AScript 開発者向けマニュアル

Updated: February 23, 2011

1. この文書について

AScript の本体およびモジュールのビルド方法と、AScript の資源にアクセスするための C++ 関数の仕様について説明します。内容は AScript v0.01 の実装に基づきます（一部 v0.02 の実装内容を含みます)。

2. ソースファイルの入手方法

AScript のソースファイルは、tar ボールのダウンロードまたはレポジトリからのチェックアウトで入手することができます。

2.1. tar ボールのダウンロード

ソースファイルをまとめた tar ボールが、SourceForge.JP のダウンロードページから取得できます。URL は以下の通りです。

http://sourceforge.jp/projects/ascript/releases/

2.2. レポジトリからのチェックアウト

AScript のソースファイルは SourceForge.JP の Subversion レポジトリで管理されています。
リリース済みのソースファイルは `tags` の下のバージョン番号が割り振られたレポジトリ内に格納されています。バージョン v0.01 のソースは以下のように取得できます。

```
$ svn co http:://svn.sourceforge.jp/svnroot/ascript/tags/0.01 ascript
```

開発中のソースファイルは `trunk` 内にあります。

```
$ svn co http:://svn.sourceforge.jp/svnroot/ascript/trunk ascript
```

3. ソースファイルのディレクトリ構成

ソースファイルのディレクトリは以下のようになっています。

ディレクトリ	内容
`build`	Visual Studioのソリューション・プロジェクトファイル
`doc`	ドキュメント
`extra`	Windowsで使用するDLLファイルやインクルードファイルなど
`include`	AScript のインクルードファイル
`lib`	AScript のライブラリファイル
`module`	スクリプトモジュールファイル。 Windowsの場合、Visual Studio用のバイナリモジュールもここに格納します。
`module.bcc`	Windowsの場合、Borland C のバイナリモジュールを格納します。
`sample`	サンプルスクリプト
`src`	ソースファイル

4. 開発環境

以下の開発環境でビルドできます。

- Windows (Borland C)
- Windows (Visual Studio 2005 / 2008 / 2010)
- Ubuntu Linux (gcc)

5. ビルド方法

5.1. Windows (Borland C)

メイクファイル `src¥Makefile.mak` を使用します。コンソールウィンドウを開いてカレントディレクトリを src に移動した後、Borland コンパイラに付属している make ユーティリティを以下のように実行してください。

```
> make
```

各ディレクトリに以下のファイルが生成されます。

ディレクトリ	ファイル
.	ascript.exe, ascriptw.exe, libascript.bcc.dll
lib	libascript.bcc.lib
module.bcc	バイナリモジュール (*.azd)

5.2. Windows (Visual Studio C++)

ディレクトリ build の下に格納されている Visual Studio プロジェクトファイルを使用します。現在 Visual Studio 2005, 2008, 2010 用のものを用意しています。それぞれ対応する以下のソリューションファイルを Visual Studio で開き、構成を "Release" にしてビルドしてください。

環境	ソリューションファイル
Visual Studio 2005	build¥vs2005¥ascript.sln
Visual Studio 2008	build¥vs2008¥ascript.sln
Visual Studio 2010	build¥vs2010¥ascript.sln

各ディレクトリに以下のファイルが生成されます。

ディレクトリ	ファイル
.	ascript.exe, ascriptw.exe, libascript.dll
lib	libascript.lib
module	バイナリモジュール (*.azd)

5.3. Linux (gcc)

autoconf/automake 関連のファイルを使用します。コンソールを開き、カレントディレクトリを `src` に移動してから以下のコマンドを実行してください。

```
$ ./configure
$ make
$ sudo make install
```

各ディレクトリに以下のファイルがインストールされます。

ディレクトリ	ファイル
/usr/local/bin	ascript
/usr/local/lib	libascript.so
/usr/local/lib/ascript	スクリプトモジュール (*.az)
/usr/local/include	ascript.h
/usr/local/include/ascript	ascript.h からインクルードされるヘッダファイル
/usr/local/share/ascript	サンプルスクリプトなど

続けて、モジュールのインストールを行います。同じく src ディレクトリで以下のコマンドを実行してください。

```
$ ascript build_modules.az
$ sudo build_modules.az install
```

