要素内の name 要素は、引数の順序で並んでいなくてはならない。 引数のデータ型の情報がある場合には、name 要素の type 属性に指定する。

5.3 varDecl 要素

varDecl 要素は、変数の宣言を行う。変数宣言を行う識別子の名前を name 要素で指定する。 以下の要素を持つ。

- name 要素　－　宣言する変数に対する name 要素を持つ。
- value 要素　－　初期値を持つ場合、この要素で指定する。配列・構造体の初期値の場合、value 要素に複数の式を指定する。

例:

```
int a[] = { 1, 2 };
<varDecl>
  <name>a</name>
  <value>
```

```
        <intConstant type="int">1</intConstant>
        <intConstant type="int">2</intConstant>
      </value>
    </varDecl>
```

5.4 functionDecl 要素

functionDecl 要素は、関数宣言を行う。
以下の要素を持つ

- name 要素　－　関数名を指定する

6 文の要素

C の文の構文要素に対応する XML 要素である。それぞれの要素には、行番号が属性として付加されており、文が現れた行番号やファイルの情報を取り出すことができる。

6.1 exprStatement 要素

exprStatment 要素で、式で表現される文を表す。式の要素を持つ。

6.2 compoundStatement 要素

compoundStatement 要素で、複文を表現する。以下の要素を持つ。

- symbols 要素　－　この複文の中で定義されているシンボルリスト
- declarations 要素　－　この複文の中で定義する宣言に対する varDecl 要素、functionDefinition 要素、および functionDecl 要素を含む
- body 要素　－　複文本体を含む

6.3 ifStatement 要素

if 文をあらわす要素。以下の要素を持つ。

- condition 要素　－　条件式を要素として含む
- then 要素　－　then 部の文を要素として含む
- else 要素　－　else 部の文を要素として含む

6.4 whileStatment 要素

while 文を表す要素。以下の要素を持つ

- condition 要素　－　条件式を要素として含む

- body 要素　－　本体の文を要素として含む

6.5　doStatement 要素

do 文を表す要素。以下の要素を持つ。

- body 要素　－　本体の文を要素として含む
- condition 要素　－　条件式を要素として含む

6.6　forStatement 要素

for 文を表す要素。以下の要素を持つ。

- init 要素　－　初期化式を要素として含む
- condition 要素　－　条件式を含む
- iter 要素　－　繰り返し式を含む
- body 要素　－　for 文の本体

6.7　breakStatement 要素

break 文を表す要素。空要素である。

6.8　continueStatement 要素

continue 文を表す要素。空要素である。

6.9　returnStatment 要素

return を表す要素。return する式を、要素として持つ。

6.10　gotoStatement 要素

goto 文を表す要素。子要素に name 要素か式のいずれかを持つ。式は GCC においてジャンプ先として指定可能なアドレスの式を表す。

- name 要素　－　ラベル名の名前を指定する。
- 式　－　ジャンプ先のアドレス値を指定する。

6.11　statementLabel 要素

goto 文のターゲットのラベルを表す。ラベル名を name 要素として持つ。

- name 要素　－　ラベル名の名前を指定する。

6.12　switchStatement 要素

switch 文を表す要素。以下の要素を持つ。

- value 要素　－　switch する値を指定する。
- body 要素　－　switch 文の本体を指定する。

6.13　caseLabel 要素

switch 文の case 文を表す要素。case の値を要素としてもつ。

- value 要素　－　case の値を指定する。

6.14　gccRangedCaseLabel 要素

gcc 拡張の case 文での範囲指定を表す要素。 case の値を要素としてもつ。

- value 要素　－　case の値の下限値を指定する。
- value 要素　－　case の値の上限値を指定する。

6.15　defaultLabel 要素

switch 文の default ラベルを表す。

6.16　pragma 要素

pragma 要素は#pragma 文を表す。内容に#pragma に指定する文字列を持つ。

6.17　text 要素

text 要素は任意のテキストを表し、コンパイラに依存したディレクティブなどの情報を要素として持つために使用する。内容に任意の文字列を持つ。この要素は globalDeclarasions にも出現する。

