Atos Worldline
HIGH-TECH TRANSACTIONAL SERVICES

Sips Payment Web JAVA 6.15
Guide du programmeur

Version 2.01 – Octobre 2010

REACH YOUR TARGETS »

Contact
By phone: +33 (0)811 107 033
By fax: +33 (0)811 107 033
By email: sips@atosorigin.com

Sips
e-Payment Solution
by Atos Worldline

www.atosworldline.com

Avertissements :

- **Le fichier Version.txt précise l'environnement dans lequel l'API a été compilée et testée. L'installation de l'API sur tout autre environnement n'est pas garantie.**
- **Afin de faciliter l'installation de l'API Java, les noms des packages de classes ont été uniformisés. Pour cette raison, l'API version 600 et ultérieure n'est pas entièrement compatible avec les précédentes.**

Sommaire

1. INTRODUCTION

Ce document vous décrit les différentes classes disponibles dans l'API Sips Payment Web JAVA, quel est leur rôle et comment les utiliser. Pour intégrer l'API, vous devrez développer au moins 3 Servlets.

L'API Sips Payment Web comprend 3 classes principales :

- SipsApiWeb
- SIPSCallParm
- SIPSResponseParm

Notez que SIPSResponseParm et SIPSCallParm dérivent toutes deux d'une même classe SipsDataObject.

Note : ce document ne décrit pas comment vous interfacer avec votre système d'information ou votre base de données. Dans les exemples fournis, les variables sont déjà renseignées, vous devrez programmer la lecture et la mise à jour des données de votre système d'information.

Logiciels pré-requis :

- Connaissances de base du langage Java.
- Connaissances de base de l'écriture et de la compilation des Servlets.
- Compilateur Java pour construire vos Servlets.
- Moteur de Servlets pour exécuter vos Servlets.

Conventions d'écriture :

- Les renvois à d'autres documentations seront notés en majuscules et en italique.
  ex *: DICTIONNAIRE DES DONNEES*
- Les classes, attributs et méthodes de l'API seront notées en italique.
  ex : *SipsApiWeb, transaction_id, sipsPaymentCallFunc*
- Les Servlets que vous devez développer seront notées en gras
  ex : **RequestServlet**

2. LA REQUÊTE DE PAIEMENT : REQUESTSERVLET

2.1 CLASSE SIPSCALLPARM

Cette classe permet d'instancier l'objet qui contient toutes les données de la transaction. On passe d'ailleurs ce dernier en paramètre à la méthode *sipsPaymentCallFunc* pour générer le code HTML qui permet à l'internaute de se connecter au Serveur Sips .

Par conséquent, pour envoyer une requête de paiement au serveur Sips , vous devrez instancier un objet *SIPSCallParm*.

Les attributs de *SIPSCallParm* sont de deux types différents : obligatoires ou facultatifs.

Il n'est pas nécessaire de renseigner tous les attributs de cet objet dans la **RequestServlet**. La méthode *initCall* de l'objet *SipsApiWeb* effectue des affectations statiques d'après les fichiers parmcom. Cette méthode affecte également aux champs *block_align*, *block_order*, *currency_code* et *header_flag* des valeurs par défaut si ils ne sont renseignés ni dans les fichiers parmcom, ni dans la servlet **RequestServlet**.

Pour positionner un attribut directement, il suffit d'utiliser la méthode *setValue(Key, Value)* de l'objet *SIPSCallParm*. Notez que tout attribut positionné par *setValue()* préemptera sur toutes les autres valeurs par défaut.

Référez-vous aux sources livrées pour des exemples concrets.

Pour déboguer vos Servlets, *SIPSCallParm* dispose d'une méthode *getValue(key)* qui vous retourne la valeur positionnée dans l'objet, ainsi que d'une méthode *check()* qui va contrôler tous les paramètres de l'objet *SIPSCallParm* en vue d'un appel de l'API.

Ces méthodes sont données à seule fin de déboguage. La méthode *check()* est évidemment appelée par l'API elle-même, il n'est donc pas utile de l 'appeler dans votre **RequestServlet** une fois vos développements terminés.

