

Создание и обработка Web Form с CGI

Carl Sassenrath
REBOL Technologies

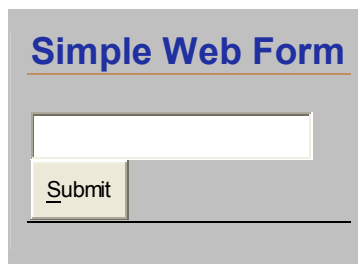
Простая Web Form

HTML Код

Этот код HTML создаст очень простую форму, которая включает область для ввода текста и кнопку:

```
<HTML>
<TITLE>Simple Web Form</TITLE>
<BODY>
<H2>Simple Web Form</H2>
<FORM ACTION="http://www.reboltech.com/cgi-bin/rebol/test.cgi">
<INPUT TYPE="TEXT" NAME="Field" SIZE="25"><BR>
<INPUT TYPE="SUBMIT" NAME="Submit" VALUE="Submit">
</FORM>
</BODY>
</HTML>
```

Сохраните этот HTML код в файл *simple.html*, откройте в браузере. На экране будет это.



Тестирование Web Form

HTML FORM тэг показанный ниже, определяет как должен быть обработан скрипт:

```
<FORM ACTION="http://www.reboltech.com/cgi-bin/rebol/test.cgi">
```

Содержит URL для простого REBOL сценария, который выполняется на одном из серверов REBOL's. Отображает web-страницу которая, показывает значения входных полей, представленных в форме.

Введите некоторый текст в форму, и нажмите кнопку Submit. Вы должны увидеть ответ от сервера www.reboltech.com, который выглядит как приведенный ниже рисунок:

CGI Form Data:

Submitted: 12-Apr-2003/12:53:01-7:00
REBOL Version: 2.5.5.4.2

Field	"Testing"
Submit	"Submit"

test.cgi URL удобен для того, чтобы проверить любой из ниже приведенных примеров.

Обработка Web Form

Этот раздел описывает, как использовать сценарии, чтобы обрабатывать web формы.

Настройки CGI Data

Вот - короткий REBOL сценарий интерфейса, который желательно запомнить:

```
#!rebol -cs
REBOL []
print "Content-type: text/html^/"
print [<HTML><BODY><PRE> mold system/options/cgi </HTML>]
```

Этот сценарий будет печатать всю информацию интерфейса, которая передается от сервера до REBOL. Получает информацию от объекта system/options/cgi REBOL.

Чтобы видеть, как это работает, меняйте тэг FORM в вышеупомянутом HTML в строке:

```
<FORM ACTION="http://www.reboltech.com/cgi-bin/rebol/probe.cgi">
```

Сохраните файл, и посмотрите это в вашем браузере. Напечатайте некоторый текст и нажмите кнопку. Вы увидите web-страницу, которая напоминает это:

```
make object! [
  server-software: "Apache/1.3.23 (Unix)"
  server-name: "www.reboltech.com"
  gateway-interface: "CGI/1.1"
  server-protocol: "HTTP/1.1"
  server-port: "80"
  request-method: "GET"
  path-info: none
  path-translated: none
  script-name: "/cgi-bin/rebol/probe.cgi"
  query-string: "Field=testing&Submit=Submit"
  remote-host: none
  remote-addr: "10.10.1.1"
  auth-type: none

  remote-user: none
  remote-ident: none
  Content-Type: none
  content-length: none
  other-headers: ["HTTP_ACCEPT" {image/gif, image/x-bitmap,...}]
]
```

Как видите, объект system/options/cgi REBOL. Это вся информация, которая передана от сервера до REBOL. Поскольку вы видите, CGI объект включает информацию об имени сервера и версии, методе запроса интерфейса компьютерной графики, пути сценария интерфейса компьютерной графики, данные которые были предоставлены (строка запроса), IP адрес браузера (отдаленный адрес), и т. д.

В скриптах вы должны изменить первую строку, чтобы обеспечить правильный путь к REBOL/Core. Другими словами, везде указываем путь к интерпретатору:

```
#!rebol -cs
```

Что-то вроде этого :

```
#!/home/myaccount/rebol -cs
```

Показ результатов CGI Form

Объект CGI содержит много информации. Часть этого может быть полезна для вашего сценария, но не всё. Вы можете поменять сценарий под свои нужды.

