

# *Gsharp WE*

## ユーザズ・ガイド

---

株式会社ケイ・ジー・ティー



---

# 目次

---

第1章 はじめに	1-1
第2章 基本的な使い方	2-1
2.1 GSL と HTML	2-2
2.1.1 セットアップ後に見ることができるサンプル	2-2
2.1.2 ファイル I/O	2-3
GsharpWE スクリプトを直接起動できる場合	2-3
shell を使って GsharpWE バイナリを起動する方法	2-4
2.1.3 GSL を利用した CGI プログラミング	2-5
データ入力 (HTML ファイル)	2-5
実行される CGI プログラム	2-5
2.1.4 対話的なグラフの送信	2-7
グラフ作成	2-7
イメージの作成	2-9
2.1.5 サンプル・ファイルとイメージの管理方法	2-10
イメージの作成用テンポラリ・ディレクトリ	2-10
イメージ・ファイル名	2-11
2.2 テンポラリ・イメージの削除	2-12
2.2.1 日付を利用した削除	2-12
イメージ・ファイル名	2-12
イメージの削除	2-12
2.3 データ・ファイルの読み込み	2-14
データ・ファイルの指定 (FORM)	2-14
ディレクトリの指定	2-14
ファイル指定	2-14
2.4 その他	2-15
日本語の扱い	2-15
カラムデータの読み込み	2-15
GsharpWE でイメージ生成だけを行うサンプル	2-15
第3章 その他の機能	3-1
3.1 イメージマップの作成	3-2
3.1.1 イメージ・マップ	3-2
3.1.2 Gsharp でイメージマップを作成する方法	3-2
3.1.3 ポリゴン領域の入手	3-3
3.2 ポストスクリプト出力	3-5
3.3 Java アプレットの作成	3-7
3.3.1 Java ドライバ	3-7
3.3.2 簡単な Java アプレット	3-7
3.3.3 GSL から Java を記述する方法	3-8
第4章 デバッグについて	4-1
FORM データの扱い	4-1
Gsharp とスクリプトのデバッグ	4-1
標準出力メッセージ	4-2

<b>第5章 関数について</b>	<b>5-1</b>
<b>5.1 組み込み関数について</b>	<b>5-2</b>
getpid().....	5-2
tmpnam().....	5-2
bounding_polygon().....	5-3
<b>5.2 libhtml GSL 関数</b>	<b>5-5</b>
set_image(float npx, float npy, string filename).....	5-5
make_image(float npx, float npy, string filename).....	5-5
HTML*().....	5-5
<b>5.3 libjava GSL 関数</b>	<b>5-7</b>
<b>5.4 イメージフォーマットの選択</b>	<b>5-8</b>
5.4.1 デバイスの選択	5-8
5.4.2 デバイスの指定	5-8
5.4.3 イメージ・サイズ	5-8
5.4.4 GIF イメージ	5-9
5.4.5 JPEG イメージ	5-9
5.4.6 PNG イメージ	5-9
5.4.7 Java への出力	5-10
<b>第6章 日本語を扱うには</b>	<b>6-1</b>
<b>6.1 日本語データベースへの変換</b>	<b>6-2</b>
関数の利用方法.....	6-2
<b>6.2 日本語ファイルを利用する方法</b>	<b>6-3</b>
<b>第7章 Gsharp と GsharpWE</b>	<b>7-1</b>
<b>7.1 チュートリアル概要</b>	<b>7-2</b>
インターフェース.....	7-2
出力結果.....	7-2
データ・ファイル.....	7-3
<b>7.2 Gsharp 上での操作</b>	<b>7-4</b>
7.2.1 Gsharp の起動	7-4
7.2.2 データ読み込みスクリプトの作成	7-4
7.2.3 グラフの作成とスクリプトの保存	7-6
7.2.4 スクリプトの編集	7-8
7.2.5 スクリプトの再実行	7-9
<b>7.3 Gsharp Web Edition への展開</b>	<b>7-11</b>
7.3.1 FORM を持ったページの作成	7-11
7.3.2 Gsharp WE による CGI スクリプトの作成	7-11
イメージの作成.....	7-12
イメージのキャッシュに関する回避と Web 上での参照.....	7-14
HTML コードの出力.....	7-15
データ IO.....	7-16
メッセージのオフ.....	7-17

---

# 第1章 はじめに

---

本マニュアルは、GsharpWE について記述されています。

## Gsharp プロダクト

Gsharp プロダクトには、現在 Gsharp と Gsharp Web Edition の2つがあります。

Gsharp は、グラフ・コンター図を作成する対話的なデータ・ビジュアライゼーション・ツールです。対話的な GUI を用いて、グラフを作成できます。また、直接画面上をクリックすることによって、その属性（色や線種、移動、拡大縮小など）を変更できます。

また、独自のスクリプト言語 Gsharp Script Language (GSL) を持っていますので、その一連の操作のながれを履歴に残したり（操作の定型化）、if-else 文や while 文を利用した処理のながれをプログラミングすることが可能です。詳細については、Gsharp のマニュアルをご参照下さい。

Gsharp Web Edition（以後 GsharpWE）は、特に Web サーバー上でグラフ・コンター図を利用したページを簡単に作成するために開発されたプロダクトです。Gsharp のスクリプト言語を実行できるインターフェースを持たないプログラムになっています。Web 上で利用するために必要な関数や機能を含んでいます。また、ライセンスは複数のユーザーが同時にアクセスできる形態（CPU 固定、ユーザー数制限なし）でリリースされています。

Gsharp と GsharpWE を用いることで、簡単に Web をベースにしたインフォメーション・ビジュアライゼーションを行なうページの作成、ブラウザで入力された要求に応じたグラフの作成などが可能となります。



---

## 第 2 章 基本的な使い方

---

この章では、Gsharp のスクリプト言語 (GSL) と Web Edition との関係について記述されています。

この章をご覧になる前に、まずは、GsharpWE のインストール、セットアップを行ってください。

この章では、セットアップ後に見ることができるサンプルについて説明しています。

注意 : *GsharpWE* のインストールやセットアップ方法については、*インストール・ガイド*、また `<inst_dir>/setup/setup.html` をご覧ください。

## 2.1 GSL と HTML

Gsharp スクリプト言語には、グラフの作成や属性の変更を行なう複数の関数が含まれています。その中にあるファイル I/O の関数を利用し、Web ページとのやりとりを行います。必要な HTTP ヘッダーや HTML を出力することができます。

この項では、GSL のスクリプト言語を用いたもっとも簡単な Web ページの作成方法について示します。また、対話的な Web のアプリケーションを作成するテクニックについて記述しています。

### 2.1.1 セットアップ後に見ることができるサンプル

まずは、インストール後に行ったセットアップが正常に終了していることを確認してください。以下のページにアクセスします。

`http://<machine>/gshwe/index.html`



上記のページに見ることができるサンプルを実行し、動作することを確認してください。UNIX 版ではほとんどのマシンで GsharpWE のスクリプトを直接実行することができません。Shell から GsharpWE を実行するサンプルのページをご利用ください。

以降の節では、このページで見ることができるサンプルの作成方法について説明しています。

## 2.1.2 ファイル I/O

まず最初に GSL で HTML を出力するには出力関数を利用して行います。GSL は以下の ANSI-C によく似たファイル I/O の関数を持っています。

```
fopen(file_name, mode)

fwrite(access_id, var1[, ... varN])

fclose(access_id)
```

これらの関数はテキストの標準出力やファイル出力に利用できます。出力ファイルとして、`stdout` のようなシステムの標準出力を行なう場合、`fopen` や `fclose` は必要ありません。システムの標準入出力へのアクセス `id` は以下の通りです。

```
stdin   = 0
stdout  = 1
stdmsg  = 2
```

GSL の I/O 関数は、HTML ページを作成するために利用することができます。以下に、簡単な例を示します。

例えば、以下のファイル (`hello.gsw`) を作成してみてください。

```
/*
 * write out HTML
 */
id = fopen("hello.html", "w");
fwrite(id, "<HTML>");
fwrite(id, "<TITLE>GsharpWE Greeting</TITLE>");
fwrite(id, "<FONT SIZE=+4><CENTER><B>");
fwrite(id, "Hello Wide World");
fwrite(id, "</B></CENTER></FONT><P>");
fwrite(id, "</BODY></HTML>");
fclose(id);
```

この GSL プログラムは `hello.html` という名前のファイルを開き、そのファイルに HTML コードを書き込むプログラムです。このファイルを実行するには、以下の方法で行います。

### GsharpWE スクリプトを直接起動できる場合

一部の UNIX マシンと Windows 版では、直接 GsharpWE スクリプトを起動することができます。

以下の方法でお試してください。

- 1 行目に GsharpWE の起動バイナリを指定します。  
`hello.gsw` を再度開き、以下のように 1 行目を追加してください。

注意: *UNIX 版のみ必要です。Windows 版では必要ありません。*  
また、`<inst_dir>/bin/GsharpWE` を `/bin/GsharpWE` にリンクしてから利用します。

```
#!/bin/GsharpWE
/*
 * write out HTML
 */
id = fopen("hello.html", "w");
fwrite(id, "<HTML>");
fwrite(id, "<TITLE>GsharpWE Greeting</TITLE>");
fwrite(id, "<FONT SIZE=+4><CENTER><B>");
fwrite(id, "Hello Wide World");
fwrite(id, "</B></CENTER></FONT><P>");
fwrite(id, "</BODY></HTML>");
fclose(id);
```

2. hello.gsw に実行権を与え（UNIX 版のみ）実行します。

```
chmod +x hello.gsw
./hello.gsw
```

- 実行したディレクトリ下にファイル hello.html ができたことをご確認ください。UNIX 版をご利用で、実行できない場合は、次に示すシェルから GsharpWE を起動する方法で利用します。

## shell を使って GsharpWE バイナリを起動する方法

一部のマシンを除き、UNIX マシンでは GsharpWE のスクリプトを直接起動することができません。まずは、環境変数を設定してから、HTML コードを出力してみましょう。以下は csh の例です。

```
setenv UNIDIR <inst_dir>
setenv LM_LICENSE_FILE <inst_dir>/base/license.dat
setenv LANG C

<inst_dir>/bin/GsharpWE hello.gsw
```

実際には GsharpWE は Web 環境上で利用しますので、環境変数の設定を shell や perl で行って利用します。以下のように shell ファイルや perl プログラムなど他のプログラムを利用して実行します。

以下はシェル・ファイルの例 (hello.sh) です。

```
#!/bin/sh
UNIDIR=/usr/local/GsharpWE3.0
export UNIDIR
LM_LICENSE_FILE=/usr/local/GsharpWE3.0/base/license.dat
export LM_LICENSE_FILE
LANG=C
export LANG

$UNIDIR/bin/GsharpWE hello.gsw
```

上記の /usr/local/GsharpWE3.0 はインストール・ディレクトリです。また、hello.sh に実行権利を与えてください。(chmod +x hello.sh)

```
./hello.sh
```

- 実行したディレクトリ下にファイル hello.html ができたことをご確認ください。

### 2.1.3 GSL を利用した CGI プログラミング

前項では、GSL プログラムを利用した HTML ファイルの出力について説明しました。次のステップでは、Common Gateway Interface (CGI) にこの GSL を用いることでブラウザから直接 HTML を生成する方法について説明します。

「サンプル 1 : GsharpWE の動作確認」のページは、この HTML の出力を GSL から行っているものです。また、「サンプル 2 : パラメーターの受け渡し」のページは、入力された文字列を GSL に渡し、HTML コードとして送信しています。

このサンプル 1、サンプル 2 のページに対応したファイルは、それぞれ以下の通りです。

サンプルページ	HTML ファイル	CGI プログラム
サンプル 1 : GsharpWE の動作確認	gshw01.html	gs_hello01.gsw
	gshs01.html	gs_shell01.sh
	gshp01.html	gs_perl01.pl
サンプル 2 : データの受け渡し	gshw02.html	gs_hello02.gsw
	gshs02.html	gs_shell02.sh
	gshp02.html	gs_perl02.pl

HTML ファイルは <www\_top\_dir>/gshwe/gsl, shell, perl の各ディレクトリに、また CGI プログラムは <www\_cgi\_dir> (/cgi-bin/) ディレクトリ下にインストール後、コピーしたものです。

#### データ入力 (HTML ファイル)

データの入力は HTML の FORM を利用して行なっています。その入力を CGI を利用して GSL に受け渡します。GSL はこの FORM の入力をもとに Web のページを作成します。

最初に、FORM が必要です。以下に HTML の FORM を利用した簡単な入力例を示します。このページは、名前を入力するパラメータを持っています。

```
<form method="post" action="http://cgi-bin/gs_hello1.gsl">
<b>名前を入力して下さい: </b><input type="text" name="name">
<input type="submit" value="Enter">
</form>
```

HTML の FORM の属性にある action によって、この入力が行なわれた際に実行される CGI のプログラムを指定することができます。上記の記述では、起動される CGI プログラムは、GSL スクリプト gs\_hello1.gsl です。

注意: FORM については、HTML の書籍などをご参照ください。

#### 実行される CGI プログラム

「2.1.2 ファイル I/O」 - 2-3 ページで述べたスクリプトを少し改良することで CGI プログラムとして利用することができます。

前項の例では GSL プログラムは単に HTML ファイルを作成するだけでした。CGI として動作させるこの例では、HyperText Transfer Protocol (HTTP) のヘッダーを、まず出力しなければなりません。以下の行を最初に追加します。

```
fwrite(id, "Content-type: text/html");
fwrite(id, "");
```

