

# **VISHNU - Le guide de l'administrateur**



## COLLABORATORS

	<i>TITLE :</i> VISHNU - Le guide de l'administrateur		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Benjamin Isnard, Daouda Traoré, Eugène Pamba Capo-Chichi, Kevin Coulomb, and Ibrahima Cissé	January 3, 2012	

## REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
1	08/03/2011	Version initiale pour le module UMS uniquement	K. COULOMB
2	18/03/2011	Ajout du lancement manuel avec forwarder et d'image de fichiers de configuration exemple	K. COULOMB
3	22/03/2011	Ajout des web services	K. COULOMB
4	11/05/2011	Réécriture du lancement avec fichier de configuration. Ajout d'un paragraphe pour le sendmail. Ajout de l'administration de TMS.	K. COULOMB, B.ISNARD
5	18/05/2011	Ajout du parametre de configuration dbConnectionsNb.	B.ISNARD
6	10/06/2011	Documentation pour IMS.	K.COULOMB
7	15/06/2011	Documentation pour FMS.	I.CISSE
8	22/06/2011	Ajout de l'option ENABLE_SWIG.	B.ISNARD
9	24/06/2011	Ajout de l'option vishnuMachineld dans les fichiers de configuration de UMS, IMS et FMS .	I.CISSE

**REVISION HISTORY**

NUMBER	DATE	DESCRIPTION	NAME
10	13/07/2011	Mise à jour du document suites aux premiers retours .	K.COULOMB
11	11/08/2011	Prise en compte du gestionnaire de ressources SLURM	D.TRAORE
12	23/08/2011	Ajout d'un lien sur des sites expliquant comment installer une base de données postgresql/mysql. Suppression des informations de mise à jour de la base (maintenant le script de création contient tout). Ajout d'une référence vers 'VISHNU_API'	K.COULOMB
13	14/12/2011	Mise à jour pour les nouveaux forwarder de DIET.	K.COULOMB
14	15/12/2011	Mise à jour de la section configuration des clés ssh requises pour FMS.	I.CISSE
15	16/12/2011	Ajout de la section configuration des clés ssh requises pour TMS.	D.TRAORE

# Contents

<b>1</b>	<b>Présentation du document</b>	<b>1</b>
1.1	Objectifs du document	1
1.2	Structure du document	1
<b>2</b>	<b>Définitions</b>	<b>2</b>
2.1	Acronymes	2
2.2	Références	2
2.3	Glossaire	2
<b>3</b>	<b>Installation à partir des sources</b>	<b>4</b>
3.1	Objectifs	4
3.2	Prérequis	4
3.3	Compilation des sources	5
<b>4</b>	<b>Configuration de la base de données</b>	<b>6</b>
4.1	Utilisation d'une base de données PostgreSQL	6
4.2	Utilisation d'une base de données MySQL	6
<b>5</b>	<b>Installation des web services</b>	<b>7</b>
5.1	Pré-requis	7
5.2	Installation de JBoss	7
5.3	Installation des module WS dans JBoss	8
5.3.1	Fichiers à installer	8
5.3.2	Variables d'environnement à définir	8
5.3.3	Lancement du serveur JBoss avec le module WS	8
<b>6</b>	<b>Déploiement de VISHNU</b>	<b>9</b>
6.1	Lancement sur un même réseau	9
6.2	Lancement sur plusieurs réseaux <b>avec une version de DIET supérieure ou égale à 2.8</b>	11
6.3	Lancement sur plusieurs réseaux <b>avec une version de DIET inférieure à 2.8</b>	12
6.4	Exemple de fichier de configuration d'un MA	14
6.5	Exemple de fichier de configuration pour un SeD DIET	14

6.6	Exemple de fichier de configuration d'un SeD UMS . . . . .	14
6.7	Exemple de fichier de configuration d'un SeD TMS . . . . .	14
6.8	Exemple de fichier de configuration d'un SeD FMS . . . . .	14
6.9	Exemple de fichier d'un forwarder . . . . .	15
6.10	Exemple de fichier du LogCentral . . . . .	15
6.11	Configuration de l'envoi des emails par VISHNU . . . . .	15
6.12	Configuration des clés privées/publiques ssh requises pour FMS . . . . .	16
6.13	Configuration des clés privées/publiques ssh requises pour TMS . . . . .	16
6.14	Test d'exécution d'un service depuis une machine client par shell . . . . .	16
<b>7</b>	<b>Administration</b>	<b>17</b>
7.1	Présentation . . . . .	17
7.2	Gestion des utilisateurs (UMS) . . . . .	17
7.3	Gestion des machines (UMS+IMS) . . . . .	17
7.4	Gestion de la plateforme (UMS) . . . . .	18
7.5	Options propres à l'administrateur dans les commandes utilisateurs(UMS+FMS) . . . . .	18
7.6	Gestion des processus VISHNU et délestage (IMS) . . . . .	19
7.7	Surveillance de l'état des machines (IMS) . . . . .	19
7.8	Définition des formats des identifiants (IMS) . . . . .	19
<b>8</b>	<b>UMS Command reference</b>	<b>20</b>
8.1	vishnu_add_user . . . . .	20
8.2	vishnu_update_user . . . . .	21
8.3	vishnu_delete_user . . . . .	22
8.4	vishnu_reset_password . . . . .	23
8.5	vishnu_save_configuration . . . . .	24
8.6	vishnu_restore_configuration . . . . .	25
8.7	vishnu_add_machine . . . . .	26
8.8	vishnu_update_machine . . . . .	28
8.9	vishnu_delete_machine . . . . .	29
8.10	vishnu_list_users . . . . .	30
8.11	vishnu_configure_default_option . . . . .	31
<b>9</b>	<b>UMS C++ API Reference</b>	<b>33</b>
9.1	addUser . . . . .	33
9.2	updateUser . . . . .	34
9.3	deleteUser . . . . .	35
9.4	resetPassword . . . . .	35
9.5	saveConfiguration . . . . .	36
9.6	restoreConfiguration . . . . .	37

9.7	addMachine . . . . .	38
9.8	updateMachine . . . . .	38
9.9	deleteMachine . . . . .	39
9.10	listUsers . . . . .	40
9.11	configureDefaultOption . . . . .	41
<b>10</b>	<b>UMS Python API Reference</b>	<b>42</b>
10.1	VISHNU.addUser . . . . .	42
10.2	VISHNU.updateUser . . . . .	43
10.3	VISHNU.deleteUser . . . . .	44
10.4	VISHNU.resetPassword . . . . .	45
10.5	VISHNU.saveConfiguration . . . . .	46
10.6	VISHNU.restoreConfiguration . . . . .	46
10.7	VISHNU.addMachine . . . . .	47
10.8	VISHNU.updateMachine . . . . .	48
10.9	VISHNU.deleteMachine . . . . .	49
10.10	VISHNU.listUsers . . . . .	50
10.11	VISHNU.configureDefaultOption . . . . .	51
<b>11</b>	<b>IMS Command reference</b>	<b>53</b>
11.1	vishnu_get_processes . . . . .	53
11.2	vishnu_set_system_info . . . . .	54
11.3	vishnu_set_system_threshold . . . . .	55
11.4	vishnu_get_system_threshold . . . . .	55
11.5	vishnu_define_user_identifier . . . . .	56
11.6	vishnu_define_machine_identifier . . . . .	57
11.7	vishnu_define_job_identifier . . . . .	58
11.8	vishnu_define_transfer_identifier . . . . .	59
11.9	vishnu_load_shed . . . . .	60
11.10	vishnu_set_update_frequency . . . . .	61
11.11	vishnu_stop . . . . .	62
11.12	vishnu_restart . . . . .	63
<b>12</b>	<b>IMS C++ API Reference</b>	<b>65</b>
12.1	getProcesses . . . . .	65
12.2	setSystemInfo . . . . .	65
12.3	setSystemThreshold . . . . .	66
12.4	getSystemThreshold . . . . .	67
12.5	defineUserIdentifier . . . . .	67
12.6	defineMachineIdentifier . . . . .	68

12.7	defineJobIdentifier	68
12.8	defineTransferIdentifier	69
12.9	loadShed	70
12.10	setUpdateFrequency	70
12.11	stop	71
12.12	restart	71

### 13 IMS Python API Reference

13.1	VISHNU.getProcesses	73
13.2	VISHNU.setSystemInfo	74
13.3	VISHNU.setSystemThreshold	74
13.4	VISHNU.getSystemThreshold	75
13.5	VISHNU.defineUserIdentifier	76
13.6	VISHNU.defineMachineIdentifier	76
13.7	VISHNU.defineJobIdentifier	77
13.8	VISHNU.defineTransferIdentifier	78
13.9	VISHNU.loadShed	79
13.10	VISHNU.setUpdateFrequency	79
13.11	VISHNU.stop	80
13.12	VISHNU.restart	81

# Chapter 1

## Présentation du document

### 1.1 Objectifs du document

Ce document présente l'administration de la plateforme VISHNU.

### 1.2 Structure du document

Ce document contient les parties suivantes:

- Définitions
  - Installation
  - Installation des web services
  - Déploiement
  - Administration
  - Référence des commandes (en anglais)
  - Référence de l'API C++ (en anglais)
  - Référence de l'API Python (en anglais)
-



## Chapter 2

# Définitions

### 2.1 Acronymes


- DB : Base de données: elle est centralisée et utilisée pour sauvegarder toutes les données manipulées par les divers modules.
- FMS : File Management Service
- IMS : Information Management Service
- MA : "Master Agent": élément de Sysfera-DS distribuant les requêtes entre les clients et les services désirés.
- SQL : Langage de requêtes sur les bases de données
- TMS : Task Management Service
- UMS : User Management Service
- WS : Web Services

### 2.2 Références

- [ARCH] D1.1g-VISHNU Technical Architecture : description de l'architecture de l'application VISHNU
- [DIET\_USERGUIDE] DIET User Guide : guide de l'utilisateur DIET (nom du projet OpenSource de l'application SysFera-DS)
- [VISHNU\_USERMANUAL] VISHNU User Manual : guide de l'utilisateur VISHNU.
- [VISHNU\_API] VISHNU API : API VISHNU contenant les signatures et la définition des objets.

### 2.3 Glossaire

- Client FMS : Cela désigne les programmes permettant un accès à distance aux services du SeD FMS.
- Client IMS : Cela désigne les programmes permettant un accès à distance aux services du SeD IMS.
- Client TMS : Cela désigne les programmes permettant un accès à distance aux services du SeD TMS.
- Client UMS : Cela désigne les programmes permettant un accès à distance aux services du SeD UMS.
- Préfrontale : Cela désigne une machine mise avant la(les) frontale(s) des calculateurs.

- SeD IMS : Cela désigne le programme contenant et exécutant les services du module IMS.
  - SeD TMS : Cela désigne le programme contenant et exécutant les services du module TMS.
  - SeD UMS : Cela désigne le programme contenant et exécutant les services du module UMS.
  - Sysfera-DS : Intergiciel open source développé par SysFera.
- 

## Chapter 3

# Installation à partir des sources

### 3.1 Objectifs

Ce chapitre va présenter comment installer VISHNU à partir des sources. Il expliquera comment installer les serveurs et/ou les clients (l'installation d'un serveur implique l'installation du client associé). Le module UMS est nécessaire pour tout les autres modules, sinon n'importe quel module peut être installé sur une machine donnée sans dépendance des autres, et cela n'empêche pas d'utiliser les services même si les serveurs sont sur des machines différentes. Toutefois, pour que IMS puisse faire du délestage sur une machine, il faut que les modules concernés (TMS et/ou FMS) par le délestage soient installés avec IMS sur la machine en question.

### 3.2 Prérequis

Les logiciels suivants doivent être installés sur le système, avec les fichiers de développement (paquets -dev):

- Dépendances requises pour tous les modules VISHNU:
  - GCC version 4.4.3 minimum
  - CMAKE version 2.6 minimum
  - BOOST version 1.45 minimum
  - OMNIORB version 4.1.4
  - libcrypt
  - SSH
  - Sysfera-DS (DIET version 2.7 minimum + le module log service associé)
- Dépendances requises pour le module TMS:
  - Torque API v3.2.6
  - IBM LoadLeveler API
  - ou SLURM API v2.2.1
- Dépendances optionnelles:
  - Base de données, à mettre sur les machines avec des serveurs, au choix :
    - \* PGSQL-API (api PostGreSQL) version 8.0 minimum
    - \* MYSQL-API (api MySQL) version 5.1 minimum

### 3.3 Compilation des sources

VISHNU utilise CMake comme système de construction et se conforme aux pratiques communément admises. Les options principales utilisables sont les suivantes:

- **BUILD\_TESTING**, compile les tests si le flag est activé (OFF par défaut).
- **CLIENT\_ONLY**, qui permet de ne compiler que les clients si le flag est activé (OFF par défaut). Sinon les clients et les serveurs sont compilés.
- **CMAKE\_INSTALL\_PREFIX**, répertoire d'installation (/usr/local par défaut sur les plateformes \*nix)
- **COMPILE\_TMS**, compile le module TMS si le flag est activé (OFF par défaut). Si ON, **COMPILE\_UMS** doit aussi être à ON et l'une des options **VISHNU\_USE\_TORQUE**, **VISHNU\_USE\_LOADLEVELER** ou **VISHNU\_USE\_SLURM** doit être à ON.
- **COMPILE\_IMS**, compile le module IMS si le flag est activé (OFF par défaut). Si ON, **COMPILE\_UMS**, **COMPILE\_TMS** et **COMPILE\_FMS** doivent aussi être à ON.
- **COMPILE\_FMS**, compile le module FMS si le flag est activé (OFF par défaut). Si ON, **COMPILE\_UMS** doit aussi être à ON.
- **COMPILE\_UMS**, compile le module UMS si le flag est activé (OFF par défaut).
- **ENABLE\_PYTHON**, qui permet d'activer la compilation du code PYTHON (OFF par défaut).
- **ENABLE\_JAVA**, qui permet d'activer la compilation des sources JAVA pour les web services (OFF par défaut).
- **ENABLE\_SWIG**, qui permet d'activer la generation du code des adapteurs Python et Java (OFF par défaut). Cette option doit obligatoirement être activée si on choisit de ne pas compiler tous les modules VISHNU c'est-à-dire si au moins l'une des options **COMPILE\_UMS**, **COMPILE\_TMS**, **COMPILE\_FMS** ou **COMPILE\_IMS** est à OFF. Si l'option **ENABLE\_SWIG** est activée, le package SWIG (voir paragraphe 'Prérequis') est nécessaire pour compiler.
- **VISHNU\_USE\_LOADLEVELER**, compile le SeD TMS pour load leveler (OFF par défaut).
- **VISHNU\_USE\_TORQUE**, compile le SeD TMS pour torque (OFF par défaut).
- **VISHNU\_USE\_SLURM**, compile le SeD TMS pour SLURM (OFF par défaut).

Par exemple, pour compiler VISHNU (clients et serveurs) avec l'API Python puis l'installer dans le répertoire /opt/vishnu sous \*nix:

- 1. créer un répertoire build à la racine du projet et se placer dedans
  - **\$ mkdir build**
  - **\$ cd build**
- 2. générer le Makefile
  - **\$ cmake -DENABLE\_PYTHON=ON -DENABLE\_SWIG=ON -DCMAKE\_INSTALL\_PREFIX=/opt/vishnu/ -DCOMPILE\_UMS=ON -DCOMPILE\_TMS=ON -DVISHNU\_USE\_TORQUE=ON -DTORQUE\_DIR=/opt/torque ..**
- 3. lancer la compilation avec 2 jobs
  - **\$ make -j 2**
- 4. installation (en mode super-utilisateur)
  - **# make install**

**Note:** pensez à ajouter le répertoire d'installation dans le \$PATH utilisateur pour avoir accès à la commande.

## Chapter 4

# Configuration de la base de données

Les fichiers de configuration de la DB sont disponibles dans le répertoire `core/database` du package d'installation VISHNU. Les seules bases de données actuellement supportées sont PostgreSQL et MySQL. Les bases de type Oracle ne sont pas supportées. Lors de l'installation toutes les machines doivent avoir soit PostgreSQL, soit MySQL, mais pas certaines machines une base et d'autres une autre, cela ne fonctionnerait pas.

### 4.1 Utilisation d'une base de données PostgreSQL

Un guide pour installer une base PostgreSQL peut être trouvé à cette adresse : <http://beuss.developpez.com/tutoriels/postgres/installation/>

Pour une installation d'une nouvelle base de données, les scripts `'postgre_create.sql'` et `'database_init.sql'` doivent être utilisés par l'administrateur de la DB.

Pour la mise à jour de la base de données lors d'un passage de la release VISHNU précédente à la nouvelle release VISHNU, les scripts `'postgre_update.sql'` et `'database_update.sql'` doivent être utilisés par l'administrateur de la DB. Le détail des numéros de release est indiqué dans les commentaires au début du script.

### 4.2 Utilisation d'une base de données MySQL

Un guide pour installer une base MySQL peut être trouvé à cette adresse : <http://dev.mysql.com/doc/refman/5.0/fr/unix-post-installation.html>

Pour une installation d'une nouvelle base de données, les scripts `'mysql_create.sql'` et `'database_init.sql'` doivent être utilisés par l'administrateur de la DB.

