M2Msoftphone

Manuel développement

Notre référence : MAN-2007-SP-01
Version : 1.1
Date : 11/01/2007

M2MSOFT - Cap Delta, Parc Technologique Delta Sud, 09340 Verniolle, France
Tel : +33 561 69 00 85 – Fax : +33 561 50 02 32 – Email : contact@m2msoft.com - http://www.m2msoft.com

ISOMMAIRE

INTRODUCTION

Le **M2Msoftphone Kit** est un kit de **développement rapide** de **softphones, switchboards, postes d'alertes, interphones**, etc.

Sa base complète en terme de protocoles, de média et d'outils graphiques vous permet de **vous concentrer uniquement sur les fonctions et le visuel souhaité.**

Avantages :

M2Msoftphone Kit accélère vos développements d'outils de téléphonie sur IP.
Entièrement JAVA, portable sur de multiples environnements:

- Multi standards VoIP **H323/SIP.**
- Multi-canaux.
- Fonctions avancées précodées.
- Moteur de plugins.

Configuration minimale de l'ordinateur

Système d'exploitation :
Windows NT, XP (SP2 minimum), 2000 ou Linux x86

Processeur :
1 Ghz ou plus

RAM :
256 Mo (Windows NT, 2000 et Linux x86), 512 Mo (Windows XP).

Java Runtime Environment :
1.5 minimum.

4.1 Installation

L'installation du softphone se fait par le biais de l'installateur du produit.

todo copie d'ecran de l'installeur

4.2 Arborescence du produit

Une fois installé le softphone génère l'arborescence suivante :

```
softphone\
        |
        |--------->conf
        |             |--------->config.ini
        |--------->lib
        |             |--------->cygwin1.dll (uniquement pour windows)
        |             |--------->gimssdk.jar
        |             |--------->load.jar
        |             |--------->mava-4.5.7.jar
        |             |--------->media(.exe)
        |             |--------->preferences.jar
        |--------->media (les fichiers doivent être au format G711 ALAW
        |             |          et sans entête).
        |             |--------->blank.sw
        |             |--------->vorak.sw
        |--------->plugins
        |             |--------->fichiers plugin
        |             |--------->stack
        |             |             |--------->fichiers stack VoIP
        |--------->sounds (les fichier doivent être au format wav ou au)
        |             |--------->busy.wav
        |             |--------->out.wav
        |             |--------->ring.wav
        |--------->spy [ répertoire pour les enregistrements audio
        |                 (prochainement disponible) ]
        |--------->Launcher.class
        |--------->lic.txt
        |--------->softphone.jar
```

4.3 Description des fichiers principaux

config.ini

Description des champs non configurables par la fenêtre de préférences :

- DESCRIPTION
  Description de la licence.
- ENABLED_RTP
  Précise si la licence autorise ou non l'envoi et la réception du flux RTP.
  Les valeurs possibles sont 1(activé) ou 0(désactivé).
- MAX_CHANNELS
  Nombre maximum d'appels entrant et sortant autorisé par la licence.
- DEBUG_FILE
  Nom du fichier recevant les traces, si le nom est vide et que l'affichage des traces est activé alors elles apparaîtrons sur la console.
- FULL_DEBUG
  Affichage des traces du moteur de plugins lors du chargement des différents jar.
  Les valeurs possibles sont 1(activé) ou 0(désactivé).
- LOADER_JAR=lib/load.jar
  Archive java prenant en charge les notifications pendant le chargement des différents plugins (ne pas mettre d'anti-slash '\' comme séparateur de fichiers).

Load.jar

Archive java prenant en charge les notifications pendant le chargement des différents plugins (voir l'exemple d'exploitation).

mava-4.5.7.jar

Archive java contenant le SDK du softphone, ainsi que quelques objets graphique permettant de simplificer le développement d'IHM.

preferences.jar

Fenêtre de préférences du softphone.

softphone.jar

Point d'entré du softphone.

lic.txt

Fichier de licence du softphone.

5.1 Description

Le loader permet au développeur d'être notifié des différentes erreurs possible lors du chargement des plugins et ainsi afficher le visuel de son choix.

Pour que le softphone puisse identifier le loader, celui-ci doit respecter 4 contraintes :

- Il doit implémenter l'interface de l'objet mava.softphone.Loadable.
- Il doit renseigner dans le fichier Manifest de son jar, le champs Main-Class et le faire pointer sur la classe implémentant l'interface mava.softphone.Loadable.
- Placer son archive java dans un répertoire du softphone (autre que le répertoire plugins).
- Renseigner son emplacement dans le champ LOADER_JAR du fichier de configuration (conf/config.ini).

5.2 Exemple de loader

Cet exemple contient 2 classes com.m2m.softphone.load.Load (prend en charge le visuel) et com.m2m.softphone.load.LoadInit (implémente l'interface mava.softphone.Loadable et fait vivre le visuel).
Note : Toutes les erreurs de type mava.softphone.Loadable.FataError déclenchent un exit de l'application.

5.2.1 Load.java

```
package com.m2m.softphone.load;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;

import javax.swing.BorderFactory;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JProgressBar;


public class Load extends JFrame {
        private static final long       serialVersionUID    = 4547781800502453587L;
        private JLabel                  lblInfo             = null;
        private JProgressBar            loadBar             = null;
        private int                     maxTask             = 0;
        private int                     currentTask         = 0;
        private JPanel                  panel               = null;
```

```
/**
 * Constructeur par défaut.
 *
 * @param title
 *          titre de la fenêtre.
 * @param maxTask
 *          nombre maximum de tâches.
 */
public Load(String title, int maxTask) {
        super(title);
        if (maxTask <= 0)
                maxTask = 1;
        this.maxTask = maxTask;
        initialize();
}

/**
 * Cette méthode initialise la fenêtre.
 */
private void initialize() {
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        panel = new JPanel();
        panel.setBorder(BorderFactory.createEtchedBorder(Color.DARK_GRAY,
                        Color.BLACK));
        setUndecorated(true);
        panel.setLayout(new BorderLayout());
        panel.add(getLblInfo(), BorderLayout.SOUTH);
        panel.add(getLoadBar(), BorderLayout.CENTER);
        setContentPane(panel);
        pack();
        setLocationRelativeTo(null);
        setAlwaysOnTop(true);
}

/**
 * Cette méthode initialise le label information.
 *
 * @return javax.swing.JLabel.
 */
private JLabel getLblInfo() {
        if (lblInfo == null) {
                lblInfo = new JLabel();
                lblInfo.setText(getTitle());
                lblInfo.setPreferredSize(new Dimension(400, 30));
        }
        return lblInfo;
}

/**
 * Change le nombre maximum de tâches.
 *
 * @param maxTask
 *          nouvelle valeur.
 */
public void setMaxTask(int maxTask) {
        this.maxTask = maxTask;
}
```

```
        /**
         * Change le texte du label.
         * Cette méthode incrémente également la barre de progression.
         *
         * @param text
         *          nouveau texte.
         */
        public void setInfoLabel(String text) {
                getLblInfo().setText("<html>" + text + "</html>");
                currentTask++;
                int res = (currentTask * getLoadBar().getMaximum()) / maxTask;
                getLoadBar().setValue(res);
                repaint();
        }

        /**
         * Change le texte du label et ajoute une couleur rouge.
         * Cette méthode incrémente également la barre de progression.
         *
         * @param text
         *          nouveau texte.
         */
        public void setErrorLabel(String text) {
                setInfoLabel("<font color=\"red\">" + text + "</font>");
        }

        /**
         * Cette méthode initialise la barre de progression.
         *
         * @return javax.swing.JProgressBar.
         */
        private JProgressBar getLoadBar() {
                if (loadBar == null) {
                        loadBar = new JProgressBar();
                        loadBar.setMaximum(100);
                        loadBar.setMinimum(0);
                        loadBar.setValue(0);
                }
                return loadBar;
        }
}
```