6.18　行番号属性

すべての文を表す要素は、文の行番号を表す以下の属性を持つことができる。

- lineno　－　文番号を値として持つ
- file　－　この文が含まれているファイル名

7　文の要素

C の式の構文要素に対応する XML 要素である。それぞれの要素には、データ型が type 属性として付加されており、式のデータ型情報を取り出すことができる。

7.1　定数の要素

定数は以下の要素によって表現する。

- intConstant 要素　－　整数の値を持つ定数を表す。内容として、十進数もしくは、16 進数（0x から始まる）を記述する。
- longlongContant 要素　－　32 ビット 16 進数(0x から始まる)の２つの数字を記述する。
- floatConstant 要素　－　float・double・long double の値を持つ定数を表す。浮動小数点数のリテラルを記述する。
- stringConstant 要素　－　内容に文字列を記述する。属性に is_wide="[1|0|true|false]" (省略時 0）を持ち、1 または true のとき wchar_t 型の文字列を表す。
- moeConstant 要素　－　enum 型の定数を表す。内容に enum 定数を記述する。
- funcAddr 要素　－　関数のアドレスを表す。内容に関数名を記述する。

定数のデータ型は、type 属性で指定する。

moeConstant の場合は、この式を含むスコープのシンボルテーブルの中に、指定された moe 定数がふくまれていなくてはならない。

7.2　変数参照の要素

変数への参照は、大域変数、パラメータ変数、局所変数、それぞれに対して、異なる要素を用いる。

- Var 要素　－　大域変数を参照する式。内容に変数名を指定する。
- varAddr 要素　－　大域変数をアドレスを参照する式。内容に変数名を指定する。
- scope 属性をつかって、局所変数を区別する。
- scope 属性　－　"local", "global", "param"のいずれか

```
<Var>var_name</Var>
```

は、

```
<PointerRef> <varAddr>var_name</varAddr></PointerRef>
```

と等価。

```
<varAddr>var_name</varAddr>
```

は,C での表現は&var_name

7.3 pointerRef 要素

pointerRef 要素は、sub 要素の式をアドレスとし、メモリ参照を行う。

7.4 配列要素の参照の要素

配列への参照には、以下の要素を用いる。

- arrayRef － 配列要素を参照する式。内容に先頭アドレスと添字を指定する。
- arrayAddr － 配列の先頭アドレスを参照する式。内容に配列名を指定する。

配列要素への参照は、arrayAddr を用いてアドレスを取得し、arrayRef で要素にアクセスする。変数参照の要素と同様に、scope 属性を使い局所変数を区別する。

7.5 構造体メンバーの参照の要素

構造体メンバへの参照は､参照アドレスを参照する memberAddr とメンバを参照する memberRef､メンバ配列のアドレスを参照する memberArrayAddr がある。

- memberAddr － 構造体メンバーのアドレスを参照する。内容に構造体のアドレスの式を指定し、member 属性値にメンバー名を指定する。
- memberRef － 構造体メンバーを参照する。内容に構造体のアドレスの式を指定し、member 属性値にメンバー名を指定する。
- memberArrayAddr － 構造体の配列メンバーのアドレスを参照する｡内容に構造体のアドレスの式を指定し、member 属性値にメンバー名を指定する。
- memberArrayRef － 構造体の配列メンバーを参照する。内容に構造体のアドレスの式を指定し、member 属性値にメンバー名を指定する。

```
<memberRef memer="xxx">addr</memberRef>
```

は

```
<pointerRef >
  <memberAddr memer="xxx">addr<memberAddr>
</pointerRef>
```

と等価。

7.6 assignExpr 要素

assignExpr 要素は、２つの式の要素を sub 要素に持ち、代入を表す。 AssginExpr 要素の左式（初

めの要素）は、lvalue でなくてはならない。

7.7　2 項演算式の要素

以下の要素は算術 2 項演算式を表す。被演算子の 2 つの要素を内容に指定する。左式が第一要素、右式が第 2 要素である。

- plusExpr　－　加算
- minusExpr　－　減算
- mulExpr　－　乗算
- divExpr　－　除算
- modExpr　－　剰余
- LshiftExpr　－　左シフト
- RshiftExpr　－　右シフト
- bitAndExpr　－　ビット論理積
- bitOrExpr　－　ビット論理和
- bitXorExpr　－　ビット論理　排他和