## Chapter 5

# Installation des web services

### 5.1 Pré-requis

- Installer **Java 1.6** (commande 'sudo apt-get install openjdk-6-jdk' ou bien 'sudo apt-get install sun-java6-jdk' sur Linux Debian) et vérifier que la variable JAVA\_HOME est bien définie et contient le répertoire racine de l'installation de java.
- Installer les modules désirés avec les options -DENABLE\_JAVA=ON et -DENABLE\_SWIG (**Note** : Ce pré-requis est optionnel car les jars sont déjà fournis dans la distribution des web services).
- Installer **Maven 2** pour compiler les jars (**Note** : Ce pré-requis est optionnel car les jars sont déjà fournis dans la distribution des web services. Par ailleurs la compilation avec maven nécessite une connexion internet).

### 5.2 Installation de JBoss

- Télécharger le package JBOSSAS : (la version binaire est disponible) sur <http://www.jboss.org/jbossas/downloads> => version 5.1.0.GA
- Télécharger le package JBOSSWS : (la version binaire est disponible) sur <http://www.jboss.org/jbossas/downloads/> => version 3.3.1.GA
- Décompresser l'archive du package JBOSSAS 5.1.0
- Définir la variable d'environnement JBOSS\_HOME sur le répertoire décompressé. Par exemple dans le .bashrc : 'export JBOSS\_HOME=/home/toto/jboss-5.1.0.GA'
- Décompresser l'archive du package JBOSSWS 3.3.1 et aller dans le répertoire créé
- Copier le fichier 'ant.properties.example' en 'ant.properties'
- Editer le fichier nouvellement créé 'ant.properties'. Mettre la valeur de la variable jboss510.home à la valeur du répertoire de JBOSSAS (même valeur que dans la variable JBOSS\_HOME). Il faut noter que si une autre version que la 5.1.0 de jboss a été prise, il faut modifier la variable correspondant à cette version
- Lancer la commande 'ant deploy-jboss510'. Si une autre version de jboss a été prise, il faut faire la commande de la version correspondante
- Le lancement du serveur se fait en lançant le script JBOSS\_HOME/bin/run.sh. Pour que le serveur jboss soit accessible depuis une autre machine, utiliser l'option '-b adresseIP' où adresseIP représente l'adresse IP du serveur, et vérifier que le firewall du serveur autorise l'accès au port 8080.
- **Vérification de l'installation:** Pour vérifier que le serveur JBoss est bien démarré et que le module web services est activé, lancer un navigateur internet sur le serveur et se connecter sur l'adresse 'http://localhost:8080/jbossws' (ou 'http://adresseIP:8080/jbossws') et vérifier que la page affiche bien la version du module web services (jbossws-cxf-3.3.1.GA).

## 5.3 Installation des module WS dans JBoss

### 5.3.1 Fichiers à installer

- VISHNULib-1.0-SNAPSHOT.jar
  - Contient les classes internes faisant le lien JAVA(JNI)/C++.
  - A copier dans **JBOSS\_HOME/server/default/lib**.
  - Le changer implique un redémarrage du serveur JBOSS.
  - Compilation (si nécessaire)
    - \* Aller dans le répertoire VISHNULib
    - \* Faire 'mvn install' (peut être long la première fois)
    - \* Le fichier .jar produit se trouve dans le répertoire target/
- WSAPI.jar
  - Contient les classes issues du WSDL et l'implémentation des WS.
  - A mettre dans **JBOSS\_HOME/server/default/deploy**.
  - Peut être mis à jour sans redémarrer le serveur JBOSS.
  - Compilation (si nécessaire)
    - \* Aller dans le répertoire WSAPI, avoir le jar VISHNULib-1.0-SNAPSHOT.jar de déjà fait
    - \* Faire 'mvn install' (peut être long la première fois)
    - \* Le fichier .jar produit se trouve dans le répertoire target/
    - \* Renommer le jar en WSAPI.jar avant de le mettre dans jboss, ceci est nécessaire
- libVISHNU.so
  - Nécessaire pour le fonctionnement des WS, cette librairie dynamique est obtenue en compilant les modules VISHNU désirés avec l'option `ENABLE_JAVA`.
  - Le fichier est installé par défaut par le package VISHNU UMS dans '/usr/local/lib' et il n'est pas nécessaire de le copier.
  - S'il y a un problème de déploiement dans le serveur jboss, vérifier qu'elle est bien accessible et dans le `LD_LIBRARY_PATH`.

### 5.3.2 Variables d'environnement à définir

- `VISHNU_CONFIG_FILE` : contient le chemin complet du fichier de configuration client de VISHNU. Se référer au guide d'installation du client [VISHNU\_USER\_GUIDE] pour connaître le contenu de ce fichier. Si l'exécution échoue avec un message d'erreur lié à initialisation de la librairie lors du connect, vérifier le contenu de cette variable.
- `LD_LIBRARY_PATH` : contient les chemins des répertoires contenant les librairies VISHNU, notamment `libVISHNU_UMS.so`.

### 5.3.3 Lancement du serveur JBoss avec le module WS

- Après installation des fichiers définis au paragraphe précédent, le serveur JBoss doit être redémarré en lançant le script `JBOSS_HOME/bin/run.sh` (avec les options indiquées au paragraphe sur le serveur JBoss).
- **Vérification de l'installation:** Pour vérifier que le serveur JBoss est bien démarré et que le module UMS WS est activé, lancer un navigateur internet sur le serveur et se connecter sur l'adresse '`http://localhost:8080/jboss/ws/services`' (ou '`http://adresseIP:8080/jboss/ws/services`') et vérifier que le "service endPoint" : **VISHNUMSPortImpl** est actif.

## Chapter 6

# Déploiement de VISHNU

### 6.1 Lancement sur un même réseau

Les modules peuvent être lancés manuellement. Le déploiement se fera en suivant la configuration montrée sur la figure 4.1 du document [ARCH]. Pour simplifier les choses, on supposera que les divers éléments dans le carré "dedicated to VISHNU" vont être déployés sur la même machine que l'on va nommer **préfrontale**.

**NOTE: Il est préférable d'éviter de lancer le même SeD sur la même machine.**

**NOTE: Il existe un bug connu sur debian (entre autre) avec boost file system, utilisé par VISHNU. Le rapport de bug est ici et le bug est actuellement ouvert: <https://svn.boost.org/trac/boost/ticket/4688>. Si lors du lancement d'un SeD, le message d'erreur suivant apparaît : `std::runtime_error: locale::facet::_S_create_c_locale name not valid`, faire un `"export LANG=C"` et cela devrait régler le problème.**

**NOTE: L'utilisateur VISHNU 'root' doit avoir un compte local sur chaque machine ou il y a un serveur IMS. Le compte UNIX correspondant sera utilisé pour le redémarrage automatique des SeD.**

Pour mieux comprendre l'architecture de déploiement, se référer au document [ARCH], le chapitre 4, 'Technical Architecture'.

1. Avoir une base de données PostgreSQL accessible et initialisée (tables créées et premières données créées). Des scripts SQL sont fournis pour cela. Avoir la base PostgreSQL configurée pour qu'elle soit accessible par VISHNU (voir le fichier de configuration `pg_hba.conf` qui se situe dans un répertoire de la forme `'/etc/postgresql/8.3/main'` pour configurer la base).
2. Lancer le service de nommage CORBA sur la machine préfrontale. La commande est **omniNames -start** pour la première fois, sinon **omniNames** suffit. Attention, dans la configuration de l'omninames, bien utiliser l'adresse de l'hôte et non pas `'localhost'` ou `'127.0.0.1'`.
3. Lancer le log service de DIET sur la machine préfrontale. La commande est **LogCentral --config config.cfg** avec `config.cfg` un fichier de configuration pour le log central.
4. Lancer le MA avec son fichier de configuration sur la machine préfrontale : **dietAgent config.cfg**. Voir le paragraphe 6.3 pour avoir un exemple de configuration. Ce fichier de configuration peut contenir les 3 lignes suivantes :
  - `'traceLevel = 0'` : Le niveau de verbosité du master agent, cette valeur peut être entre 0 et 10 compris.
  - `'agentType = DIET_MASTER_AGENT'` : Le type de l'agent, l'autre type disponible est `DIET_LOCAL_AGENT` mais dans notre cas il faut `DIET_MASTER_AGENT`.
  - `'name = MA0'` : Le nom que l'on veut donner à l'agent, par exemple on le nomme MA0 ici.
5. Lancer le SeD UMS sur la préfrontale avec la commande **umssed ~/ums\_sed.cfg**. Le paramètre est un fichier de configuration VISHNU. Voir le paragraphe 6.5 pour un exemple de fichier de configuration. Les paramètres à fournir correspondent à :
  - `'dietConfigFile=/usr/local/etc/SeD.cfg'` : le chemin jusqu'au fichier de configuration DIET du SeD. Voir le paragraphe 6.4 pour un exemple de configuration. Ce fichier peut par exemple contenir les 2 lignes suivantes :



- 'traceLevel = 0' : Le niveau de verbosité du SeD UMS, ce niveau peut être entre 0 (minimum) et 10 compris.
  - 'parentName = MA0' : Le nom du MA auquel le SeD UMS doit se lier. Ce doit être exactement le même nom que celui donné au même dans le champs 'name' du MA en question, sur cet exemple c'est MA0.
  - 'useLogService = 1' : Pour utiliser le service de log dans DIET (sinon les SeD n'émettent aucun message).
- 'vishnuId=1' : L'id de VISHNU à utiliser dans la DB (valeur '1' par défaut) .
  - 'databaseType=postgresql' : Pour utiliser une base postgresQL. Pour utiliser une base MySQL, il faut mettre 'mysql' comme type.
  - 'databaseHost=localhost' : Le nom DNS du serveur de bases de données, ou 'localhost' si la base est locale.
  - 'databaseName=vishnu' : Le nom de la base de données
  - 'databaseUserName=vishnu\_user' : Le nom d'utilisateur pour se connecter à la DB.
  - 'databaseUserPassword=vishnu\_user' : Le mot de passe pour se connecter à la DB.
  - 'databaseConnectionsNb=5' : Le nombre de connexions ouvertes sur la base de données par le SeD afin de traiter des requêtes en parallèle. (le nombre par défaut est 10). Si le nombre d'utilisateurs simultanés du SeD est grand, cette valeur doit être augmentée dans la limite du nombre de connexions simultanées autorisées par le serveur de base de données.
  - 'sendmailScriptPath=/usr/local/vishnu/sbin/sendmail.py' : Le script à utiliser pour envoyer les mails. Il est installé avec le module UMS dans le sous-répertoire sbin/ du répertoire d'installation.
6. Sur la machine hôte de Torque, lancer le serveur (pbs\_serv), le scheduler (pbs\_sched) et l'ordonnanceur (pbs\_mom) de Torque.
  7. Lancer le SeD TMS sur la machine hôte de Torque. La commande de lancement est : **tmssed ~/tms\_sed.cfg**. Le paramètre est un fichier de configuration VISHNU. Voir le paragraphe 6.6 pour un exemple de fichier de configuration. Les paramètres à fournir correspondent à ceux de UMS avec de plus :
    - 'intervalMonitor = 1' : La fréquence (en secondes) de mise à jour des données dans la base concernant les états des jobs.
    - 'batchSchedulerType=TORQUE' : Le type du batch scheduler utilisé.
    - 'vishnuMachineId=machine\_1' : L'identifiant VISHNU de la machine ou le SeD TMS est lancé (obtenu en utilisant la commande vishnu\_list\_machines).
  8. Sur la machine hôte de SLURM, lancer les serveurs *slurmd*, *slurmctld* et *slurmdbd*.
  9. Lancer le SeD TMS sur la machine hôte de SLURM. La commande de lancement est : **tmssed ~/tms\_sed.cfg**. Le paramètre est un fichier de configuration VISHNU. Voir le paragraphe 6.6 pour un exemple de fichier de configuration. Les paramètres à fournir correspondent à ceux de UMS avec de plus :
    - 'intervalMonitor = 1' : La fréquence (en secondes) de mise à jour des données dans la base concernant les états des jobs.
    - 'batchSchedulerType=SLURM' : Le type du batch scheduler utilisé.
    - 'vishnuMachineId=machine\_2' : L'identifiant VISHNU de la machine ou le SeD TMS est lancé (obtenu en utilisant la commande vishnu\_list\_machines).
  10. Lancer le SeD IMS sur toutes la machine préfrontale. La commande de lancement est : **imssed ~/ims\_sed.cfg**. Le paramètre est un fichier de configuration VISHNU. Voir le paragraphe 6.5 pour un exemple de fichier de configuration. Les paramètres à fournir correspondent à ceux de UMS. Il faut noter que IMS doit être lancé sur toutes les machines à monitorer.
  11. Lancer le SeD FMS sur une machine (sur la figure exemple 4.1 elle est différente de la préfrontale et de la machine avec le batch, mais cela peut être n'importe laquelle). La commande de lancement est : **fmssed ~/fms\_sed.cfg**. Le paramètre est un fichier de configuration VISHNU. Voir le paragraphe 6.7 pour un exemple de fichier de configuration. Les paramètres à fournir correspondent à ceux de UMS avec de plus :
    - 'intervalMonitor = 1' : La fréquence (en secondes) de mise à jour des données dans la base concernant les états des transferts de fichiers.
  12. Les modules de VISHNU sont prêts à être utilisés. Pour ce faire, un client doit se connecter et soumettre des requêtes à VISHNU au moyen des commandes UMS, TMS, FMS et IMS (voir le manuel de l'utilisateur VISHNU pour plus d'informations sur les commandes utilisateurs disponibles).

## 6.2 Lancement sur plusieurs réseaux avec une version de DIET supérieure ou égale à 2.8

Un module VISHNU peut être lancé manuellement. Pour le déploiement on se base sur le modèle de la figure 4.2 du document [ARCH] en supposant pour simplifier que sur un réseau tout est lancé sur la même machine et que les réseaux "Network C" et "Network D" sont confondus (pas de forwarder à mettre entre eux).

1. Avoir une base de données PostgreSQL accessible et initialisée (tables créées et premières données créées). Des scripts SQL sont fournis pour cela. Avoir la base PostgreSQL configurée pour qu'elle soit accessible par VISHNU (voir le fichier de configuration `pg_hba.conf` qui se situe dans un répertoire de la forme `'/etc/postgresql/8.3/main'` pour configurer la base).
2. Lancer le service de nommage CORBA sur les différents réseaux. La commande est **omniNames -start** pour la première fois, sinon **omniNames** suffit. Attention, dans la configuration de l'omninames, bien utiliser l'adresse de l'hôte et non pas `'localhost'` ou `'127.0.0.1'`.
3. Lancer un démon forwarder sur le réseau 2 (Network B) :  
**dietForwarder --name forwarder2**
  - forwarder2 : nom qui identifie le démon forwarder sur ce réseau
4. Lancer un démon forwarder sur le réseau 1 (network C) :  
**dietForwarder --peer-name forwarder2 --ssh-host nom\_DNS\_machine\_distante --remote-port 50005 --name forwarder --remote-host localhost --ssh-login login**
  - --peer-name : le nom donné au forwarder de l'autre côté, ici forwarder2.
  - --ssh-host : la nom DNS de la machine distante pour connecter le tunnel SSH.
  - --remote-port : le port à utiliser pour le tunnel ssh, ici 50005.
  - --name : le nom identifiant ce démon forwarder, c'est-à-dire `'forwarder1'`.
  - --remote-host : le nom de l'adresse loopback, c'est-à-dire `'localhost'`.
  - --ssh-login : le login de connection sur la machine distante.
5. Si la clé ssh est accessible (le chemin par défaut est le répertoire `$HOME/.ssh/`) et que la clé est protégée par une passphrase, la passphrase est demandée et la connexion ssh s'établit.
6. Lancer un démon forwarder pour le log service sur le réseau 2 (network B) :  
**logForwarder --name logforwarder2**
  - logforwarder2 : nom qui identifie le démon forwarder sur ce réseau
7. Lancer un démon log forwarder sur le réseau 1 (network C) :  
**logForwarder --peer-name logforwarder2 --ssh-host nom\_DNS\_machine\_distante --remote-port 50005 --name log-forwarder --remote-host localhost --ssh-login login**
  - --peer-name : le nom donné au forwarder de l'autre côté, ici forwarder2.
  - --ssh-host : la nom DNS de la machine distante pour connecter le tunnel SSH.
  - --remote-port : le port à utiliser pour le tunnel ssh, ici 50005.
  - --name : le nom identifiant ce démon forwarder, c'est-à-dire `'forwarder1'`.
  - --remote-host : le nom de l'adresse loopback, c'est-à-dire `'localhost'`.
  - --ssh-login : le login de connection sur la machine distante.
8. Si la clé ssh est accessible (le chemin par défaut est le répertoire `$HOME/.ssh/`) et que la clé est protégée par une passphrase, la passphrase est demandée et la connexion ssh s'établit.
9. Lancer le log service de DIET sur la machine du network B. La commande est **LogCentral --config config.cfg** avec `config.cfg` un fichier de configuration pour le log central.

