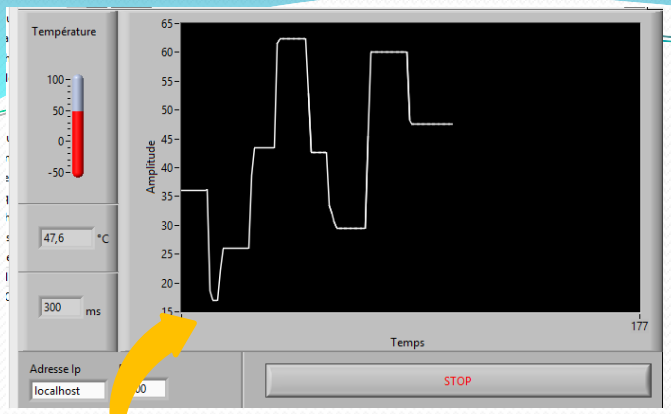
Labview Programmation réseau Communication par sockets

François SCHNEIDER

Mise en situation



Température

FOUR

Capteur température Ethernet

Le PC effectue une requête de connexion au capteur de température, il reçoit la valeur de la mesure de la température et l'affiche

Réseau

Wifi, …

Ethernet, …

Le capteur attend une demande de connexion du PC pour envoyer la valeur de la température mesurée

Client-serveur

Client

Demande un service

La

Communication

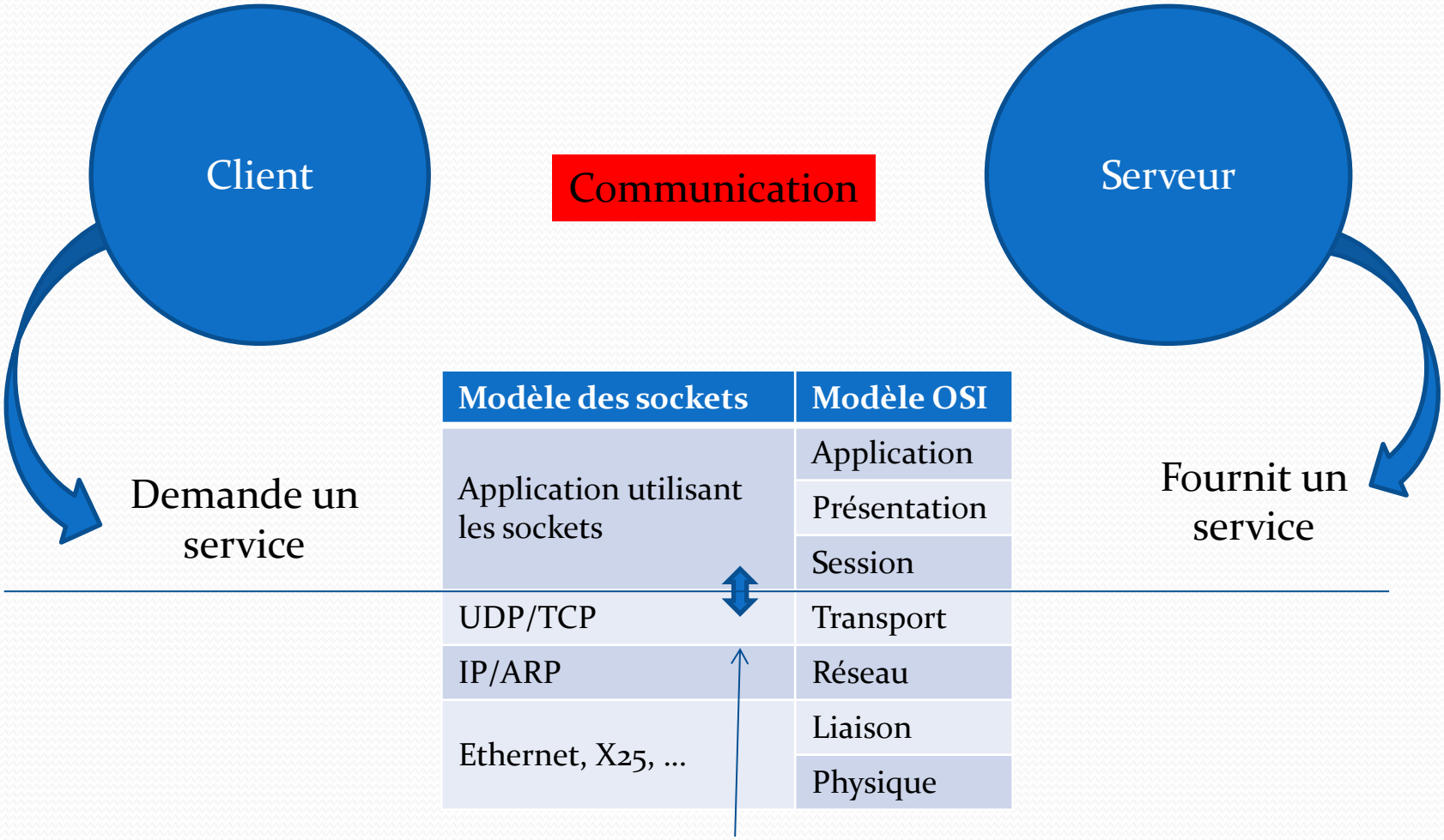
s'effectue sous le principe client-serveur à travers les couches réseau du modèle OSI

Serveur

Fournit un service

Capteur température Ethernet

Client-serveur : modèle sockets



Il apparait au niveau du client et du serveur le besoin d'un connecteur, qui relie la couche application à la couche transport. On parle de socket.

Sockets – couche réseau

Les sockets se situent dans le modèle OSI au dessus de la couche Transport.

Ils assurent la connexion entre le transport et l'application.

Les couches jusqu'à la couche Transport sont prises en charge par le matériel et le système d'exploitation.