注意: *Content-type* を出力する場合、2行目に必ず空白行が必要です。

次に、ファイルへの出力を直接 `stdout` 標準出力に行なうように変更します。 `fopen` と `fclose` 文は必要ありません。

```
id = 1;
```

最後に、FORM からの入力を利用できるように以下の部分を変更します。

```
fwrite(id, "Hello ", FORM_name);
```

GsharpWE は、HTML ファイルに記述した FORM の `method` が "GET" の場合、`QUERY_STRING` 環境変数から、また、`method` が "POST" の場合、標準入力から自動的に解釈し、FORM で指定された変数名を入手できるようになっています。HTML ページの FORM で指定された名前(上記例では `name`)に "FORM\_" の前頭子をつけて受け渡されます。FORM を記述した HTML の以下の行を再度確認してみてください。

```
</b><input type="text" name="name">
```

この `name` が受け渡されます。

この行では、FORM からの入力が "name" という名前で定義されています。GsharpWE は、この入力パラメータを標準入力から解釈し、`fwrite` 文で利用されている "FORM\_name" という名前で認識します。

以下に GSL プログラムを示します。

チェックポイント

```
#!/bin/GsharpWE
/*
 * Open file, write out HTML, close file
 */

id = 1;
fwrite(id, "Content-type: text/html");
fwrite(id, "");
fwrite(id, "<HTML><BODY BGCOLOR=#FFFFFF>");
fwrite(id, "<TITLE>GsharpWE Greeting</TITLE>");
fwrite(id, "<FONT SIZE=+4><CENTER><B>");
fwrite(id, "Hello ", FORM_name);
fwrite(id, "</B></CENTER></FONT><P>");
fwrite(id, "</BODY></HTML>");
```

CGI プログラムを実行するには、そのプログラムはサーバーで定義されている `cgi-bin` のディレクトリに置かなければなりません。

UNIX 版をご利用で Shell から起動する場合は、このスクリプトを GsharpWE バイナリの引数に与えて実行します。

「shell を使って GsharpWE バイナリを起動する方法」 - 2-4 ページで述べた `hello.sh` を再度ご覧下さい。Shell の中で環境変数を設定し、この GsharpWE バイナリを実行します。

## 2.1.4 対話的なグラフの送信

次に CGI (Common Gateway Interface) を通して、Web のブラウザから入力された情報をもとにグラフを作成するテクニックについて説明します。前項の CGI を利用したサンプルを応用し、さらに、グラフを作成するように編集します。

「サンプル 3：イメージ生成の基本」のページは、選択されたグラフ・タイプのグラフを作成し、イメージを作成、その結果を送信しています。

このサンプル 3 のページに対応したファイルは、それぞれ以下の通りです。

サンプルページ	HTML	CGI プログラム
サンプル 3：イメージ生成の基本	gshw03.html gshs03.html gshp03.html	gs_hello03.gsw gs_shell03.sh gs_perl03.pl

### グラフ作成

まず、グラフを作るための再利用できるコード、グラフのテンプレートを作成します。グラフを作成するスクリプトの書き方については Gsharp (対話型アプリケーション) でグラフを作成し、その状態を GSL に保存することで見ることができます。

まず HTML の FORM を拡張します。以下のコードには、グラフのタイプを選択するためのリストが追加されています。

```
<form method="post" action="http://cgi-bin/g_s_graph.gsl">
<b>Please type your name: </b><input type="text" size=20 name="name">
<b>Graph Type: </b><select name="graph">
  <option value="bar">Bar
  <option value="line">Line
  <option value="pie">Pie
</select>

<input type="submit" value="Enter">
</form>
```

次に、グラフのテンプレートが必要です。通常、予め Gsharp を利用して対話的にグラフの生成、GSL への保存を行ったものを利用します。

ここでは、まずは、Web とのやりとりを見るために、GSL 文の中でランダムな値を発生させ、そのグラフを作成する例をもとに説明します。

グラフを作るには、以下のようにビューポート、ドメイン、グラフを順に作成します。

```
create Viewport page_1.viewport_1 ;
create Domain page_1.viewport_1.domain_1 ;
create Graph page_1.viewport_1.domain_1.graph_1
  ( XuNyData = "rnd(10)"
  );
```

この GSL は、ランダムに発生させた 10 個のデータ (rnd(10)) に対してグラフを作成するものです。ただし、ここでは、まだ新しく設定した FORM のグラフタイプの値を利用していません。

FORM で指定されたパラメータ (例: name と FORM\_name) は自動的に受け渡されることを思い出して下さい。このケースでは、FORM の選択によって FORM\_graph というパラメータが作成されます。この情報は直接 GSL のグラフを作成する文に割り当てることができます。

```

create Graph page_1.viewport_1.domain_1.graph_1
  ( XuNyData = "rnd(10)",
    XuNgraphType = FORM_graph
  );

```

このように記述することで FORM で指定されたタイプを作成するグラフタイプに設定することができます。

すなわち、HTML のページ (FORM) で line が選択されると折れ線グラフ、bar が選択されると棒グラフ、pie が選択されると円グラフとしてグラフが作成されます。

GSL スクリプトでは、また、この FORM から入力されたパラメータを処理のロジックに利用することができます。例えば、グラフのレイアウトを以下のように選択されたパラメータに応じて変更できます。

```

if FORM_graph = "line" then
  set page_1.viewport_1.domain_1.graph_1
    ( XuSmoothingFactor = 9,
      XuNcolor = 2
    );
elif FORM_graph = "bar" then
  page_1.viewport_1.domain_1.graph_1.XuNcolor = 4;
endif

```

上記の例は、if-else 処理を用いて、ラインが選択された場合と他の場合の場合分けを行っています。

HTML への出力を行なった最初の CGI のサンプルと違い、今回の GSL はグラフを作成しなければなりません。この 2 つの処理 (HTML への出力とグラフの作成) を分けて考えるために、グラフのテンプレートを GSL の関数として定義することができます。以下のようにグラフ作成を makeGraph() といった関数で準備すると整理しやすいでしょう。

#### チェックポイント

```

function makeGraph()
  string type;
  type = "line";
  if exists(WORK.FORM_graph) type = FORM_graph;
  create Viewport page_1.viewport_1 ;
  create Domain page_1.viewport_1.domain_1 ;
  create Graph page_1.viewport_1.domain_1.graph_1
    ( XuNyData = "rnd(10)",
      XuNgraphType = type
    );
  if type = "line" then
    set page_1.viewport_1.domain_1.graph_1
      ( XuSmoothingFactor = 9,
        XuNcolor = 2
      );
  elif type = "bar" then
    page_1.viewport_1.domain_1.graph_1.XuNcolor = 4;
  endif
endfunction

```

上記の関数で、直接入力値を用いる代わりに、グラフタイプのデフォルト値として type = "line" を設定している点に注目して下さい。これは、FORM からの入力がない場合でも処理が行なえるようにコーディングしている方法の例です。FORM からの入力がなくともテストを行なうことができます。

## イメージの作成

グラフのテンプレートが作成できたら、次のステップは、Web で理解できるイメージにグラフを出力することです。GsharpWE では、よく用いられる HTTP や HTML のヘッダーの作成や、Web で理解できるイメージの生成などを簡単に行なうためのいくつかのマクロ化された関数が準備されています。libhtml と libjava という名前でライブラリ化されています。これらの関数の中から、イメージを作成する関数を利用できます。

GSL のライブラリにある関数を呼び出す前に、まず、ライブラリを読み込まなければいけません。その関数が参照される前に GSL スクリプトのメインの部分で、GSL の include 文によって行なわなければなりません。

```
include "$UNIDIR/lib/libhtml.gsl";
```

\$UNIDIR は、GsharpWE で知られている環境変数です。どこに GsharpWE がインストールされているかを認識します。libhtml ライブラリにある GSL 関数 make\_image を利用します。以下のように定義されています。

```
make_image(float xres, float yres, string filename)
```

イメージのフォーマットは filename の拡張子で判断されています。

```
".gif" - GIF image format
".jpg" - JPEG image format
".png" - PNG image format
".java" - Java source
```

注意: PNG は GIF イメージの代わりに WWW コンソーシアム (W3C) で新しく採用されたイメージフォーマットです。ブラウザのベンダーやバージョンによってはサポートされていない場合もあります。

この関数をコールすることによってグラフのイメージを生成することができます。

```
make_image(400, 300, "/usr/local/WWW/gshwe/img_tmp/hello2.gif");
```

出力イメージとしてある固定された名前を利用すると、繰り返し実行された場合や同時にアクセスされた場合の問題が生じます。例えば、通常ブラウザでは、表示されたイメージはキャッシュされています。そのため、新しくイメージを生成しても、古いイメージが表示されてしまいます。また、複数のユーザーが同時アクセスするとイメージファイルを作成できないケースがあります。

このような現象を解決するために、UNIX 版では getpid() 関数を用いてユニークな名前を作ることができます。

```
image = "/gshwe/img_tmp/hello2"+getpid()+".gif";
```

注意: getpid() 関数は、プロセス ID を入手する GSL の関数です。UNIX 版では、ユニークな名前をつけるために利用することができます。Windows 版では、この ID は取ることができません。ランダム値などを利用してユニークな名前をつけて下さい。

また、Gsharp のスクリプトでは文字列は上記のように足し算することができません。

チェックポイント 以下にグラフ作成を行う GSL ファイルの例を示します。実際にサンプルで利用しているファイルについては次項で詳しく説明しています。

```
#!/bin/GsharpWE
```

```
function makeGraph()
  string type;
  type = "line";
  if exists(WORK.FORM_graph) type = FORM_graph;
  create Viewport page_1.viewport_1 ;
  create Domain page_1.viewport_1.domain_1 ;
  create Graph page_1.viewport_1.domain_1.graph_1
    ( XuNyData = "rnd(10)",
      XuNgraphType = type
    );
  if type = "line" then
    set page_1.viewport_1.domain_1.graph_1
      ( XuNsmoothingFactor = 9,
        XuNcolor = 2
      );
  elif type = "bar" then
    page_1.viewport_1.domain_1.graph_1.XuNcolor = 4;
  endif
endfunction

include "$UNIDIR/lib/libhtml.gsl";

string image; image = "/gshwe/img_tmp/hello2"+getpid()+".gif";

/*
 * create a graph and image
 */
makeGraph();
make_image(400, 300, "$UNIDIR/example/" + image);

/*
 * write out HTML
 */
id = 1;
fwrite(id, "Content-type: text/html");
fwrite(id, "");
fwrite(id, "<HTML><BODY BGCOLOR=#FFFFFF>");
fwrite(id, "<TITLE>GsharpWE Greeting</TITLE>");
fwrite(id, "<CENTER><FONT SIZE=+4><B>");
fwrite(id, "Hello ",FORM_name);
fwrite(id, "</B></FONT><P>");
fwrite(id, "<IMG SRC=¥ ",image, " ¥ ">");
fwrite(id, "</CENTER>");
fwrite(id, "</BODY></HTML>");
```

### 2.1.5 サンプル・ファイルとイメージの管理方法

もう少しイメージの管理方法について見てみましょう。  
サンプル3のページにある HTML ファイルと CGI プログラムの詳細について説明しています。

#### イメージの作成用テンポラリ・ディレクトリ

gs\_hello03.gsw をご覧ください。  
このファイルの最初の方でイメージを作成するテンポラリ・ディレクトリを扱っていません。

```

/* WEB 環境 */
IMGWEB = "/gshwe/img_tmp/";
if os_type() = "Windows" then
  IMGDIR = "c: ¥¥ inetpub ¥¥ wwwroot ¥¥ gshwe ¥¥ img_tmp ¥¥ ";
else
  IMGDIR = "/usr/local/WWW/gshwe/img_tmp/";
endif

```

- IMGWEB は WWW 上で見たディレクトリ名です。  
例えば、上記の例では (http://<machine>) /gshwe/img\_tmp/ としてアクセスできるディレクトリになっています。
- IMGDIR は実際のマシン上のディレクトリ名です。

2 つの変数を利用してマシン上、WWW 上のディレクトリ名を設定しています。

## イメージ・ファイル名

先述したように、キャッシュの問題や同時アクセスの問題を回避するために、イメージ名にはユニークな名前が必要です。

gs\_hello03.gsw ファイルの 46 行目以降をご参照ください。

```

seed(secsince(time));
tnumber = int(rnd(1)*100000);
if os_type() = "Windows" then
  string imgname; imgname = "hello"+ tnumber + ".gif";
else
  string imgname; imgname = "hello"+ getpid() + ".gif";
endif
string imgfile; imgfile = IMGWEB + imgname;
string imgfile2; imgfile2 = IMGDIR + imgname;

```

- imgname はイメージ・ファイルの名前です。Windows ではプロセス ID を取る関数でユニークな番号が決まらないため、tnumber というランダム値 (rnd 関数) を使っています。なお、seed 関数はランダム値のもとになる値を決める関数で念のため時間を使って同じ値が発生しないようにしています。
- imgfile は WWW のネットワーク上で見たときのファイル名です。(http://<machine>) /gshwe/img\_tmp/hello2363.gif といった名前が設定されます。
- imgfile2 はマシン上で見たときのファイル名です。/usr/local/WWW/gshwe/img\_tmp/hello2363.gif といった名前が設定されます。