10. Lancer le MA avec son fichier de configuration sur le réseau network B : **dietAgent config.cf**. Voir le paragraphe 6.3 pour un exemple de configuration. Ce fichier de configuration peut contenir les 3 lignes suivantes :
  - 'traceLevel = 0' : Le niveau de verbosité du master agent, cette valeur peut être entre 0 et 10 compris.
  - 'agentType = DIET\_MASTER\_AGENT' : Le type de l'agent, l'autre type disponible est DIET\_LOCAL\_AGENT mais dans notre cas il faut DIET\_MASTER\_AGENT.
  - 'name = MA0' : Le nom que l'on veut donner à l'agent, dans notre exemple on va l'appeler MA0.
11. Lancer le SeD, par exemple un UMS sur le réseau network C. Un exemple de commande de lancement est : **umssed ~/ums\_sed.cfg**. Se référer au chapitre de déploiement pour le contenu du fichier, c'est le même à utiliser que dans un déploiement sur un seul réseau.
12. Le module, ici UMS, de VISHNU est prêt à être utilisé. Pour ce faire, un client doit se connecter et soumettre des requêtes à VISHNU au moyen de clients UMS. Si le client est encore sur un autre réseau il faudra mettre la paire de dietForwarder entre le réseau client et l'un des réseaux connecté à la hiérarchie déployée.

## 6.3 Lancement sur plusieurs réseaux avec une version de DIET inférieure à 2.8

Un module VISHNU peut être lancé manuellement. Pour le déploiement on se base sur le modèle de la figure 4.2 du document [ARCH] en supposant pour simplifier que sur un réseau tout est lancé sur la même machine et que les réseaux "Network C" et "Network D" sont confondus (pas de forwarder à mettre entre eux).

1. Avoir une base de données PostgreSQL accessible et initialisée (tables créées et premières données créées). Des scripts SQL sont fournis pour cela. Avoir la base PostgreSQL configurée pour qu'elle soit accessible par VISHNU (voir le fichier de configuration pg\_hba.conf qui se situe dans un répertoire de la forme '/etc/postgresql/8.3/main' pour configurer la base).
2. Lancer le service de nommage CORBA sur les différents réseaux. La commande est **omniNames -start** pour la première fois, sinon **omniNames** suffit. Attention, dans la configuration de l'omninames, bien utiliser l'adresse de l'hôte et non pas 'localhost' ou '127.0.0.1'.

3. Lancer un démon forwarder sur le réseau 2 (Network B) :

**dietForwarder --name forwarder2 --net-config forwarder2.cfg**

- forwarder2 : nom qui identifie le démon forwarder sur ce réseau
- forwarder2.cfg : fichier de configuration (voir le paragraphe 6.8 pour un exemple de configuration) contenant des règles appliquées aux connections entre agents de l'intergiciel. Les règles sont de deux sortes : 'accept:' ou 'reject:'. Dans l'exemple de la figure 6.8, les règles utilisées sont :
  - une règle 'accept' correspondant au filtre sur la source des connections acceptées. '.' signifie les connections provenant de n'importe quelle adresse IP.
  - deux règles 'reject' contenant les adresses destination à rejeter. Il est nécessaire que l'adresse IP du serveur sur lequel tourne le démon forwarder soit indiquée dans une règle de ce type. Dans l'exemple le serveur ayant deux adresses IP on a indiqué une règle pour chaque adresse.

4. Lancer un démon forwarder sur le réseau 1 (network C) :

**dietForwarder -C --peer-name forwarder2 --ssh-host nom\_DNS\_machine\_distante --remote-port 50005 --name forwarder --remote-host localhost --net-config forwarder1.cfg --ssh-login login**

- -C : indique que c'est ce démon forwarder qui crée le tunnel SSH.
- --net-config : le fichier de configuration, ici forwarder1.cfg. Ce fichier doit contenir des règles 'reject' et 'accept' également, sur le même modèle que forwarder2.cfg.
- --peer-name : le nom donné au forwarder de l'autre côté, ici forwarder2.
- --ssh-host : la nom DNS de la machine distante pour connecter le tunnel SSH.
- --remote-port : le port à utiliser pour le tunnel ssh, ici 50005.
- --name : le nom identifiant ce démon forwarder, c'est-à-dire 'forwarder1'.

- `--remote-host` : le nom de l'adresse loopback, c'est-à-dire 'localhost'.
  - `--ssh-login` : le login de connection sur la machine distante.
5. Si la clé ssh est accessible (le chemin par défaut est le répertoire \$HOME/.ssh/) et que la clé est protégée par une passphrase, la passphrase est demandée et la connexion ssh s'établit.
6. Lancer un démon forwarder pour le log service sur le réseau 2 (network B) :
- logForwarder --name logforwarder2 --net-config logforwarder2**
- `logforwarder2` : nom qui identifie le démon forwarder sur ce réseau
  - `logforwarder2.cfg` : fichier de configuration (voir le paragraphe 6.8 pour un exemple de fichier de configuration) contenant des règles appliquées aux connections entre agents de l'intergiciel. Les règles sont de deux sortes : 'accept:' ou 'reject:'. Dans l'exemple de le paragraphe 6.8, les règles utilisées sont :
    - une règle 'accept' correspondant au filtre sur la source des connections acceptées. '.' signifie les connections provenant de n'importe quelle adresse IP.
    - deux règles 'reject' contenant les adresses destination à rejeter. Il est nécessaire que l'adresse IP du serveur sur lequel tourne le démon forwarder soit indiquée dans une règle de ce type. Dans l'exemple le serveur ayant deux adresses IP on a indiqué une règle pour chaque adresse.
7. Lancer un démon log forwarder sur le réseau 1 (network C) :
- logForwarder -C --peer-name logforwarder2 --ssh-host nom\_DNS\_machine\_distante --remote-port 50005 --name logforwarder --remote-host localhost --net-config logforwarder1.cfg --ssh-login login**
- `-C` : indique que c'est ce démon forwarder qui crée le tunnel SSH.
  - `--peer-name` : le nom donné au forwarder de l'autre côté, ici `forwarder2`.
  - `--net-config` : le fichier de configuration, ici `forwarder1.cfg`. Ce fichier doit contenir des règles 'reject' et 'accept' également, sur le même modèle que `forwarder2.cfg`.
  - `--ssh-host` : la nom DNS de la machine distante pour connecter le tunnel SSH.
  - `--remote-port` : le port à utiliser pour le tunnel ssh, ici 50005.
  - `--name` : le nom identifiant ce démon forwarder, c'est-à-dire 'forwarder1'.
  - `--remote-host` : le nom de l'adresse loopback, c'est-à-dire 'localhost'.
  - `--ssh-login` : le login de connection sur la machine distante.
8. Si la clé ssh est accessible (le chemin par défaut est le répertoire \$HOME/.ssh/) et que la clé est protégée par une passphrase, la passphrase est demandée et la connexion ssh s'établit.
9. Lancer le log service de DIET sur la machine du network B. La commande est **LogCentral --config config.cfg** avec `config.cfg` un fichier de configuration pour le log central.
10. Lancer le MA avec son fichier de configuration sur le réseau network B : **dietAgent config.cf**. Voir le paragraphe 6.3 pour un exemple de configuration. Ce fichier de configuration peut contenir les 3 lignes suivantes :
- `'traceLevel = 0'` : Le niveau de verbosité du master agent, cette valeur peut être entre 0 et 10 compris.
  - `'agentType = DIET_MASTER_AGENT'` : Le type de l'agent, l'autre type disponible est `DIET_LOCAL_AGENT` mais dans notre cas il faut `DIET_MASTER_AGENT`.
  - `'name = MA0'` : Le nom que l'on veut donner à l'agent, dans notre exemple on va l'appeler MA0.
11. Lancer le SeD, par exemple un UMS sur le réseau network C. Un exemple de commande de lancement est : **umssed ~/ums\_sed.cfg**. Se référer au chapitre de déploiement pour le contenu du fichier, c'est le même à utiliser que dans un déploiement sur un seul réseau.
12. Le module, ici UMS, de VISHNU est prêt à être utilisé. Pour ce faire, un client doit se connecter et soumettre des requêtes à VISHNU au moyen de clients UMS. Si le client est encore sur un autre réseau il faudra mettre la paire de `dietForwarder` entre le réseau client et l'un des réseaux connecté à la hiérarchie déployée.

## 6.4 Exemple de fichier de configuration d'un MA

```
traceLevel = 0
name = MA0
agentType = DIET_MASTER_AGENT
```

## 6.5 Exemple de fichier de configuration pour un SeD DIET

```
traceLevel = 0
parentName = MA0
useLogService = 1
```

## 6.6 Exemple de fichier de configuration d'un SeD UMS

```
# Configuration of the VISHNU UMS SeD
dietConfigFile=/usr/local/etc/SeD.cfg
vishnuId=1
databaseType=postgresql
databaseHost=localhost
databaseName=vishnu
databaseUserName=vishnu_user
databaseUserPassword=vishnu_user
databaseConnectionsNb=5
sendmailScriptPath=/usr/local/sbin/sendmail.py
vishnuMachineId=machine_1
```

## 6.7 Exemple de fichier de configuration d'un SeD TMS

```
# Configuration of the VISHNU TMS SeD
dietConfigFile=/usr/local/etc/SeD.cfg
vishnuId=1
databaseType=postgresql
databaseHost=localhost
databaseName=vishnu
databaseUserName=vishnu_user
databaseUserPassword=vishnu_user
databaseConnectionsNb=5
sendmailScriptPath=/usr/local/sbin/sendmail.py
vishnuMachineId=machine_1
# Configuration propres à TMS
# batchSchedulerType=SLURM si le batchScheduler est de type SLURM
batchSchedulerType=TORQUE
intervalMonitor = 1
```

## 6.8 Exemple de fichier de configuration d'un SeD FMS

```
# Configuration of the VISHNU FMS SeD
dietConfigFile=/usr/local/etc/SeD.cfg
vishnuId=1
databaseType=postgresql
```

```

databaseHost=localhost
databaseName=vishnu
databaseUserName=vishnu_user
databaseUserPassword=vishnu_user
databaseConnectionsNb=5
vishnuMachineId=machine_1
# Configuration propres à FMS
intervalMonitor = 1

```

## 6.9 Exemple de fichier d'un forwarder

```

# accept everything from everyone
accept:.*

# reject its own ip
reject:192\.\168\.\1\.\6

```

## 6.10 Exemple de fichier du LogCentral

```

# an empty configuration file for LogService
[General]

[DynamicTagList]
[StaticTagList]
[UniqueTagList]
[VolatileTagList]

```

## 6.11 Configuration de l'envoi des emails par VISHNU

Le processus UMS SeD utilise le fichier 'sendmail.py' (fourni dans l'installation VISHNU, dans le sous-répertoire sbn/) pour envoyer des emails aux utilisateurs lors de certaines opérations. Ce fichier peut être modifié par l'administrateur afin de s'adapter à la méthode d'envoi d'email propre au serveur sur lequel est installé le SeD. Par défaut, la configuration fournie se connecte sur le serveur SMTP de 'localhost' sur le port 25, sans authentification.

Les paramètres suivants peuvent être configurés dans le script sendmail.py :

Option	Ligne du script sendmail.py à modifier
login	<code>parser.add_option("--login", dest="login", help="", default="[ ↵ login_utilisateur]")</code>
password	<code>parser.add_option("--password", dest="password", help="smtp password", ↵ default="[password_utilisateur]")</code>
hostname	<code>parser.add_option("--hostname", dest="host", help="smtp host", default="[ ↵ nom_serveur_SMTP]")</code>
port	<code>parser.add_option("--port", dest="port", help="smtp port [default: 25]", ↵ type=int, default="[no_port]")</code>



Option	Ligne du script sendmail.py à modifier
SSL	<pre> parser.add_option("--ssl", action="store_true", dest="use_ssl", help="enable ↔ ssl support [default: %default - default port: 587 ]", default=True) </pre>

## 6.12 Configuration des clés privées/publiques ssh requises pour FMS

Toutes les commandes exécutées par le SeD FMS sont lancées via ssh sous le nom de l'utilisateur ayant émis la requête. Les services FMS sont de deux types : il y a ceux qui n'impliquent qu'une machine distante : Exemple `getFilesInfo`, `listDir`, etc.. et ceux qui impliquent au moins deux machines distantes : machine source et destination pour les transferts de fichiers.

- Dans le premier cas le SeD se connecte sur la machine distante et effectue la commande. Par conséquent la clé publique du SeD doit être ajoutée au fichier `authorized_keys` (`$HOME/.ssh/authorized_keys`) de l'utilisateur de la machine distante concernée.
- Dans le second cas, deux connexions ssh sont nécessaires. Le SeD se connecte sur la machine source et lance le transfert (seconde connexion) vers la machine destination. Par conséquent:
  - la clé publique du SeD doit être ajoutée au fichier `authorized_key` de la machine source pour permettre la première connexion.
  - La machine source doit pouvoir se connecter sur la machine destination par ssh, avec la clé privée enregistrée dans la base de VISHNU lors de l'ajout du compte (local account) liant la machine source à VISHNU. Par ailleurs si le mécanisme d'agent forwarding (de ssh) est activée entre ces différentes machines, il n'est alors plus nécessaire qu'il y ait un autre couple de clés entre la machine source et destination.

En somme, il est alors obligatoire que la clé publique du SeD soit ajoutée à tous les comptes utilisateurs des machines impliquées par les requêtes FMS. Toutes les clés protégées par des passphrases devront être stockées par un agent ssh pour permettre les authentifications automatiques.

## 6.13 Configuration des clés privées/publiques ssh requises pour TMS

Les commandes de soumission, d'annulation et de récupération des résultats de jobs exécutées par le SeD TMS sont lancées via ssh sous le nom de l'utilisateur ayant émis la requête. Pour pouvoir exécuter ces commandes correctement, la clé publique du compte dédié au SeD TMS doit être ajoutée au fichier `authorized_keys` (`$HOME/.ssh/authorized_keys`) de l'utilisateur. Toutes les clés protégées par des passphrases devront être stockées par un agent ssh pour permettre les authentifications automatiques.

## 6.14 Test d'exécution d'un service depuis une machine client par shell

1. Une fois que la plateforme a été installée, se mettre sur un poste client avec VISHNU d'installé. Se référer au document [VISHNU\_USERMANUAL] pour l'installation de la partie client.
2. Exporter la variable d'environnement `VISHNU_CONFIG_FILE` dans un script de configuration client. Se référer au guide d'installation du client [VISHNU\_USER\_GUIDE] pour connaître le contenu d'un fichier client.
3. Lancer la commande dans le shell `'vishnu_connect user'`, ou `user` représente un nom d'utilisateur dans la base de données `vishnu`
4. Taper le mot de passe associé correspondant à l'utilisateur `'user'`
5. Sur le client, un affichage doit signaler que le service a réussi. Dans le terminal où le SeD UMS a été lancé et dans le terminal où le MA tourne, selon le niveau de verbosité, plus ou moins d'informations, concernant le service effectué, doivent apparaître. Le message affiché contient au moins une ligne similaire : `sessionId: root-2011-Jul-11-14:22:14.403491:86690`, qui indique l'identifiant de la session ouverte.
6. Fermer la session avec `'vishnu_close'`. Aucune erreur ne doit être remontée lors de ces tests.

## Chapter 7

# Administration

### 7.1 Présentation

Le module UMS correspond à la gestion des utilisateurs et des machines de VISHNU. Il permet aussi de sauvegarder la configuration de VISHNU à chaud et de la restaurer si besoin est. Dans toute la suite du chapitre, on supposera que l'utilisateur est déjà connecté avec un compte administrateur de VISHNU pour pouvoir réaliser ces manipulations. De plus, on présentera l'utilisation des commandes depuis le shell, mais cela reste valable depuis les API Python ou C++.

L'API est disponible dans le document [VISHNU\_API]

### 7.2 Gestion des utilisateurs (UMS)

1. L'ajout d'un utilisateur se fait à l'aide de la commande `'vishnu_add_user'`. Elle prend en paramètre le prénom de l'utilisateur, son nom de famille, les droits qui lui sont associés dans VISHNU (administrateur ou simple utilisateur) et son adresse de courriel. Tout ces paramètres sont obligatoires. Un privilège à 1 signifie administrateur, un privilège à 0 signifie un utilisateur. L'identifiant de l'utilisateur est généré et renvoyé.
2. La mise à jour d'un utilisateur ne peut être faite que par un administrateur. Cette mise à jour se fait avec un appel à la commande `'vishnu_update_user'` et permet de modifier les paramètres de l'ajout (nom, prénom, statut, courriel). Il faut avoir l'identifiant de l'utilisateur (généré lors de la création de l'utilisateur) pour le désigner lors de la mise à jour.  
Note : le changement du statut d'un utilisateur à l'état "INACTIVE" correspond à un blocage de son compte.
3. La suppression d'un utilisateur efface toutes les informations liées à l'utilisateur de la base de donnée. Cette suppression se fait à l'aide de la commande `'vishnu_delete_user'`.
4. La liste des utilisateurs ne peut être faite que par un administrateur. Cela se fait avec la commande `'vishnu_list_user'`. Cette commande peut prendre en paramètre l'identifiant d'un utilisateur pour n'avoir les informations que concernant cet utilisateur.
5. Seul un administrateur peut réinitialiser le mot de passe d'un utilisateur de VISHNU. Pour ce faire, il doit appeler la commande `'vishnu_reset_password'` en fournissant l'identifiant de l'utilisateur dont l'administrateur veut réinitialiser le mot de passe. Le nouveau mot de passe est temporaire et renvoyé par la commande. Lors de la prochaine connexion, l'utilisateur devra changer son mot de passe avec `'vishnu_change_password'`.

### 7.3 Gestion des machines (UMS+IMS)

1. L'ajout d'une machine se fait à l'aide de la commande `'vishnu_add_machine'`. Cette commande prend en paramètre le nom de la machine, le site où elle se trouve, le langage de la description qui sera donnée pour la machine, le fichier contenant la clé publique et la description. Ces paramètres sont obligatoires, en passant par le shell, la description n'a pas besoin



d'être fournie en paramètre mais elle est alors demandée à l'administrateur avant d'ajouter la machine. A la fin de l'ajout, l'identifiant généré pour la machine est renvoyé.

