# Introduction

L’architecture de la ZPPP11 peut être schématisée de la façon suivante :

|  |
| --- |
| Couche 4 : interface utilisateur |
| Couche 3 : logiciel embarqué |
| Couche 2 : électronique |
| Couche 1 : matériel |

La ZPPP11 expose une interface utilisateur HTML accessible depuis n’importe quel client HTTP.

L’interface par elle-même est réalisée en PHP / CodeIgniter ; elle est motorisée par un serveur HTTP embarqué.

Le présent document détaille l’interface entre les couches 3 et 4.

# Principes généraux

Par exception au fonctionnement habituel des sites internet, avec lesquels elle partage les mêmes technologies, l’interface utilisateur est par essence mono-tâche (mais pas nécessairement mono-utilisateur).

Un mécanisme de type « sémaphore » est prévu afin qu’en cours de travail (processus initié par la couche logicielle en attente de terminaison), toute requête HTTP émanant de tout client renvoie la page d’attente associée au travail en cours.

Afin de faciliter toute externalisation éventuelle, les éléments exposés (noms des fichiers, codes d’erreur, etc.) doivent être rédigés en anglais.

Afin de faciliter les évolutions parallèles des couches, les messages destinés à l’utilisateur ne nécessitant pas d’interaction (ex. les messages d’erreur) doivent pouvoir être transmis directement depuis la couche logicielle. Un mécanisme spécifique est prévu afin d’en assurer la traduction.

Dans tous les cas c’est l’interface utilisateur qui initie la communication à la demande du client HTTP ; le processus général est alors le suivant :

1. Vérification de la disponibilité de la couche logicielle
2. Vérification de la configuration de la connexion
3. Vérification du travail en cours
4. Appel de l’exécutable associé à la tâche à accomplir avec les paramètres ad-hoc
5. Récupération d’un code de retour
6. Lecture cyclique du fichier destiné à recevoir le résultat de l’exécution et retour du contenu affichable associé

# Répertoires standards

Trois répertoires sont définis :

* /hardconf
* /conf
* /model
* /bin

Les fichiers de configuration contenus dans ces répertoires (et leurs sous-répertoires) sont structurés en Json (RFC 4627), postfixés par « .json » et encodés - sans BOM - en UTF-8 (RFC 3629). Les dates sont codées suivant la norme ISO-8601.

(pour mémoire JSON met en œuvre un stockage de type « dictionnaire » associant une clef à une valeur, directement tiré des tableau associatifs du Javascript dont il est issu)

Lorsque la structure des informations n’est pas spécifiée par une RFC, la bonne pratique consiste à regrouper les valeurs par thème n’excédant pas une centaine d’éléments et à les stocker dans un fichier portant le nom du thème.

A l’intérieur de chaque fichier, les clefs des paires respectent une notation pointée garantissant la lisibilité et l’univocité des informations ; ex. :

Fichier : NetworkConfiguration.json

{

"Subnets.client.nodes.2.physical\_interfaces.eth2.name":"Broadcom",

"Subnets.client.nodes.2.physical\_interfaces.eth2.status":"UP"

}

## Répertoire /hardconf

Ce répertoire contient :

* Les informations insusceptibles de modification (ex. : version logicielle, etc.) ;
* Les réglages par défaut qui peuvent être copiés dans le répertoire /conf ;
* Les fichiers de traductions des messages envoyés par la couche logicielle

Ces fichiers sont accessibles exclusivement en lecture.

Ce répertoire est situé sous la racine de mise à jour (son contenu est intégralement remplacé lorsqu’une nouvelle version logicielle est déployée).

Ce répertoire contient un fichier intitulé « Software.json » contenant :

{"Version":"*identifiant de la version en cours*"}

## Répertoire /conf

Ce répertoire contient l’ensemble des informations courantes, éventuellement copiées depuis le répertoire /hardconf

Ces fichiers sont accessibles en lecture par l’interface utilisateur, en lecture/écriture par la couche logicielle. Le système d’exploitation garantit la gestion des verrous nécessaires à l’intégrité des accès.

Au premier allumage, ce répertoire est vide et est initialisé en recevant une copie du fichier « Software.json ».

Ce répertoire n’est pas situé sous la racine des mises à jour (son contenu est inchangé lorsqu’une nouvelle version logicielle est déployée – mais peut être modifié par programme lors de l’intialisation subséquente sur la base d’une éventuelle différence entre les contenus des fichiers « Software.json »des répertoires /hardconf et /conf).

Ce répertoire content le fichier « Connection.json » qui contient les paramètres de configuration de la connexion.

