# Gura 開発者向けマニュアル

Updated: March 11, 2011

copyright © 2011 Yutaka SAITO

# 目次

# 1. この文書について

Gura の本体およびモジュールのビルド方法と、Gura のライブラリやインクルードファイルが提供する関数・マクロ・クラスについて説明します。

内容は Gura v0.1.0 の実装に基づきます。

# 2. ソースファイルの入手方法

Gura 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/gura/releases/

## 2.2. レポジトリからのチェックアウト

Gura 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。開発中のソースファイルは `trunk` レポジトリ内にあります。以下のようにチェックアウトしてください。

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

# 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` | Gura のインクルードファイル |
| `lib` | Gura 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。<br>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

以下、Windows と Linux のコンソールプロンプトをそれぞれ ">" および "$" で表します。

### 5.1. Windows (Borland C)

メイクファイル gura¥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 gura¥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 |
|---|---|
| gura | gura.exe, guraw.exe, libgura.bcc.dll |
| gura¥lib | libgura.bcc.lib |
| gura¥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 | gura¥build¥vs2005¥gura.sln |
| Visual Studio 2008 | gura¥build¥vs2008¥gura.sln |
| Visual Studio 2010 | gura¥build¥vs2010¥gura.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 |
|---|---|
| gura | gura.exe, guraw.exe, libgura.dll |
| gura¥lib | libgura.lib |
| gura¥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 gura/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 | gura |
| /usr/local/lib | libgura.so |
| /usr/local/lib/gura | スクリプトモジュール (*.az) |
| /usr/local/include | gura.h |
| /usr/local/include/gura | gura.h からインクルードされるヘッダファイル |
| /usr/local/share/gura | 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。

```
$ gura build_modules.az
$ sudo build_modules.az install
```

これで、/usr/local/lib/gura 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

Gura は、バイナリモジュールを開発するためのフレームワークを用意しています。以下、バイナリモジュール hoge を作る過程を見ていきます。

### 6.1. 雛形の作成

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。

```
$ gura -i lets_module hoge
```

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。

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。

```
$ gura -i lets_module net hoge
```

### 6.2. ビルド方法

以下のコマンドを実行すると、ソースファイルのコンパイルおよびリンクを行ってバイナリモジュールを生成します。

```
$ gura build.az
```

バイナリモジュールを出力するディレクトリは開発環境によって異なり、以下のようになります。

| | |
|---|---|
| Borlanc C | bcc |
| Visual C++ | msc |
| gcc | gcc |

実際の開発プロセスではビルドとテストを繰り返すことになります。モジュールのサーチパスにはカレントディレクトリが含まれるので、バイナリモジュールがソースファイルと同じディレクトリに出力されると便利です。そのようなときは以下のように --here オプションをつけてビルドします。

```
$ gura build.az --here
```

### 6.3. インストール方法

モジュールを Gura 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。

```
$ sudo gura build.az install
```

### 6.4. モジュールソースファイルの内部構成

自動生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見ていきます。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 <gura.h>
 2
 3  Gura_BeginModule(hoge)
 4
 5  Gura_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  Gura_ImplementFunction(test)
14  {
15      return Value(args.GetNumber(0) + args.GetNumber(1));
16  }
17
18  Gura_ModuleEntry()
19  {
20      Gura_AssignFunction(test);
21  }
22
23  Gura_ModuleTerminate()
24  {
25  }
26
```