5.2.2 LoadInit.java

```
package com.m2m.softphone.load;


import javax.swing.JOptionPane;


import mava.lang.MSystem;
import mava.lang.MSystem.Language;
import mava.softphone.CurrentConfig;
import mava.softphone.Loadable;


public class LoadInit implements Loadable{

	private Load			load		= null;
	private Language		language	= null;
	private static final String	MEDIA_NAME	= "media";

	/**
	 * Point d'entrée de l'interface.
	 * @param language
	 *		Langue sélectionné dans la configuration.
	 * @param defaultMaxTask
	 *		Nombre maximum de tâches.
	 * @see mava.lang.MSystem.Language#ENGLISH
	 * @see mava.lang.MSystem.Language#FRENCH
	 */
	public void init(Language language, int defaultMaxTask){
		this.language = language;
		if(language == Language.FRENCH)
			load = new Load("Chargement ...", defaultMaxTask);
		else
			load = new Load("Loading ...", defaultMaxTask);
	}

	/**
	 * Récupération des erreurs critiques (connexion avec le
	 * module média, licence, accès aux fichiers, etc...).
	 *
	 * @param err
	 *		Type de l'erreur
	 * @see mava.softphone.Loadable.FatalError#APP_ALREADY_LAUNCHED
	 * @see mava.softphone.Loadable.FatalError#AUDIO_DEVICE
	 * @see mava.softphone.Loadable.FatalError#GET_MEDIA_VERSION
	 * @see mava.softphone.Loadable.FatalError#INVALID_LIC
	 * @see mava.softphone.Loadable.FatalError#INVALID_STACK
	 * @see mava.softphone.Loadable.FatalError#INVALID_STACKCOMMUNICATOR
	 * @see mava.softphone.Loadable.FatalError#MEDIA_MODULE_NOT_FOUND
	 * @see mava.softphone.Loadable.FatalError#READ_LIC_FILE
	 * @see mava.softphone.Loadable.FatalError#STACK_NOT_FOUND
	 * @see mava.softphone.Loadable.FatalError#TCP_CONNECTION_FAILED
	 * @see mava.softphone.Loadable.FatalError#UDP_CONNECTION_FAILED
	 */
	public void fatalError(FatalError err){
		switch(err){
			/*
			 * Cette erreur est levée si l'application est déjà
			 * lancé.
			 */
			case APP_ALREADY_LAUNCHED :
				if(language == Language.ENGLISH)
					display("Application is already launched !");
				else
					display("L'application est d&eacute;j&agrave; lanc&eacute;e !");
				break;
```

```
/*
 * Cette erreur est levée si le moteur ne parvient pas à
 * lire le fichier licence.
 */
case READ_LIC_FILE :
        if(language == Language.ENGLISH)
                display("Unable to read the lic.txt file.");
        else
                display("Impossible de lire le fichier lic.txt.");

        break;



/* Cette erreur est levée si la licence n'est pas valide. */
case INVALID_LIC :
        if(language == Language.ENGLISH)
                display("License is invalid.<br>Check your parameters.");
        else
                display("Licence invalide.<br>" +
                        "V&eacute;rifier vos param&egrave;tres.");
        break;
/*
 * Cette erreur est levée si le module média n'est pas
 * trouvé.
 */
case MEDIA_MODULE_NOT_FOUND :
        if(MSystem.isLinuxOS()){
                if(language == Language.ENGLISH)
                        display("Impossible to find " + "lib/" +
                                        MEDIA_NAME + " !");
                else
                        display("Impossible de trouver " + "lib/" +
                                        MEDIA_NAME + " !");
        }else {
                if(language == Language.ENGLISH)
                        display("Impossible to find " + "lib\\" +
                                        MEDIA_NAME + ".exe !");
                else
                        display("Impossible de trouver " + "lib\\" +
                                        MEDIA_NAME + ".exe !");
        }
        break;
/* Cette erreur est levée si aucune stack n'est trouvée. */
case STACK_NOT_FOUND :
        if(language == Language.ENGLISH)
                display("Unable to continue without H323 or SIP stack !");
        else
                display("Impossible de continuer sans la stack H323 ou SIP !");
        break;
/*
 * Cette erreur est levée si le moteur ne parvient pas à
 * établir une connexion TCP avec le module média.
 */
case TCP_CONNECTION_FAILED :
        if(language == Language.ENGLISH)
                display("Media TCP connection failed on port " +
                CurrentConfig.getInternalTCP() + ".<br>Process halted !");
        else
                display("La connexion TCP avec le module Media sur le port " +
                CurrentConfig.getInternalTCP() +
                " a &eacute;chou&eacute;.<br>L'application va se terminer !");
        break;
```

```
                /*
                 * Cette erreur est levée si le moteur ne parvient pas à
                 * établir une connexion UDP avec le module média.
                 */
                case UDP_CONNECTION_FAILED :
                        if(language == Language.ENGLISH)
                                display("Media UDP connection failed on port " +
                                CurrentConfig.getInternalUDP() + ".<br>Process halted !");
                        else
                                display("La connexion UDP avec le module Media sur le port " +
                                CurrentConfig.getInternalUDP() +
                                " a &eacute;chou&eacute;.<br>L'application va se terminer !");
                        break;
                /*
                 * Cette erreur est levée si aucune des stacks trouvées
                 * n'est valide.
                 */
                case INVALID_STACK :
                        if(language == Language.ENGLISH)
                                display("Impossible to continue without a valid VOIP stack !");
                        else
                                display("Impossible de continuer sans une stack VOIP valide !");
                        break;
                /*
                 * Cette erreur est lev&ée si aucun plugin de type
                 * StackCommunicator n'est valide.
                 */
                case INVALID_STACKCOMMUNICATOR :
                        if(language == Language.ENGLISH)
                                display("Impossible to continue without a " +
                                        "'stack communicator' plugin !");
                        else
                                display("Impossible de continuer sans un plugin de type " +
                                        "'stack communicator' !");
                        break;
                /*
                 * Cette erreur est levée si la JVM ne parvient pas à
                 * prendre la mains sur les périphériques audio (micro
                 * et/ou haut-parleur).
                 */
                case AUDIO_DEVICE :
                        String message = "";
                        if(language == Language.FRENCH)
                                message = "Une des causes suivante &agrave; provoqu&eacute;" +
                                                " l'erreur : <br>" +
                                        "<ul><li>Le chemin des fichiers audio ne " +
                                                "sont pas correcte.</li>" +
                                        "<li>Votre p&eacute;riph&eacute;rique audio est" +
                                                " occup&eacute; par une autre application.</li>" +
                                        "<li>Un ou des fichier(s) audio requis par l'application" +
                                                " sont manquant.</li></ul>";
                        else
                                message = "One of the following problems has been detected :" +
                                        "<br><ul><li>The audio path files are not correct.</li>" +
                                        "<li>Your audio device is busy.</li>" +
                                        "<li>Audio files required by the application are" +
                                                " missing.</li></ul>";
                        display(message);
                        break;
                /*
                 * Cette erreur est levée si le moteur ne parvient pas à
                 * récupérer la version du module média.
                 */
                case GET_MEDIA_VERSION :
                        if(language == Language.ENGLISH)
                                display("Cannot hear media module.<br>System halted !");
                        else
                                display("Impossible de communiquer avec le module " +
                                        "m&eacute;dia !");
                        break;
                }
        }
```

```
/**
 * Notification sur le nombre maximum de tâches.
 *
 * @param maxTask
 *          Nombre maximum de tâches.
 */
public void setMaxTask(int maxTask){
        load.setMaxTask(maxTask);
}

/**
 * L'utilisateur peut afficher la fenêtre de chargement.
 *
 * @see #dispose()
 */
public void show(){
        load.setVisible(true);
}

/**
 * L'utilisateur peut masquer la fenêtre.
 *
 * @see #show()
 */
public void dispose(){
        load.dispose();
}

/**
 * Message sur l'état du chargement des plugins.
 *
 * @param pluginName
 *          Nom du plugins.
 * @param info
 *          Type de l'information.
 *
 * @see mava.softphone.plugins.Loadable.LoadingInfo#ERROR_NULL_CLASS
 * @see mava.softphone.plugins.Loadable.LoadingInfo#ERROR_NULL_OR_EMPTY_NAME
 * @see mava.softphone.plugins.Loadable.LoadingInfo#ERROR_NOT_SUBCLASS_OF
 * @see mava.softphone.plugins.Loadable.LoadingInfo#ERROR_CREATE_INSTANCE
 * @see mava.softphone.plugins.Loadable.LoadingInfo#ERROR_NULL_OBJECT
 * @see mava.softphone.plugins.Loadable.LoadingInfo#ERROR_INIT_METHOD
 * @see mava.softphone.plugins.Loadable.LoadingInfo#ERROR_START_METHOD
 * @see mava.softphone.plugins.Loadable.LoadingInfo#INFO_PLUGIN_LOAD
 * @see mava.softphone.plugins.Loadable.LoadingInfo#INFO_PREFERENCES_LOAD
 */
public void display(String pluginName, LoadingInfo info){
        switch(info){
                /*
                 * Cette erreur est levée si le moteur ne peut
                 * récupérer la classe représentant le point
                 * d'entrée du plugin dans le jar.
                 */
                case ERROR_NULL_CLASS :
                        if(language == Language.FRENCH)
                                load.setErrorLabel("Echec de chargement du plugin " +
                                        pluginName + " (Erreur : " + info + ") ...");
                        else
                                load.setErrorLabel("Loading failed of " + pluginName +
                                        " plugin (Error : " + info + ") ...");
                        break;
```

```
/*
 * Cette erreur est levée si le nom du plugin est
 * 'null' ou vide.
 */
case ERROR_NULL_OR_EMPTY_NAME :
        if(language == Language.FRENCH)
                load.setErrorLabel("Echec de chargement du plugin " +
                        pluginName + " (Erreur : " + info + ") ...");
        else
                load.setErrorLabel("Loading failed of " + pluginName +
                        " plugin (Error : " + info + ") ...");
        break;
/*
 * Cette erreur est levée si le point d'entrée du plugin
 * n'est pas une sous classe de 'Stack', 'StackCommunicator' ou de 'Application'.
 */
case ERROR_NOT_SUBCLASS_OF :
        if(language == Language.FRENCH)
                load.setErrorLabel("Echec de chargement du plugin " +
                        pluginName + " (Erreur : " + info + ") ...");
        else
                load.setErrorLabel("Loading failed of " + pluginName +
                        " plugin (Error : " + info + ") ...");
        break;
/*
 * Cette erreur est levée si le moteur ne peut créer une
 * instance du plugin.
 */
case ERROR_CREATE_INSTANCE :
        if(language == Language.FRENCH)
                load.setErrorLabel("Echec de chargement du plugin " +
                        pluginName + " (Erreur : " + info + ") ...");
        else
                load.setErrorLabel("Loading failed of " + pluginName +
                        " plugin (Error : " + info + ") ...");
        break;
/*
 * Cette erreur est levée si le cast 'Object' vers le type du plugin ( 'Stack',
 * 'StackCommunicator' ou 'Application') a échoué.
 */
case ERROR_NULL_OBJECT :
        if(language == Language.FRENCH)
                load.setErrorLabel("Echec de chargement du plugin " +
                        pluginName + " (Erreur : " + info + ") ...");
         else
                load.setErrorLabel("Loading failed of " + pluginName +
                        " plugin (Error : " + info + ") ...");
        break;
/*
 * Cette erreur est levée si le moteur intercepte une exception
 * dans la méthode 'init' du plugin.
 */
case ERROR_INIT_METHOD :
        if(language == Language.FRENCH)
                load.setErrorLabel("Echec de chargement du plugin " +
                        pluginName + " (Erreur : " + info + ") ...");
        else
                load.setErrorLabel("Loading failed of " + pluginName +
                        " plugin (Error : " + info + ") ...");
        break;
/*
 * Cette erreur est levée si le moteur intercepte une exception
 * dans la méthode 'start' du plugin.
 */
case ERROR_START_METHOD :
        if(language == Language.FRENCH)
                load.setErrorLabel("Echec de chargement du plugin " +
                        pluginName + " (Erreur : " + info + ") ...");
        else
                load.setErrorLabel("Loading failed of " + pluginName +
                        " plugin (Error : " + info + ") ...");
        break;
```

```
                        /*
                         * Ce message précise à l'utilisateur que le moteur
                         * commence à charger le plugin.
                         */
                        case INFO_PLUGIN_LOAD :
                                if(language == Language.FRENCH)
                                        load.setInfoLabel("Chargement plugin " + pluginName + " ...");
                                else
                                        load.setInfoLabel("Loading " + pluginName + " plugin ...");
                                break;

                        /*
                         * Ce message précise à l'utilisateur que le moteur
                         * commence à charger la fenêtre de préférences.
                         */
                        case INFO_PREFERENCES_LOAD :
                                if(language == Language.FRENCH)
                                        load.setInfoLabel("Chargement du module " +
                                                pluginName + " ...");
                                else
                                        load.setInfoLabel("Loading " + pluginName + " module ...");
                                break;

                }
        }

        /**
         * Affiche une boite de dialogue.
         *
         * @param text
         *          Texte à afficher.
         */
        private void display(String text){
                String title = "ERROR";
                if(language == Language.FRENCH)
                        title = "ERREUR";
                JOptionPane.showMessageDialog(null, "<html>" + text + "</htmL>", title,
                                JOptionPane.ERROR_MESSAGE);
        }

}
```

