eyBuild 中文手册

版本状态

文件状态： ［ ］草稿 ［√］正式发布 ［ ］正在修改	文件标识：	eyBuild Group
	当前版本：	0.2.2
	作　　者：	newzy
	完成日期：	2006-2-13

修订日志

版本	日期	变更位置	变更内容	变更者
0.1.0	2006-1-22		新建	newzy
0.2.0	2006-2-6	全部	完成初版	newzy
0.2.1	2006-2-9	CSP 注释符	CSP 注释符由#改为!	newzy
0.2.2	2006-2-13	头部	CSP 的解释、联系信息等	newzy

更多信息：

请访问eyBuild的网站 http://www.eybuild.com

或发送email到: newzy@eybuild.com, xxt@eybuild.com

目录

第1章　什么是 eyBuild

eyBuild，是 Easy Build 的简写，它是 CSP（C Language Service Page）开发套件的总称。eyBuild 是嵌入式设备上 WEB 应用的最理想的开发工具，同是也是服务器 WEB 应用开发最有益的补充，它包括 CSP2BIN、WEB2BIN、DONEMAP、eyBuildLib 等开发组件。

CSP 是 C Language Service Page 的缩写，它是一种基于 C 语言的脚本开发技术。跟 ASP、JSP 类似，它们都是 CGI（Common Gateway Interface）的一种变种。我们可以直接在 HTML 等文件中嵌入任意 C 语句形成 CSP 的源文件，然后用 CSP2BIN 将该文件转换成 C 语言的源程序。

1.1　CSP 与 CGI

自 CGI 产生以来，C 语言以其高效性、灵活性和通用性一直是开发交互式 WEB 应用的最有吸引力的选择。但近年来，能直接内嵌于 HTML 文档中间的各种脚本工具，以其简便性易用性使一部分用户开始放弃了直接用 C 来开发 CGI 脚本。

但还有两类用户没有放弃用 C 来开发它们的应用，一是对性能追求较高的高端开发者，二是嵌入式设备的开者。前者选择 C 语言来开发它们的 WEB 应用，是因为 C 高效性、灵活性和通用性是各种脚本工具无法取代的。后者选择 C 语言，是因嵌入式设备的特点（内存、CPU 等资源有限等，不可在设备上运行如 ASP，PHP，PERL 等的脚本的运行环境）决定的（另外，目前嵌入式设备主要以 C 语言开发为主）。

自 CGI 出现以来，就有人开发方便 C 语言开发的开发库 cgilib，从目前的情况看 cgilib 并没有得到广泛的应用；再反看下各种脚本开发工具，它们则受到 WEB 开发者的直接异常青睐。对比后不难发现，基于 C 语言的 cgilib 并没有从根本上解决交互式 WEB 应用的真正需求，这是因为直接用 C 语言无法像其它脚本一样嵌入在 HTML 文档中间，使得用输出 HTML 文档时非常烦琐，源代码可读性较差，维护困难。

现在用 CSP 来开发 WEB 应用程序则就不同了，你可以像 ASP、JSP、PHP 等工具一样在 HTML 文档中自由地嵌入 C 语言的程序代码。如果你是个 C/C++或 JAVA 程序员，你不必重新学习任何一种新语言，即可以像高效地开发 WEB 应用，工作模式跟原来几乎一样。如果曾经用 ASP、JSP、JSP 开发过并有一定的 C 语言基础，你会发现 CSP 简单就是它们的一种变种。无需太多学习即可用 CSP 做开发了。

下表是一段简单的 CSP 代码（头文件包含不是必须的，仅用作示范），它是将一段 C 程序直接嵌套在 HTML 模板文件中：

1	<html>	
2	<body>	
3	<%	语句起始标签
4	# list includes files	注释行
5	@ include <stdio.h>	C 语言头文件包含
6	@ include <my_header.h>	C 语言头文件包含
7	%>	语句结束标签

8	输出：<% = “hello world ” %>	输出字符串
9	也可以这样输出：	
10	<%	
11	ebprintf (ebfp, “%s ”, “*** hello world ***”);	C 语句
12	%>	
13	</body>	
14	</html>	

1.2 为什么要选择 eyBuild 开发 Web 站点

eyBuild 是基于 CSP 技术的开发套件的总称，它具有如下特点：

1. 规则简单，易学易用

   简单的说 CSP 就是直接 C 语句嵌入到 HTML 页面中，用 C 语言来响应请求、控制页面的动态输出行为，并通过简单的指令来控制页面的流程。用翻译器将 CSP 源文件翻译成 C 程序的文件。eyBuild 还提供了 MAP 工具，通过它可以非常容易将 CSP 源程序和其它非程序文件（如图片、静态 HTML 文件、CSS 文件）集成。

2. 嵌入式设备 WEB 应用开发的最佳选择

   目前嵌入式设备上的 WEB 应用最主要还是 C 语言直接开发 CGI 程序。除在序言中提到了直接用 C 开发 CGI 的诸多缺点外，直接将 HTML 代码用 C 语句输出，不能对两都进行有效的分离，将大大降低 HTML 源程序和 C 源程序的可读性、维护性；直接用 C 语言开发还必须手工对页面输出的流程进行控制，还将大大降低了开发的效率。

   eyBuild 提供的 CSP 翻译工具，将 CSP 源程序翻译成 C 语言源程序。开发都只要编辑好 HTML 页面模板，然后在适当地方嵌入 C 语句即可以有效控制页面的输出；上文提到，还可以通过指示指令对输出页面的进行控制。使得 WEB 界面的开发和执行控制动作进行了有效的分离。使得嵌入式设备的 WEB 开发变得高效、有序。