これで、/usr/local/lib/ascript にバイナリモジュールがインストールされます。エラーが出る場合は、必要なライブラリがシステムにインストールされていない可能性があります。エラーメッセージに必要な Debian パッケージ名が表示されるので、それに基づいてインストールしてください。

6. バイナリモジュール開発

6.1. 雛形の作成

AScript は、オリジナルのバイナリモジュールの開発を支援するため、ソースファイルの雛形を生成する仕組みを用意しています。以下、仮想のモジュール hoge を作る過程を見ていきます。これらの過程は Linux、Windows とも共通です。

1. コンソールを開き、適当な作業用ディレクトリを作成した後、そのディレクトリ内で以下のコマンドを実行しますビルド用スクリプト build.az とソースファイルの雛形 Module_hoge.cpp が生成されます。

```
$ ascript –i lets_module hoge
```

2. Module_hoge.cpp を編集して、コードを実装します。ビルドするときは、以下のコマンドを実行します。現在のディレクトリに hoge.azd が生成されます。

```
$ ascript build.az --here
```

3. モジュールを AScript のディレクトリにインストールするときは、以下のコマンドを実行します。

```
$ ascript build.az --here install
```

階層構造の下にモジュールを作成するときは、親のモジュール名と本体のモジュール名を引数に指定します。以下に例を示します。

```
$ ascript –i lets_module net hoge
```

これをビルドすると、net_hoge.azd という名前のモジュールファイルが作成されます。install でインストールすると、モジュールディレクトリ中の net ディレクトリに hoge.azd という名前でモジュールファイルをコピーします。

6.2. モジュールソースファイルの内部構成

前節で作成した雛形をもとに、モジュールソースファイルの内部構成をみていきます。以下のソースは、自動生成されたソースからコメントを取り除いたものです。

```
 1  #include <ascript.h>
 2
 3  AScript_BeginModule(hoge)
 4
 5  AScript_DeclareFunction(test)
 6  {
 7      SetMode(RSLTMODE_Normal, FLAG_None);
 8      DeclareArg(env, "num1", VTYPE_Number);
 9      DeclareArg(env, "num2", VTYPE_Number);
10      SetHelp("adds two numbers and returns the result.");
11  }
12
13  AScript_ImplementFunction(test)
14  {
15      return Value(args.GetNumber(0) + args.GetNumber(1));
16  }
17
18  AScript_ModuleEntry()
19  {
20      AScript_AssignFunction(test);
21  }
22
23  AScript_ModuleTerminate()
24  {
25  }
26
27  AScript_EndModule(hoge, hoge)
28
29  AScript_RegisterModule(hoge)
```

1行　　全てのモジュールは ascript.h をインクルードします。

3行　　AScript_BeginModuleマクロでモジュール実装の開始を宣言します。

5-11行　AScript_DeclareFunctionマクロで関数の宣言をし、戻り値や関数のタイプ、引数の名前や型、ヘルプなどを定義します。

13-16行　AScript_DeclareFunction の内容で宣言した関数の実行内容を AScript_ImplementFunctionマクロに続いて記述します。

18-21行　`AScript_ModuleEntry`でモジュールをインポートしたときに実行する内容を記述します。この中には、関数や変数のアサイン、シンボル定義、クラス定義などが含まれます。

23-25行　`AScript_ModuleTerminate`でモジュールを削除したときに実行する内容を記述します。

27行　`AScript_EndModule`マクロでモジュール実装の終了を宣言します。
`AScript_BeginModule`から`AScript_EndModule`までが一つのモジュールの実装単位になります。

29行　`AScript_RegisterModule`で、実装したモジュールの登録を行います。

7. C++ インターフェース

AScript の本体から提供されるマクロ、クラスおよび関数を説明します。

7.1. モジュールを構成する最小要素

AScript_BeginModule(name)
AScript_EndModule(name)
AScript_ModuleEntry()
この関数の中では以下の変数が参照できます。
env　Environment インスタンスの参照です。
sig　Signal インスタンスです。
AScript_ModuleTerminate()
AScript_RegisterModule()