これに加えて、代入式とあわせた以下の要素がある。

- asgPlusExpr　－　加算
- asgMinusExpr　－　減算
- asgMulExpr　－　乗算
- asgDivExpr　－　除算
- asgModExpr　－　剰余
- asgLshiftExpr　－　左シフト
- asgRshiftExpr　－　右シフト
- asgBitAndExpr　－　ビット論理積
- asgBitOrExpr　－　ビット論理和
- asgBitXorExpr　－　ビット論理　排他和

以上の代入つき演算式の左式（初めの要素）は、lvalue でなくてはならない。
以下は論理 2 項演算式を表す要素である。被演算子の 2 つの要素を内容に指定する。

- ogEQExpr　－　等価
- logNEQExpr　－　非等価
- logGEExpr　－　大なり、または同値
- logGTExpr　－　大なり
- logLEExpr　－　小なり、または等価
- logLTExpr　－　小なり

- logAndExpr　－　論理積
- logOrExpr　－　論理和

7.8　単項演算式の要素

以下の要素は算術単項演算式を表す。被演算子の式の要素を内容に指定する。

- unaryMinusExpr　－　符号反転
- bitNotExpr　－　ビット反転

以下の要素は、論理単項演算式を表す。被演算子の式の要素を内容に指定する。

- logNotExpr　－　論理否定

以下の要素は、sizeof 演算子と GCC 拡張の演算子を表す。

- sizeOfExpr　－　sizeof 演算子。子要素に式または typeName 要素を指定する。
- gccAlignOfExpr　－　GCC の__alignof__演算子を表す。子要素に式または typeName 要素を指定する。
- gccLabelAddr　－　GCC の&&単項演算子を表す。内容にラベル名を指定する。

7.9　functionCall 要素

functionCall 要素は関数呼び出しを表す。以下の２つの要素を持つ。

- function 要素　－　呼び出す関数のアドレスを指定する。
- arguments 要素　－　引数の式を指定する。

7.10　commaExpr 要素

コンマ式（順番に評価し、最後の要素の式を返す式）は、commaExpr 要素で表す。

7.11　postIncrExpr 要素、postDecrExpr 要素、preIncrExpr 要素、preDecrExpr 要素

postIncrExpr 要素、postDecrExpr 要素は、C 言語のポストインクリメント、デクリメント式を表す。内容は、lvalue でなくてはならない。 preIncrExpr 要素、preDecrExpr 要素は、C 言語のプレインクリメント、デクリメント式を表す。内容は、lvalue でなくてはならない。

7.12　castExpr 要素

castExpr 要素は型変換の式、または複合リテラルを表す。

以下の属性を持つ

- type 属性　－　変換後の式のタイプを指定する。
- is_gccExtension 属性

以下の子要素を持つ。

- value　－　複合リテラルのリテラル部を表す。

7.13　condExpr 要素

condExpr 要素は、""演算子に対応する要素である。

次の例の様に子要素として 3 つの式の要素を持つ。

```
<condExpr>
  (式)
  (式)
  (式)
</condExpr>
```

7.14　gccCompoundExpr 要素

gcc 拡張の複文式に対応する要素。
compoundStatement 要素を要素としてもつ。

- compoundStatement 要素　－　複文式の内容を指定する。

8　XcalableMP の要素

8.1　coArrayType 要素

"#pragma xmp coarray" によって宣言された、Co-Array 型を表す。 次の属性を持つ。

- type　－　派生データ型名。
- element_type　－　Co-Array の要素のデータ型名。データ型名に対応する型が coArrayType のときは、2 次元以上の Co-Array 型を表す。
- array_size　－　Co-Array 次元を表す。