5.3 Manuel de référence des fonctions

5.3.1 mava.softphone.Loadable

NOM	SIGNATURE	DESCRIPTION
FatalError	**public enum FatalError**	Énumération contenant les erreurs de connexion avec le module média, de licence, d'accès aux fichiers, etc... **APP_ALREADY_LAUNCHED :** Cette erreur est levée si l'application est déjà lancé. **READ_LIC_FILE :** Cette erreur est levée si le moteur ne parvient pas à lire le fichier licence. **INVALID_LIC :** Cette erreur est levée si la licence n'est pas valide. **MEDIA_MODULE_NOT_FOUND :** Cette erreur est levée si le module média n'est pas trouvé. **STACK_NOT_FOUND :** Cette erreur est levée si aucune stack n'est trouvée. **TCP_CONNECTION_FAILED :** Cette erreur est levée si le moteur ne parvient pas à établir une connexion TCP avec le module média. **UDP_CONNECTION_FAILED :** Cette erreur est levée si le moteur ne parvient pas à établir une connexion UDP avec le module média. **INVALID_STACK :** Cette erreur est levée si aucune des stacks trouvées n'est valide. **INVALID_STACKCOMMUNICATOR :**

NOM	SIGNATURE	DESCRIPTION
		Cette erreur est levée si aucun plugin de type StackCommunicator n'est valide. **AUDIO_DEVICE :** Cette erreur est levée si la JVM ne parvient pas à prendre la mains sur les périphériques audio (micro et/ou haut-parleur). **GET_MEDIA_VERSION :** Cette erreur est levée si le moteur ne parvient pas à récupérer la version du module média.
LoadingInfo	**public enum LoadingInfo**	Informations sur le chargement. **ERROR_NULL_CLASS :** Cette erreur est levée si le moteur ne peut récupérer la classe représentant le point d'entrée du plugin dans le jar. **ERROR_NULL_OR_EMPTY_NAME :** Cette erreur est levée si le nom du plugin est **null** ou vide. **ERROR_NOT_SUBCLASS_OF :** Cette erreur est levée si le point d'entrée du plugin n'est pas une sous classe de ***Stack***, ***StackCommunicator*** ou de ***Application***. **ERROR_CREATE_INSTANCE :** Cette erreur est levée si le moteur ne peut créer une instance du plugin. **ERROR_NULL_OBJECT :** Cette erreur est levée si le cast ***Object*** vers le type du plugin (***Stack***, ***StackCommunicator*** ou ***Application***) a échoué. **ERROR_INIT_METHOD :** Cette erreur est levée si le

NOM	SIGNATURE	DESCRIPTION
		moteur intercepte une exception dans la méthode ***init*** du plugin. **ERROR_START_METHOD :** Cette erreur est levée si le moteur intercepte une exception dans la méthode ***start*** du plugin. **INFO_PLUGIN_LOAD :** Ce message précise à l'utilisateur que le moteur commence à charger le plugin. **INFO_PREFERENCES_LOAD;** Ce message précise à l'utilisateur que le moteur commence à charger la fenêtre de préférences.
init	**public void init(** **Language** language**,** **int** defaultMaxTask **)**	Point d'entrée de l'interface. **Paramètre :** language Langue sélectionné dans la configuration . **Paramètre :** defaultMaxTask Nombre maximum de tâches. **voir :** mava.lang.MSystem.Language#ENGLISH **voir :** mava.lang.MSystem.Language#FRENCH
fatalError	**public void fatalError(** **FatalError** err **)**	Récupération des erreurs critiques (connexion avec le module média, licence, accàs aux fichiers, etc...). **Paramètre :** err Type de l'erreur. **voir :** mava.softphone.Loadable.FatalError#APP_ALREADY_LAUNCHED **voir :** mava.softphone.Loadable.FatalError#AUDIO_DEVICE **voir :** mava.softphone.Loadable.Fatal