Reportez-vous au *DICTIONNAIRE DES DONNEES* pour la description des différents attributs, leur utilité, ainsi que les Keys à utiliser dans *setValue()* pour les positionner.

2.2 CLASSE SIPSAPIWEB

2.2.1 Le fichier pathfile

Le fichier pathfile est le fichier de configuration des autres fichiers paramètres de l'api. Seul paramètre du constructeur de la classe *SipsApiWeb*, il permet ensuite aux différentes méthodes d'accéder à tous les autres fichiers paramètres, soit le parmcom.defaut, le parmcom.011223344551111 et le fichier certificat. Le fichier pathfile contient également le chemin internet des logos des moyens de paiement.

Le chemin à fournir au constructeur doit être un chemin physique (en aucun cas un chemin internet). On recommande un chemin absolu, comme c'est le cas dans les sources d'exemples de SERVLETS livrés.

La syntaxe des lignes de ce fichier est la suivante :

- Chaque ligne commençant par # est une ligne de commentaires.
- Les chemins sont paramétrés par « mot-clé!chemin! »

Les trois paramètres essentiels sont les trois variables F_CERTIFICATE, F_PARAM, et F_DEFAULT qui doivent contenir respectivement les chemins complets des fichiers certif.fr.011223344551111.jsp, parmcom.011223344551111, et parmcom.defaut. Seul le fichier parmcom.defaut doit être paramétré avec son extension .defaut. Les extensions des autres fichiers seront ajoutées par l'API d'après les paramètres renseignés dans l'objet *SIPSCallParm* (méthode *SipsPaymentCallFunc*), ou la chaîne de caractères *cyphered (méthode SipsPaymentResponseFunc)*.

Il est recommandé de placer des chemins complets dans ces variables. De plus, ces chemins doivent être des chemins physiques du serveur, et non pas des chemins internet.

Le paramètre F_CTYPE doit contenir le type du certificat, celui-ci correspond à l'extension du fichier certificat qui sera ici jsp. On renseignera alors F_CTYPE!jsp!

Le dernier paramètre D_LOGO sert à l'affichage des moyens de paiement. Lui seul doit être renseigné avec une portion de chemin internet, soit l'URL du répertoire des logos de l'API sur le site web, amputée du nom de domaine de ce dernier. (cf. *GUIDE D'INSTALLATION*).

Le dernier paramètre DEBUG sert à l'affichage des moyens de paiement en mode DEBUG.

2.2.2 Le mode DEBUG

Le mode DEBUG permet d'afficher (en format HTML) l'environnement complet de l'api, ainsi que tous les paramètres de la requête et de la réponse d'un paiement.

Pour activer ce mode, le paramètre DEBUG du fichier pathfile doit être renseigné à YES.

Le mode DEBUG est constitué de différentes rubriques indiquant les ressources utilisées par l'api. Il permet ainsi de vérifier par exemple le contenu du fichier pathfile, l'URL du serveur de paiement ou encore la valorisation des champs de la requête de paiement.

En phase de production, ce mode devra être désactivé (renseignement de la valeur du paramètre DEBUG autre que YES).

2.2.3 Constructeur

Prototype : *public SIPSApiWeb(String pathFile)*

Paramètres : *String pathFile* le chemin du fichier pathfile.

Le constructeur de cette classe prend en argument le chemin du fichier pathfile, pour retourner une instance de la classe *SipsApiWeb*.

2.2.4 Méthode sipsPaymentCallFunc

Prototype : *public String sipsPaymentCallFunc (SIPSDataObject sipsData)*

Valeur retour : La méthode retourne une chaîne de caractères HTML

Paramètres : *SIPSDataObject* *sipsData* l'objet *SIPSCallParm* de requête.

Cette méthode a pour objet de générer du code HTML contenant :

- La liste des logos des moyens de paiement.
- L'URL du serveur Sips ,
- des données cachées à transmettre au serveur Sips .

