AnalysisDrive 1.1.2

Reference Guide

2015/07/12

1. 概要

本ライブラリは，数値解析プログラムを作成する上で必要になるパーサーなどの補助的な機能を集めたものである。

2. ライセンス

本ライブラリは，次の三条項 BSD ライセンスに従う。

参考までに，日本語訳を下に示す。

ただし，本ライブラリは Apache License, Version 2.0 のライセンスで配布されている成果物を含んでいる。該当する機能と対応する成果物の関係は次の通りである。

該当する機能	Apache License, Version 2.0 で配布されている成果物
MsgpackParser クラス	Msgpack Copyright (C) 2008-2014 FURUHASHI Sadayuki
Json クラスの DumpMsgpack 関数	

Apache License, Version 2.0 については，http://www.apache.org/licenses/LICENSE-2.0 を参照されたい。

3. ディレクトリ構成

ディレクトリ構成を下に示す。

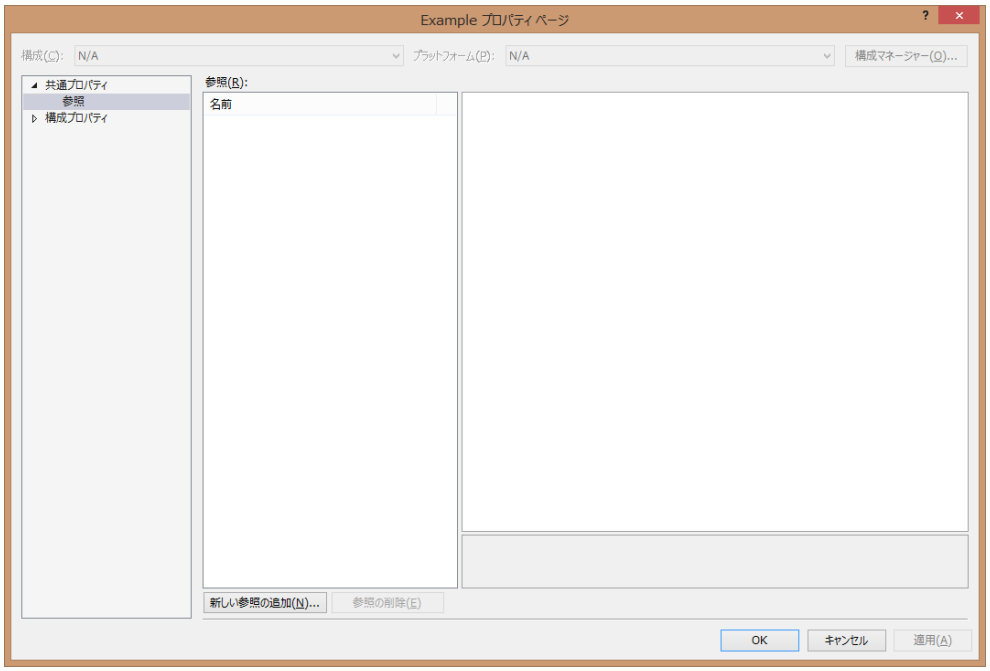
AnalysisDrive

- doc … ドキュメント(index というショートカットをクリックすると閲覧できる)
- AnalysisDrive　本ライブラリと必要なヘッダーが入ったフォルダ
  このフォルダがあれば本ライブラリを使用できる
- src … 本ライブラリのソースコード(.cpp)
  改変する場合のみ必要
- inc … 本ライブラリのソースコード(.h)
  改変する場合のみ必要
- libs…　本ライブラリで使用している他のライブラリ
  改変する場合のみ必要
- Setting … Visual Studio 2013 の設定データ(.vssettings)が入ったフォルダ
  特に必要ない
- AnalysisDrive.sln … Visual Studio 2013 のソリューションファイル
  改変する場合のみ必要
- LICENSE.txt … ライセンスが記載されたテキストファイル
  再配布時はこのファイルを使用すれば良い

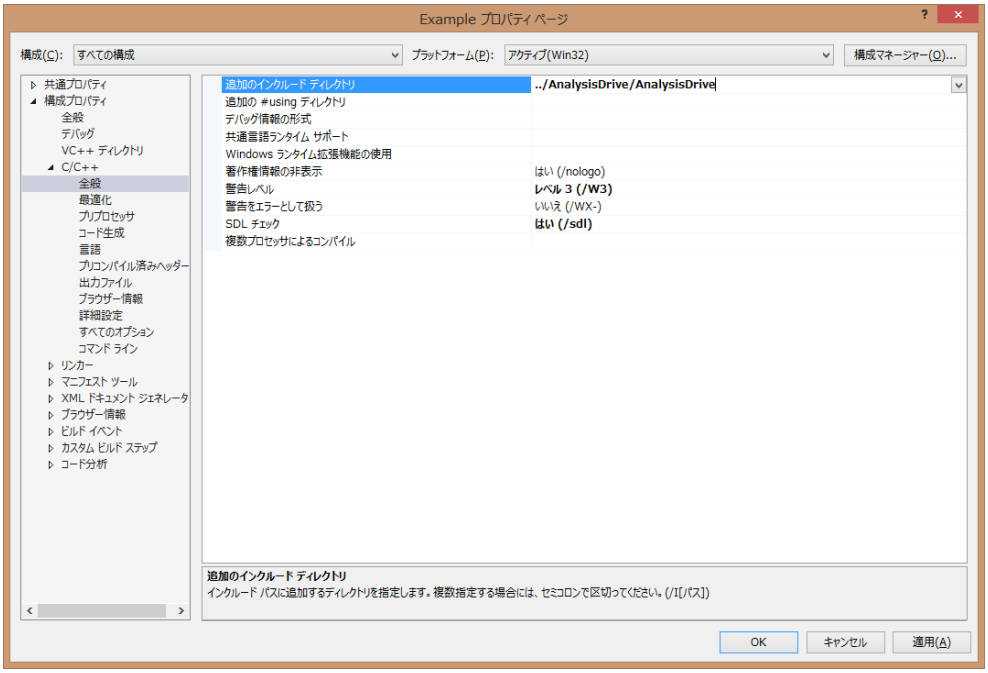
4. 導入方法(Visual Studio 2013)