NOM	SIGNATURE	DESCRIPTION
		Error#GET_MEDIA_VERSION **voir :** mava.softphone.Loadable.FatalError#INVALID_LIC **voir :** mava.softphone.Loadable.FatalError#INVALID_STACK **voir :** mava.softphone.Loadable.FatalError#INVALID_STACKCOMMUNICATOR **voir :** mava.softphone.Loadable.FatalError#MEDIA_MODULE_NOT_FOUND **voir :** mava.softphone.Loadable.FatalError#READ_LIC_FILE **voir :** mava.softphone.Loadable.FatalError#STACK_NOT_FOUND **voir :** mava.softphone.Loadable.FatalError#TCP_CONNECTION_FAILED **voir :** mava.softphone.Loadable.FatalError#UDP_CONNECTION_FAILED
setMaxTask	**public void setMaxTask(** **int** maxTask **)**	Notification sur le nombre maximum de tâches. **Paramètre :** maxTask Nombre maximum de tâches.
show	**public void show()**	L'utilisateur peut afficher la fenêtre de chargement. **voir :** mava.softphone.Loadable.dispose()
dispose	**public void dispose()**	L'utilisateur peut masquer la fenêtre. **voir :** mava.softphone.Loadable.show()
display	**public void display(** **String** pluginName**,**	Message sur l'état du chargement des plugins.

NOM	SIGNATURE	DESCRIPTION
	LoadingInfo info **)**	**Paramètre :** pluginName Nom du plugins. **Paramètre :** info Type de l'information. **voir:** mava.softphone.Loadable.LoadingInfo#ERROR_NULL_CLASS **voir:** mava.softphone.Loadable.LoadingInfo#ERROR_NULL_OR_EMPTY_NAME **voir:** mava.softphone.Loadable.LoadingInfo#ERROR_NOT_SUBCLASS_OF **voir:** mava.softphone.Loadable.LoadingInfo#ERROR_CREATE_INSTANCE **voir:** mava.softphone.Loadable.LoadingInfo#ERROR_NULL_OBJECT **voir:** mava.softphone.Loadable.LoadingInfo#ERROR_INIT_METHOD **voir:** mava.softphone.Loadable.LoadingInfo#ERROR_START_METHOD **voir:** mava.softphone.Loadable.LoadingInfo#INFO_PLUGIN_LOAD **voir:** mava.softphone.Loadable.LoadingInfo#INFO_PREFERENCES_LOAD

6.1 Description

Pour que le softphone puisse identifier un plugin, celui-ci doit respecter 3 contraintes :

- Il doit hériter de l'objet mava.softphone.plugins.Application ou de l'objet mava.softphone.plugins.StackCommunicator.
- Il doit renseigner dans le fichier Manifest de son jar, le champs Main-Class et le faire pointer sur la classe héritant de l'objet mava.softphone.plugins.Application ou de l'objet mava.softphone.plugins.StackCommunicator.
- Placer son plugin dans le répertoire plugins du softphone.

6.2 Plugin de type mava.softphone.plugins.Application

Un plugin de type mava.softphone.plugins.Application permet de faire dialoguer plusieurs plugin de type mava.softphone.plugins.Application. Ce type de plugin peut être dupliqué sans limites mais ne peux en aucun cas envoyer des commandes à la stack VoIP.
Note : Pour l'envoi d'un message d'un plugin Application à un autre il est impératif de connaître le nom du plugin source et celui du plugin de destination.

6.3 Plugin de type mava.softphone.plugins.StackCommunicator

Un plugin de type mava.softphone.plugins.StackCommunicator permet de communiquer avec une stack VoIP et ainsi effectuer tous les traitements possible sur un appel (appel sortant, accepter un appel entrant, libérer un appel, mettre un appel en attente, etc ...). Ce type de plugin ne peut être dupliqué qu'une seule fois.
Note : un plugin de type mava.softphone.plugins.StackCommunicator et implicitement un plugin de type mava.softphone.plugins.Application.

6.4 Exemple de plugin de type StackCommunicator

Cet exemple contient 3 classes com.m2m.softphone.simple_call.Ihm (prend en charge le visuel), com.m2m.softphone.simple_call.IhmAction (prend en charge les actions de l'ihm) et com.m2m.softphone.simple_call.SimpleCall (hérite de la classe mava.softphone.plugins.StackCommunicator).

6.4.1 Ihm.java

```
package com.m2m.softphone.simple_call;

import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.JToggleButton;
import javax.swing.SwingConstants;

import mava.graphics.JarIcons;
import mava.graphics.GraphicsContants.Icons;
import mava.softphone.plugins.preference.PreferencesTreeObject;
import mava.swing.tree.TreeIconSet;

/**
 * <p>Title: Ihm.java</p>
 * <p>Description: Création de la fenêtre graphique</p>
 */
public class Ihm extends JFrame {
	private static final long		serialVersionUID	= -2028290646357069022L;
	private JLabel				lblInfo			= null;
	private JPanel				jContentPane		= null;
	private JTextField			txt			= null;
	private JButton				btCall			= null;
	private JButton				btRelease		= null;
	private JButton				btPreferences		= null;
	private JToggleButton			btWait			= null;
	private IhmAction			ihmAction		= null;

	/**
	 * Constructeur de l'objet
	 *
	 * @param simpleCall
	 *          Référence sur l'objet SimpleCall
	 */
	public Ihm(SimpleCall simpleCall) {
		super("Simple Call");
		// création du listener
		ihmAction = new IhmAction(this, simpleCall);
		// initialisation de l'ihm
		initialize();


		//céation d'une entrée dans la fenêtre de préférences.
		String simpleCallIcon = "SimpleCallIcon";//création de la famille de l'icône.
		//ici le '.' est important il permet de séparer la famille de l'icone du libellé du noeud.
		String simpleCallNodeName = simpleCallIcon + ".SimpleCall";
		PreferencesTreeObject tree = new PreferencesTreeObject(
				null, simpleCallNodeName, simpleCallIcon,
				//pour l'exemple nous utilisons une icône provenant du jar 'mava'
				//mais il est possible de mettre une icône externe.
				new TreeIconSet(JarIcons.getIcon(Icons.INFO)),
				ihmAction//listener pour la validation des préférences.
		);
		//ajout du visuel pour la configuration du plugin ici nous n'ajoutons qu'un label avec un texte
		JLabel lbl = new JLabel("Plugin SimpleCall");
		lbl.setHorizontalAlignment(SwingConstants.CENTER);
		tree.setContainer(lbl, "MyPanel");
		simpleCall.getPref().addTreeObject(tree);

	}
```

```
/**
 * Initialise l'ihm.
 */
private void initialize() {
	setDefaultCloseOperation(EXIT_ON_CLOSE);
	setContentPane(getJContentPane());
	pack();
	setLocationRelativeTo(null);
}

/**
 * Initialise le conteneur principal.
 *
 * @return javax.swing.JPanel
 */
private JPanel getJContentPane() {
	if (jContentPane == null) {
		jContentPane = new JPanel();
		jContentPane.setLayout(new GridBagLayout());
		lblInfo = new JLabel();
		lblInfo.setText("Call number :");

		GridBagConstraints gridLblInfo = new GridBagConstraints();
		gridLblInfo.gridx = 0;
		gridLblInfo.gridy = 0;
		gridLblInfo.gridwidth = 2;
		gridLblInfo.fill = GridBagConstraints.BOTH;
		gridLblInfo.insets = new Insets(10, 10, 10, 10);

		GridBagConstraints gridTxt = new GridBagConstraints();
		gridTxt.gridx = 2;
		gridTxt.gridy = 0;
		gridTxt.gridwidth = 2;
		gridTxt.fill = GridBagConstraints.BOTH;
		gridTxt.insets = new Insets(10, 0, 10, 10);

		GridBagConstraints gridBtPreferences = new GridBagConstraints();
		gridBtPreferences.gridx = 0;
		gridBtPreferences.gridy = 1;
		gridBtPreferences.insets = new Insets(0, 10, 10, 10);

		GridBagConstraints gridBtCall = new GridBagConstraints();
		gridBtCall.gridx = 1;
		gridBtCall.gridy = 1;
		gridBtCall.insets = new Insets(0, 0, 10, 10);

		GridBagConstraints gridBtWait = new GridBagConstraints();
		gridBtWait.gridx = 2;
		gridBtWait.gridy = 1;
		gridBtWait.insets = new Insets(0, 0, 10, 10);

		GridBagConstraints gridBtRelease = new GridBagConstraints();
		gridBtRelease.gridx = 3;
		gridBtRelease.gridy = 1;
		gridBtRelease.insets = new Insets(0, 0, 10, 10);

		jContentPane.add(lblInfo, gridLblInfo);
		jContentPane.add(getTxt(), gridTxt);
		jContentPane.add(getBtPreferences(), gridBtPreferences);
		jContentPane.add(getBtCall(), gridBtCall);
		jContentPane.add(getBtWait(), gridBtWait);
		jContentPane.add(getBtRelease(), gridBtRelease);
	}
	return jContentPane;
}
```

```
	/**
	 * Initialise le bouton de mise en attente.
	 *
	 * @return javax.swing.JToggleButton
	 */
	public JToggleButton getBtWait() {
		if (btWait == null) {
			btWait = new JToggleButton();
			btWait.setText("Wait");
			btWait.setEnabled(false);
			btWait.addActionListener(ihmAction);
		}
		return btWait;
	}

	/**
	 * Initialise le bouton d'envoi d'appel
	 *
	 * @return javax.swing.JButton
	 */
	public JButton getBtCall() {
		if (btCall == null) {
			btCall = new JButton();
			btCall.setText("Call");
			btCall.addActionListener(ihmAction);
		}
		return btCall;
	}

	/**
	 * Initialise le bouton de libération d'appel
	 *
	 * @return javax.swing.JButton
	 */
	public JButton getBtRelease() {
		if (btRelease == null) {
			btRelease = new JButton();
			btRelease.setText("Release");
			btRelease.addActionListener(ihmAction);
			btRelease.setEnabled(false);
		}
		return btRelease;
	}

	/**
	 * Initialise le bouton d'affichage des préférences
	 *
	 * @return javax.swing.JButton
	 */
	public JButton getBtPreferences() {
		if (btPreferences == null) {
			btPreferences = new JButton();
			btPreferences.setText("Prefs");
			btPreferences.addActionListener(ihmAction);
		}
		return btPreferences;
	}

	/**
	 * Initialise le champ texte pour la saisie du numéro à appeler.
	 *
	 * @return javax.swing.JTextField
	 */
	public JTextField getTxt() {
		if (txt == null) {
			txt = new JTextField();
		}
		return txt;
	}

}
```