次の子要素を持つ。

- arraySize　－　Co-Array 次元を表す。arraySize 要素を持つときの array_size 属性の値は "*" とする。

例:

```
int A[10];
#pragma xmp coarray [*][2]::A
```

上記の変数 A の型を表す要素は、次の coArrayType "C2"になる。

```
<arrayType type="A1" element_type="int" array_size="10"/>
<coArrayType type="C1" element_type="A1"/>
<coArrayType type="C2" element_type="C1" array_size="2"/>
```

8.2　coArrayRef 要素

Co-Array 型の変数への参照を表す。
次の子要素を持つ。

- 1 番目の式　－　Co-Array 変数を表す式。
- 2 番目以降の式　－　Co-Array 次元を表す式。複数の次元を持つ場合は、複数の式を指定する。

8.3　subArrayRef 要素

部分配列の参照を表す。
次の子要素を持つ。子要素を省略することはできない。

- 第一の要素として配列を表す式をもつ。
- 2 番目以降の式　－　添字または添字 3 つ組を表す式。複数の次元を持つ場合は、複数の式を指定する。

8.4　indexRange 要素

3 つ組(triplet)を表す。
次の子要素を持つ。子要素を省略することはできない。

- lowerBound　－　下限のインデックスを表す。子要素に式を持つ。
- upperBound　－　上限のインデックスを表す。子要素に式を持つ。
- step　－　インデックスの刻み幅を表す。子要素に式を持つ。

9　その他の要素・属性

9.1　typeName 要素

typeName 要素は型名を表す。sizeOfExpr、gccAlignOfExpr、builtin_op などの子要素として指定される。属性にデータ型識別名を示す type を持つ。

9.2　is_gccExtension 属性

is_gccExtension 属性は、GCC の __extension__ キーワードを要素の先頭に付加するかどうかを定義し、値は 0 または 1 (false または true) である。is_gccExtension 属性は省略可能で、指定しないときは値 0 を指定したときと同じ意味である。次の要素に is_gccExtension 属性を持つことができる。

- id
- functionDefinition
- castExpr
- gccAsmDefinition

例:

"__extension__ typedef long long int64_t" に対応する定義は次のようになる。

```
<id type="long_long" sclass="typedef" is_gccExtension="1">
  <name>int64_t</name>
</id>
```

9.3　gccAsm 要素、gccAsmDefinition 要素、gccAsmStatement 要素

gccAsm 要素・gccAsmDefinition 要素・gccAsmStatement 要素は、GCC の asm/__asm__ キーワードを定義する。要素として asm の引数の文字列を持つ。

- gccAsm　－　asm 式を表す。次の子要素を持つ。
- stringConstant (1 個)　－　アセンブラコードを表す。
- gccAsmDefinition　－　asm 定義を表す。子要素は gccAsm と同じ。
- gccAsmStatement　－　asm 文を表す。

次の属性を持つ。

- is_volatile　－　volatile が指定されているかどうかの情報、0 または 1、false または true。

次の子要素を持つ。

- stringConstant (1 個) － アセンブラコードを表す。
- gccAsmOperands (2 個) － 1 番目が出力オペランド、2 番目が入力オペランドを表す。オペランドを省略する場合は、子要素を持たないタグを記述する。子要素に gccAsmOperand(複数)を持つ。
- gccAsmClobbers (0-1 個) － クロバーを表す。子要素に 0 個以上の stringConstant を持つ。
- gccAsmOperand － 入出力オペランドを表す。

次の属性を持つ。

- match (省略可) － matching constraint の代わりに指定する識別子を表す ("[識別子]" に対応)。
- constraint (省略不可) － constraint/constraint modifier を表す。