Ce formulaire, retourné sous forme de chaîne de caractères par la méthode sipsPaymentCallFunc, doit-être ensuite affiché sur le poste de l'internaute.

Les blocs d'icônes de paiement proposés par la boutique incluent chacun un en-tête en option (cf. *GUIDE DE PERSONNALISATION*).

Cette méthode prend en entrée un objet *SIPSCallParm* contenant les données obligatoires de la transaction (*merchant_id, merchant_country, amount, …*).

Le service Sips Payment Web propose ses propres valeurs par défaut (par exemple, s*ipsPaymentCallFunc* génèrera automatiquement un *transaction_id* par la méthode *GetTransactionId* si l'attribut *transaction_id* de *SIPSCallParm* est vide).

La méthode *sipsPaymentCallFunc* construit ensuite un message crypté à partir de toutes les données que vous aurez renseignées, et le signe par un MAC (Message Authentication Code), un scellement calculé avec un algorithme DES. Ce message crypté apparaît (en variable cachée) dans le formulaire retourné par la méthode. Il sera posté sur l'URL du serveur Sips , et permettra d'identifier le commerçant et d'authentifier le message.

Lorsqu'une erreur survient, une *SipsException* est levée.(cf. *MESSAGES D'ERREUR DE L'API*)

2.2.5 Méthode initCall

Cette méthode est appelée par s*ipsPaymentCallFunc* en début de traitement. Elle initialise l'objet *SIPSCallParm* à partir des fichiers de paramètres parmcom et des valeurs de l'objet *SIPSCallParm* renseignées par la méthode *setValue()*.

Les fichiers paramètres sont lus séquentiellement dans l'ordre suivant : parmcom.defaut, parmcom.011223344551111. Ainsi, si des valeurs sont positionnées dans les deux fichiers, elles seront écrasées au fur et à mesure de la lecture de ces derniers. La méthode *initCall()* écrasera ensuite ces valeurs si elles ont été renseignées dynamiquement par la méthode *setValue*() (l'activation du mode DEBUG permet de s'assurer de la valeur finale des paramètres de la requête de paiement).

Par conséquent, vous avez la possibilité de renseigner les champs de la requête de deux façons différentes :

- Statiquement en renseignant un des fichiers parmcom.
- Dynamiquement en utilisant la méthode *setValue(Key, Value)* de l'objet *SIPSCallParm*.

Enfin, référez-vous au *DICTIONNAIRE DES DONNEES* pour connaître les noms des attributs de l'objet *SIPSCallParm* à utiliser *dans setValue()* et les mots-clés des fichiers parmcom. Attention, mots-clés et noms d'attributs ne sont pas nécessairement identiques (TEMPLATE dans le parmcom et *setValue(templatefile, « mon_template »)* par exemple).

2.2.6 Méthode getTransactionId

Prototype : *public static String getTransactionId()*

Valeur retour : La méthode retourne un numéro de transaction

Paramètres : Aucun.

Cette méthode calcule un numéro de transaction en se basant sur l'heure système (format hhmmss). La référence obtenue est conforme au format de l'attribut *transaction_id* (6 caractères numériques). Le résultat de l'appel de cette méthode peut donc être conservé directement dans l'attribut *transaction_id* de l'objet *SIPSCallParm*.

Cette méthode peut être considérée comme une fonction utilitaire et pour cette raison, elle est facultative.

WARNING : Comme cette méthode s'appuie sur l'heure système pour générer un numéro de transaction, on notera deux inconvénients : d'abord on ne couvre que 86400 nombres (24h*60mn*60s) et non pas toute la plage possible de numéros de transaction (000000 à 999999). Ceci peut être gênant pour de très gros sites, le numéro de transaction devant être unique sur une journée. D'autre part, il subsiste un risque de doublons, si deux internautes distincts souhaitant payer au même instant appellent **RequestServlet** dans la même seconde. Une des deux transactions sera alors nécessairement rejetée comme un doublon, pour la même cause d'unicité que précédemment.