6.4.2 IhmAction.java

```
package com.m2m.softphone.simple_call;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JOptionPane;

import mava.softphone.plugins.com.Plugin.PluginCall;
import mava.softphone.plugins.preference.Validable;

/**
 * <p>Title: IhmAction.java</p>
 * <p>Description: Action des boutons</p>
 */
public class IhmAction implements ActionListener, Validable {

	private Ihm		adaptee		= null;
	private SimpleCall	simpleCall	= null;

	/**
	 * Constructeur de l'objet.
	 *
	 * @param adaptee
	 *		Référence sur l'objet contenant les composants graphique.
	 * @param simpleCall
	 *		Référence sur l'objet SimpleCall contenant la référence sur le
	 *		fenêtre de préférences.
	 */
	public IhmAction(Ihm adaptee, SimpleCall simpleCall) {
		this.adaptee = adaptee;
		this.simpleCall = simpleCall;
	}

	/**
	 * Action sur un bouton.
	 */
	public void actionPerformed(ActionEvent e) {
		if (e.getSource().equals(adaptee.getBtPreferences()))
			// affiche la fenêtre de préférences.
			simpleCall.getPref().displayUI();
		else if (e.getSource().equals(adaptee.getBtRelease())) {
			// libère un appel (ici le champs name de la zone de texte
			// contient l'identifiant d'appel).
			PluginCall.release(adaptee.getTxt().getName());
			adaptee.getBtRelease().setEnabled(false);
			adaptee.getBtWait().setEnabled(false);
			adaptee.getBtWait().setSelected(false);
		} else if (e.getSource().equals(adaptee.getBtCall())) {
			if (adaptee.getTxt().getText().trim().compareTo("") != 0) {
				// demande l'envoi d'un appel.
				PluginCall.newCall(adaptee.getTxt().getText());
				//joue le fichier de tonalitée
				simpleCall.getCurrentConfig().getToneAudio()
					.setInterruptionTime(800);
				simpleCall.getCurrentConfig().getToneAudio().loop();
				adaptee.getTxt().setEnabled(false);
				adaptee.getBtCall().setEnabled(false);
			}
		} else if (e.getSource().equals(adaptee.getBtWait())) {
			// met un appel en attente (ici le champs name de la zone de texte
			// contient l'identifiant d'appel)
			PluginCall.wait(adaptee.getTxt().getName());
		}
	}
```

```
        /**
         * Méthode utilisée quand l'utilisateur appui sur les boutons
         * 'Valider' et/ou 'Appliquer' de la fenêtre de .préférences
         *
         * @return Retourne un message d'erreur ou une chaîne vide. Si la
         *         méthode retourne un message d'erreur alors la
         *         fenêtre de préférences ne sera pas fermée et affichera celui-ci pour que l'utilisateur puisse
         *         corriger son paramétrage du ou des plugins, sinon la fenêtre de
         *         préférences se fermera normalement.
         */
        public String apply() {
                JOptionPane.showMessageDialog(
                                adaptee,
                                "<html>L'utilisateur &agrave; valider la configuration !</html>",
                                "Information", JOptionPane.INFORMATION_MESSAGE);
                return null;
        }
}
```