Visual Studio 2013 での導入方法について説明する.

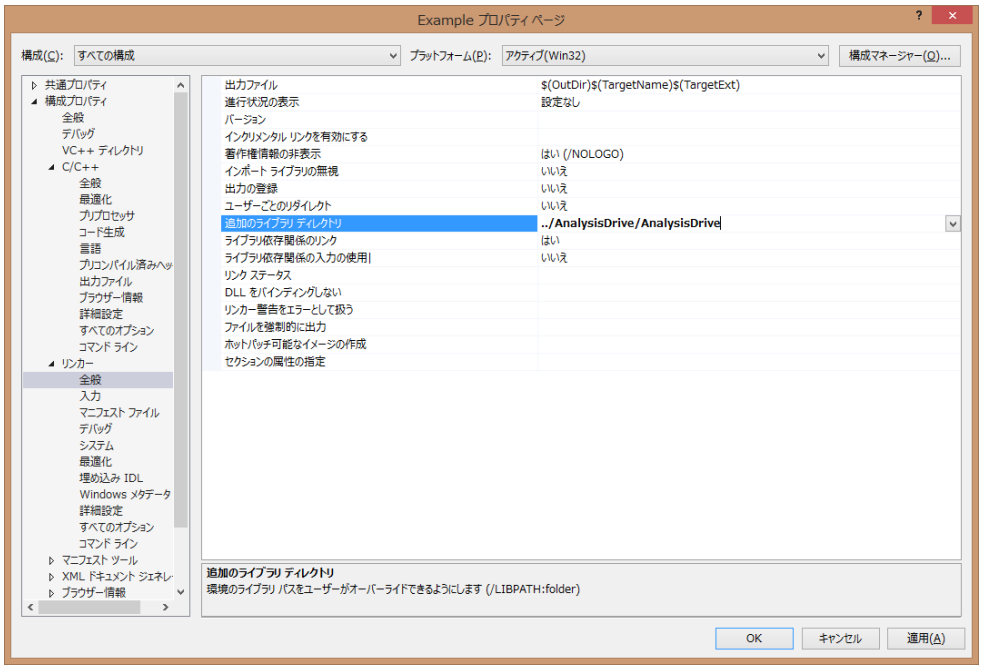
① プロジェクト作成後，メニューから
［プロジェクト］⇒［***のプロジェクト］でプロパティページを開く



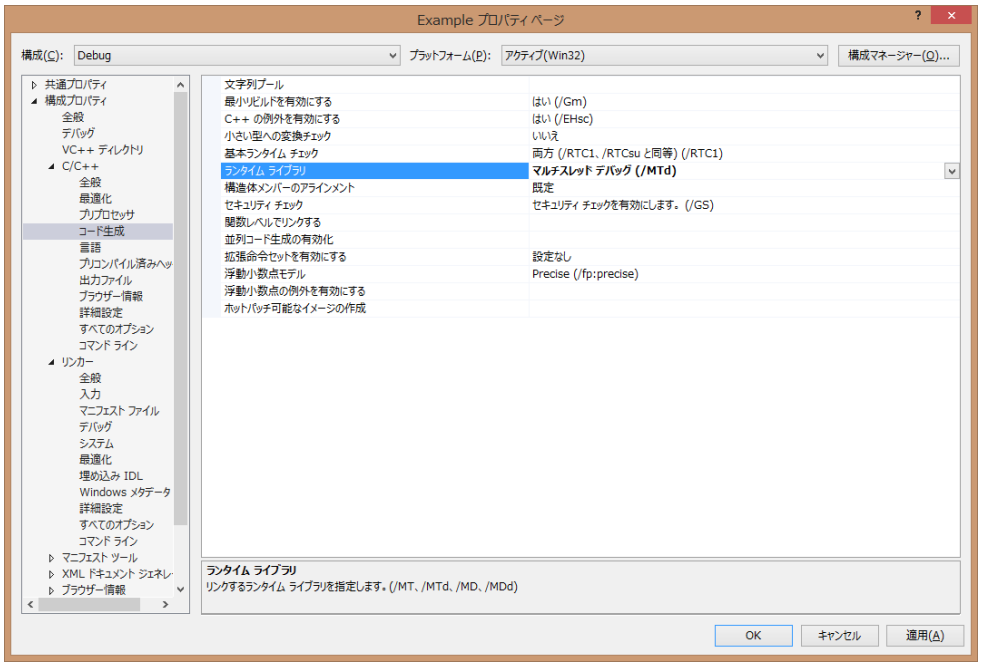
② 左上の構成を[すべての構成]にし，
［C/C++］⇒［全般］と進んで追加のインクルードディレクトリに
AnalysisDrive/AnalysisDrive のパスを追加する(このフォルダだけ必要).