## Répertoire /model

Ce répertoire est destiné à stocker le modèle en cours de traitement. Le contenu du répertoire devra être effacé par la couche logicielle à chaque démarrage et pourra l’être via l’interface utilisateur.

Ce répertoire contient les vues rendues à partir du modèle 3d, il doit être accessible depuis la racine du site.

## Répertoire /bin

Ce répertoire est destiné à stocker les exécutables de la couche logiciel. Ce répertoire est situé sous la racine de mise à jour (son contenu est intégralement remplacé lorsqu’une nouvelle version logicielle est déployée).

# Traductions

Le répertoire /hardconf contient des sous-répertoires nommés d’après les codes des langues supportées par l’interface utilisateur (au sens de la norme ISO 639-1), ex. :

* /hardconf/de
* /hardconf/en
* /hardconf/fr
* …

Chaque répertoire contient des fichiers nommés d’après les contextes associés aux textes qu’ils contiennent, ex. :

* ArduinoFaillure.json
* HardwareFaillure.json
* …

Chaque fichier contient un tableau codé en JSON associant un identifiant et le code HTML (pouvant contenir des balises) destiné à être présenté à l’utilisateur, ex. :

/hardconf/fr/ArduinoFaillure.json

{

"USB.NoResponse":"Une erreur interne ne permet pas l’initialisation de votre imprimante.<br><br>Merci de prendre contact avec le <a href=\"http://fr.zeepro.ch/contacter-le-service-apres-vente.aspx\">service après-vente</a>."

}

Chaque répertoire doit contenir une version de chacun des fichiers dans la langue correspondante. Si la traduction dans la langue courante est introuvable le message est affiché dans sa version anglaise. Si la version anglaise du message est introuvable, l’identifiant du message est affiché à la place.

Ce dispositif de traduction est indépendant de celui mis en œuvre par l’interface utilisateur elle-même (le travail de traduction dépendant fortement du contexte, l’uniformisation des mécanismes n’est, pour le moment, pas un objectif).

# Messages

Afin de rendre les couches aussi indépendantes que possibles, la couche logicielle pourra communiquer directement à l’utilisateur les messages qu’elle aura défini (essentiellement les messages d’erreur).

Un message est une structure JSON de la forme :

{"Context":"*nom du fichier sans extension*", "Id":"*identifiant*"}

Ex ; : {"Context":"ArduinoFaillure", "Id":"USB.NoResponse"}

(l’interface utilisateur détermine le patron et la langue dans laquelle le message doit être affiché)

# Fichier d’échange

L’exécution d’une commande est un processus asynchrone piloté grâce au fichier d’échange intitulé « Work.json » stocké dans le répertoire /conf.

La couche logicielle est appelée par l’interface utilisateur à l’aide d’une ligne de commande accompagnée des paramètres suivants :

* Version du fichier d’échange
* URL devant être rappelée
* Autres paramètres nécessaires à l’exécution de la tâche.

Le programme appelé effectue alors les opérations suivantes :

* Création ou écrasement du fichier « Work.json » avec le contenu :

{

"Version":"*version du fichier*",

"CommandLine":"*ligne de commande complète envoyée au système d’exploitation* ",

"Sate":< ”Working” | “Paused” | “Halted” | “Done” > ,

"Cancel":"ligne de commande devant être exécutée",

"PauseOrResume":"ligne de commande devant être exécutée",

"CallBackURL":" *URL devant être rappelée (passée en paramètre)* ",

"Message": *message destiné à l’utilisateur*

}

* Renvoie d’un code de succès (0) (le cas d’un code d’échec sera traité ultérieurement)
* Réalisation de la tâche par elle-même

Tant que le travail n’est pas achevé, la valeur de « State » est fixé à « Working » et la couche logicielle met à jour, le cas échant, le contenu de « Message » (par exemple pour indiqué un pourcentage d’avancement).

Si la valeur de « Cancel » est définie, l’interface utilisateur propose un bouton d’annulation dont l’activation provoque l’appel en ligne de commande de la valeur spécifiée. La couche logicielle a la responsabilité d’interrompre la tâche en cours, de passer la valeur de « State » à « Halted » et de spécifier le message de confirmation ad-hoc avant de retourner un code de succès. Si la valeur de « Cancel » n’est pas définie, le travail ne peut pas être interrompu.

Si la valeur de « PauseOrResume » est définie, l’interface utilisateur propose un bouton dépendant de la valeur de « State » (« Suspendre » si « Sate » vaut « Working », « Reprendre » si « State » vaut « Paused ») dont l’activation provoque l’appel en ligne de commande de la valeur spécifiée. La couche logicielle a la responsabilité de mettre en pause ou de reprendre le travail en cours et de spécifier le message as-hoc avant de retourner un code de succès. Si la valeur de « PauseOrResume » n’est pas définie, le travail ne peut pas être suspendu.