В примере ниже, данные могут быть в поле QUERY-STRING объекта CGI. Вы можете легко переделать вышеупомянутый сценарий:

```
#!/rebol -cs
REBOL []
print "Content-type: text/html^/"
print [
  <HTML><BODY><PRE>
    mold system/options/cgi/query-string
  </HTML>
]
```

Декодирование данных Web Form

Обратите внимание в сценарии ниже, данные Web Form специально закодированы, чтобы разрешить передачу назад к сценарию CGI.

REBOL обеспечивает функцию DECODE-CGI, чтобы облегчать декодирование данных формы CGI. (Я не буду описывать кодирующий метод здесь, но если вы интересуетесь, вы можете найти информацию об этом <http://www.xcf.berkeley.edu/help-sessions/cgi/x159.html>.)

Функция DECODE-CGI преобразовывает необработанные данные формы в блок REBOL, который содержит слова, сопровождаемые их значениями. Например данные CGI:

```
name=Fred&class=101&math=2+%2B+2+%3D+4&
```

был бы конвертирован DECODE-CGI в:

```
[name: "Fred" class: "101" math: "2 + 2 = 4"]
```

Вы можете сделать это сами вызвав функцию DECODE-CGI:

```
>> decode-cgi {name=Fred&class=101&math=2+%2B+2+%3D+4&}
== [name: "Fred" class: "101" math: "2 + 2 = 4"]
```

В более новых версиях REBOL (Core 2.5.5 и выше), сеть вводит имена полей, которые используются не раз и объединяет в блочное значение. Например:

```
name=Fred&status=good&status=happy&
```

возвращает:

```
[name: "Fred" status: ["good" "happy"]]
```

Это полезно подобно переключателям, где результат может быть выбран больше чем один.

Обратите внимание, что блок, возвращенный DECODE-CGI может использоваться, чтобы создать объект REBOL, который облегчает возможность обращаться к результатам Web Form. Это будет описано более подробно ниже.

Используйте правильные слова для имен полей

Когда используете функцию DECODE-CGI, вы должны убедиться, что входные имена полей - допустимые REBOL слова. Безопасно использовать только буквы или символы, сопровождаемые числами, но не начинать слово с цифры (0-9) и избегайте использовать специальные знаки (кроме "-") в пределах слова.

Верно:

```
<INPUT TYPE="TEXT" NAME="oneway">  
<INPUT TYPE="TEXT" NAME="one-way">
```

Ошибочно:

```
<INPUT TYPE="TEXT" NAME="1way">  
<INPUT TYPE="TEXT" NAME="1-way">  
<INPUT TYPE="TEXT" NAME="one ;way">
```

Пример test.cgi

Вот - REBOL сценарий CGI, который обрабатывает форму, рассмотренную выше и отображает декодированный результат в таблице:

```
#!rebol -cs  
REBOL []  
print "Content-type: text/html^/"  
  
html: make string! 2000  
emit: func [data] [repend html data]  
  
emit [  
  <HTML><BODY BGCOLOR="#FFC080">  
  <H2> "CGI Form Data:" </H2>  
  "Submitted: " now <BR>  
  "REBOL Version: " system/version <P>  
  <TABLE BORDER="1" CELLSPACING="0" CELLPADDING="5">  
]  
  
foreach [var value] decode-cgi system/options/cgi/query-string [  
  emit [<TR><TD> mold var </TD><TD> mold value </TD></TR>]  
]  
  
emit [</TABLE></BODY></HTML>]  
print html
```

Обратите внимание, что большинство кода используется, чтобы создать HTML, который будет отображен вашим web-браузером. Описание сценария:

- Строка создана для обработки HTML.
- Функция EMIT формирует HTML для вывода в браузер. Функция REPEND используется для

добавления новых данных в конце кода HTML. Эта функция будет использоваться часто, поэтому ее стоит запомнить.

- Функция DECODE-CGI декодирует поля web form в блок (как описано ранее).
- Цикл FOREACH создает строки в таблице для каждого слова и значения которое было возвращено.
- Оператор PRINT выводит финальный код HTML в браузер.