7.2. シンボル定義

AScript_DeclarePrivSymbol(name)
AScript_RealizePrivSymbol(name)

7.3. 関数定義

AScript_DeclareFunction(name)
AScript_ImplementFunction(name)
AScript_AssignFunction(name)

7.4. クラス定義

AScript_DeclarePrivClass(name)
AScript_ImplementPrivClass(name)
AScript_RealizePrivClass(name, str, pClassBase)

7.5. メソッド定義

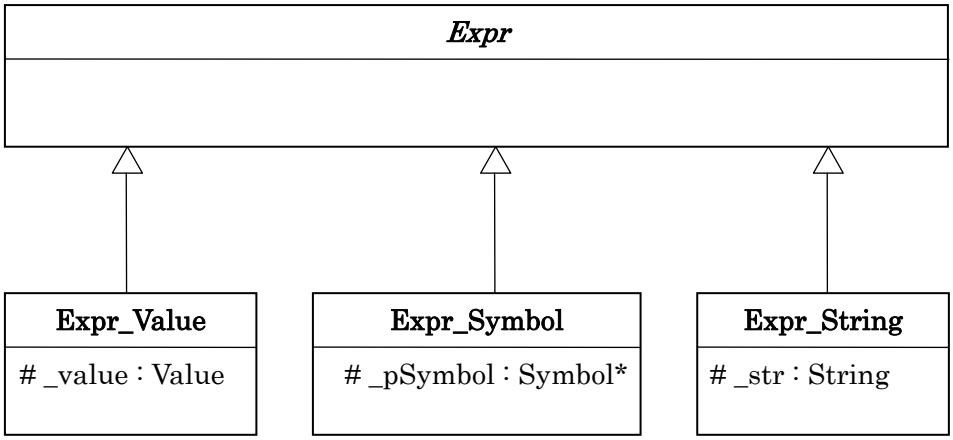
AScript_DeclareMethod(name)
AScript_ImplementMethod(name)
AScript_AssignMethod(name)