GsharpWE のスクリプトでイメージ・ファイルを作る時には、imgfile2 の名前を使います。HTML で送信するときは imgfile の名前を送信します。

このサンプルでは、イメージの削除処理は行っていません。そのため、次々に新しくファイルが作成されますので、あるタイミングでファイルを削除することが必要です。

イメージの削除については、次項でさらに説明しています。

## 2.2 テンポラリ・イメージの削除

テンポラリ・イメージは、次々に作成されるため、あるタイミングで削除する必要があります。マシンのクーロンやデーモンなどを利用してファイルを削除してください。

### 2.2.1 日付を利用した削除

ここでは、このテンポラリ・イメージを GSL が起動された時に古いイメージを削除する方法のひとつを示します。

実際のサンプルは、以下のファイルにあります。

サンプルページ	HTML	CGI プログラム
サンプル4：古いイメージの削除例	gshw04.html gshs04.html gshp04.html	gs_hello04.gsw gs_shell04.sh gs_perl04.pl

### イメージ・ファイル名

まずは、gs\_hello04.gsw をご覧ください。  
このサンプルではイメージの名前付けを以下のように行っています。

```
seed(secsince(time));
tnumber = int(rnd(1)*100000);
if os_type() = "Windows" then
  string imgname; imgname = daysince(today) + "hello"+ tnumber + ".gif";
else
  string imgname; imgname = daysince(today) + "hello"+ getpid() + ".gif";
endif
string imgfile; imgfile = IMGWEB + imgname;
string imgfile2; imgfile2 = IMGDIR + imgname;
```

ファイル名の先頭に daysince 関数を使って日付情報を付加しています。  
このスクリプトが実行されると、145475hello3523.gif のように hello の前に数値が追加されます。

daysince 関数は、Gsharp の持つ内部的な日付のカウントで、today を引数に与えることで 1601-1-1 からの日付が返り値に渡されます。

### イメージの削除

gs\_hello04.gsw の最後の方をご覧ください。  
以下のようにイメージを削除します。

```
/*
 * 古いイメージを削除します。
 */
/**** よくご確認の上、ご利用ください ****
if os_type() = "Windows" then
  tday = daysince(today);
  setdir(IMGDIR);
  tempfile = tnumber + ".tmp";
  system("dir /B > " + tempfile);
  import_ascii(tempfile,,,,,"files","text");
  for i = 1 to sizex(files)
```

```
        oday = tovalue(files[i] - strstr(files[i], "hello"));
        if (oday < tday) then
            system("del " + files[i]);
        endif
    endfor
    system("del " + tempfile);
else
    tday = daysince(today);
    setdir(IMGDIR);
    tempfile = tnumber + ".tmp";
    system("ls > " + tempfile);
    import_ascii(tempfile,,,,,"files","text");
    for i = 1 to sizex(files)
        oday = tovalue(files[i] - strstr(files[i], "hello"));
        if (oday < tday) then
            system("rm -f " + files[i]);
        endif
    endfor
    system("rm -f " + tempfile);
endif
*****/
```

- tday は今日の日付のカウント数です。
- setdir コマンドはそのディレクトリに移動します。
- system コマンドを利用して Windows では dir、UNIX では ls コマンドを実行しています。実行後、その結果をファイルに保存しています。
- import\_ascii はデータの読み込み関数です。リストが保存されたファイルから hello 文字列を抜き出し、古い日付（カウント数）を oday にセットしています。
- tday より小さい oday（古い日付）のファイルは system コマンドを利用して削除しています。

この処理はデフォルトではコメントアウトされています。よくご確認の上、コメントをはずしテストしてみてください。

## 2.3 データ・ファイルの読み込み

次の例は、データ・ファイルの読み込みを組み込んだ例です。  
サンプルは、以下のファイルにあります。

サンプルページ	HTML	CGI プログラム
サンプル 5 : データファイルの読み込み	gshw05.html gshs05.html gshp05.html	gs_hello05.gsw gs_shell05.sh gs_perl05.pl

### データ・ファイルの指定 (FORM)

グラフィックタイプを指定した方法と同じ方法で 2 種のデータ・ファイルを選択しています。

```
<p> ファイル名を選択してください。 :
<SELECT NAME="filename">
  <OPTION VALUE="test1.dat"> 1997年のデータ
  <OPTION VALUE="test2.dat"> 1998年のデータ
</SELECT>
```

FORM\_filename で test1.dat か test2.dat のいずれかの名前が渡されます。

### ディレクトリの指定

ファイルの存在するディレクトリ名は、GSL スクリプトの中で定義されています。

```
/* WEB 環境 */
IMGWEB = "/gshwe/img_tmp/";
if os_type() = "Windows" then
  IMGDIR = "c: ¥¥ inetpub ¥¥ wwwroot ¥¥ gshwe ¥¥ img_tmp ¥¥ ";
  DATADIR = "c: ¥¥ inetpub ¥¥ cgi-bin ¥¥ gs_data ¥¥ ";
else
  IMGDIR = "/usr/local/WWW/gshwe/img_tmp/";
  DATADIR = "/usr/local/WWW/cgi-bin/gs_data/";
endif
```

### ファイル指定

ファイル名は、以下のように文字列の足し算で指定しています。

```
DFILE = DATADIR+FORM_filename
```

GSL では、文字列の変数も直接 + 記号を使って足し算することができます。  
DFILE に /usr/local/WWW/cgi-bin/gs\_data/test2.dat という文字列が設定されます。

この DFILE 変数名をデータファイルの読み込み関数の引数に与えて利用します。

```
import_report(DFILE);
```

## 2.4 その他

サンプル6、サンプル7のページについて説明しています。

### 日本語の扱い

Gsharp では日本語はコードで扱われています。

次の例は、ファイルからデータと日本語の情報を読み込みグラフ処理を行った例です。サンプルは、以下のファイルにあります。

サンプルページ	HTML	CGIプログラム
サンプル6：日本語の使用例	gshw06.html gshs06.html gshp06.html	gs_hello06.gsw gs_shell06.sh gs_perl06.pl

日本語をコードに変換するには、convjp 関数を利用して行います。convjp 関数やコードについては、Gsharp のマニュアルをご参照ください。

### カラムデータの読み込み

Gsharp では CSV データを直接読み込むことはできません。

注意：*Gsharp for Windows 版には CSV データを読み込む関数が組み込まれていますが、その組み込み関数を Gsharp for UNIX や GsharpWE では利用できません。*

次の例は、CSV 形式のデータを読み込みグラフ処理を行った例です。サンプルは、以下のファイルにあります。

サンプルページ	HTML	CGIプログラム
サンプル7：カラムデータの読み込み例	gshw07.html gshs07.html gshp07.html	gs_hello07.gsw gs_shell07.sh gs_perl07.pl

このサンプルでは、1行を文字列として読み込み、ループをまわしながらカンマを取り除いています。

詳細は gs\_hello07.gsw 内のコメントをご参照ください。

### GsharpWE でイメージ生成だけを行うサンプル

Perl を利用したサンプル・ページにあるサンプル8は GsharpWE をイメージ生成だけに利用した例です。

サンプルは、以下のファイルにあります。

サンプルページ	HTML	CGIプログラム
サンプル8：GsharpWE でイメージ生成だけを行う例	gshp08.html	gs_perl08.pl

イメージファイル名などもすべて Perl 側で環境変数を利用して設定しています。

また、この例では、Perl の中でイメージファイルの日付を入手し、その 0.5 日前に作成されたファイルを削除する方法の例が記述されています（デフォルトではコメントアウトされています）。

テンポラリ・イメージを削除する一例としてご参照ください。

---

## 第 3 章      その他の機能

---

この章では、イメージ・マップなど、その他の機能について説明します。

## 3.1 イメージマップの作成

イメージマップは、ブラウザでクリックできるイメージの領域を作成するためのものです。設計された領域をクリックすると、URL で指定された部分、その他の html ページや cgi プログラムを参照することができます。

イメージマップは通常 HTML ページの tool bar などに用いられています。この機能を用いて、イメージからドリル・ダウンすることによって、さらに詳細な表示を表現するようなページの開発に用いることができます。ページ的设计者は、棒グラフの棒や、チャートのセグメントなどをイメージの要素として用いることができます。このテクニックは、インフォメーション・ドリル・ダウンなどに利用されています。

Gsharp は `bounding_polygon()` という関数を持っています。Gsharp で作成されたグラフ上にあるグラフィカルなオブジェクトの座標値を返すことができます。このポリゴンの領域の座標値を HTML のイメージマップとして利用できます。また、その要素を URL にリンクさせることができます。

### 3.1.1 イメージ・マップ

イメージマップは HTML の標準の機能です。USEMAP というイメージマップとして利用される名前をタグで指定します。以下に HTML の簡単なイメージマップ例を示します。

```
<IMG SRC="toolbar.gif" USEMAP=#mymap>
<MAP NAME="mymap">
  <AREA SHAPE=rect COORDS="0,0,115,33" HREF="support.html">
  <AREA SHAPE=circle COORDS="116,30,20" HREF="sales.html">
  <AREA SHAPE=polygon COORDS="217,0 247,33 277,0" HREF="info.html">
</MAP>
```

HTML のイメージマップでは、通常、イメージからの参照を上記のように定義しています。MAP は複数の AREA 要素を含んでいます。それぞれの AREA は長方形や円、ポリゴンで指定でき、イメージの領域内で定義されています。そこがクリックされた際の URL を含んでいます。また、利用できない領域が選択された場合の URL も DEFAULT 要素を用いて指定できます。

### 3.1.2 Gsharp でイメージマップを作成する方法

組み込まれている `building_polygon()` に加えて、GsharpWE にはイメージマップのための HTML の要素を作成する関数を含んでいます。以下の関数が `$/UNIDIR/lib/libhtml.gsl` (`$/UNIDIR` は GsharpWE が直接理解できる GsharpWE のインストールディレクトリです。) に含まれています。

```
HTMLimagemap(stream, image, mapname)
HTMLmaprect(stream, url, left, bottom, right, top)
HTMLmapcircle(stream, url, centre, radius)
HTMLmappoly(stream, url, x, y)
```

例えば、以下のように利用します。

```
include "$UNIDIR/lib/libhtml.gsl"

HTMLimagemap(stdout, "toolbar.gif", "mymap");
HTMLmaprect(stdout, "support.html", 0, 0, 115, 33);
HTMLmapcircle(stdout, "sales.html", (116, 30), 20);
```

```
HTMLmappoly(stdout, "info.html", 217//247//277, 0//33//0);
echo("</MAP>");
```

### 3.1.3 ポリゴン領域の入手

`bounding_polygon()` 関数は以下の引数を持っています。

```
float bounding_polygon(string object, float numvertices, float x, float y);
```

この関数の戻り値は、指定されたオブジェクトに含まれるポリゴン数です。numvertices はそれぞれのポリゴンの頂点数です。x と y はポリゴンデータの配列です。

注意: 棒グラフや チャートのポリゴンを入手する場合、ポリゴンの順番とデータの順番が逆になります。 - 最初のポリゴンは最後の棒やセグメントになります。また、最後のポリゴンが最初のデータになります。

注意: *GSL* の *repaint* コマンド (描画コマンド) が `bounding_polygon()` の前にコールされていなければなりません。

注意: 座標値はスクリーンサイズに依存します。 - イメージと一緒に利用するには、ピクセル値に変換しなければなりません。

便利な関数として、`HTMLbounding_polygon()` が `$UNIDIR/lib/libhtml.gsl` で提供されています。repaint を行ない、`bounding_polygon()` 関数を実行します。また、座標値をピクセル値に変換します。

`HTMLbounding_polygon()` は、`bounding_polygon()` とほぼ同じ引数を持っています。しかし、2つの `imagewidth` と `imageheight` が追加されています。以下のコードは、`HTMLbounding_polygon()` を利用して、作成された棒グラフからイメージマップを作成するサンプルです。

このファイルは `<inst_dir>/example/cgi-bin/gs_bars.gsw` です。

```
include "$UNIDIR/lib/libhtml.gsl";
float xpixs; xpixs = 400;
float ypixs; ypixs = 300;
float nvert, x, y;

create Viewport page_1.view;
create Domain page_1.view.domain;
create Graph page_1.view.domain.graph
  ( XuNgraphType = "bar",
    XuNyData = "log(1:4)+1"
  );

f = fopen("bars.html", "w");

HTMLheader(f, "Drillable Bar Chart", "Anon");
HTMLimagemap(f, "bars.gif", "bars");

npoly = HTMLbounding_polygon("page_1.view.domain.graph",
                             nvert, x, y, xpixs, ypixs);
offset = 1; #index into coordinate array

for i = 1 to npoly
  lines = offset:(offset+nvert[i]-1);
  # lines=(1:5), (6:10), ... for i = 1, 2, ...
```

```
offset = offset+invert[i];
X = slicex(x,lines);
Y = slicex(y,lines);
url = "http://cgi-bin/drill.cgi?bar="+npoly-i+1);
HTMLmappoly(f, url, X, Y);
endfor

fwrite(f,"</MAP>");

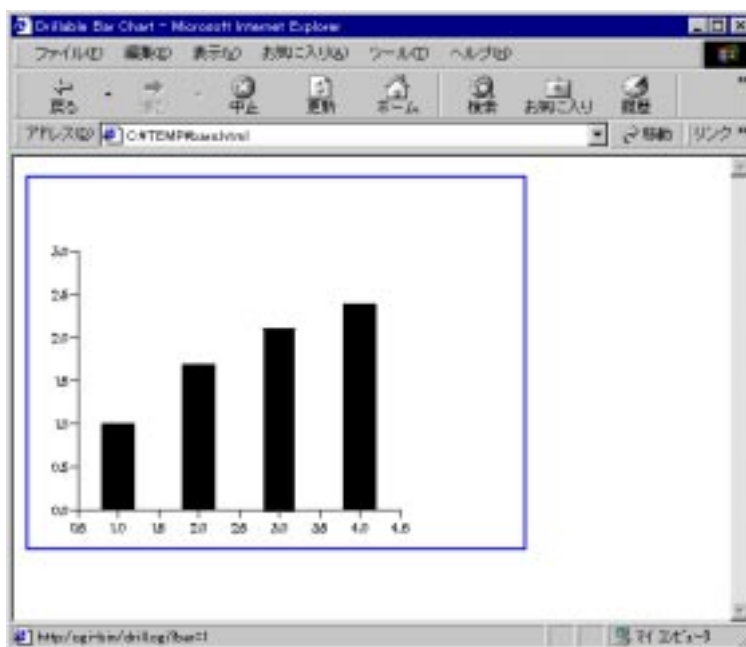
make_image(400,300,"bars.gif");
```