En cas d’échec, « State » est fixé à « Halted » et « Message » contient le message d’erreur correspondant.

Lorsque le travail est achevé, « State » est fixé à « Done » et « Message » peu contenir un message à destination de l’utilisateur.

Si « Work.json » existe au lancement de la couche logicielle, « Sate » doit valoir « Done » ; dans le cas contraire l’état de /conf est jugé incohérent.

Lorsqu’une requête parvient à l’interface utilisateur, celle-ci vérifie (étape 3 du processus général) - si « Work.json » existe et que « Sate » vaut « Working » - que l’URL correspond à « CallBackURL ». Dans le cas contraire elle effectue une redirection vers cette valeur. Ce mécanisme garantie le caractère mono-tâche de l’interface.

La version du fichier d’échange permet de faire fonctionner des couches de maturités différentes.

La valeur de « CommandLine » est définie à des fins d’analyse.

# Fichier de connexion

Les informations de connexions de l’imprimante sont regroupées dans le fichier « Connection.json » stocké dans le répertoire /conf.

Sa structure est la suivante :

{"Connection.Topology":< "Network" | "P2P" >,

"Connection.Support":< ”Wi-Fi” | “RJ45” >,

"Wi-Fi.Mode":< ”AP” | “Single” >,

"Wi-Fi.SSID":"*SSID du réseau Wi-Fi* ",

"Wi-Fi.Encryption":< ”None” | ”WEP” | “WPA” | “WPA2” >,

"Wi-Fi.Password":"*mot de passe du réseau Wi-Fi*",

"IP.addresses.V4":[{“Address”:"*adresse n°1 IPV4*",

“Gateway":” *adresse n°1 IPV4*",

“Mask":” *masque n°1 IPV4*"},

{“Address”:"*adresse n°2 IPV4*",

“Gateway":” *adresse n°2 IPV4*",

“Mask":” *masque n°3 IPV4*"},

…

],

"IP.lease":*date d’expiration du bail DHCP*

}

Si ce fichier est absent (lors de la première installation, ou après chaque réinitialisation des paramètres de connexion qui provoque son effacement), le système :

* paramètre la connexion réseau en fonction du fichier « Connection.json » contenu dans le répertoire /hardconf et ayant le contenu :

{"Connection.Topology":"Network",

"Connection.Support":”Wi-Fi”,

"Wi-Fi.Mode":”AP”,

"Wi-Fi.SSID":"ZeePro Initialisation",

"Wi-Fi.Encryption":"None",

"IP.address.V4":[{“Address": "169.254.0.1",

“Mask":"255.255.0.0"},

{“Address”: ”10.0.0.1”,

“Mask":"255.0.0.0"},

{“Address”: ”172.16.0.1”,

“Mask":"255.240.0.0"},

{“Address”: ”192.168.0.1”,

“Mask":"255.255.0.0"},

],

}

* et redirige toutes les requêtes vers la page d’initialisation.

En fonction du choix de l’utilisateur, le fichier est ensuite recréé dans le répertoire /conf à l’aide des informations éventuellement fournies.

# Initialisation de la couche logicielle

Seule la communication pouvant intervenir entre la couche logicielle et l’interface utilisateur au moment de l’initialisation de la couche logiciel est abordée ci-après ; l’initialisation par elle-même fait l’objet de la RFC …

Pour mémoire, le processus de mise à jour est antérieur à toute autre action (en particulier le lancement du serveur HTTP permettant d’accéder à l’interface utilisateur).

Afin de permettre une éventuelle remonté d’erreur à l’utilisateur, la couche logicielle respecte le protocole suivant :

* Création ou écrasement du fichier « Boot.json » avec le contenu :

{

"Version":"*version du fichier*",

"Message": *message destiné à l’utilisateur*

}

* Réalisation des tests successifs nécessaires à la suite l’initialisation et mise à jour, le cas échéant, du contenu de « Message ».
* Effacement du fichier « Boot.json »

Lorsqu’une requête parvient à l’interface utilisateur (étape 1 du processus général), la présence du fichier « Boot.json » provoque l’affichage du message associé, ce qui permet de faire patienter l’utilisateur pendant la phase d’initialisation, de l’informer d’une éventuelle erreur ou de mettre en évidence la dernière étape atteinte en cas de blocage.