3. 服务器 WEB 应用开发工具的最有益补充

   用 eyBuild 开发的 WEB 应用程序可与任意服务器 WEB 应用进行有效的接合，因为 eyBuild 开发的 WEB 应用程序是一个独立的自治单元，它不依赖于 WEB Serever 或其它脚本程序。所以你可以在其它任意脚本程序中通过 URL 发起 eyBuild 开发程序。eyBuild 最终会将源文件生成 C 代码，它生成的应用具有极高的执行效率。这样，你就可以用 eyBuild 来开发所需要的脚本组件，以解决效率的瓶颈。

4. 模板选择有了更大的灵活性

   很幸运，你几乎可以以任意文本文件（如 HTML、XML、JS、CSS、TXT）为模板，在其中插入 CSP 语句。这一点与其它解释性脚本语言大不相同，这是因为 CSP2BIN 仅从中卒取 CSP 语句解析翻译，而视其它的数据为普通文本。这一特性，给 CSP 模板的选择大大增加了灵活性。

5. 通用、高效、跨平台性

   这是因为：（1）eyBuild会将CSP源文件及其它非程序文件翻译成C语言的源程序，所以CSP天然地继承了C语言的一切特性。（2）eyBuild库的实现是完全遵循CGI1.1标准，所以你可以在任意操作系统（支持：Windows、Linux、BSD Unix、Free BSD、Sun Solaris、VxWorks、WinCE、uC/OS-II）上任意遵循CGI1.1 标准的WEB服务器

（支持Apache，IIS，GoAhead，httpd，mini-httpd ……）下运行最终的CGI程序。

6. 上传文件更容易

   eyBuild专为大量数据传送（如提供一般表单项同时上传文件，见“在CSP中获取输入”）提供了一个API接口。使得多个文件上传，以及上传到服务器后的文件管理更加简单。同时还为特殊应用提供了场合提供了HOOKS接口。

7. eyBuild 提供了高效的运行库

   eyBuild 为不同操作系统平台提供了高效的运行库，可以直接在 CSP 中的 C 语句直接调用也可以在独立的 C 文件中调用。eyBuild 运行库类似于 cgilib，如获取用户输入参数、进行页面输出缓冲、处理文件上传)。

8. 使得脚本程序管理更简单

   eyBuild 提供了模块化处理功能，即将许多相关的 CGI 处理页面及相关文件（如：图片、CSS、.js 文件）生成到一个 cgi 执行文件中，并为每个处理页面或文件在 cgi 执行文件内部提供虚路径的映射。这样，每个最终生成的 cgi 文件就一个具有相关页面集合的功能单。甚至将整个站点生成在一个 cgi 文件中，或者作为一个独立的模块集成到其它应用程序中。这一点在站点的发布和维护有用上非常有用，应用实例见后文。

9. 源程序安全保密

   因为最终会将所有的 CSP 源文件生成 C 文件并最终编译成在一个二进制文件，而不像其它脚本程序一样由脚本解释器解释脚本源程序执行，所以并不存在源代码安全隐患。对源代码具有很好的保密性。

第2章 CSP 的语法

在 CSP 源文件中界于<%和%>之间的语句（可以跨越多行）都被称为 CSP 语句，CSP 翻译工具 CSP2BIN 将对<%和%>之间 CSP 语句进行解释。在 CSP 语句的任意字符都将被视为 HTML 源程序的语句，CSP2BIN 工具一般不对其进行解析。

2.1 CSP 的语句

CSP 语句包括单行语句和多行语句。凡<%和%>出现在同一个自然行的 CSP 语句称为单行语句；凡<%和%>跨越两个或两个以上的自然行的 CSP 语句称为多行语句。如下所示：

单行语句输出的例子：

```
1 单行语句输出：<% = “hello world<BR>” %>再输出一次<% = “hello world<BR>” %>
```

多行语句输出的例子：

```
1 <%
2 = “这是第二行<BR>”
3 = “这是第三行<BR>”
4 %>
```

2.2 语句格式

CSP 语句以行为解析单位，每一行作为一个独立的语句 CSP 语句行的格式如下：

CSP 语句总是以上边界符“<%”为开始，以下边界符“%>”为结束。CSP 语句内部每一个自然行为一个 CSP 语句行。CSP 翻译器以 CSP 语句行为单位对 CSP 语句进行解析。每一个语句行由三部分组成：前缀符、语句体、分号。如下所示：

1	<%	[前缀符]	[语句体 1]	[;]		
2		[前缀符]	[语句体 2]	[;]		
3		[前缀符]	[语句体 3]	[;]	%>	

CSP 语句行中的三部分全部省略时，称为空语句。空语句用于书写时的格式控制，翻译时翻译器将跳过空语句行。

2.3 CSP 的语句行前缀符

目前 eyBuild 对 CSP 的语句行前缀符定义如下：

前缀符	意 义
#	注释，# 后的内容不作任何解释
@	指令指示符，用于对 CSP 进行特殊控制，如包含头文件、包含其它 CSP 页面
=	输出语句，计算 = 后面的函数、变量或常量的值并作为字符串进行输出
%	调用页面内建函数，如%echo, %print (两者功能相同)
$	保留，未定义
?	保留，未定义
<	保留，未定义
>	保留，未定义
:	保留，未定义
备注：	其它可打印字符视为 C 语句的一部分，不作为前缀符解释，如&, !, ~

说明：

1. 当 CSP 语句行有前缀符时，行末的分号不是必须的，翻译器会跟据需要为最终果添加上必要的 C 语句结束符——分号（;）。
2. 当 CSP 语句行没有前缀符时，翻译器会将其作为 C 语言的语句直接输出，且不会自动在语句最后为其加上分号。
3. 类似于 ASP、JSP 等，CSP 也可以用等进行语句输出，等号后面可以是一个常量、变量或一个函数，但是值（如函数返回的值）必须是一个字符串。
4. 在 CSP 语句行中插入 C 语句时，必须注意行首字符不要跟 CSP 的前缀符冲突。一般地，按 K&R 著的 The C Programming Language 的 C 程序风格式，一般就不会与 CSP 语法冲突。