Результат:

CGI Form Data:

Submitted: 12-Apr-2003/12:53:01-7:00
REBOL Version: 2.5.5.4.2

Field	"Testing"
Submit	"Submit"

GET и POST метод

GET метод

Важно знать, что примеры web form показанные выше используют HTTP, **GET request method** для предоставления данных серверу.

GET метод посылает результаты серверу, кодируя их как часть URL. Вы наверное заметили эти URLs, когда использовали web-браузер для посещения сайтов в поисковых системах. Пример ниже:

```
http://www.google.com/search?as_q=&num=50&hl=en&btnG=Google+Search&...
```

GET method хорошо работает там где отправляется малый объем данных, длинные URLs будут ограничены.

Решение этой проблемы - тема следующего раздела.

POST метод

Лучший подход для больших **HTTP - POST request method**. Этот метод направляет данные к стандартному входному порту сценария CGI, разрешая сценарию CGI читать это напрямую.

Формат данных, посланных с **POST method** тот же самый, поскольку это послано с GET методом. Вы будете должны использовать **DECODE-CGI**, чтобы декодировать ваши данные web form (как показано ранее).

Чтобы использовать post метод, Вы должны добавить POST атрибут к тэгу FORM, в вашем HTML файле. Например:

```
<FORM ACTION="http://www.reboltech.com/cgi-bin/rebol/test.cgi"  
METHOD="post">
```

Разрешает серверу переадресовывать строку запроса к сценарию CGI входного порта.

Чтобы читать данные от стандартного входного порта, вашему сценарию CGI нужен цикл READ-IO как здесь:

```
data: make string! 1020
buffer: make string! 16380
while [positive? read-io system/ports/input buffer 16380][
  append data buffer
  clear buffer
]
```

Читает данные сети от входного порта и добавляет в конец каждую порцию данных к строке DATA. Эта строка может использоваться тем же самым способом как строка запроса, показанная выше, вызовом функции DECODE-CGI, чтобы декодировать входные данные.

Text/Binary Преобразование

Входной порт для сценариев CGI открыт в текстовом режиме, и признаки конца линии будут автоматически преобразованы к стандарту newline формат, используемый REBOL (символ LF).

Если вы читали binary данные от входного порта, вы должны изменить входной режим порта, вызвав функцию SET-MODES:

```
set-modes system/ports/input [binary: true]
```

Обратите внимание, что это обычно не требуется для ввода CGI, потому что binary символы кодируется как шестнадцатеричный текст, который расшифрован функцией DECODE-CGI.

Удобная функция чтения CGI

Вместо выбора между GET или POST methods, функция показанная ниже обрабатывает оба метода в пределах вашего сценария:

```
read-cgi: func [
  ;Read CGI data. Return data as string or NONE.
  /local data buffer
][
  switch system/options/cgi/request-method [
    "POST" [
      data: make string! 1020
      buffer: make string! 16380
      while [positive? read-io system/ports/input buffer 16380][
        append data buffer
        clear buffer
      ]
    ]
    "GET" [data: system/options/cgi/query-string]
  ]
  data
]
```

Ваш сценарий может вызвать READ-CGI чтобы получить строку запроса независимо от выбранного метода GET или POST. Дальше во все примеры будет включена эта функция.

Модернизация скрипта test.cgi

Функцию READ-CGI можно добавить к сценарию test.cgi, чтобы работало с любой длиной предоставленной web-form.

```
#!rebol -cs
REBOL []
print "Content-type: text/html^/"

html: make string! 2000
```

```

emit: func [data] [repend html data]

read-cgi: func [
  ;Read CGI data. Return data as string or NONE.
  /local data buffer
][
  switch system/options/cgi/request-method [
    "POST" [
      data: make string! 1020
      buffer: make string! 16380
      while [positive? read-io system/ports/input buffer 16380][
        append data buffer
        clear buffer
      ]
    ]
    "GET" [data: system/options/cgi/query-string]
  ]
  data
]

emit [
  <HTML><BODY BGCOLOR="#FFC080">
  <H2> "CGI Form Data:" </H2>
  "Submitted: " now <BR>
  "REBOL Version: " system/version <P>
  <TABLE BORDER="1" CELLSPACING="0" CELLPADDING="5">
]

foreach [var value] decode-cgi read-cgi [
  emit [<TR><TD> mold var </TD><TD> mold value </TD></TR>]
]

emit [</TABLE></BODY></HTML>]
print html

```

Пример для сайта

Расширенный пример web form, который может пригодится для использования на вашем сайте.