Il sera donc toujours préférable de gérer un numéro de séquence pour l'attribut *transaction_id* sur le site web commerçant, plutôt que d'utiliser cette méthode.

3. LA REPONSE A UN PAIEMENT : RESPONSESERVLET

3.1 CLASSE SIPSRESPONSEPARM

Cette classe permet d'instancier l'objet qui contiendra tous les éléments de la réponse de la transaction.

On y retrouve les attributs généraux de la transaction tels *merchant_id*, *amount*, et *currency_code*, mais aussi les champs renseignés par le serveur Sips suite à la demande d'autorisation : *response_code, payment_certificate*, etc…

Tous ces attributs sont décrits plus précisément dans le *DICTIONNAIRE DES DONNEES*.

Le serveur Sips renvoie les données de la transaction en méthode POST et de façon cachée. En fait elle « poste » une variable DATA cryptée sur l' URL de votre **ResponseServlet**. La méthode *sipsPaymentResponseFunc* de l'objet *SipsApiWeb* décrypte cette variable et retourne un objet *SIPSResponseParm* avec tous ses attributs renseignés.

Après avoir fait appel à *sipsPaymentResponseFunc*, vous pouvez accéder à l'ensemble des paramètres par la méthode *getValue(Key)* de *SIPSResponseParm*.

3.2 CLASSE SIPSAPIWEB

3.2.1 Constructeur

C'est bien sûr le même qu'au paragraphe 2.2.3.

3.2.2 Méthode sipsPaymentResponseFunc

Prototype	**:** *public SIPSDataObject sipsPaymentResponseFunc(String cyphered)*	
Valeur retour	**:** L'objet SIPSResponseParm de réponse	
Paramètres	**:** *String cyphered*	La variable "DATA" cryptée retournée par le serveur Sips .

La méthode *sipsPaymentResponseFunc* a pour but de décrypter les données renvoyées par le serveur Sips .

En effet, après la validation des coordonnées bancaires de l'internaute, le serveur Sips affiche en cas d'acceptation de la transaction une page de réponse. Cette page résume les données de la transaction, et présente un bouton « RETOUR A LA BOUTIQUE » qui reconnecte l'internaute à l'URL *normal_return_url*. Dans le cas d'un refus de l'autorisation par les serveurs bancaires, le serveur Sips affiche une page de refus avec un bouton « ANNULATION – RETOUR A LA BOUTIQUE » qui renvoie l'internaute vers l'URL *cancel_return_url*.

Le bouton « ANNULATION – RETOUR A LA BOUTIQUE » présent sur la page de saisie des coordonnées carte bancaire à le même effet que son homonyme de la page de refus.

Dans chacun de ces cas, une variable cachée DATA contenant la chaîne de caractères cryptée des données de la transaction est postée sur l'URL *normal_return_url* ou *cancel_return_url*.

Ce mode permet au commerçant de reconnecter l'internaute à son site, puisque le développeur a une totale liberté de paramétrage de ces URL. Il faut malgré tout s'assurer que le serveur commerçant autorise la réception de données en méthode POST.

Dans la **ResponseServlet**, il faut donc commencer par récupérer cette variable avec une commande *request.getParameter("DATA");*

Cette chaîne de caractères est le seul paramètre de la méthode *sipsPaymentResponseFunc*.

L'appel à *sipsPaymentResponseFunc* retourne un objet *SIPSResponseParm* contenant les variables renvoyées par le serveur Sips . Ces variables sont d'une part les données de réponse de la transaction, et d'autre part les éventuelles données passées par le commerçant dans sa requête d'appel.

Lorsqu'une erreur survient lors du décryptage, une *SIPSException* est levée.

Les attributs de l'objet *SIPSResponseParm* ne sont complètement renseignés que dans le cas où la valeur de l'attribut *response_code* est 00 (transaction acceptée). Ainsi, dans le cas où l'attribut *response_code* aurait la valeur 05 (paiement refusé), les attributs de de l'objet *SIPSResponseParm* propres à l'autorisation bancaire ne seront pas renseignés.