例 1：加与不加分号：

1	<% @ include <my_header.h> %>	语句最后**不必**加分号
2	<% %prv_func(“xx”, “yyy”); %>	语句最后**不必**加分号
3	<% %prv_func(“xx”, “yyy”); %>	. .
4	<% my_func (	语句最后**不能**加分号
5	parameter1,	. .
6	parameter2,	. .
7	parameter3	. .
8	);	必须手工加上分号
9	%>	

例 2：用 = 进行输出：

1	<% {	
2	char * pstr = “this is a test ”;	定义一个 C 变量
3	= "Build Date:" __DATE__ __TIME__ " "	输出常

4	= pstr	输出变量
5	= cgiPrefix()	输出函数
6	} %>	

2.4 CSP 的注释前缀符（#）

如果 CSP 语句行以前缀符“#”为开头，那么该行将被作为注释处理，即忽略#后面的任何内容。当然，也可以直接用 C/C++的注释：“/* xxx */”和“//”。它们的区别是，翻译器会将 C/C++的注释语句写入目录 C 程序中去，而“#”则注释语句则不会。注释行的语句体格式如下：

#	任意内容

例 1：CSP 的注释和 C/C++的注释

1	<%	
2	# This is CSP comment	CSP 注释
3	var++; /* this is C/C++ comment */	C/C++ 语句 + 注释
4	%>	

2.5 CSP 的指令前缀符（@）

包含前缀符“@”的语句行，被称为指令行。指令语句可以写在 CSP 源文件的任意处，因为它总是在翻译阶段被处理。指令行的语句体格式如下：

指令	分隔符	[']	指令的值	[']

1. CSP 的指令定义及其意义如下：

pagesize	设置页面缓冲区的大小，默认为 4K
autoflush	是否允许自动 Flush 缓冲区，默认为 TRUE
mimeheader	设置页面的 MIMI 头
require	指令所在处语句处插入另外一个页面
include	C 语言的文件包含语句，相当于 #include

2. 分隔符可以为：等号、空格、TAB，等号两边的空格或 TAB 将被忽略
3. 指令值两边的分号是可选的，即可以输入也可以不输入，但必须成对出现。

例 1：指令分隔符

1	<%	
2	@pagesize = 4K	用 = 作分隔
3	@pagesize 4K	用空格或 TAB 分隔
4	%>	

例 2：指令两边的分号

1	<%	
2	@ autoflush = “TRUE”	加上引号
3	@ autoflush = FALSE	不加引号
4	%>	

例 3：其它例子：

1	<%	
2	@mimeheader = “Content-Type=html/text\n”	MIME 第一句
3	@mimeheader = “Cache-Control: no-cache\n\n”	将连在上一句后面
4	@require “/demo/header.csp”	将另一页面插入在此处
5	@include <myheader.h>	包含头文件
6	%>	

2.6 CSP 的 require 指令

require指令用于在请求处插入另外一个页面。在翻译时仅是做了一个标记，并不把请求的文件原样复制到请求处。脚本程序执行时，会根据标记将请求页面的执行结果插入到请求处。被请求页面与请求页面共用一个请求页面的缓冲区，必要时可以通过pagesize指令设置请求页面的缓冲区大小。另外，require 的最大引用深度为 6 级。

需要特别说明的是，require 只能引用当前模块（即同一 CGI 脚本程序）中的已存在的页面，否则 C 连接器会报告找不到所需要的函数错误。

例 1：用 require 其它 CSP 引用页面

1	Hello world. This is an English Version	文件：/demo/body_en.csp
2	<Add any thing here>	

1	你好，这是一个中文版	文件：/demo/body_ch.csp

2	<添加其它内容>	

1	<html><body>	文件：/demo/main.csp
2	<%	
3	if (!strcmp(“English”, getParameter(“version”))	获取版本类型
4	@require “/demo/body_en.csp”;	引用英文版的内容
5	else	
6	@require “/demo/body_ch.csp”;	引用中文版的内容
7	%>	
8	</body></html>	

2.7 CSP 的输出前缀符（=）

类似于 ASP、JSP 等，CSP 也可以用等进行输出字符串常量、变量、和函数返回值，甚至空指针“NULL”。输出语句行的格式如下

=	任意 C 语句表达式	[;]

例 1：用输出前缀符进行输出

1	<%	
2	char * pstr = “Hello world. ”;	定义并初始化 C 变量
3	= “Build Date:” __DATE__ __TIME__ 	输出常字符串
4	= pstr	输出变量
5	= strchr(pstr, ‘w’)	输出函数返回值
6	= (TRUE==FALSE) ? “ERROR”: “RIGHT”	输出表达式的结果
7	= “ ”	
8	%>	

2.8 CSP 的内建函数前缀符（%）

前缀符“%”用于指示内建函数。目前支持的内建函数有：echo 和 print。翻译器翻译 CSP 源文件时会将 echo 或 print 替换成 ebprintf（ebfp, fmt, …）格式输出。目的是为了兼容书写习惯，同时简化输出语句。格式参考 C 标准函数 printf()。

1	<%	

2	% echo “hello word ”	定义并初始化 C 变量
3	% echo “%s”, “hello world ”	输出常字符串
4	% print “hello word ”	输出变量
5	% print “%s”, “hello world ”	输出函数返回值
6	%>	输出表达式的结果

2.9 在 CSP 中进行输出

从前文已经看到，这里只是做一些补充和总结：

1. 直接“=”前缀符输出
2. 用 %echo, %print 输出
3. 直接调用 eyBuildLib 中库函数 ebprintf 输出

注意：

无论在 CSP 源文件中还是在与基相关的 C 程序源文件中都不要直接用标准输出函数 printf()等进行输出，因为它们是不会对页面进行缓冲的，当后面出现错误时，错误信息将不能正确的输出。另外它们的输出还会打乱正常的输出次序，使输出结果中的次序错误。