Код HTML

Эта HTML форма включает в себя текстовые поля, кнопки, checkboxes и меню.

The image shows a web form with the following elements:

- A text input field labeled "Full name?".
- A text input field labeled "Email address?".
- Three radio buttons for the question "Send free widget?", with options "Yes", "No", and "Maybe".
- Three checkboxes for the question "Status?", with options "Student", "Consultant", and "Programmer".
- A dropdown menu labeled "Send it Now".
- A text area labeled "Notes:".

HTML код для этой формы:

```

<FORM ACTION="http://www.reboltech.com/cgi-bin/rebol/test.cgi"
METHOD="POST">
Full name? <BR>
<INPUT TYPE="TEXT" NAME="name" SIZE="40">
<P>
Email address? <BR>
<INPUT TYPE="TEXT" NAME="email" SIZE="40">
<P>
Send free widget?
<INPUT TYPE="RADIO" NAME="send" VALUE="yes"> Yes
<INPUT TYPE="RADIO" NAME="send" VALUE="no"> No
<INPUT TYPE="RADIO" NAME="send" VALUE="maybe"> Maybe
<P>
Status?
<INPUT TYPE="CHECKBOX" NAME="status" VALUE="student"> Student
<INPUT TYPE="CHECKBOX" NAME="status" VALUE="consult"> Consultant
<INPUT TYPE="CHECKBOX" NAME="status" VALUE="prog"> Programmer
<P>
<SELECT NAME="when">
<OPTION SELECTED>Send it Now</OPTION>
<OPTION>Send it later</OPTION>
<OPTION>Send it next month</OPTION>
<OPTION>Send it next year</OPTION>
</SELECT>
<P>
Notes: <BR>
<TEXTAREA NAME="description" ROWS="8" COLS="45"></TEXTAREA>
<P>
<INPUT TYPE="SUBMIT" NAME="submit" VALUE="Submit">
</FORM>

```

Тестирование формы

Форма выше может быть проверена тем же самым способом как более ранние примеры, отправкой к сценарию test.cgi. Обратите внимание, что POST метод используется, чтобы позволить входным полям иметь любую длину.

Вот - ответ примера от test.cgi:

CGI Form Data:	
Submitted: 12-Apr-2003/16:20:42-7:00	
REBOL Version: 2.5.5.4.2	
name	"Luke Lakeswimmer"
email	"luke@rebol.com"
send	"yes"

status	["student" "prog"]
when	"Send it next month"
description	"I use REBOL for all my programming."
submit	"Submit"

Обратите внимание на Результат Состояния

Обратите внимание, что результат состояния был возвращен как блок, который содержит два значения строки. Это указывает, что были выбраны два состояния checkboxes.

Если вы пробуете этот пример, и состояние появляется не раз в левом столбце, то вы не используете REBOL/Core 2.5.5 (или выше). Это не большая проблема, но может быть проблемой, если вы конвертируете DECODE -CGI блок в объект.

Полезные подсказки

Вот несколько методов которые могут быть полезными при обработке CGI форм в REBOL.

Отладка

Поскольку сценарии CGI, выполненные на сервере и должны быть загружены каждый раз после редактирования и изменения кода, на это может требоваться много времени, чтобы найти ошибки.

Одна полезная методика должна добавить строку кода к вашему сценарию CGI, который распечатывает строку запроса, посылаемую от браузера. Таким образом вы знаете входные данные когда сценарий начинает обработку.

Если вы используете функцию READ-CGI, предложенную выше, просто добавьте в конце строки :

```
probe data
```

Теперь, когда выполняется сценарий, будет видно код CGI сверху вашей web-страницы. (некоторые web-браузеры не могут показать это.)

Тестирование сценариев на локальной машине

Вы сэкономите много времени, тестируя сценарии на локальной машине перед загрузкой на сервер.