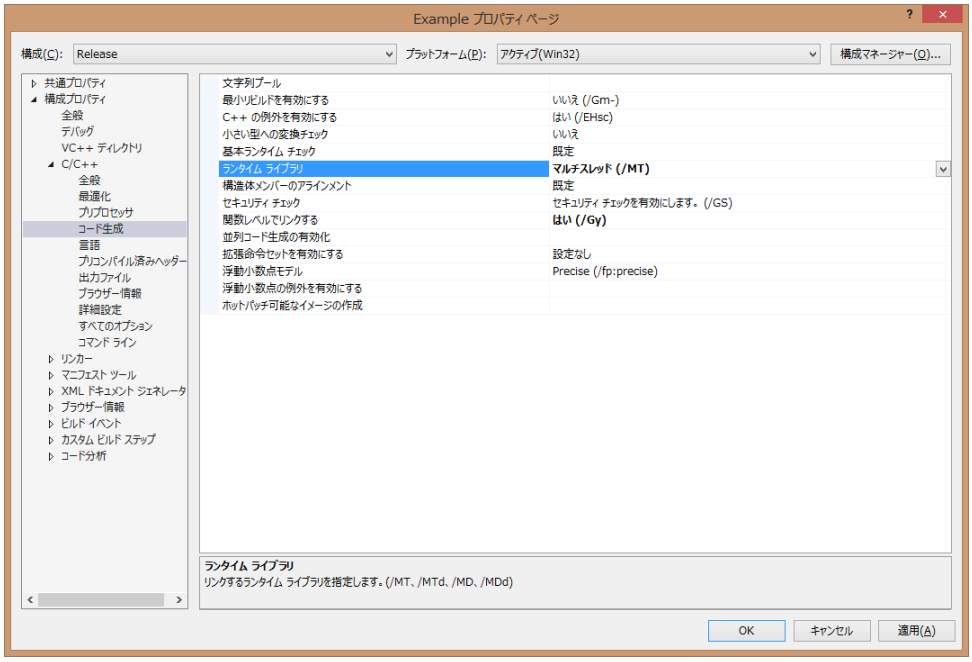
③ ［リンカー］⇒［全般］で追加のライブラリディレクトリに
AnalysisDrive/AnalysisDrive のパスを追加する.



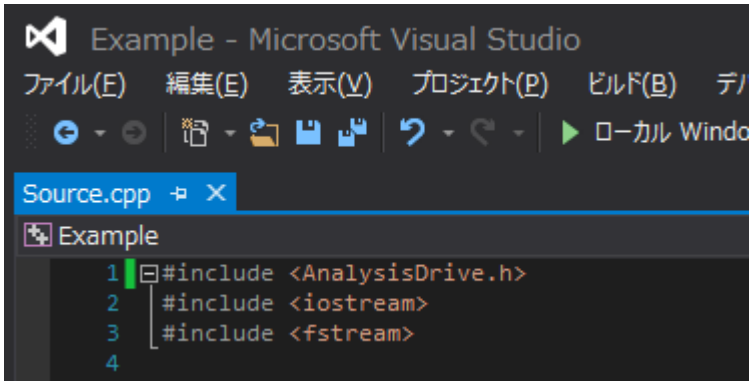
④ 左上の構成を[Debug]にし,
[C/C++] ⇒［コード生成］と進んでランタイムライブラリを/MTd にする.



⑤ 左上の構成を[Release]にし，
[C/C++] ⇒ [コード生成] と進んでランタイムライブラリを/MT にする.



⑥ <AnalysisDrive.h>をインクルードして使用する.

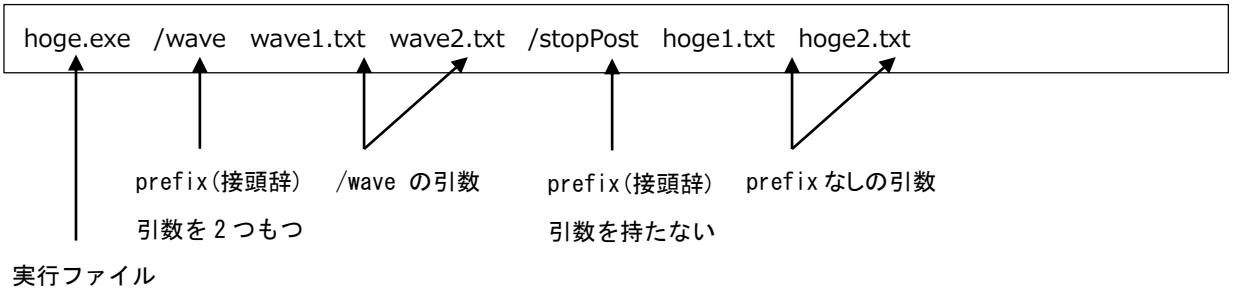


5. 機能紹介と使用方法

5.1. コマンドライン引数のパース

5.1.1. 概要

本機能を使用すると，次のような形式のコマンド内容をパースできる(接頭辞や引数の数を設定できる).



接頭辞の文字，接頭辞の後ろの引数の数を定義することができる.