2.10 在 CSP 中获取输入

通常，客户端通过 URL 和表单（FORM）两种形式进行提交请求。通过 URL 时采用的是 HTTP 的 GET 方法，通过 FORM 形式提交时我们可以指定是采用 GET 方法还是 POST 方法。在采用 POST 方法时，我可以通过“ENCTYPE”在表单中指明采用的编码方式为“application/x-www-form-urlencoded”或“multipart/form-data”（前者为默认的编码方式）。我们知道，在上传文件时必须采用“multipart/form-data”编码方式。

eyBuildLib 提供了两类 API 函数，它们分别用于：

（1）从 POST 方法下编码方式为“multipart/form-data”的提交方式取得数据（参阅 dopost()及其相关 API

（2）从前者以外的提交方式中取得数据（参阅 getParameter()及其相关 API）

下面给出向个例子以说明如何使用它们。

例1. 从 URL 中获得请求参数

```
1  <html><body>
2  连接到页面：
3  <a href="<% =cgiPrefix(NULL) %>/demo.csp&arg1=Hello+world">
```

```
4  demo.csp
5  </a>
6  </body></html>
```

```
1  <html><body>
2  这是在 demo.csp 页面中，参数  arg1=
3  <% =getParameter(“arg1”) %>
4  </a>
5  </body></html>
```

```
1  这是在 demo.csp 页面中，参数  arg1= Hello world
```

说明：

第一段程序用于显示在客户端，提交参数 arg1=Hello+world

第二段程序为服务器端程序

第三段为当点击了客户端的超级连接后在浏览器上的输出结果

其中的“cgiPrefix(NULL) ”用于设置 CGI 前缀。

例2. 通过表单提交

```
1  <form action="<% =cgiPrefix(NULL) %>/demo.csp" method="post">
2  请在输入: <input type="text" name="inputbox1"><br>
3  <input type="submit" name="testit" value="testit"><br>
4  </form>
```

```
1  <html><body>
2  你输入的是: <% =getParameter("inputbox1") %><BR>
3  </form>
```

例3. 从上传文件中获取输入

```
1  <form method=post action="<% =cgiPrefix(NULL) %>/demo/uploads.csp"
2  ENCTYPE="multipart/form-data">
3  选择文件 1：<input type="file" name="upload_file1" size=250><BR>
4  选择文件 2：<input type="file" name="upload_file2" size=250><BR>
5  上传到目录：<input type="text" name="destpath" value="c:/upload" size=100><BR>
6  <input type="submit" name="Send" value="Send">
7  </form>
```

文件/demo/uploads.csp:

```
<html>
<head><title>Upload Server</title></head>
<body>
<%
@ include <ebpost.h>
@ include <direct.h>
{
    POST_PARAM *    p_param = NULL;
    THIS_POST *     p_post;
    int             i, filecount;
    char *          destdir = NULL;

    /* create post parameters:
     *
     * extern POST_PARAM * createPostParam
     *     (
     *     int          max_post_size,      /@ max post size @/
     *     int          file_max_size,      /@ max size each file @/
     *     int          max_file_number,    /@ max file may uploads @/
     *     char *       upload_tmp_dir,     /@ where to store temp @/
     *     int          options             /@ other options @/
     *     );
     */
    if (NULL == (p_param=createPostParam(0, 0, 3, NULL, 0)))
        return ERROR;

    /* done post */
    if (NULL == (p_post=dopost(p_param, ebfp)) )
        return ERROR;

    mkdir (destdir=getParameter("destpath"));
    filecount = getPostFileCount(p_post);
    for (i=0; i < filecount; i++)
    {
        int     ret = movePostFile(p_post, i, destdir);

%>
    <table>
        <tr><td>源地址:</td>
            <td><%% print("%s", getPostSrcFileName(p_post, i)); %></td>
```

```
            <tr><td>存储到目录: </td>
                <td><% = destdir %></td>
            <tr><td>状态:</td>
                <td><%% print("%s", (OK==ret ? "OK": "ERROR")); %></td>
    </table>
<%
    } /* end for */
    distoryPost(p_post, TRUE);
    p_post = NULL; p_param = NULL;
} /* end */
%>
```

执行结果举例（类似于）：

```
<html>
<head><title>Upload Server</title></head>
<body>
<table>
    <tr><td>源地址:</td>
        <td>C:\cspc.c</td>
    <tr><td>存储到目录: </td>
        <td>E:/uploads</td>
    <tr><td>状态:</td>
        <td>OK</td>
</table>
<table>
    <tr><td>源地址:</td>
        <td>c:\demo.c</td>
    <tr><td>存储到目录: </td>
        <td>E:/uploads</td>
    <tr><td>状态:</td>
        <td>OK</td>
</table>
</table>
```

说明：

第一段程序用于显示在客户端
第二段程序为服务器端程序
第三段为当客户端提交两个文件(cspc.c，demo.c)并指定目标目录为“E:/upload”时的输出结果

2.11 CGI 前缀和 ROM 前缀

什么是 CGI 前缀和 ROM 前缀？

CGI 前缀和 ROM 前缀分别是指 eyBuildLib 的两个 API 函数 cgiPrefix()和 romPrefix()。它们的函数原型为：

```
char * cgiPrefix(char * cginame);
char * romPrefix(char * cginame);
```