上記スクリプトファイル ( test.gsw ) を作成し、以下のように GsharpWE を実行してみてください。

```
<inst_dir>/bin/GsharpWE gs_bars.gsw
```

起動ディレクトリに bars.html と bars.gif ファイルが作成されます。この bars.html ファイルを読み込むと、以下のように各棒グラフをクリックできるクリックブル・イメージ・マップが作成できます。

以下は表示例です。

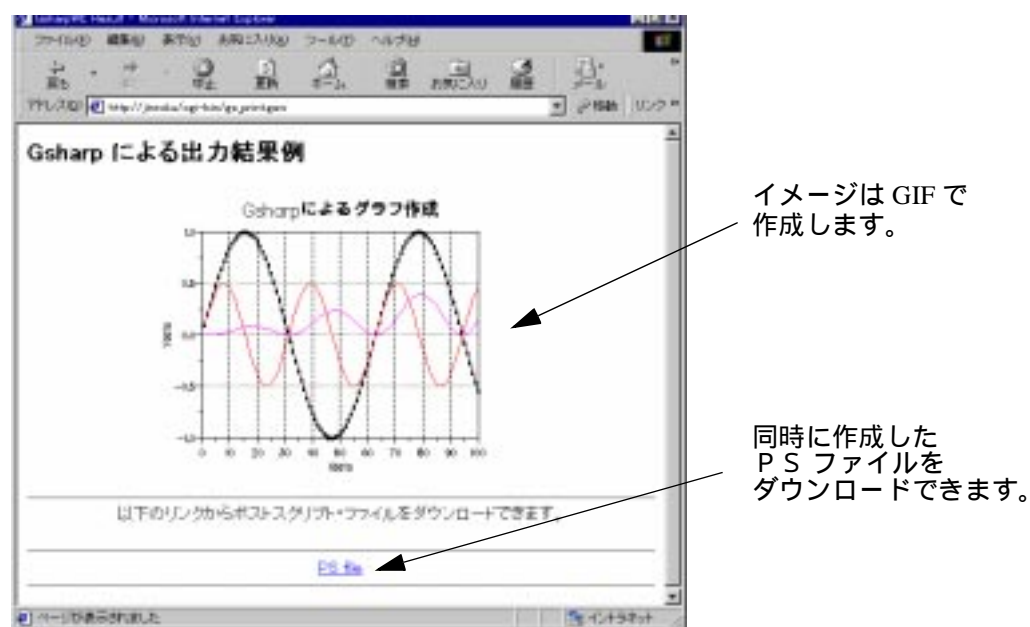


## 3.2 ポストスクリプト出力

Gsharp はイメージ出力の他、ポストスクリプト・ファイルへの出力を行うことができます。この機能を用いて、画面上では GIF イメージを表示し、同時にポストスクリプト・ファイルをダウンロードする仕組みを作ることができます。

サンプルは、以下のファイルにあります。

サンプルページ	HTML	CGI プログラム
ポストスクリプト出力	outps/index.html	gs_print.gsw gs_print.sh gs_print.pl



このサンプルで利用している gs\_print.gsw をご覧下さい。

1. まず、FORM で選択された出力ポストスクリプトのタイプに応じて使用するデバイス名を決定しています。

```
if FORM_dtype = "PSWB" then
  dname = "hposta4";
elif FORM_dtype = "PSC" then
  dname = "hcposta4";
elif FORM_dtype = "EPS" then
  dname = "hcposteps";
endif
```

2. 次に GIF ファイルと同様、出力するファイル名をプロセス ID やランダム値を使って指定しています。

```
if os_type() = "Windows" then
  psname = "gs_ps" + tnumber + ".dat";
else
  psname = "gs_ps" + getpid() + ".dat";
endif
```

```
psfile = IMGWEB + psname;  
psfile2 = IMGDIR + psname;
```

- 最後にポストスクリプトへの出力を行います。  
ページのサイズを a4 に設定します。  
hardcopy オブジェクトに出力ファイル名やデバイス名を設定します。

```
set page_1  
  ( XuNformat = "a4"  
  );  
set hardcopy  
  ( XuNdevice = dname,  
    XuNplacement = "center",  
    XuNplotCommandEnabled = false,  
    XuNplotFileName = psfile2  
  );
```

```
repaint;  
print;
```

repaint コマンドで再描画を行い、print コマンドでプリントを実行します。

- この別途作成した PS ファイルをダウンロードできるように HTML ファイルへリンクします。

```
fwrite(id, "<HR>");  
fwrite(id, "<a href=¥"",psfile," ¥">PS file</a>");  
fwrite(id, "<HR>");
```

## 3.3 Java アプレットの作成

### 3.3.1 Java ドライバ

GsharpWE はグラフを Java の draw メソッドに変換する Java ドライバを持っています。このメソッドを作成するには、Java ドライバを選択し、ファイル名を設定します。それから、グラフをプリントします。GSL では以下のように記述します。

```
hardcopy.XuNdevice = "ljava";
hardcopy.XuNplotFileName = "gsharp.java";
print;
```

プロット・サイズは print が呼ばれた際のキャンバスのサイズに依存します。ページサイズは mm で与えられています。Java ドライバは 1mm を 0.3pixel で割り当てます。ページサイズは以下の GSL 文を利用して設定できます。

```
page_1.XuNsize = (400*.3,300*.3);
```

この処理を簡単に行なうために、便利な関数として make\_image() が \$UNIDIR/lib/libhtml.gsl に提供されています。GSL 文では、以下のように記述できます。

```
make_image(400,300,"gsharp.java");
```

make\_image() はファイル名の拡張子 .gif .jpg .png や .java によって Gsharp のドライバを判断します。

例えば、以下のファイル (test.gsl) を作成してみてください。

このコードは、Gsharp の example のひとつである flight.gsl スクリプトを実行し、その結果を Java のコードに出力します。

```
#!/bin/GsharpWE
include "$UNIDIR/lib/libhtml.gsl";
include "$UNIDIR/lib/libjava.gsl";

exec("$UNIDIR/example/Gsharp/flight.gsl");
repaint;
appname = "graph";

make_image(400,300,appname + ".java");
```

このコードを実行すると、そのディレクトリに graph.java ファイルが作成 されます。

### 3.3.2 簡単な Java アプレット

上記の方法で GsharpWE を用いて Java の draw のクラスが作成されている場合、以下のアプレット gsharpApp.java を作成することで、draw メソッドを見ることができます。

例えば、先ほど作成した graph.java を見るには、以下のコード(ファイル graphApp.java) を vi などのエディタを利用して作成します。

```
import java.applet.*;
import java.awt.*;
```

```
public class graphApp extends Applet {
    private static Image i = null;
    public void paint(Graphics g) {
        if (i == null) {
            i = createImage(size().width,size().height);
            graph.draw(this, i.getGraphics());
        }
        g.drawImage(i,0,0,this);
    }
}
```

上記のコード中の "graph"(graph.draw) は、GSL 文 ( 上の例では test.gsl ) の make\_image(400,300,"graph.java") にある出力ファイル名に一致していなければなりません。

このファイルを gsharpApp.java という名前で保存して下さい。ファイル名は、自由につけることができますが、ファイル中に定義されているクラス名 ( 上記例にある 3 行目の名前 ) に一致していなければなりません。これで、このディレクトリには、2 つの java ファイルが存在します。GsharpWE によって出力された gsharp.java と新たに作成した gsharpApp.java です。

Gsharp の Java ドライバは、ディレクトリ <destination>/GsharpWE/lib/mjava にあるいくつかのユーティリティ・クラスを必要とします。アプレットを実行する前に、このディレクトリにリンクを作成しなければなりません。

```
In -s <destination>/GsharpWE/lib/mjava mjava
```

これで、Java コンパイラを用いて Java アプレットをコンパイルできます。

```
javac gsharpApp.java
```

コンパイルを行なうと、それぞれの .java ファイルに対する .class ファイルが作成されます。この例では、graph.class と graphApp.class です。

作成されたアプレットは、以下のような HTML ファイルを用いて読み込むことができます。

```
<HTML>
<TITLE>My First Java Applet from Gsharp</title>
<APPLET CODE="graphApp.class" width=400 height=300>
</APPLET>
</HTML>
```

この HTML ファイルはブラウザで表示することができます。もしくは Java コンパイラと一緒に提供されている appletviewer で確認できます。

### 3.3.3 GSL から Java を記述する方法

また、Java コードを Gsharp から直接出力することも可能です。GsharpWE に含まれている libjava.gsl には、java コードを出力するための関数がいくつか準備されています。

前のセッションで説明した Java ファイルは、以下の GSL スクリプトで記述できます。

```
#!/bin/GsharpWE

include "$UNIDIR/lib/libhtml.gsl";
include "$UNIDIR/lib/libjava.gsl";
```

---

```
# You can replace the following line with your own graph template code.
exec("$UNIDIR/example/Gsharp/flight.gsl");

appname = "graph";

#Create Graph output
make_image(400,300,appname+".java");

#Create viewer applet
f = fopen(appname+"App.java","w");

javaappleheaders(f);
javaawtheaders(f);
javastartappletclass(f,appname + "App");

fwrite(f,"private static Image i = null;");

javapaintmethod(f,"if (i == null) {"/
    " i = createImage(size().width,size().height);"/
    " "+appname+".draw(this,g);"/
    "}/"/
    "g.drawImage(i,0,0,this);");

javaendclass(f);

fclose(f);

#Create HTML file
f = fopen(appname+".html","w");

HTMLheader(f,"My First Java Applet from Gsharp","Anon");
HTMLapplet(f,appname+"App.class",400,300);
HTMLfooter(f);

fclose(f);

#Create *.class
system("javac "+appname+"App.java");
```

GSL を使って、新しいアプレットを作成すると同様に、Gsharp のデータセット、すなわち、読み込まれたデータやそのデータを組み込み関数などを利用して変更した新しいデータを、アプレットによって利用する static な値として出力することができます。javaexportfloat() と javaexportint() 関数が libjava.gsl に含まれています。どのような次元数の float / int 値も出力することができます。



---

## 第 4 章 デバッグについて

---

GSL スクリプトを `cgi-bin` で実行する場合、ブラウザでは、多くのケースではエラーメッセージを見ることができません。そのため、何が悪いかを知るのが困難なケースがよくあります。

また、よく見るメッセージとして "Document contains no data" といったものがあります。

これらの現象をチェックするには、まず、コマンドラインからスクリプトを実行してみてください。例えば、

```
cd /WWW/cgi-bin
./<your script>.cgi
```

Web ブラウザ上で、CGI によって実行されるのと同様の結果を得ることができます。

注意： 標準の *UNIRAS* や *Gsharp* による環境設定が行なわれていないことを確認して下さい。シェルの環境として *UNIDIR* が設定されていると *GsharpWE* はそのインストールされているディレクトリを見ることができません。

### FORM データの扱い

FORM を利用して入力を行なうスクリプトを作成している場合、入力値がない場合には、デフォルト値が扱われるように記述して下さい。そのようにすることで、コマンドライン から動作させることができます。例えば、

```
text = "Hello Wide World";
if exists(WORK.FORM_text) text = WORK.FORM_text;
Title.XuNtitleText = text;
```

この例では、FORM のデータが存在しない場合の処理を行なっています。デバックの際にコマンドラインから実行したり、データがない場合にも実行することができます。

便利な関数として、`HTMLget_value()` があります。`libhtml.gsl` にある関数で、FORM の値が存在する場合にはその値を、ない場合にはデフォルト値を返します。

```
HTMLget_value("text", "Hello Wide World");
```

もし、スクリプトで決められた値が入力されている場合しか実行しないのであれば、`HTMLget_value()` は、初期値の設定として利用できるでしょう。

### Gsharp とスクリプトのデバッグ

Gsharp は対話的に実行できるプロダクトです。Gsharp の持つスクリプト・ビルダというツールを利用して、作成することができます。グラフのテンプレートを作成するのに、簡単で効果的な方法です。もし、Gsharp をご利用であれば、スクリプトを GsharpWE と一緒に利用する前に、対話的にテストしてみてください。

