## HTTP : un protocole de communication

### Qu'est-ce qu'un protocole de communication ?

Ensemble des règles qui organisent les échanges de messages entre 2 interlocuteurs

- Le type des messages et leur forme («syntaxe»)
- L'organisation temporelle, la synchronisation (ou pas)

### Différentes «couches» réseau

Les communications réseaux reposent sur plusieurs protocoles : du niveau matériel jusqu'au niveau des applications.

### HTTP est un protocole applicatif

- Protocole de «haut» niveau qui définit utilisé par les navigateurs (côté client) et les serveurs web (côté ... serveur)
- HTTPS : sa version cryptée (Secure HTTP)
- Exemple d'autres protocoles applicatifs : FTP, SMTP (envoi de mail), IMAP (réception mail)...



## Dialogue Client / Serveur en HTTP

### Le client envoie au serveur une **requête** (commande)

- **GET** requête ordinaire pour demander un document.
- **POST** requête comportant **l'expédition de données**.
- d'autres commandes moins utilisées comme HEAD, OPTIONS, PUT, DELETE, TRACE.

### Le serveur envoie une **réponse**

éventuellement accompagnée de données. En voici quelques unes possibles :

- **100 Continue**
- **200 OK**
- **304 Not Modified**
- **404 Not Found**



## Exemple (1/3)

### exemple : Accès par un client wget à la page `http://www.fil.univ-lille1.fr/technoweb/exos/bonjour.html`

Message du client

```
GET /technoweb/exos/bonjour.html HTTP/1.1
User-Agent: Wget/1.15 (darwin11.4.2)
Accept: */*
Host: www.fil.univ-lille1.fr
```

- première ligne : commande (GET) suivie du nom du document demandé et de la version HTTP utilisée.
- ensuite l'**entête HTTP** : lignes sous la forme ***Nom_du_champ : valeur_du_champ***
- La demande se termine par une ligne vide.



## Exemple (2/3)

Réponse du serveur (simplifiée)

```
HTTP/1.1 200 OK
Date: Wed, 03 Feb 2016 16:03:09 GMT
Server: Apache
Last-Modified: Wed, 03 Feb 2016 15:59:13 GMT
ETag: "18c284-147-52adfb303d640"
Accept-Ranges: bytes
Content-Length: 327
Content-Type: text/html

<!DOCTYPE HTML>
 <html xmlns="http://www.w3.org/1999/xhtml">
  <head>
      <meta charset="UTF-8" />
      <link rel="stylesheet" href="styleBonjour.css" type="text/css" />
      <title>Bonjour</title>
  </head>
  <body>
    <h1>Bonjour</h1>
    <p><img src="lille1.png" alt="logo Lille1" /></p>
  </body>
</html>
```



## Message HTTP

### Syntaxe générique des **messages HTTP** (requêtes ou réponses)

- Ligne de début (requête ou status)
- En-tête HTTP
  Un nombre quelconque de **champs d'en-tête** ***Nom_du_champ : valeur_du_champ***
- Une ligne vide
- Éventuellement une **partie « données »** sous forme de texte (le binaire est encodé en texte).

### GET / POST

- Une requête GET n'envoie PAS de partie données
- Une requête POST envoie une partie données.



## Dialogue complet pour l'affichage de la page

Pour afficher complètement la page, le navigateur devra faire encore 2 autres connexions au serveur : l'une pour la feuille de style, l'autre pour l'image contenue dans le document.

| | | | |
|---|---|---|---|
| 192.168.[illegible] | 195.83.34.212 | HTTP | 407 GET /technoweb/exos/bonjour.html HTTP/1.1 |
| 195.83.34.212 | 192.168.[illegible] | HTTP | 549 HTTP/1.1 200 OK (text/html) |
| 192.168.[illegible] | 195.83.34.212 | HTTP | 434 GET /technoweb/exos/styleBonjour.css HTTP/1.1 |
| 195.83.34.212 | 192.168[illegible] | HTTP | 454 HTTP/1.1 200 OK (text/css) |
| 192.168.[illegible] | 195.83.34.212 | HTTP | 443 GET /technoweb/exos/lille1.png HTTP/1.1 |
| 195.83.34.212 | 192.168.[illegible] | HTTP | 1293 HTTP/1.1 200 OK (PNG) |

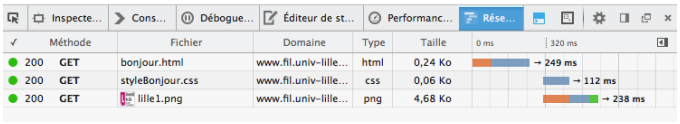
### Le protocole HTTP **ne gère pas** de continuité de session

- 3 requêtes distinctes et indépendantes les unes des autres.
- Pour le serveur, le 2ème GET n'a rien à voir avec le 1er ni le 3ème avec les autres.