参数cginame是指可执的CGI文件的文件名。

函数cgiPrefix()和romPrefix()返回以cginame为基础构成的前缀，若cginame为NULL，则返回用自身URL构成的前缀，如（以/cgi-bin/demo.cgi中执行结果为例）：

```
1 <%                                          结果
2 = cgiPrefix(NULL);                          demo.cgi?cgi=
3 = cgiPrefix(“/cgi-bin/test.cgi”);           /cgi-bin/test.cgi?cgi=
4 = romPrefix(NULL);                          demo.cgi?file=
5 = romPrefix(“/cgi-bin/test.cgi”);           /cgi-bin/test.cgi?file=
6 %>
```

为什么要用 CGI 前缀和 ROM 前缀，如何使用它们？

一般地我们会将一系列相关的CSP源程序文件、CSS文件、图片文件等编译后生成一个独立的CGI文件（或称其为一个模块）。这个模块具有很高的自治性。当我们需要引用其中的文件时，可以通过CGI/ROM前缀来引用指定模块的名称，通过前缀后面的页面名称指定其中的子文件。子文件引用本模块的子模块时，不需要指定模块名称。下面给几个例子予以说明：

设：x.cgi 是由 a.csp、b.csp两个CSP文件和一个图片文件 c.jpg 所生成。
y.cgi 是由 d.csp 和图片文件 e.jpg 两个文件组成。

例 1. 在 d.csp 中引用 e.jpg 和 a.csp、b.csp、和 c.jpg，d.csp 的源程序如下：

```
1 <html><body>
2 显示图片 e.jpg: <img src=”<% =romPrefix(NULL) %>/e.jpg”>
3 <br>
4 显示图片 x.cgi->c.jpg: <img src=”<% =romPrefix(“x.cgi”) %>/e.jpg”>
5 <br>
6 <a href=”<% =cgiPrefix(“x.cgi”) %>/a.csp>转到 x.cgi->a.csp</a>
```

```
7 <br>
8 <a href=”<% =cgiPrefix(“x.cgi”) %>/b.csp>转到 x.cgi->b.csp</a>
9 </body></html>
```

当该客户端通过 URL（类似于 http://xx.yy.com/cgi-bin/y.cgi?cgi=d.csp）取得 d.csp 的页面时，其结果将为：

```
1 <html><body>
2 显示图片 e.jpg: <img src=”y.cgi?file=/e.jpg”>
3 <br>
4 显示图片 x.cgi->c.jpg: <img src=” y.cgi?file=/e.jpg”>
5 <br>
6 <a href=”y.cgi?cgi=/a.csp>转到 x.cgi->a.csp</a>
7 <br>
8 <a href=”y.cgi?cgi=/b.csp>转到 x.cgi->b.csp</a>
9 </body></html>
```

说明：

第一段程序为工作在服务器端的 CSP 源程序

第二段是输出到客户端浏览器的结果，即当该客户端通过 URL（类似于 http://xx.yy.com/cgi-bin/y.cgi?cgi=d.csp）取得 d.csp 的页面时输出的结果。

上传给出了如何通过 CGI/ROM 前缀取得指定的文件。其中第二行是引用当前模块中的文件，第四、六、八行引用了别的 CGI 模块中的文件。

注意：

以 CGI 前缀为开头的文件，必须为可执行的 CSP 文件，以 ROM 前缀为开头的可以是任意文件，如图片、CSS、文本文件。

在用 CGI/ROM 前缀时不要忘记前面的等号，它表示将函数返回值输出在当前位置。

第3章　eyBuild 开发环境简介

3.1　安装 eyBuild

目前 eyBuild 还没有针对各种开发平台的集成开发环境（IDE），不过它的安装和设置也十分简单。一般只需要如下简单的两步操作，下面以 Windows 环境为例：

1. 将 eyBuild 开发包 eyBuild-xx-yy-zz.tar.gz 解压到一个目录，如 E:\eyBuild
2. 设置环境变量 EYBUILD_BASE=E:\eyBuild，方法是：
   A. 从菜单“开始→设置→控制面板→系统”打开系统属性对话框
   B. 从“高级”选项卡上点击“环境变量”按钮
   C. 可根据需要在“用户变量”或“系统变量”将 EYBUILD_BASE 正确添加进去

3.2　eyBuild 目录结构

下面是 Windows 环境下安装好了的目录结构：

```
EYBUILD
├──bin
│       csp2bin.exe
│       donemap.bat
│       web2bin.exe
│       wintar.exe
│
├──demo
│   │   cspmap.map
│   │   demo.bat
│   │   rommap.map
│   │
│   ├──csp
│   │       main.csp
│   │       test.csp
│   │
│   ├──cspsrc
│   │   │   demo_csp_maplist.c
│   │   │   demo_rom_maplist.c
│   │   │
│   │   ├──csp_p
│   │   │       _demo_main_csp.c
│   │   │       _demo_test_csp.c
│   │   │
```

```
│   │   └─usr
│   ├─img
│   │       face.gif
│   │
│   └─prj
│           demo.c
│           demo.dsp
│           demo.dsw
│
├─doc
│   ├─API
│   │       dopost.html
│   │       ebfrm.html
│   │       ebio.html
│   │       ebrespond.html
│   │       libIndex.htm
│   │       rtnIndex.htm
│   │
│   └─manual
│           eyBuild 用户手册.doc
│
├─include
│       ebfrm.h
│       ebio.h
│       eblib.h
│       ebpost.h
│       ebrequest.h
│       ebrespond.h
│       itemlist.h
│
└─lib
        eybuild.lib
```

从上图可以看出，本版 eyBuild 开发环境下包括 5 个主目录：

1. bin　　bin 目录是 eyBuild 所有需要的工具包
2. demo　　是一个示例程序
3. doc　　关于 eyBuild 的文档
4. include　　C 头文件所在的目录
5. lib　　库文件所在的目录