次の子要素を持つ。

- 式 (1 個) － 入力または出力に指定する式を表す。

例:

```
asm volatile (
        "661:¥n"
        "¥tmovl %0, %1¥n662:¥n"
        ".section .altinstructions,¥"a¥"¥n"
        ".byte %c[feat]¥n"
        ".previous¥n"
        ".section .altinstr_replacement,¥"ax¥"¥n"
        "663:¥n"
        "¥txchgl %0, %1¥n"
        : "=r" (v), "=m" (*addr)
        : [feat] "i" (115), "0" (v), "m" (*addr));
```

```
<gccAsmStatement is_volatile="1">
  <stringConstant><![CDATA[661:¥n¥tmovl .. (省略) ..]]></stringConstant>
  <gccAsmOperands>
    <gccAsmOperand constraint="=r">
```

```
        <Var>v</Var>
      </gccAsmOperand>
      <gccAsmOperand constraint="=m">
        <pointerRef><Var>addr</Var></pointerRef>
      </gccAsmOperand>
    </gccAsmOperands>
    <gccAsmOperands>
      <gccAsmOperand match="feat" constraint="i">
        <intConstant>115</intConstant>
      </gccAsmOperand>
      <gccAsmOperand constraint="m">
        <pointerRef><Var>addr</Var></pointerRef>
      </gccAsmOperand>
    </gccAsmOperands>
  </gccAsmStatement>
```

9.4 gccAttributes 要素

gccAttributes 要素は GCC の __attribute__ キーワードを定義する。要素として、__attribute__ の引数の文字列を持つ。gccAttributes 要素は、gccAttribute 要素を子要素に複数持つ。

- 型を表す要素全てが gccAttributes 要素を子要素に持つ（0～1 個）。
- id 要素が gccAttributes 要素を子要素に持つ（0～1 個）。
- functionDefinition 要素が gccAttributes 要素を子要素に持つ（0～1 個）。

例:
型を表す要素の子要素に、gccAttributes を設定する例

```
typedef __attribute__((aligned(8))) int ia8_t;
ia8_t __attribute__((aligned(16)) n;
```

```
 <typeTable>
    <basicType type="B0" name="int" align="8" size="4"/>
      <gccAttributes>
        <attribute>aligned(8)</attribute>
     </gccAttributes>
    </basicType>
    <basicType type="B1" name="int" align="16" size="4"/>
     <gccAttributes>
```

```
            <attribute>aligned(8)</attribute>
            <attribute>aligned(16)</attribute>
          </gccAttributes>
    </basicType>
  </typeTable>
 <globalSymbols>
    <id type="B0" sclass="typedef_name">
        <name>ia8_t</name>
     </id>
    <id type="B1">
        <name>n</name>
     </id>
  </globalSymbols>
 <globalDeclarations>
     <varDecl>
        <name>n</name>
     </varDecl>
  </globalDeclarations>
```

id 要素、functionDefinition 要素の子要素に、gccAttributes を設定する例

```
 void func(void);
 void func2(void) __attribute__(alias("func"));


  void __attribute__((noreturn)) func() {
      ...
  }
```

```
 <typeTable>
    <functionType type="F0">
      <params>
          <name type="void"/>
      </params>
    </functionType>
    <functionType type="F1">
      <params>
          <name type="void"/>
```

```
        </params>
      </functionType>
    </typeTable>
    <globalSymbols>
      <id type="F0" sclass="extern_def">
        <name>func</name>
      </id>
      <id type="F1" sclass="extern_def">
        <name>func2</name>
        <gccgccAttributes>
          <gccAttribute>alias("func")</gccAttribute>
        </gccgccAttributes>
      </id>
    </globalSymbols>
    <globalDeclarations>
      <functionDefinition>
        <name>func</name>
        <gccgccAttributes>
          <gccAttribute>noreturn</gccAttribute>
        </gccgccAttributes>
        <body>...</body>
      </functionDefinition>
    </globalDeclarations>
```

9.5 builtin_op 要素

builtin_op 要素はコンパイラ組み込みの関数呼び出しを表す。以下の要素をそれぞれ 0～複数持つ。子要素の順番は関数引数の順番と一致していなければならない。

- 式 － 呼び出す関数の引数として、式を指定する。
- typeName － 呼び出す関数の引数として、型名を指定する。
- gccMemberDesignator － 呼び出す関数の引数として、構造体・共用体のメンバ指示子を指定する。属性に構造体・共用体の派生データ型名を示す ref、メンバ指示子の文字列を示す member を持つ。子要素に配列インデックスを表す式(0-1 個)と、gccMemberDesignator 要素(0-1 個)を持つ。