5.1.2. 使用方法

コマンドライン引数のパースをするには次の 3 つのクラスを使用する.

class adlib::CommandLineSetting (コマンドラインの読込形式を定義するクラス)			
メンバ関数	内容	引数	返り値
CommandLineSetting	コンストラクタ	[第 1 引数] 接頭辞(空文字は禁止) [第 2 引数] 接頭辞の後ろの引数の数 [第 3 引数] ヘルプ表示	なし
(static) ShowHelp	ヘルプを表示する	[第 1 引数] 本クラスが任意数格納されたコンテナ(std::vector) [第 2 引数] 接頭辞なしの引数に関するヘルプ表示 [第 3 引数] 出力先のストリーム(std::ostream*)	なし

class adlib::CommandLineParser　(コマンドラインの内容をパースするクラス)			
メンバ関数	内容	引数	返り値
Parse	コマンドライン引数をパースする	[第 1 引数] コマンドラインの読み込み形式定義(adlib::CommandLineSetting)が格納されたコンテナ(std::vector) [第 2 引数] main 関数の第 1 引数(argc) [第 3 引数] main 関数の第 2 引数(argv)	パース結果(adlib::Command)が格納されたコンテナ(std::map) ※　map のキーはコマンドラインの接頭辞
GetCommandArguments	前回 Parse に渡したコマンドラインの内容を繋げて文字列にして返す	なし	前回 Parse に渡したコマンドラインの内容(std::string)
ErrorMessage	エラーメッセージを取得する	なし‘	エラーメッセージが格納されたコンテナ(std::vector)

class adlib::Command　(パース結果)			
メンバ関数	内容	引数	返り値
Prefix	コマンドラインに入力された接頭辞を取得する	なし	接頭辞(const std::string) “”であれば接頭辞なしの引数
Args	引数が格納されたコンテナの参照を取得する	なし	引数が格納されたコンテナの参照(const std::vector<std::string>&)

まず，CommandLineSetting クラスでコマンドラインの内容(接頭辞，接頭辞の後ろの引数の数，ヘルプ表示)を定義する．1 つの CommandLineSetting クラスで 1 つの接頭辞を定義し，接頭辞の数だけコンテナ(vector)に格納する．空の接頭辞”“は使用してはならない(接頭辞なしの引数は定義しなくても取得できる)．

```
// コマンドラインの定義(接頭辞, 接頭辞の後ろの引数の数, ヘルプ表示)
std::vector<adlib::CommandLineSetting> settings =
{
  adlib::CommandLineSetting(“/?”, 0, “ヘルプを表示するフラグ”),
  adlib::CommandLineSetting(“/wave”, 2, “入力波のファイルパス”),
  adlib::CommandLineSetting(“/stopAnalysis”, 0, “解析を行わないフラグ”),
  adlib::CommandLineSetting(“/stopPost”, 0, “ポスト処理を行わないフラグ”)
};
```

次に，CommandLineParser クラスの Parse 関数でコマンドライン引数をパースする．その際，コマンドラインの定義のコンテナを第 1 引数にセットする．Parse 関数はパース結果として Command クラスのコンテナ(map)を返す．例を示す．

入力されたコマンドライン(実行ファイル名を hoge.exe とする)

```
hoge.exe   /wave   wave1.txt   wave2.txt   /stopPost   hoge1.txt   hoge2.txt
```

値を取得するコード

```
int main(int argc, const char* argv[])
{
  adlib::CommandLineParser parser;
  // コマンドライン引数のパース(Command クラスの map が返される)
  auto commands = parser.Parse(settings, argc, argv);

  // “/wave”のチェック(map クラスの count により入力されているか確認する)
  if (commands.count(“/wave”) > 0)
  {
    // Command クラスの NumberOfArguments 関数で引数の数を確認してループ
    for (int i = 0; i < commands[“/wave”].Args().size(); ++i)
    {
      // []演算子でインデックスを指定すれば引数を参照できる
      std::cout << commands[“/wave”].Args()[i] <<std::endl;
    }
  }

  // 接頭辞なしの引数は“”で参照できる
  if (commands.count(“”) > 0)
  {
    for (int i = 0; i < commands[“”].Args().size(); ++i)
    {
      std::cout << commands[“”].Args()[i] <<std::endl;
    }
  }
```

出力結果

```
wave1.txt
wave2.txt
hoge1.txt
hoge2.txt
```

5.1.3. エラーメッセージ

CommandLineParser クラスの Parse 関数がコマンドライン引数のパースに失敗した場合，空の map を返す．その際のエラー内容は，CommandLine クラスの ErrorMessage 関数により，エラーメッセージの入ったコンテナ(std::vector<std::string>型)を取得することで分かる．このコンテナには，前回の Parse 関数の結果が格納される．エラーが発生していなければサイズが 0 となる(vector の size 関数で確認できる)．エラーメッセージの内容は次の通りである．

エラーメッセージ	エラー内容
(Program Error) Prefix "***" is Defined Double.	設定された接頭辞が重複している (ライブラリ使用者のエラー).
(Program Error) There is no Commandline Contents.	コマンドライン引数の内容がない (実行ファイル名もない．ライブラリ使用者のエラー).
(Program Error) There is Prefix "***" in Commandline Setting.	設定された接頭辞に空文字""がある (ライブラリ使用者のエラー).
Prefix "***" is Input Double.	入力された接頭辞が重複している.
Parameter of Prefix "***" is Insufficient.	接頭辞の後ろの引数が不足している.
Parameter "***" is Invalid.	定義されていない接頭辞が入力されている.