```
27  Gura_EndModule(hoge, hoge)
28
29  Gura_RegisterModule(hoge)
```

1行　　全てのモジュールは gura.h をインクルードします。

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

5-11行　Gura_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　Gura_DeclareFunction の内容で宣言した関数の実行内容を Gura_ImplementFunctionマクロに続いて記述します。

18-21行　Gura_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　Gura_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　　Gura_EndModuleマクロでモジュール実装の終了を宣言します。
Gura_BeginModuleからGura_EndModuleまでが一つのモジュールの実装単位になります。

29行　　Gura_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

Gura のライブラリやインクルードファイルが提供する関数・マクロ・クラスについて説明します。

### 7.1. モジュールのフレームワークを構成する要素

`Gura_BeginModule(name)`

モジュールの開始を宣言します。name にはモジュール名を指定します。

`Gura_EndModule(name, alias)`

モジュールの終了を宣言します。name には Gura_BeginModule で指定したものと同じ名前を渡します。alias は通常は name と同じものを指定します。階層構造を持ったモジュールの場合、alias はモジュールのベース名を指定します。

`Gura_ModuleEntry()`

モジュールをインポートしたときに呼ばれる関数を定義します。関数の内容を、このマクロに続いて "{" と "}" の間に記述します。

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。 |

`Gura_ModuleTerminate()`

モジュールを解放したときに呼ばれる関数を定義します。関数の内容を、このマクロに続いて "{" と "}" の間に記述します。

`Gura_RegisterModule(name)`

モジュールを登録します。name は Gura_BeginModule で指定したものを渡します。

### 7.2. シンボル定義

`Gura_DeclarePrivSymbol(name)`

モジュール内で使用するシンボルを宣言します。シンボルはスクリプト全体で管理されるので、すでに Gura 本体で宣言されているものと同じ名前のシンボルをここで宣言しても、メモリ効率などに影響しません。

`Gura_RealizePrivSymbol(name)`

Gura_DeclarePrivSymbol で宣言したシンボルを生成します。通常 Gura_ModuleEntry の関数内に記述します。

### 7.3. 関数定義

`Gura_DeclareFunction(name)`

関数の宣言をします。

`Gura_ImplementFunction(name)`

関数の処理内容を、このマクロに続いて "{" と "}" の間に記述します。

`Gura_AssignFunction(name)`

通常 Gura_ModuleEntry の関数内に記述します。

### 7.4. クラス定義

`Gura_DeclarePrivClass(name)`

クラスの宣言をします。

`Gura_ImplementPrivClass(name)`

クラスの内容を、このマクロに続いて "{" と "}" の間に記述します。