もし、GsharpWE だけで利用されている関数を利用している場合、例えば `bounding_polygon()` や Java ドライバなどを利用している場合、以下のような記述に変更することで、スクリプトのテストを行なうことができます。

Gsharp の起動時に参照される `argv` 変数に GsharpWE を与えます。

```
if strstr(argv[1], "Gsharp") = "GsharpWE" then
    npoly=bounding_polygon(nvert,x,y);
endif
```

GsharpWE の場合のみ、この関数は実行されます。

また、Gsharp の組み込み関数である `batch()` を利用することができます。

```
if batch() then
    npoly=bounding_polygon(nvert,x,y);
endif
```

対話モードでは、この関数は呼ばれません。

## 標準出力メッセージ

Gsharp のスクリプト関数の多くはメッセージを出力します。

例えば、データの読み込み関数 (`import_report`, `import_ascii` など) を実行すると読み込んだデータの情報が出力されます。

作成した CGI プログラムによっては、これらのメッセージがクライアント側に出力されたり、またこれらのせいで正常に動作しないケースがあります。

以下の関数をスクリプトの最初に実行し、メッセージ出力をオフに設定してください。

```
set_messages(0,0,0,0);
```

この関数の引数はそれぞれ、エラー、ワーニング、情報、ユーザーメッセージを示しています。1, 0 でオン、オフを指定できます。

すべての引数に 0 を与えることで、いずれのメッセージも出力しないようになります。

---

## 第5章 関数について

---

この章では、関数ならびにイメージフォーマットについて記述されています。

---

## 5.1 組み込み関数について

この章では、GsharpWE (Gsharp Web Edition) でのみ利用されている組み込み関数について記述しています。Gsharp で利用されている関数に関しては、Gsharp のマニュアルをご参照下さい。

### getpid()

カテゴリ

String

シンタックス

```
getpid()
```

利用方法

Gsharp の String データとしてプロセス ID を返します。

利用例

```
pid = getpid();  
echo("pid = ",pid,", type = ",typeof(pid));
```

結果

```
pid = 27696, type = string
```

### tmpnam()

カテゴリ

String

シンタックス

```
tmpnam()
```

利用方法

テンポラリ領域のファイル名を返します。UNIX では、一般的には /tmp や /var/tmp です。NT では、C: や C:¥TMP などです。ファイルは作成されません。

利用例

```
tmp = tmpnam();  
echo("tmp = ",tmp,", type = ",typeof(tmp));
```

結果

```
tmp = /var/tmp/bcaa006kk, type = string
```

---

**bounding\_polygon()**

カテゴリ

Float

シンタックス

`bounding_polygon(string objname, float &nvert, float &xpoly, float &ypoly)`

利用方法

`bounding_polygon()` は、名前で指定されたオブジェクトのグラフィカルな要素を構成する境界を持ったポリゴン数を返します。  
`nvert` 配列は、そのオブジェクトに含まれるそれぞれのポリゴンに対する頂点数を示します。`xpoly` と `ypoly` は、そのポリゴンの境界を構成する座標値です。

利用例

```
function show_polygons(string obj)
  float nvert, xpoly, ypoly, i, j, n, offset;
  string str;
  n = bounding_polygon(obj, nvert, xpoly, ypoly);
  echo("");echo(obj," has ", n, " polygon",if(n=1,"","s"));
  if n = 0 return;
  offset = 0;
  for i = 1 to n
    echo(" Polygon no. ", i, " has ", nvert[i], " vertices");
    str = " ";
    for j = 1 to nvert[i]
      str = str + "(" + xpoly[j+offset] + "," + ypoly[j+offset] + ") ";
      if mod(j,3) = 0 then
        echo(str);
        str = " ";
      endif
    endfor
    offset = offset + nvert[i];
    echo(str);echo("");
  endfor
endfunction

function string branchof(string root)
  string branch, limbs, ptr;
  branch = root;
  limbs = branch + "." + childrenof($branch);
  if limbs branch + "."
    for node in limbs
      branch = branch // branchof(node);
    endfor
  return branch;
endfunction

function create_graph(string type)
  create Viewport page_1.viewport_1;
  create Title page_1.viewport_1.title_1
  ( XuNtitleText = type + "graph"
  );
  create Domain page_1.viewport_1.domain_1;
```

```
page_1.viewport_1.domain_1.legend.XuNobjectEnabled=true;
create Graph page_1.viewport_1.domain_1.graph_1
  ( XuNgraphType = type,
    XuNyData = "rnd(5)"
  );
endfunction

/*
* main program
*/
reset all;
graph_type = "line"//"pie"//"bar";
create_graph(graph_type[1]);
repaint; #repaint is necessary to set object sizes
hierarchy = branchof("page_1");
for object in hierarchy
  show_polygons(object);
endfor
```

#### 結果

```
page_1 has 0 polygons
page_1.viewport_1 has 1 polygon
Polygon no. 1 has 5 vertices
(44.8,35.84) (232,35.84) (232,197.12)
(44.8,197.12) (44.8,35.84)
...
```

## 5.2 libhtml GSL 関数

以下の関数が libhtml.gsl で定義されています。

### set\_image(float npx, float npy, string filename)

set\_image() は、ファイルの拡張子から指定されたサイズのイメージを作成するような場合に、そのキャンバスのサイズを設定します。

```
.gif  GIF
.jpg  JPEG
.png  PNG
.java Java
```

キャンバスのサイズに影響されるオブジェクト（例えば注釈）に対する HTMLbounding\_polygon() などを利用する場合には、set\_image() でサイズを設定して下さい。

### make\_image(float npx, float npy, string filename)

make\_image() 関数は、set\_image() をコールし、その後、print 描画コマンドを実行します。

```
float HTMLbounding_polygon
    (string object, float nvertices, float x, float y,
     float imagewidth, float imageheight)
```

HTMLbounding\_polygon は、組み込み関数である bounding\_polygon() を利用しやすくしたものです。自動的にピクセル座標に変換します。

この関数の利用方法に関する詳細については、イメージ・マップの作成を御参照下さい。

### HTML\*()

libhtml.gsl には複数の HTML コードを生成する便利な関数が含まれています。これらはすべて、Gsharp の fwrite() 関数を利用して文字列を出力するようになっています。

例えば、

```
fwrite(1,"<IMG SRC=\""+imgfile+"\">");
```

この関数は、

```
HTMLImage(stdout,imgfile);
```

で利用できます。HTML\*() 関数と fwrite() 関数は一緒に利用することができます。

これらの関数の第一引数は、( fopen() の戻り値である ) ファイルナンバーです。ファイルや標準出力に簡単に出力できます。ファイルへの出力はテストに便利です。また、標準出力は、CGI としてスクリプトを実行する場合に必要です。両方のケースに対し、1つのスクリプトで記述することができます。

```
stdout = 1;
if batch() then
  f = stdout;
else
  f = fopen("myfile.html", "w");
endif

HTMLheader(f, "Title", "Author");
HTMLheading(f, 1, "Hello Wide World");
HTMLfooter(f);

if not batch() fclose(f);
```

---

## 5.3 libjava GSL 関数

libjava.gsl は、Java コードへ出力するための関数です。どのように GSL から Java への出力を行なうことができるかの例として利用できます。GsharpWE の Java を利用した例のひとつとして作成されました。便利なコードや、ユーザ自身の開発を行なう基本関数として利用できる関数が含まれています。

java\*() 関数は、Gsharp の出力関数 fwrite() を利用しています。fwrite() と一緒に用いることができます。

以下の関数が含まれています。

```
function javaappleheaders(float i)
function javaawtheaders(float i)
function javastartappletclass(float i, string name)
function javastartclass(float i, string name)
function javaendclass(float i)
function javamethod(float i,string name, string method)
function javapaintmethod(float i,string method)
function javamousedragmethod(float i,string method)
function javaexportfloat(float i, string name, float dat)
function javaexportint(float i, string name, float dat)
function javainclude(float i, string includefile)
```

libjava 関数の利用例は、Java アプレットの作成を御参照下さい。

## 5.4 イメージフォーマットの選択

`make_image()` 関数は、`libhtml.gsl` で提供されています。もし、ユーザ自身のルーチンを組み込んだり、このデフォルト値を変更する場合、以下の記述を参照して下さい。

### 5.4.1 デバイスの選択

出力イメージのフォーマットは `hardcopy.XuNdevice` というハードコピーに対する属性を利用して `GSL` で設定されています。出力デバイスを指定する名前を設定しなければなりません。

```
hardcopy.XuNdevice = "ggifl8";
```

Web で利用できるリストは、後述のテーブルをご参照ください。

### 5.4.2 デバイスの指定

新たにデバイスを指定する場合には、`Gsharp` のスタートアップ・ファイル `$UNIDIR/base/Gsharp.gsl` で定義しなければなりません。以下のように `define_device()` という `GSL` の組み込み関数を利用して指定します。

```
define_device(string token, string description,  
              pagesize,x,y,orien,string default_filename);
```

GIF や JPEG のようなイメージのデバイスでは、`pagesize,x,y,orien` のパラメータは、`0,,,` に設定します。

イメージのデバイスとして "SJPG18 - RGB Intensity Device" を利用したいのなら、まず、`$UNIDIR/base/Gsharp.gsl` に以下の行を追加します。

```
define_device("sjpgi8","JPEG (RGB)",0,,,"gsharp.jpg");
```

その後、イメージを作成することができます。

```
hardcopy.XuNdevice = "sjpgi8";  
print;
```

インストールされているファイルをご参照ください。

### 5.4.3 イメージ・サイズ

イメージのサイズは、ページのサイズが基本に作成されます。それぞれのデバイスは、ページの `mm` に対する異なったピクセル数を持っています。それぞれのドライバは、ページの `1 mm` に対して `1 pixel` のデバイスを持っています。H, M や L の符合を持っていないデバイスがこれにあたります。デフォルトでは、このデバイスが用いられています。 `px x py` ピクセルの GIF イメージを作成するには、以下のコマンドで行ないます。

```
hardcopy.XuNdevice = "ggifl8";  
page_1.XuNsize = (px,py),  
print;
```

ファイル名 `hardcopy.XuNfilename` 属性は、イメージの名前を指定するのに用いられます。

```
hardcopy.XuNfilename = "myplot.gif";
```

#### 5.4.4 GIF イメージ

GIF は 256 カラーの 8bpp フォーマットです。以下のデバイスを指定できます。

GGIFL8	CMY Lookup Device 1 d.p.mm
GGIFL8H	CMY Lookup Device 300 d.p.i
GGIFL8M	CMY Lookup Device 150 d.p.i
GGIFL8L	CMY Lookup Device 75 d.p.i
SGIFL8	RGB Lookup Device 1 d.p.mm
SGIFL8H	RGB Lookup Device 300 d.p.i
SGIFL8M	RGB Lookup Device 150 d.p.i
SGIFL8L	RGB Lookup Device 75 d.p.i

GIF イメージフォーマットは AVS Inc. が UNISYS よりライセンスされている圧縮アルゴリズムを含んでいます。契約上、GsharpWE のユーザは、以下の条件で利用できます。

" この製品の一部分は、U.S. 契約番号 4558302 でライセンスされています。販売元よりライセンスされたパーソナル・コンピュータやワークステーションのみで利用できます。PDF や TIFF, GIF, PostScript などの機能は、ソフトウェアの保管やバックアップの目的以外によるコピーや修正、発表、再配布などは禁止されています。 "

#### 5.4.5 JPEG イメージ

JPEG は、高圧縮をサポートしている 24 bpp イメージフォーマットです。一部のデータが損失した圧縮が行われます。すなわち、ピクセルの損失が生じてしまいます。ピクセルの損失が少ないように、Gsharp のドライバでは高いイメージ・クオリティを利用していますが、ピクセルの損失がラインやテキストに生じるケースがあるでしょう。JPEG がベクトル・グラフィックスに適していないためです。PNG や GIF による出力の方が適しています。以下に JPEG のデバイスをリストします。

GJPGI8	CMY Intensity Device 1 d.p.mm
GJPGI8H	CMY Intensity Device 300 d.p.i
GJPGI8M	CMY Intensity Device 150 d.p.i
GJPGI8L	CMY Intensity Device 75 d.p.i
SJPGI8	RGB Intensity Device 1 d.p.mm
SJPGI8H	RGB Intensity Device 300 d.p.i
SJPGI8M	RGB Intensity Device 150 d.p.i
SJPGI8L	RGB Intensity Device 75 d.p.i

#### 5.4.6 PNG イメージ

PNG "Portable Network Graphics" は、Web のために設計されたイメージフォーマットです。PNG は 48 bpp の true カラーです。Gsharp では、256 カラーの 8 bpp を利用しています。以下にデバイスを示します。

GPNGL8	CMY Lookup Device 1 d.p.mm
GPNGI8	CMY Intensity Device 1 d.p.mm
GPNGL8H	CMY Lookup Device 300 d.p.i
GPNGI8H	CMY Intensity Device 300 d.p.i
GPNGL8M	CMY Lookup Device 150 d.p.i
GPNGI8M	CMY Intensity Device 150 d.p.i
GPNGL8L	CMY Lookup Device 75 d.p.i
GPNGI8L	CMY Intensity Device 75 d.p.i
SPNGL8	RGB Lookup Device 1 d.p.mm
SPNGI8	RGB Intensity Device 1 d.p.mm
SPNGL8H	RGB Lookup Device 300 d.p.i