Modèle des sockets	Modèle OSI
Application utilisant les sockets	Application
	Présentation
	Session
UDP/TCP	Transport
IP/ARP	Réseau
Ethernet, X25, …	Liaison
	Physique

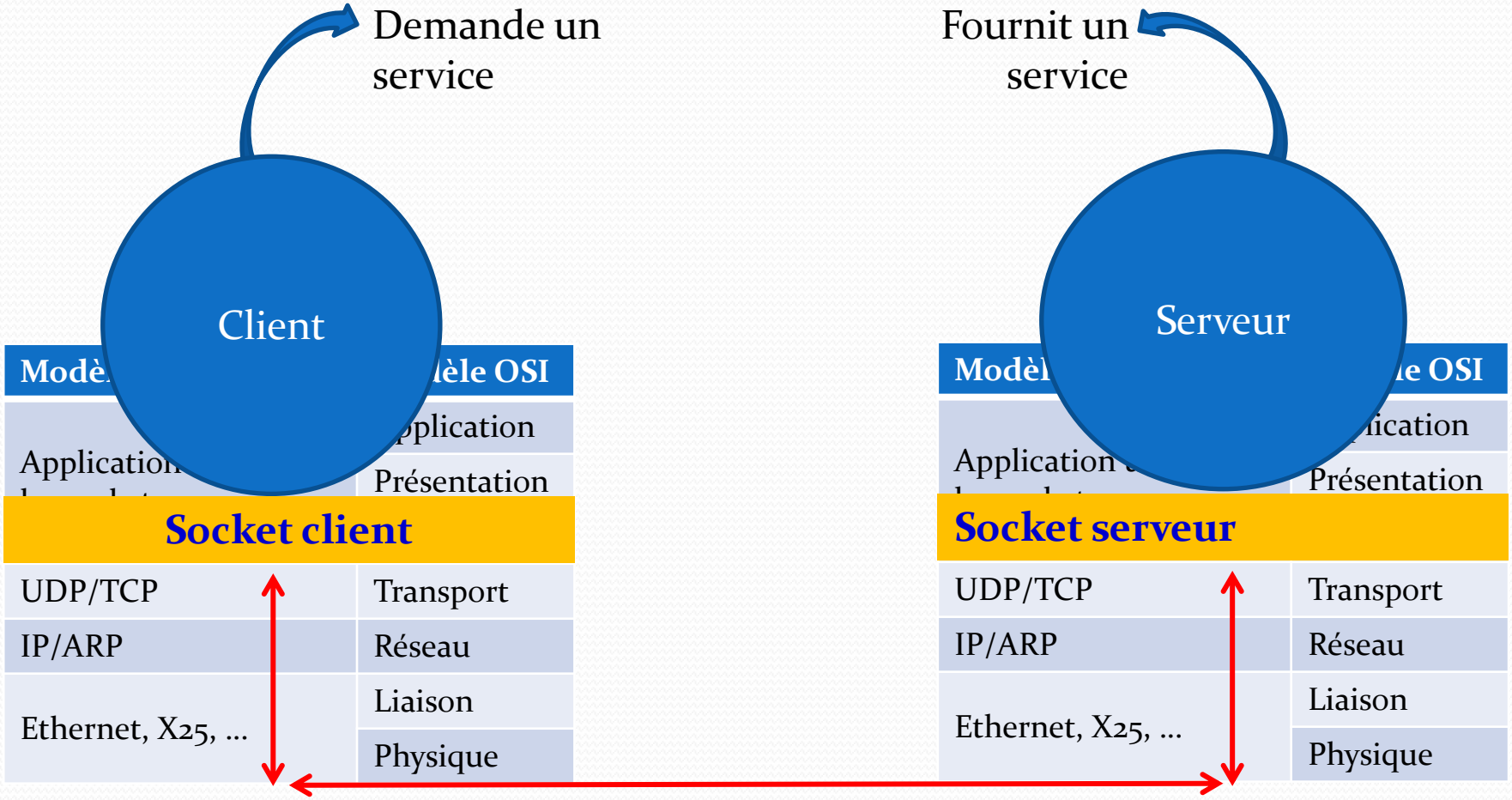
Modes de communication

Les sockets assurent la communication suivant 2 modes de communication :

- **Mode connecté :** protocole **TCP.**
- **Mode non connecté :** protocole **UDP.**

Modèle des sockets	Modèle OSI
Application utilisant les sockets	Application
	Présentation
	Session
UDP/TCP	Transport
IP/ARP	Réseau
Ethernet, X25, …	Liaison
	Physique

Client-serveur

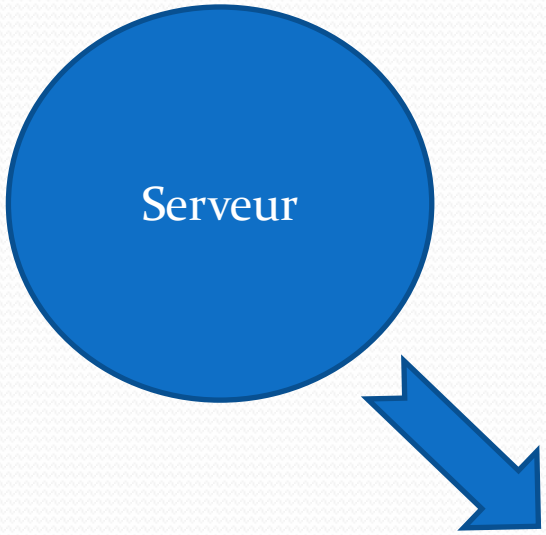


La communication s'effectue entre deux hôtes nommés **Client / Serveur.**

Langage de programmation

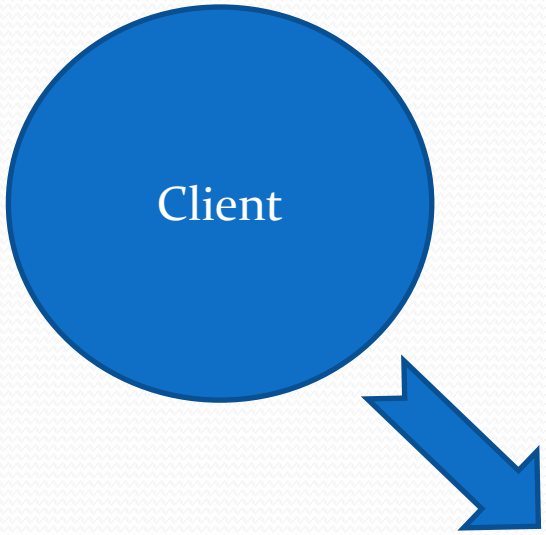
Un langage de programmation réseau possède les éléments pour gérer les clients et les serveurs sockets.

