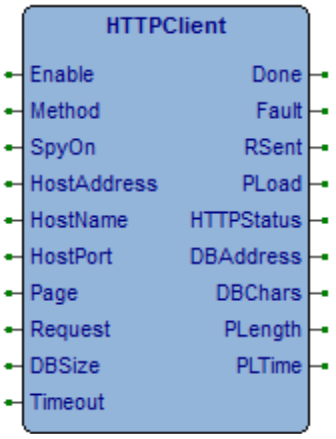
### 1.1.12 HTTPClient, HTTP client

| Type | Library |
|---|---|
| FB | eLLabNetworkLib_A700 |




Questo blocco funzione esegue la richiesta di una pagina web con il protocollo HTTP. Attivando ***Enable*** viene inviata la richiesta HTTP della pagina definita in ***Page*** all'indirizzo IP o all'URL del server definito in ***HostAddress***. E' possibile definire anche l'***HostName*** che sarà utilizzato nella richiesta (Di solito è uguale a ***HostAddress***). La pagina viene richiesta con i parametri definiti nel buffer ***Request*** passati secondo la definizione di ***Method*** (GET o POST).

In ***DBSize*** occorre definire la dimensione del buffers che l'FB alloca (Con SysRMalloc) per la gestione dei pacchetti TCP in trasmissione e ricezione. La dimensione minima è 256 bytes, aumentando la dimensione si velocizza il trasferimento (Sono effettuati meno frazionamenti). **E' inutile definire lunghezze superiori al limite del pacchetto TCP 1500 bytes**.

La pagina viene ricevuta in frammenti successivi in base al tipo di trasferimento del server, ad ogni ricezione di un frammento di pagina ***DBChars*** viene valorizzato (Per un solo loop di esecuzione) con il numero di bytes ricevuti che possono essere letti dal buffer all'indirizzo ***DBAddress***. In questo modo è possibile ricevere pagine di qualsiasi dimensione, sarà cura del programma utente trasferire i dati di pagina rticevuti in una stringa o in un file:

Al termine dell'invio della richiesta si attiva per un loop di programma l'uscita ***Rsent***, su ricezione pagina si attiva per un loop di programma l'uscita ***PLoad*** ed in ***PLength*** è ritornata la lunghezza della pagina ricevuta, mentre in ***PLTime*** il tempo necessario per l'intera richiesta.

In caso di errore esecuzione o tempo di esecuzione comando superiore al tempo definito in ***Timeout***, viene attivata per un loop di programma l'uscita ***Fault***.

L'uscita ***Done*** si attiva al termine della esecuzione della richiesta e su errore. Per acquisire nuovamente la pagina occorre disabilitare e poi riabilitare l'ingresso ***Enable***.

| | |
|---|---|
| **Enable** (BOOL) | Comando attivazione richiesta pagina. |
| **Method** (BOOL) | Metodo gestione richiesta (FALSE=GET, TRUE=POST). |
| **SpyOn** (BOOL) | Se attivo permette di spiare il funzionamento della FB. |
| **HostAddress** (@BYTE) | Indirizzo IP o URL del server a cui connettersi. |
| **HostName** (@BYTE) | Nome del server utilizzato nella richiesta (Di solito è uguale a ***HostAddress***). |
| **HostPort** (UINT) | Numero porta TCP a cui connettersi (Default 80). |
| **Page** (@BYTE)) | Stringa di definizione pagina web richiesta. |
| **Request** (@BYTE) | Indirizzo buffer dati da inviare con la richiesta. |
| **DBSize** (UINT) | Dimensione buffers Rx/Tx allocati da FB (SysRMalloc). Minimo 256 massimo 1500. |
| **Timeout** (UINT) | Timeout esecuzione richiesta pagina (mS). |
| **Done** (BOOL) | Attivo a fine esecuzione, si attiva anche in caso di ***Fault***. |
| **Fault** (BOOL) | Attivo per un loop di programma se errore gestione. |
| **RSent** (BOOL) | Attivo per un loop di programma al termine dell'invio richiesta HTTP. |
| **PLoad** (BOOL) | Attivo per un loop di programma su fine ricezione pagina.<br>***I dati di pagina non vanno acquisiti su PLoad ma ad ogni valorizzazione di DBChars.*** |
| **HTTPStatus** (STRING[64]) | Status risposta HTTP ricevuta. |
| **DBAddress** (@BYTE) | Indirizzo buffer dati ricevuti allocato da FB (Con SysRMalloc) di dimensione DBSize. |
| **DBChars** (UDINT) | Bytes di pagina ricevuti, viene valorizzato per un loop.<br>***Ad ogni valorizzazione occorre estrarre i dati di pagina da DBAddress.*** |
| **PLength** (UINT) | Dimensione pagina ricevuta. |
| **PLTime** (REAL) | Tempo impiegato per caricamento pagina (S). |

## Trigger di spy

Se ***SpyOn*** attivo viene eseguita la funzione ***SysSpyData*** che permette di spiare il funzionamento della FB. Sono previsti vari livelli di triggers.