## Le dialogue complet (outils de développement du navigateur)

**Bonjour**

Université de Lille 1 SCIENCES ET TECHNOLOGIES






## Envoi de paramètres dans l'URL

**Une URL peut comporter des paramètres**

- La partie « paramètres » commence par un caractère `?`
- Définition de paramètre : *nom* `=` *valeur*
- Les définitions sont séparées par `&`
- Exemple avec 2 paramètres : couleur (blue) et taille (20px)

```
couleurs.php?couleur=blue&taille=20px
```

**méthode « GET »**

- Utilisé avec la méthode GET
- Appelé « passage en mode GET »

**Limitation**

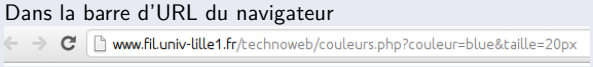
La longueur de l'URL !



## Comment envoyer des paramètres dans l'URL ?

**Là où l'on peut spécifier une URL**

- Dans la barre d'URL du navigateur




- Dans une balise `a`

```
<a href="couleurs.php?couleur=blue&taille=20px">Lien</a>
```

ou plutôt (conformité HTML)

```
<a href="couleurs.php?couleur=blue&amp;taille=20px">
Lien
</a>
```

- **Par un formulaire de saisie**, avec méthode GET (cf suite du cours)
- ... etc ...



## Envoi de paramètres par la méthode POST

- Les paramètres ne sont pas envoyés dans l'URL
- Ils sont envoyés **dans la partie données** du message HTTP
- La longueur totale des paramètres n'est plus limitée
- Permet même d'envoyer des fichiers (upload)
- Exemple

Extrait d'un message "POST"

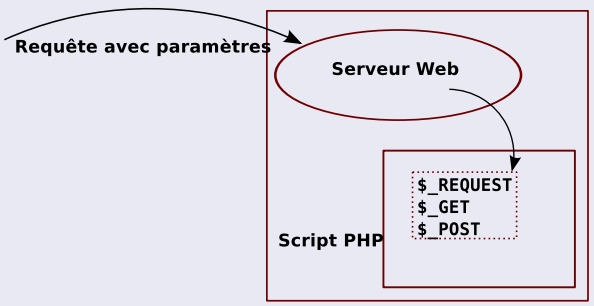
```
POST /traiteformu.php HTTP/1.1.
Connection: keep-alive.
Content-Type: application/x-www-form-urlencoded.
Content-Length: 23.

var1=314&var2=bidule%21
```

**Comment envoyer des paramètres en mode POST ?**

- **Par un formulaire de saisie**, avec méthode POST



## Accès aux paramètres en PHP




**tableaux superglobaux associatifs** (clés = noms des paramètres)

- `$_GET` pour les paramètres reçus en mode GET
- `$_POST` pour les paramètres reçus en mode POST
- `$_REQUEST` union des tableaux `$_GET`, `$_POST`, `$_COOKIE`



## Accès aux paramètres en PHP (exemple)

couleurs.php utilisant les paramètres du GET

```
<html>
 <head>
  <style>
  <?php
    echo "* \{ color: {$_GET['couleur']};
                font-size: {$_GET['taille']}; \}";
  ?>
  </style>
 </head>
 <body>
  Affichage couleur: <?php echo $_GET['couleur']; ?>
 </body>
</html>
```

**Attention, sécurité : TOUJOURS tester si les valeurs de paramètres reçues sont correctes**

C'est un exemple naïf : on n'a pas testé la validité des valeurs

## Accès aux paramètres en PHP : sécurité

**Attention, sécurité : TOUJOURS tester si les valeurs de paramètres reçues sont correctes**

- Établir la liste ou la forme des valeurs acceptables.
- Vérifier systématiquement que la valeur reçue **est** acceptable
- Prévoir ce que l'on fait dans le cas contraire : soit choisir une valeur « par défaut » si cela a un sens, soit produire une page d'erreur.