Le socket serveur



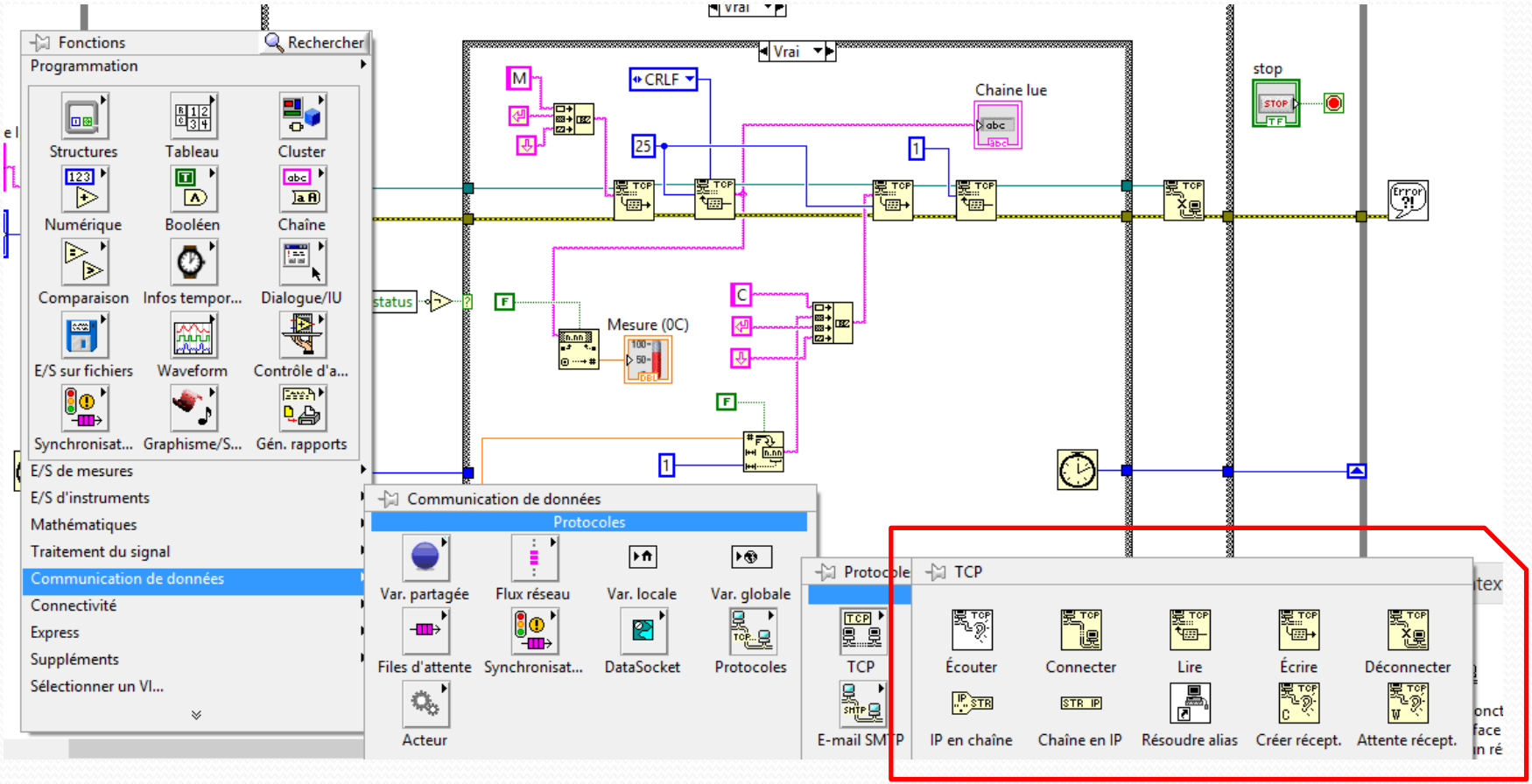
1. Création du socket serveur en lui attribuant un port.
2. Mise en écoute : attente d'une requête par un client.
3. Accepter la requête en établissant la connexion.
4. Dialoguer avec le client.
5. Fermer la connexion.

Le socket client



1. Création du socket client.
2. Demande de requête associée à un port et à une adresse Ip du serveur.
4. Dialoguer avec le serveur.
5. Fermer la connexion.

Labview – sockets TCP.



VI – Sockets TCP.

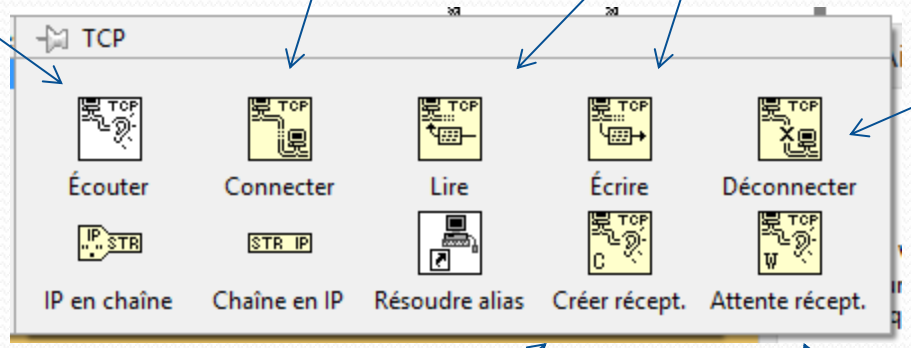
Labview – VI - sockets TCP.

Serveur : Crée un récepteur et attend une requête client

Client: demande de connexion TCP à un serveur

Serveur et client : Lire et écrire

Serveur et client : Fermer la connexion



Serveur : Crée un récepteur

Serveur : attend une requête client

Serveur – socket et application.