On utilise souvent la même **ResponseServlet** sur les URL *normal_return_url* et *cancel_return_url*, mais cela n'a rien d'obligatoire.

Les développements à effectuer sur cette Servlet sont au libre choix du commerçant, puisqu'il s'agit essentiellement de reconnecter l'internaute à son site. On peut par exemple tester le code retour de la transaction pour afficher un message de confirmation à l'internaute, lui proposer un autre moyen de paiement tel le chèque dans le cas d'un code 05 (paiement refusé), ou le rediriger tout simplement vers la page d'accueil du site.

Les champs dédiés au commerçant (retournés sans modifications par le serveur Sips) tels *return_context* ou *caddie* sont utiles dans cette **ResponseServlet** pour restaurer le contexte de l'internaute. Il est déconseillé de passer des variables de session sur l'URL, car leur présence est susceptible de perturber le bon fonctionnement de l'API.

3.3 CLASSE DEBUGOUTPUTSTREAM

Cette classe permet de rediriger la sortie HTML du mode DEBUG de la sortie standard vers un navigateur Internet.

En effet, en mode réponse, l'affichage des données du mode DEBUG se fera sur la sortie standard du serveur d'application où sont installé les servlets **AutoresponseServlet** et **ResponseServlet** (fichiers traces …).

3.3.1 Constructeur

Aucun. La classe DebugOutputStream est basée sur une instance static permettant son unicité au sein de l'environnement de l'API.

3.3.2 Méthode setOut

Prototype : *public static void setOut (OutputStream os)*

Valeur retour : Aucune

Paramètres : *OutputStream os*

Flux de sortie vers lequel le mode DEBUG sera redirigé (normalement la sortie des servlets **AutoresponseServlet et ResponseServlet**)

Cette méthode permet de rediriger les données du mode DEBUG vers la sortie des servlets **AutoresponseServlet** et **ResponseServlet** (et donc dans un navigateur Internet).

4. LA REPONSE AUTOMATIQUE : AUTORESPONSESERVLET

4.1 POURQUOI DEUX REPONSES

Comme expliqué au paragraphe précédent, l'appel des URL *normal_return_url* (ou *cancel_return_url*) est déclenché par l'internaute, lorsqu'il clique sur le bouton « RETOUR A LA BOUTIQUE » du ticket électronique ou « ANNULATION – RETOUR A LA BOUTIQUE » du ticket ou de la page de saisie des coordonnées bancaires. Par conséquent, il se peut qu'aucune de ces URL ne soit appelée lors d'un paiement, pour peu que l'internaute perde sa connexion, ou tout simplement ferme son navigateur sans avoir cliqué sur un de ces boutons. Le commerçant risque alors d'avoir une commande initialisée dans sa base de données, mais qui ne serait pas confirmée avant la réception de son journal des transactions le lendemain.

Pour pallier à ce cas, le serveur Sips renvoie une réponse automatique de la transaction en même temps qu'il affiche la page de réponse à l'internaute.

Ce mode permet au commerçant d'être informé systématiquement en temps réel de toutes les transactions qui sont effectuées sur son site.

4.2 PRINCIPE DE FONCTIONNEMENT

La réponse automatique ressemble en tous points à la réponse manuelle, ie le serveur poste la même variable DATA cryptée que celle de la réponse manuelle sur l'URL automatic_response_url.

La différence majeure est que l'envoi de cette réponse automatique est fait par le serveur Sips , et non pas par le navigateur de l'internaute.

L'absence de navigateur implique des différences du point de vue du traitement de cet appel. La servlet **AutoResponseServlet** ne peut pas faire d'affichages, ni de redirections (pas de navigateur à l'écoute). On ne peut y effectuer que des traitements automatiques, tels une mise à jour de base de données, envoi de mail, ou tout simplement sauvegarde des données dans un fichier comme cela est présenté dans l'exemple.