## Formulaires HTML

### Comportement par défaut

- Interface graphique pour la saisie par l'utilisateur
- Après **validation**, **charge une page**
  **en envoyant les valeurs saisies sous forme de paramètres**

### Un **formulaire** : élément `<form>`

- Définit un **destinataire du formulaire** : attribut `action`
- Définit une **méthode d'envoi** : attribut `method` (GET, par défaut)
- Définit des **champs de saisie**. Pour chacun :
  - Un **nom de variable** (attribut `name`)
  - Une **valeur** saisie ou choisie par l'utilisateur (l'attribut `value`)



## Formulaires HTML

*méthode HTTP d'envoi* *URL de destination*

```
<form method="get" action="couleurs.php">
<input
    type="text"
    name="taille"
    value="10pt"
/>
<select name="couleur">
    <option value="blue">Bleu</option>
    <option value="red">Rouge</option>
    <option value="green">Vert</option>
</select>
</form>
```

*1er champ de saisie*

*Noms des variables*

*Valeur (modifiable par l'utilisateur)*

*2ème champ de saisie*

*Valeur (à choisir par l'utilisateur)*




## Contenu de l'élément `<form>`

### Champs de saisie - attribut `name` obligatoire

- `<input>` Nombreux types différents
  attribut `type` important
- `<select>` Liste de choix
- `<button>` Bouton. (attribut `type` à préciser)
- `<textarea>` texte sur plusieurs lignes

### Éléments de mise en page

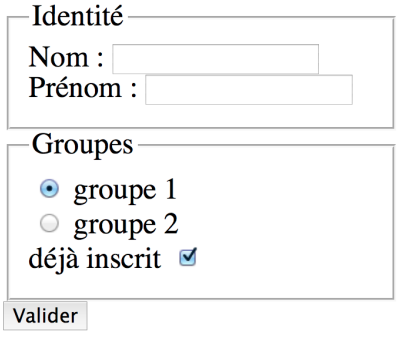
- `<label>`
- `<fieldset>`
- `<legend>`

### Assistance à la saisie

- `<datalist>`



## Exemple de formulaire (1/3)






## Exemple de formulaire (2/3)

```
<form action="traiteformu.php" method="post">
 <fieldset id="identite">
  <legend>Identité</legend>
  <label for="nom">Nom :</label>
  <input name="nom" id="nom" type="text"
           size="15" maxlength="30" />
  <br />
  <label for="prenom">Prénom :</label>
  <input name="prenom" id="prenom" type="text"
           size="15" maxlength="30" />
 </fieldset>
```

à suivre

## Exemple de formulaire (3/3)

suite

```
 <fieldset id="lesgroupes">
  <legend>Groupes</legend>
  <input type="radio" name="groupe" id="groupe1"
                                    value="1" />
  <label for="groupe1">groupe 1</label> <br />
  <input type="radio" name="groupe" id="groupe2"
                                    value="2" />
  <label for="groupe2">groupe 2</label> <br />
  <label for="inscrit">déjà inscrit</label>
  <input type="checkbox" name="inscrit" id="inscrit"
                                     value="1" />
 </fieldset>
 <div id="validation">
  <button type="submit" name="valid" value="ok">Valider
  </button>
 </div>
</form>
```



## L'élément <input> *(sans contenu)*

### Différents types de <input>

- `type="text"` Texte sur une seule ligne.
  - `size` : largeur du cadre
  - `maxlength` : maxi de la chaîne saisie
  - `value` : valeur pré-remplie
- `type="password"` Texte non affiché.
- `type="checkbox"` Case à cocher. `value` indispensable
- `type="radio"` Bouton radio. `value` indispensable
- `type="hidden"` sans interface. `value` indispensable
- `type="button"` `type="submit"` `type="reset"` `type="image"`
  Préférer l'élément <button>
- `type="file"` Upload de fichier