Чтобы обеспечивать тестирование вашему сценарию, вы можете установить поля объекта CGI непосредственно в пределах вашего сценария. Например, код ниже:

```
if none? system/options/cgi/request-method [  
  system/options/cgi/request-method: "GET"  
  system/options/cgi/query-string: "your-string"  
]
```

Однако, метод, который я предпочел бы, состоял в том, чтобы изменить функцию READ-CGI, чтобы обеспечить заданное по умолчанию значение возвращения, если метод запроса не был найден (что означает, что сценарий выполняется на локальной машине а не на сервере). Это можно сделать, добавив входные данные на последнюю линию функции READ-CGI:

```
any [data "my-test-data-here"]
```

Если переменная DATA не одна, то тест - данные используются.

Вы можете использовать PROBE, делая отладку метода, показанного ранее, чтобы создать CGI кодируемую строку. Только "вырезать и вставить" для локального теста.

Обращение к данным формы как к объекту

Для больших форм, которые имеют много полей, можете проще обращаться к их значениям, преобразовывая DECODE -CGI блок результата к объекту. Это может быть сделано с помощью функции CONSTRUCT (который в отличие от MAKE, или CONTEXT не оценивает содержание объекта):

```
request: construct decode-cgi read-cgi
```

Результат - объект, и вы можете обратиться к его полям данных на прямую. Например:

```
request/name  
request/email  
request/status  
...
```

Если поле имеет многократные значения возвращения для отдельного имени (как обычно делается для checkbox полей), результатом может быть строка или блок. Вы можете использовать функцию BLOCK?, чтобы найти многократные значения. Например:

```
if block? request/status [  
    ...multiple checkboxes clicked...  
]
```

Смотри DECODE-CGI раздел для дополнительной информации.

Если вы хотите убедиться, что объект всегда имеет все его требуемые поля (это может зависеть от используемых полей, представленных в web form), вы можете определить шаблон, где вы используете CONSTRUCT:

```
template: context [command: name: email: status: none]  
request: construct/with decode-cgi read-cgi template
```

Теперь ваш сценарий может проверить поля объекта, чтобы определить, были ли они установлены перед доступом к ним:

```
if request/command [  
    switch request/command [...]  
]
```

Проверка значений Web Form

Вероятно самая трудная часть написания большинства сценариев CGI, проверка полей web form, чтобы удостовериться, что они без ошибок перед обработкой. Например, вы могли бы требовать допустимого адреса электронной почты, номера целого числа, URL, и т.д.

Один способ проверять поля состоит в том, чтобы использовать функцию PARSE. Например, если поле должна содержать только цифры, вы можете использовать код типа:

```
digits: charset "0-9"  
if not parse/all value [some digits] [  
    print "Error in number" quit  
]
```

Обратите внимание, что PARSE/ALL необходим, если вы хотите обнаружить пробелы, символы табуляции в строке.

Вы можете легко проверить ваш PARSE код используя REBOL console:

```
>> digits: charset "0123456789"  
>> parse/all "123" [some digits]  
== true
```

```
>> parse/all "123x" [some digits]
== false
```

Другой способ проверять результаты для установленных значений, подобно возвращенным из радио-кнопок или текстовых полей, состоит в том, чтобы использовать функцию FIND:

```
if not find [
  "send it now"
  "send it later"
  "send it next month"
  "send it next year"
] request/when [
  print "Error in value" quit
]
```

Для более сложных типов значений, вы можете также использовать встроенный загрузчик REBOL. Вот - проверка для допустимого целого числа:

```
value: load/all request/age
if not integer? value/1 [
  print "Invalid age"
]
```

Вот - проверка для допустимого REBOL адреса электронной почты:

```
value: load/all request/age
if not email? value/1 [
  print "Invalid email"
]
```

Подрезка полей

Это - обычная практика, чтобы урезать продвижение и перемещение пробелов в текстовых полях для формирования web form.

Пример подрезки полей в коде REBOL:

```
trim/head/tail request/email
```

Сохранение данных Web Form

В зависимости от ваших требований, есть много способов сохранить ваши данные web form в пределах вашего сценария CGI.

Самый простой метод состоит в том, чтобы использовать функцию SAVE и извлечь из REBOL объект, использованном в примере выше:

```
save %cgi-data request
```

Не забудьте использовать команду -cs, если вы используете REBOL, чтобы написать скрипт. См. [Справочник\(Руководство\) <http://www.rebol.com/docs/cgi1.html>](http://www.rebol.com/docs/cgi1.html).