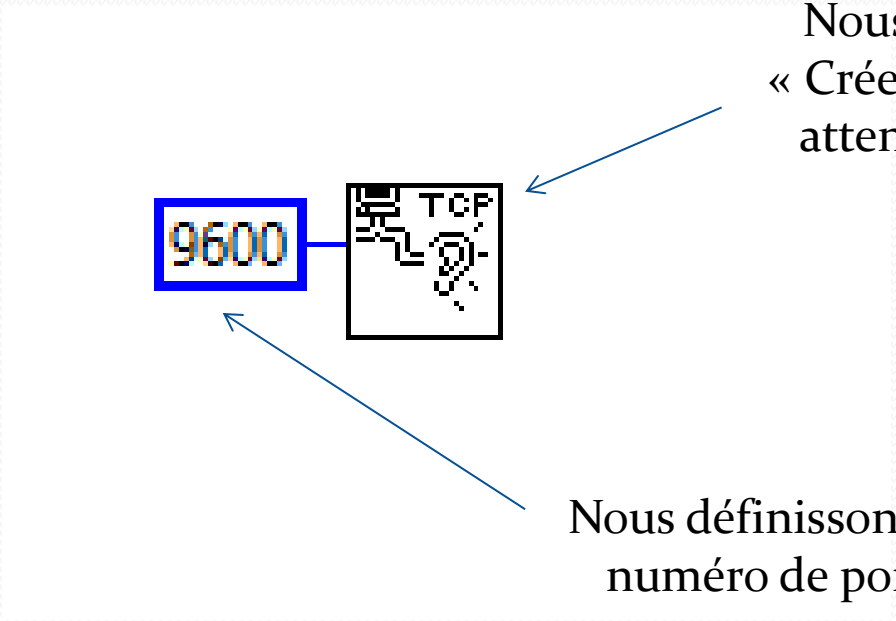
Nous allons écrire un serveur très simple. Il a le fonctionnement suivant :

- Création du récepteur et attente une requête client,
- Lors d'une requête client envoi du message « bonjour » suivi des caractères CR, LF,
- Fermeture de la connexion,
- Et fin de l'application.

Remarque : nous ne gérons pas les erreurs.

Création du serveur

Nous écrivons le VI au niveau du diagramme et laissons la face avant vide.

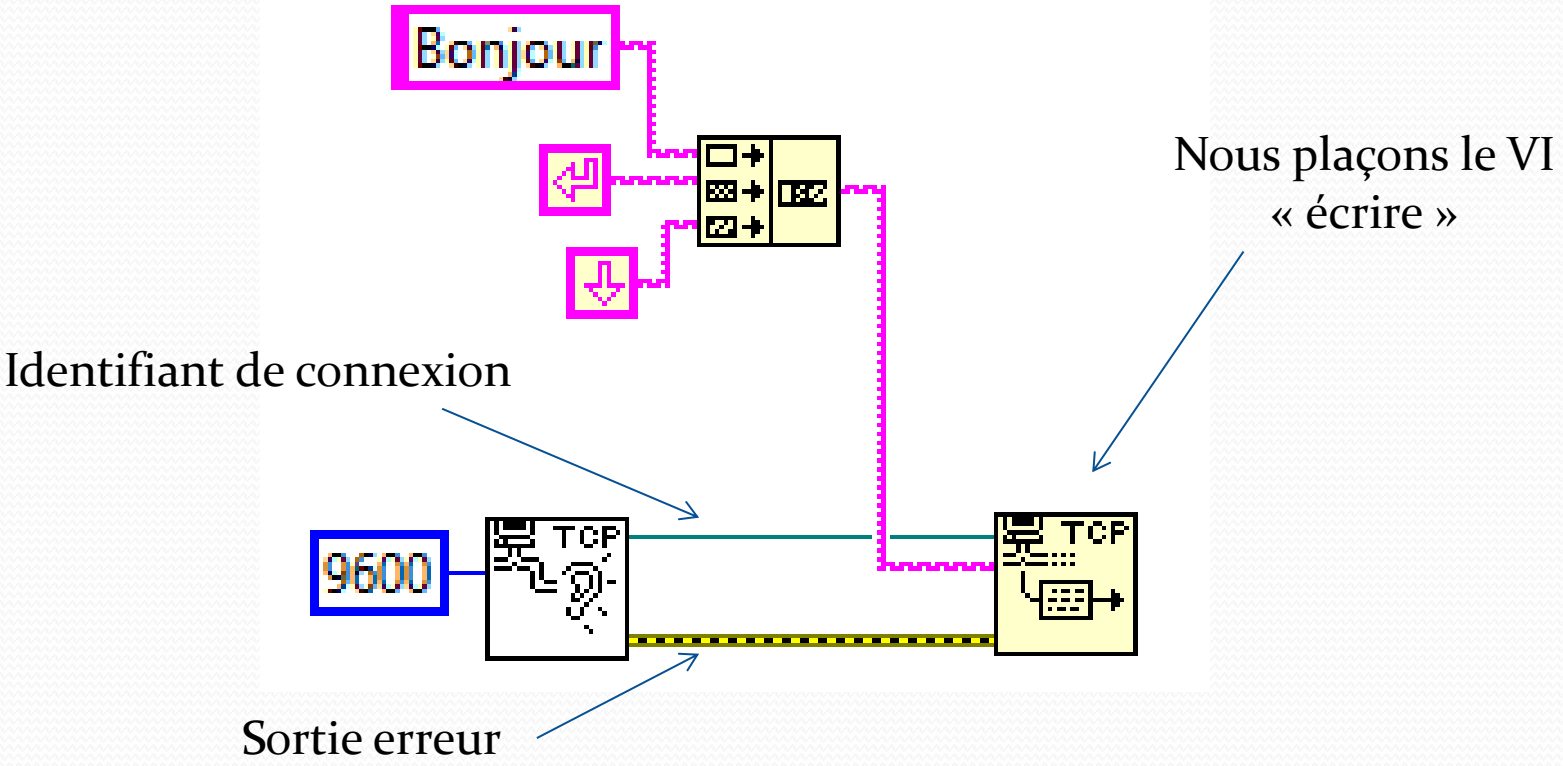


Nous plaçons le VI « Crée un récepteur et attend une requête client »