## L'élément <select> : liste de choix

### exemple

```
<select name="couleur">
  <option value="blue"> Bleu </option>
  <option value="red" selected="selected"> Rouge </option>
</select>
```

### description

- Contient des élements <option>
  - Chaque option définit une valeur proposée (attribut `value`)
  - Le texte affiché est distinct de la valeur envoyée.
  - Présélection d'option possible (`selected="selected"`)
- Par défaut, un seul choix possible
- `multiple="multiple"` : le choix devient multiple



## Select à choix multiple

### exemple

```
<select multiple="multiple" size="4" name="tt[]">
  <option value="TW"> Tinky-Winky </option>
  <option value="D"> Dipsy </option>
  <option value="LL"> Laa-Laa </option>
  <option value="P"> Po </option>
</select>
```

- Utilisation d'un nom avec crochets (vides)
- La variable renvoyée « sera » une table contenant toutes les valeurs sélectionnées.




## Élément <button> : boutons

### Syntaxe

```
<button type="submit" name="valid" value="bouton1">
Envoyer <!-- contenu à afficher dans le bouton -->
        <!-- on pourrait y mettre une image -->
</button>
```

Ce qui est affiché dans le bouton est distinct de ce qui est envoyé (`value`)

### Les types

- `type="submit"` : Envoi du formulaire.
  Si plusieurs boutons de type submit sont présents, seule la valeur du bouton activé est envoyée.
- `type="reset"` : Réinitialisation des champs de saisie
- `type="button"` : pour déclencher une fonction javascript.



## Les aides à la validation (HTML5)

### Vérification des champs de texte <input>

- Distinguer ce qui doit être accepté vs rejeté (**contrainte**)
  - par une expression rationnelle : attribut `pattern`
  - par l'attribut `required="required"` : chaîne vide interdite.
  - avec un champ texte spécialisé : attribut `type=` : `email` `url` `tel` `date` `number` `month` ...
- Quand le contenu ne respecte pas la contrainte, le champ est dans l'état **invalide**
- Si un champ est invalide, le formulaire n'est pas envoyé. Un message est affiché.
- En CSS : pseudo-classes `:valid` et `:invalid` pour changer la présentation selon l'état du champ (couleur, cadre, etc...)
- L'état du champ peut aussi être passé à invalide (ou valide) par une fonction javascript.

## Exemple de vérification de texte

### un champ de type email et un champ contrôlé par pattern

```
<form action="test.php">
  <input type="text" name="ident"
    required="required"
    pattern="[a-zA-Z]{3,6}"
  />
  <input type="email" name="adresse"
    required="required"
  />
 <button type="submit" name="valid">Valider</button>
</form>
```



## L'aide à la saisie : <datalist> (HTML5)

### Propositions de réponses à un champ <input type="text">

- Une liste de réponses proposées
- Peut-être associée à un ou plusieurs champs de texte
  Attribut `list` du champ `<input type="text">`
- Ce ne sont **que des propositions, la saisie libre reste possible** ⇒ différent d'un `select`

```
<form action="test.php">
  <input type="text" name="ue"
         list="maliste"  />
  <datalist id="maliste">
   <option value="TW">Techno Web</option>
   <option value="POO">Programmation Orientee Objet</option>
   <option value="AEL">Automates et Langages</option>
  </datalist>
 <button name="valid" type="submit">Valider</button>
</form>
```



## Formulaires et javascript

### Vérification des données AVANT envoi

- Seul moyen qui existait en HTML 4
- HTML5 permet de se passer de javascript dans un grand nombre de cas :
  **Utiliser `required`, `pattern` et les nouveaux types de `<input>`**
- Javascript peut venir en complément, si nécessaire.
- Traitement facilité par la nouvelle API sur les contraintes.
  **Utiliser `validity`, `setCustomValidity()` ....**

### Utilisation du formulaire pour mettre à jour le document ou déclencher un traitement javascript

- Un outil pour les "applications Web"
- Évènement `submit` du formulaire
- Annuler l'action par défaut



## API javascript pour les formulaires

### Méthodes et attributs d'un nœud `<form>`

| | |
|---|---|
| `elements` | liste des champs de saisie |
| `length` | nombre de champs de saisie |
| `['nomDeChamp']` | soit le nœud du champ portant ce nom<br>soit une liste des nœuds portant ce nom<br>attention : recherche par id ou name |
| `reset()` | réinitialisation |
| `submit()` | envoie le formulaire |
| `checkValidity()` | teste si tous les champs sont valides |