说明：

1. CSP 源程序可以由 WEB 程序员先编辑好 HTML 模板，再由 CGI 程序在模板上嵌入相关的程序
2. 可以根据平台需要可自由选择 C 程序编译器和链接器，如 gcc，VC++等均可。需要

说明的时，必须用与之平台对应的 eyBuildLib 库。

3.3 什么是虚目录

什么是虚目录，为什么要用虚目录？

虚拟是指在跟在 CGI/ROM 前缀后面的目录名，它用于区分同名的 CSP 或 ROM 文件。如<% =cgiPrefix(NULL) %>/demo/demo.csp 和 <% =cgiPrefix(NULL) %>/rt/demo.csp，其中的/demo/和/rt/就是两个虚目录。因为在最后生成的 CGI 文件中并不存在目录，所以称它为虚目录。同时虚目录还可以有效地隐藏源文件目录的一些细节。

3.4 MAP 文件

在开发工具的安装目录结构中我们注意到，demo 目录下有个 cspmap.map 文件。这个就是一个 map 文件。它是项目的配置文件，用于指示哪些目录或源文件需要被翻译，翻译后它们将使用什么名字的虚目录（见上一节）。

MAP 文件的语法格式：

1. 解析时以行为单位，一行最多不能多于 255 字符
2. 解析时空行和每行开头的空格、制表位被忽略
3. 以“!”为开头的行被视为注释行
4. 映射行格式：
   （1）文件名==> 虚目录
   （2）目录名==> 虚目录
5. 虚目录可以省略（但“==>”不可以省略），省略时被默认为“/”
6. 映射行若以目录名为开头，目录名最后必须以“\”或“/”结束
7. 文件名可以包括通配置符“*”和“?”
8. “目录名==> 虚目录”表示，下面的若干行是以该目录为搜索路径，并指定的文件映射到这个虚目录下

例 1．映射文件

1	! This is comment	注释语句
2	d:/demo/csp/X.csp ==> X/	映射文件到虚目录“X/”下
3	d:/demo/csp/Y.csp ==> /Y/	映射文件到虚目录“/Y/”下
4	d:/demo/csp/Z.csp ==> /Z_	映射到根“/”并加下前缀 Z_
5	d:/demo/csp/AA.csp ==> /	映射到根“/”
6	d:/demo/csp/BB.csp ==>	省略时同结果上

这样，我们可以在 CSP 文件中这样引用它们了：<% cgiPrefix(NULL) %>/

1		备注
2	<% cgiPrefix(NULL) %>X/X.csp	对应上面的第二行

3	<% cgiPrefix(NULL) %>/Y/Y.csp	
4	<% cgiPrefix(NULL) %>/Z_Z.csp	
5	<% cgiPrefix(NULL) %>/AA.csp	
6	<% cgiPrefix(NULL) %>/BB.csp	

例 2．使用配符“*”或“?”

1	d:/demo/*.csp ==> /demo/	映射扩展名为“.csp”的文件
2	d:/test/*.html ==> /demo/	映射到同一目录

假设：
在目录 d:/demo/ 中存在文件 X.jpg、Y.csp
在目录 d:/test/ 中存在 X.html、X.html
那么我们就可以这样用如下方法引用它们了，好像它们来自同一真实目录一样：

1		备注
2	<% romPrefix(NULL) %>/demo/X.jpg	作为 ROM 类型文件引用
3	<% romPrefix(NULL) %>/demo/Y.csp	
4	<% romPrefix(NULL) %>/demo/X.html	
5	<% romPrefix(NULL) %>/demo/Y.html	

例 3．映射目录到虚目录

1	d:/demo/ ==> /demo/	**指明目录映射**
2	X.html	列举文件
3	Y.css	列举文件
4	d:/test/ ==> /demo/	**指明目录映射**
5	A.txt	列举文件
6	B.doc	列举文件

引用方法跟上面的例子一样。

3.5 WEB2BIN

工具 WEB2BIN 用于把 map 文件中所列的文件，全部翻译成 C 代码。将它与其它 C 源程序一起，用 C 编译器编译到最终的可执行脚本文件中去。当通过 URL 获取同名的文件时，脚本程序再把它原样地输出给请求的客户端程序。

这个工具很有用，是因为我们可以用它将不可执行的文件（如 JPEG、GIF、CSS 等）

翻译成 C 代码，并在编译到最终的脚本文件中去。将它作为模块的一部分，当我们需要移动它们时，只移动一个可执行的脚本文件。

WEB2BIN 的命令行格式如下：
web2bin [map-file] [map-list-name] [output]
map-file 必须的，它用于指示从哪儿读取 MAP 文件
map-list-name 可选的，生成的列表名称，即生成的 C 代码的列表
output 可选的，将结果输出到该文件

如果不指明 map-list-name 和 output，那么将会依据 map-file 的文件自动生成为它们生成一个合法的字名

3.6 CSP2BIN

顾名思义，CSP2BIN 就是用于把 map 文件中所列的文件，全部翻译成 C 代码的工具。

CSP2BIN 的命令行格式如下：
csp2bin [map-file] [map-list-name] [dest dir]
map-file 必须的，它用于指示从哪儿读取 MAP 文件
map-list-name 可选的，生成的列表名称，即生成的 C 代码的列表
dest dir 指出翻译的结果存放在哪个文件夹中

3.7 DONEMAP

一般可以直接使用 CSP2BIN 和 WEB2BIN 等工具。为了简单操作，我们也可以使用 DONEMAP 工具为我们快速配置翻译环境。配置 DONEMAP 工具，即指定如下五个变量的值：

