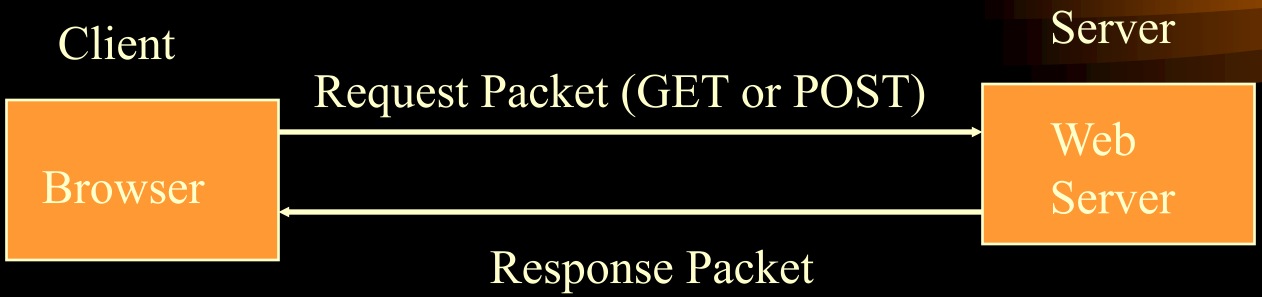
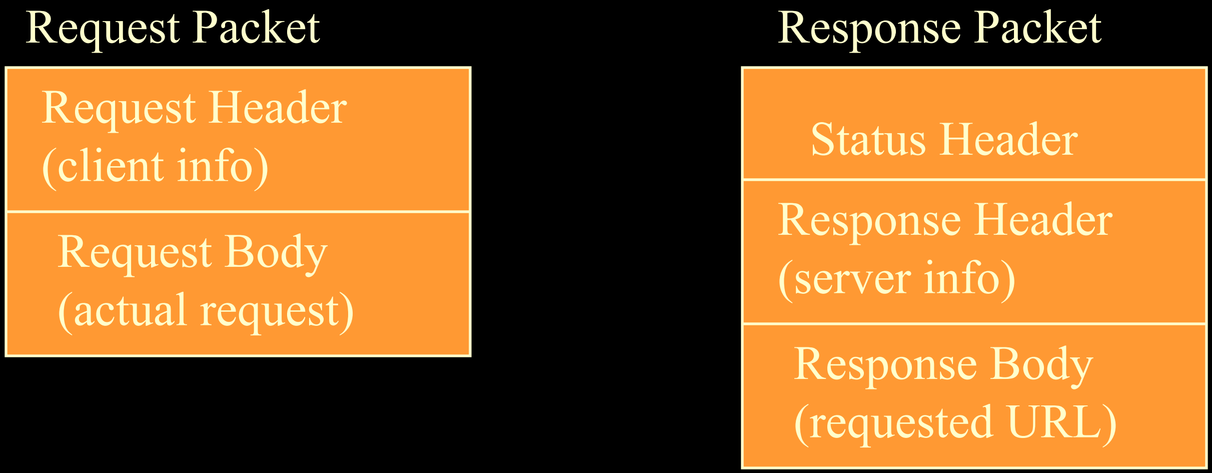
HTML Forms Validation

On to forms processing...

- The processing of a form is done in two parts:
  - Client-side
    - at the browser, before the data is passed to the back-end processors
      - could be done via Javascript, VBScript or Jscript (might even be done via Java applets)
      - data validation done locally to relieve the back-end processors of having to do it
  - Server-side
    - processing done in the back-end system
      - CGI Scripts
      - VBScript, JavaScript, Java
      - Database

HTTP Protocol




Request Packet
Request Header (client info)
Request Body (actual request)
Response Packet
Status Header
Response Header (server info)
Response Body (requested URL)

Series of events

- The processing of a form follows the series of events:
  - user requests a form
  - form is received by browser, rendered and Javascript initialization processing is done .
  - User fills in form and any event processing required by form is performed
  - users click on a Submit Button
  - submission Javascript is run
  - if OK the data is submitted to the server via the method coded on the <form> tag; i.e. by an http GET or POST.
  - Data sent to server is processed by the mechanism coded in the METHOD attribute of the <form> tag.

GET and POST

- In the HTTP protocol, data from the client is sent to the server in one of two primary ways:
  - GET
    - attaches the forms data (URL encoded) to the requested URL and places the URL in the body of an http request packet
      - usually the URL for a CGI-script to process the data or the URL for a meta processor (a ColdFusion, ASP or PHP page)
    - as the URL has the data attached it is visible in the Location box of the browser, Since this allow the data (possibly a password) to be seen by any with in view of the screen it is not preferred.
  - POST
    - places the forms data into the body of the http request packet, this way the data will not be seen by snooping eyes, it is far from secure but it is more private than GET. This is the preferred way to send data to the server.