5.1.4. その他の機能

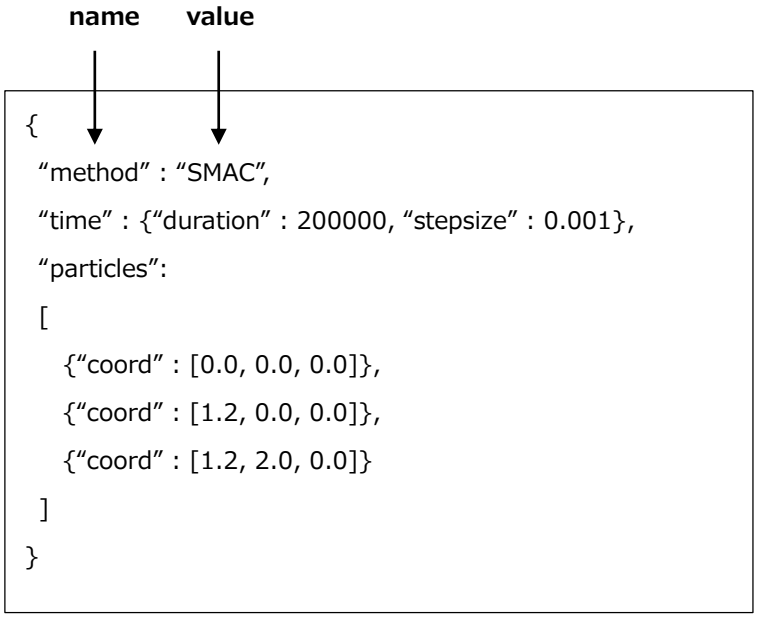
その他に次の機能がある.

- CommandLineSetting クラスの static メンバである ShowHelp 関数を使用すれば，ヘルプを表示できる(関数の仕様は上述の通り).
- CommandLineParser クラスの GetCommandArguments 関数を使用すれば，前回 Parse した際のコマンドの内容を文字列で取得できる(関数の仕様は上述の通り).

5.2. JSON のパース

5.2.1. 概要

JSON は次のように，name/value のペアの集まりで記述されるデータ言語である。



value には次の種類がある。

value	内容	例
string	""で囲まれた文字列(UNICODE の並び)	"SMAC"
number	数値(整数や浮動小数点数)	200000
object	{}で囲まれた name/value ペアのセット	{"duration" : 200000, "stepsize" : 0.001}
array	[]で囲まれた value の集まり(配列)	[1.2, 2.0, 0.0]
bool	true or false	true
null	-	null

トップレベルの要素として，object か array を 1 つ記述する(上の例では object としている)．
本ライブラリを使用すれば，JSON を分析し，value を取得することができる．ただし，文字コードセットは UTF-8 に限定する(BOM の有無は問わない)．ASCII の範囲内(ローマ字，数字など)であれば他の文字コードセットでも UTF-8 と同様に扱える可能性がある(その場合は BOM 無しの形式にするか，BOM を除外してからパースする必要がある)．

5.2.2. 使用方法

詳しい仕様は doc フォルダ内の Doxygen ドキュメントや，実際のソースコードを参考にされたい．ここでは基本的な使用方法について説明する．

次の JsonParser クラスを使用すれば，JSON 形式の文字列をパースすることができる．

class adlib::JsonParser　(JSON 形式の文字列をパースするクラス)			
メンバ関数	内容	引数	返り値
Parse	JSON 形式の文字列をパースする	パースする文字列	Json オブジェクト
ErrorMessage	エラーメッセージを取得する	なし‘	エラーメッセージが格納されたコンテナ(std::vector)
WarningMessage	ワーニングメッセージを取得する	なし‘	ワーニングメッセージが格納されたコンテナ(std::vector)
IsNameRepeated	前回の Parse で object の name が重複していたか確認	なし	前回の Parse で name が重複していれば true を返す

Parse 関数に JSON 形式の文字列を渡すと，Json というクラスを返す．

```
// JSON で書かれたファイルを読み込む
std::ifstream ifs;
ifs.open(“test.txt”);

// バッファにファイルの内容を格納
std::string buf(“”);
std::string tmp(“”);
while(std::getline(ifs, tmp)) buf += tmp;

// パーサーを用意
adlib::JsonParser parser;
// パースして Json オブジェクトを取得
adlib::Json json = parser.Parse(buf);
```

JsonParser は JSON の各 value を，Json というクラスに変換する．この Json オブジェクトを通して value を取得できる．JSON の value の型によって，取得できる値の型は決まっている．その対応は次の通りである(adlib は本ライブラリの名前空間)．

value (JSON)	Json オブジェクトから取得できる値の型	取得に使用する関数 (Json のメンバ関数)
string	std::string	GetString()
number	int, double (選択可能)	int なら GetNumberInt() double なら GetNumberDouble()
object	std::map<std::string, adlib::Json>	GetObject()
array	std::vector<adlib::Json>	GetArray()
true	bool	GetBool()
false	bool	GetBool()
null	なし (内部的には std::nullptr_t で扱う)	IsNull() で true が返ったら null

尚，前述のコードで取得している Json オブジェクト(変数名 json)は，ファイルに記述されたトップレベルの object(あるいは array)である.
例を示す．次の JSON をパースしたとき，その下に記載しているコードで値を取得できる.