### Évènement d'un nœud `<form>`

| | |
|---|---|
| submit | Quand le formulaire est validé, de quelle que manière que ce soit. |



## API javascript pour les formulaires

### Méthodes et attributs d'un nœud de champ de saisie

| | |
|---|---|
| `form` | Nœud `<form>` auquel il appartient |
| `value` | Valeur **courante** du champ. |
| `validity` | objet représentant l'état du champ vis à vis des différentes contraintes. |
| `checkValidity()` | (booléen) : test global de validité |
| `setCustomValidity(message)` | le champ devient invalide si *message* $\neq$ "" |

### Évènements d'un nœud de champ de saisie

| | |
|---|---|
| change | Quand le champ perd le focus et que sa valeur a été modifiée |
| input | Lors de toute saisie, de toute origine (clavier, souris, paste). |



## Exemple 1

Fichier HTML (extraits)

```
...
 <style>
   input:valid {background-color:#eee;}
   input:invalid {background-color:yellow;}
 </style>
 <script type="text/javascript" src="form.js"></script>
....
<form action="test.php" id="formulaire">
   <input type="number" name="nombre"
    required="required"
    pattern="[0-9]+"
    placeholder="multiple de 7"
  />
 <button type="submit" value="b1">Valid</button>
 </form>
...
```

## Exemple 1

**Contrainte : vérification "multiple de 7" qui s'ajoute à celle définie en HTML (pattern)**

form.js

```
function init(){
  var f = document.getElementById("formulaire");
  f['nombre'].addEventListener('input',verifMultiple,false);
}
function verifMultiple(ev){
 if (! this.validity.valueMissing &&
     ! this.validity.patternMismatch){
  var i =  parseInt(this.value);
  if ( i % 7 != 0)
   this.setCustomValidity("Entrez un multiple de 7");
  else
   this.setCustomValidity("");
 }
}
window.addEventListener("load",init);
```



## Exemple 2

**Formulaire utilisé "localement" (pas de requête HTTP, modification du DOM)**

form.js

```
function init(){
  var f = document.getElementById("formulaire");
  f['nombre'].addEventListener('input',verifMultiple);
  f.addEventListener('submit',afficheReponse);
}
function verifMultiple(ev){ //...
}
function afficheReponse(ev){
 ev.preventDefault(); //annule l'action par défaut
 var reponse = this['nombre'].value;
 var item = document.createElement("li");
 item.appendChild(document.createTextNode(reponse));
 document.getElementById("liste").appendChild(item);
}
window.addEventListener("load",init);
```



## Type des données transmises en HTTP

**Annoncer le type des données transmises**

- Le type des données envoyées dans un message HTTP est annoncé par une **propriété de l'en-tête HTTP**.
  ⇒ propriété `Content-Type`
- Le client (navigateur en général) a **besoin** de cette information pour savoir comment traiter les données reçues.
  ⇒ considérer le `Content-Type` comme **obligatoire**
- Valeur de la propriété `Content-Type` :
  - le type MIME
  - optionnellement un codage de caractère (`charset`)

  Exemple :
  `Content-type: text/plain;charset=UTF-8`



## Bien définir le type MIME

**Qu'est le type MIME ?**

- le type MIME (Multipurpose Internet Mail Extensions) décrit le format des données échangées sur Internet
- la liste des types MIME est enregistrée par IANA (Internet Assigned Numbers Authority)
- Exemples de Mime Type :
  - `text/html` `text/css` `text/csv` `text/xml` `text/plain`
  - `application/javascript` `application/json` `application/pdf` `application/zip` `application/xml`
  - `image/jpeg` `image/png` `image/gif`
  - `audio/mpeg` `video/mpeg` `video/webm`



## Bien définir le type MIME

**PHP et le Content-Type**

- comme tout langage, PHP peut produire tous types de données
- produire du `text/html` n'est que l'utilisation la plus courante.
- exemples d'autres usages courants : CSS, images, PDF, sons,
- les serveurs sont configurés pour envoyer un type Mime pour chaque type de fichier (selon l'extension, par exemple).
- en général (99,9% des cas !) ils envoient `text/html` pour tout script PHP
- si l'on produit une autre type de données, il faut agir pour redéfinir `Content-Type`
  ⇒ fonction PHP : `header(`*chaîne*`)`
- par exemple : `header("Content-Type: image/png");`
- `header()` doit être utilisée avant toute production de donnée.