2. La mise à jour d'une machine se fait à l'aide de la commande 'vishnu\_update\_machine' et permet de modifier les paramètres mis lors de l'ajout de la machine. Il faut utiliser l'identifiant de la machine pour l'identifier lors de la mise à jour.

Note : le changement du statut d'une machine à l'état "INACTIVE" correspond à un blocage de la machine. Cela rend la machine inaccessible aux utilisateurs de VISHNU mais toujours visible pour les administrateurs.

3. La suppression d'une machine se fait à l'aide de la commande 'vishnu\_delete\_machine' avec l'identifiant de la machine à supprimer. Cela supprime la machine de la base de données, ainsi que toutes les informations qui y sont attachées (Attention : cette commande est irréversible).
4. Les utilisateurs peuvent lister les machines, mais un administrateur a en plus une option qui est l'identifiant d'un utilisateur. Ceci lui permet de lister les machines sur lesquelles l'utilisateur a un compte VISHNU.
5. La mise à jour d'informations système d'une machine se fait à l'aide de la commande 'vishnu\_set\_system\_info' et permet d'ajouter ou modifier des informations système d'une machine. Il faut utiliser l'identifiant de la machine pour l'identifier lors de la mise à jour.

## 7.4 Gestion de la plateforme (UMS)

1. L'administrateur peut faire une sauvegarde à chaud à un moment donné de VISHNU. Ceci sauvegarde les utilisateurs, les machines et les comptes des utilisateurs. Le fichier, dans lequel la configuration est, est retourné par la fonction. La fonction est 'vishnu\_save\_configuration', pas besoin de paramètres.
2. L'administrateur peut recharger une configuration précédente de VISHNU à l'aide de la commande 'vishnu\_restore\_configuration' qui a besoin du fichier de sauvegarde pour recharger la base. Avant de pouvoir lancer cette restauration, tous les utilisateurs de VISHNU doivent être déconnectés.
3. Un administrateur peut également définir les valeurs par défaut des options de VISHNU pour tout les utilisateurs (ces options sont le temps de déconnexion par défaut et le type de fermeture d'une session par défaut). Cela se fait en appelant 'vishnu\_configure\_default\_option' en donnant le nom de l'option et sa nouvelle valeur.

## 7.5 Options propres à l'administrateur dans les commandes utilisateurs(UMS+FMS)

1. Dans la fonction 'vishnu\_connect', un administrateur peut donner l'identifiant d'un utilisateur pour se connecter sous le nom de cet utilisateur dans VISHNU.
2. Dans la fonction 'vishnu\_list\_history\_cmd', l'administrateur peut lister toutes les commandes de tout les utilisateurs ou les commandes d'un utilisateur en particulier en fournissant l'identifiant de l'utilisateur.
3. Dans la fonction 'vishnu\_list\_local\_accounts', l'administrateur peut lister toutes les comptes de tout les utilisateurs ou les comptes d'un utilisateur particulier en fournissant l'identifiant de l'utilisateur.
4. Dans la fonction 'vishnu\_list\_options', l'administrateur peut lister toutes les options de tout les utilisateurs ou les options d'un utilisateur en particulier en fournissant l'identifiant de l'utilisateur.
5. Dans la fonction 'vishnu\_list\_sessions', l'administrateur peut lister toutes les sessions de tous les utilisateurs ou les sessions d'un utilisateur en particulier en fournissant l'identifiant de l'utilisateur, ou les sessions sur une machine particulière en fournissant l'identifiant de la machine.
6. Dans la fonction 'vishnu\_list\_file\_transfers', l'administrateur peut lister tous les transferts de fichiers de tous les utilisateurs ou ceux d'un utilisateur en particulier en fournissant l'identifiant de l'utilisateur, ou lister les transferts impliquant une machine particulière (qui peut être source ou destination du transfert) en fournissant l'identifiant de la machine.
7. Dans la fonction 'vishnu\_stop\_file\_transfers', l'administrateur peut annuler tous les transferts de fichiers de tous les utilisateurs ou ceux d'un utilisateur en particulier en fournissant l'identifiant de l'utilisateur, ou annuler les transferts impliquant une machine particulière (qui peut être source ou destination du transfert) en fournissant l'identifiant de la machine.

## 7.6 Gestion des processus VISHNU et délestage (IMS)

- L'administrateur peut lister les processus VISHNU, sur toute la plateforme ou sur une machine particulière. Fonction associée : `vishnu_get_processes`.
- L'administrateur peut arrêter un processus VISHNU, ce processus ne sera pas redémarrer automatiquement. Attention : l'administrateur doit avoir un compte sur la machine Fonction associée : `vishnu_stop`
- L'administrateur peut redémarrer un processus VISHNU sur une machine, ce processus doit avoir déjà tourné pour VISHNU sur cette machine et l'administrateur doit avoir un compte sur la machine. Fonction associée : `vishnu_restart`
- L'administrateur peut délester une machine selon deux modes. Dans le mode HARD, tout les processus VISHNU de la machine sont arrêtés. Dans le mode SOFT, seul FMS et TMS sont touchés, on arrête tout leurs transferts et jobs en cours. Fonction associée : `vishnu_load_shed`

## 7.7 Surveillance de l'état des machines (IMS)

- Un administrateur peut fixer la fréquence de mise à jour de l'enregistrement de l'état des machines. Fonction associée : `vishnu_set_update_frequency`
- Un administrateur peut obtenir la fréquence de mise à jour de l'enregistrement de l'état des machines. Fonction associée : `vishnu_get_update_frequency`
- Un administrateur peut fixer un seuil sur une machine. Ce seuil peut être l'utilisation du CPU, la mémoire libre restante ou l'espace disque restant. Lors de l'enregistrement de l'état d'une machine, si un seuil est atteint sur cette machine, l'administrateur est notifié par mail de ce dépassement. Fonction associée : `vishnu_set_threshold`
- Un administrateur peut obtenir les seuils fixés sur une machine. Pour plus d'informations sur les seuils voir la partie pour fixer les seuils ci-dessus. Fonction associée : `vishnu_get_threshold`

## 7.8 Définition des formats des identifiants (IMS)

- Un administrateur peut fixer le format des identifiants VISHNU automatiquement générés pour les utilisateurs, les machines, les jobs soumis aux batchs scheduler et les transferts de fichiers. Ces identifiants peuvent contenir plusieurs variables :
  - '\$DAY' : Variable qui sera remplacée par le jour de création (1-31)
  - '\$MONTH' : Variable qui sera remplacée par le mois de création (1-12)
  - '\$YEAR' : Variable qui sera remplacé par l'année de création (0-99)
  - '\$CPT' : Variable compteur qui est automatiquement incrémentée
  - '\$SITE' : Pour les utilisateurs ou machine, une information sur le lieu
  - '\$UNAME' : Pour les utilisateurs, variable remplacée par le nom de l'utilisateur
  - '\$MANAME' : Pour les machines, variable remplacée par le nom de la machine

Attention la variable conteur est obligatoire pour éviter la génération d'un identifiant déjà existant. Fonctions associées : `define_file_format`, `define_machine_format`, `define_task_format`, `define_user_format`.

## Chapter 8

# UMS Command reference

### 8.1 vishnu\_add\_user

vishnu\_add\_user — adds a new VISHNU user

#### Synopsis

```
vishnu_add_user [-h] firstname lastname privilege email
```

#### DESCRIPTION

This command allows an admin to add a new user in VISHNU. Several user information are necessary such as: lastname, firstname and email address. The admin also gives a VISHNU privilege to the new user and a new userId and password are sent to the user by email.

#### OPTIONS

**-h** *help* help about the command.

#### ENVIRONMENT

**VISHNU\_CONFIG\_FILE** Contains the path to the local configuration file for VISHNU.

#### DIAGNOSTICS

The following diagnostics may be issued on stderr and the command will return the code provided within brackets:

"Vishnu not available (Service bus failure)" [1]

"Vishnu not available (Database error)" [2]

"Vishnu not available (Database connection)" [3]

"Vishnu not available (System)" [4]

"Internal Error: Undefined exception" [9]

"There is no open session in this terminal" [13]

"Missing parameters" [14]

"Vishnu initialization failed" [15]

"Undefined error" [16]

"The userId already exists in the database" [22]

"The user is locked" [23]

"The user is not an administrator" [25]

"The mail adress is invalid" [27]

"The session key is unrecognized" [28]

"The sessionKey is expired. The session is closed." [29]

"The machine is locked" [34]

## EXAMPLE

To add the user Jean DUPONT as a simple user and with the mail dupont@gmail.com:

```
vishnu_add_user Jean DUPONT 0 dupont@gmail.com
```

## 8.2 vishnu\_update\_user

`vishnu_update_user` — updates the user information except the `userId` and the password

### Synopsis

```
vishnu_update_user [-h] [-f firstname] [-l lastname] [-p privilege] [-m email] [-s status] userId
```

### DESCRIPTION

This command allows an admin to update a VISHNU user information or to lock a user. When a user is locked, she/he can not uses VISHNU. However, it is not possible to change the privilege of another admin.

### OPTIONS

**-h** *help* help about the command.

**-f** *firstname* represents the updated firstname of the user.

**-l** *lastname* represents the updated lastname of the user.

**-p** *privilege* represents the updated privilege of the user. The value must be an integer. Predefined values are: 0 (USER), 1 (ADMIN).

**-m** *email* represents the updated email address of the user.

**-s** *status* represents the status of the user (LOCKED or ACTIVE). The value must be an integer. Predefined values are: 0 (INACTIVE), 1 (ACTIVE).

## ENVIRONMENT

**VISHNU\_CONFIG\_FILE** Contains the path to the local configuration file for VISHNU.

## DIAGNOSTICS

The following diagnostics may be issued on stderr and the command will return the code provided within brackets:

"Vishnu not available (Service bus failure)" [1]  
"Vishnu not available (Database error)" [2]  
"Vishnu not available (Database connection)" [3]  
"Vishnu not available (System)" [4]  
"Internal Error: Undefined exception" [9]  
"There is no open session in this terminal" [13]  
"Missing parameters" [14]  
"Vishnu initialization failed" [15]  
"Undefined error" [16]  
"The userId is unknown" [21]  
"The user is locked" [23]  
"Trying to lock a user account that is already locked" [24]  
"The user is not an administrator" [25]  
"The mail adress is invalid" [27]  
"The session key is unrecognized" [28]  
"The sessionKey is expired. The session is closed." [29]

## EXAMPLE

To update the mail address of a user user\_1 to jdupont@gmail.com:

```
vishnu_update_user -m jdupont@gmail.com user_1
```

### 8.3 vishnu\_delete\_user

**vishnu\_delete\_user** — removes a user from VISHNU

#### Synopsis

```
vishnu_delete_user [-h] userId
```

#### DESCRIPTION

This command allows an admin to delete a user from VISHNU. When a user is deleted from VISHNU all of her/his information are deleted from VISHNU. However, it is not possible to delete the VISHNU root user.

## OPTIONS

**-h** *help* help about the command.

## ENVIRONMENT

**VISHNU\_CONFIG\_FILE** Contains the path to the local configuration file for VISHNU.

## DIAGNOSTICS

The following diagnostics may be issued on stderr and the command will return the code provided within brackets:

"Vishnu not available (Service bus failure)" [1]

"Vishnu not available (Database error)" [2]

"Vishnu not available (Database connection)" [3]

"Vishnu not available (System)" [4]

"Internal Error: Undefined exception" [9]

"There is no open session in this terminal" [13]

"Missing parameters" [14]

"Vishnu initialization failed" [15]

"Undefined error" [16]

"The userId is unknown" [21]

"The user is locked" [23]

"The user is not an administrator" [25]

"The session key is unrecognized" [28]

"The sessionKey is expired. The session is closed." [29]

## EXAMPLE

To delete the user user\_1:

```
vishnu_delete_user user_1
```

## 8.4 vishnu\_reset\_password

`vishnu_reset_password` — resets the password of a user

### Synopsis

```
vishnu_reset_password [-h] userId
```

## DESCRIPTION

This command allows an admin to reset the password of the user. The password generated is temporary and must be changed for using VISHNU.

## OPTIONS

**-h** *help* help about the command.

## ENVIRONMENT

**VISHNU\_CONFIG\_FILE** Contains the path to the local configuration file for VISHNU.

## DIAGNOSTICS

The following diagnostics may be issued on stderr and the command will return the code provided within brackets:

"Vishnu not available (Service bus failure)" [1]

"Vishnu not available (Database error)" [2]

"Vishnu not available (Database connection)" [3]

"Vishnu not available (System)" [4]

"Internal Error: Undefined exception" [9]

"There is no open session in this terminal" [13]

"Missing parameters" [14]

"Vishnu initialization failed" [15]

"Undefined error" [16]

"The userId is unknown" [21]

"The user is locked" [23]

"The user is not an administrator" [25]

"The session key is unrecognized" [28]

"The sessionKey is expired. The session is closed." [29]

## EXAMPLE

To reset the password of the user user\_1:

```
vishnu_reset_password user_1
```

## 8.5 vishnu\_save\_configuration

`vishnu_save_configuration` — saves the configuration of VISHNU

## Synopsis

```
vishnu_save_configuration [-h]
```

## DESCRIPTION

This command allows an admin to save the VISHNU configuration. This configuration contains the list of users, the lists of machines and the list of local user configurations. It is saved on a xml format on a file registered on the directory \$HOME/.vishnu/configuration.

## OPTIONS

**-h** *help* help about the command.

## ENVIRONMENT

**VISHNU\_CONFIG\_FILE** Contains the path to the local configuration file for VISHNU.

## DIAGNOSTICS

The following diagnostics may be issued on stderr and the command will return the code provided within brackets:

"Vishnu not available (Service bus failure)" [1]

"Vishnu not available (Database error)" [2]

"Vishnu not available (Database connection)" [3]

"Vishnu not available (System)" [4]

"Internal Error: Undefined exception" [9]

"There is no open session in this terminal" [13]

"Missing parameters" [14]

"Vishnu initialization failed" [15]

"Undefined error" [16]

"The user is not an administrator" [25]

"A problem occurs during the configuration saving " [39]

## EXAMPLE

To save the current configuration:

```
vishnu_save_configuration
```

## 8.6 vishnu\_restore\_configuration

`vishnu_restore_configuration` — restores the configuration of VISHNU



## Synopsis

```
vishnu_restore_configuration [-h] filePath
```

## DESCRIPTION

This function must be used carefully as it replaces all the content of the VISHNU central database with the information stored in the provided file. This file contains the list of users, the lists of machines and the list of local user configurations. It can be created using the `vishnu_save_configuration` command. The "root" VISHNU user is the only user authorized to call this function, and this action must be done without any other user connected to VISHNU. After restoring, the vishnu database is re-initialized.

## OPTIONS

**-h** *help* help about the command.

## ENVIRONMENT

**VISHNU\_CONFIG\_FILE** Contains the path to the local configuration file for VISHNU.

## DIAGNOSTICS

The following diagnostics may be issued on stderr and the command will return the code provided within brackets:

"Vishnu not available (Service bus failure)" [1]  
"Vishnu not available (Database error)" [2]  
"Vishnu not available (Database connection)" [3]  
"Vishnu not available (System)" [4]  
"Internal Error: Undefined exception" [9]  
"There is no open session in this terminal" [13]  
"Missing parameters" [14]  
"Vishnu initialization failed" [15]  
"Undefined error" [16]  
"The user is not an administrator" [25]  
"A problem occurs during the configuration restoring" [40]

## EXAMPLE

To restore the configuration in `/tmp/toto.cfg`:

```
vishnu_restore_configuration /tmp/toto.cfg
```

## 8.7 vishnu\_add\_machine

`vishnu_add_machine` — adds a new machine in VISHNU

## Synopsis

```
vishnu_add_machine [-h] name site language sshPublicKeyFile machineDescription
```

## DESCRIPTION

This command allows an admin to add a new machine in VISHNU. Several machine information are mandatory such as: name, site, language and the public ssh key of the VISHNU system account on the machine. This public key will be provided automatically to all new VISHNU users who will have to add it to the authorized keys of their own account on the machine.

## OPTIONS

**-h** *help* help about the command.

## ENVIRONMENT

**VISHNU\_CONFIG\_FILE** Contains the path to the local configuration file for VISHNU.

## DIAGNOSTICS

The following diagnostics may be issued on stderr and the command will return the code provided within brackets:

"Vishnu not available (Service bus failure)" [1]

"Vishnu not available (Database error)" [2]

"Vishnu not available (Database connection)" [3]

"Vishnu not available (System)" [4]

"Internal Error: Undefined exception" [9]

"There is no open session in this terminal" [13]

"Missing parameters" [14]

"Vishnu initialization failed" [15]

"Undefined error" [16]

"The user is not an administrator" [25]

"The session key is unrecognized" [28]

"The sessionKey is expired. The session is closed." [29]

"The machineId already exists in the database" [33]

"The closure policy is unknown" [42]

## EXAMPLE

To add the machine perceval in paris with the description in french "ceci est une description" with the public key in /tmp/key.pub:

```
vishnu_add_machine perceval paris fr /tmp/key.pub "ceci est une description"
```

## 8.8 vishnu\_update\_machine

vishnu\_update\_machine — updates machine description

### Synopsis

```
vishnu_update_machine [-h] [-n name] [-s site] [-d machineDescription] [-l language] [-t status] [--  
k sshPublicKeyFile] machineId
```

### DESCRIPTION

This command allows an admin to update a VISHNU machine or to locked it. A machine locked is not usable.