1	DEST_DIR	结果存放的目录
2	CSP_MAPFILE	CSP MAP 文件名
3	CSP_MAPLIST	CSP 文件列表名
4	ROM_MAPFILE	ROM MAP 文件名
5	ROM_MAPLIST	ROM 文件列表名

说明：

1. DEST_DIR 是必须的，它指示生成的文件存放的目录
2. CSP_MAPFILE、ROM_MAPFILE 之一可以为空，它表示没有 CSP 文件或 ROM 文件需要翻译
3. 通常，可以修改下面的脚本文件（以 Windows 下的批处理文件 domap.bat 为例，表格第二行中所列的五变量）来配置翻译环境
4. REM 表示注释，SET 用于设置环境变量
5. 注意：等号“=”前后不要随意加空格，空格不会自动删除

domap.bat 模板文件

```
1  @echo off
2  REM ###########################################################
   set DEST_DIR=destdir

   set CSP_MAPFILE=cspmap.map
   set CSP_MAPLIST=demo_csp_maplist
   set ROM_MAPFILE=
   set ROM_MAPLIST=
   REM ###########################################################
3
   if not defined EYBUILD_BASE (
       echo Not defined "EYBUILD_BASE".
       pause & goto :EOF
   )

   REM convert file
   call %EYBUILD_BASE%/BIN/DONEMAP.BAT %CD% %~dp0 %1
   echo on
```

3.8 脚本的入口 cgimain()

最终生成的可执行脚本的入口是 cgimain()，它相当于应用程序的 main()主口。该函数必须由用户定义，用户可以通过它来控制后文脚本的处理行为。

一般地，cgimain()要完成如下功能：

（1） 设置调试环境，如调试时预置CGI环境变量，设置程序异常中断等

（2） 设置模块的主页面，即当不指定任何请求页面时返回的页面。未指定主页时，默认地为/main.csp。主页的设置方法是，设置环境变量 MAIN_PAGE，例如：putenv("MAIN_PAGE=/demo/main.csp");

（3） 启动 eyBuild 环境，响应客户端请求。即调用函数 eyBuildExec()，基函数原型为：

```
extern int eyBuildExec
    (
    void *      cgitab,     /* cgi pages table */
    void *      filetab,    /* rom files table */
    void *      ebenv       /* ebuild environment, NULL is to set */
    );
```

（4） 下面是 cgimain()的模板，应用时根据需要对局部（一般将必须将 xxx 换成你项目中的名字）稍微改变即可。

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdLib.h>
4  #include <ebrequest.h>
5  #include <ebrespond.h>
6
7  int cgimain()
8  {
9      extern EB_CSP_MAPLIST    xxx_csp_maplist[];
10     extern EB_ROM_MAPLIST    xxx_rom_maplist[];
11 #if 0 /* for debug */
12     int                      v=0, *p1=&v, *p2=NULL;
13     *p2 = 0;
14 #endif
15
16     ebSetDebug("GET", "cgi=/demo/main.csp"); /* set debug environment */
17
18     putenv("MAIN_PAGE=/demo/main.csp");    /* set main-page */
19
20     /* respond require */
21     eyBuildExec(xxx_csp_maplist, xxx_rom_maplist, NULL);
22     return OK;
23 }
```

说明：

(1) 1~9 行是头文件包含

(2) 9~10 两行，声明了外部定义的 maplist。它们通常在 DONEMAP 配置环境中指定，并被动态生成。

(3) 11~14 行，用于设置异常中断调试，一般不使用

(4) 16 行，设置调试环境。手工设置 HTTP 的 QUERY_STRING 和 REQUEST_METHOD。

(5) 21 行，由 eyBuild 响应请求。

第4章　建立工程并生成 CGI 可执行文件

4.1　建立工程的一般步骤

创建一个工程并生成 CGI 文件通常需要如下几步：

1. 编辑 CSP 源程序及其相关的 C 程序源文件和 ROM 文件（如: 图片、CSS 文件、JavaScript 文件等）
2. 配置翻译器（map 文件和 DONEMAP 环境）
3. 将 CSP 文件和 ROM 文件翻译成 C 源程序文件
4. 用 C 编译器第一步编写的 C 程序文件和第三步生成的 C 程序文件编译成目标文件。用链接器将目标文件与库 eyBuildLib 一起链接，生成可执行的 CGI 文件。

说明：

1. CSP 源程序可以由 WEB 程序员先编辑好 HTML 模板，再由 CGI 程序在模板上嵌入相关的程序
2. 可以根据平台需要可自由选择 C 程序编译器和链接器，如 gcc，VC++等均可。需要说明的时，必须用与之平台对应的 eyBuildLib 库。

4.2　示例

下面给出 Windows 环境下的一个示例，以演示如何建一个新的工程、生成可执行文件并 WEB 服务器下测试它。

4.2.1 环境准备：

操作系统：	Windows 2000 或 Windows XP
Web Server：	安装好 IIS4.0 以上 或 Apache 1.3 或 Apache 2.x
C 编译器：	安装好 VC++ 6.0
eyBuild：	安装好eyBuild 00.06.00 或以上版本

4.2.2 源文件编写

（1）为工程新建立如下目录：

demo	根目录
├──csp	存放 CSP 源文件
├──cspsrc	存入 CSP 翻译生成的及其相关的 C 源文件
└──prj	存放项目文件

（2）在 demo/csp/目录中建立两个 CSP 源文件，分别为 main.csp 和 test.csp，并分别为

他们添加如下内容：

文件: demo/csp/main.csp

```
1  <html><body>
2  Hello world. <img src=”<% =romPrefix(NULL) %>/img/face.gif”>
3  <BR>
4  <form action="<% =cgiPrefix(NULL) %>/demo/test.csp" method="post">
5  请在输入: <input type="text" name="inputbox1"><br>
6  <input type="submit" name="testit" value="testit"><br>
7  </form>
8  </body></html>
```

文件: demo/csp/test.csp