## Pourquoi produire en PHP des images, des fichiers PDF ...etc... ?

### Quelques exemples typiques

- Les données sont dans une base de données. Le PHP doit alors les extraire de la base et les envoyer au client.
- On souhaite limiter l'accès aux données. Le script PHP doit réserver l'envoi de certains fichiers aux utilisateurs enregistrés et connectés (sessions). Un fois la vérification faite, le script va lire le fichier et diriger son contenu vers la sortie standard.
- On doit produire les données de façon dynamique. Quelques exemples
  - production à la volée d'un document PDF personnalisé (facture, état, attestation, ...)
  - redimensionnement ou autre transformation d'une image



## Envoyer des données sur la sortie standard

### Quelques fonctions

- `echo` (on connaît !)
- `print()` (aussi)
- `readfile(`*nom de fichier*`)` envoie le contenu du fichier sur la sortie standard
- `fpassthru(`*ressource*`)` idem mais à partir d'une ressource, ouverte par `fopen()` par exemple
- Ouvrir la sortie standard comme une ressource flux :

```
$fp = fopen("php://stdout","w");
fwrite($fp,...);
fclose($fp);
```



## Le format JSON

### «JavaScript Object Notation»

- format léger pour échanger des données entre composants logiciels
- parfois aussi être utilisé comme format de sauvegarde
- c'est un format texte (et non un format binaire)
- issu de la syntaxe JS, mais n'est pas lié à JS. Des bibliothèques existent pour d'autres langages, comme PHP.
- beaucoup moins expressif que XML mais beaucoup plus économe en place et plus facile à analyser (parser).
- pas de meta informations.
- C'est un format « a minima »



## Le format JSON

### Un texte JSON = une valeur

Chaque texte JSON correct représente UNE valeur qui peut être enregistrée dans une variable du programme.

### Type des données

- **nombre** : notation **décimale** usuelle
- **booléen** : `true`, `false`
- **chaîne** délimitée par `""` uniquement. échappement par `\`
- `null`
- **tableau** classique avec indexation par défaut
- **objet** ou **tableau associatif**.
  seuls les attributs sont représentés, pas les méthodes.
  pas de typage ou de nom de classe
- Pas de commentaires ;-(



## Le format JSON

### Types composés

- **tableaux** Exemples :

```
[1, 1, 2, 3, 5, 8, 13, 21, 34]
["rouge","bleu","vert"]
```

- **objet** Exemple :

```
{ "salle":"A16", "heure":"8:00", "personnes":24 }
```

### Autre exemple

```
{
  "donneesOK" : true,
  "datetime" : "2035-01-06 11:11:11",
  "couleurs" : ["red","green","blue"],
  "lieu" : {"latitude" : 50.65, "longitude": 3.14}
}
```



## Le format JSON

### variable <-> JSON en PHP

- `json_encode(`*variable*`)` produit un texte JSON
- `json_decode(`*texte json*`)` produit une valeur PHP

### Exemple

```
$t = array(1,1,2,3,5,8,13,21,34);
echo json_encode($t);


[1, 1, 2, 3, 5, 8, 13, 21, 34]
```

## Le format JSON

### Exemple

```php
<?php
class Geo {
 public $latitude;
 public $longitude;
 public function getLatitude(){
  return $this->latitude;
 }
 public function __construct($lat, $lon){
  $this->latitude = $lat;
  $this->longitude = $lon;
 }
}

$p = new Geo(50.65, 3.14);
header("Content-Type: application/json");
echo json_encode($p);
print_r($_SERVER);
?>
```

```
{"latitude":50.65,"longitude":3.14}
```



## Le format JSON

### variable <-> JSON en Javascript

- `JSON.stringify(`*variable*`)` produit un texte JSON
- `JSON.parse(`*texte json*`)` produit une valeur JS
- Attention : l'usage de `eval` **est à proscrire**