Если вы хотите сохранить историю запросов, вы могли бы добавить в конец блок запроса, возвращенный от DECODE-CGI в файл:

```
data: decode-cgi read-cgi
write/append %cgi-log append mold data newline
```

Этот пример добавляет в конец разделитель строк, чтобы поместить каждый запрос в отдельную линию в пределах файла.

Вы можете также добавить поля подобно дате, запрашивая адрес IP компьютера, и еще что вам

необходимо:

```
write/append %cgi-log reform [  
    now/date  
    system/options/cgi/remote-addr  
    mold data  
    newline  
]
```

Сохранение данных Web в уникальный файл

В сценариях CGI, создавая уникальное имя файла, чтобы хранить каждый запрос CGI сложнее чем это кажется.

Проблема в том, что отдельный сценарий CGI может быть выполнен многократно сервером за один промежуток времени.

Метод состоит в том, чтобы использовать текущую дату и время, чтобы создать имя файла. Это не работает также, потому что в то же самое время, другой скрипт может использовать то же самое значение даты.

Использование REBOL Forms с CGI

CGI также работает хорошо для форм, которые созданы в REBOL. Фактически, используя графический интерфейс пользователя REBOL вы можете создать формы, которые являются более динамическими и правильными перед посылкой их на сервер (отличающийся лучшей работой и меньшим количеством трафика на сервере).

Проблемы защиты CGI

При обработке форм CGI, вы должны знать о потенциальных проблемах защиты.

Данные из формы, совет

- **Никогда не выполнять данные**, которые послали вашему сценарию от web form. (в REBOL, не используйте функцию LOAD и LOAD/ALL. Если используете REBOL/Command, то не выполняйте внешних программ основанных на данных полученных от web forms.)
- **Используйте методы проверки**, для проверки корректности заполнения полей web form.
- **Лимит на ввод символов в поле формы** где это имеет смысл. Например, названия, адреса и т. д. Могут быть урезаны функцией CLEAR в REBOL.
- **Лимит отправляемых данных** к максимальной длине. Пример ниже.

Вот - код примера, который ограничивает строку 100 символами:

```
clear skip string 100
```

Ограничить отправку данных CGI можно использованием функции READ-CGI, чтобы прерваться, если ввод превышает указанную длину:

```
read-cgi: func [  
    ;Read CGI data. Return data as string or NONE.  
    /local data buffer  
][  
    switch system/options/cgi/request-method [  
        "POST" [  
            data: make string! 1020  
            buffer: make string! 16380  
            while [positive? read-io system/ports/input buffer 16380][  
                append data buffer  
                clear buffer  
                if (length? data) > 10000 [print "aborted" quit]  
            ]  
        ]  
    ]
```

```
"GET" [data: system/options/cgi/query-string]
]
data
]
```

Обратите внимание, что вы не должны ограничивать GET данные, потому-что сервер делает это по умолчанию.

Доступ к файлам сервера

Чтобы сохранять данные от ваших форм, ваш сценарий CGI может иметь доступ к одному или более каталогам файла на вашем сервере. **Вы должны быть осторожны**, если вы не хотите случайно дать пользователю сайта способность читать файлы данных или, еще хуже, переписать или редактировать любой файл.

Лучший способ избежать таких проблем не разрешать, чтобы читались данные из web form, видеть имя файла, которое используется, чтобы обратиться к файлам на сервере..

Если вы хотите иметь дополнительную защиту, вы можете использовать функцию REBOL SECURE, чтобы ограничить обращение файла, к одному или более определенным каталогам. Например, вы можете добавить несколько строк кода в начале вашего сценария (перед обращением к файлам):

```
secure [
  file quit
  %cgi-data/ [quit all allow write]
]
```

Если по какой-то причине ваш сценарий CGI пытается читать или написать файлы вне каталога "cgi-data" или пробует читать любые файлы в каталоге "cgi-data", сценарий автоматически закончится.

```
secure [
  file quit
  %zip-codes.r [quit all allow read]
  %prices.r [quit all allow read]
  %cgi-data/ [quit all allow write]
]
```

Необходимость -s опции

Убедитесь, что определили "s" опцию в вашем пути команды REBOL CGI, иначе запрос функции SECURE прервет ваш сценарий. Например, как показано ранее, используйте командную строку типа:

```
#!/home/myaccount/rebol -cs
```