SPNGI8H	RGB Intensity Device 300 d.p.i
SPNGL8M	RGB Lookup Device 150 d.p.i
SPNGI8M	RGB Intensity Device 150 d.p.i
SPNGL8L	RGB Lookup Device 75 d.p.i
SPNGI8L	RGB Intensity Device 75 d.p.i

#### 5.4.7 Java への出力

ljava	Java Draw Method Driver
-------	-------------------------

---

## 第 6 章 日本語を扱うには

---

GsharpWE で作成されるグラフの注釈や、座標のタイトルなどに日本語を扱うことができます。Gsharp は日本語を含むデータベースを持っています。このデータベースに登録されているフォントを利用して表示を行ないます。

## 6.1 日本語データベースへの変換

日本語をこのデータベースのフォントに変換する関数が準備されています。convjp() という関数を利用して行います。この関数については Gsharp のマニュアルにも記載されていますのであわせてご参照ください。

### 関数の利用方法

convjp() 関数は以下の引数を持っています。

```
string convjp(string japanese, int type);
```

type パラメータ

type = 1 : 明朝体  
type = 2 : ゴシック体

明朝体とゴシック体の2種のフォントをサポートしています。

入力された japanese と type に応じてデータベース番号 (インデックス) を返り値として返します。

convjp() 関数からの返り値は、Gsharp のデータベースから文字列を指定するためのインデックスと一緒に設定されます。日本語を表示するには、必ず、以下の textoption の設定が必要です。

例 :

```
set textoption
( XuNunicldIndexChar = "^"
);

jap = convjp(" 日本語 ", 1);

create Viewport page_1.viewport_1 ;
create Domain page_1.viewport_1.domain_1 ;
create Note page_1.viewport_1.domain_1.note_1
( XuNheight = 16.8574,
  XuNhorizontalJustification = "left",
  XuNposition = (35.8433,57.1429),
  XuNtext = jap
);
```

例えば、上記の例は、注釈オブジェクトを利用した例です。まず Gsharp におけるグラフテンプレートを作成した際に適当な文字列を利用して GSL ファイルを作成してください。

その後、上記の textoption の指定、convjp 関数を利用した日本語変換を行い、そのコードを文字列に利用します。

## 6.2 日本語ファイルを利用する方法

Gsharp (GsharpWE) では、前節で説明したように日本語はコードを用いて表示するようになっています。日本語を含むファイルを利用して、グラフのタイトルや軸のラベルなどを作成する場合、この関数を利用してコード変換を行いません。

例：

以下のデータファイルを利用した処理例をもとに説明します。

データファイル：test.dat

```

===== ここから =====
東京都における温度分布
1月 8.0
2月 12.0
3月 15.0
4月 18.0
5月 23.0
6月 18.0
7月 22.0
8月 24.0
9月 13.0
10月 15.0
11月 17.0
12月 9.0
===== ここまで =====

```

### 1. ステップ 1

まず、データファイルからグラフのタイトルとして利用する文字列と各月のラベルデータ、各データを読み込みます。

```

GSL ファイル： readA.gsl
GSL ファイル： readB.gsl

```

このデータファイルは、2つの方法で読み込むことができます。  
 1つは Report (レポート) タイプを利用して読み込む方法です。  
 レポートタイプのオプション設定を利用して、1行目をそのデータのタイトルとして読み込みます。ただし、この場合はデータセット名が A1, T1 になります。  
 (A1: 文字列データ、T1: 数値データ)

注意： SJIS コードのファイルを読み込む場合、コードの問題で文字が正常に読み込めない場合があります。以下に示す ASCII タイプをご利用ください。

もう1つは、Ascii (アスキー) タイプを利用して読み込む方法です。月のデータが1カラム目から6カラム目までになっている点に注目し、その設定を行いません。

以下にそれぞれの読み込み GSL ファイルを示します。

```

===== 読み込み GSL ファイル readA.gsl =====
DATAFILE = "/home/user/uniras/jap/test.dat";

import_report(DATAFILE,"","gtitle");
mdata = T1;
tdata = T2;
===== ここまで =====

```

まず、上記の GSL ファイルはレポートタイプで読み込んだ例です。読み込む際に 1 行目をタイトル gtitle として読み込んでいます。データセット名がない場合、T1,T2 の番号付けで読み込まれますので、そのデータを mdata, tdata にそれぞれ変換しています。

```
===== 読み込み GSL ファイル readB.gsl =====
DATAFILE = "/home/user/uniras/jap/test.dat";

import_ascii(DATAFILE,1,1,,,"gtitle","text");
import_ascii(DATAFILE,2,,1,6,,,"mdata","text");
import_ascii(DATAFILE,2,,7,,,"tdata","real");
===== ここまで =====
```

次は、アスキータイプを利用した例です。

まず、gtitle という名前で、1 行目から 1 行目まで (1 行目のみ) を text データ (文字列) として読み込みます。

次に、mdata という名前で、左列の月データを、2 行目から読み込みます。また、その際に 1 カラム (列) から 6 カラム (列) までを text データ (文字列) として読み込みます。

最後に、実際のデータ値を 2 行目から後ろ、また、7 カラム (列) から後ろを real データ (数値) として読み込みます。

この GSL ファイルは、データ入力を対話的に行ない (アスキータイプ利用) 操作の履歴を残す方法で保存できます。

## 2. ステップ 2

GSL ファイル: conv.gsl

```
===== 文字列変換 GSL ファイル =====
# create japanese label
string graphlabel[sizex(mdata)];

# create japanese
graphtitle = convjp(gtitle,1);
for i = 1 to sizex(mdata)
  graphlabel[i] = convjp(mdata[i],2);
endfor
===== ここまで =====
```

次に、読み込まれた文字列を日本語コードに変換する必要があります。

- sizex(mdata) = 12  
この例では、1 月 ~ 12 月の 12 個が計算されます。
- graphlabel[]  
グラフの X 軸ラベルとしてプロットするための変数を指定します。配列の数は、読み込んだデータと同じ数に設定しています。
- graphtitle = convjp(gtitle,1);  
読み込んだデータファイルの 1 行目をコードに変換します。
- for i = 1 to sizex(mdata)  
graphlabel[i] = convjp(mdata[i],2);

```
endfor
読み込んだ 1 月 ~ 1 2 月をコードにそれぞれ変換します。
```

### 3. ステップ 3

上記のステップ 1、2 を行なう GSL ファイルを作成します。  
下の例では、アスキータイプによる読み込み GSL ファイル ( readB.gsl ) を利用しています。

GSL ファイル: input.gsl

```
===== データの読み込みと文字列変換 GSL ファイル =====
/* input data file */
DATAFILE = "/home/user/uniras/jap/test.dat";
import_ascii(DATAFILE,1,1,,,"gtitle","text");
import_ascii(DATAFILE,2,,1,6,,,"mdata","text");
import_ascii(DATAFILE,2,,7,,,"tdata","real");

# textoption
set textoption
  ( XuNucicIdIndexChar = "^"
  );

# create japanese label
string graphlabel[sizex(mdata)];

# create japanese
graphtitle = convjp(gtitle,1);
for i = 1 to sizex(mdata)
  graphlabel[i] = convjp(mdata[i],2);
endfor
===== ここまで =====
```

日本語は、^ というコードを元に unicId データベースから検索が行なわれます。  
そのテキストのオプションを設定しています。

### 4. ステップ 4

Gsharp を起動し、ステップ 3 で作成した GSL ファイルを読み込んで下さい。データ・マネージャを起動し、データセットをチェックしてみてください。  
graphlabel, graphtitle などのコード番号を見ることができます。

### 5. ステップ 5

タイトルやグラフのラベルなどに graphtitle, graphlabel を用いてグラフを作成します。その後、そのグラフの状態を GSL ファイルに保存して下さい。

以下は、保存された GSL ファイルにステップ 3 のデータ入力部分を合わせて ( include を利用してインクルードしています ) 完成された GSL ファイルです。

```
===== 完成 GSL ファイル =====
include "./input.gsl";

set displayoption
  ( XuNautoApply = false,
    XuNautoRepaint = true,
    XuNgraphPicking = true
  );
set page_1
  ( XuNsize = (145.656,116.156) mm
  );
```

```
create Viewport page_1.viewport_1
  ( XuNfirstDiagonalPoint = (15.8931,13.4039) %,
    XuNsecondDiagonalPoint = (80.9423,83.5097) %
  );
create Domain page_1.viewport_1.domain_1;
set page_1.viewport_1.domain_1.xaxis1
  ( XuNaxleType = "userLabeled",
    XuNaxleUserLabelsData = "graphlabel"
  );
create Graph page_1.viewport_1.domain_1.graph_1
  ( XuNgraphType = "line",
    XuNyData = "tdata"
  );
create Title page_1.viewport_1.title_1
  ( XuNtitleText = "graphtitle"
  );
repaint;
===== ここまで =====
```

この GSL では、graphtitle をグラフのタイトルに、また、X 軸にユーザ定義ラベルを指定しています。その X 軸のラベルデータに graphlabel データを用いています。

---

## 第 7 章 Gsharp と GsharpWE

---

この章では、Gsharp と GsharpWE の関係、また、実際の作業のステップについて説明しています。

Gsharp で作成したグラフのテンプレートと GsharpWE へのコーディング方法について記述しています。各ステップに沿ってテストしてみてください。

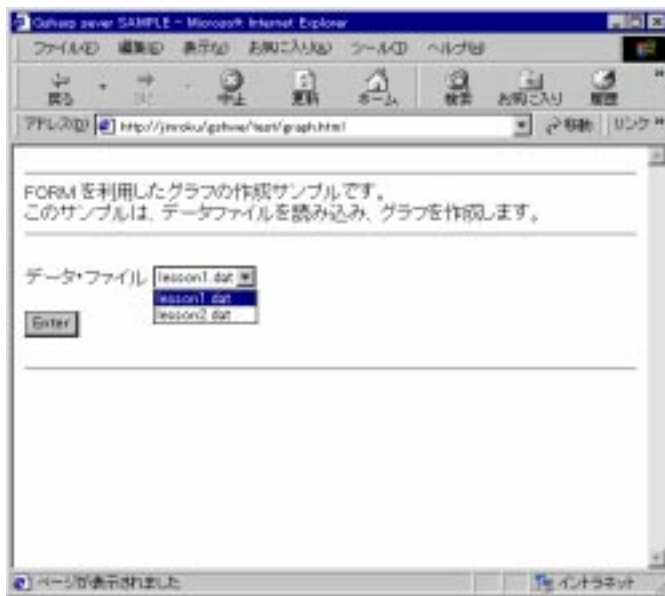
また、本章では、Gsharp のグラフ作成の手順から記述されていますが、Gsharp によるグラフ作成の詳細については、Gsharp のマニュアルをご参照ください。

## 7.1 チュートリアル概要

このチュートリアルで利用するデータ、出力結果は以下の通りです。

### インターフェース

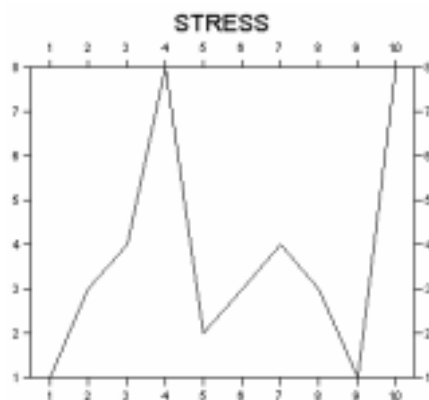
グラフを作成する HTML ページです。  
以下のページにアクセスし、グラフを作成します。



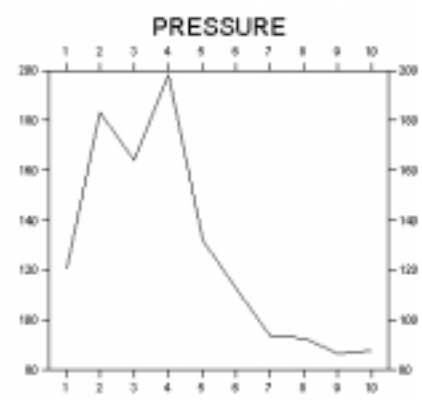
2つのタイプのデータファイルを選択します。  
その選択に応じて以下の結果を出力します。

### 出力結果

以下はそのアクセス結果です。



lesson1.dat が選択された場合



lesson2.dat が選択された場合

## データ・ファイル

このチュートリアルでは以下のフォーマットのデータファイルを利用します。  
実際にステップにそって実行してみる場合、データファイルを2つ作成してください。  
(中身は適当で結構です。)