### OPTIONS

- h** *help* help about the command.
- n** *name* represents the name of the machine.
- s** *site* represents the location of the machine.
- d** *machineDescription* represents the description of the machine.
- l** *language* represents the language used for the description of the machine.
- t** *status* represents the status of the machine. The value must be an integer. Predefined values are: 0 (INACTIVE), 1 (ACTIVE).
- k** *sshPublicKeyFile* contains the path to the SSH public key used by VISHNU to access local user accounts.

### ENVIRONMENT

**VISHNU\_CONFIG\_FILE** Contains the path to the local configuration file for VISHNU.

### DIAGNOSTICS

The following diagnostics may be issued on stderr and the command will return the code provided within brackets:

"Vishnu not available (Service bus failure)" [1]

"Vishnu not available (Database error)" [2]

"Vishnu not available (Database connection)" [3]

"Vishnu not available (System)" [4]

"Internal Error: Undefined exception" [9]

"There is no open session in this terminal" [13]

"Missing parameters" [14]

"Vishnu initialization failed" [15]

"Undefined error" [16]

"The user is not an administrator" [25]

"The session key is unrecognized" [28]

"The sessionKey is expired. The session is closed." [29]

"The machine id is unknown" [32]

"The closure policy is unknown" [42]

## EXAMPLE

To change the name of the machine whose id is machine\_1 to provencal:

```
vishnu_update_machine machine_1 -n provencal
```

## 8.9 vishnu\_delete\_machine

`vishnu_delete_machine` — removes a machine from VISHNU

### Synopsis

```
vishnu_delete_machine [-h] machineId
```

### DESCRIPTION

This command allows an admin to delete a machine from VISHNU. When a machine is deleted all of its information are deleted from VISHNU.

### OPTIONS

`-h help` help about the command.

### ENVIRONMENT

**VISHNU\_CONFIG\_FILE** Contains the path to the local configuration file for VISHNU.

### DIAGNOSTICS

The following diagnostics may be issued on stderr and the command will return the code provided within brackets:

"Vishnu not available (Service bus failure)" [1]

"Vishnu not available (Database error)" [2]

"Vishnu not available (Database connection)" [3]

"Vishnu not available (System)" [4]

"Internal Error: Undefined exception" [9]

"There is no open session in this terminal" [13]

"Missing parameters" [14]

"Vishnu initialization failed" [15]

"Undefined error" [16]

"The user is not an administrator" [25]

"The session key is unrecognized" [28]

"The sessionKey is expired. The session is closed." [29]

"The machine id is unknown" [32]

## EXAMPLE

To delete the machine machine\_1:

```
vishnu_delete_machine machine_1
```

## 8.10 vishnu\_list\_users

vishnu\_list\_users — lists VISHNU users

### Synopsis

```
vishnu_list_users [-h] [-i userIdOption]
```

### DESCRIPTION

This command allows an admin to display all users information except the passwords.

### OPTIONS

**-h** *help* help about the command.

**-i** *userIdOption* allows an admin to get information about a specific user identified by his/her *userId*.

### ENVIRONMENT

**VISHNU\_CONFIG\_FILE** Contains the path to the local configuration file for VISHNU.

### DIAGNOSTICS

The following diagnostics may be issued on stderr and the command will return the code provided within brackets:

"Vishnu not available (Service bus failure)" [1]

"Vishnu not available (Database error)" [2]

"Vishnu not available (Database connection)" [3]

"Vishnu not available (System)" [4]

"Internal Error: Undefined exception" [9]  
"There is no open session in this terminal" [13]  
"Missing parameters" [14]  
"Vishnu initialization failed" [15]  
"Undefined error" [16]  
"The userId is unknown" [21]  
"The user is not an administrator" [25]  
"The session key is unrecognized" [28]  
"The sessionKey is expired. The session is closed." [29]

## EXAMPLE

To list all the users:

```
vishnu_list_users
```

## 8.11 vishnu\_configure\_default\_option

`vishnu_configure_default_option` — configures a default option value

### Synopsis

```
vishnu_configure_default_option [-h] optionName value
```

### DESCRIPTION

Options in VISHNU corresponds to parameters of some VISHNU commands (e.g. the close policy for `vishnu_connect`) that can be preset in the user configuration stored by the VISHNU system. This command allows an administrator to configure the default value of an option; this is the value that will be applied when the user has no configuration defined for that option using the `vishnu_configure_option` command.

### OPTIONS

`-h help` help about the command.

### ENVIRONMENT

**VISHNU\_CONFIG\_FILE** Contains the path to the local configuration file for VISHNU.

## DIAGNOSTICS

The following diagnostics may be issued on stderr and the command will return the code provided within brackets:

**"Vishnu not available (Service bus failure)" [1]**

**"Vishnu not available (Database error)" [2]**

**"Vishnu not available (Database connection)" [3]**

**"Vishnu not available (System)" [4]**

**"Internal Error: Undefined exception" [9]**

**"There is no open session in this terminal" [13]**

**"Missing parameters" [14]**

**"Vishnu initialization failed" [15]**

**"Undefined error" [16]**

**"The user is not an administrator" [25]**

**"The session key is unrecognized" [28]**

**"The sessionKey is expired. The session is closed." [29]**

**"The name of the user option is unknown" [41]**

**"The value of the timeout is incorrect" [43]**

**"The value of the transfer command is incorrect" [44]**

## EXAMPLE

To configure the option VISHNU\_TIMEOUT with the value 42:

```
vishnu_configure_default_option VISHNU_TIMEOUT 42
```

## Chapter 9

# UMS C++ API Reference

### 9.1 addUser

`addUser` — adds a new VISHNU user

#### Synopsis

```
int vishnu::addUser(const string& sessionKey, User& newUser);
```

#### DESCRIPTION

This command allows an admin to add a new user in VISHNU. Several user information are necessary such as: lastname, firstname and email address. The admin also gives a VISHNU privilege to the new user and a new `userId` and password are sent to the user by email.

#### ARGUMENTS

***sessionKey*** Input argument. The `sessionKey` is the encrypted identifier of the session generated by VISHNU.

***newUser*** Input/Output argument. Object containing the new user information.

#### EXCEPTIONS

The following exceptions may be thrown:

"Vishnu not available (Service bus failure)" [1]

"Vishnu not available (Database error)" [2]

"Vishnu not available (Database connection)" [3]

"Vishnu not available (System)" [4]

"Internal Error: Undefined exception" [9]

"The userId already exists in the database" [22]

"The user is locked" [23]



"The user is not an administrator" [25]

"The mail adress is invalid" [27]

"The session key is unrecognized" [28]

"The sessionKey is expired. The session is closed." [29]

"The machine is locked" [34]

## 9.2 updateUser

updateUser — updates the user information except the userId and the password

### Synopsis

```
int vishnu::updateUser(const string& sessionKey, const User& user);
```

### DESCRIPTION

This command allows an admin to update a VISHNU user information or to lock a user. When a user is locked, she/he can not uses VISHNU. However, it is not possible to change the privilege of another admin.

### ARGUMENTS

*sessionKey* Input argument. The sessionKey is the encrypted identifier of the session generated by VISHNU.

*user* Input argument. Object containing user information.

### EXCEPTIONS

The following exceptions may be thrown:

"Vishnu not available (Service bus failure)" [1]

"Vishnu not available (Database error)" [2]

"Vishnu not available (Database connection)" [3]

"Vishnu not available (System)" [4]

"Internal Error: Undefined exception" [9]

"The userId is unknown" [21]

"The user is locked" [23]

"Trying to lock a user account that is already locked" [24]

"The user is not an administrator" [25]

"The mail adress is invalid" [27]

"The session key is unrecognized" [28]

"The sessionKey is expired. The session is closed." [29]

## 9.3 deleteUser

deleteUser — removes a user from VISHNU

### Synopsis

```
int vishnu::deleteUser(const string& sessionKey, const string& userId);
```

### DESCRIPTION

This command allows an admin to delete a user from VISHNU. When a user is deleted from VISHNU all of her/his information are deleted from VISHNU. However, it is not possible to delete the VISHNU root user.

### ARGUMENTS

*sessionKey* Input argument. The sessionKey is the encrypted identifier of the session generated by VISHNU.

*userId* Input argument. UserId represents the VISHNU user identifier of the user who will be deleted from VISHNU.

### EXCEPTIONS

The following exceptions may be thrown:

"Vishnu not available (Service bus failure)" [1]

"Vishnu not available (Database error)" [2]

"Vishnu not available (Database connection)" [3]

"Vishnu not available (System)" [4]

"Internal Error: Undefined exception" [9]

"The userId is unknown" [21]

"The user is locked" [23]

"The user is not an administrator" [25]

"The session key is unrecognized" [28]

"The sessionKey is expired. The session is closed." [29]

## 9.4 resetPassword

resetPassword — resets the password of a user

### Synopsis

```
int vishnu::resetPassword(const string& sessionKey, const string& userId, string& tmpPassword);
```

## DESCRIPTION

This command allows an admin to reset the password of the user. The password generated is temporary and must be changed for using VISHNU.

## ARGUMENTS

***sessionKey*** Input argument. The sessionKey is the encrypted identifier of the session generated by VISHNU.

***userId*** Input argument. UserId represents the VISHNU user identifier of the user whose password will be reset.

***tmpPassword*** Output argument. The temporary password generated by VISHNU.

## EXCEPTIONS

The following exceptions may be thrown:

"Vishnu not available (Service bus failure)" [1]

"Vishnu not available (Database error)" [2]

"Vishnu not available (Database connection)" [3]

"Vishnu not available (System)" [4]

"Internal Error: Undefined exception" [9]

"The userId is unknown" [21]

"The user is locked" [23]

"The user is not an administrator" [25]

"The session key is unrecognized" [28]

"The sessionKey is expired. The session is closed." [29]

## 9.5 saveConfiguration

saveConfiguration — saves the configuration of VISHNU

### Synopsis

```
int vishnu::saveConfiguration(const string& sessionKey, Configuration& configuration);
```

## DESCRIPTION

This commands allows an admin to save the VISHNU configuration. This configuration contains the list of users, the lists of machines and the list of local user configurations. It is saved on a xml format on a file registered on the directory \$HOME/.vishnu/configuration

## ARGUMENTS

***sessionKey*** Input argument. The sessionKey is the encrypted identifier of the session generated by VISHNU.

***configuration*** Output argument. The configuration is an object which encapsulates the configuration description.

## EXCEPTIONS

The following exceptions may be thrown:

"Vishnu not available (Service bus failure)" [1]

"Vishnu not available (Database error)" [2]

"Vishnu not available (Database connection)" [3]

"Vishnu not available (System)" [4]

"Internal Error: Undefined exception" [9]

"The user is not an administrator" [25]

"A problem occurs during the configuration saving " [39]

## 9.6 restoreConfiguration

restoreConfiguration — restores the configuration of VISHNU

### Synopsis

```
int vishnu::restoreConfiguration(const string& sessionKey, const string& filePath);
```

### DESCRIPTION

This function must be used carefully as it replaces all the content of the VISHNU central database with the information stored in the provided file. This file contains the list of users, the lists of machines and the list of local user configurations. It can be created using the `vishnu_save_configuration` command. The "root" VISHNU user is the only user authorized to call this function, and this action must be done without any other user connected to VISHNU. After restoring, the vishnu database is re-initialized.

### ARGUMENTS

*sessionKey* Input argument. The sessionKey is the encrypted identifier of the session generated by VISHNU.

*filePath* Input argument. The filePath is the path of the file used to restore VISHNU configuration.

### EXCEPTIONS

The following exceptions may be thrown:

"Vishnu not available (Service bus failure)" [1]

"Vishnu not available (Database error)" [2]

"Vishnu not available (Database connection)" [3]

"Vishnu not available (System)" [4]

"Internal Error: Undefined exception" [9]

"The user is not an administrator" [25]

"A problem occurs during the configuration restoring" [40]

## 9.7 addMachine

addMachine — adds a new machine in VISHNU

### Synopsis

```
int vishnu::addMachine(const string& sessionKey, Machine& newMachine);
```

### DESCRIPTION

This command allows an admin to add a new machine in VISHNU. Several machine information are mandatory such as: name, site, language and the public ssh key of the VISHNU system account on the machine. This public key will be provided automatically to all new VISHNU users who will have to add it to the authorized keys of their own account on the machine.

### ARGUMENTS

*sessionKey* Input argument. The sessionKey is the encrypted identifier of the session generated by VISHNU.

*newMachine* Input/Output argument. Is an object which encapsulates the information of the machine which will be added in VISHNU except the machine id which will be created automatically by VISHNU.

### EXCEPTIONS

The following exceptions may be thrown:

"Vishnu not available (Service bus failure)" [1]

"Vishnu not available (Database error)" [2]

"Vishnu not available (Database connection)" [3]

"Vishnu not available (System)" [4]

"Internal Error: Undefined exception" [9]

"The user is not an administrator" [25]

"The session key is unrecognized" [28]

"The sessionKey is expired. The session is closed." [29]

"The machineId already exists in the database" [33]

"The closure policy is unknown" [42]

## 9.8 updateMachine

updateMachine — updates machine description

### Synopsis

```
int vishnu::updateMachine(const string& sessionKey, const Machine& machine);
```

## DESCRIPTION

This command allows an admin to update a VISHNU machine or to locked it. A machine locked is not usable.

## ARGUMENTS

*sessionKey* Input argument. The sessionKey is the encrypted identifier of the session generated by VISHNU.

*machine* Input argument. Existing machine information.

## EXCEPTIONS

The following exceptions may be thrown:

"Vishnu not available (Service bus failure)" [1]

"Vishnu not available (Database error)" [2]

"Vishnu not available (Database connection)" [3]

"Vishnu not available (System)" [4]

"Internal Error: Undefined exception" [9]

"The user is not an administrator" [25]

"The session key is unrecognized" [28]

"The sessionKey is expired. The session is closed." [29]

"The machine id is unknown" [32]

"The closure policy is unknown" [42]

## 9.9 deleteMachine

deleteMachine — removes a machine from VISHNU

### Synopsis

```
int vishnu::deleteMachine(const string& sessionKey, const string& machineId);
```

## DESCRIPTION

This command allows an admin to delete a machine from VISHNU. When a machine is deleted all of its information are deleted from VISHNU.

## ARGUMENTS

*sessionKey* Input argument. The sessionKey is the encrypted identifier of the session generated by VISHNU.

*machineId* Input argument. MachineId represents the identifier of the machine.

## EXCEPTIONS

The following exceptions may be thrown:

"Vishnu not available (Service bus failure)" [1]  
"Vishnu not available (Database error)" [2]  
"Vishnu not available (Database connection)" [3]  
"Vishnu not available (System)" [4]  
"Internal Error: Undefined exception" [9]  
"The user is not an administrator" [25]  
"The session key is unrecognized" [28]  
"The sessionKey is expired. The session is closed." [29]  
"The machine id is unknown" [32]

## 9.10 listUsers

listUsers — lists VISHNU users

### Synopsis

```
int vishnu::listUsers(const string& sessionKey, ListUsers& listuser, const string& userIdOption = string());
```

### DESCRIPTION

This command allows an admin to display all users information except the passwords.

### ARGUMENTS

**sessionKey** Input argument. The sessionKey is the identifier of the session generated by VISHNU.

**listuser** Output argument. Listuser is the list of users .

**userIdOption** Input argument. Allows an admin to get information about a specific user identified by his/her userId.

## EXCEPTIONS

The following exceptions may be thrown:

"Vishnu not available (Service bus failure)" [1]  
"Vishnu not available (Database error)" [2]  
"Vishnu not available (Database connection)" [3]  
"Vishnu not available (System)" [4]  
"Internal Error: Undefined exception" [9]  
"The userId is unknown" [21]  
"The user is not an administrator" [25]  
"The session key is unrecognized" [28]  
"The sessionKey is expired. The session is closed." [29]

## 9.11 configureDefaultOption

configureDefaultOption — configures a default option value

### Synopsis

```
int vishnu::configureDefaultOption(const string& sessionKey, const OptionValue& optionValue);
```

### DESCRIPTION

Options in VISHNU corresponds to parameters of some VISHNU commands (e.g. the close policy for vishnu\_connect) that can be preset in the user configuration stored by the VISHNU system. This command allows an administrator to configure the default value of an option; this is the value that will be applied when the user has no configuration defined for that option using the vishnu\_configure\_option command.

### ARGUMENTS

*sessionKey* Input argument. The sessionKey is the encrypted identifier of the session generated by VISHNU.

*optionValue* Input argument. The optionValue is an object which encapsulates the option information.

### EXCEPTIONS

The following exceptions may be thrown:

"Vishnu not available (Service bus failure)" [1]

"Vishnu not available (Database error)" [2]

"Vishnu not available (Database connection)" [3]

"Vishnu not available (System)" [4]

"Internal Error: Undefined exception" [9]

"The user is not an administrator" [25]

"The session key is unrecognized" [28]

"The sessionKey is expired. The session is closed." [29]

"The name of the user option is unknown" [41]

"The value of the timeout is incorrect" [43]

"The value of the transfer command is incorrect" [44]



## Chapter 10

# UMS Python API Reference

### 10.1 VISHNU.addUser

VISHNU.addUser — adds a new VISHNU user

#### Synopsis

```
ret=VISHNU.addUser(string sessionKey, User newUser);
```

#### DESCRIPTION

This command allows an admin to add a new user in VISHNU. Several user information are necessary such as: lastname, firstname and email address. The admin also gives a VISHNU privilege to the new user and a new userId and password are sent to the user by email.