Nous définissons le numéro de port

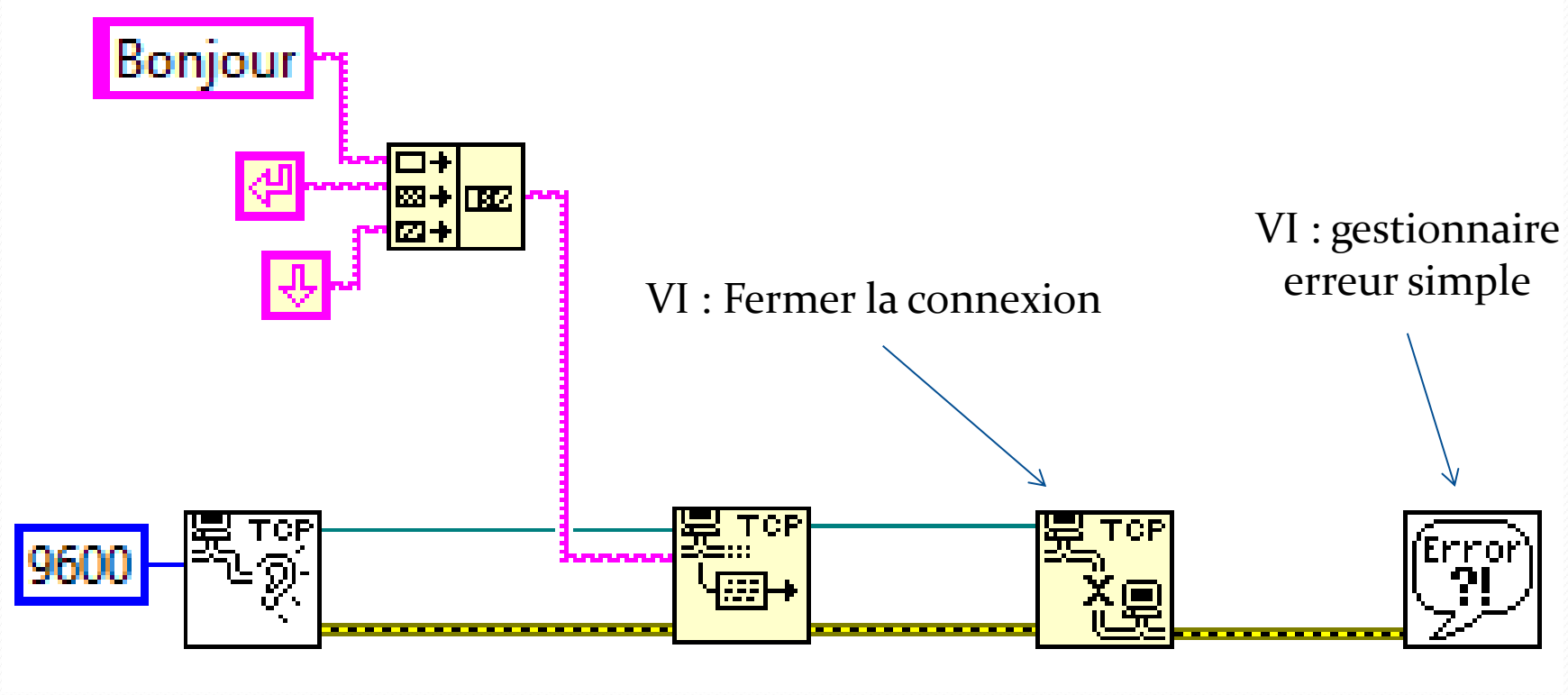
Envoi du message « bonjour » suivi des caractères CR, LF

Nous ajoutons la création d'une chaine de caractère « bonjour » suivi de CR LF et effectuons l'envoi.



Fermeture de la connexion

Nous ajoutons la fermeture de connexion et un gestionnaire d'erreur simple.

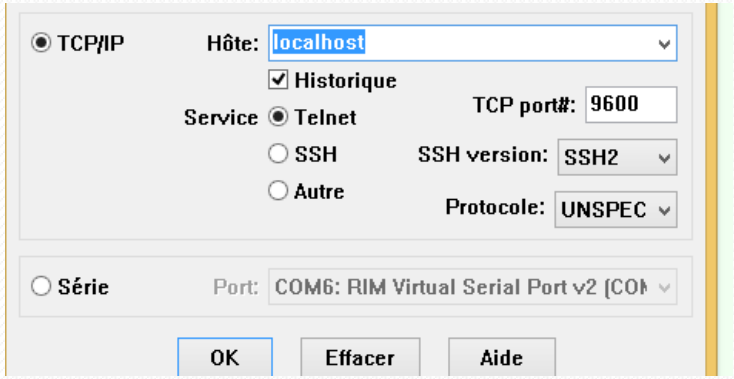


Serveur

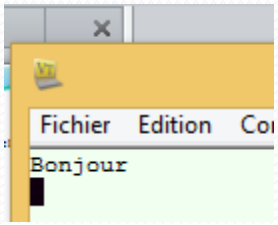
Le serveur est prêt à être utilisé. Si vous possédez « TeraTermPortable », vous pouvez tester le serveur.

1. Vous lancez le serveur :

2. Vous lancez « TeraTermPortable » et définissez la connexion : adresse Ip : localhost, port : 9600 et vous cliquez OK.



Nous voyons la réponse du serveur :



Vous enregistrez le serveur.