```
1  <html><body>
2  This is in test.csp<BR>
3  你在文本框输入的内容为：<% =getParameter(“inputbox1”) %><BR>
4  [ <A href=”<% =cgiPrefix(NULL) %>/demo/main.csp”>返回</A> ] <BR>
   </body></html>
```

4.2.3 配置翻译器

（1）在 demo/目录中建立两个 MAP 文件，分别为 cspmap.map 和 rommap.map，并分别为他们添加如下内容（注意，仅第 5 行有区别）：

文件: demo/cspmap.map

```
1  # this file list how to map files to virtual directory
2
3  # source directory              virtual directory
4  # -----------------            -----------------
5  csp\*.csp                ==>       /demo/
6
```

文件: demo/cspmap.map

```
1  # this file list how to map files to virtual directory
2
3  # source directory              virtual directory
4  # -----------------            -----------------
5  img\*.*                  ==>       /img/
6
```

（2）在demo/目录中建立一个批处理文件demo.bat。方法是复制domap.bat的内容并把下表显示的部分修改成如下内容（突出显示的部分）：

```
2  REM ########################################################
   set DEST_DIR=cspsrc

   set CSP_MAPFILE=cspmap.map
   set CSP_MAPLIST=demo_csp_maplist
   set ROM_MAPFILE=rommap.map
   set ROM_MAPLIST=demo_rom_maplist
   REM ########################################################
```

4.2.4 翻译 CSP 文件和 ROM 文件

上一步已经配置好了翻译环境，现在就可以直接双击 demo.bat 进行翻译文件了。如果在 CSP 源文件中有错误，翻译器会输出错误的行号和错误的原因。下面是屏幕上输出的结果：

```
1  ########## Build csp ##########
   /demo/main.csp
   /demo/test.csp
   2 files convert OK.

   ########## Build romfile: ##########
   /img/face.gif
   1 files convert OK.
   请按任意键继续. . .
```

再让我们看翻译的结果，即目录 demo/cspsrc/下的目录结构：

```
DEMO
|   cspmap.map
|   demo.bat
|   rommap.map
|
├──csp
|       main.csp
|       test.csp
|
├──cspsrc
|   |   demo_csp_maplist.c
```

```
│   │   demo_rom_maplist.c
│   │
│   ├──csp_p
│   │       _demo_main_csp.c
│   │       _demo_test_csp.c
│   │
│   └──usr
├──img
│       face.gif
│
└──prj
```

容易发现：demo/cspsrc 目录中多了两个文件夹（csp_p/和 usr/）和两个 C 文件。两个 C 文件的名字正是在 dome.bat 所指定的。

说明：

（1）csp_p 后面的_p 意思是说私有的（private），用户一般不用关心的部分。用户不能直接编译其中的 C 源程序文件，因为它们都是动态生成的，每次翻译后它们的内容将会改变。

（2）usr 目录，顾名思义它代表用户自己维护的目录。用户可以直接将与翻译结果相关的 C 源文件（如 CSP 中引用的用户自定义函数等）放在此目录中，此目录中的内容只有用户才可以改变它们，翻译器只是帮助创建这个目录。

4.2.5 创建项目编译环境

（1）打开 VC++，在 prj 目录中创建一个空的 Win32 Console Application 项目。项目名为 demo.dsp

（2）将/demo/cspsrc/demo_csp_maplist.c 和/demo/cspsrc/demo_rom_maplist.c 添加到工程中（注意，不要将/demo/cspsrc/csp_p/中的文件添加到项目）

（3）添加头文件搜索路径，方法是：打开 Project → Setting →C/C++ 选项卡，在分类（Category）下拉框中选择 Preprocessor，并在 Additional include directories:中添加：$(EYBUILD_BASE)/include

（4）添加 eyBuildLib 库的搜索路径，方法是：打开 Project → Setting →Link 选项卡，在分类（Category）下拉框中选择 General，并在 Object/library modules:输入框最后（项与项之间用空格分隔）添加项$(EYBUILD_BASE)/lib/eybuild.lib

（5）最后，还必须手工添加一个 cgimain()。方法是，新建一个空的 C 文件/demo/prj/demo.c，将 cgimain() 模板中的内容粘贴到该文件中去。注意，别忘了将其中的“xxx”改为“demo”。

至此，就可以编译项目了。编译完毕，默认的会在 demo/prj/debug 目录生成一个可执行文件 demo.exe。

我们也可以用上面添加 eyBuildLib 库的搜索路径的方法，在同一选项页里的“Output file name:”输入框里改变量输出文件名字和路径（如改为：d:/cgi-bin/demo.cgi）。

4.2.6 运行结果

好了，至此我们就可以测试下上面的成果了（以 d:/cgi-bin/为例）：

（1） 在 IIS 或 Apache 下配置好了目录 d:/cgi-bin/具有执行执行权限，并重新启动
（2） 将编译结果结果（即 demo.cgi）生成到了 d:/cgi-bin/目录中，或拷贝到该目录。
（3） 打开IE 浏览器，在URL中输入地址：http://127.0.0.1/cgi-bin/demo.cgi

说明：
有关如何在 IIS 或 Apache 下运行脚本，请参阅相关手册。

第5章　调试

eyBuild 开发环境本身没有提供调试环境，但根据 C 编译器依然有目前有两种方法可以对生成的脚本程序进行调试：用 ebSetDebug()调试、异常中断调试。

5.1　用 ebSetDebug()调试

就是在 cgimain()中预置 CGI 变量，然后逐步根跟踪调试，这种方法虽然具有一定的局限性（如调试上传文件），但已经能满足大多数需要。

5.2　异常中断调试

这种方法是种非常极端的方法，一般不建议采用，但它却是运行时调试最有效的方法。原理是，故意构造异常（如让指针引用 0 地址下的变量），然后让操作系统和调试环境（如 VC++）捕捉该异常并暂停（如同设置了一个断点）。通过将修改寄存器的值，使程序能够进一步运行，然后在单步跟踪程序进行调试。