#### ARGUMENTS

*sessionKey* Input argument. The sessionKey is the encrypted identifier of the session generated by VISHNU.

*newUser* Input/Output argument. Object containing the new user information.

#### RETURNED OBJECTS

*errorCode (integer)* Output parameter. Contains 0 on success and the error code on failure.

#### EXCEPTIONS

The following exceptions may be thrown:

**SystemException("Vishnu not available (Service bus failure)" [1])**

**SystemException("Vishnu not available (Database error)" [2])**

**SystemException("Vishnu not available (Database connection)" [3])**

**SystemException("Vishnu not available (System)" [4])**  
**SystemException("Internal Error: Undefined exception" [9])**  
**UMSVishnuException("The userId already exists in the database" [22])**  
**UMSVishnuException("The user is locked" [23])**  
**UMSVishnuException("The user is not an administrator" [25])**  
**UMSVishnuException("The mail adress is invalid" [27])**  
**UMSVishnuException("The session key is unrecognized" [28])**  
**UMSVishnuException("The sessionKey is expired. The session is closed." [29])**  
**UMSVishnuException("The machine is locked" [34])**

## 10.2 VISHNU.updateUser

VISHNU.updateUser — updates the user information except the userId and the password

### Synopsis

**ret=VISHNU.updateUser**(string sessionKey, User user);

### DESCRIPTION

This command allows an admin to update a VISHNU user information or to lock a user. When a user is locked, she/he can not uses VISHNU. However, it is not possible to change the privilege of another admin.

### ARGUMENTS

*sessionKey* Input argument. The sessionKey is the encrypted identifier of the session generated by VISHNU.

*user* Input argument. Object containing user information.

### RETURNED OBJECTS

*errorCode (integer)* Output parameter. Contains 0 on success and the error code on failure.

### EXCEPTIONS

The following exceptions may be thrown:

**SystemException("Vishnu not available (Service bus failure)" [1])**  
**SystemException("Vishnu not available (Database error)" [2])**  
**SystemException("Vishnu not available (Database connection)" [3])**  
**SystemException("Vishnu not available (System)" [4])**

**SystemException("Internal Error: Undefined exception" [9])**  
**UMSVishnuException("The userId is unknown" [21])**  
**UMSVishnuException("The user is locked" [23])**  
**UMSVishnuException("Trying to lock a user account that is already locked" [24])**  
**UMSVishnuException("The user is not an administrator" [25])**  
**UMSVishnuException("The mail adress is invalid" [27])**  
**UMSVishnuException("The session key is unrecognized" [28])**  
**UMSVishnuException("The sessionKey is expired. The session is closed." [29])**

### 10.3 VISHNU.deleteUser

VISHNU.deleteUser — removes a user from VISHNU

#### Synopsis

**ret=VISHNU.deleteUser**(string sessionKey, string userId);

#### DESCRIPTION

This command allows an admin to delete a user from VISHNU. When a user is deleted from VISHNU all of her/his information are deleted from VISHNU. However, it is not possible to delete the VISHNU root user.

#### ARGUMENTS

**sessionKey** Input argument. The sessionKey is the encrypted identifier of the session generated by VISHNU.

**userId** Input argument. UserId represents the VISHNU user identifier of the user who will be deleted from VISHNU.

#### RETURNED OBJECTS

**errorCode (integer)** Output parameter. Contains 0 on success and the error code on failure.

#### EXCEPTIONS

The following exceptions may be thrown:

**SystemException("Vishnu not available (Service bus failure)" [1])**  
**SystemException("Vishnu not available (Database error)" [2])**  
**SystemException("Vishnu not available (Database connection)" [3])**  
**SystemException("Vishnu not available (System)" [4])**  
**SystemException("Internal Error: Undefined exception" [9])**

**UMSVishnuException("The userId is unknown" [21])**  
**UMSVishnuException("The user is locked" [23])**  
**UMSVishnuException("The user is not an administrator" [25])**  
**UMSVishnuException("The session key is unrecognized" [28])**  
**UMSVishnuException("The sessionKey is expired. The session is closed." [29])**

## 10.4 VISHNU.resetPassword

VISHNU.resetPassword — resets the password of a user

### Synopsis

**ret, tmpPassword=VISHNU.resetPassword(string sessionKey, string userId);**

### DESCRIPTION

This command allows an admin to reset the password of the user. The password generated is temporary and must be changed for using VISHNU.

### ARGUMENTS

**sessionKey** Input argument. The sessionKey is the encrypted identifier of the session generated by VISHNU.

**userId** Input argument. UserId represents the VISHNU user identifier of the user whose password will be reset.

**tmpPassword** Output argument. The temporary password generated by VISHNU.

### RETURNED OBJECTS

**errorCode (integer)** Output parameter. Contains 0 on success and the error code on failure.

**tmpPassword(string)** The temporary password generated by VISHNU

### EXCEPTIONS

The following exceptions may be thrown:

**SystemException("Vishnu not available (Service bus failure)" [1])**  
**SystemException("Vishnu not available (Database error)" [2])**  
**SystemException("Vishnu not available (Database connection)" [3])**  
**SystemException("Vishnu not available (System)" [4])**  
**SystemException("Internal Error: Undefined exception" [9])**  
**UMSVishnuException("The userId is unknown" [21])**  
**UMSVishnuException("The user is locked" [23])**  
**UMSVishnuException("The user is not an administrator" [25])**  
**UMSVishnuException("The session key is unrecognized" [28])**  
**UMSVishnuException("The sessionKey is expired. The session is closed." [29])**

## 10.5 VISHNU.saveConfiguration

VISHNU.saveConfiguration — saves the configuration of VISHNU

### Synopsis

**ret**=VISHNU.saveConfiguration(string sessionKey, Configuration configuration);

### DESCRIPTION

This commands allows an admin to save the VISHNU configuration. This configuration contains the list of users, the lists of machines and the list of local user configurations. It is saved on a xml format on a file registered on the directory \$HOME/.vishnu/configuration

### ARGUMENTS

**sessionKey** Input argument. The sessionKey is the encrypted identifier of the session generated by VISHNU.

**configuration** Output argument. The configuration is an object which encapsulates the configuration description.

### RETURNED OBJECTS

**errorCode (integer)** Output parameter. Contains 0 on success and the error code on failure.

### EXCEPTIONS

The following exceptions may be thrown:

SystemException("Vishnu not available (Service bus failure)" [1])

SystemException("Vishnu not available (Database error)" [2])

SystemException("Vishnu not available (Database connection)" [3])

SystemException("Vishnu not available (System)" [4])

SystemException("Internal Error: Undefined exception" [9])

UMSVishnuException("The user is not an administrator" [25])

UMSVishnuException("A problem occurs during the configuration saving " [39])

## 10.6 VISHNU.restoreConfiguration

VISHNU.restoreConfiguration — restores the configuration of VISHNU

### Synopsis

**ret**=VISHNU.restoreConfiguration(string sessionKey, string filePath);

## DESCRIPTION

This function must be used carefully as it replaces all the content of the VISHNU central database with the information stored in the provided file. This file contains the list of users, the lists of machines and the list of local user configurations. It can be created using the `vishnu_save_configuration` command. The "root" VISHNU user is the only user authorized to call this function, and this action must be done without any other user connected to VISHNU. After restoring, the vishnu database is re-initialized.

## ARGUMENTS

***sessionKey*** Input argument. The sessionKey is the encrypted identifier of the session generated by VISHNU.

***filePath*** Input argument. The filePath is the path of the file used to restore VISHNU configuration.

## RETURNED OBJECTS

***errorCode (integer)*** Output parameter. Contains 0 on success and the error code on failure.

## EXCEPTIONS

The following exceptions may be thrown:

**SystemException("Vishnu not available (Service bus failure)" [1])**

**SystemException("Vishnu not available (Database error)" [2])**

**SystemException("Vishnu not available (Database connection)" [3])**

**SystemException("Vishnu not available (System)" [4])**

**SystemException("Internal Error: Undefined exception" [9])**

**UMSVishnuException("The user is not an administrator" [25])**

**UMSVishnuException("A problem occurs during the configuration restoring" [40])**

## 10.7 VISHNU.addMachine

VISHNU.addMachine — adds a new machine in VISHNU

### Synopsis

```
ret=VISHNU.addMachine(string sessionKey, Machine newMachine);
```

## DESCRIPTION

This command allows an admin to add a new machine in VISHNU. Several machine information are mandatory such as: name, site, language and the public ssh key of the VISHNU system account on the machine. This public key will be provided automatically to all new VISHNU users who will have to add it to the authorized keys of their own account on the machine.

## ARGUMENTS

***sessionKey*** Input argument. The sessionKey is the encrypted identifier of the session generated by VISHNU.

***newMachine*** Input/Output argument. Is an object which encapsulates the information of the machine which will be added in VISHNU except the machine id which will be created automatically by VISHNU.

## RETURNED OBJECTS

***errorCode (integer)*** Output parameter. Contains 0 on success and the error code on failure.

## EXCEPTIONS

The following exceptions may be thrown:

**SystemException("Vishnu not available (Service bus failure)" [1])**

**SystemException("Vishnu not available (Database error)" [2])**

**SystemException("Vishnu not available (Database connection)" [3])**

**SystemException("Vishnu not available (System)" [4])**

**SystemException("Internal Error: Undefined exception" [9])**

**UMSVishnuException("The user is not an administrator" [25])**

**UMSVishnuException("The session key is unrecognized" [28])**

**UMSVishnuException("The sessionKey is expired. The session is closed." [29])**

**UMSVishnuException("The machineId already exists in the database" [33])**

**UMSVishnuException("The closure policy is unknown" [42])**

## 10.8 VISHNU.updateMachine

VISHNU.updateMachine — updates machine description

### Synopsis

```
ret=VISHNU.updateMachine(string sessionKey, Machine machine);
```

### DESCRIPTION

This command allows an admin to update a VISHNU machine or to locked it. A machine locked is not usable.

## ARGUMENTS

***sessionKey*** Input argument. The sessionKey is the encrypted identifier of the session generated by VISHNU.

***machine*** Input argument. Existing machine information.

---

## RETURNED OBJECTS

*errorCode* (*integer*) Output parameter. Contains 0 on success and the error code on failure.

## EXCEPTIONS

The following exceptions may be thrown:

**SystemException("Vishnu not available (Service bus failure)" [1])**

**SystemException("Vishnu not available (Database error)" [2])**

**SystemException("Vishnu not available (Database connection)" [3])**

**SystemException("Vishnu not available (System)" [4])**

**SystemException("Internal Error: Undefined exception" [9])**

**UMSVishnuException("The user is not an administrator" [25])**

**UMSVishnuException("The session key is unrecognized" [28])**

**UMSVishnuException("The sessionKey is expired. The session is closed." [29])**

**UMSVishnuException("The machine id is unknown" [32])**

**UMSVishnuException("The closure policy is unknown" [42])**

## 10.9 VISHNU.deleteMachine

VISHNU.deleteMachine — removes a machine from VISHNU

### Synopsis

```
ret=VISHNU.deleteMachine(string sessionKey, string machineId);
```

### DESCRIPTION

This command allows an admin to delete a machine from VISHNU. When a machine is deleted all of its information are deleted from VISHNU.

### ARGUMENTS

*sessionKey* Input argument. The sessionKey is the encrypted identifier of the session generated by VISHNU.

*machineId* Input argument. MachineId represents the identifier of the machine.

## RETURNED OBJECTS

*errorCode* (*integer*) Output parameter. Contains 0 on success and the error code on failure.

---



## EXCEPTIONS

The following exceptions may be thrown:

**SystemException("Vishnu not available (Service bus failure)" [1])**

**SystemException("Vishnu not available (Database error)" [2])**

**SystemException("Vishnu not available (Database connection)" [3])**

**SystemException("Vishnu not available (System)" [4])**

**SystemException("Internal Error: Undefined exception" [9])**

**UMSVishnuException("The user is not an administrator" [25])**

**UMSVishnuException("The session key is unrecognized" [28])**

**UMSVishnuException("The sessionKey is expired. The session is closed." [29])**

**UMSVishnuException("The machine id is unknown" [32])**

## 10.10 VISHNU.listUsers

VISHNU.listUsers — lists VISHNU users

### Synopsis

```
ret=VISHNU.listUsers(string sessionKey, ListUsers listuser, string userIdOption = string());
```

### DESCRIPTION

This command allows an admin to display all users information except the passwords.

### ARGUMENTS

**sessionKey** Input argument. The sessionKey is the identifier of the session generated by VISHNU.

**listuser** Output argument. Listuser is the list of users .

**userIdOption** Input argument. Allows an admin to get information about a specific user identified by his/her userId.

### RETURNED OBJECTS

**errorCode (integer)** Output parameter. Contains 0 on success and the error code on failure.

## EXCEPTIONS

The following exceptions may be thrown:

**SystemException("Vishnu not available (Service bus failure)" [1])**

**SystemException("Vishnu not available (Database error)" [2])**

**SystemException("Vishnu not available (Database connection)" [3])**

**SystemException("Vishnu not available (System)" [4])**

**SystemException("Internal Error: Undefined exception" [9])**

**UMSVishnuException("The userId is unknown" [21])**

**UMSVishnuException("The user is not an administrator" [25])**

**UMSVishnuException("The session key is unrecognized" [28])**

**UMSVishnuException("The sessionKey is expired. The session is closed." [29])**

### 10.11 VISHNU.configureDefaultOption

VISHNU.configureDefaultOption — configures a default option value

#### Synopsis

```
ret=VISHNU.configureDefaultOption(string sessionKey, OptionValue optionValue);
```

#### DESCRIPTION

Options in VISHNU corresponds to parameters of some VISHNU commands (e.g. the close policy for vishnu\_connect) that can be preset in the user configuration stored by the VISHNU system. This command allows an administrator to configure the default value of an option; this is the value that will be applied when the user has no configuration defined for that option using the vishnu\_configure\_option command.

#### ARGUMENTS

**sessionKey** Input argument. The sessionKey is the encrypted identifier of the session generated by VISHNU.

**optionValue** Input argument. The optionValue is an object which encapsulates the option information.

#### RETURNED OBJECTS

**errorCode (integer)** Output parameter. Contains 0 on success and the error code on failure.

---

## EXCEPTIONS

The following exceptions may be thrown:

**SystemException("Vishnu not available (Service bus failure)" [1])**

**SystemException("Vishnu not available (Database error)" [2])**

**SystemException("Vishnu not available (Database connection)" [3])**

**SystemException("Vishnu not available (System)" [4])**

**SystemException("Internal Error: Undefined exception" [9])**

**UMSVishnuException("The user is not an administrator" [25])**

**UMSVishnuException("The session key is unrecognized" [28])**

**UMSVishnuException("The sessionKey is expired. The session is closed." [29])**

**UMSVishnuException("The name of the user option is unknown" [41])**

**UMSVishnuException("The value of the timeout is incorrect" [43])**

**UMSVishnuException("The value of the transfer command is incorrect" [44])**

## Chapter 11

# IMS Command reference

### 11.1 vishnu\_get\_processes

`vishnu_get_processes` — displays the list of the VISHNU processes running on machines

#### Synopsis

```
vishnu_get_processes [-h] [-p machineId]
```

#### DESCRIPTION

This command with restricted access is used to get the list of VISHNU server processes that are running on the infrastructure or on a single machine. The results contain both the VISHNU identifier of the process and the DIET underlying middleware identifier.

#### OPTIONS

- `-h help` help about the command.
- `-p machineId` The id of the machine.

#### ENVIRONMENT

**VISHNU\_CONFIG\_FILE** Contains the path to the local configuration file for VISHNU.

#### DIAGNOSTICS

The following diagnostics may be issued on stderr and the command will return the code provided within brackets:

- "If a parameter is invalid" [10]
- "There is no open session in this terminal" [13]
- "Missing parameters" [14]
- "Vishnu initialization failes" [15]
- "Undefined error" [16]

## EXAMPLE

To get the list of the vishnu processes that are running and monitored on machine\_1:

```
vishnu_get_processes -p machine_1
```

## 11.2 vishnu\_set\_system\_info

`vishnu_set_system_info` — updates the system information of a machine

### Synopsis

```
vishnu_set_system_info [-h] [-m memory] [-d diskSpace] machineId
```

### DESCRIPTION

This command with restricted access is used to set system information on a machine in the VISHNU database. The machine must first be registered using the UMS "addMachine" call. Using the machine identifier, information such as the total memory and available diskspace on the machine can be added.

### OPTIONS

**-h** *help* help about the command.

**-m** *memory* Amount of RAM memory available on the machine (in Bytes).

**-d** *diskSpace* Amount of disk space available on the machine (in Bytes).

### ENVIRONMENT

**VISHNU\_CONFIG\_FILE** Contains the path to the local configuration file for VISHNU.

### DIAGNOSTICS

The following diagnostics may be issued on stderr and the command will return the code provided within brackets:

"The database generated an error" [2]

"If a parameter is invalid" [10]

"There is no open session in this terminal" [13]

"Missing parameters" [14]

"Vishnu initialization failes" [15]

"Undefined error" [16]

## EXAMPLE

To set the diskspace size to 300 on machine\_1:

```
vishnu_set_system_info -d 300 machine_1
```

## 11.3 vishnu\_set\_system\_threshold

`vishnu_set_system_threshold` — sets a threshold on a machine of a system

### Synopsis

`vishnu_set_system_threshold [-h] value machineId type handler`

### DESCRIPTION

This function allows an administrator to set a threshold. Each time an IMS server records the state of a machine, it checks the values defined, if a threshold is reached (over a use of the cpu or under the free memory or disk space available), the administrator responsible for the threshold will receive an email. These thresholds will help the administrator to be aware of critical situations on a machine. Warning, an email is sent for each time the threshold is reached, if a value swings around a threshold, the administrator may receive lots of emails depending on the update frequency.