読み込んだ JSON

```
{
 “method” : “SMAC”,
 “time” : { “duration” : 200000, “stepsize” : 0.001 },
 “particles”: [ {“coord” : [0.0, 0.0, 0.0]}, {“coord” : [1.2, 0.0, 0.0]}, {“coord” : [1.2, 2.0, 0.0]} ]
}
```

値を取得するコード

```
// “method”の値を string として取得
auto sMethod = json.GetObject()[“method”].GetString();
// “method”の値を表示
std::cout << sMethod << std::endl;

// “time”の中の”stepsize”の値を double として取得
auto dStepsize = json.GetObject()[“time”].GetObject()[“stepsize”].GetNumberDouble();
// “stepsize”の値を表示
std::cout << dStepsize << std::endl;
```

出力結果

```
SMAC
0.001
```

array は vector として取得できるため，上例の”particles”の内容は次のように取得する.

値を取得するコード

```
// “particles”の値を vector として取得
auto vParticles = json.GetObject()[“particles”].GetArray();
// イテレータで vector の成分にアクセス
for (auto p = vParticles.begin(); p != vParticles.end(); ++p)
{
  // “coord”の成分数を取得
  size_t size = p->GetObject()[“coord”].GetArray().size();
  // インデックスで vector の成分にアクセス(上の for 文のようにイテレータでアクセスしてもよい)
  for (size_t i = 0; i < size; ++i)
  {
    auto x = p-> GetObject()[“coord”].GetArray()[i].GetNumberDouble();
    std::cout << x  << “ “ << std::endl;
  }
  std::cout << std::endl;
}
```

出力結果

```
0 0 0
1.2 0 0
1.2 2 0
```

ここまで簡単な例を示してきたが，厳密には次のような仕様になっているため注意されたい.

■ **JSON の value の型と取得に使用した関数が対応していない場合，無意味な値が返される**

例えば value が string であるのにも関わらず，GetNumberInt で int 値を取得しようとした場合，value の文字に関わらず 0 が返る．失敗時に返される値は次の通りである.

Json のメンバ関数	失敗時に返される値
GetString()	“”
GetNumberInt()	0
GetNumberDouble()	0.0
GetObject()	サイズ 0 の map
GetArray()	サイズ 0 の vector
GetBool()	false

そのため，value が想定通りの型であるかを，あらかじめ調べておくことを推奨する．これには次の関数を使用する.

確認する JSON の型	調べるための関数 （Json のメンバ関数．型があっていれば true が返る）
string	IsString()
number	IsNumber() ※　IsNumberInt(), IsNumberDouble()で 整数か浮動小数点数かを調べられる
object	IsObject()
array	IsArray()
true	IsBool()
false	IsBool()
null	IsNull()

また，Json のメンバ関数 GetValueType の返り値によって型を調べることもできる.

Json のメンバ関数 GetValueType()の返り値	JSON の型
Json::JSON_VALUE_TYPE::STRING	string
Json::JSON_VALUE_TYPE::NUMBER_INT	number (整数)
Json::JSON_VALUE_TYPE::NUMBER_DOUBLE	number (浮動小数点数)
Json::JSON_VALUE_TYPE::OBJECT	object
Json::JSON_VALUE_TYPE::ARRAY	array
Json::JSON_VALUE_TYPE::BOOL	bool (true または false)
Json::JSON_VALUE_TYPE::NUL	null

■ **GetObject で返される map に存在しない name でアクセスしたとき，null の Json オブジェクトが map に追加される**

例を示す.

読み込んだ JSON

```
{ “method” : “SMAC” }
```

値を取得するコード

```
// “met”の値を string として取得
auto sMethod = json.GetObject()[“met”].GetString();
// “met”の値を表示
std::cout << sMethod << std::endl;

// null かどうか調べる
if (json.GetObject()[“met”].IsNull())   std::cout << “null” << std::endl;
```

出力結果

```
null
```

null の Json オブジェクトが追加されるため，上の例の json.GetObject()[“met”].GetString()のように存在しない name から GetString()を呼び出してもエラーとはならない．しかし，次のように name が存在するかどうかをあらかじめ調べてからアクセスすることを推奨する.

```
// “met”か存在するかどうか調べる
if (json.GetObject().count(“met”) > 0)   std::cout << “exist” << std::endl;
```

上の例では map クラスの count により name の数を調べているが，find でイテレータを取得し，name の存在を調べても良い.

```
// “met”か存在するかどうか調べる
if (json.GetObject().find(“met”) != json.GetObject().end())   std::cout << “exist” << std::endl;
```

しかし，1 度でも存在しない name でアクセスすると null の Json オブジェクトが追加されるため，その後では上記の方法で調べることができない.
null の Json オブジェクトが追加されることを禁止するには，json 変数に const を付ければ良い.

```
// パースして Json オブジェクトを取得
const adlib::Json& json = parser.Parse(buf);
// “method”の値を string として取得
auto sMethod = json.GetObject().at(“method”).GetString();
// “met”の値を string として取得(失敗すると map から例外 std::out_of_range が投げられる)
auto sMet = json.GetObject().at(“met”).GetString();
```

こうすれば，少なくとも json 変数には暗黙に null の Json オブジェクトが追加されることはない.
ただし，map の[]演算子が使用できないため，上の例のように at 関数でアクセスする必要がある.
この場合，存在しない name でアクセスすると例外 std::out_of_range が投げられる.