8. 式を構成する要素

AScript の構文を構成する要素は、構文木を構成する際にリーフになるものとノードになるものに大別されます。

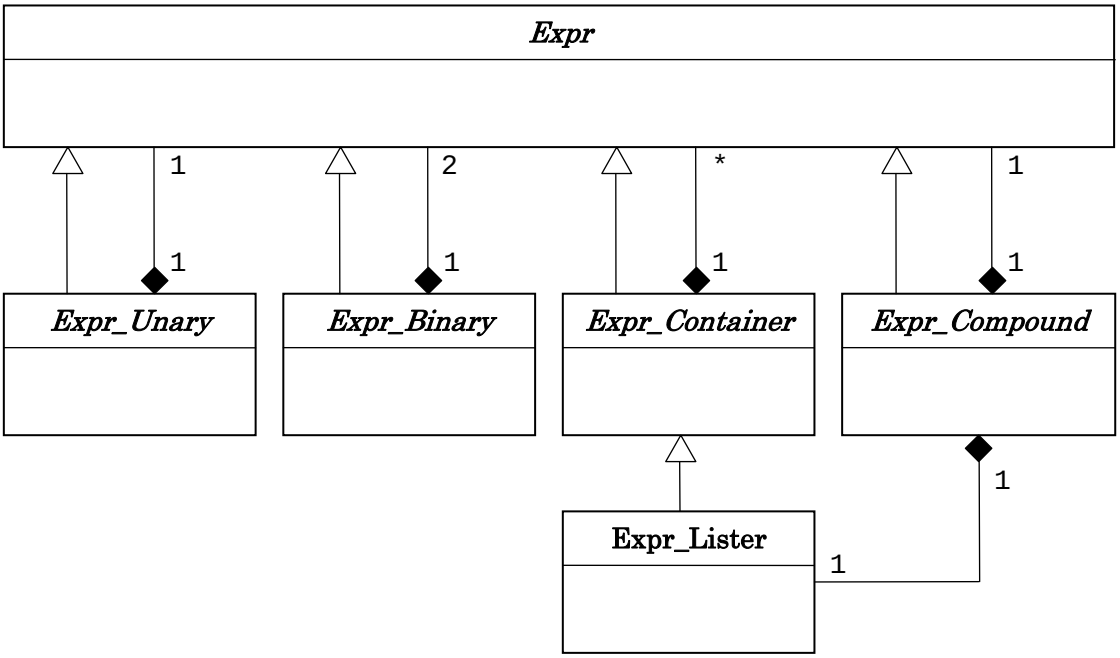
8.1. リーフになる式

リーフになる式には、Expr から派生した Expr_Value, Expr_Symbol, Expr_String があります。

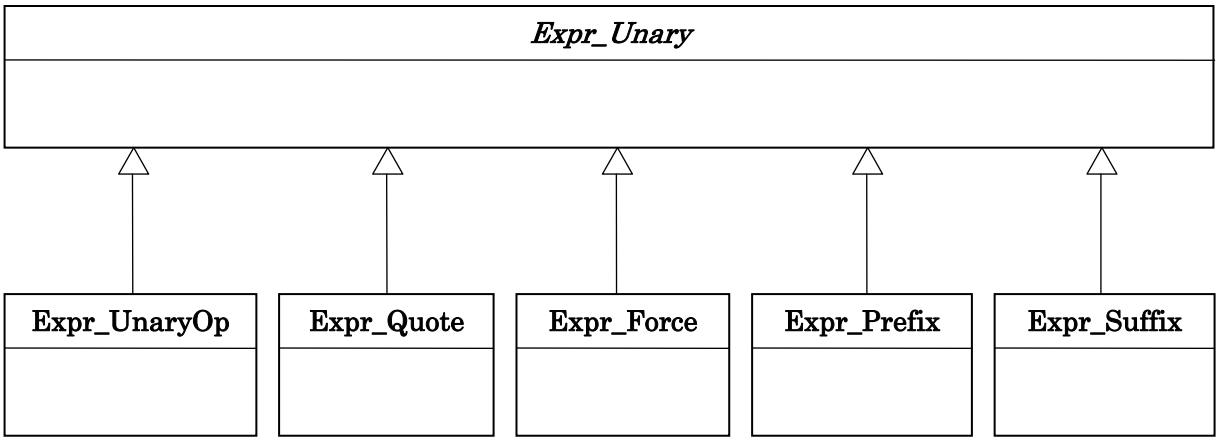


8.2. ノードになる式

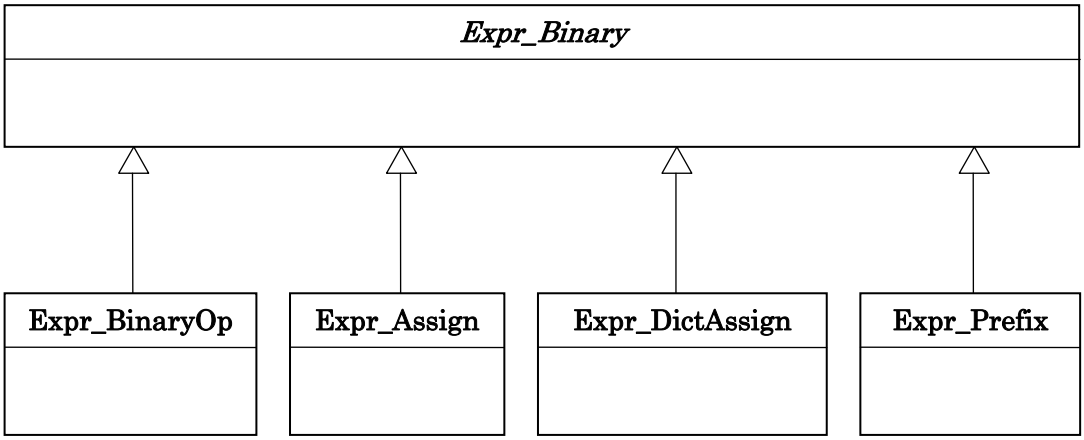
ノードになる式には、Expr から派生した Expr_Unary, Expr_Binary, Expr_Container および Expr_Compound があります。