### OPTIONS

`-h help` help about the command.

### ENVIRONMENT

**VISHNU\_CONFIG\_FILE** Contains the path to the local configuration file for VISHNU.

### DIAGNOSTICS

The following diagnostics may be issued on stderr and the command will return the code provided within brackets:

"The database generated an error" [2]

"If a parameter is invalid" [10]

"There is no open session in this terminal" [13]

"Missing parameters" [14]

"Vishnu initialization failed" [15]

"Undefined error" [16]

### EXAMPLE

To set a threshold of type use of the CPU (value=1) of value 99% on machine\_1 and handled by the user admin\_1:

```
vishnu_set_system_threshold 99 machine_1 1 admin_1
```

## 11.4 vishnu\_get\_system\_threshold

`vishnu_get_system_threshold` — gets a system threshold on a machine

## Synopsis

```
vishnu_get_system_threshold [-h] [-m machineId] [-t metricType]
```

## DESCRIPTION

This function allows an administrator to get the thresholds that may be defined on a machine. This may be used to check the parameters defined to monitor the machine. Each time a threshold is reached, a mail is sent. So checking the values of the threshold is important for the administrator to make sure they will not get tons of emails.

## OPTIONS

**-h *help*** help about the command.

**-m *machineId*** The id of the machine where the metric is defined.

**-t *metricType*** The type of the metric. The value must be an integer. Predefined values are: 0 (UNDEFINED), 1 (CPUUSE), 2 (FREEDISKSPACE), 3 (FREEMEMORY).

## ENVIRONMENT

**VISHNU\_CONFIG\_FILE** Contains the path to the local configuration file for VISHNU.

## DIAGNOSTICS

The following diagnostics may be issued on stderr and the command will return the code provided within brackets:

"The database generated an error" [2]

"If a parameter is invalid" [10]

"There is no open session in this terminal" [13]

"Missing parameters" [14]

"Vishnu initialization failes" [15]

"Undefined error" [16]

## EXAMPLE

To get all the thresholds:

```
vishnu_get_system_threshold
```

## 11.5 vishnu\_define\_user\_identifier

**vishnu\_define\_user\_identifier** — defines the shape of the identifiers automatically generated for the users

## Synopsis

```
vishnu_define_user_identifier [-h] format
```

## DESCRIPTION

This function allows an administrator to define the format of the identifier that will be automatically generated for the users. Once the format is defined, each time a user is added, the format will be used to define its identifier. The format can contain various variables, a variable is preceded by the '\$' symbol. Moreover, the counter variable '\$CPT' MUST be present in the format, otherwise it will be rejected. The available variables are: \$CPT: a counter \$NAME: The name of the user \$UNAME: The name of the user \$DAY: The day the user is added \$MONTH: The month the user is added \$YEAR: The year the user is added \$SITE: The site the user is \$TYPE: The 'U' symb to remind it is a user id

## OPTIONS

**-h** *help* help about the command.

## ENVIRONMENT

**VISHNU\_CONFIG\_FILE** Contains the path to the local configuration file for VISHNU.

## DIAGNOSTICS

The following diagnostics may be issued on stderr and the command will return the code provided within brackets:

"The database generated an error" [2]

"If a parameter is invalid" [10]

"There is no open session in this terminal" [13]

"Missing parameters" [14]

"Vishnu initialization failes" [15]

"Undefined error" [16]

## EXAMPLE

To define the format to user\_\$CPT:

```
vishnu_define_user_identifier user_\$CPT
```

## 11.6 vishnu\_define\_machine\_identifier

**vishnu\_define\_machine\_identifier** — defines the shape of the identifiers automatically generated for the machines

### Synopsis

```
vishnu_define_machine_identifier [-h] format
```



## DESCRIPTION

This function allows an administrator to define the format of the identifier that will be automatically generated for the machines. Once the format is defined, each time a machine is added, the format will be used to define its identifier. The format can contain various variables, a variable is preceded by the '\$' symbol. Moreover, the counter variable '\$CPT' MUST be present in the format, otherwise it will be rejected. The available variables are: \$CPT: a counter \$MANAME: The hostname of the machine \$NAME: The hostname of the machine \$DAY: The day the machine is added \$MONTH: The month the machine is added \$YEAR: The year the machine is added \$SITE: The site the machine is \$TYPE: The 'M' symb to remind it is a machine id

## OPTIONS

**-h help** help about the command.

## ENVIRONMENT

**VISHNU\_CONFIG\_FILE** Contains the path to the local configuration file for VISHNU.

## DIAGNOSTICS

The following diagnostics may be issued on stderr and the command will return the code provided within brackets:

"The database generated an error" [2]

"If a parameter is invalid" [10]

"There is no open session in this terminal" [13]

"Missing parameters" [14]

"Vishnu initialization failes" [15]

"Undefined error" [16]

## EXAMPLE

To define the format to machine\_\$CPT:

```
vishnu_define_machine_identifier machine\_CPT
```

## 11.7 vishnu\_define\_job\_identifier

**vishnu\_define\_job\_identifier** — defines the shape of the identifiers automatically generated for the jobs

### Synopsis

```
vishnu_define_job_identifier [-h] format
```

## DESCRIPTION

This function allows an administrator to define the format of the identifier that will be automatically generated for the jobs submitted through TMS. Once the format is defined, each time a job is submitted, the format will be used to define its identifier. The format can contain various variables, a variable is preceded by the '\$' symbol. Moreover, the counter variable '\$CPT' MUST be present in the format, otherwise it will be rejected. The available variables are: \$CPT: a counter \$DAY: The day the job is submitted \$MONTH: The month the job is submitted \$YEAR: The year the job is submitted \$TYPE: The 'J' symb to remind it is a job id

## OPTIONS

**-h** *help* help about the command.

## ENVIRONMENT

**VISHNU\_CONFIG\_FILE** Contains the path to the local configuration file for VISHNU.

## DIAGNOSTICS

The following diagnostics may be issued on stderr and the command will return the code provided within brackets:

"The database generated an error" [2]

"If a parameter is invalid" [10]

"There is no open session in this terminal" [13]

"Missing parameters" [14]

"Vishnu initialization failes" [15]

"Undefined error" [16]

## EXAMPLE

To define the format to job\_\$CPT:

```
vishnu_define_job_identifier job\_CPT
```

## 11.8 vishnu\_define\_transfer\_identifier

**vishnu\_define\_transfer\_identifier** — defines the shape of the identifiers automatically generated for the file transfers

### Synopsis

```
vishnu_define_transfer_identifier [-h] format
```

## DESCRIPTION

This function allows an administrator to define the format of the identifier that will be automatically generated for the file transfers. Once the format is defined, each time a file transfer is done, the format will be used to define its identifier. The format can contain various variables, a variable is preceded by the '\$' symbol. Moreover, the counter variable '\$CPT' MUST be present in the format, otherwise it will be rejected. The available variables are: \$CPT: a counter \$DAY: The day the file transfer is done \$MONTH: The month the file transfer is done \$YEAR: The year the file transfer is done \$TYPE: The 'F' symb to remind it is a file transfer id

## OPTIONS

**-h help** help about the command.

## ENVIRONMENT

**VISHNU\_CONFIG\_FILE** Contains the path to the local configuration file for VISHNU.

## DIAGNOSTICS

The following diagnostics may be issued on stderr and the command will return the code provided within brackets:

"The database generated an error" [2]

"If a parameter is invalid" [10]

"There is no open session in this terminal" [13]

"Missing parameters" [14]

"Vishnu initialization failes" [15]

"Undefined error" [16]

## EXAMPLE

To define the format to transfer\_\$CPT:

```
vishnu_define_transfer_identifier transfer\_SCPT
```

## 11.9 vishnu\_load\_shed

**vishnu\_load\_shed** — sheds load on a machine

### Synopsis

```
vishnu_load_shed [-h] machineId loadShedType
```

## DESCRIPTION

This function allows an administrator to shed load on a machine. Two modes are available: SOFT mode will cancel all the submitted jobs and file transfers for all VISHNU users (Note that jobs and file transfers not initiated through VISHNU will not be impacted). HARD mode will additionally stop all the VISHNU processes on the machine. If a user without administrator rights uses this function, all the user's jobs and file transfers will be cancelled on the machine. In the HARD mode, the stopped processes will not be automatically restarted. Type values: HARD = 1 SOFT = 2

## OPTIONS

**-h** *help* help about the command.

## ENVIRONMENT

**VISHNU\_CONFIG\_FILE** Contains the path to the local configuration file for VISHNU.

## DIAGNOSTICS

The following diagnostics may be issued on stderr and the command will return the code provided within brackets:

"The database generated an error" [2]

"If a parameter is invalid" [10]

"There is no open session in this terminal" [13]

"Missing parameters" [14]

"Vishnu initialization failes" [15]

"Undefined error" [16]

"If a component is unavailable" [301]

## EXAMPLE

To make a hard load shedding on machine\_1:

```
vishnu_load_shed machine_1 1
```

## 11.10 vishnu\_set\_update\_frequency

**vishnu\_set\_update\_frequency** — sets the update frequency of the IMS tables

### Synopsis

```
vishnu_set_update_frequency [-h] freq
```

## DESCRIPTION

This function allows an admin to set the update frequency. This frequency corresponds to how often the state of the machines is automatically polled by the IMS server. The value must be in seconds.

## OPTIONS

**-h** *help* help about the command.

---

## ENVIRONMENT

**VISHNU\_CONFIG\_FILE** Contains the path to the local configuration file for VISHNU.

## DIAGNOSTICS

The following diagnostics may be issued on stderr and the command will return the code provided within brackets:

"The database generated an error" [2]

"If a parameter is invalid" [10]

"There is no open session in this terminal" [13]

"Missing parameters" [14]

"Vishnu initialization failes" [15]

"Undefined error" [16]

## EXAMPLE

To set the frequency to 100:

```
vishnu_set_update_frequency 100
```

### 11.11 vishnu\_stop

**vishnu\_stop** — To stop (and do not try to relaunch) a SeD

## Synopsis

```
vishnu_stop [-h] processName machineId
```

## DESCRIPTION

This function allows an admin to stop a VISHNU server on a machine. The stopped process will not be restarted automatically. The important parameters in the process are the names and the machine. The *processName* must be UMS, TMS, IMS or FMS , in upper case.

## OPTIONS

**-h** *help* help about the command.

## ENVIRONMENT

**VISHNU\_CONFIG\_FILE** Contains the path to the local configuration file for VISHNU.

---

## DIAGNOSTICS

The following diagnostics may be issued on stderr and the command will return the code provided within brackets:

"The database generated an error" [2]

"If a parameter is invalid" [10]

"There is no open session in this terminal" [13]

"Missing parameters" [14]

"Vishnu initialization failes" [15]

"Undefined error" [16]

## EXAMPLE

To stop the UMS process on machine\_1:

```
vishnu_stop UMS machine_1
```

### 11.12 vishnu\_restart

`vishnu_restart` — To restart a SeD or a MA

#### Synopsis

```
vishnu_restart [-h] [-v vishnuConf] [-t sedType] machineId
```

#### DESCRIPTION

This function allows an admin to restart a VISHNU server on a machine. Warning when restarting a server, first it is tried to stop it, so if one is running it is stopped and then another is restarted.

#### OPTIONS

**-h** *help* help about the command.

**-v** *vishnuConf* The path to the vishnu configuration file.

**-t** *sedType* The type of the vishnu sed. The value must be an integer. Predefined values are: 0 (UNDEFINED), 1 (UMS), 2 (TMS), 3 (FMS), 4 (IMS).

#### ENVIRONMENT

**VISHNU\_CONFIG\_FILE** Contains the path to the local configuration file for VISHNU.

## DIAGNOSTICS

The following diagnostics may be issued on stderr and the command will return the code provided within brackets:

**"The database generated an error" [2]**

**"If a parameter is invalid" [10]**

**"There is no open session in this terminal" [13]**

**"Missing parameters" [14]**

**"Vishnu initialization failes" [15]**

**"Undefined error" [16]**

## EXAMPLE

To restart using the configuration file ums.cfg an UMS sed on machine\_1:

```
vishnu_restart -v /tmp/ums.cfg -t 1 machine_1
```

## Chapter 12

# IMS C++ API Reference

### 12.1 getProcesses

getProcesses — displays the list of the VISHNU processes running on machines

#### Synopsis

```
int vishnu::getProcesses(const string& sessionKey, ListProcesses& process, const ProcessOp& options = ProcessOp());
```

#### DESCRIPTION

This command with restricted access is used to get the list of VISHNU server processes that are running on the infrastructure or on a single machine. The results contain both the VISHNU identifier of the process and the DIET underlying middleware identifier.

#### ARGUMENTS

*sessionKey* Input argument. The session key.

*process* Output argument. The list of the Vishnu processes on the machine.

*options* Input argument. The options to search for the processes.

#### EXCEPTIONS

The following exceptions may be thrown:

"If a parameter is invalid" [10]

### 12.2 setSystemInfo

setSystemInfo — updates the system information of a machine

#### Synopsis

```
int vishnu::setSystemInfo(const string& sessionKey, const SystemInfo& systemInfo);
```



## DESCRIPTION

This command with restricted access is used to set system information on a machine in the VISHNU database. The machine must first be registered using the UMS "addMachine" call. Using the machine identifier, information such as the total memory and available disk space on the machine can be added.

## ARGUMENTS

*sessionKey* Input argument. The session key.

*systemInfo* Input argument. Contains system information to store in Vishnu database.

## EXCEPTIONS

The following exceptions may be thrown:

"The database generated an error" [2]

"If a parameter is invalid" [10]

## 12.3 setSystemThreshold

setSystemThreshold — sets a threshold on a machine of a system

### Synopsis

```
int vishnu::setSystemThreshold(const string& sessionKey, const Threshold& threshold);
```

## DESCRIPTION

This function allows an administrator to set a threshold. Each time an IMS server records the state of a machine, it checks the values defined, if a threshold is reached (over a use of the cpu or under the free memory or disk space available), the administrator responsible for the threshold will receive an email. These thresholds will help the administrator to be aware of critical situations on a machine. Warning, an email is sent for each time the threshold is reached, if a value swings around a threshold, the administrator may receive lots of emails depending on the update frequency.

## ARGUMENTS

*sessionKey* Input argument. The session key.

*threshold* Input argument. The threshold to set.

## EXCEPTIONS

The following exceptions may be thrown:

"The database generated an error" [2]

"If a parameter is invalid" [10]

## 12.4 getSystemThreshold

getSystemThreshold — gets a system threshold on a machine

### Synopsis

```
int vishnu::getSystemThreshold(const string& sessionKey, ListThreshold& value, const ThresholdOp& options);
```

### DESCRIPTION

This function allows an administrator to get the thresholds that may be defined on a machine. This may be used to check the parameters defined to monitor the machine. Each time a threshold is reached, a mail is sent. So checking the values of the threshold is important for the administrator to make sure they will not get tons of emails.

### ARGUMENTS

*sessionKey* Input argument. The session key.

*value* Output argument. The thresholds value.

*options* Input argument. The options for the threshold.

### EXCEPTIONS

The following exceptions may be thrown:

"The database generated an error" [2]

"If a parameter is invalid" [10]

## 12.5 defineUserIdentifier

defineUserIdentifier — defines the shape of the identifiers automatically generated for the users

### Synopsis

```
int vishnu::defineUserIdentifier(const string& sessionKey, const string& format);
```

### DESCRIPTION

This function allows an administrator to define the format of the identifier that will be automatically generated for the users. Once the format is defined, each time a user is added, the format will be used to define its identifier. The format can contain various variables, a variable is preceded by the '\$' symbol. Moreover, the counter variable '\$CPT' MUST be present in the format, otherwise it will be rejected. The available variables are: \$CPT: a counter \$NAME: The name of the user \$UNAME: The name of the user \$DAY: The day the user is added \$MONTH: The month the user is added \$YEAR: The year the user is added \$SITE: The site the user is \$TYPE: The 'U' symb to remind it is a user id

## ARGUMENTS

*sessionKey* Input argument. The session key.

*format* Input argument. The new format to use.

## EXCEPTIONS

The following exceptions may be thrown:

**"The database generated an error"** [2]

**"If a parameter is invalid"** [10]

## 12.6 defineMachineIdentifier

defineMachineIdentifier — defines the shape of the identifiers automatically generated for the machines

### Synopsis

```
int vishnu::defineMachineIdentifier(const string& sessionKey, const string& format);
```

## DESCRIPTION

This function allows an administrator to define the format of the identifier that will be automatically generated for the machines. Once the format is defined, each time a machine is added, the format will be used to define its identifier. The format can contain various variables, a variable is preceded by the '\$' symbol. Moreover, the counter variable '\$CPT' MUST be present in the format, otherwise it will be rejected. The available variables are: \$CPT: a counter \$MANAME: The hostname of the machine \$NAME: The hostname of the machine \$DAY: The day the machine is added \$MONTH: The month the machine is added \$YEAR: The year the machine is added \$SITE: The site the machine is \$TYPE: The 'M' symb to remind it is a machine id

## ARGUMENTS

*sessionKey* Input argument. The session key.