9.6 is_gccSyntax 属性

is_gccSyntax 属性はそのタグに対応する式、文、宣言が gcc 拡張を使用しているかどうかを定義する。 値として 0 または 1 (false または true) を持つ。この属性は省略可能であり、省略された場合は

値に 0 を指定した時と同じ意味になる。

9.7 is_modified 属性

is_modified 属性はそのタグに対応する式、文、宣言がコンパイルの過程で変形されたかどうかを定義する。値として 0 または 1 (false または true) を持つ。この属性は省略可能であり、省略された場合は値に 0 を指定した時と同じ意味になる。

次の要素に is_gccSyntax 属性、is_modified 属性を持つことができる。

- varDecl
- 文の要素
- 式の要素

10 コード例

例 1:

```
int a[10];
int xyz;
struct {      int x;     int y;} S;
foo() {
      int *p;
      p =   &xyz;            /* 文 1 */
      a[4] = S.y;            /* 文 2 */
}
```

文 1:

```
<exprStatement>
  <assignExpr type=" P6fc98">
    <ponterRef type=" P6fc98">
      <varAddr scope="local" type="P70768">p</varAddr>
    </pointerRef>
    <varAddr type=" P70828">xyz</varAddr>
  </assignExpr>
</exprStatement>
```

もしくは、

```
<exprStatement>
  <assignExpr type=" P6fc98">
   <Var scope="local" type=" P6fc98">p</Var>
     <varAddr type=" P70828">xyz</varAddr>
 </assignExpr>
</exprStatement>
```

文 2:

```
<exprStatement>
 <assignExpr type="int">
    <pointerRef type="int">
      <plusExpr type=" P6fc98">
        <arrayAddr type=" P708e8">a</arrayAddr>
        <intConstant type="int">4</intConstant>
      </plusExpr>
   </pointerRef>
   <pointerRef type="int">
      <memberAddr type="P0dede" member="y">
         <varAddr type= "P70988">S</varAddr>
      </memberAddr>
  </pointerRef>
 </assignExpr>
</exprStatement>
```