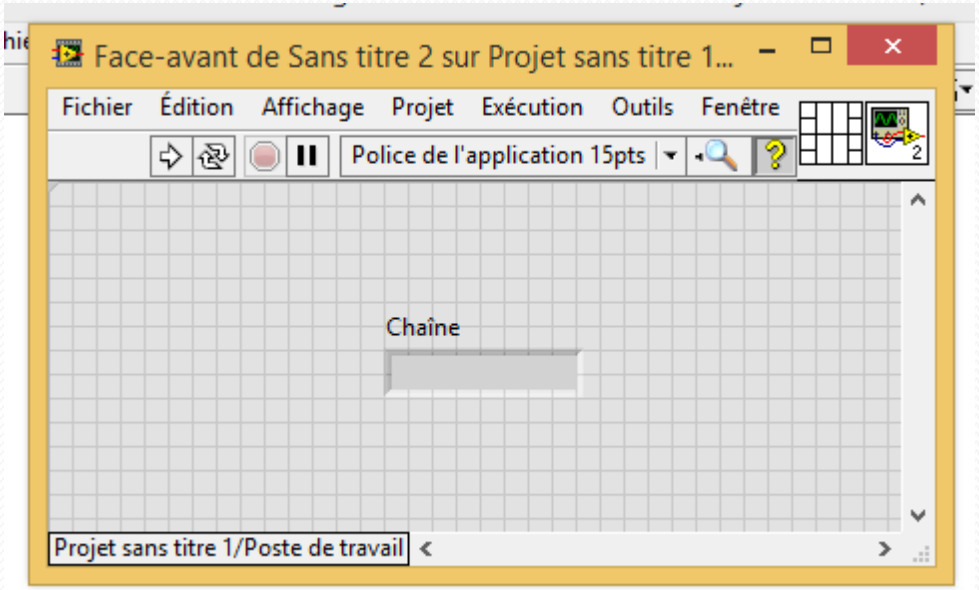
Client – socket et application.

Nous allons écrire un client, qui va avoir un fonctionnement simple :

1. Définir les paramètres de connexion et se connecter au serveur,
2. Attendre le message suivant protocole CRLF,
3. Afficher le message,
4. Fermeture de la connexion,
5. Et fin de l'application.

Création d’un client.

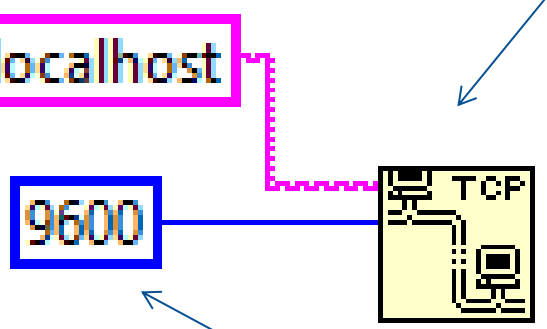
Nous allons placer sur la face avant un message, qui va afficher le message envoyé par le serveur.



Définir les paramètres de connexion et se connecter au serveur.

Nous définissons l'adresse IP ou le nom de machine du serveur. Ici localhost car le serveur est sur la machine locale.

Nous plaçons le VI « ouvrir une connexion TCP»



Nous définissons le numéro de port identique à celui du serveur

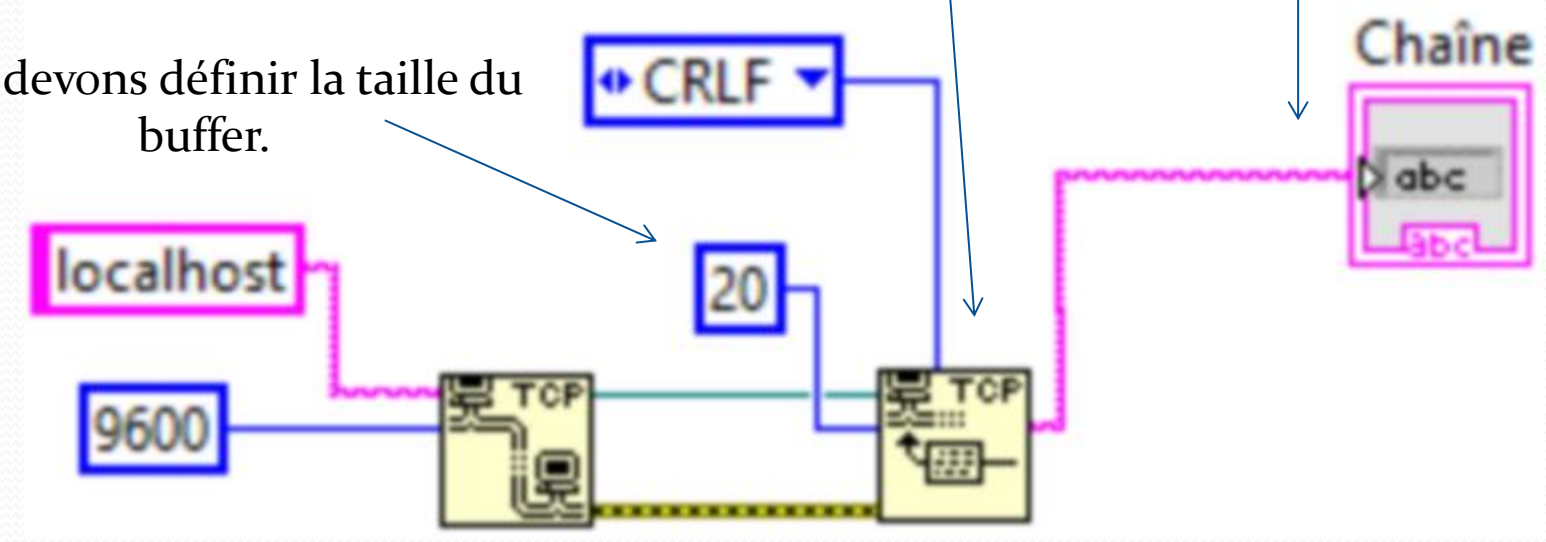
Définir les paramètres de connexion et se connecter au serveur.

Nous définissons le mode de fonctionnement du VI « Lire ».