| TFlags | Descrizione |
|---|---|
| 16#00000001 | 'Tx' Invio dati verso server HTTP |
| 16#00000002 | 'Rx' Ricezione dati da server HTTP |
| 16#00000004 | 'Rq' Stringa richiesta |
| 16#08000000 | 'Lg' Log dati di esecuzione |
| 16#10000000 | 'Pi' Informazioni di pagina |
| 16#40000000 | 'Er' Errori di esecuzione |

## Codici di errore

In caso di errore si attiva l'uscita ***Fault***, con ***SysGetLastError*** è possibile rilevare il codice di errore.

| Codice | Descrizione |
|---|---|
| 10054020 | FB eseguita in una task diversa dalla task di background. |
| 10054030 | Valore di ***DBSize*** errato. |
| 10054100 | Timeout richiesta pagina. |
| 10054200 | Errore acquisizione valore di lunghezza pagina |
| 10054210 | Errore ricezione pagina |
| 10054300 | Errore acquisizione valore di lunghezza chunk |

## Esempio ricezione pagina su stringa

Nell'esempio attivando ***Di00CPU*** viene eseguita la richiesta di una pagina sul sito altervista, sono passati in GET 2 parametri ***Dividend*** e ***Divisor***. La pagina ruichiesta è uno script PHP che esegue la divisione tra i valori passati. E' possibile testare il funzionamento dello script digitando in un browser l'indirizzo:

http:\\www.slimline.altervista.org/Mdp095a000/Ptp119b000/Division.php?Dividend=500&Divisor=10

Se lo script è attivo viene ritornata una pagina con: ***The result is: 50***

La stessa stringa ritornata nel browser è visibile nella variabile ***Result***.

### Definizione variabili

```
VAR
  CaseNr : USINT; (* Case gestione *)
  HTTPRq : HTTPClient; (* HTTP client *)
  i : UDINT; (* Auxiliary variable *)
  Result : STRING[ 32 ]; (* Result string *)
END_VAR
```

### Esempio ST *(PTP119B500, ST_HTTPClient)*

```
  (* Program initializations. *)

  IF (SysFirstLoop) THEN
    HTTPRq.SpyOn:=TRUE; (* Activate the spy *)
    HTTPRq.Method:=FALSE; (* Request method, GET *)
    HTTPRq.HostAddress:=ADR('www.slimline.altervista.org'); (* Main coordinator *)
    HTTPRq.HostName:=HTTPRq.HostAddress; (* Hostname *)
    HTTPRq.HostPort:=80; (* Server port *)
    HTTPRq.Page:=ADR('/Mdp095a000/Ptp119b000/Division.php'); (* Web page *)
    HTTPRq.Request:=ADR('Dividend=500$26Divisor=10'); (* Request string *)
    HTTPRq.DBSize:=256; (* Data buffer size *)
    HTTPRq.Timeout:=10000; (* Execution timeout *)
  END_IF;

  HTTPRq(); (* HTTP client *)

    (* Case gestione sequenze programma. *)

  CASE (CaseNr) OF

    (* Se comando attivo aspetto si disattivi. *)

    0:
    HTTPRq.Enable:=FALSE; (* HTTP get page enable *)
    IF NOT(Di00CPU) THEN CaseNr:=CaseNr+1; END_IF;

    (* Attesa comando, controllo fronte salita. *)

    1:
    IF NOT(Di00CPU) THEN RETURN; END_IF;
    i:=Sysmemset(ADR(Result), 0, SIZEOF(Result)); (* Empty result *)
    HTTPRq.Enable:=TRUE; (* HTTP get page enable *)
    CaseNr:=CaseNr+1; (* Case gestione *)

    (* Acquisizione pagina, su ricezione eseguo trasferimento in stringa. *)

    2:
    IF (HTTPRq.DBChars <> 0) THEN
      IF ((Sysstrlen(ADR(Result))+HTTPRq.DBChars) < SIZEOF(Result)) THEN
        i:=Sysmemmove(ADR(Result)+Sysstrlen(ADR(Result)), HTTPRq.DBAddress, HTTPRq.DBChars);
      END_IF;
    END_IF;

    (* Se Done fine esecuzione. *)

    IF (HTTPRq.Done) THEN
      CaseNr:=0; (* Case gestione *)
      IF NOT(HTTPRq.PLoad) THEN RETURN; END_IF; (* Se Done ma non Ok la richiesta HTTP è in errore. *)
    END_IF;
  END_CASE;

(* [End of file] *)
```

Lo script PHP sul server è del tipo:

```
<?php
  echo "The result is: ".($_REQUEST["Dividend"]/$_REQUEST["Divisor"]);
?>
```

Come si vede lo script invia in echo il risultato della divisione. I parametri posti in GET alla richiesta sono automaticamente trasferiti negli statements ***$_REQUEST['Dividend']*** e ***$_REQUEST['Divisor']***. Come si vede è quindi possibile passare in GET allo script tutti i valori che si desiderano.

Il valore di ritorno dallo script definito con lo statement echo verrà trasferito nel buffer ***Result*** del nostro programma e quindi è possibile operare su di esso con le istruzioni di gestione stringa.