もしくは、

```
<exprStatement>
 <assignExpr type="int">
    <pointerRef type="int">
      <plusExpr type=" P6fc98">
        <arrayAddr type=" P708e8">a</arrayAddr>
        <intConstant type="int">4</intConstant>
      </plusExpr>
   </pointerRef>
   <memberRef type="int" member="y">
      <varAddr type= "P70988">S</varAddr>
  </memberRef>
```

```
    </assignExpr>
  </exprStatement>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<XcodeProgram source="t3.c">
  <!--
    typedef struct complex {
       double real;
       double img;
    } complex_t;

    complex_t x;
    complex_t complex_add(complex_t x, double y);

    main()
    {
       complex_t z;

       x.real = 1.0;
       x.img = 2.0;

       z = complex_add(x,1.0);

       printf("z=(%f,%f)¥n",z.real,z.img);

    }
    complex_t complex_add(complex_t x, double y)
    {
       x.real += y;
       return x;
    }
  -->
  <typeTable>
    <pointerType type="P0" ref="S0"/>
    <pointerType type="P1" ref="S0"/>
    <pointerType type="P2" ref="S0"/>
    <pointerType type="P3" ref="S0"/>
    <pointerType type="P4" ref="S0"/>
    <pointerType type="P5" ref="F0"/>
    <pointerType type="P6" is_restrict="1" ref="char"/>
    <pointerType type="P7" ref="F2"/>
    <structType type="S0">
      <symbols>
        <id type="double">
          <name>real</name>
        </id>
        <id type="double">
          <name>img</name>
        </id>
      </symbols>
    </structType>
    <functionType type="F0" return_type="S0">
      <params>
        <name type="S0">x</name>
        <name type="double">y</name>
      </params>
    </functionType>
    <functionType type="F1" return_type="int">
      <params/>
    </functionType>
```

```
  <functionType type="F2" return_type="int">
    <params/>
  </functionType>
  <functionType type="F3" return_type="S0">
    <params>
      <name type="S0">x</name>
      <name type="double">y</name>
    </params>
  </functionType>
</typeTable>
<globalSymbols>
  <id type="F0" sclass="extern_def">
    <name>complex_add</name>
  </id>
  <id type="S0" sclass="extern_def">
    <name>x</name>
  </id>
  <id type="F1" sclass="extern_def">
    <name>main</name>
  </id>
  <id type="F2" sclass="extern_def">
    <name>printf</name>
  </id>
  <id type="S0" sclass="typedef_name">
    <name>complex_t</name>
  </id>
  <id type="S0" sclass="tagname">
    <name>complex</name>
  </id>
</globalSymbols>
<globalDeclarations>
  <varDecl>
    <name>x</name>
  </varDecl>
  <funcDecl>
    <name>complex_add</name>
  </funcDecl>
  <functionDefinition>
    <name>main</name>
    <symbols>
      <id type="S0" sclass="auto">
        <name>z</name>
      </id>
    </symbols>
    <params/>
    <body>
      <compoundStatement>
        <symbols>
          <id type="S0" sclass="auto">
            <name>z</name>
          </id>
        </symbols>
        <declarations>
          <varDecl>
            <name>z</name>
          </varDecl>
        </declarations>
```

```
      <body>
        <exprStatement>
          <assignExpr type="double">
            <memberRef type="double" member="real">
              <varAddr type="P0" scope="local">x</varAddr>
            </memberRef>
            <floatConstant type="double">1.0</floatConstant>
          </assignExpr>
        </exprStatement>
        <exprStatement>
          <assignExpr type="double">
            <memberRef type="double" member="img">
              <varAddr type="P1" scope="local">x</varAddr>
            </memberRef>
            <floatConstant type="double">2.0</floatConstant>
          </assignExpr>
        </exprStatement>
        <exprStatement>
          <assignExpr type="S0">
            <Var type="S0" scope="local">z</Var>
            <functionCall type="S0">
              <function>
                <funcAddr type="P5">complex_add</funcAddr>
              </function>
              <arguments>
                <Var type="S0" scope="local">x</Var>
                <floatConstant type="double">1.0</floatConstant>
              </arguments>
            </functionCall>
          </assignExpr>
        </exprStatement>
        <exprStatement>
          <functionCall type="int">
            <function>
              <funcAddr type="F2">printf</funcAddr>
            </function>
            <arguments>
              <stringConstant>z=(%f,%f)¥n</stringConstant>
              <memberRef type="double" member="real">
                <varAddr type="P2" scope="local">z</varAddr>
              </memberRef>
              <memberRef type="double" member="img">
                <varAddr type="P3" scope="local">z</varAddr>
              </memberRef>
            </arguments>
          </functionCall>
        </exprStatement>
      </body>
    </compoundStatement>
  </body>
</functionDefinition>
<functionDefinition>
  <name>complex_add</name>
  <symbols>
    <id type="S0" sclass="param">
      <name>x</name>
    </id>
```

```
      <id type="double" sclass="param">
        <name>y</name>
      </id>
    </symbols>
    <params>
      <name type="S0">x</name>
      <name type="double">y</name>
    </params>
    <body>
      <compoundStatement>
        <symbols>
          <id type="S0" sclass="param">
            <name>x</name>
          </id>
          <id type="double" sclass="param">
            <name>y</name>
          </id>
        </symbols>
        <declarations/>
        <body>
          <exprStatement>
            <asgPlusExpr type="double">
              <memberRef type="double" member="real">
                <varAddr type="P4" scope="param">x</varAddr>
              </memberRef>
              <Var type="double" scope="param">y</Var>
            </asgPlusExpr>
          </exprStatement>
          <returnStatement>
            <Var type="S0" scope="param">x</Var>
          </returnStatement>
        </body>
      </compoundStatement>
    </body>
  </functionDefinition>
 </globalDeclarations>
</XcodeProgram>
```