STRESS	PRESSURE
1.0	121.0
3.0	183.0
4.0	164.0
8.0	198.0
2.0	132.0
3.0	113.0
4.0	94.0
3.0	93.0
1.0	87.0
8.0	88.0

lesson1.dat

lesson2.dat

2つのデータを読み替え、1行目の文字列をグラフのタイトルに設定します。

## 7.2 Gsharp 上での操作

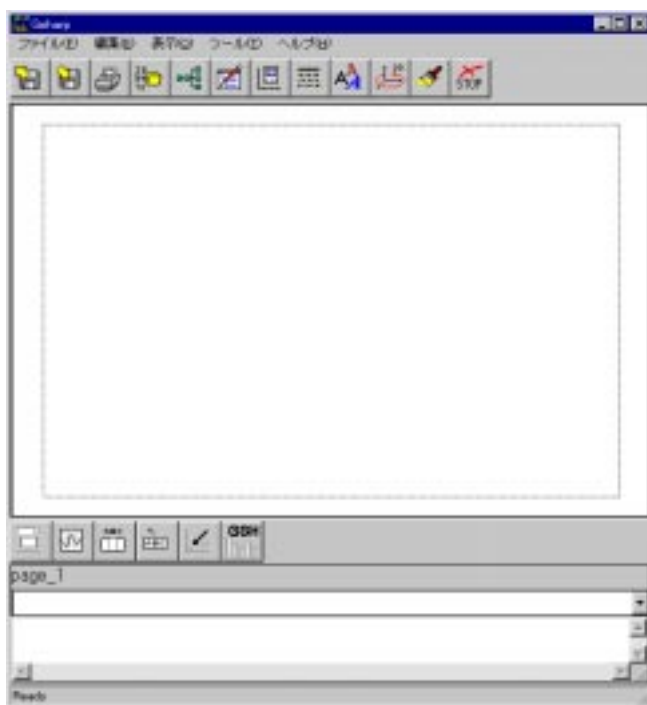
まず最初に Gsharp を利用してグラフ処理を行います。  
作成したグラフのパターンをグラフ・テンプレートとして保存します。

注意： 以降のステップでは *Gsharp for Windows* の画面を例に説明していません。  
*Gsharp for UNIX* をご利用の場合も同じパラメーターに値を設定してください。

また、*Gsharp* の操作方法の詳細については *Gsharp* のマニュアルをご参照ください。

### 7.2.1 Gsharp の起動

Gsharp を起動します。

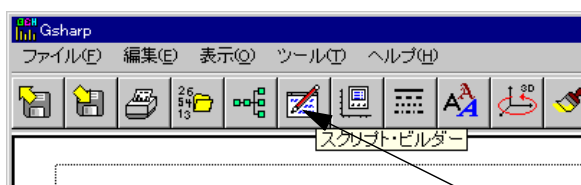


起動方法については、*Gsharp* のインストール・ガイド、チュートリアル・ガイドをご参照ください。

### 7.2.2 データ読み込みスクリプトの作成

次にデータファイルの読み込みスクリプトを作成します。

1. スクリプト・ビルダーを起動します。



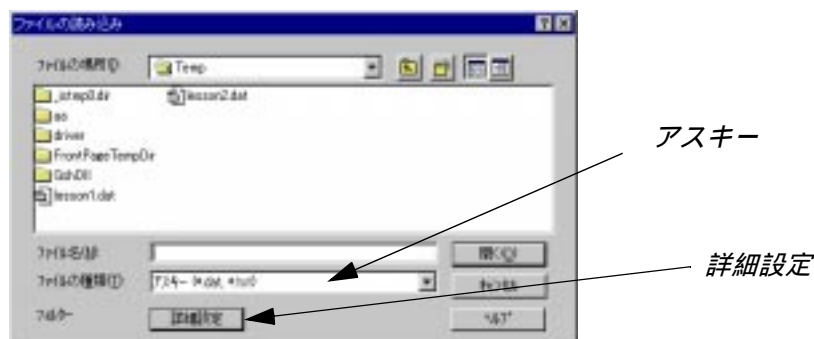
スクリプト・ビルダー・アイコン

2. オプション・メニューにある操作ログの作成を選択（オン）します。



3. データの読み込みを行います。  
メイン・ウィンドウのファイルの読み込みメニューを選択します。

開いたダイアログでファイル・タイプをアスキーに設定します。



アスキー

詳細設定

4. Windows 版では詳細設定ボタンを、UNIX 版では押すボタンを押して、読み込み方法を指定します。



データセット名 = label

タイプ = 文字

読み込み開始行 = 1

読み込み終了行 = 1

lesson1.dat の 1 行目のタイトル用の文字列を読み込むための指定を行います。

5. ファイル名 (lesson1.dat) を指定して読み込みを行います。

データ・マネージャーを起動し、データが正常に読み込めたかどうかを確認してください。

また、スクリプト・ビルダー内に残ったログをご覧ください。  
以下のようなログが残っているはずです。

```
import_ascii("C:/TEMP/test/lesson1.dat",1,1,,,"label","text");
```

続けて、ファイル内の実際のデータを読み込みます。

同様にファイル・ダイアログを開き、アスキータイプで以下の設定を行います。

データセット名 = Ydata

タイプ = 数値

読み込み開始行 = 2

読み込み終了行 = UNDEF

パラメーターの左にあるチェックを  
クリックします。

Windows 版では 0 と表示されます。

6. ログをファイルに保存します。  
例えば、read.gsl という名前で保存してください。

```
import_ascii("C:/TEMP/test/lesson1.dat",1,1,,,"label","text");
```

```
import_ascii("C:/TEMP/test/lesson1.dat",1,,,,,"Ydata");
```

### 7.2.3 グラフの作成とスクリプトの保存

次にグラフを作成し、レイアウトを整えます。

1. クリアとデータの読み込み  
一度、すべてをクリアし、前のステップで保存したデータの読み込みスクリプトが正常に読み込めることを確認してください。  
Windows 版では編集メニューにあるすべてを削除を選択します。  
UNIX 版ではファイルメニューにあるすべてを削除を選択します。

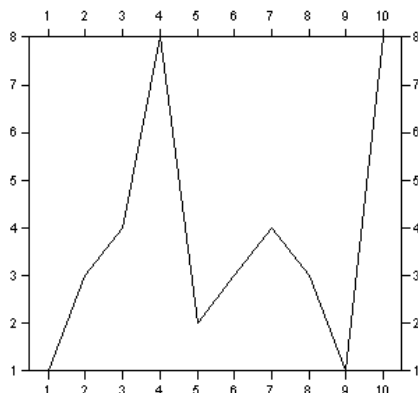
また、スクリプト・ビルダーは利用しませんので閉じてください。

ファイルの読み込みメニューでファイル・タイプを GSL スクリプトに設定し、  
read.gsl ファイルを選択します。

また、データ・マネージャーを利用してデータが読み込めたことを確認してください。

2. グラフの作成  
Windows 版では編集メニューにあるオブジェクトの作成 / グラフを選択します。  
UNIX 版では作成 / グラフ・メニューを選択します。

今回読み込んだデータは Y データのみです。グラフタイプを曲線グラフに設定し、Y データのみに value を設定します。



適当に編集を行なってみて下さい。  
図の例では、第 2 座標軸も表示状態に設定しています。  
(階層ブラウザを起動し、マウス右ボタンから表示を選択します。)

3. タイトルの作成  
タイトルを作成するには、まずビューポートを指定しなければなりません。ビューポートを指定した後、Windows 版では編集メニューにあるオブジェクトの作成 / タイトルを、UNIX 版では作成 / タイトルメニューを選択します。

データファイルからタイトルに利用する文字列のデータを読み込んでいますので、その文字データを指定します。  
タイトルの文字列の入力部分の横にある [...] ボタンを押して、データセット label を指定します。

4. ページ・サイズの設定  
このグラフ処理の結果はイメージで利用しますので、ページサイズを設定します。ページ (何も表示されていない場所) を選択した状態で編集のダイアログを開きます。



custom(ユーザー指定)を選択します。

サイズに custom を選択します。  
サイズを例えば 400 x 350 に設定します。

このチュートリアルでは GIF の 1 mm = 1 pixel のデバイスを利用します。このサイズがイメージのドットサイズになります。

## 5. スクリプトの保存

作成したグラフの状態をスクリプトに保存します。

ファイル・メニューにある保存を選択します。

ファイル・タイプを GSL スクリプトに設定し、適当なファイル（例えば graph.gsl）で保存します。

```
set displayoption
  ( XuNautoRepaint = true
  );
set textoption
  ( XuNunicIdIndexChar = "^"
  );
set page_1
  ( XuNformat = "custom",
    XuNsize = (400,350) mm
  );
create Viewport page_1.viewport_1
  ( XuNfirstDiagonalPoint = (16.4039,10.3175) %,
    XuNsecondDiagonalPoint = (83.7838,79.2929) %
  );
create Domain page_1.viewport_1.domain_1;
set page_1.viewport_1.domain_1.xaxis2
  ( XuNaxle = true
  );
set page_1.viewport_1.domain_1.yaxis2
  ( XuNaxle = true
  );
create Graph page_1.viewport_1.domain_1.graph_1
  ( XuNyData = "Ydata"
  );
create Title page_1.viewport_1.title_1
  ( XuNobjectEnabled = true,
    XuNtitleText = "label"
  );
```

注意： データの読み込みに関する記述 (*import\_xxx*) が入っている場合は、削除してください。読み込み部分は *read.gsl* として別途保存していますので、そちらを利用します。

## 7.2.4 スクリプトの編集

スクリプトの編集を行います。

作成した2つのスクリプト (*read.gsl* と *graph.gsl*) から自動実行できるスクリプト (*auto.gsl*) を作成します。

GSL の関数化などを利用して以下に示すスクリプトに編集してみてください。

```
function make_graph()
  set displayoption
    ( XuNautoRepaint = true
    );
  set textoption
    ( XuNunicIdIndexChar = "^"
    );
  set page_1
    ( XuNformat = "custom",
      XuNsize = (400,350) mm
    );
```

```

create Viewport page_1.viewport_1
  ( XuNfirstDiagonalPoint = (16.4039,10.3175) %,
    XuNsecondDiagonalPoint = (83.7838,79.2929) %
  );
create Domain page_1.viewport_1.domain_1;
set page_1.viewport_1.domain_1.xaxis2
  ( XuNaxle = true
  );
set page_1.viewport_1.domain_1.yaxis2
  ( XuNaxle = true
  );
create Graph page_1.viewport_1.domain_1.graph_1
  ( XuNyData = "Ydata"
  );
create Title page_1.viewport_1.title_1
  ( XuNobjectEnabled = true,
    XuNtitleText = "label"
  );
endfunction

# main program

DATAFILE="C:/TEMP/test/lesson1.dat";
import_ascii(DATAFILE,1,1,,,"label","text");
import_ascii(DATAFILE,1,,,,,"Ydata");

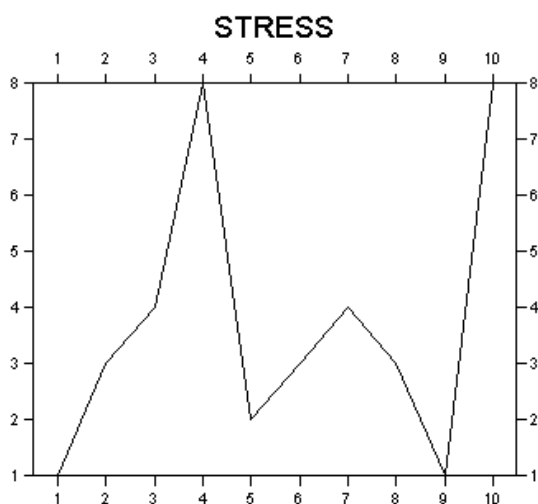
make_graph();
repaint;

```

グラフ作成を関数化し、データファイル名を変数で扱っています。なお、repaint; は再描画命令です。

## 7.2.5 スクリプトの再実行

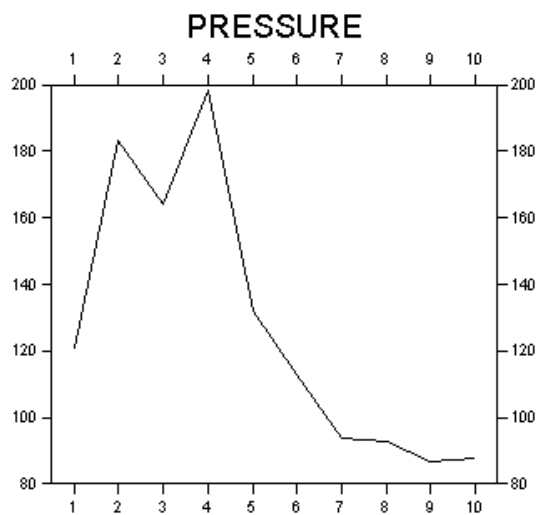
以上で Gsharp 上での操作は終了です。一度 Gsharp 上ですべてを削除し、このスクリプト auto.gsl が正常に動作することを確認してください。



また、DATAFILE の行を以下のように lesson2.dat に書き換えて、2 つ目のデータファイルに対して、同じパターンのグラフが作成できることを確認してください。

```
DATAFILE="C:/TEMP/test/lesson2.dat";
```

ファイルの読み込みメニューを選択し、開いたダイアログで GSL スクリプトを指定します。auto.gsl を読み込んでみてください。



lesson2.dat の読み込み結果

## 7.3 Gsharp Web Edition への展開

前節で作成したスクリプトを WWW の CGI に展開します。

### 7.3.1 FORM を持ったページの作成

まず、ユーザ情報を入力するページを作成します。以下の HTML ファイルを作成します。

FORM を持ったページ

```

=====
<HTML>
<HEAD>
<TITLE> Gsharp sever SAMPLE </TITLE>
</HEAD>
<BODY>
<HR>
FORM を利用したグラフの作成サンプルです。<br>
このサンプルは、データファイルを読み込み、グラフを作成します。<br>
<HR>
<FORM METHOD="post" ACTION="http://cgi-bin/gsharp_test.cgi">
  グラフ <SELECT NAME="graphName">
    <OPTION VALUE = "/usr/local/www/gsharpwe/test/lesson1.dat">lesson1.dat
    <OPTION VALUE = "/usr/local/www/gsharpwe/test/lesson2.dat">lesson2.dat
  </SELECT>
<BR>

  <INPUT TYPE="submit" VALUE="Enter">
</FORM><HR>

</BODY>
</HTML>
=====