6.4.3 SimpleCall.java

```
package com.m2m.softphone.simple_call;


import javax.swing.JOptionPane;

import mava.io.Log;
import mava.io.Log.Level;
import mava.softphone.CurrentConfig;
import mava.softphone.plugins.StackCommunicator;
import mava.softphone.plugins.com.Plugin.PluginCall;
import mava.softphone.plugins.com.application.AppMessage;
import mava.softphone.plugins.com.stack.StackEvent;
import mava.softphone.plugins.preference.Preferences;

/**
 * <p>Title: SimpleCall.java</p>
 * <p>Description: Point d'entrée du plugin</p>
 */
public class SimpleCall extends StackCommunicator {

        private Ihm                     ihm             = null;
        private Preferences             pref            = null;
        private CurrentConfig           config          = null;
        private static final int        BUSY_TIME_TO_SLEEP  = 2000;

        /**
         * Méthode appelée par le moteur de plugin lors de
         * l'arrêt de l'application. C'est à ce moment que l'objet peut
         * relâcher toutes les ressources qu'il utilise.
         */
        @Override
        public void destroy() throws Throwable {
                ihm = null;
                pref = null;
                config = null;
        }

        /**
         * Récupération du nom du plugin.<br>
         * Note : Ce nom doit être unique.
         *
         * @return Retourne le nom du plugin.
         */
        @Override
        public String getAppName() {
                return "M2M SimpleCall";
        }
```

```java
/**
 * Point d'entrée du plugin.
 *
 * @param pref
 *          Cet référence correspond à la fenêtre de gestion des préférences c'est aussi grâce à cet
 *          objet qu'un plugin peut se rajouter dans les configuration des préférences.
 * @param config
 *          Cet objet représente la configuration courante.
 *
 */
@Override
public void init(Preferences pref, CurrentConfig config) throws Throwable {
        this.pref = pref;
        this.config = config;
        ihm = new Ihm(this);
        //Change la référence du plugin de type StackCommunicator.
        //Cette méthode est obligatoire avant tout envois de messages vers une stack.
        PluginCall.setStackCommunicator(this);
}

/**
 * Méthode appelée lors de la validation du plugin par le moteur de plugin.
 * C'est à ce moment que l'objet peut commencer à s'initialiser (ex: affichage d'une fenêtre graphique).
 */
@Override
public void startApp() {
        ihm.setVisible(true);
        // exemple d'appel pour le changement du volume (micro et haut-parleur)
        config.setVolMic(9);
        config.setVolSpeaker(14);
        PluginCall.updateGain();
}

/**
 * Méthode écouteur pour les messages de type Stack et
 * StackCommunicator.
 *
 * @param e
 *          Le message.
 */
public void eventStackProcess(StackEvent e) {
        switch (e.getType()) {
                /*
                 * Echec de connexion ou de déconnexion avec un GK(H323) ou
                 * un PROXY(SIP)
                 */
                case GK_PROXY_CONNECTION_FAILED:
                        Log.display("PLUGIN MSG", Level.INFO, "IHM : " + e.getType());
                        break;
                /*
                 * Rejet d'un appel sortant
                 */
                case REJECT:
                        Log.display("PLUGIN MSG", Level.INFO, "IHM : " + e.getType());
                        // arrêt du fichier tonalité
                        config.getToneAudio().stop();
                        // joue le fichier occupé
                        config.getBusyAudio().loop();
                        try {// attente de 2 secondes
                                Thread.sleep(BUSY_TIME_TO_SLEEP);
                        } catch (InterruptedException e1) {
                        }
                        // arr&ecirc;t du fichier occupé
                        config.getBusyAudio().stop();
                        // change l'état des boutons
                        ihm.getTxt().setEnabled(true);
                        ihm.getBtCall().setEnabled(true);
                        ihm.getBtRelease().setEnabled(false);
                        ihm.getBtWait().setSelected(false);
                        ihm.getBtWait().setEnabled(false);
                        break;
```

```
/* Echec d'envoi d'un appel */
case FAILED:
        Log.display("PLUGIN MSG", Level.INFO, "IHM : " + e.getType());
        // arrêt du fichier tonalité
        config.getToneAudio().stop();
        // joue le fichier occupé
        config.getBusyAudio().loop();
        try {// attente de 2 secoundes
                Thread.sleep(BUSY_TIME_TO_SLEEP);
        } catch (InterruptedException e1) {
        }
        // arrêt du fichier occupé
        config.getBusyAudio().stop();
        // change l'état des boutons
        ihm.getTxt().setEnabled(true);
        ihm.getBtCall().setEnabled(true);
        ihm.getBtRelease().setEnabled(false);
        ihm.getBtWait().setSelected(false);
        ihm.getBtWait().setEnabled(false);
        break;
/*
 * Réception d'un nouvel appel.
 */
case CALL:
        Log.display("PLUGIN MSG", Level.INFO, "IHM : " + e.getType());
        // joue le fichier sonnerie
        config.getAlertingAudio().loop();
        int ret = JOptionPane.showConfirmDialog(ihm, "Call received : "
                        + e.getCallNumber(), "Call received",
                        JOptionPane.YES_NO_OPTION);
        if (ret == JOptionPane.YES_OPTION) {
                // accept l'appel
                PluginCall.accept(e.getCallId());
                // place le call id dans le champs name de la zone de texte
                ihm.getTxt().setName(e.getCallId());
        } else
                // refuse l'appel
                PluginCall.reject(e.getCallId());
        // change l'état des boutons
        ihm.getTxt().setText(e.getCallNumber());
        ihm.getTxt().setEnabled(false);
        ihm.getBtCall().setEnabled(false);
        break;
/* Appel établi */
case ESTABLISH:
        Log.display("PLUGIN MSG", Level.INFO, "IHM : " + e.getType());
        // change l'état des boutons
        ihm.getBtRelease().setEnabled(true);
        ihm.getBtWait().setEnabled(true);
        // stop les fichiers audio
        config.getToneAudio().stop();
        config.getBusyAudio().stop();
        config.getAlertingAudio().stop();
        break;
/*
 * Libération d'un appel
 */
case RELEASE:
        Log.display("PLUGIN MSG", Level.INFO, "IHM : " + e.getType());
        // change l'état des boutons
        ihm.getTxt().setEnabled(true);
        ihm.getBtCall().setEnabled(true);
        ihm.getBtRelease().setEnabled(false);
        ihm.getBtWait().setSelected(false);
        ihm.getBtWait().setEnabled(false);
        // stop les fichiers audio
        config.getToneAudio().stop();
        config.getBusyAudio().stop();
        config.getAlertingAudio().stop();
        break;
```

```
                        /*
                         * Récupération de l'identifiant d'appel suite à un nouvel appel sortant.
                         */
                        case GET_CALL_ID:
                                Log.display("PLUGIN MSG", Level.INFO, "IHM : " + e.getType());
                                // place le call id dans le champs name de la zone de texte
                                ihm.getTxt().setName(e.getCallId());
                                break;
                        /*
                         * Erreur de codec.
                         */
                        case GET_CALL_ID:
                                Log.display("PLUGIN MSG", Level.INFO, "IHM : " + e.getType());
                                // change l'état des boutons
                                ihm.getTxt().setEnabled(true);
                                ihm.getBtCall().setEnabled(true);
                                ihm.getBtRelease().setEnabled(false);
                                ihm.getBtWait().setSelected(false);
                                ihm.getBtWait().setEnabled(false);
                                // stop les fichiers audio
                                config.getToneAudio().stop();
                                config.getBusyAudio().stop();
                                config.getAlertingAudio().stop();
                                JOptionPane.showMessageDialog(null,
                                        "Codec Error", "ERROR",
                                        JOptionPane.ERROR_MESSAGE);
                                break;
                        }
                }

                /**
                 * Méthode écouteur pour les messages de type Application.
                 *
                 * @param e
                 *          Le message application.
                 */
                public void eventAppProcess(AppMessage e) {
                }

                /**
                 * Récupération de la fenêtre de préférences.
                 *
                 * @return Retourne la fenêtre de préférences.
                 */
                public Preferences getPref() {
                        return pref;
                }

                /**
                 * Récupération de la configuration courante .
                 *
                 * @return Retourne la configuration courante
                 */
                public CurrentConfig getCurrentConfig() {
                        return config;
                }
}
```

mava.softphone

CurrentConfig

Cette classe permet une gestion plus souple des différentes configurations du softphone.

NOM	SIGNATURE	DESCRIPTION
CallDisplay	**public enum CallDisplay**	Enumération représentant la chaîne à afficher lors des appels. **ALIAS_NUMBER :** Affichage du numéro de téléphone. **ALIAS_DISPLAY :** Affichage de la chaîne libre (H323 = H323-ID / SIP = Display). **NOTHING :** Ne rien afficher sur le poste du distant.
CurrentConfig	**public CurrentConfig(** **File** alertingFile**,** **File** busyFile**,** **File** toneFile **)**	Constructeur par défaut de l'objet. **Paramètre :** alertingFile Fichier audio pour les appels entrant. **Paramètre :** busyFile Fichier audio si le distant est occupé. **Paramètre :** toneFile Fichier audio pour la tonalité.
getDebugFileName	**public static final String** **getDebugFileName()**	Récupération du nom du fichier recevant les traces, si le nom est vide et que l'affichage des traces est activé alors elles apparaîtrons sur la console. ***Retour* :** Retourne le nom du fichier recevant les traces.
setDebugFileName	**public static final void** **setDebugFileName(** **String** fileName **)**	Change le nom du fichier recevant les traces, si le nom est vide et que l'affichage des traces est activé alors elles apparaîtrons sur la console. **Paramètre :** fileName Le nouveau nom.

NOM	SIGNATURE	DESCRIPTION
getLoaderJarName	**public static final String getLoaderJarName()**	Récupération du nom du fichier prenant en charge l'affichage du chargement des plugins. ***Retour*** **:** Retourne le nom du fichier prenant en charge l'affichage du chargement des plugins.
setLoaderJarName	**public static final void setLoaderJarName(** **String** fileName **)**	Change le nom du fichier prenant en charge l'affichage du chargement des plugins. **Paramètre :** fileName Le nouveau nom.
isFullDebug	**public static final boolean isFullDebug()**	Récupération du statut affichage des traces du moteur de plugin. ***Retour*** **:** Retourne **true** si l'affichage des traces du moteur de plugin sontactivées.
setFullDebug	**public static final void setFullDebug(** **boolean** debug **)**	Change le statut affichage des traces du moteur de plugin. **Paramètre :** debug Le nouveau statut.
getAppList	**public final static String[] getAppList()**	Récupération de la liste des plugins de type ***Application*** et ***StackCommunicator***. ***Retour*** **:** Retourne la liste des plugins de type ***Application*** et ***StackCommunicator***.
getStackList	**public final static String[] getStackList()**	Récupération de la liste des plugins de type ***Stack***. ***Retour*** **:** Retourne la liste des plugins de type ***Stack***.
getLocalIpAddress	**public final String getLocalIpAddress()**	Récupération de l'adresse IP local ('*' correspond à une auto détéction). ***Retour*** **:** Retourne l'adresse IP local.
getDescription	**public final static String getDescription()**	Récupération de la description correspondant à la licence. ***Retour*** **:** Retourne la description correspondant à la licence.