*format* Input argument. The new format to use.

## EXCEPTIONS

The following exceptions may be thrown:

**"The database generated an error"** [2]

**"If a parameter is invalid"** [10]

## 12.7 defineJobIdentifier

defineJobIdentifier — defines the shape of the identifiers automatically generated for the jobs

---

## Synopsis

```
int vishnu::defineJobIdentifier(const string& sessionKey, const string& format);
```

## DESCRIPTION

This function allows an administrator to define the format of the identifier that will be automatically generated for the jobs submitted through TMS. Once the format is defined, each time a job is submitted, the format will be used to define its identifier. The format can contain various variables, a variable is preceded by the '\$' symbol. Moreover, the counter variable '\$CPT' MUST be present in the format, otherwise it will be rejected. The available variables are: \$CPT: a counter \$DAY: The day the job is submitted \$MONTH: The month the job is submitted \$YEAR: The year the job is submitted \$TYPE: The 'J' symb to remind it is a job id

## ARGUMENTS

*sessionKey* Input argument. The session key.

*format* Input argument. The new format to use.

## EXCEPTIONS

The following exceptions may be thrown:

"The database generated an error" [2]

"If a parameter is invalid" [10]

## 12.8 defineTransferIdentifier

defineTransferIdentifier — defines the shape of the identifiers automatically generated for the file transfers

## Synopsis

```
int vishnu::defineTransferIdentifier(const string& sessionKey, const string& format);
```

## DESCRIPTION

This function allows an administrator to define the format of the identifier that will be automatically generated for the file transfers. Once the format is defined, each time a file transfer is done, the format will be used to define its identifier. The format can contain various variables, a variable is preceded by the '\$' symbol. Moreover, the counter variable '\$CPT' MUST be present in the format, otherwise it will be rejected. The available variables are: \$CPT: a counter \$DAY: The day the file transfer is done \$MONTH: The month the file transfer is done \$YEAR: The year the file transfer is done \$TYPE: The 'F' symb to remind it is a file transfer id

## ARGUMENTS

*sessionKey* Input argument. The session key.

*format* Input argument. The new format to use.

## EXCEPTIONS

The following exceptions may be thrown:

"The database generated an error" [2]

"If a parameter is invalid" [10]

## 12.9 loadShed

loadShed — sheds load on a machine

### Synopsis

```
int vishnu::loadShed(const string& sessionKey, const string& machineId, const LoadShedType& loadShedType);
```

### DESCRIPTION

This function allows an administrator to shed load on a machine. Two modes are available: SOFT mode will cancel all the submitted jobs and file transfers for all VISHNU users (Note that jobs and file transfers not initiated through VISHNU will not be impacted). HARD mode will additionally stop all the VISHNU processes on the machine. If a user without administrator rights uses this function, all the user's jobs and file transfers will be cancelled on the machine. In the HARD mode, the stopped processes will not be automatically restarted. Type values: HARD = 1 SOFT = 2

### ARGUMENTS

*sessionKey* Input argument. The session key.

*machineId* Input argument. The id of the machine to stop.

*loadShedType* Input argument. Selects a load shedding mode (SOFT: stops all services and they can be restarted, HARD: stops all services, they cannot be restarted).

## EXCEPTIONS

The following exceptions may be thrown:

"The database generated an error" [2]

"If a parameter is invalid" [10]

"If a component is unavailable" [301]

## 12.10 setUpdateFrequency

setUpdateFrequency — sets the update frequency of the IMS tables

### Synopsis

```
int vishnu::setUpdateFrequency(const string& sessionKey, const int& freq);
```

## DESCRIPTION

This function allows an admin to set the update frequency. This frequency corresponds to how often the state of the machines is automatically polled by the IMS server. The value must be in seconds.

## ARGUMENTS

*sessionKey* Input argument. The session key.

*freq* Input argument. Frequency the data are updated, in second.

## EXCEPTIONS

The following exceptions may be thrown:

**"The database generated an error" [2]**

**"If a parameter is invalid" [10]**

## 12.11 stop

stop — To stop (and do not try to relaunch) a SeD

### Synopsis

```
int vishnu::stop(const string& sessionKey, const Process& process);
```

## DESCRIPTION

This function allows an admin to stop a VISHNU server on a machine. The stopped process will not be restarted automatically. The important parameters in the process are the names and the machine. The processName must be UMS, TMS, IMS or FMS , in upper case.

## ARGUMENTS

*sessionKey* Input argument. The session key.

*process* Input argument. The process to stop and do not try to restart anymore.

## EXCEPTIONS

The following exceptions may be thrown:

**"The database generated an error" [2]**

**"If a parameter is invalid" [10]**

## 12.12 restart

restart — To restart a SeD or a MA

## Synopsis

```
int vishnu::restart(const string& sessionKey, const string& machineId, const RestartOp& options);
```

## DESCRIPTION

This function allows an admin to restart a VISHNU server on a machine. Warning when restarting a server, first it is tried to stop it, so if one is running it is stopped and then another is restarted.

## ARGUMENTS

*sessionKey* Input argument. The session key.

*machineId* Input argument. The id of the machine where to restart.

*options* Input argument. The option for the restart.

## EXCEPTIONS

The following exceptions may be thrown:

**"The database generated an error" [2]**

**"If a parameter is invalid" [10]**

## Chapter 13

# IMS Python API Reference

### 13.1 VISHNU.getProcesses

VISHNU.getProcesses — displays the list of the VISHNU processes running on machines

#### Synopsis

```
ret=VISHNU.getProcesses(string sessionKey, ListProcesses process, ProcessOp options = ProcessOp());
```

#### DESCRIPTION

This command with restricted access is used to get the list of VISHNU server processes that are running on the infrastructure or on a single machine. The results contain both the VISHNU identifier of the process and the DIET underlying middleware identifier.

#### ARGUMENTS

*sessionKey* Input argument. The session key.

*process* Output argument. The list of the Vishnu processes on the machine.

*options* Input argument. The options to search for the processes.

#### RETURNED OBJECTS

*errorCode (integer)* Output parameter. Contains 0 on success and the error code on failure.

#### EXCEPTIONS

The following exceptions may be thrown:

**UserException("If a parameter is invalid" [10])**



## 13.2 VISHNU.setSystemInfo

VISHNU.setSystemInfo — updates the system information of a machine

### Synopsis

```
ret=VISHNU.setSystemInfo(string sessionKey, SystemInfo systemInfo);
```

### DESCRIPTION

This command with restricted access is used to set system information on a machine in the VISHNU database. The machine must first be registered using the UMS "addMachine" call. Using the machine identifier, information such as the total memory and available disk space on the machine can be added.

### ARGUMENTS

*sessionKey* Input argument. The session key.

*systemInfo* Input argument. Contains system information to store in Vishnu database.

### RETURNED OBJECTS

*errorCode (integer)* Output parameter. Contains 0 on success and the error code on failure.

### EXCEPTIONS

The following exceptions may be thrown:

**SystemException("The database generated an error" [2])**

**UserException("If a parameter is invalid" [10])**

## 13.3 VISHNU.setSystemThreshold

VISHNU.setSystemThreshold — sets a threshold on a machine of a system

### Synopsis

```
ret=VISHNU.setSystemThreshold(string sessionKey, Threshold threshold);
```

### DESCRIPTION

This function allows an administrator to set a threshold. Each time an IMS server records the state of a machine, it checks the values defined, if a threshold is reached (over a use of the cpu or under the free memory or disk space available), the administrator responsible for the threshold will receive an email. These thresholds will help the administrator to be aware of critical situations on a machine. Warning, an email is sent for each time the threshold is reached, if a value swings around a threshold, the administrator may receive lots of emails depending on the update frequency.

## ARGUMENTS

*sessionKey* Input argument. The session key.

*threshold* Input argument. The threshold to set.

## RETURNED OBJECTS

*errorCode (integer)* Output parameter. Contains 0 on success and the error code on failure.

## EXCEPTIONS

The following exceptions may be thrown:

**SystemException("The database generated an error" [2])**

**UserException("If a parameter is invalid" [10])**

## 13.4 VISHNU.getSystemThreshold

VISHNU.getSystemThreshold — gets a system threshold on a machine

### Synopsis

```
ret=VISHNU.getSystemThreshold(string sessionKey, ListThreshold value, ThresholdOp options);
```

### DESCRIPTION

This function allows an administrator to get the thresholds that may be defined on a machine. This may be used to check the parameters defined to monitor the machine. Each time a threshold is reached, a mail is sent. So checking the values of the threshold is important for the administrator to make sure they will not get tons of emails.

## ARGUMENTS

*sessionKey* Input argument. The session key.

*value* Output argument. The thresholds value.

*options* Input argument. The options for the threshold.

## RETURNED OBJECTS

*errorCode (integer)* Output parameter. Contains 0 on success and the error code on failure.

---

## EXCEPTIONS

The following exceptions may be thrown:

**SystemException("The database generated an error" [2])**

**UserException("If a parameter is invalid" [10])**

## 13.5 VISHNU.defineUserIdentifier

VISHNU.defineUserIdentifier — defines the shape of the identifiers automatically generated for the users

### Synopsis

**ret=VISHNU.defineUserIdentifier(string sessionKey, string format);**

### DESCRIPTION

This function allows an administrator to define the format of the identifier that will be automatically generated for the users. Once the format is defined, each time a user is added, the format will be used to define its identifier. The format can contain various variables, a variable is preceded by the '\$' symbol. Moreover, the counter variable '\$CPT' MUST be present in the format, otherwise it will be rejected. The available variables are: \$CPT: a counter \$NAME: The name of the user \$UNAME: The name of the user \$DAY: The day the user is added \$MONTH: The month the user is added \$YEAR: The year the user is added \$SITE: The site the user is \$TYPE: The 'U' symb to remind it is a user id

### ARGUMENTS

*sessionKey* Input argument. The session key.

*format* Input argument. The new format to use.

### RETURNED OBJECTS

*errorCode (integer)* Output parameter. Contains 0 on success and the error code on failure.

## EXCEPTIONS

The following exceptions may be thrown:

**SystemException("The database generated an error" [2])**

**UserException("If a parameter is invalid" [10])**

## 13.6 VISHNU.defineMachineIdentifier

VISHNU.defineMachineIdentifier — defines the shape of the identifiers automatically generated for the machines

## Synopsis

```
ret=VISHNU.defineMachineIdentifier(string sessionKey, string format);
```

## DESCRIPTION

This function allows an administrator to define the format of the identifier that will be automatically generated for the machines. Once the format is defined, each time a machine is added, the format will be used to define its identifier. The format can contain various variables, a variable is preceded by the '\$' symbol. Moreover, the counter variable '\$CPT' MUST be present in the format, otherwise it will be rejected. The available variables are: \$CPT: a counter \$MANAME: The hostname of the machine \$NAME: The hostname of the machine \$DAY: The day the machine is added \$MONTH: The month the machine is added \$YEAR: The year the machine is added \$SITE: The site the machine is \$TYPE: The 'M' symb to remind it is a machine id

## ARGUMENTS

*sessionKey* Input argument. The session key.

*format* Input argument. The new format to use.

## RETURNED OBJECTS

*errorCode (integer)* Output parameter. Contains 0 on success and the error code on failure.

## EXCEPTIONS

The following exceptions may be thrown:

**SystemException("The database generated an error" [2])**

**UserException("If a parameter is invalid" [10])**

## 13.7 VISHNU.defineJobIdentifier

VISHNU.defineJobIdentifier — defines the shape of the identifiers automatically generated for the jobs

## Synopsis

```
ret=VISHNU.defineJobIdentifier(string sessionKey, string format);
```

## DESCRIPTION

This function allows an administrator to define the format of the identifier that will be automatically generated for the jobs submitted through TMS. Once the format is defined, each time a job is submitted, the format will be used to define its identifier. The format can contain various variables, a variable is preceded by the '\$' symbol. Moreover, the counter variable '\$CPT' MUST be present in the format, otherwise it will be rejected. The available variables are: \$CPT: a counter \$DAY: The day the job is submitted \$MONTH: The month the job is submitted \$YEAR: The year the job is submitted \$TYPE: The 'J' symb to remind it is a job id

## ARGUMENTS

*sessionKey* Input argument. The session key.

*format* Input argument. The new format to use.

## RETURNED OBJECTS

*errorCode (integer)* Output parameter. Contains 0 on success and the error code on failure.

## EXCEPTIONS

The following exceptions may be thrown:

**SystemException("The database generated an error" [2])**

**UserException("If a parameter is invalid" [10])**

## 13.8 VISHNU.defineTransferIdentifier

VISHNU.defineTransferIdentifier — defines the shape of the identifiers automatically generated for the file transfers

### Synopsis

```
ret=VISHNU.defineTransferIdentifier(string sessionKey, string format);
```

### DESCRIPTION

This function allows an administrator to define the format of the identifier that will be automatically generated for the file transfers. Once the format is defined, each time a file transfer is done, the format will be used to define its identifier. The format can contain various variables, a variable is preceded by the '\$' symbol. Moreover, the counter variable '\$CPT' MUST be present in the format, otherwise it will be rejected. The available variables are: \$CPT: a counter \$DAY: The day the file transfer is done \$MONTH: The month the file transfer is done \$YEAR: The year the file transfer is done \$TYPE: The 'F' symb to remind it is a file transfer id

## ARGUMENTS

*sessionKey* Input argument. The session key.

*format* Input argument. The new format to use.

## RETURNED OBJECTS

*errorCode (integer)* Output parameter. Contains 0 on success and the error code on failure.

## EXCEPTIONS

The following exceptions may be thrown:

**SystemException**("The database generated an error" [2])

**UserException**("If a parameter is invalid" [10])

## 13.9 VISHNU.loadShed

VISHNU.loadShed — sheds load on a machine

### Synopsis

**ret**=VISHNU.loadShed(string sessionKey, string machineId, LoadShedType loadShedType);

### DESCRIPTION

This function allows an administrator to shed load on a machine. Two modes are available: SOFT mode will cancel all the submitted jobs and file transfers for all VISHNU users (Note that jobs and file transfers not initiated through VISHNU will not be impacted). HARD mode will additionally stop all the VISHNU processes on the machine. If a user without administrator rights uses this function, all the user's jobs and file transfers will be cancelled on the machine. In the HARD mode, the stopped processes will not be automatically restarted. Type values: HARD = 1 SOFT = 2

### ARGUMENTS

**sessionKey** Input argument. The session key.

**machineId** Input argument. The id of the machine to stop.

**loadShedType** Input argument. Selects a load shedding mode (SOFT: stops all services and they can be restarted, HARD: stops all services, they cannot be restarted).

### RETURNED OBJECTS

**errorCode (integer)** Output parameter. Contains 0 on success and the error code on failure.

## EXCEPTIONS

The following exceptions may be thrown:

**SystemException**("The database generated an error" [2])

**UserException**("If a parameter is invalid" [10])

**IMSVishnuException**("If a component is unavailable" [301])

## 13.10 VISHNU.setUpdateFrequency

VISHNU.setUpdateFrequency — sets the update frequency of the IMS tables

## Synopsis

```
ret=VISHNU.setUpdateFrequency(string sessionKey, int freq);
```

## DESCRIPTION

This function allows an admin to set the update frequency. This frequency corresponds to how often the state of the machines is automatically polled by the IMS server. The value must be in seconds.

## ARGUMENTS

*sessionKey* Input argument. The session key.

*freq* Input argument. Frequency the data are updated, in second.

## RETURNED OBJECTS

*errorCode (integer)* Output parameter. Contains 0 on success and the error code on failure.

## EXCEPTIONS

The following exceptions may be thrown:

**SystemException("The database generated an error" [2])**

**UserException("If a parameter is invalid" [10])**

## 13.11 VISHNU.stop

VISHNU.stop — To stop (and do not try to relaunch) a SeD

## Synopsis

```
ret=VISHNU.stop(string sessionKey, Process process);
```

## DESCRIPTION

This function allows an admin to stop a VISHNU server on a machine. The stopped process will not be restarted automatically. The important parameters in the process are the names and the machine. The processName must be UMS, TMS, IMS or FMS , in upper case.

## ARGUMENTS

*sessionKey* Input argument. The session key.

*process* Input argument. The process to stop and do not try to restart anymore.

## RETURNED OBJECTS

*errorCode (integer)* Output parameter. Contains 0 on success and the error code on failure.

## EXCEPTIONS

The following exceptions may be thrown:

**SystemException("The database generated an error" [2])**

**UserException("If a parameter is invalid" [10])**

## 13.12 VISHNU.restart

VISHNU.restart — To restart a SeD or a MA

### Synopsis

**ret=VISHNU.restart**(string sessionKey, string machineId, RestartOp options);

### DESCRIPTION

This function allows an admin to restart a VISHNU server on a machine. Warning when restarting a server, first it is tried to stop it, so if one is running it is stopped and then another is restarted.

### ARGUMENTS

*sessionKey* Input argument. The session key.

*machineId* Input argument. The id of the machine where to restart.

*options* Input argument. The option for the restart.

## RETURNED OBJECTS

*errorCode (integer)* Output parameter. Contains 0 on success and the error code on failure.

## EXCEPTIONS

The following exceptions may be thrown:

**SystemException("The database generated an error" [2])**

**UserException("If a parameter is invalid" [10])**