4.3 IMPLEMENTATION DE AUTORESPONSESERVLET

Le développement de **AutoResponseServlet** est très proche de celui de **ResponseServlet**.

Il faut commencer par récupérer la variable DATA cryptée, puis la décrypter grâce à la méthode *SipsPaymentResponseFunc* de l'objet *SipsApiWeb*.

Ceci effectué, on se trouve en possession d'un objet *SIPSResponseParm* contenant toutes les données de la transaction.

Le traitement qui suit est au libre choix du commerçant, en fonction de son système d'information.

Si votre **AutoresponseServlet** ne fonctionne pas, consultez le chapitre *MESSAGES D'ERREUR DE L'API*.

5. MESSAGES D'ERREUR DE L'API

Vous trouverez ci-dessous les messages d'erreur qui apparaissent le plus souvent durant les phases d'installation, et la manière de les résoudre.

5.1 LORSQUE L'ON APPELLE L'API

Le message s'affiche sur la page sur laquelle on devrait voir les logos des cartes dans ce cas, les logos ne s'affichent pas.

Un message d'erreur (ex : Error in call parameters structure (merchant_id not filled)) est constitué de deux parties :

- le message d'erreur (ex : Error in call parameters structure)
- le code diagnostic (ex : merchant_id not filled)

Ci-dessous vous trouverez les principaux messages renvoyés par l'API.

Message	cause	solution
Error reading pathfile (chemin du fichier pathfile)	L'API ne peut pas ouvrir le fichier «chemin du fichier pathfile»	Vérifiez le chemin du fichier pathfile.
Error reading pathfile permission denied (chemin du fichier pathfile)	L'API ne peut pas ouvrir le fichier «chemin du fichier pathfile»	Vérifiez les droits de lecture du fichier pathfile.
Error reading pathfile (no keyword *F_CERTIFICATE*)	Le fichier pathfile ne contient pas le mot clé *F_CERTIFICATE*	Vérifier la présence du mot clé *F_CERTIFICATE* dans le fichier pathfile
Error reading pathfile (no keyword *F_PARAM*)	Le fichier pathfile ne contient pas le mot clé *F_PARAM*	Vérifier la présence du mot clé *F_PARAM* dans le fichier pathfile
Error reading pathfile (no keyword *F_DEFAULT*)	Le fichier pathfile ne contient pas le mot clé *F_DEFAULT*	Vérifier la présence du mot clé *F_DEFAULT* dans le fichier pathfile
Error reading pathfile (no keyword *D_LOGO*)	Le fichier pathfile ne contient pas le mot clé *D_LOGO*	Vérifier la présence du mot clé *D_LOGO* dans le fichier pathfile
Error reading default parameters definition (chemin du fichier parmcom.defaut)	L' API ne peut ouvrir le fichier « chemin du fichier parmcom.defaut»	Vérifiez le chemin du fichier.
Error reading default parameters definition permission denied (chemin du fichier parmcom.defaut)	L' API ne peut ouvrir le fichier « chemin du fichier parmcom.defaut »	Vérifiez les droits de lecture du fichier.
Error reading merchant parameters definition (chemin du fichier parmcom.011223344551111)	L'API ne peut ouvrir le fichier (chemin du fichier parmcom.011223344551111)	Vérifiez le chemin du fichier.
Error reading merchant parameters definition permission denied (chemin du fichier parmcom.011223344551111)	L'API ne peut ouvrir le fichier (chemin du fichier parmcom.011223344551111)	Vérifiez les droits de lecture du fichier.
Error open certificate file (chemin du fichier certif.fr.011223344551111.jsp)	L'API ne peut pas ouvrir le fichier certificat « chemin du fichier certif.fr.011223344551111.jsp »	Vérifiez le chemin du certificat. Vérifiez également la cohérence entre les paramètres merchant_country, merchant_id, et le nom de votre fichier certificat. Vérifier dans le pathfile le paramètrage du mot clé F_CTYPE à jsp