Client-side Forms Processing…

- Now that we've been introduced to the components that can make up a form what can we do with them
- One major use of Javascript in conjunction with HTML forms is for validation of forms data before allowing the data to be submitted to a backend server for processing
  - doing the data validation at the client (browser) relieves the server from having to do it
    - on a busy server this can free up a lot of processing cycles that can be better used to process more user requests and/or backend database functions.
    - Most client machines are under-utilized and doing the validation at the client machine increases the utilization rate of the client machines
    - usually client side processing MIPs are cheaper than server-side processing MIPs

Validation Functions

- Validation functions can usually be written in either of two ways
  - traditional iterative programming methodologies
    - write a small program (function) that iterates through the user input and validates the data type and returns true or false depending on result
    - requires Javascript level iterative processing
      - can be relatively slow, especially if form has many fields to be validated
  - pattern matching (via regular expressions, supported in Javascript 1.2 as in Perl 5)
    - create a regular expression pattern for the data type and let the system validate the user input via the pattern matching capabilities of the
    - pattern matching is done at a low level in the Javascript processor, so validation is very quick (even for long forms)

Approaches

- Point and click components (buttons, selection lists, radio buttons and checkboxes) don’t need to be validated unless there are rules that deal will combinations that need to be validated.
- This leaves text input fields
  - Validate each field as the user fills it in (using onBlur( ))
    - if user fills in the field and then clicks Submit, the field won’t be checked
  - place all validation scripts in a single file for inclusion via SRC attribute of <script> tag and place one forme specific script in each form that calles the validation routines for the individual fields
    - means writing a script for each form
      - too costly and tedious
  - make a checkForm script that will iterate through a form’s elements array to validate entire form
    - requires a naming convention for naming fields
    - must be generic enough for handling all type of fields (dates, ssns, integers, reals,
    - each form’s onSubmit invokes checkForm( )

Field Naming Convention

- First 3 chars of field name indicate type:
  - INT ; indicated field is to be an integer
  - DEC ; indicated a decimal number
  - DAT ; indicates a date field
  - SSN ; indicates a Social Security Number
  - LIT ; literal alphnumeric field
- If field is a certain length, follow type with length
  - INT5 ; integer between 0 and 99999
  - LIT8; 8 character literal
- For required fields precede name with RQD or a special character
  - RQDINT3 ; required integer between 0 and 999
- You get the idea; this technique is easily extensible
  - all it takes is a unique identifier
  - a new decoder in checkForm( )
  - and new validation function in your included function library

Validation Functions

- Validation functions can usually be written in either of two ways
  - traditional iterative programming methodologies
    - write a small program (function) that iterates through the user input and validates the data type and returns true or false depending on result
    - requires Javascript level iterative processing
      - can be relatively slow, especially if form has many fields to be validated
  - pattern matching (via regular expressions, supported in Javascript 1.2 as in Perl 5)
    - create a regular expression pattern for the data type and let the system validate the user input via the pattern matching capabilities of the
    - pattern matching is done at a low level in the Javascript processor, so validation is very quick (even for long forms)

Regular Expression Syntax

/n,/r,/t	match literal newline, carraige return, tab
\\, \/, *	match a special character literally, ignoring or escaping its special meaning
[…]	Match any one character between the brackets
[^…]	Match any one character not between the brackets
.	Match any character other than the newline
\w, \W	Match any word\non-word character
\s, \S	Match any whitespace/non-whitespace
\d, \D	Match any digit/non-digit
^, $	require match at beginning/end of a string or in multi-line mode, beginning/end of a line
\b, \B	require a match at a word/non-word boundary
?	Optional term : Match zero or one time
+	Match previous term one or more times

Regular Expressions (more)

*	Match term zero or more times
{n}	Match pervious term n times
{n,}	Match previous term n or more times
{n,m}	Match prev term at least n time but no more than m times
a | b	Match either a or b
(sub)	Group sup-expression *sub* into a single term and remember the text that it matched
\n	Match exactly the same chars that were matched by sup-expression number n
$n	In replacement strings, substitute the text that matched the nth sub-expression

search( ) and test( )

- A couple of very useful methods
  - search( ) - similar to indexOf( )
    - method on String
    - String.search(regex)
    - return -1 for no match; integer starting position if a match is found
    - argument is a regular expression
  - test( )
    - method on regexObject
    - returns boolean
    - regex.test(String)
    - false if no matches are found in String, true otherwise
    - argument is a String

INT5 ; two ways

- inInt.search(^ \ + | \ - ? \ d{1,2,3,4,5} $ /)
  - at the beginning of a line there may be a plus or, 0 or 1 minus', followed by one or two or three or four or five digits at the end of the line
    - the plus is escaped (preceded with a \ ) as it is also meta character
- r1 = ^ \ + | \ - ? \ d{1,2,3,4,5} $ / ; if (r1.test(field . value))...

HTML5 Validation

- New attributes for <input>
  - required ='required'
  - pattern= 'a regular expression'
    can be used with <input> types, **text**, **search**, **url**, **telephone**, **email**, and **password**.
  - min=   max = step=
    can be used with <input type=range>