Nous plaçons le VI « Lire»

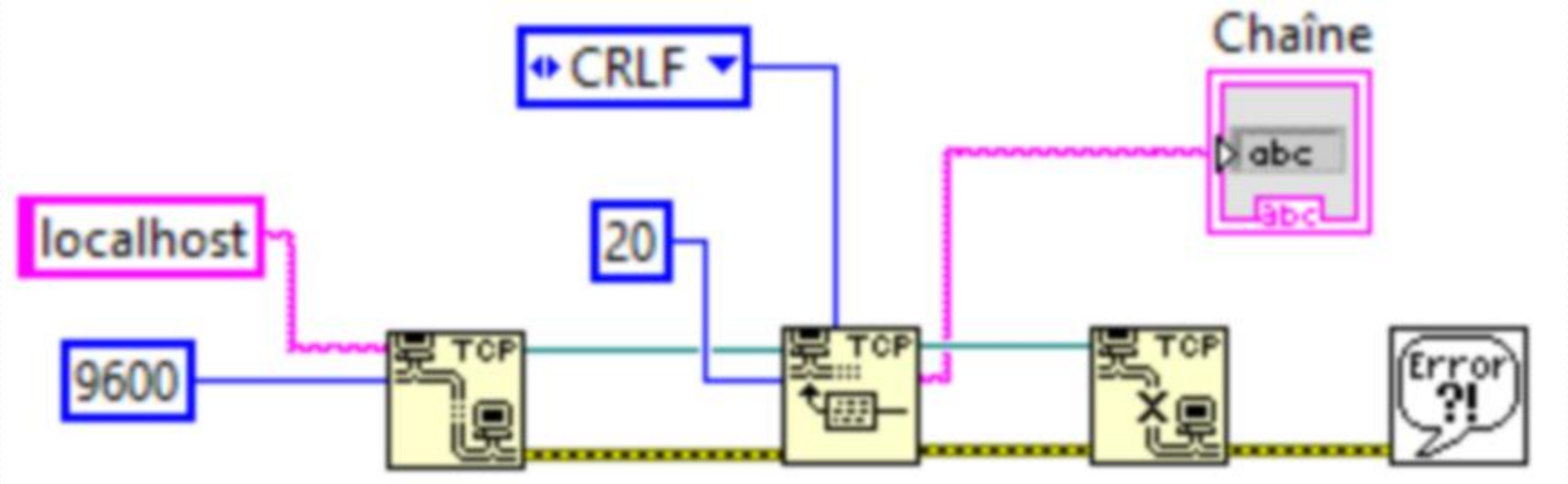
Nous connectons la chaine d'affichage

Nous devons définir la taille du buffer.



Astuce : pour définir le mode de fonctionnement, vous placez la souris sur la borne « mode », avec la touche droite vous activez créer «constante ».

Fermeture de la connexion



VI : Fermer la connexion

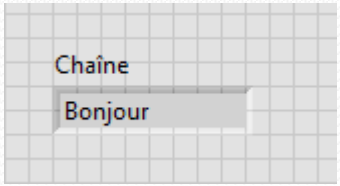
VI : gestionnaire erreur simple

Le VI client est prêt, il ne reste plus qu'à le tester.

Test du VI Client.

Le client est prêt à être utilisé.

1. Vous lancez le serveur.
2. Vous lancez le client et vous vérifiez le résultat.



Si le client affiche bonjour, alors bravo.
Vous êtes prêt à développer des applications réseau en TCP avec Labview.

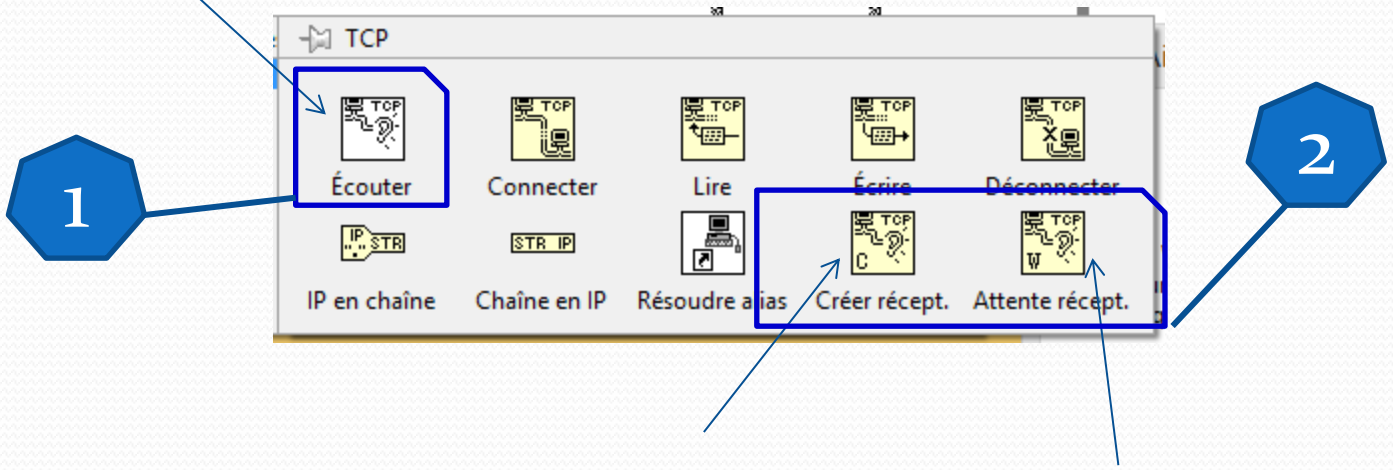
Les sockets permettent le dialogue entre un client et un serveur sur une seule machine. C'est le cas ici et cela est pratique pour tester le fonctionnement.

En principe le client et le serveur sont sur des machines différentes.

Vous pouvez essayer vos VI client et serveur sur des machines différentes.

Quelques VI particuliers : **Serveur**.