```

このページでは、Enter ボタンが押されると cgi-bin にある gsharp\_test.cgi プログラムが実行されるようになっています。

また、その際に graphName という名前でその選択メニューで指定された情報が渡されます。

### 7.3.2 Gsharp WE による CGI スクリプトの作成

まず、グラフのテンプレートを思い出して下さい。

グラフ作成スクリプト auto.gsl

```

=====
function make_graph()
  set displayoption
    ( XuNautoRepaint = true
    );
  set textoption
    ( XuNunicIdIndexChar = "^"
    );
  set page_1
    ( XuNformat = "custom",
      XuNsize = (400,350) mm
    )

```

```

    );
    create Viewport page_1.viewport_1
      ( XuNfirstDiagonalPoint = (16.4039,10.3175) %,
        XuNsecondDiagonalPoint = (83.7838,79.2929) %
      );
    create Domain page_1.viewport_1.domain_1;
    set page_1.viewport_1.domain_1.xaxis2
      ( XuNaxle = true
      );
    set page_1.viewport_1.domain_1.yaxis2
      ( XuNaxle = true
      );
    create Graph page_1.viewport_1.domain_1.graph_1
      ( XuNyData = "Ydata"
      );
    create Title page_1.viewport_1.title_1
      ( XuNobjectEnabled = true,
        XuNtitleText = "label"
      );
endfunction

# main program

DATAFILE="C:/TEMP/test/lesson1.dat";
import_ascii(DATAFILE,1,1,,,,,"label","text");
import_ascii(DATAFILE,1,1,,,,,"Ydata");

make_graph();
repaint;
=====

```

このスクリプトでは、データの読み込み、曲線グラフの作成を行いません。  
 また、データは DATAFILE という変数でユーザが指定してグラフを書くようになっています。  
 このスクリプトにイメージの作成、データの受け渡しを組み込みます。

## イメージの作成

まず最初にイメージ生成を組み込みます。

1行目に以下の便利な関数を定義しているファイルを include します。

```
include "$UNIDIR/lib/libhtml.gsl";
```

また、make\_graph() 関数の呼び出し、repaint; コマンドの次にイメージ作成関数を組み込みます。

```
string imgfile2;
imgfile2 = "C:/TEMP/lesson.gif";
make_image(400, 350, imgfile2);
```

以下のように変更します。

```

      イメージ作成スクリプト autoweb.gsl
      =====
      include "$UNIDIR/lib/libhtml.gsl";

```

```

function make_graph()
  set displayoption
    ( XuNautoRepaint = true
    );
  set textoption
    ( XuNunicIdIndexChar = "^"
    );
  set page_1
    ( XuNformat = "custom",
      XuNsize = (400,350) mm
    );
  create Viewport page_1.viewport_1
    ( XuNfirstDiagonalPoint = (16.4039,10.3175) %,
      XuNsecondDiagonalPoint = (83.7838,79.2929) %
    );
  create Domain page_1.viewport_1.domain_1;
  set page_1.viewport_1.domain_1.xaxis2
    ( XuNaxle = true
    );
  set page_1.viewport_1.domain_1.yaxis2
    ( XuNaxle = true
    );
  create Graph page_1.viewport_1.domain_1.graph_1
    ( XuNyData = "Ydata"
    );
  create Title page_1.viewport_1.title_1
    ( XuNobjectEnabled = true,
      XuNtitleText = "label"
    );
endfunction

# main program

DATAFILE="C:/TEMP/test/lesson1.dat";
import_ascii(DATAFILE,1,1,,,"label","text");
import_ascii(DATAFILE,1,,,,,"Ydata");

make_graph();
repaint;

string imgfile2;
imgfile2 = "C:/TEMP/test/lesson.gif";
make_image(400, 350, imgfile2);
=====

```

まずは、このスクリプトが動作し、イメージが作成できることを GsharpWE を使って確認してください。

```
<inst_dir>/bin/GsharpWE autoweb.gsl
```

注意: UNIX 版では、環境変数が必要です。  
「2.1.2 ファイル I/O」 - 2-3 ページをご参照ください。

指定したイメージファイルが作成できることを確認します。

## イメージのキャッシュに関する回避と Web 上での参照

次にイメージの名前は常に同じだとキャッシュされてしまいます。また同時アクセスされた場合、異なる絵を作ることができません。以下のようにイメージの名前をプロセス ID やランダム数で作成するように設定します。

また、イメージファイルは、その Web 上で認識できるディレクトリに作成しなければなりません。Web の http で見ることができるディレクトリ名と実際のディレクトリ名の 2 つを変数で扱い、イメージの名前を決めます。

**ディレクトリ名の指定** まず、スクリプトの最初の方で、Web 環境に応じたディレクトリ名を変数定義します。

```
IMGWEB = "/gshwe/img_tmp/";
IMGDIR = "c:/inetpub/wwwroot/gshwe/img_tmp/";
```

- IMGWEB は `http://<machine>/gshwe/img_tmp/` のように Web 上でアクセスできるディレクトリ名です。  
IMGDIR はその実際のマシン上でのディレクトリ名です。

「イメージの作成」 - 2-9 ページをあわせてご参照ください。

**ファイル名の決定** 次にファイル名を指定します。先に作成した `imgfile2` のファイル名の変わりに以下のように定義します。

```
seed(secsince(time));
tnumber = int(rnd(1)*100000);
if os_type() = "Windows" then
    string imgname; imgname = "hello"+ tnumber + ".gif";
else
    string imgname; imgname = "hello"+ getpid() + ".gif";
endif
string imgfile; imgfile = IMGWEB + imgname;
string imgfile2; imgfile2 = IMGDIR + imgname;
```

Windows では `getpid()` 関数でユニークな ID が決まらないので、ランダム値を利用します。  
「イメージ・ファイル名」 - 2-11 ページをご参照ください。

以下に変更したスクリプトを示します。

```

    イメージ作成スクリプト 2  autoweb.gsl
=====
include "$UNIDIR/lib/libhtml.gsl";

IMGWEB = "/gshwe/img_tmp/";
IMGDIR = "c:/inetpub/wwwroot/gshwe/img_tmp/";

function make_graph()
    set displayoption
        ( XuNautoRepaint = true
        );
    set textoption
        ( XuNunicIdIndexChar = "^"
        );
    set page_1
        ( XuNformat = "custom",
          XuNsize = (400,350) mm
        );
endfunction
```

```

create Viewport page_1.viewport_1
  ( XuNfirstDiagonalPoint = (16.4039,10.3175) %,
    XuNsecondDiagonalPoint = (83.7838,79.2929) %
  );
create Domain page_1.viewport_1.domain_1;
set page_1.viewport_1.domain_1.xaxis2
  ( XuNaxle = true
  );
set page_1.viewport_1.domain_1.yaxis2
  ( XuNaxle = true
  );
create Graph page_1.viewport_1.domain_1.graph_1
  ( XuNyData = "Ydata"
  );
create Title page_1.viewport_1.title_1
  ( XuNobjectEnabled = true,
    XuNtitleText = "label"
  );
endfunction

# main program

DATAFILE="C:/TEMP/test/lesson1.dat";
import_ascii(DATAFILE,1,1,,,"label","text");
import_ascii(DATAFILE,1,1,,,"Ydata");

make_graph();
repaint;

# image name
seed(secsince(time));
tnumber = int(rnd(1)*100000);
if os_type() = "Windows" then
  string imgname; imgname = "hello"+ tnumber + ".gif";
else
  string imgname; imgname = "hello"+ getpid() + ".gif";
endif
string imgfile; imgfile = IMGWEB + imgname;
string imgfile2; imgfile2 = IMGDIR + imgname;

make_image(400, 350, imgfile2);
=====

```

## HTML コードの出力

以上の設定でイメージファイルを作成することができました。  
HTML をクライアント側に返すには、GSL スクリプトの中で行っても結構ですし、Perl や shell など上かぶせしたスクリプトから出力しても結構です。

以下の例は、GSL のスクリプトからイメージを返す方法です。  
make\_image 関数の後ろ（スクリプトの最後）に以下のコードを記述します。

```

/*
 * write out HTML
 */
id = 1;
fwrite(id, "Content-type: text/html");

```

```

fwrite(id, "");
fwrite(id, "<HTML><BODY BGCOLOR=#FFFFFF>");
fwrite(id, "<TITLE>GsharpWE Greeting</TITLE>");
fwrite(id, "<CENTER><FONT SIZE=+4><B>");
fwrite(id, "First Web Step");
fwrite(id, "</B></FONT><P>");
fwrite(id, "<IMG SRC=¥", imgfile, "¥ ">");
fwrite(id, "</CENTER>");
fwrite(id, "</BODY></HTML>");

```

イメージファイルを Web から参照する場合は、imgfile を送ります。  
以下のように実行し、HTML コードが出力されることを確認してください。

```

C: ¥ TEMP ¥ test> <inst_dir>/GsharpWE autoweb.gsl
Import ASCII: 1 values read into label
Import ASCII: 10 values read into Ydata
Content-type: text/html

```

```

<HTML><BODY BGCOLOR=#FFFFFF>
<TITLE>GsharpWE Greeting</TITLE>
<CENTER><FONT SIZE=+4><B>
First Web Step
</B></FONT><P>
<IMG SRC="
/gshwe/img_tmp/hello37784.gif
">
</CENTER>
</BODY></HTML>

```

```
C: ¥ TEMP ¥ test>
```

また、イメージファイルができていることを確認します。

## データ 10

最後にデータ 10 を組み込みます。

まず、Web のページ (HTML+FORM) を作成します。  
以下に例を示します。

```

<HTML>
<HEAD>
<TITLE> Gsharp server SAMPLE </TITLE>
</HEAD>
<BODY>

<HR>
FORM を利用したグラフの作成サンプルです。<p>
このサンプルは、データファイルを読み込み、グラフを作成します。<p>
<HR>
<FORM ACTION="/cgi-bin/autoweb.gsl" METHOD="POST">

```

データ・ファイル

```

<SELECT NAME="filename">
  <OPTION VALUE="c:/TEMP/test/lesson1.dat">lesson1.dat
  <OPTION VALUE="c:/TEMP/test/lesson2.dat">lesson2.dat
</SELECT>

```

```
<p><INPUT TYPE="SUBMIT" VALUE="Enter"><p>
</FORM>
<HR>

</BODY>
</HTML>
```

上記例では、ファイル名を絶対パスで指定しています。

この filename を GSL スクリプトで入手し、データの読み込みに利用します。

```
DATAFILE=FORM_filename;
import_ascii(DATAFILE,1,1,,,"label","text");
import_ascii(DATAFILE,2,,,"Ydata","real");
```

FORM で指定された文字列は、GsharpWE では FORM\_ で認識できます。「実行される CGI プログラム」 - 2-5 ページをご参照ください。

## メッセージのオフ

最後に、Gsharp から出力されるメッセージを消去します。

Gsharp では、データの読み込み ( import\_report ) などを行うとそのメッセージが標準出力に出力されます。

Web 上で GSL スクリプトを利用する場合、これらのメッセージが出力されると、クライアントにそのメッセージが送信されたり、場合によっては正常に動作できません。以下の関数をスクリプトの最初 ( main 文の最初 ) に組み込んでください。

```
set_messages(0,0,0,0);
```

この関数は Gsharp が出力するエラーメッセージやユーザーメッセージなどを 0,1 でオン、オフ指定する関数です。

すべての引数を 0 に設定することで、メッセージは何も表示されなくなります。

これまで見てきたスクリプトでは、# main の次の行 ( データ読み込みの前 ) に最初に上記関数を指定してください。

以上で HTML ファイルとその FORM から実行される CGI スクリプトの作成は終了です。UNIX 版では Shell や perl から上記 GSL を呼び出すようにしてご利用ください。

また、「イメージの削除」 - 2-12 ページなどを参考にテンポラリ・イメージファイルを削除する仕組みなど組み込んでみてください。



## 注意

本書で解説するソフトウェア（対象ソフトウェア）は、所定のライセンス契約が締結された場合に限り、その使用あるいは複製が許可されます。

本書は、Advanced Visual Systems 社の Gsharp Web Edition Users Guide に基づいたものであり、その著作権は株式会社ケイ・ジー・ティーが保有しています。本書中の解説および図、表は株式会社ケイ・ジー・ティーの文書による許可なしに、その全部または一部を、いかなる場合にも再版あるいは複製することを禁じます。

本書に記載されている事項は、予告なく変更されることがありますので、あらかじめご了承おき下さい。万一、本書に偽りがあった場合でも、株式会社ケイ・ジー・ティーは一切その責任を負いかねます。

株式会社ケイ・ジー・ティーは、株式会社ケイ・ジー・ティーまたは株式会社ケイ・ジー・ティーの指定する会社から納入された機器以外で対象ソフトウェアを使用した場合、その性能あるいは信頼性について一切責任を負いかねます。

Copyright © 1997 ~ 2000  
KGT Inc.  
All Right Reserved  
Printed in Japan

AVS, Gsharp, AVS/Express は米国 Advanced Visual Systems 社の商標です。  
上記以外の製品名も一般に開発各社の商標、あるいは登録商標です。

株式会社ケイ・ジー・ティー  
〒160 東京都新宿区新宿 2-8-8 とみん新宿ビル 5F