Message	cause	solution
Error open certificate file permission denied (chemin du fichier certif.fr.011223344551111.jsp)	L'API ne peut pas ouvrir le fichier certificat « chemin du fichier certif.fr.011223344551111.jsp »	Vérifiez les droits de lecture du certificat.
Error invalid separator in file (chemin du fichier pathfile)	Le fichier concerné a une syntaxe incorrecte (ici, le fichier pathfile)	Vérifiez les lignes du fichier cité dans le message. Il manque un ou plusieurs séparateurs. « ! »
Error in call parameters structure	Un des paramètres est invalide, regarder la fin du message pour en connaître le nom et la cause.	Agir en fonction du message d'erreur et du format de la donnée. (Se référer au *DICTIONNAIRE DES DONNEES*)
Error Invalid Key	La clé utilisée dans setValue ou getValue est incorrecte	Se référer au *DICTIONNAIRE DES DONNEES*
Autres messages		Contacter le Centre d'Assistance Technique

Si le message d'erreur contient un chemin de fichier, ce dernier dépend de votre système d'exploitation :

- Windows : "c:\\repertoire\\pathfile"
- Unix : "/home/repertoire/pathfile"

5.2 LORSQUE L'API APPELLE LE SERVEUR SIPS

Le serveur Sips affiche ces messages d'erreur. Pour les erreurs graves, elles sont affichées en rouge sur un fond jaune. Dans les autres cas, elles sont affichées sur la page HTML standard de paiement. En mode « démonstration », les messages d'erreur peuvent s'afficher sur fond vert. Le message affiché fourni le paramètre en erreur.

Message	Cause	Solution
Security error	MAC reçu incorrect	Vérifiez le chemin et le nom du certificat dans le fichier pathfile
Invalid Transaction	Un des paramètres de paiement n'est pas valide (mauvais format)	Vérifiez le format des paramètres de la transaction
	message modifié	fraude suspectée, Appelez l'équipe technique
	Un des paramètres de paiement ne correspond pas à votre configuration bancaire	Vérifiez la cohérence entre votre contrat vente à distance et votre transaction (devise acceptée par exemple)
Transaction already processed	L'identifiant de transaction a déjà été utilisé dans la journée	Configurer un transaction_id unique sur une journée pour chaque paiement
Autres messages		Contacter le Centre d'Assistance Technique

5.3 LA REPONSE AUTOMATIQUE NE FONCTIONNE PAS

- L'URL doit être accessible depuis un accès Internet extérieur. Un login/password, ou un firewall sont susceptibles de bloquer l'accès à votre serveur.
- Vous ne devez pas utiliser de protocole https.
- Contrôlez les log d'accès de votre serveur (historique).

- Si vous utilisez un port non standard, ce dernier doit se situer dans la plage de 80 à 9999.
- Vous ne devez pas passer de variables de session avec l'URL, pour transférer des informations utiliser les champs CADDIE (2048 caractères) et RETURN_CONTEXT (256 caractères).
- Si vous avez validé le fait que le serveur Sips appelle bien votre URL *automatic_response_url*, il y a sans doute une erreur de programmation dans votre **AutoresponseServlet**. La similitude entre les réponses manuelle et automatique permet de déboguer facilement **AutoresponseServlet**. Il suffit pour cela de renseigner l'URL *normal_return_url* avec l'URL de **AutoresponseServlet**. Ainsi, cette dernière sera déclenchée par le bouton « RETOUR A LA BOUTIQUE » du ticket électronique, et ce dans un navigateur. La présence de ce navigateur vous permettra d'ajouter des affichages tout au long du traitement pour tracer les commandes défectueuses. Bien entendu, ces affichages seront à supprimer quand vous paramétrerez **AutoresponseServlet** dans l'URL *automatic_response_url*.
- Enfin, si votre code de réponse automatique fonctionne quand il est appelé via l'URL de réponse manuelle, mais ne fonctionne pas via l'URL de réponse automatique, il y a probablement un cookie, une redirection, ou toute autre chose nécessitant la présence d'un navigateur dans le code de **AutoResponseServlet**.