5.2.3. エラーメッセージ

JsonParser クラスの Parse 関数が JSON のパースに失敗した場合，null の Json オブジェクトを返す．その際のエラー内容は，JsonParser クラスの ErrorMessage 関数により，エラーメッセージの入ったコンテナ(std::vector<std::string>型)を取得することで分かる．このコンテナには，前回の Parse 関数の結果が格納される．エラーが発生していなければサイズが 0 となる(vector の size 関数で確認できる)．エラーメッセージの内容は次の通りである．

エラーメッセージ	エラー内容
Parse Failure.	JSON のパース失敗(詳細不明)
There is '0' Prefixial Number.	0 接頭の number がある
There is Bad Word in Number.	number に無効な文字が含まれている
"***" is out of Integer Range.	number(整数)が int 型の範囲外
"***" is out of Double Range.	number(浮動小数点数)が double 型の範囲外
There is not '0'~'9' Follower of '.' in Number.	number の小数点の後ろに数値がない
There is not '0'~'9' Follower of 'e' or 'E' in Number.	number の指数表記'e', 'E'の後ろに数値がない
There is not '"' end of "***".	文字列を閉じる'"'がない
Escape Character of "***" is Failure.	不正なエスケープシーケンスがある
There is code in the range U+D800 ~ U+DFFF (surrogate).	エスケープシーケンスでサロゲート領域の UNICODE が指定されている (本ライブラリは UTF-8 に従うため，サロゲート領域は禁止)
There is not '}' in object.	object を閉じる'}'がない
There is not ',' in object.	object 内の 2 つ目以降の name/value ペアの前に","がない
There is not '"' in object.	object に文字列の開始を表す'"'がない
There is not ':' in object.	object に":"がない
There is not ']' in array.	array を閉じる']'がない
There is not ',' in array.	array 内の 2 つ目以降の値の前に','がない
Value "***" is Failure.	不正な値がある

同様に，WarningMessage 関数によりワーニングメッセージの入ったコンテナを取得できる．ワーニングは注意すべき事がある場合に発生し，エラーさえ起きていなければ正常に Json オブジェクトを返す．
ワーニングメッセージの内容は次の通りである．

ワーニングメッセージ	ワーニング内容
Name "***" Repeated, and Later Name is Given Priority.	object の name が重複している

「Key "***" Repeated, and Later Key is Given Priority.」は，object 内の name に重複があった場合に発生するワーニングメッセージである．JSON の仕様(RFC4627)において，name は unique であるべきとされている．しかし，重複した場合も，最後の name/value ペアを優先することにして，そのまま処理を続行するソフトウェアは少なくない．本ライブラリでもその方針ではあるが，このワーニングによって重複を検知することができる．また，JsonParser クラスの IsNameRepeated 関数は，前回の Parse で name が重複していれば true を返す．