NOM	SIGNATURE	DESCRIPTION
isRTPEnabled	**public final static boolean isRTPEnabled()**	Récupération de la permission d'ouvrir le flux RTP. Cette permission est faite par la licence. ***Retour*** **:** Retourne **true** si l'application à le droit d'ouvrir le flux RTP.
getInternalTCP	**public final static int getInternalTCP()**	Récupération du port pour la connexion interne avec le média (TCP). ***Retour*** **:** Retourne le port pour la connexion interne avec le média (TCP).
getInternalUDP	**public final static int getInternalUDP()**	Récupération du port pour la connexion interne avec le média (UDP). ***Retour*** **:** Retourne le port pour la connexion interne avec le média (UDP).
getTimeoutCallFailed	**public final int getTimeoutCallFailed()**	Récupération de la durée d'attente avant notification d'échec d'un appel (en secondes). ***Retour*** **:** Retourne la durée d'attente avant notification d'échec d'un appel (en secondes) (5 <= timeout <= 30).
getWaitingFile	**public final String getWaitingFile()**	Récupération du nom du fichier pour la mise en attente lors d'une conversation audio. ***Retour*** **:** Retourne le nom du fichier pour la mise en attente lors d'une conversation audio.
isDoNotDisturb	**public final boolean isDoNotDisturb()**	Récupération du statut ne pas déranger. ***Retour*** **:** Retourne **true** si ne pas déranger est activé.
isAlwaysOnTop	**public final boolean isAlwaysOnTop()**	Récupération du statut fenêtre toujours au premiers plan. ***Retour*** **:** Retourne **true** si fenêtre toujours au premiers plan est activé.

NOM	SIGNATURE	DESCRIPTION
isDebug	**public final boolean isDebug()**	Récupération du statut affichage des traces. ***Retour*** **:** Retourne **true** si l'affichage des traces sont activées.
getAliasNumber	**public final String getAliasNumber()**	Récupération du numéro de téléphone du poste. ***Retour*** **:** Retourne le numéro de téléphone du poste.
getAliasDisplay	**public final String getAliasDisplay()**	Récupération du 'display'. Correspondances : • H323 = H323ID • SIP = DISPLAY • ***Retour*** **:** Retourne le 'display'.
getCallDisplay	**public final CallDisplay getCallDisplay()**	Récupération de la chaîne que doit afficher la stack (sip ou h323) lors d'un appel. ***Retour*** **:** Retourne le display. **voir :** mava.softphone.CurrentConfig.CallDisplay#ALIAS_NUMBER **voir :** mava.softphone.CurrentConfig.CallDisplay#ALIAS_DISPLAY **voir :** mava.softphone.CurrentConfig.CallDisplay#NOTHING
getCurrentRTPPort	**public final int getCurrentRTPPort()**	Récupération du port RTP local. ***Retour*** **:** Retourne le port RTP local.
getAlertingAudio	**public final AudioPlayer getAlertingAudio()**	Récupération du player pour le fichier audio pour les appels entrant. ***Retour*** **:** Retourne le player pour le fichier audio pour les appels entrant.
getBusyAudio	**public final AudioPlayer getBusyAudio()**	Récupération du player pour le fichier audio si le distant est occupé. ***Retour*** **:** Retourne le player le fichier audio si le distant est occupé.

NOM	SIGNATURE	DESCRIPTION
getToneAudio	**public final AudioPlayer getToneAudio()**	Récupération du player pour le fichier audio de tonalité. ***Retour*** **:** Retourne le player le fichier audio de tonalité.
getVolMic	**public final int getVolMic()**	Récupération de la valeur du micro (0 <= volMic <= 100). ***Retour*** **:** Retourne la valeur du micro.
setVolMic	**public final void setVolMic(** **int** volMic **)**	Change la valeur du micro (0 <= volMic <= 100). **Paramètre :** volMic La nouvelle valeur.
getVolSpeaker	**public final int getVolSpeaker()**	Récupération de la valeur du haut-parleur (0 <= volSpeaker <= 100). ***Retour*** **:** Retourne la valeur du haut-parleur.
setVolSpeaker	**public final void setVolSpeaker(** **int** volSpeaker **)**	Change la valeur du haut-parleur (0 <= volSpeaker <= 100). **Paramètre :** volSpeaker La nouvelle valeur.
isDialin	**public final static boolean isDialin()**	Récupération du sens d'un appel. ***Retour*** **:** Retourne **true** si il s'agit d'un appel entrant.
getMaxChannels	**public final static int getMaxChannels()**	Récupération du nombre maximum de canaux alloués par la licence. ***Retour*** **:** Retourne le nombre maximum de canaux alloués par la licence.
getCurrentStackName	**public abstract String getCurrentStackName()**	Récupération du nom de la stack active. ***Retour*** **:** Retourne le nom de la stack active.

Application

Cette classe abstraite représente le point d'entrée des plugins de type Application ce type de plugin permet la communication entre plusieurs plugins de même type.
Elle implémente également l'interface mava.softphone.plugins.com.application.AppListener.

NOM	SIGNATURE	DESCRIPTION
Application	**public Application()**	Constructeur par défaut de l'objet.
init	**public abstract void init(** **Preferences** pref**,** **CurrentConfig** config **) throws** Throwable	Point d'entrée du plugin. **Paramètre :** pref Cet correspond à la fenêtre de gestion des préférences c'est aussi grâce à cet objet qu'un plugin peut se rajouter dans les configuration des préférences. **Paramètre :** config Cet objet représente la configuration courante.
destroy	**public abstract void destroy()** **throws** Throwable	Méthode appelée par le moteur de plugin lors de l'arrêt de l'application. C'est à ce moment que l'objet peut relacher toutes les ressources qu'il utilise.
getAppName	**public abstract String** **getAppName()**	Récupération du nom du plugin. **Note :** Ce nom doit être unique. ***Retour*** **:** Retourne le nom du plugin.
startApp	**public abstract void** **startApp()** **throws** Throwable	Méthode appelée lors de la validation du plugin par le moteur de plugin. C'est à ce moment que l'objet peut commencer à s'initialiser (ex: affichage d'une fenêtre graphique).

StackCommunicator

Cette classe abstraite représente le point d'entrée du plugin de type StackCommunicator ce type de plugin permet la communication entre plusieurs plugins de type Application et également avec des stack VoIP.
Cette classe hérite de la classe mava.softphone.plugins.Application et implémente l'interface mava.softphone.plugins.com.stack.StackListener.

NOM	SIGNATURE	DESCRIPTION
StackCommunicator	**public StackCommunicator()**	Constructeur par défaut de l'objet.

mava.softphone.plugins.com.application

AppListener

Interface écouteur pour les messages de type Application.

NOM	SIGNATURE	DESCRIPTION
eventAppProcess	**public void eventAppProcess(** **AppMessage** e **)**	Méthode écouteur pour les messages de type Application. **Paramètre :** e Le message application.

AppMessage

Cette interface représente le message permettant la communication entre plugin de type application.

NOM	SIGNATURE	DESCRIPTION
getObject	**public Object getObject()**	Récupération d'un objet qu'un plugin de type Application à passé en paramètre. ***Retour* :** Retourne un objet qu'un plugin de type Application à passé en paramètre.
setObject	**public void setObject(** **Object** object **)**	Objet qu'un plugin de type Application peut passer en paramètre. **Paramètre :** object L'objet utilisateur.
getSourceName	**public String getSourceName()**	Récupération du nom du plugin source émetteur du message. ***Retour* :** Retourne le nom du plugin source émetteur du message.

mava.softphone.plugins.com.stack

StackListener

Écouteur pour les messages provenant d'un objet de type ***mava.softphone.plugins.Stack*** et ***mava.softphone.plugins.StackCommunicator***.

NOM	SIGNATURE	DESCRIPTION
eventStackProcess	**public void eventStackProcess(StackEvent e)**	Méthode écouteur pour les messages de type ***mava.softphone.plugins.Stack*** et ***mava.softphone.plugins.StackCommunicator***. **Paramètre :** e Le message.

StackEvent

Cette classe représente l'événement envoyé depuis un plugin de type ***mava.softphone.plugins.StackCommunicator*** ou depuis une stack VoIP.