Serveur : Crée un récepteur et attend une requête client



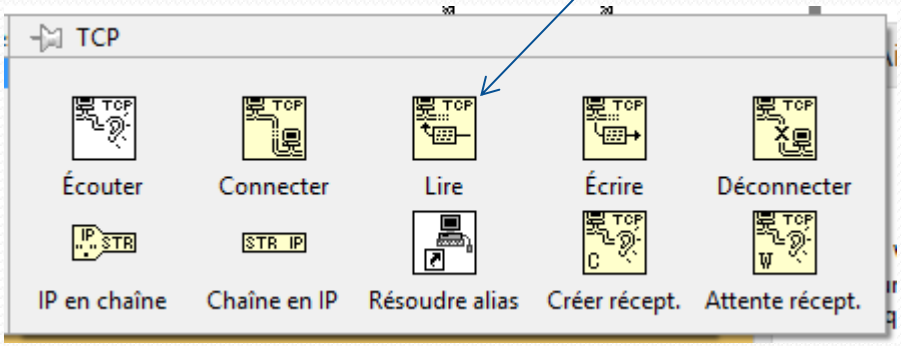
Serveur : Crée un récepteur

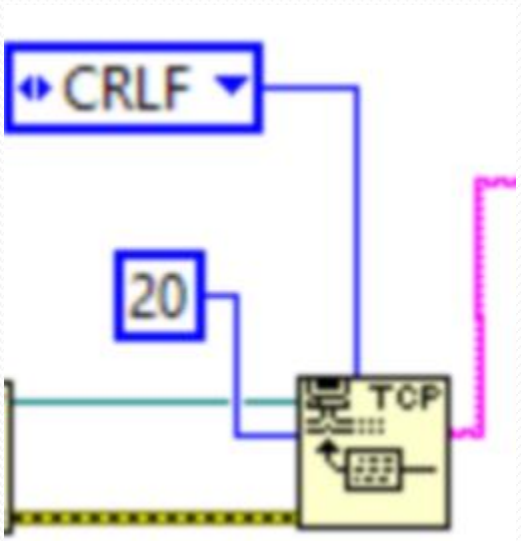
Serveur : attend une requête client

1 regroupe dans un seul VI les 2 VI de 2

Quelques VI particuliers : **Lire**.

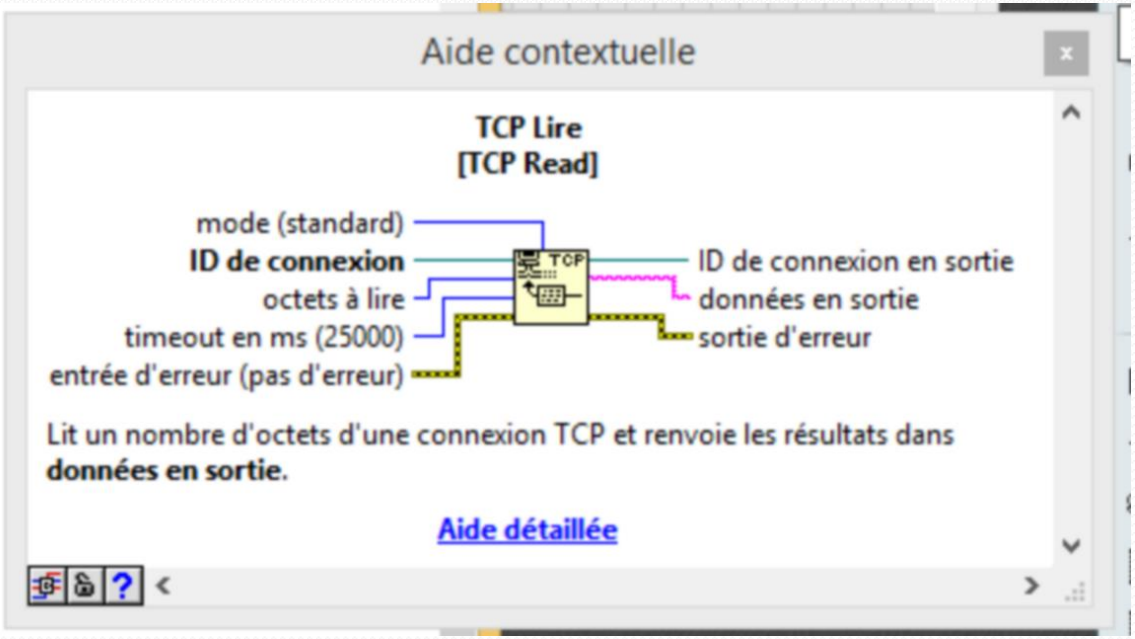
Serveur et client : Lire





Nous avons vu comment utiliser le VI « Lire » en mode CRLF.
D'autres modes sont possibles.

Quelques VI particuliers : **Lire**.



mode indique le comportement de l'opération de lecture.

0	**Standard** (valeur par défaut) — Attend que tous les octets que vous avez spécifiés dans **octets à lire** aient été reçus ou jusqu'à expiration du **timeout en ms**. Renvoie le nombre d'octets à lire jusqu'ici. Si le nombre d'octets reçus est inférieur au nombre d'octets demandés, le nombre partiel d'octets est renvoyé et un dépassement du temps imparti est signalé.
1	**Buffered** — Attend que tous les octets que vous avez spécifiés dans **octets à lire** aient été reçus ou jusqu'à expiration du **timeout en ms**. Si le nombre d'octets reçus est inférieur au nombre d'octets demandés, aucun octet n'est renvoyé et un dépassement de délai est signalé.
2	**CRLF** — Attend que tous les octets que vous avez spécifiés dans **octets à lire** soient arrivés ou que la fonction ait reçu un retour chariot (CR) suivi d'un retour à la ligne (LF) dans le nombre d'octets que vous avez spécifié dans **octets à lire**, ou attend jusqu'à expiration du temps imparti défini par **timeout en ms**. La fonction renvoie les octets jusqu'à CR et LF inclus si elle les trouve dans la chaîne.
3	**Immediate** — Attend que la fonction ait reçu l'un des octets que vous avez spécifié dans **octets à lire**. Patiente durant la totalité du timeout uniquement si la fonction ne reçoit pas d'octets. Renvoie le nombre d'octets qui ont été reçus jusqu'à présent. Signale une erreur de timeout si la fonction ne reçoit pas d'octets.

Des applications particulières à base de serveurs Flyport et Arduino seront traitées dans les prochains chapitres.

François SCHNEIDER