## Scelta tra metodo GET o POST

Ecco una tabella che riporta le differenze tra il metodo GET ed il POST.

| | **GET** | **POST** |
|---|---|---|
| BACK button/Reload | Harmless | Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted) |
| Bookmarked | Can be bookmarked | Cannot be bookmarked |
| Cached | Can be cached | Not cached |
| Encoding type | application/x-www-form-urlencoded | application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data |
| History | Parameters remain in browser history | Parameters are not saved in browser history |
| Restrictions on data length | Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters) | No restrictions |
| Restrictions on data type | Only ASCII characters allowed | No restrictions. Binary data is also allowed |
| Security | GET is less secure compared to POST because data sent is part of the URL | POST is a little safer than GET because the parameters are not stored in browser history or in web server logs |
| Visibility | Data is visible to everyone in the URL | Data is not displayed in the URL |

Molti server (Altervista compreso) utilizzano un servizio CDN che esegue il caching delle pagine, se si usa il metodo GET per essere certi che non venga ritornata la pagina in catch è preferibile aggiungere nel request un parametro variabile (Esempio Rq=xxxx dove xxxx è un numero diverso per ogni richiesta.

## Esempio ricezione pagina su file

Nell'esempio attivando ***Di00CPU*** viene eseguita la richiesta della pagina ***index.htm*** dal sito www.slimline.altervista.org. Il contenuto della pagina è trasferito nel file ***Storage/index.htm***.

### Definizione variabili

```
VAR
  CaseNr : USINT; (* Case gestione *)
  Error : USINT; (* Execution error *)
  i : INT; (* Auxiliary variable *)
  Filename : STRING[ 32 ]; (* File appoggio pagina *)
  Fp : FILEP; (* File pointer *)
  HTTPRq : HTTPClient; (* HTTP client *)
END_VAR
```

### Esempio ST *(PTP119B500, ST_HTTPClient_ToFile)*

```
  (* Program initializations. *)

  IF (SysFirstLoop) THEN
    HTTPRq.Method:=FALSE; (* Request method, GET *)
    HTTPRq.SpyOn:=TRUE; (* Activate the spy *)
    HTTPRq.HostAddress:=ADR('www.slimline.altervista.org'); (* Main coordinator *)
    HTTPRq.HostName:=HTTPRq.HostAddress; (* Hostname *)
    HTTPRq.HostPort:=80; (* Server port *)
    HTTPRq.Page:=ADR('/index.htm'); (* Web page *)
    HTTPRq.Request:=NULL; (* Request string *)
    HTTPRq.DBSize:=256; (* Data buffer size *)
    HTTPRq.Timeout:=10000; (* Execution timeout *)
    Filename:='Storage/index.htm'; (* File appoggio pagina *)
  END_IF;

    (* ---------------------------------------------------------------------- *)
    (* Eseguo FBs. *)

  HTTPRq(); (* HTTP client *)

    (* ---------------------------------------------------------------------- *)
    (* Case gestione sequenze programma. *)

  CASE (CaseNr) OF

    (* ---------------------------------------------------------------------- *)
    (* Se comando attivo aspetto si disattivi. *)

    0:
    HTTPRq.Enable:=FALSE; (* HTTP get page enable *)
    IF NOT(Di00CPU) THEN CaseNr:=CaseNr+1; END_IF;

    (* ---------------------------------------------------------------------- *)
    (* Attesa comando, controllo fronte salita. *)

    1:
    IF NOT(Di00CPU) THEN RETURN; END_IF;
    i:=Sysremove(Filename); (* Cancellazione file *)
     HTTPRq.Enable:=TRUE; (* HTTP get page enable *)
    Error:=0; (* Execution error *)
    CaseNr:=CaseNr+1; (* Case gestione *)

    (* ---------------------------------------------------------------------- *)
    (* Acquisizione pagina, su ricezione eseguo trasferimento nel file. *)

    2:
    IF (HTTPRq.DBChars <> 0) THEN

      (* Apertura file in append. Se non esiste, viene creato. *)

      Fp:=Sysfopen(Filename, 'a'); (* File pointer *)
      IF (Fp = NULL) THEN Error:=10; CaseNr:=0; RETURN; END_IF;

      (* Eseguo scrittura su file. *)

      IF (Sysfwrite(HTTPRq.DBAddress, TO_INT(HTTPRq.DBChars), 1, Fp) <> TO_INT(HTTPRq.DBChars)) THEN
        i:=Sysfclose(Fp); (* Eseguo chiusura file *)
        Error:=20; (* Execution error *)
        CaseNr:=0; (* Case gestione *)
        RETURN;
```

```
      END_IF;

      (* Eseguo chiusura file. *)

      i:=Sysfclose(Fp); (* Eseguo chiusura file *)
    END_IF;

    (* Controllo se terminato. *)

    IF (HTTPRq.Done) THEN

      (* Se Done ma non Ok la richiesta HTTP è in errore. *)

      IF NOT(HTTPRq.PLoad) THEN Error:=50; CaseNr:=0; RETURN; END_IF;
      CaseNr:=0; (* Case gestione *)
    END_IF;
  END_CASE;

(* [End of file] *)
```