# Autres cas d’erreur

Les erreurs doivent normalement être prises en charge grâce aux contenus des fichiers « Boot.json » et « Work.json » ; mais, par exception, certains autres cas peuvent survenir :

* Le retour d’un code d’erreur de la couche logiciel à l’interface utilisateur
* Une défaillance de l’interface utilisateur (HTTP 500).

Ces cas reflètent des dysfonctionnements graves probablement insusceptibles de résolution par l’utilisateur et provoquent l’affiche d’un message d’erreur spécifique.

L’appel d’une URL inconnue provoque une redirection vers la page d’accueil de la langue préférée du client HTTP.

# API

En un premier temps, plusieurs modèles ne pourront pas être traités simultanément.

## Fichiers

### /model/Model.json

{"File":"*nom du fichier contenant le modèle*",

"xTrans":*translation sur l’axe x (en unité du modèle)*,

"yTrans":*translation sur l’axe y (en unité du modèle)*,

"zTrans":*translation sur l’axe z (en unité du modèle)*,

"xRot":*rotation autour de l’axe x (en degrés)*,

"yRot":*rotation autour de l’axe x (en degrés)*,

"zRot":*rotation autour de l’axe x (en degrés)*,

"Factor":*facteur de grossissement,*

"Facets":*nombre de facettes*

}

## Commandes

### Effacement du modèle en cours

Ligne de commande :

ClearModel

Paramètres :

-callbackurl url\_de\_redirection

Exemple :

ClearModel -callbackurl /wait

Provoque l’effacement du répertoire /model

Cette commande est appelée au chargement d’un nouveau modèle.

### Chargement d’un nouveau modèle

Ligne de commande :

LoadModel

Paramètres :

-file nom\_du\_fichier

-callbackurl url\_de\_redirection

Exemple :

LoadModel -file mickey.stl -callbackurl /wait

Efface le répertoire /model, charge le nouveau modèle dans ce même répertoire, le valide, créer et renseigne Model.json.

### Déplacements, rotations et changement de taille

Ligne de commande :

ChangeModel

Paramètres :

-xtrans translation sur l’axe x (en unité du modèle) (entier, optionnel)

-ytrans translation sur l’axe y (en unité du modèle) (entier, optionnel)

-ztrans translation sur l’axe z (en unité du modèle) (entier, optionnel)

-xrot rotation autour de l’axe x (en degrés) (entier, optionnel)

-yrot rotation autour de l’axe x (en degrés) (entier, optionnel)

-zrot rotation autour de l’axe x (en degrés) (entier, optionnel)

-factor facteur de grossissement (réel, optionnel)

-callbackurl url\_de\_redirection

Exemple :

ChangeModel –yrot 180 –f 0.5 -callbackurl /wait

effectue une rotation de 180° sur l’axe des y et divise par deux la taille de l’objet

Met à jour le fichier Model.json.

### Rendu

Ligne de commande :

RenderModel

Paramètres :

-callbackurl url\_de\_redirection

Exemple :

RenderModel -callbackurl /wait

Génère un rendu du modèle courant dans une image PNG dont le nom est constitué à partir des paramètres de Model.json :

/model/modelxt*xt*yt*yt*zt*zt*xr*xr*yr*yr*zr*zr*f*f*.png

(dans le cas de l’exemple précédent : /model/modelxt0yt0zt0xr0yr180zr0f0.5.png)

Les images sont conservées jusqu’à l’effacement du modèle ; la précision du facteur de grossissement est de 6 chiffres après la virgule.

### Reset

Ligne de commande :

ResetModel

Paramètres :

-callbackurl url\_de\_redirection

Exemple :

ResetModel -callbackurl /wait

Réinitialisation Model.json.

### Normalise

Ligne de commande :

NormalizeModel

Paramètres :

-callbackurl url\_de\_redirection

Exemple :

NormalizeModel -callbackurl /wait

Calcule et affecte au fichier Model.json les translations nécessaires à l’alignement de son point bas sur 0, et au centrage de sa projection sur le plan XY par rapport au centre du plateau.

### Print

Ligne de commande :

PrintModel

Paramètres :

-callbackurl url\_de\_redirection

Exemple :

PrintModel -callbackurl /waitprint

Lance l’imrpiession du modèle et renvoie vers la page /waitprint qui propose l’annulation ou la pose de l’impression

Imprime le modèle. Cette commande doit renseigner les champs "Cancel" et "PauseOrResume" du fichier Work.json.

### CancelPrint

Ligne de commande :

CancelPrintModel

Paramètres :

-callbackurl url\_de\_redirection

Exemple :

Cancelprintmodel -callbackurl /wait

Interrompt l’impression en cours.