`Gura_RealizePrivClass(name, str, pClassBase)`

通常 Gura_ModuleEntry の関数内に記述します。

### 7.5. メソッド定義

`Gura_DeclareMethod(name)`

`Gura_ImplementMethod(name)`

メソッドの処理内容を、このマクロに続いて "{" と "}" の間に記述します。

`Gura_AssignMethod(name)`

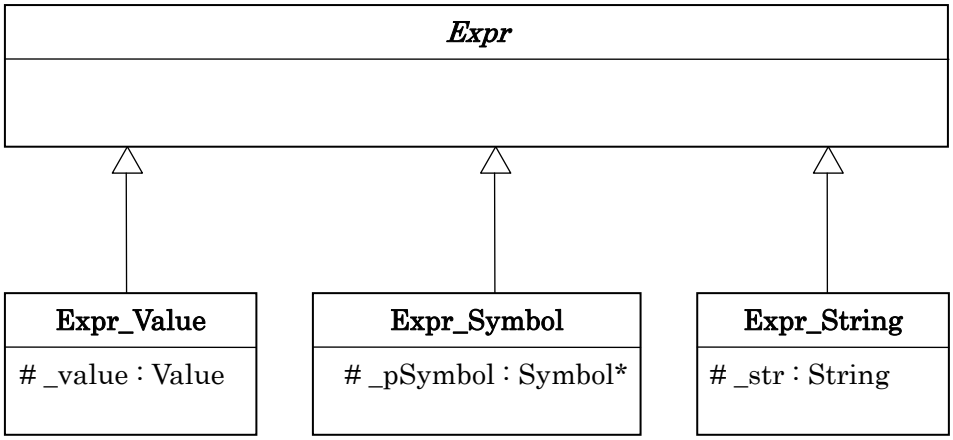
通常 Gura_ImplementPrivClass の関数内に記述します。

# 8. 式を構成する要素

Gura 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。

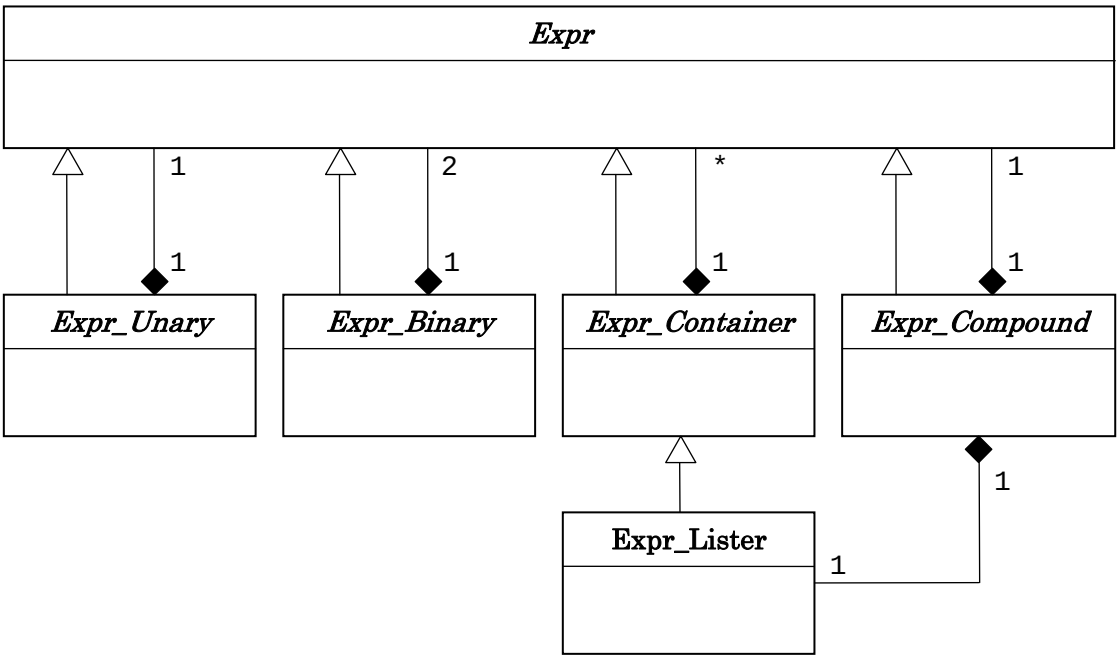
## 8.1. リーフになる式

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。

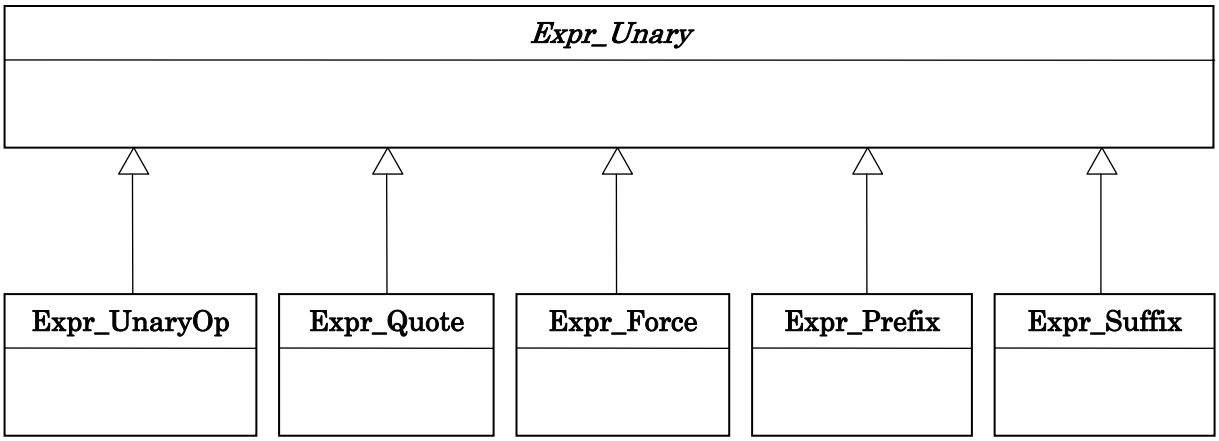



## 8.2. ノードになる式

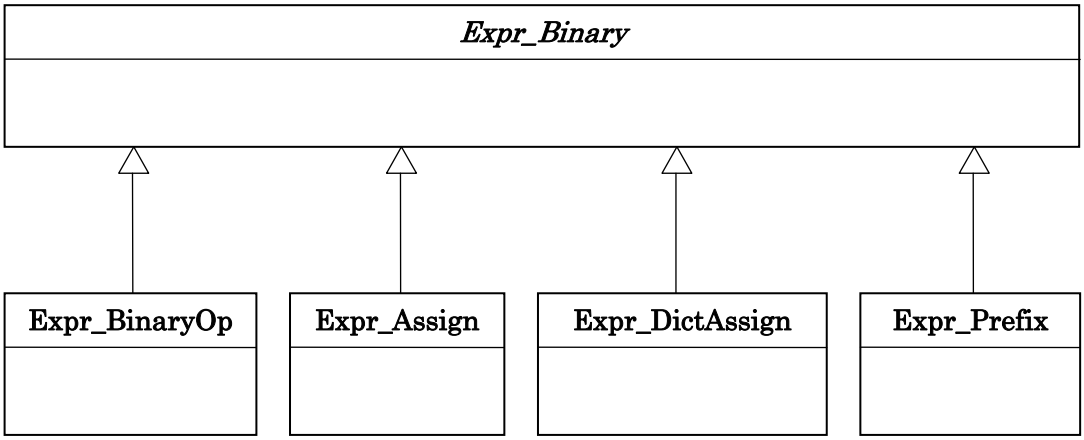
ノードになる式には、Expr から派生した Expr_Unary, Expr_Binary, Expr_Container および Expr_Compound があります。



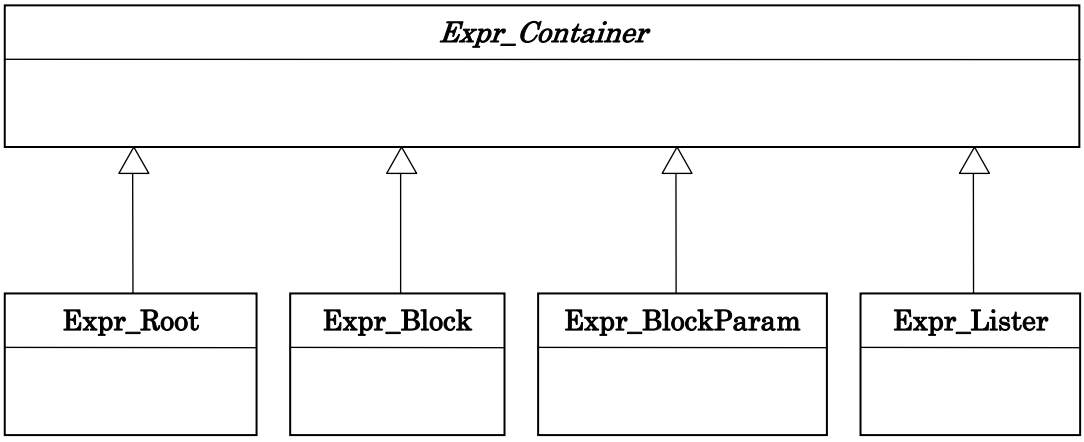
### 8.2.1. Expr_Unary からの派生クラス



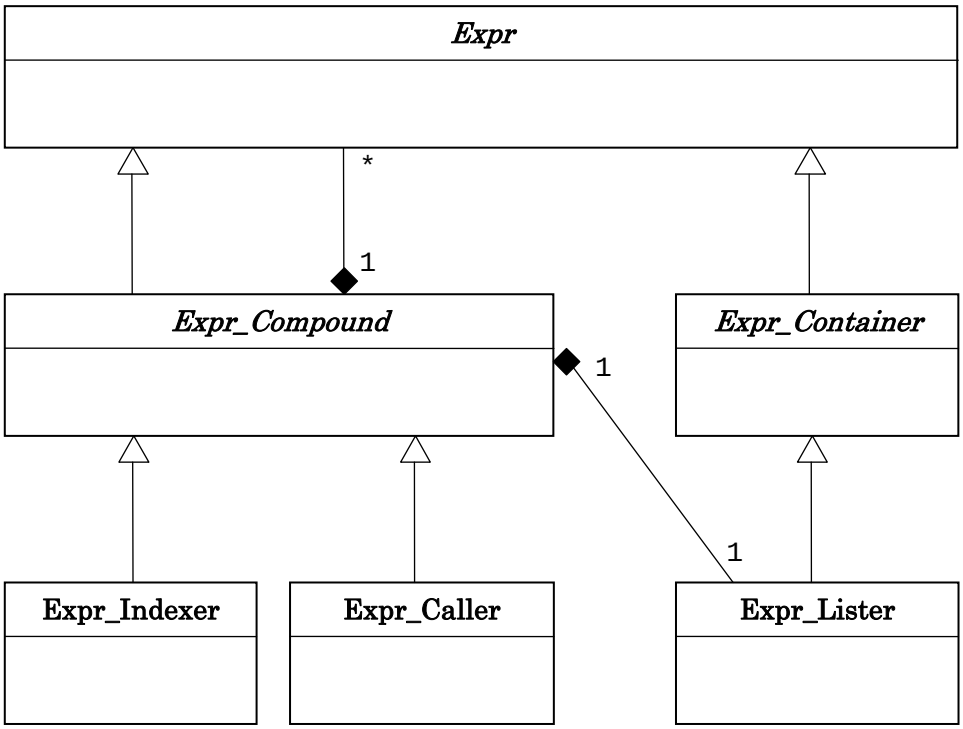

### 8.2.2. Expr_Binary からの派生クラス




### 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