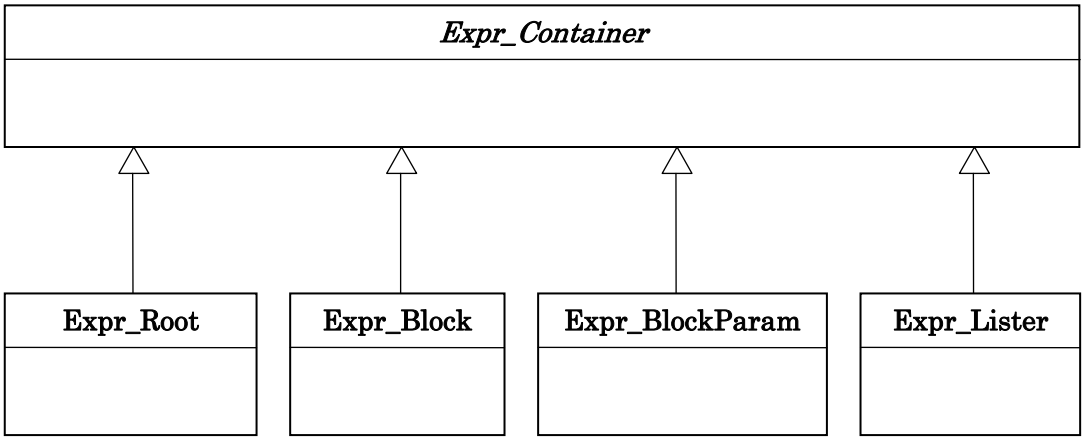
8.2.1. Expr_Unary からの派生クラス



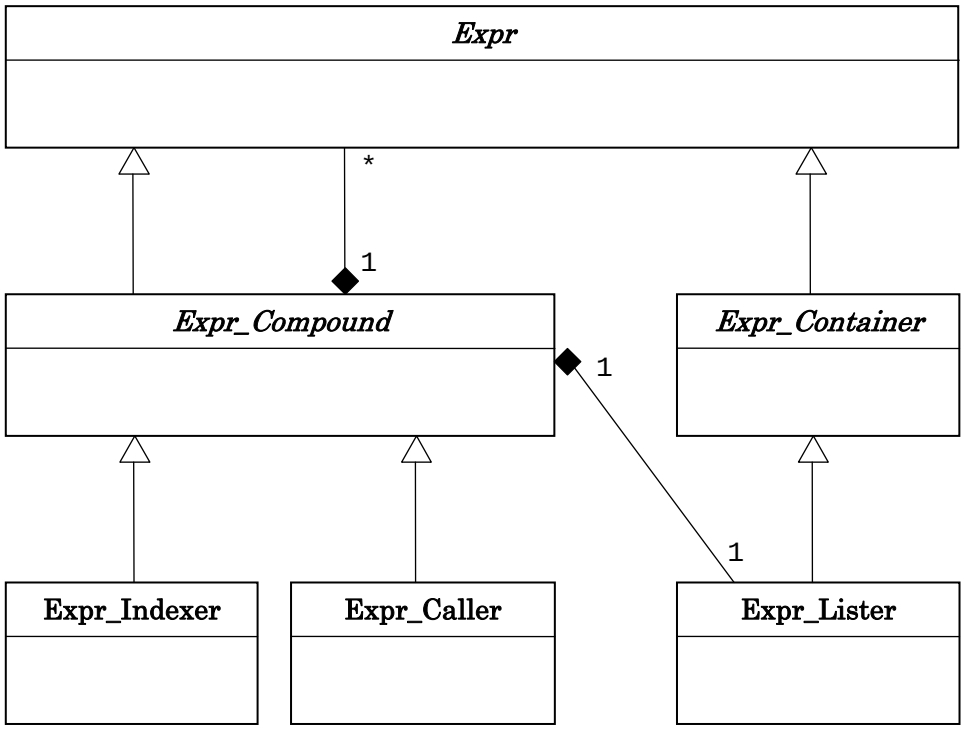
8.2.2. Expr_Binary からの派生クラス



8.2.3. Expr_Container からの派生クラス



8.2.4. 複合式


Expr
*
1
Expr_Compound
Expr_Container
1
1
Expr_Indexer
Expr_Caller
Expr_Lister