NOM	SIGNATURE	DESCRIPTION
Type	**public enum Type**	Type de l'événement. **CALL :** Création ou réception d'un nouvel appel. **GET_CALL_ID :** Récupération de l'identifiant d'appel suite à un nouvel appel sortant. **REJECT :** Rejet d'un appel sortant ou entrant. **RELEASE :** Libération d'un appel. **ESTABLISH :** Appel établi. **FAILED :** Échec d'envoi d'un appel. **GK_PROXY_CONNECTION_FAILED :** Echec de connexion ou de déconnexion avec un GK(H323) ou un PROXY(SIP)

NOM	SIGNATURE	DESCRIPTION
		CODEC_ERROR : Erreur de codec.
getCallNumber	**public final String getCallNumber()**	Récupération du numéro d'appel. ***Retour :*** Retourne le numéro d'appel.
getAppName	**public final String getCallId()**	Récupération de l'identifiant d'appel. ***Retour :*** Retourne l'identifiant d'appel.
getType	**public final Type getType()**	Récupération du type du message. ***Retour :*** Retourne le type du message.

mava.softphone.plugins.com

Plugin

Cette classe statique permet les envois de messages vers des plugins de type ***mava.softphone.plugins.Application***.

NOM	SIGNATURE	DESCRIPTION
writeApplicationToApplication	**public static final boolean writeApplicationToApplication(String** sourceName**, String** destinationName**, AppMessage** event **)**	Écriture d'un message d'un plugin de type Application vers un plugin de type Application. **Paramètre :** sourceName Le nom du plugin émetteur du message. **Paramètre :** destinationName Le nom du plugin qui doit recevoir le message. **Paramètre :** event Le message à envoyer. ***Retour* :** Retourne **false** si une erreur s'est produite.

Plugin.PluginCall

Cette classe statique interne à la classe Plugin permet les envois de messages d'un plugin de type ***mava.softphone.plugins.StackCommunicator*** *vers une stack VoIP*.

NOM	SIGNATURE	DESCRIPTION
setStackCommunicator	**public static void setStackCommunicator(StackCommunicator** stackCommunicator **)**	Change la référence du plugin de type StackCommunicator pour toute la durée de l'application. Cette méthode est obligatoire avant tout envois de messages vers une stack. **Paramètre :** stackCommunicator Objet de type StackCommunicator.
newCall	**public static final boolean newCall(String** callNumber **)**	Création d'un nouvel appel. **Paramètre :** callNumber Le numéro à appeler. ***Retour* :** Retourne **false** si une erreur s'est produite.

NOM	SIGNATURE	DESCRIPTION
release	**public static final boolean** **release(** **String** callId **)**	Libère un appel. **Paramètre :** callId L'identifiant d'appel. ***Retour* :** Retourne **false** si une erreur s'est produite.
reject	**public static final boolean** **reject(** **String** callId **)**	Rejet d'un appel entrant. **Paramètre :** callId L'identifiant d'appel. ***Retour* :** Retourne **false** si une erreur s'est produite.
accept	**public static final boolean** **accept(** **String** callId **)**	Acceptation d'un appel entrant. **Paramètre :** callId L'identifiant d'appel. ***Retour* :** Retourne **false** si une erreur s'est produite.
wait	**public static final boolean** **wait(** **String** callId **)**	Mise en attente ou récupération d'un appel. **Paramètre :** callId L'identifiant d'appel. ***Retour* :** Retourne **false** si une erreur s'est produite.
updateGain	**public static final boolean** **updateGain()**	Change le volume du haut-parleur et/ou du micro pour un appel. ***Retour* :** Retourne **false** si une erreur s'est produite.
registration	**public static final boolean** **registration()**	*Demande un enregistrement ou désenregistrement aupràs d'un GK(H323) ou d'un PROXY(SIP).* ***Retour* :** Retourne **false** si une erreur s'est produite.

mava.softphone.plugins.preference

Preferences

Cette interface contient des méthodes permettant d'utiliser la fenêtre de préférences.

NOM	SIGNATURE	DESCRIPTION
disoseUI	**public void disoseUI()**	Ferme la fenêtre de préférences.
displayUI	**public void displayUI()**	Affiche la fenêtre de préférences.
addTreeObject	**public void addTreeObject(** **PreferencesTreeObject** object **)**	Ajoute une nouvelle configuration dans la fenêtre de préférences. **Paramètre :** object La configuration plugin à ajouter.
isVisible	**public boolean isVisible()**	Récupération de l'état de la fenêtre. ***Retour* :** Retourne **true** si la fenêtre est visible
getStackNode	**public TreeIconNode getStackNode()**	Récupération d'une référence sur le noeud 'stack' de la fenêtre. ***Retour* :** Retourne la référence sur le noeud 'stack' de la fenêtre.
getAudioNode	**public TreeIconNode getAudioNode()**	Récupération d'une référence sur le noeud 'audio' de la fenêtre. ***Retour* :** Retourne la référence sur le noeud 'audio' de la fenêtre.
getCodecNode	**public TreeIconNode getCodecNode()**	Récupération d'une référence sur le noeud 'codec' de la fenêtre. ***Retour* :** Retourne la référence sur le noeud 'codec' de la fenêtre.
getPluginNode	**public TreeIconNode getPluginNode()**	Récupération d'une référence sur le noeud 'plugin' de la fenêtre. ***Retour* :** Retourne la référence sur le noeud 'plugin' de la fenêtre.

NOM	SIGNATURE	DESCRIPTION
getMediaNode	**public TreeIconNode getMediaNode()**	Récupération d'une référence sur le noeud 'media' de la fenêtre. ***Retour :*** *Retourne la référence sur le noeud 'media' de la fenêtre.*
needRestart	**public void needRestart(** **boolean** restart **)**	Spécifie à l'utilisateur si il doit ou non redémarrer l'application après un changement dans la configuration d'un plugin. **Paramètre :** restart Si la valeur est **true** alors on affiche la boite de dialogue notifiant le besoin de redémarrer *l'application.*
isRestartNeeded	**public boolean isRestartNeeded()**	***Retour :*** *Retourne **true** si un plugin a déjà demandé de redémarrer l'application.*

Validable

Cette interface est utilisée pour être notifé quand l'utilisateur appui sur les boutons 'Valider' et/ou 'Appliquer' de la fenêtre de préférences.

NOM	SIGNATURE	DESCRIPTION
apply	**public String apply()**	Méthode utilisée quand l'utilisateur appui sur les boutons 'Valider' et/ou 'Appliquer' de la fenêtre de préférences. ***Retour :*** Retourne un message d'erreur ou une chaîne vide. Si la méthode retourne un message d'erreur alors la fenêtre de préférences ne sera pas fermée et affichera celui-ci pour que l'utilisateur puisse corriger son paramétrage du ou des plugins, sinon la fenêtre de préférences se fermera normalement.

PreferencesTreeObject

Cette interface permet l'ajout d'une nouvelle page de configuration dans la fenêtre de préférences.

NOM	SIGNATURE	DESCRIPTION
disoseUI	**public PreferencesTreeObject(** **List<TreeIconNode>** parent**,** **Object** child**,** **String** iconName**,** **TreeIconSet** set**,** **Validable** validable **)**	Constructeur par défaut de l'objet. **Paramètre :** parent Si le parent est null alors le noeud sera ajouté à la racine, sinon le noeud sera ajouté sous le *parent. Voir les méthodes* ***getAudioNode***, ***getCodecNode***, ***getMediaNode***, ***getPluginNode*** et **getStackNode** de l'interface ***Preferences***. **Paramètre :** child Le noeud à ajouter. **Paramètre :** iconName Le nom de l'icône, ce nom correspond à la famille de l'icône (Le nom ne doit pas contenir de '.'). **Paramètre :** set Cet objet permet la création d'une icône. *Les constructeurs possible de l'objet sont :* ***mava.swing.tree.TreeIconSet#TreeIconSet(Icon leafIcon)*** et ***mava.swing.tree.TreeIconSet#TreeIconSet(Icon openIcon, Icon closeIcon)*** **Paramètre :** validable Interface permettant d'être notifié quand l'utilisateur appui sur les boutons 'Valider' et/ou 'Appliquer'.
setContainer	**public void setContainer(** **Container** panel**,** **String** panelName **)**	Ajoute le conteneur pour la configuration du plugin. **Paramètre :** panel Le conteneur pour la configuration du plugin. **Paramètre :** panelName Le nom du conteneur.