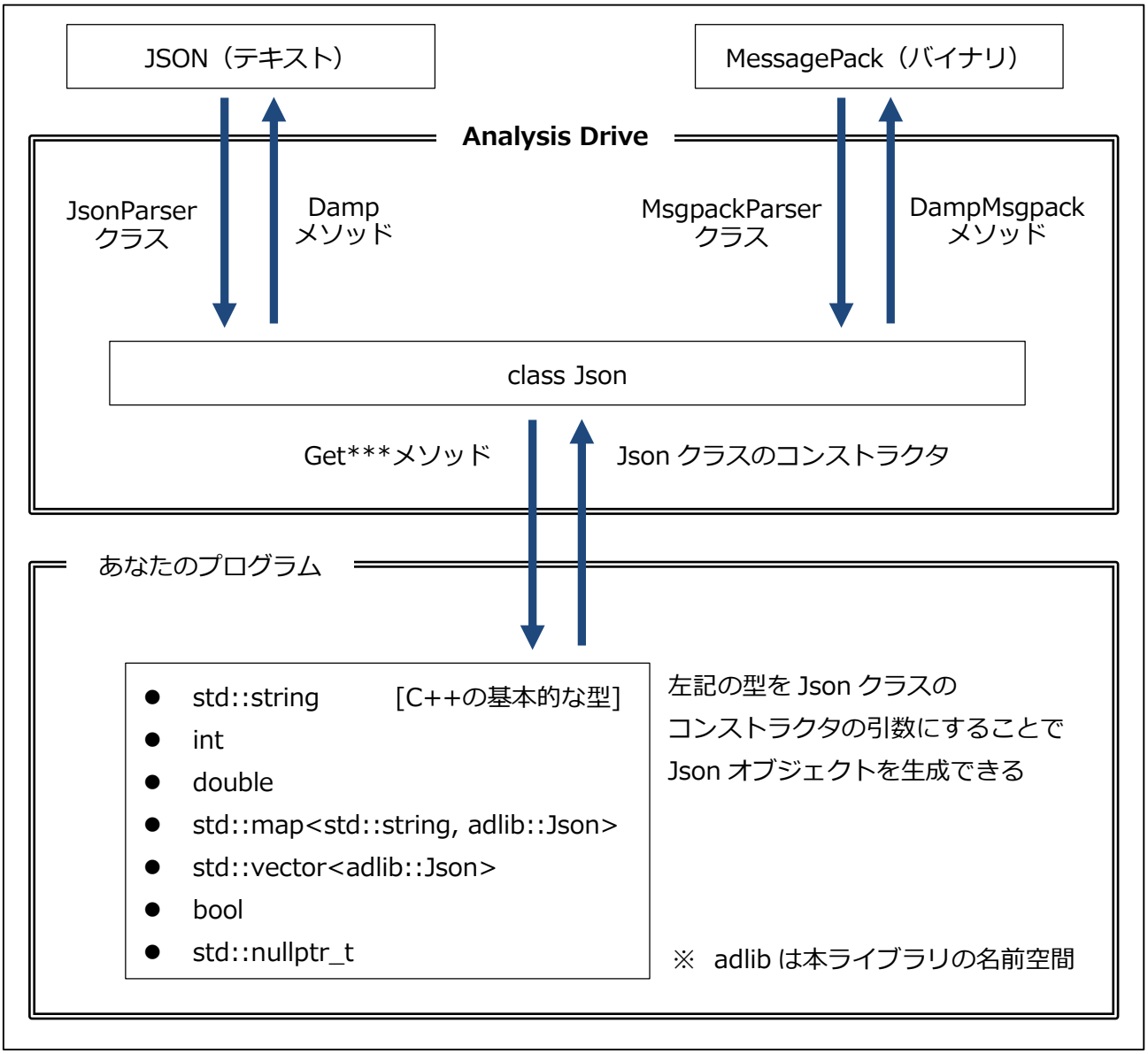
5.2.4. その他の機能

その他に次の機能がある．

- Json クラスの Damp 関数を使用すれば，引数のストリームに JSON 形式の文字列を出力できる．
- Json クラスの DampMsgpack 関数を使用すれば，引数のストリームに MessagePack 形式のバイナリを出力できる．

5.2.5. Json オブジェクト作成方法のまとめ

JSON と Json クラスとの関係を下図に示す。JsonParser クラスで Json オブジェクトを作成すれば、Get***メソッドを通して C++の基本的な型としてあなたのプログラムで利用できる。逆に、Json クラスのコンストラクタの引数に C++の基本的な型を入力すれば、Json オブジェクトを作成できる（下図参照）。



5.3. MessagePack のパース

5.3.1. 概要

MessagePack とは JSON ライクなバイナリフォーマットである(http://msgpack.org/)。本ライブラリの MessageParser クラスを使用すると，MessagePack 形式(Version 0.5.9)のバイナリをパースし，Json オブジェクトを作成できる(Json オブジェクトについては 5.2 節を参照)．

5.3.2. 使用方法

次の MsgpackParser クラスを使用すれば，MessagePack 形式のバイナリをパースすることができる．

class adlib::MsgpackParser　(MessagePack 形式のバイナリをパースするクラス)			
メンバ関数	内容	引数	返り値
Parse	MessagePack 形式のバイナリをパースする	パースするバイナリ(文字列)	Json オブジェクト
ErrorMessage	エラーメッセージを取得する	なし‘	エラーメッセージが格納されたコンテナ(std::vector)
WarningMessage	ワーニングメッセージを取得する	なし‘	ワーニングメッセージが格納されたコンテナ(std::vector)
IsNameRepeated	前回の Parse で Map の name が重複していたか確認	なし	前回の Parse で name が重複していれば true を返す

JSON をパースする JsonParser クラスと同様のインターフェイスであり，使用方法も同様である(JsonParser クラスについては 5.2 節を参照)．MessagePack と取得できる JSON(Json オブジェクト)の型の対応は次の通りである。

MessagePack の型	JSON(Json オブジェクト)の型
Nil	null
Boolean	bool
Integer	number(整数)
Float	number(浮動小数点数)
String	string
Array	array
Map	object
Binary	未対応(null)
Extension	未対応(null)

Binary 型など未対応の型が検出された場合は，代わりに null が格納される．

5.3.3. エラーメッセージ

MsgpackParser クラスの Parse 関数が MessagePack のパースに失敗した場合，null の Json オブジェクトを返す．その際のエラー内容は，MsgpackParser クラスの ErrorMessage 関数により，エラーメッセージの入ったコンテナ(std::vector<std::string>型)を取得することで分かる．このコンテナには，前回の Parse 関数の結果が格納される．エラーが発生していなければサイズが 0 となる(vector の size 関数で確認できる)．エラーメッセージの内容は次の通りである．

エラーメッセージ	エラー内容
Unknown Error.	予期せぬエラー(主に Msgpack ライブラリ関連)

同様に，WarningMessage 関数によりワーニングメッセージの入ったコンテナを取得できる．ワーニングは注意すべき事がある場合に発生し，エラーさえ起きていなければ正常に Json オブジェクトを返す．
ワーニングメッセージの内容は次の通りである．

ワーニングメッセージ	ワーニング内容
Name "***" Repeated, and Later Name is Given Priority.	Map の name が重複している
Binary Type is not Supported, and null is Given Value.	Binary 型(未対応)が検出された
Extension Type is not Supported, and null is Given Value.	Extension 型(未対応)が検出された
Unknown Type is Detected, and null is Given Value.	未対応の型が検出された

「Key "***" Repeated, and Later Key is Given Priority.」は，Map 内の name に重複があった場合に発生するワーニングメッセージである．MsgpackParser クラスの IsNameRepeated 関数は，前回の Parse で name が重複していれば true を返す．

6. リリースノート

Version	修正項目	追加機能
1.0.0	なし	■ コマンドラインパーサー ■ JSON パーサー
1.1.0	■ JSON クラスに JSON 形式でのダンプ機能を追加 ■ JSON クラスに MessagePack 形式でのダンプ機能を追加	■ MessagePack パーサー
1.1.1	■ デバッグ情報を生成しないように修正(WARNING が出てしまうため)	なし
1.1.2	■ CommandLineParser で接頭文字なしの引数がない場合にサイズ 0 のコンテナが追加されてしまうバグを修正	